Tópicos Avançados de Programação Funcional

DCC/FCUP — 14 de junho, 2016

- Este exame contém **cinco (5) questoes** e **seis (6) páginas**
- Pode consultar a folha de apêndice com resumos de documentação de algumas bibliotecas Haskell
- Escreva as respostas numa folha de exame separada; pode alterar a ordem das questões desde que estas estejam claramente identificadas

Questão 1 (20%)

(a) Considere a seguinte definição recursiva da função `getLine` do prelúdio-padrão de Haskell.

```
getLine :: IO [Char]
getLine = do
   x <- getChar
   if x=='\n' then
      return []
   else do
     xs <- getLine
     return (x:xs)
```

Re-escreva a definição acima traduzindo a notação-*do* para aplicações dos operadores monádicos (`>>=` ou `>>`).

(b) A função `replicateM` da biblioteca `Control.Monad` constroi uma lista executando n vezes uma ação monádica. O seu tipo mais geral é:

```
replicateM :: Monad m => Int -> m a -> m [a]
```

Por exemplo: `replicateM 5 getLine` é a ação I/O que lê 5 linhas da entrada-padrão e constroi a lista com os resultados.

Escreva uma definição recursiva de `replicateM`; pode usar notação-*do* e/ou os operadores monádicos mas não deve outras funções de ordem superior.

Questão 2 (20%)

Considere a definição de classe de tipos `Cmp` com um único método `cmp` para *comparações numa ordem total* (i.e., uma versão simplificada da classe *Ord*).

```
data Ordering = LT | EQ | GT  -- resultado duma comparação

class Cmp a where
   cmp :: a -> a -> Ordering

instance Cmp Int where
  cmp = cmpInt   -- primitiva

instance (Cmp a, Cmp b) => Cmp (a,b) where
  cmp (x,y) (x',y') = case cmp x x' of
                        EQ -> cmp y y' -- x, x' são iguais
                        r -> r         -- x, x' são diferentes
```

O resultado de `cmp x y` é `LT` quando `x` é estritamente menor do que `y`, `EQ` quando `x` e `y` são iguais e `GT` quando `x` é estritamente maior do que `y`.

O código acima define ainda instâncias para inteiros (usando uma operação primitiva `cmpInt`) e uma instância paramétrica para pares correspondente à *ordem lexicográfica* das componentes, e.g. `cmp (2,3) (1,5)` dá `GT` porque $2 > 1$.

(a) Traduza a definição da classe `Cmp` e das duas instâncias acima usando dicionários.

(b) Traduza a seguinte definição da função `max` para passar explicitamente o dicionário da classe `Cmp`.

```
max :: Cmp a => a -> a -> a
max x y = case cmp x y of
               LT -> y
               _  -> x
```

Questão 3 (20%)

Considere a definição e instância da classe `Monad` para um tipo paramétrico `Result e a` que representa computações que podem abortar com um *erro* de tipo `e` ou produzir um resultado de tipo `a`.

```
data Result e a = Error e | OK a

instance Monad (Result e) where
   return          = OK
   Error e  >>= k = Error e
   Result v >>= k = k v
```

(a) Mostre que as três leis de mónadas se verificam para as definições de `return` e `>>=` acima.

(b) Considere agora o tipo algébrico `Expr` para expressões com nomes e operações de soma e divisão inteira:

```
data Expr = Var Name
          | Const Int
          | Add Expr Expr
          | Div Expr Expr
```

Escreva uma definição dum avaliador `eval :: Env -> Expr -> Result Error Int` em que os tipos de ambientes de variáveis e erros são dados pelas definições seguintes:

```
type Name = String
type Env = [(Name, Int)]
data Error = ZeroDivision | UndefinedName
```

Questão 4 (20%)

Considere a função `group :: Eq a => [a] -> [[a]]` da bibloteca base (módulo `Data.List`) que agrupa elementos idênticos consecutivos numa lista. Alguns exemplos de uso:

```
group []            = []
group [1,2,1]       = [[1],[2],[1]]
group [0,1,1,2,1]   = [[0],[1,1],[2],[1]]
group "Mississippi" = ["M","i","ss","i","ss","i","pp","i"]
```

(a) Escreva propriedades *QuickCheck* que exprimam as seguintes condições sobre esta função:

`prop_group_len` a soma dos comprimentos de `group xs` é igual ao comprimento de `xs`

`prop_group_concat` a concatenção de `group xs` é igual a `xs` (esta condição implica a anterior)

`prop_group_equal` cada lista em `group xs` só tem elementos iguais entre si

(b) Se tentar testar as propriedades acima com o QuickCheck poderá notar que o gerador de listas *standard* em que *a probabilidade de ocorrerem elementos repetidos é baixa*, o que faz com que o comprimento dos grupos seja pequeno (1 ou 2). Prentende-se que escreva um gerador de listas especial em que o comprimento do grupos aumente à medida que o tamanho da lista aumenta.

Escreva um gerador `genGroups :: Arbitrary a => Int -> Gen [a]` cujo argumento é o tamanho da lista a gerar; eis uma estratégia possível:

0. se o tamanho é zero, então termina imediatamente com a lista vazia;
1. escolhemos o tamanho n do primeiro grupo (entre 1 e o tamanho total);
2. escolhemos um valor v arbitrário; o primeiro grupo consiste do elemento v repetido k vezes;
3. recursivamente geramos a lista de grupos com com o tamanho em falta.

Questão 5 (20%)

Considere uma DSEL para intervalos de inteiros com o seguinte tipo e operações:

```
data Interval                              -- tipo abstrato

-- construtores
empty :: Interval                          -- vazio
range :: Integer -> Integer -> Interval    -- limites inferior e superior

-- combinadores
(/\) :: Interval -> Interval -> Interval   -- interseção
(\/) :: Interval -> Interval -> Interval   -- união

-- observação
isEmpty :: Interval -> Bool
lowerBound, upperBound :: Interval -> Integer   -- não definidos para vazio
```

Alguns exemplos de uso:

```
isEmpty empty                          = True
isEmpty (range 1 5)                    = False
isEmpty (range 5 1)                    = True  -- porque 5 > 1

lowerBound (range 1 5 /\ range 3 7)    = 3
upperBound (range 1 5 /\ range 3 7)    = 5

lowerBound (range 1 5 /\ range 7 10)   -- ERRO: indefinido
isEmpty    (range 1 5 /\ range 7 10)   = True

lowerBound (range 1 5 \/ range 7 10)   = 1
upperBound (range 1 5 \/ range 7 10)   = 10
  -- NOTA: só podemos pedir limites de intervalos
  -- logo não conseguimos observar o "buraco" entre 5 e 7
```

Implemente esta DSEL como um módulo `Interval` em Haskell que exporta apenas as entidades necessárias. Sugestão: use *shallow embedding*.
