Examen final de Llenguatges de Programació

Grau en Enginyeria Informàtica
Temps estimat: 2h i 50m
21 de juny de 2016

Els Problemes 1, 2 i 3 s'han de resoldre amb Haskell usant només l'entorn Prelude. Es valorarà l'ús que es faci de funcions d'ordre superior predefinides i l'eficiència i la simplicitat de la solució.

Problema 1 (1.25 punts): *kfibonacci*. Feu una funció `kfib :: Integer -> [Integer]`, que rep un enter k més gran que zero i genera una llista infinita d'enters $a_0, a_1, \ldots$, on els k primers elements són $0, 1, \ldots k-1$ i a partir d'aquí cada nou element és la suma dels k anteriors. Noteu que `(kfib 2)` genera la llista dels nombres de Fibonacci. També temin que

```
kfib 4 = [0,1,2,3,6,12,23,44,85,164,...
kfib 7 = [0,1,2,3,4,5,6,21,42,83,164,325,646,1287,...
```

Problema 2 (1.25 punts): *Cerca múltiple en un BST*. Suposeu que tenim els arbres binaris definits com sempre:

```
data BTree a = Node a (BTree a) (BTree a) | Empty
```

Feu una funció `search :: Ord a => [a] -> BTree a -> Bool`, que donada una llista d'elements ordenada creixentment i un BST (arbre binari de cerca) com a la pràctica (amb els petits a l'esquerra i el grans a la dreta) ens retorna si tots els elements de la llista estan al BST o no. Es valorarà que feu un sol recorregut de l'arbre.

Per exemple, amb el BST

```
bst1 = Node 9 (Node 4 (Node 2 Empty Empty) (Node 6 Empty (Node 7 Empty Empty)))
              (Node 14 (Node 13 (Node 11 Empty Empty) Empty) (Node 16 Empty Empty))
search [2,6,6,9,16] bst1 = True
search [2,6,8,9,16] bst1 = False
```

Problema 3 (2 punts): *Tries*. Volem definir un tipus arbre anomenat *Trie*, per a representar un diccionari on les claus són cadenes de caràcters i els valors són d'un únic tipus general (polimòrfic). Així un possible Trie seria:

```
t1 = TNode Nothing [('s',
                  TNode Nothing [('a',TNode Nothing [('l',TNode (Just 21) [('a', TNode (Just 17) []) ])]),
                                 ('o',TNode Nothing [('l',TNode (Just 38) []),
                                                     ('n', TNode (Just 57) [])
                                                    ])
  ])]
```

que té les parelles clau/valor: `("sal",21), ("sala",17), ("sol",38), ("son",57)`. Noteu estem usant el tipus `Maybe` per representar si un node té clau o no.

1. Definiu el tipus polimòrfic `Trie` més simple que defineixi arbres com el de l'exemple (respectant el nom dels constructors i els tipus).

   - Feu la funció `val :: [Char] -> Trie a -> Maybe a` que ens retorna el valor (encapsulat amb un `Just`) de la clau donada, si existeix, i `Nothing` en cas contrari.
   - Feu la funció `clean :: Trie a -> Trie a` que elimina tots els subarbres del Trie que contenen `Nothing` en tots els nodes. Feu que `Trie a` sigui instance de la classe `Eq`, on dos `Trie` són iguals si el resultat del `clean` és exactament igual. Per exemple, `t1` és igual que el següent `Trie`

```
TNode Nothing [('s',
              TNode Nothing [('a',TNode Nothing [('l',TNode (Just 21) [('a', TNode (Just 17) []) ])]),
                             ('o',TNode Nothing [('l',TNode (Just 38) [('a', TNode Nothing [])]),
                                                 ('n', TNode (Just 57) [])
                                                ]),
                             ('u',TNode Nothing [('r',TNode Nothing [])])
                            ])]
```

Problema 4 (2.5 punts): *Inferència de tipus.* Cal escriure l'arbre decorat de les expressions i generar les restriccions de tipus. Resoleu-les per obtenir la solució. Assenyaleu el resultat final amb un requadre.

1. Tenint en compte que `(:) :: a -> [a] -> [a]`, inferiu el tipus més general de `fun1`:

   ```
   fun1 f x = x:(fun1 f (f x))
   ```

2. Tenint en compte que `(,) :: a -> b -> (a,b)` i que `(>) :: Ord a => a -> a -> Bool` , inferiu el tipus més general de `fun2`:

   ```
   fun2 f g (x,y) = let (n,m) = (f x, g y) in if n > m then (n,y) else (m,y)
   ```

Problema 5 (2.5 punts): *Python.*

1. Feu una funció Python `invert`, que transformi les llistes en tuples i les tuples en llistes fins que troba un objecte que no és de cap d'aquests dos tipus (que el retorna igual). Per exemple

   ```
   print invert(([3,3,(2,1)],5,[1,1,[1,1]]))
   [(3, 3, [2, 1]), 5, (1, 1, (1, 1))]
   print invert(invert(([3,3,(2,1)],5,[1,1,[1,1]])))
   ([3, 3, (2, 1)], 5, [1, 1, [1, 1]])
   ```

2. Volem definir en Python la classe `Trie` seguint la mateixa idea que al problema 3, però en aquest cas les claus són cadenes de caràcters i els valors objectes de qualsevol tipus. La classe, a més de l'operació creadora, ha de tenir les operacions `value`, que donada una clau ens torna el valor associat si hi és i `None` en cas contrari, i `insert` que donada una clau i un valor l'afegeix al `Trie` (si ja existeix sobreescriu el valor). Completeu la implementació que es dona

```
class Trie:                              |      # exemple de crides i resultat
    def __init__  ....                   |       a = Trie()
                                         |       a.insert("sala",16)
                                         |       print a.value("sala")
    def value ....                       |       16
        if len(k) == 0:                  |       a.insert("sal",21)
            return self.val              |       a.insert("sala",17)
        elif k[0] not in self.entries:   |       a.insert("sol",38)
        .                                |       a.insert("son",57)
        .                                |       print a.value("sol")
        .                                |       38
                                         |       print a.value("sala")
    def insert ....                      |       17
                                         |       print a.value("salsa")
                                         |       None
                                         |       print a.value("son")
                                         |       57
                                         |       print a.value("sal")
                                         |       21
```

Problema 6 (0.5 punts): *Conceptes de llenguatges de programació.*

1. Indiqueu les propietats del sistema de tipus de Haskell.

2. Indiqueu quin llenguatge us va tocar en el Treball Dirigit (TD) de Competències Transversals, si és o no de scripting i quins paradigmes inclou. Si té extensions importants, indiqueu quins paradigmes incorporen.