

- Per una banda: $fmap\ (g1\ .\ g2)\ \mathbf{Nothing} \equiv \mathbf{Nothing}$ i, per altra banda:
 $((fmap\ g1).(fmap\ g2))\ \mathbf{Nothing} \equiv fmap\ g1\ (fmap\ g2\ \mathbf{Nothing})$
 $\equiv fmap\ g1\ \mathbf{Nothing} \equiv \mathbf{Nothing} \checkmark$

- Per una banda: $fmap (g1 \cdot g2) (\mathbf{Just} \ x) \equiv \mathbf{Just} ((g1 \cdot g2) \ x)$ i, per altra banda: $((fmap \ g1) \cdot (fmap \ g2)) (\mathbf{Just} \ x) \equiv fmap \ g1 (\mathbf{Just} \ (g2 \ x)) \equiv \mathbf{Just} (g1 \ (g2 \ x)) \equiv \mathbf{Just} ((g1 \cdot g2) \ x) \checkmark$

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 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 $\gg=$ return \equiv return x \equiv 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 $\gg=$ ($\backslash x \rightarrow f \ x \gg= g$) \equiv Nothing.** \checkmark
- Per una banda, **(Just x $\gg=$ f) $\gg=$ g \equiv f x $\gg=$ g.** Per l'altra banda, **Just x $\gg=$ ($\backslash x \rightarrow f \ x \gg= g$) \equiv ($\backslash x \rightarrow f \ x \gg= g$) 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;

root : comandes* EOF;

comandes : ID ASSIGN expr
| (LE | WI | ML | MW) param_expr
;

expr : PARO exp PARC
| expr SQ expr
| expr PA expr
| expr AT num
| MKDIR param_pas
| MKFILE param_pas
| mv
| ID
;

param_pas : PARO ID PARC ;

mv : MV PARO (mv | expr) COM expr PARC
;

LE : 'length' ;

WI : 'width' ;

ML : 'maxlength' ;

MW : 'maxwidth' ;

MKDIR : 'mkdir' ;

MKFILE : 'mkfile' ;

MV : 'move' ;

ID : '[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 /= 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
  where
    m = exec i Map.empty
    printVar k = do
      putStr k
      putStr " -> "
      print $ fromJust $ Map.lookup k m
```