Llenguatges de Programació, FIB, 18 de juny de 2020

Possibles solucions

1. El Trivial d'LP

1. a: LISP, b: ADA, c: Perl, d: Pascal, e: COBOL, f: Python.
2. FORTRAN.
3. COBOL.
4. LISP o Prolog. [Avui en dia Python.]
5. Un llenguatge de programació és *type safe* quan les úniques operacions que es poden aplicar a les dades són les que permetenen el tipus de les dades. En aquest sentit, Java està dissenyat per ser *type safe*. [Realment és molt més complicat, però caldria molt d'espai i tecnicismes per explicar-ho.]
6. Transformar una funció que accepta n paràmetres i convertir-la en una funció que, donat un paràmetre (el primer) retorna una funció que accepta $n-1$ paràmetres (i són semànticament equivalents).
7. C [amb Assembler].
8. Simula 67 [tot i que hi ha trets de l'OOP moderna que no hi eren].
9. Una funció que transforma una funció en una altra funció amb determinada funcionalitat afegida, modificant el seu comportament original. / Una notació per retornar noves funcions a partir de funcions.
10. Smalltalk.
11. ADA.
12. Una funció que captura el seu context lèxic en el moment de la seva creació.
13. Una classe abstracta és una classe que té alguns mètodes declarats però no implementats.
14. Java utilitza pas per valor (en el cas d'objectes, passa per valor referències als objectes).
15. La Màquina de Turing és un model matemàtic. Com a tal, no té un cost.

2. Haskell

TreeFold:

```
treeFold _ z Empty = z
treeFold f z (Node x l r) = f x (treeFold f z l) (treeFold f z r)
```

Funcions:

```
size = treeFold plusOne 0
    where plusOne _ x y = x + y + 1

height = treeFold maxPlusOne 0
    where maxPlusOne _ x y = 1 + max x y

treeMap f = treeFold build Empty
    where build x l r = Node (f x) l r

inOrder = treeFold cat []
    where cat x l r = l ++ x : r

isBST = isSorted . inOrder
    where isSorted xs = and zipWith(<)xs tail xs
```

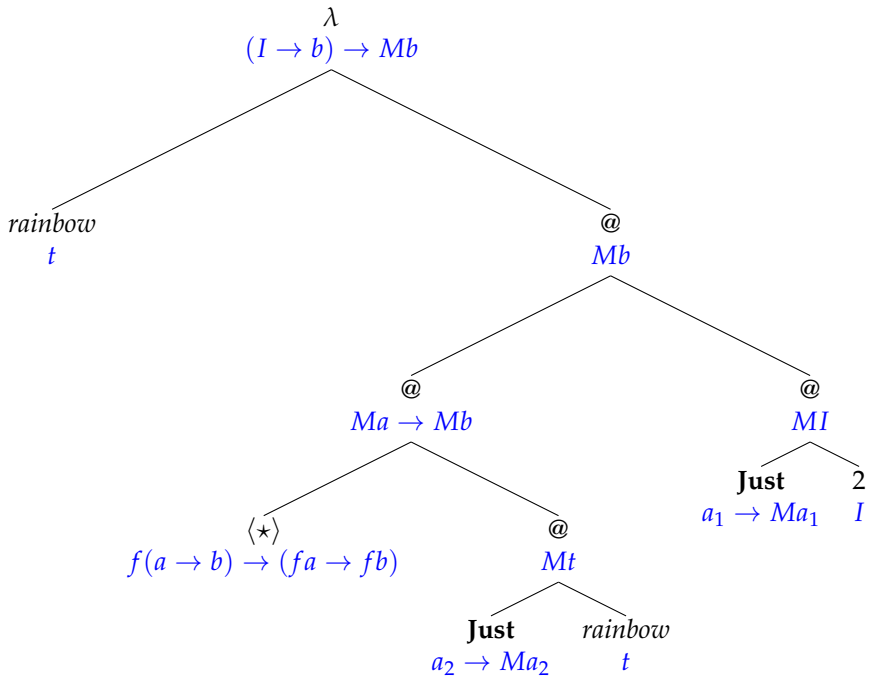
Instanciació:

```
instance Functor Tree where
    fmap = treeMap

instance Show a ⇒ Show (Tree a) where
    show = treeFold sh "*"
        where sh x l r = concat ["<", show x, " ", l, " ", r, ">"]
```

3. Inferència de tipus

Un cop dibuixat l'arbre de sintàxi abstracta (negre), etiquetem les fulles amb els seus tipus (blau) i, de les fulles cap a l'arrel podem inferir tots els tipus dels nodes.



Hem usat I i M com a abreviatura d'**Int** i de **Maybe** i hem trobat que $a_1 = I = a$, $a_2 = t = a \to b$ i $f = M$. Per tant, tenim *unicorn* :: (**Int** $\to b$) $\to$ **Maybe** b.

4. Python

- Donat un natural n, *mystery1(n)* retorna el factorial de n.
- Donat un natural n, *mystery2(n)* retorna l'n-èsim nombre triangular, és a dir, $\sum_{i=1}^{n} i = n(n+1)/2$.
- Donat un natural n, *mystery3(n)* retorna $n+1$.
- Donat un natural n, *mystery4(n)* retorna la llista $[n, n-1, n-2, ..., 1]$.
- Donat un natural n, *mystery5(n)* retorna una aproximació de π (més exacta quan més gran sigui la n).
- Donat un natural n, *mystery6(n)* retorna les 2^n llistes de n zeros o uns.

El cost de *mystery4(n)* és $O(n^2)$ perquè cada concatenació té un cost lineal.

Error freqüent: No especificar que n ha de ser un natural. Amb un nombre negatiu o real, les funcions es penjarien. Amb un paràmetre d'un altre tipus, petarien.

5. Compilació

1. Una gramàtica és ambigua si una mateixa expressió pot tenir dues derivacions diferents.

2. Hi ha moltes solucions possibles, però una que modifica mínimament la segona solució de l'enunciat i és fàcilment entendible seria:

grammar *Expr*;

prog : *expr EOF* ;

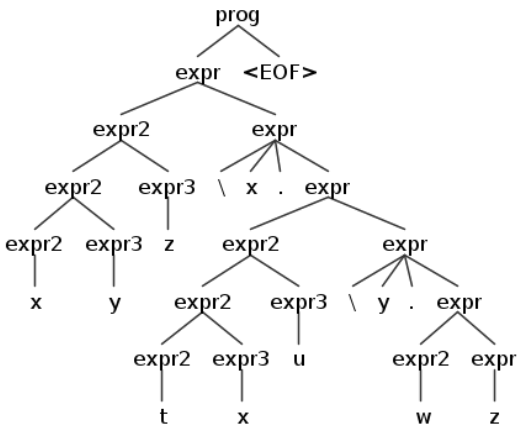
expr : *VAR*
| '\\' *VAR* '.' *expr*
| '(' *expr* ')'
| *expr2 expr*
;

expr2 : *VAR*
| '(' *expr* ')'
| *expr2 expr3*
;

expr3 : *VAR*
| '(' *expr* ')'
;

VAR : [*a*–*z*]
;

Que per l'entrada *x y z* *x . t x u* *y . w z* generaria:



La idea és que *expr* pugui derivar en qualsevol expressió, *expr2* pot derivar en una seqüencia d'aplicacions que no conté cap lambda-expressió a nivell extern, i *expr3* és encara més restringit i no conté ni lambda-abstraccions ni aplicacions a nivell extern, o sigui és simplement un identificador o una expressió encapsulada entre parèntesis.