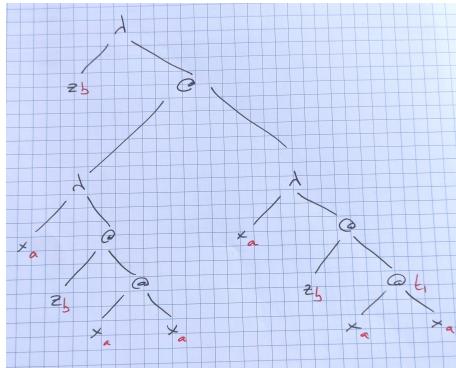


Llenguatges de Programació, FIB, 10 de juny de 2022

Possibles solucions i errors freqüents

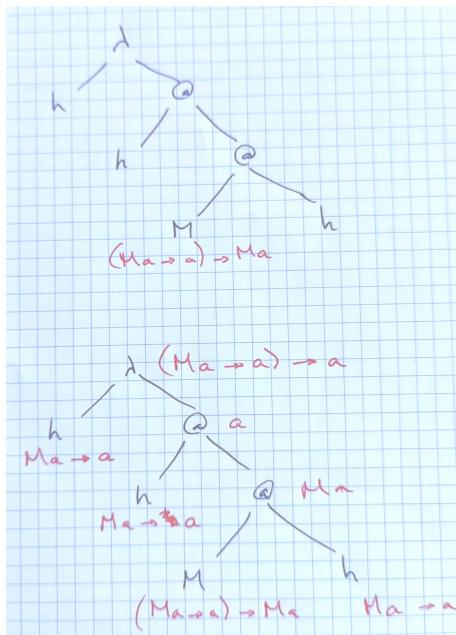
1 Inferència de tipus i el combinador Y

1. L'arbre de sintàxi anotat amb tipus és el següent:

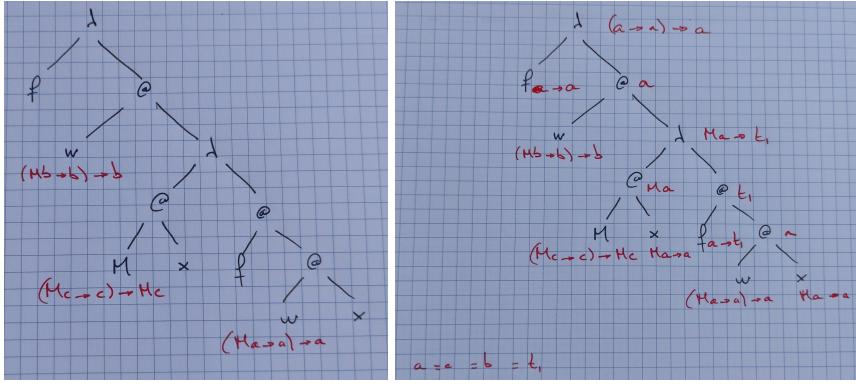


Només escriure la restricció del node de més a la dreta ens trobem un problema: La restricció és $a = a \rightarrow t_1$ que correspon a crear un tipus infinit.

2. L'arbre de sintàxi és a la part superior de la figura. Només cal etiquetar el tipus de M correctament i anar pujant per l'arbre per trobar que $w :: (M a \rightarrow a) \rightarrow a$ i tots els tipus de les subexpressions (part inferior de la figura).



3. L'arbre de sintàxi és a la part esquerra de la figura, amb les anotacions dels tipus coneguts. Després es poden anar inferint tota la resta de tipus de dalt a baix (part dreta de la figura) fins trobar que $y :: (a \rightarrow a) \rightarrow a$.



4. Tenim:

$$\begin{aligned}
 y \ r &= \\
 &= w (\lambda(M x) \rightarrow r(w x)) && \text{-- def de } y \\
 &= (\lambda(M x) \rightarrow r(w x)) (M (\lambda(M x) \rightarrow r(w x))) && \text{-- def de } w \\
 &= (\lambda(M x) \rightarrow r(w x)) (M (\lambda(M z) \rightarrow r(w z))) && \text{-- renombrament} \\
 &= r (w (\lambda(M z) \rightarrow r(w z))) && \text{-- aplicació} \\
 &= r (w (\lambda(M x) \rightarrow r(w x))) && \text{-- renombrament} \\
 &= r (y \ r) && \text{-- primera equació}
 \end{aligned}$$

I per tant hem implementant un punt fixe en Haskell sense errors de tipus.

Errors freqüents

- Interpretar $z(x \ x)$ como $(z \ x)x$.
- Interpretar que $x \ x$ es una única variable y no x aplicado a x .
- No saber reconocer el error de tipado.
- No saber definir el tipo de M .
- (1.4) Pensar que lo que se pedía era ver que el tipo de $y \ r$ es el mismo que el tipo de $r(y \ r)$.
- (1.4) Pensar que se pedía ver $YF = F(YF)$.

2 Llistes i mònades

$$\begin{aligned}
 1. \ xs \gg= f &= \\
 &= \text{concat} (\text{map } f \ xs) && \text{-- definició de } \gg= \\
 &= \text{concat} (\text{map } (\lambda x \rightarrow [x, 2*x]) [1, 2, 3]) && \text{-- definició de } f \text{ i } xs \\
 &= \text{concat} ([[1, 2], [2, 4], [3, 6]]) && \text{-- aplicació de map} \\
 &= [1, 2, 2, 4, 3, 6] && \text{-- aplicació de concat}
 \end{aligned}$$

2. La llista per comprensió

$$[(x,y) \mid x \leftarrow "abc", y \leftarrow [1, 2, 3]]$$

és desensucra a notació **do**:

```

do
  x ← "abc"
  y ← [1, 2, 3]
  return (x,y)
  
```

i aquesta es desensucra a notació funcional:

```
"abc" >>= \x →
[1, 2, 3] >>= \y →
return (x,y)
```

3. La llista per comprensió

$[(x,y,z) \mid x \leftarrow [1..n], y \leftarrow [x..n], z \leftarrow [y..n], x*x + y*y == z*z]$

és desensucra a notació **do**:

```
do
  x ← [1..n]
  y ← [x..n]
  z ← [y..n]
  guard (x*x + y*y == z*z)
  return (x,y,z)
```

i aquesta es desensucra a notació funcional:

```
[1.. n] >>= \x →
[x .. n] >>= \y →
[y .. n] >>= \z →
guard (x*x + y*y == z*z) >>
return (x,y,z)
```

4. Es pot fer així:

```
guard True = return ()
guard False = []
```

5. Per desensucrar els **lets** dins de llistes per comprensió, només cal posar-los dins la notació **do**, que ja té **lets**. En el cas de l'exemple quedaría així:

```
do
  x ← [1..n]
  y ← [x..n]
  let z = isqrt (x*x + y*y)
  guard (z ≤ n)
  guard (x*x + y*y == z*z)
  return (x,y,z)
```

Errors freqüents

- Fer que **guard** no segueixi la seva signatura.
- Confondre **guard** i **filter**.
- No desensucrar *totalment*.
- Intentar avaluar expressions (menys la primera).
- Nota: No s'ha penalitzat usar $\gg=$ enlloc de \gg després del **guard**, ja que \gg no apareixia a l'enunciat. En canvi, usar \rightarrow no té sentit.

3 Raonament equacional

- Cas base: la llista buida. Per una banda tenim:

$$\begin{aligned} (\ell . c) [] &= \\ &= \ell(c []) &-- (e) \\ &= \ell [] &-- (c_1) \\ &= 0 &-- (b_1) \end{aligned}$$

Per altra banda tenim:

$$\begin{aligned}
 (s . (m \ell)) [] &= \\
 &= s (m \ell []) \quad \text{-- (e)} \\
 &= s [] \quad \text{-- (d}_1\text{)} \\
 &= 0 \quad \text{-- (a}_1\text{)}
 \end{aligned}$$

Per tant, $(\ell . c) [] = (s . (m \ell)) []$.

- Cas inductiu: agafem

$$(\ell . c) xs = (s . (m \ell)) xs$$

com a hipòtesi d'inducció. Volem veure que $(\ell . c) (x:xs) = (s . (m \ell)) (x:xs)$:

Per una banda tenim:

$$\begin{aligned}
 (\ell . c) (x:xs) &= \\
 &= \ell (c (x:xs)) \quad \text{-- (e)} \\
 &= \ell (x ++ c xs) \quad \text{-- (c}_2\text{)} \\
 &= \ell x + \ell (c xs) \quad \text{-- (L)} \\
 &= \ell x + (\ell . c) xs \quad \text{-- (e)} \\
 &= \ell x + (s . (m \ell)) xs \quad \text{-- (HI)}
 \end{aligned}$$

Per altra banda tenim:

$$\begin{aligned}
 (s . (m \ell)) (x:xs) &= \\
 &= s (m \ell (x:xs)) \quad \text{-- (e)} \\
 &= s (\ell x : m \ell xs) \quad \text{-- (d}_2\text{)} \\
 &= \ell x + s (m \ell xs) \quad \text{-- (a}_2\text{)} \\
 &= \ell x + (s . (m \ell)) xs \quad \text{-- (e)}
 \end{aligned}$$

Per tant, $(\ell . c) (x:xs) = (s . (m \ell)) (x:xs)$.

Conseqüentment, $\ell . c = s . (m \ell)$.

Errors freqüents

- No estructurar la demostració.
- No anunciar la hipòtesi d'inducció.
- No usar la hipòtesi d'inducció.
- Usar `[]` com a cas base.
- No anotar cada pas amb l'etiqueta de l'equació utilitzada.
- Utilitzar símbols direccionals per les equacions (\rightarrow , \Rightarrow i semblants). Són igualtats.
- No seguir la sintàxi de Haskell.
- Usar el·lipsi (punts suspensius) en una demostració per inducció.

4 Subtipos

1. De acuerdo con la definición `punto mover2 (punto p, double x1, double y1)`, la llamada `mover2(*p, 2.0, 1.0)` es correcta, ya que `*p` tiene tipo estático `punto` y tipo dinámico `puntocolor`. También por su definición sabemos que el resultado de esa llamada tiene tipo `punto`. Por tanto, `p1` tiene tipo `punto`. Como consecuencia, la llamada `p1.reset()` será una llamada a la operación `reset` de la clase `punto`.
2. Sería incorrecto, tanto en C++ como en Java, ya que el resultado de `mover2` tiene tipo `punto` y no se puede asignar una expresión de tipo `punto` a una variable de tipo `puntocolor`.

3. Sería incorrecto, tanto en C++ como en Java, ya que la llamada `p.cambiar_color(0)` sería incorrecta, porque `p` es de tipo punto y `cambiar_color` está definida en la clase `puntocolor`.
4. Es correcto, tanto en C++ como en Java. En el caso de C++ la llamada `p.reset()` sería una llamada a la operación de la clase punto, ya que `p` tiene ese tipo estático y es un objeto estático. En cambio, en el caso de Java, la llamada sería a la operación de la clase `puntocolor`, ya que `p` tiene tipo dinámico `puntocolor`.

Errors freqüents

- Pensar que `mover2(c,-,-)` es de tipo `puntocolor`, cuando `c` es de tipo `puntocolor`.
- Poner que en C++ o en Java no se permite `p = c;` cuando `p` es de tipo punto y `c` es de tipo `puntocolor`.
- Poner que en C++ o en Java se permite `c = p;` cuando `p` es de tipo punto y `c` es de tipo `puntocolor`.