Llenguatges de Programació, FIB, 26 de gener de 2018

L'examen dura 3 hores. Es valorarà la concisió, claredat i brevetat a més de la completesa i l'exactitud de les respostes. No es pot consultar cap material addicional.

Els qui tingueu nota ≥ 4 al parcial de Haskell, podeu no fer els problemes 4 i 5.

1. Entrants (2 punts)

1. **Cultura dels LP.** Quina relació tenien el senyor Haskell (el del llenguatge) i el senyor Curry (el de la *currificació*)?

2. **λ-càlcul.** Siguin $T \equiv \lambda xy.x$ i $F \equiv \lambda xy.y$ termes per a cert i fals respectivament. Comproveu que $NEG \equiv \lambda x.(xFT)$ és la negació de booleans tot construint la seva taula de veritat. Mostreu cada β-reducció aplicada.

3. **Python.** Implementeu una funció *compose*(f, g) que, donades dues funcions f i g d'un sol paràmetre, retorni una funció que sigui llur composició ($f \circ g$). Per exemple, fent

   def $f(x)$: **return** x+1
   def $g(x)$: **return** 2$*x$
   h = *compose*(f, g)

   llavors $h(2)$ hauria de valer 5.

4. **Sistemes de tipus.** Indiqueu les propietats del sistema de tipus del llenguatge que us va tocar en el Treball Dirigit de Competències Transversals (digueu quin era el llenguatge!).

2. Inferència de tipus (2 punts)

Recordeu la funció *fsmap* del parcial? Tant, és... Aquí la teniu:

fsmap = **foldl** \$ **flip** (\$)

Inferiu el tipus més general de *fsmap* utilitzant els tipus següents:

foldl :: $(b \to a \to b) \to b \to [a] \to b$
flip :: $(a \to b \to c) \to b \to a \to c$
(\$) :: $(a \to b) \to a \to b$

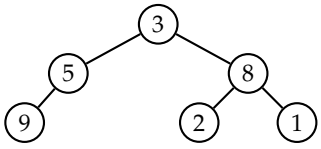
Per a fer-ho, dibuixeu l'arbre decorat de les expressions i genereu les restriccions de tipus. Resoleu-les per obtenir la solució i assenyaleu el resultat final amb un requadre.

3. Arbre general a arbre binari en Python (2 punts)

Considerem que enmagatzem arbres *binaris* utilitzant nodes formats per tres elements desats en una llista: el valor del node, el subarbre esquerre i el subarbre dret. L'arbre buit es representa amb *None*. Per exemple,

[3, [5, [9, *None*, *None*], *None*], [8, [2, *None*, *None*], [1, *None*, *None*]]]

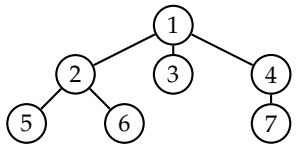
representa aquest arbre binari:



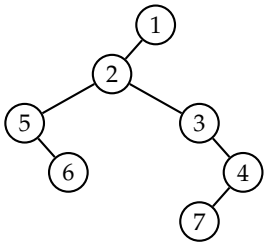
Considerem també que enmagatzem arbres *generals* utilitzant tuples de dos elements: el primer camp és el valor del node, el segon camp és una llista amb els seus fills. No hi ha arbres generals buits, però les fulles tenen una llista buida de fills. Per exemple,

(1, [(2, [(5, []), (6, [])]), (3, []), (4, [(7, [])])])

representa aquest arbre general:



Es pot convertir un arbre general a un arbre binari equivalent tot utilitzant el mètode de posar el primer fill com a subarbre esquerre i els demés fills com a subarbres cap a la dreta (*left-child right sibling representation*). Per exemple, la transformació a arbre binari de l'arbre general anterior és:



que es representa així:

[1, [2, [5, *None*, [6, *None*, *None*]], [3, *None*, [4, [7, *None*, *None*], *None*]]], *None*]

Feu una funció que, donat un arbre general, el transformi a arbre binari. Es valorarà l'ús d'alguna funció d'ordre superior de Python a la vostra implementació. 10 línies de codi són suficients.

4. Programa misteriós **(2 punts)**

Considereu el programa següent en Haskell:

```
import Data.List (sort)

foo = unlines . reverse . take 5 . sort . filter (not . odd . length) . lines

main = do
    bar ← getContents
    let norf = foo bar
    putStr norf
```

1. Quin és el tipus de (.) ?
2. Què fa aquest programa?
3. Tranformeu el seu *main* de notació **do** a notació purament funcional.

5. Fluffy i misty **(2 punts)**

Definiu les funcions marcades amb punts suspensius amb una implementació no trivial que respecti els tipus.

```
class Fluffy f where
    furry :: (a → b) → f a → f b

instance Fluffy Maybe where                    -- Apartat 5.1
    furry = ···

instance Fluffy [] where                       -- Apartat 5.2
    furry = ···

class Misty m where                            -- Apartat 5.3
    banana  :: (a → m b) → m a → m b
    unicorn :: a → m a
    faun    :: (a → b) → m a → m b
    faun = ···
        -- useu banana i/o unicorn

instance Misty Maybe where                     -- Apartat 5.4
    banana = ···
    unicorn = ···

instance Misty [] where                        -- Apartat 5.5
    banana = ···
    unicorn = ···
```