Examen final de Llenguatges de Programació

Grau en Enginyeria Informàtica
Temps estimat: 2h i 45m
25 de juny de 2018

Els Problemes 1 i 2 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.

Si veu treure més d'un 4 del l'examen parcial de Haskell (sense comptar la pràctica), podeu optar per no respondre aquests dos problemes, i en aquest cas la nota d'aquests dos problemes (4.5) serà 4.5*NotaParcialHaskell/10.

Problema 1 (2.5 punts): *Llistes combinades.* Feu les següents funcions:

1. Feu la funció `genAllSize ::[a] -> [[a]]`, que rep una llista l i genera totes les llistes en ordre creixent de mida que es poden construir amb elements de l. Per exemple `genAllSize [2,4,8]` és

   ```
   [[],[2],[4],[8],[2,2],[4,2],[8,2],[2,4],[4,4],[8,4],[2,8],[4,8],[8,8],[2,2,2],[4,2,2],[8,2,2], ...
   ```

   No importa l'ordre en que genereu les d'igual mida.

2. Feu la funció `groupSizes ::[[a]] -> [[[a]]]`, que rep una llista de llistes en ordre creixent de mida i les agrupa per mides tornant una llista que primer té la llista amb les de mida 0, després la llista amb les de mida 1, i així amb totes les mides mentre quedin llistes. Per exemple

   ```
   groupSizes [[],[], [2],[5],[1,8],[9,2],[1,8],[6,1],[3,3,3],[1,5,2],[8,4,4],[5,1,1,2,3]] =
   [[[],[]],[[2],[5]],[[1,8],[9,2],[1,8],[6,1]],[[3,3,3],[1,5,2],[8,4,4]],[],[[5,1,1,2,3]]]
   ```

   Ha de poder funcionar amb llistes infinites.

3. Usant les dues funcions anteriors, feu la funció `combineBySize ::[a] -> [a] -> [[a]]` que donades dues llistes que no comparteixen elements i no tenen repetits, genera totes les llistes en ordre creixent de mida formades pel mateix nombre d'elements de la primera llista que de la segona i amb els elements de la primera llista al davant. Per exemple `combineBySize [2,4,8] [1,5]` és

   ```
   [[],[2,1],[2,5],[4,1],[4,5],[8,1],[8,5],[2,2,1,1],[2,2,5,1],[2,2,1,5],[2,2,5,5],[4,2,1,1],
   [4,2,5,1],[4,2,1,5],[4,2,5,5],[8,2,1,1],[8,2,5,1],[8,2,1,5],[8,2,5,5],[2,4,1,1],.....
   ```

   No importa l'ordre en que genereu les d'igual mida.

Problema 2 (2 punts): *Funcions booleanes amb negació.* Volem representar funcions booleanes amb les operacions `and` i `or` i amb negació però sense `not` i amb strings per representar les variables. Per això, usarem un booleà per indicar si l'expressió és afirmada (usant `True`) o negada (usant `False`).

1. Definiu un data `NBExp` amb els constructors `AND` i `OR` (que reben un `Bool` i dues expressions) i el constructor `Var` (que rep un `Bool` i un `String`) per a les variables. Així la funció booleana $f(x, y, z) = \neg(\ (\neg x \vee y) \wedge \neg(\ z \vee \neg y\ )\ )$ es representa amb l'expressió:

   ```
   AND False (OR True (Var False "x") (Var True "y")) (OR False (Var True "z") (Var False "y"))
   ```

2. Feu la funció `normalize :: NBExp -> NBExp` que donada una funció booleana ens retorna una funció booleana equivalent on només las variables poden estar negades. Això es pot fer aplicant les regles de distribució de la negació sobre la disjunció i la conjunció: $\neg(\ a \vee b) \equiv \neg a \wedge \neg b$) i $\neg(\ a \wedge b) \equiv \neg a \vee \neg b$); i la doble negació: $\neg\neg a \equiv a$. Per exemple, `normalize` aplicat a la funció booleana anterior ens retorna:

   ```
   OR True (AND True (Var True "x") (Var False "y")) (OR True (Var True "z") (Var False "y"))
   ```

Problema 3 (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 -> b -> (a,b)` i `fst :: (a,b) -> a`, inferiu el tipus més general de `fun1`:

   ```
   fun1 x = let y = fst x in (y,y)
   ```

   Heu de transformar el let usant abstracció i aplicació com s'ha vist a classe.

2. Tenint en compte que `(>) :: Ord a => a -> a -> Bool`, `0 :: Num a => a`, `(:) :: a -> [a] -> [a]` i `(.) :: (b -> c) -> (a -> b) -> a -> c`, inferiu el tipus més general de `fun2`:

   ```
   fun2 (x:l) y = (x.y) l > 0
   ```

Problema 4 (2.5 punts): *Python.* Feu una funció Python `encode(s)`, que rep un string no buit i retorna un enter positiu que el representa.

Així, si el string té un únic caràcter, la codificació és el número que representa el caràcter, que s'obté amb `ord(s)`. En altre cas, la codificació s'obté sumant la codificació de s sense el primer caràcter (el sufix) i la codificació de s sense l'últim (el prefix). Per exemple

```
encode("h") = 104
encode("o") = 111
encode("l") = 108
encode("a") = 97
encode("hola") = 858
encode("Pyhon es un llenguatge de scripting") = 1765361508634
```

La vostra solució ha de ser eficient i evitar repetir càlculs.
Podeu usar funcions auxiliars, però no han de ser visibles. És a dir, l'única funció executable des de fora (per exemple, des del programa principal) ha de ser `encode`.
Recordeu que en Python, `s[n:]` és el string sense els n primers caràcters de s i `s[:-n]` és el string sense els darrers n caràcters de s.

Problema 5 (0.5 punts): *Conceptes de llenguatges de programació.*

1. Indiqueu què significa que el "tipat" en un llenguatge de programació sigui estàtic o dinàmic.

2. Indiqueu si el tipat és dinàmic o estàtic en el llenguatge que us va tocar en el Treball Dirigit (TD) de Competències Transversals.