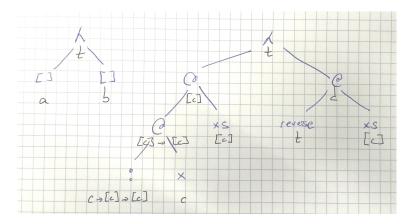
Llenguatges de Programació, FIB, 14 de gener de 2020

Possibles solucions

1. Inferència de tipus

(1'5 punts)

a) Aquest és l'arbre etiquetat amb els tipus, aplicant les unificacions trivials de baix a dalt:



Les equacions que s'obtenen són:

$$t = [a] \to [b]$$
$$t = [c] \to d$$

Per tant, a = c i d = [b] i llavors $t = [a] \rightarrow [b]$ no es pot simplificar més.

- *b*) El programador hauria d'adonar-se que ha comès un error perquè revessar una llista hauria de ser de tipus $[a] \rightarrow [a]$.
- c) Com que que el tipus obtingut sense la declaració de tipus és més general que l'inferit, la comprovació de tipus no hagués detectat cap error, senzillament hauria identificat que a=b.

2. Lleis de functors i mònades en Haskell

a)

instance Functor Maybe where fmap f Nothing = Nothing fmap f (Just x) = Just (f x)

- 1. La primera llei es pot comprovar amb dos casos: valors **Nothing** i valors **Just** *x*:
 - fmap id Nothing \equiv Nothing \equiv id Nothing \checkmark
 - fmap id (Just x) \equiv Just (id x) \equiv Just x \equiv id (Just x) \checkmark
- 2. La segona llei es pot comprovar també amb els mateixos dos casos:
 - Per una banda: fmap (g1 . g2) Nothing \equiv Nothing i, per altra banda: ((fmap g1).(fmap g2)) Nothing \equiv fmap g1 (fmap g2 Nothing) \equiv fmap g1 Nothing \equiv Nothing \checkmark

• Per una banda: $fmap\ (g1 . g2)\ (Just\ x) \equiv Just\ ((g1 . g2)\ x)\ i$, per altra banda: $((fmap\ g1)\ .\ (fmap\ g2))\ (Just\ x) \equiv fmap\ g1\ (Just\ (g2\ x))$ $\equiv Just\ (g1\ (g2\ x)) \equiv Just\ ((g1\ . g2)\ x)\ \checkmark$

b)

instance Monad Maybe where

return = Just
Nothing
$$\gg = f$$
 = Nothing
Just $x \gg = f = f x$

- 1. La primera llei es pot comprovar així:
 - return $x \gg = f \equiv \text{Just } x \gg = f \equiv f x \checkmark$
- 2. La segona llei es pot comprovar també amb els dos casos:
 - Nothing \gg return \equiv Nothing \checkmark
 - Just $x \gg = \text{return } \equiv \text{ return } x \equiv \text{ Just } x \checkmark$
- 3. La tercera llei es pot comprovar també amb els dos casos:
 - Per una banda, (Nothing $\gg = f$) $\gg = g \equiv$ Nothing $\gg = g \equiv$ Nothing. Per l'altra banda, Nothing $\gg = (\ \ x \to f \ x \gg = g) \equiv$ Nothing. \checkmark
 - Per una banda, (Just $x \gg = f$) $\gg = g$ \equiv $f x \gg = g$. Per l'altra banda, Just $x \gg = (\setminus x \to f x \gg = g)$ \equiv $(\setminus x \to f x \gg = g)$ $x \equiv f x \gg = g$. \checkmark

3. Python

- a) El programa escriu 0 0 0 seguit de 2 0 2.
- b) El programa escriu ['unicorn', 'rainbow'].

c) **def** nombres(i): **while** True:

yield i i += 1

d)

def *filtre* (*s*, *p*):
 for *x* **in** *s*:
 if *p*(*x*):
 yield *x*

e) Python és un llenguatge amb tipat fort i dinàmic.

4. Herència

- a) La segona i la tercera assignacions són incorrectes.
- b) A a1.x2, la variable a1 que és de tipus C1 no té un atribut x2.
- c) En els dos casos es crida al *m* de C3.
- d) La segona i la tercera assignacions són incorrectes.
- e) A a1.x2, la variable a1 que és de tipus C1 no té un atribut x2.
- f) En el primer cas es crida al m de C3 i en el segon al de C.

5. Compilació

```
grammar Gestor;
      : comandes* EOF;
root
comandes : ID ASSIGN expr
            | (LE | WI | ML | MW) param_expr
            : PARO exp PARC
expr
            | expr SQ expr
            | expr PA expr
            | expr AT num
            | MKDIR param_pas
            | MKFILE param_pas
            | mv
            | ID
           : PARO ID PARC ;
param_pas
            : MV PARO (mv | expr) COM expr PARC
mv
LE
           : 'length';
WI
           : 'width';
ML
MW
          : 'maxlength';
MW : 'maxwidth';

MKDIR : 'mkdir';

MKFILE : 'mkfile';

MV : 'maxwidth';
           : 'move';
MV
: [a-z]+;
NUM : [0-9]+;
ASSIGN : '.-'
           : ';';
SQ
           : ',';
COM
           : '<->' ;
PA
AT
           : ,^, ;
           : '(';
PARO
            : ')';
PARC
WS
           : [ \n] + -> skip ;
```

6. Avaluació Única Haskell

```
--a) eval :: Expr -> Mem -> Int
eval (Val x) m = x
eval (Var v) m =
   case Map.lookup v m of
        Nothing -> 0
        Just x -> x
eval (Add e1 e2) m = eval, e1 e2 m (+)
eval (Sub e1 e2) m = eval' e1 e2 m (-)
eval (Neq e1 e2) m = bool2int $ eval' e1 e2 m (/=)
eval (Lth e1 e2) m = bool2int $ eval' e1 e2 m (<)
eval' e1 e2 m op = op (eval e1 m) (eval e2 m)
bool2int = fromEnum -- també es pot més llarg fer amb condicionals
--b) exec :: Instr -> Mem -> Mem
exec (Ass v e) m = Map.insert v (eval e m) m
exec (Seq []) m = m
exec (Seq (i:is)) m = exec (Seq is) (exec i m)
exec (Cond b i1 i2) m = exec (if eval b m /= 0 then i1 else i2) m
exec (Loop b i) m =
    if eval b m \neq 0
        then exec (Loop b i) (exec i m)
        else m
-- c) exec es penja si el programa es penja
-- d) run :: Instr -> IO ()
run i = mapM_ printVar $ sort $ Map.keys m
        m = exec i Map.empty
        printVar k = do
            putStr k
            putStr " -> "
            print $ fromJust $ Map.lookup k m
```