Programação Funcional
Revisão

Prof. Wladimir Araújo Tavares

IO Haskell

1. A função `putStrLn :: String -> IO()` recebe uma string e retorna uma ação de I/O. Uma ação de I/O é algo que, quando realizado, resulta em uma ação com efeito colateral (que pode ser ler ou escrever alguma coisa no dipositivo padrão de entrada) e que pode conter algum tipo de valor de retorno. Mostra uma string no terminal não precisa retorna nem um valor com significado então o valor () deve ser usado.

   A função putStrLn pode ser definido em termos de putChar usando combinadores da seguinte maneira:

```
putstrln []     = putChar '\n'
putstrln (c:cs) = putChar c >> putstrln cs
```

   Escreva uma definição de `putStrLn :: IO ()` em termos de putChar usando a notação do.
2. A função getLine :: IO String, que lê uma linha do dipositivo de entrada padrão até que um caractere '
   n' seja encontrado. A função getLine pode ser definido em termos de getChar usando combinadores da seguinte maneira:

```
getline = getChar >>=
        (\c -> if c == '\n' then return ""
               else getline >>= \l -> return (c:l) )
```

   Escreva uma definição de getLine :: IO String em termos de getChar usando a notação do.
3. Defina uma função `repeat:: (a -> IO Bool) -> IO a -> IO [a]` tal que repeat p a realize a ação a repetidamente, acumulando os resultados em uma lista, enquanto o teste p retornar um valor verdadeiro para esse resultado.

   Defina getString em termos de repeat e getChar.
4. Escreva duas definições `accumulate::[IO a] -> IO[a]`, que "realiza uma lista de ações, acumulando o resultado dessas ações em uma lista uma usando combinadores e a outra usando a notação do.
5. Escreva duas definições de `sequencia::[IO a] -> IO()`, que realiza a lista de ações mas descarta os resultados uma usando combinadores e a outra usando a notação do.
6. Use a definição de sequencia para definir uma outra versão de `putStrLn :: IO ()` usando map, putChar e sequência.

Tipos de Dados

1. Considere uma representação de pontos no plano pelo par das suas coordenadas cartesianas: type Ponto = (Float, Float)

   (a) Escreva uma definição da função que calcula a distância euclidiana entre dois pontos (x_1, y_1) e (x_2, y_2): $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
   `dist :: Ponto -> Ponto -> Float`

   (b) Considere agora um percurso dado como uma lista de pontos consecutivos. Escreva uma função `comprimento :: [Ponto] -> Float` que calcule o comprimento total dum percurso (isto é, a soma das distâncias entre pontos consecutivos). Pode usar recursão ou listas em compreensão e deve usar a função dist da alínea anterior para calcular as distâncias. Tenha atenção de tratar corretamente os percursos degenerados (vazios ou com apenas um ponto).

2. Considere um tipo de dados em Haskell para representar proposições lógicas.

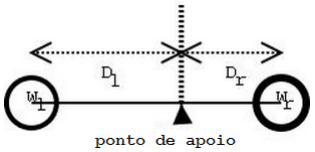
```
data Prop = Var Char
| Neg Prop
| Conj Prop Prop
| Disj Prop Prop
```

   Escreva uma função `contar :: Prop -> [(Char, Int)]` cujo resultado é uma lista de associações entre cada variável e o número de vezes que ocorre na proposição.

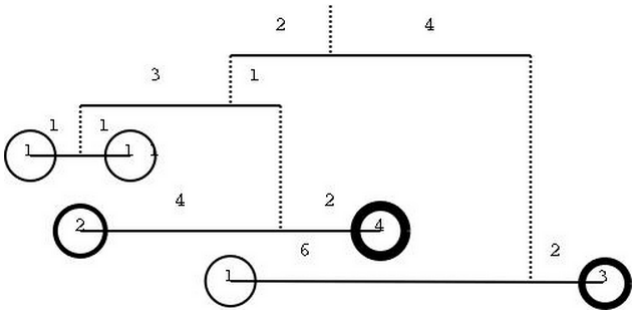
   Exemplo:

   `contar (Conj (Var 'a') (Disj (Var 'b') (Var 'a')) = [('a', 2); ('b', 1)]` (a ordem dos pares no resultado não é importante).
3. Um mobile é uma estrutura constituída por uma haste, a partir da qual objetos ponderados ou outras hastes são penduradas.

   A figura abaixo ilustra um mobile. É uma haste suspensa por uma corda, com objetos pendurados de cada lado. Também pode ser visto como uma espécie de alavanca com o ponto de apoio sendo o ponto em que a haste está suspensa pela corda. Pelo princípio da alavanca, sabemos que equilibrar um móvel temos que o produto do peso dos objetos pela distância ao ponto de apoio devem ser iguais, ou seja, $Wl \times Dl = Wr \times Dr$ onde Dl é a distância da esquerda, Dr é a distância da direita, WL é o peso da esquerda e Wr é o peso da direita.



   Num sistema mais complexo, um objeto pode ser substituído por um sub-mobile, tal como mostrado na figura a seguir. Neste caso, não é tão simples para verificar se um mobile está em equilíbrio por isso precisamos de você para escrever um programa que, recebe um mobile como entrada, verifica se o móvel está em equilíbrio ou não.



```
data Mobile = Haste Mobile Int Mobile Int | Objeto Int

m1 = Haste (Objeto 1) 6 (Objeto 3) 2
m2 = Haste (Objeto 2) 4 (Objeto 4) 2
m3 = Haste (Objeto 1) 1 (Objeto 1) 1
m4 = Haste (m3) 3 (m2) 1
m5 = Haste (m4) 2 (m1) 6
```

   (a) Escreva uma função `peso:: Mobile -> Int` que dado um Mobile retorna o peso sustentado por ele.

   (b) Escreva uma função `equilibrio:: Mobile -> Int` que dado um Mobile retorna True, se todo o sistema está em equilíbrio,caso contrário, retorna False.

4. Considere a definição em Haskell dum tipo de dados para multiconjuntos (i.e. coleções sem ordem mas com repetições) representado como árvore de pesquisa:

   `data MConj a = Vazio | No a Int (MConj a) (MConj a)`

   Cada nó contém um valor e a sua multiplicidade (i.e. o número de repetições); para facilitar a pesquisa, a árvore deve estar ordenada pelos valores. Por exemplo: `No 'A' 2 Vazio (No 'B' 1 Vazio Vazio)` representa o multi-conjunto {A, A, B} com dois caracteres 'A' e um 'B'.

   (a) Escreva uma definição recursiva da função `ocorre :: Ord a => a -> MConj a -> Int` que procura o número de ocorrências de um valor num multi-conjunto; o resultado deve ser 0 se o valor não pertencer ao multi-conjunto.

   (b) Escreva uma definição recursiva da função `inserir :: Ord a => a -> MConj a -> MConj a` que insere um valor num multiconjunto mantendo a árvore de pesquisa ordenada.

5. Vamos representar pontos de um plano cartesiano com duas coordernadas e regiões desse plano como funções, usando os seguintes tipos:

```
data Ponto = Pt Float Float
type Regiao = Ponto -> Bool
```

   Se r representa uma região do plano, então um ponto p está nessa região do plano se r p é igual a True.

   (a) Defina funções `retang::Ponto->Ponto->Regiao` e `circ::Ponto->Raio->Regiao` tais que:
   retang p q retorne (a região que representa) o retângulo tal que p é o ponto mais à esquerda e mais baixo, e q o ponto mais à direta e mais alto.
   circ p r retorne o círculo de raio r e centro p.
   Lembre-se: regiões são representadas por funções.

   (b) Defina funções `uniao::Regiao->Regiao->Regiao`, `interseccao::Regiao->Regiao->Regiao` e `complemento::Regiao->Regiao` tais que p está em uniao r r' se e somente se p está na uniao das regiões r e r', e analogamente para intersecção e complemento.

6. Considere a representação de um grafo com vértices inteiros como um par ordenado uma lista de vértices e uma lista de arestas (isto é, pares ordenados de vertices).

```
type Vert = Int
type Grafo = ( [ Vert ] , [ ( Vert , Vert ) ] )
```

   Escreva uma função caminho :: `Grafo -> [Vert] -> Bool` tal que caminho g xs é True se xs é uma lista de vértices que representa um caminho no grafo (isto é, se cada dois vértices consecutivos correspondem a uma aresta) e False, caso contrário.

   Exemplos:

```
G = ([1,2,3], [(1, 2), (2, 1), (2, 3)])
caminho  G [1, 2, 1, 2, 3] = True
caminho  G [1, 2, 1, 3] = False
```