① Propietat punt fixe $\Theta R = R(\Theta R)$

$$\Theta = (\lambda xy.\, y(xxy))(\lambda xy.\, y(xxy))$$

$$\Theta R = \lambda xy.\, y(xxy)\,(\lambda xy.\, y(xxy))\, R \xrightarrow{\text{reduim de forma normal}}$$

$$\lambda y.\, y((\lambda xy.\, y(xxy))(\lambda xy.\, y(xxy))y)R \xrightarrow{\text{reduim de forma normal}}$$

$$R(\underbrace{(\lambda xy.\, y(xxy))(\lambda xy.\, y(xxy))}_{\Theta})R = R(\Theta R)$$

② EOF root;

```
root = root lambdaG
     |
     ;

lambdaG = 'λ' [a-z]+ '.' lambdaG
        | '(' lambdaG ')'
        | [a-z]*
        | lambdaG lambdaG
        ;
```

No seria una LL(1) ja que és una gramàtica ambigua.

LP - FIB

Mark Smithson Rivas

21755150W

③ <u>Llei 2</u>:

fmap id = id

~~fmap Empty = Empty~~

~~Empty = Empty~~ ✓

<u>2.1)</u> fmap id Empty = id Empty

Empty = Empty ✓

~~fmap~~

<u>2.2)</u> fmap id (Node v t1 t2) = id (Node v t1 t2)

Node (id v) Empty Empty = (Node v t1 t2) ✗

Aquesta no és complex

<u>Llei 1</u>: ~~2.2~~ 2.1) fmap (g1 . g2) = fmap g1 . fmap g2

fmap (g1 . g2) Empty = fmap g1 . fmap g2 Empty

Empty = Empty ✓

<u>2.2)</u> ~~fmap (g1 . g2) (Node v t1 t2) = fmap (Node v t1 t2)~~ g1 .

~~fmap t~~

fmap (g1 . g2) (Node v t1 t2) = fmap g1 . fmap g2 (Node v t1 t2)

= Node (g1 . g2 v) Empty Empty = fmap g1 (Node (g2 v) ~~t1 t2~~ Empty Empty)

= Node (g1 . g2 v) Empty Empty = Node (g1 (g2 v)) Empty Empty ✓

Per tant complex la 1 però no la 2.

LP-FIB

Mark Smithson Rivas

21755150W

4 1) a) join Nothing

```
do
  m ← Nothing
  m
```

Nothing >>= \m → m =

= Nothing >>= Just m = Nothing

b) join (Just (Just 3))

```
do
  m ← Just (Just 3)
  m
```

Just (Just 3) >>= \m → m =

= Just (Just 3) >>= Just m =

= Just (Just (Just 3)) = 3

c) join (Just Nothing)

```
do
  m ← Just Nothing
  m
```

Just Nothing >>= \m → m =

= Just Nothing >>= Just m =

= Just (Just Nothing) = Nothing

d) join ([[1,2],[3,4]])

```
do
  m ← [[1,2],[3,4]]
  m
```

[[1,2],[3,4]] >>= \m → m =

=[[1,2],[3,4]] >>= [m] = concat (map [m] [[1,2],[3,4]]) =

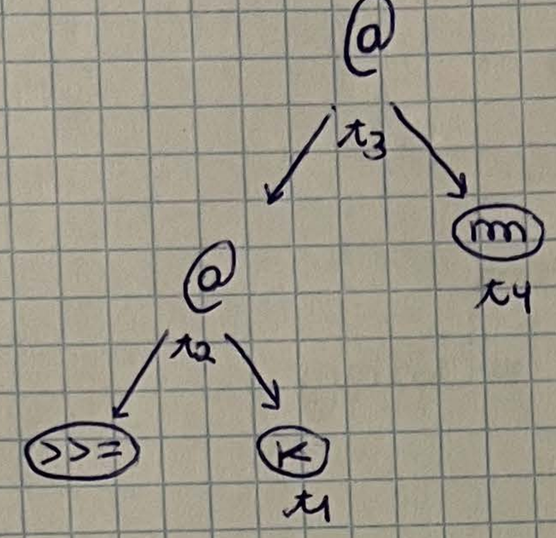
= [1,2,3,4]

2) join k = do

m ← k

m

~~k ←~~ k >>= \m → m



ma → (a → mb) → mb

ma → (a → mb) → mb = t1 → t2

t2 = t4 → t3

t1 = ma

t2 = (a → mb) → mb

t4 = (a → mb)

t3 = mb

Per tant el tipus de join és <u>mb</u>

LP-FIB

Mark Smithson Rivas

21755150W

5 1.) covariant: tenim t_1 i t_2 com dos tipus ~~diferents~~ qualsevols diem que T és covariant si és complex que

$t_1 \leq t_2$ llavors ~~[illegible]~~ $T<t_1> \leq T<t_2>$

contravariant: tenim t_1 i t_2 com dos tipus ~~diferents~~ qualsevols diem que T és contravariant si és complex que

$t_1 \leq t_2$ llavors $T<t_1> \geq T<t_2>$

invariant: si no és ni covariant ni contravariant

2.) El constructor de tipus tuple és ~~contravariant~~ covariant ja que tenim Gos $\leq$ Animal però després a les assignacions veiem que la correcta és quan tuple [Gos, Animal] $\leq$ tuple [Animal, Animal]

3) El constructor de tipus list és invariant ja que tenim que Gos $\leq$ Animal però després veiem com List [Gos] $\leq$ List [Animal] no es compleix i List [Animal] $\leq$ List [Gos] tampoc.

4) La raó és perquè en cas que tinguessin la mateixa variància perdria el sentit de cada un d'aquests tipus de dades a l'hora de programar.

LP-FIB

Mark Smithson Rivas

21755150W

⑥ 1.) a) Error ja que e no té g la funció g en el seu tipus estàtic per tant donarà error de compilació. La funció g que es crida es la de Empleat

b) Error ja que e.g(v) retorna un Empleat i v és un Venedor per tant, error de tipus.

c) Tot correcte i la funció h la crida Comptable i la g també

d) Tot correcte i la funció g h la crida ~~Empleat~~ Comptable i la g Empleat

2.)

```
class Empleat {
    int code, m;
    Empleat g (Empleat x) {
        if (code == x.code) return x; else return this;
    }

    void h (int x) { m = x;}
}
class Venedor: Empleat {
    int n;
    Venedor g (Venedor x) {
        if (code == x.code) return this; else return x;
    }
    void g (int x) { m = x;}

class Comptable: Empleat {
    int a;
    Comptable g (Comptable x)
        if (code == x.code) return x, else return this;
    }
    void h (int x) { a = x;}
}
```