Princípios de Programação

Trabalho Individual 3

Licenciatura em Engenharia Informática

2016/2017

Pretende-se construir um módulo Haskell que providencie o tipo de dados abstrato **conjunto com etiquetas** (em inglês, *tagged set*).

Um conjunto com etiquetas é um conjunto de elementos, onde cada elemento tem associado um conjunto de *strings* que designamos por **etiquetas**. Sendo um conjunto não há repetição de elementos, e em cada elemento não há repetição de etiquetas (porém, os elementos podem partilhar etiquetas).

As operações associadas a definir no módulo `TaggedSet` são:

- `empty` devolve o conjunto vazio
- `nullSet` verifica se o conjunto dado está vazio
- `belongs` verifica se um elemento pertence ao conjunto
- `singleton` dado um elemento, cria o conjunto com esse elemento (ainda sem etiquetas associadas)
- `insertSet` insere um novo elemento a um conjunto (se o elemento já existir, não altera o conjunto)
- `removeSet` remove um elemento de um conjunto (se o elemento não existir, não altera o conjunto)
- `insertTag` dada uma etiqueta e um elemento, adiciona essa etiqueta ao elemento se e só se o elemento existir no conjunto. Caso contrário devolve o conjunto inicial sem o alterar
- `merge` dado dois conjuntos, devolve um novo conjunto que contém a união dos elementos e, para cada elemento comum, a união das suas etiquetas.

Decidiu-se que o tipo de dados é representado pela definição seguinte:

```
data TaggedSet a = TS [(a,[String])]
```

Assim, as suas assinaturas são:

- `empty :: TaggedSet a`
- `nullSet :: TaggedSet a -> Bool`
- `belongs :: Ord a => a -> TaggedSet a -> Bool`

- `singleton :: a -> TaggedSet a`
- `insertSet :: Ord a => a -> TaggedSet a -> TaggedSet a`
- `removeSet :: Ord a => a -> TaggedSet a -> TaggedSet a`
- `insertTag::Ord a => String -> a -> TaggedSet a -> TaggedSet a`
- `merge :: Ord a => TaggedSet a -> TaggedSet a -> TaggedSet a`

Para visualizarmos o conteúdo de um conjunto de etiquetas incluímos na definição do módulo a seguinte instanciação à classe `Show`:

```
instance Show a => Show (TaggedSet a) where
    show (TS ts) = "{" ++ printTaggedSet ts ++ "}"

printTaggedSet :: Show a => [(a,[String])] -> String
printTaggedSet [] = " "
printTaggedSet [(elem,tags)] = show elem ++ "#" ++ show tags
printTaggedSet ((elem,tags):elems) = show elem ++ "#" ++
                  show tags ++ "," ++ printTaggedSet elems
```

Para além de exportar estas operações, o vosso módulo deve igualmente exportar o tipo `TaggedSet` e o seu construtor `TS`[1].

Em relação à implementação pretende-se que a lista de elementos se mantenha ordenada, bem como cada uma das listas de etiquetas.

Vejamos um exemplo de uso do módulo:

```
import TaggedSet

ts1 = insertTag "todo" 'a' $
      insertTag "ok" 'c' $
      insertTag "urgent" 'c' $
      insertTag "urgent" 'b' $
      insertTag "ok" 'b' $
      insertSet 'b' $
      insertSet 'c' $
      singleton 'a'

ts2 = insertTag "another todo" 'a' $
      insertTag "ok" 'c' $
      insertTag "nok" 'c' $
      insertTag "urgent" 'c' $
      insertTag "ok" 'd' $
      insertSet 'd' $
      insertSet 'c' $
      insertSet 'a' empty
```

[1]Ou seja, o módulo deve começar com `module TaggedSet(TaggedSet(TS),...`

```
main = do putStrLn $ show ts1
          putStrLn $ show ts2
          putStrLn $ show (empty::TaggedSet Int)
          putStrLn $ show $ nullSet ts1
          putStrLn $ show $ belongs 'x' ts1
          putStrLn $ show $ singleton 12
          putStrLn $ show $ insertSet 12 (singleton 23)
          putStrLn $ show $ removeSet 'a' ts1
          putStrLn $ show $ insertTag "todo" 'a' ts1
          putStrLn $ show $ insertTag "todo" 'x' ts1
          putStrLn $ show $ merge ts1 ts2
```

Ao executar este código o resultado deverá ser o seguinte:

```
{'a'#["todo"],'b'#["ok","urgent"],'c'#["ok","urgent"]}
{'a'#["another todo"],'c'#["nok","ok","urgent"],'d'#["ok"]}
{ }
False
False
{12#[]}
{12#[],23#[]}
{'b'#["ok","urgent"],'c'#["ok","urgent"]}
{'a'#["todo"],'b'#["ok","urgent"],'c'#["ok","urgent"]}
{'a'#["todo"],'b'#["ok","urgent"],'c'#["ok","urgent"]}
{'a'#["another todo","todo"],'b'#["ok","urgent"],
'c'#["nok","ok","urgent"],'d'#["ok"]}
```

Notas

1. Deve juntar a assinatura para cada função que escrever.
2. Pode usar as funções do **`Prelude`** e do módulo **`Data.List`**.
3. Este é um trabalho de resolução individual. Os trabalhos devem ser entregues no Moodle até às 23:55 do dia 24 de Novembro de 2016. O nome do ficheiro deve ter o formato `t3_fcXXXXX.hs`, sendo `XXXXX` o seu número de aluno.
4. Os trabalhos serão avaliados semi-automaticamente. Respeite os nomes e as assinaturas das funções.