Llenguatges de Programació, FIB, 17 de gener de 2022

L'examen dura tres hores. Es valorarà la concisió, claredat i brevetat a més de la completesa i l'exactitud de les respostes. Feu bona lletra. Poseu el vostre nom a cada full. Contesteu els problemes 1 a 4 en fulls diferents dels problemes 5 a 7.

1 Pragmàtica dels LPs (1 punt)

Perquè els llenguatges de la família del C (com C, C++, Java, ...) requereixen parèntesis al voltant de les condicions dels **if**s i **while**s?

Limiteu la vostra resposta a cinc línies o menys.

2 λ-cálculo (1 punt)

Sabiendo que en el λ-cálculo **true** se representa por $\lambda xy\ .\ x$ y **false** por $\lambda xy\ .\ y$, se pide demostrar que $\lambda ab\ .\ ab(\lambda xy\ .\ y)$ es la implementación de la función **and**.

3 Subtipos (1 punt)

Explica si los punteros se pueden considerar covariantes, contravariantes o invariantes, justificando la respuesta. Es decir, que suponiendo que t_1 es subtipo de t_2, si se puede considerar que t_1* ha de ser subtipo de t_2* o si, por el contrario, t_2* ha de ser subtipo de t_1* o si no se puede considerar ninguna de las dos cosas.

4 Inferència de tipus (2 punts)

Aquest *one-liner* trobat per Internet calcula el producte cartesià de dues llistes:

$$\begin{array}{lcl} \mathit{cartesian} & :: & [a] \to [a] \to [[a]] \\ \mathit{cartesian} & = & ((.)\ \ \mathit{sequence}) \quad . \quad (\backslash x\ y \to [x, y]) \end{array}$$

La definició sembla funcionar (per exemple, *cartesian* [1,2,3] [1,2] retorna el que cal: [[1,1], [1,2], [2,1], [2,2], [3,1], [3,2]]) però, malauradament, no sabem què és la funció *sequence*.

Inferiu el tipus de *sequence* seguint els passos següents:

1. Dibuixeu l'arbre de sintàxi abstracta de la definició de *cartesian* .
2. Anoteu l'arbre amb els tipus de cada node.
3. Genereu el conjunt de restriccions d'igualtat de tipus.
4. Resoleu les restriccions per trobar el tipus de *sequence*.

5 Python (1 punt)

Implementeu en Python els decoradors *make_bold*, *make_italic* , i *make_underline* que permetin formatejar en negreta, itàlica o subratllat d'HTML els textos retornats per altres funcions. Per exemple, amb

```
@make_bold
@make_italic
@make_underline
def foo ():
    return "foo"
```

```
@make_italic
def bar ():
    return "bar"
```

la crida a *foo* () hauria de retornar `"<b><i><u>foo</u></i></b>"` i la crida a *bar* () hauria de retornar `"<i>bar</i>"`.

6 Haskell (2 punts)

Recordeu les definicions de les classes *Functor* i *Monad*:

```
class Functor f where
    fmap :: (a → b) → (f a → f b)

class Monad m where
    return :: a → m a
    (>>=) :: m a → (a → m b) → m b
```

Recordeu també les lleis dels functors:

- *fmap* **id** = **id**
- *fmap* (*f* . *g*) = *fmap f* . *fmap g*

Considereu aquest tipus de dades per a arbres binaris amb informacions a les fulles:

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
    deriving (Show)
```

1. Feu que *Tree* sigui instància de *Functor*.
2. Demostreu que la vostra instanciació compleix les lleis dels functors.
3. Feu que *Tree* sigui instància de *Monad* (ignoreu *Applicative*).
4. Escriviu una funció

   replace :: *Eq a* ⇒ [(*a*, *b*)] → *Tree a* → *Tree* (*Maybe b*)

   que reemplaci les ocurrències de tipus *a* en un *Tree a* per quelcom de tipus *Maybe b* buscant a la llista associativa. Per exemple,

   - *replace* [] (*Leaf* 'x') = *Leaf Nothing*
   - *replace* [('x', 3)] (*Leaf* 'x') = *Leaf* (*Just* 3)
   - *replace* [('x', 3)] (*Node* (*Leaf* 'x') (*Leaf* 'y'))
     = *Node* (*Leaf* (*Just* 3)) (*Leaf Nothing*)

   Per definir el vostre *replace* , heu d'usar (forçosament) el >>= de l'apartat anterior i podeu usar la funció estàndard *lookup* :: *Eq a* ⇒ *a* → [(*a*, *b*)] → *Maybe b*.

7 Compilació (2 punts)

A continuació teniu un fragment d'una gramàtica ANTLR per la definició de tipus al llenguatge de programació Standard ML:

```
grammar LittleML ;

root    : type EOF
        ;
type    : type ID                          #typeType
        | '(' type (',' type)+ ')' ID      #typeApp
        | type '*' type                    #typePair
        | <assoc=right> type '->' type     #typeArrow
        | '{' record? '}'                  #typeRecord
        | '(' type ')'                     #typeParen
        | TVAR                             #typeVar
        | ID                               #typeName
        ;
record  : ID ':' type (',' record)?
        ;
ID      : [A-Za-z] [A-Za-z0-9]* ;
TVAR    : '\'' [A-Za-z0-9]+     ;
WS      : [ \t\n]+ -> skip      ;
```

1. Digueu quins visitadors crea ANTLR per la classe `LittleMLVisitor`.
2. Digueu quines de les entrades següents són vàlides per a la gramàtica anterior. En cas negatiu, expliqueu perquè; en cas afirmatiu, doneu el seu arbre de parsing.
   (a) `unsigned_int`
   (b) `int list tree`
   (c) `int * float + char`
   (d) `(int, char) tree`
   (e) `{x:int, y:float}`
   (f) `'a 'b -> int`
3. Per a casdacuna de les entrades vàlides anteriors, digueu en quin ordre es cridaran els visitadors quan es parseji.
4. Expliqueu raonadament si, tal com es troba, aquesta gramàtica es pot parsejar amb un parser LL(1).