Exercícios sobre Tipos Algébricos Recursivos II

prof. André Rauber Du Bois

Universidade Federal de Pelotas
http://sites.google.com/site/haskellufpel/
dubois@inf.ufpel.edu.br

1 Questionário

Podemos definir uma lista usando um tipo algébrico recursivo:

```
data Lista a = Vazio | Cons a (Lista a)
```

Na verdade, internamente no interpretador, listas são representadas usando um tipo algébrico muito parecido com esse. Dessa forma, toda a vez que escrevemos a lista

```
[1,2,3,4]
```

internamente essa lista é representada da seguinte forma (aqui usando o tipo `Lista` que criei ali em cima):

```
Cons 1 (Cons 2 (Cons 3 (Cons 4 Vazio)))
```

Obviamente, fica muito mais difícil descrever uma lista dessa forma, mas como listas são muito usadas, o Haskell fornece essas outras notações mais simples. Mas com essa definição de lista usando tipos algébricos, podemos fazer as mesmas coisas que faziamos com as outras listas. Por exempo, a função `tamanho` que definiamos da seguinte forma:

```
tamanho :: [a] -> Int
tamanho [] = 0
tamanho (x:xs) = 1 + tamanho de xs
```

poderiamos definir também para esse novo tipo `Lista` criado:

```
tamanho :: Lista a -> Int
tamanho Vazio = 0
tamanho (Cons x xs) = 1 + tamanho de xs
```

Para os exercícios a seguir, use o tipo `Lista` aqui definido:

1. Definir as funções `mapL`, `foldrL`, `filterL`, `reverseL` e `concatL` que equivalem respectivamente ao `map`, `foldr`, `filter`, `reverse` e `concat` estudados em aula

2. Uma tupla de dois valores pode ser também definida como um tipo algébrico:

   ```
   data Tup a b = Tupla a b
   ```

   Por exemplo, a função `fst` que pega o primeiro elemento de uma tupla

   ```
   fst :: (a,b) -> a
   fst (m,n) = m
   ```

   pode ser definida da seguinte forma usando a nova implementação de tuplas:

   ```
   fst :: Tup a b -> a
   fst (Tupla m n) = m
   ```

   Definir as funções `zip`, `unzip` e `splitAt` usando as novas definições para listas e tuplas.