Programació funcional

Albert Rubio

Especialitat de Computació
Grau en Enginyeria Informàtica

FIB

L'algorisme de Milner

1. S'assigna un tipus a l'expressió i a cada subexpressió.
   - Si el tipus és conegut se li assigna aquest tipus.
   - Sinó se li assigna una variable de tipus.

   Recordeu que les funcions són expressions.
2. Es genera un conjunt de restriccions (d'igualtat principalment) a partir l'arbre de l'expressió.
   - Aplicació.
   - Abstracció.
   - Let
   - ...
3. Es resolen les restriccions usant unificació.

L'algorisme de Milner

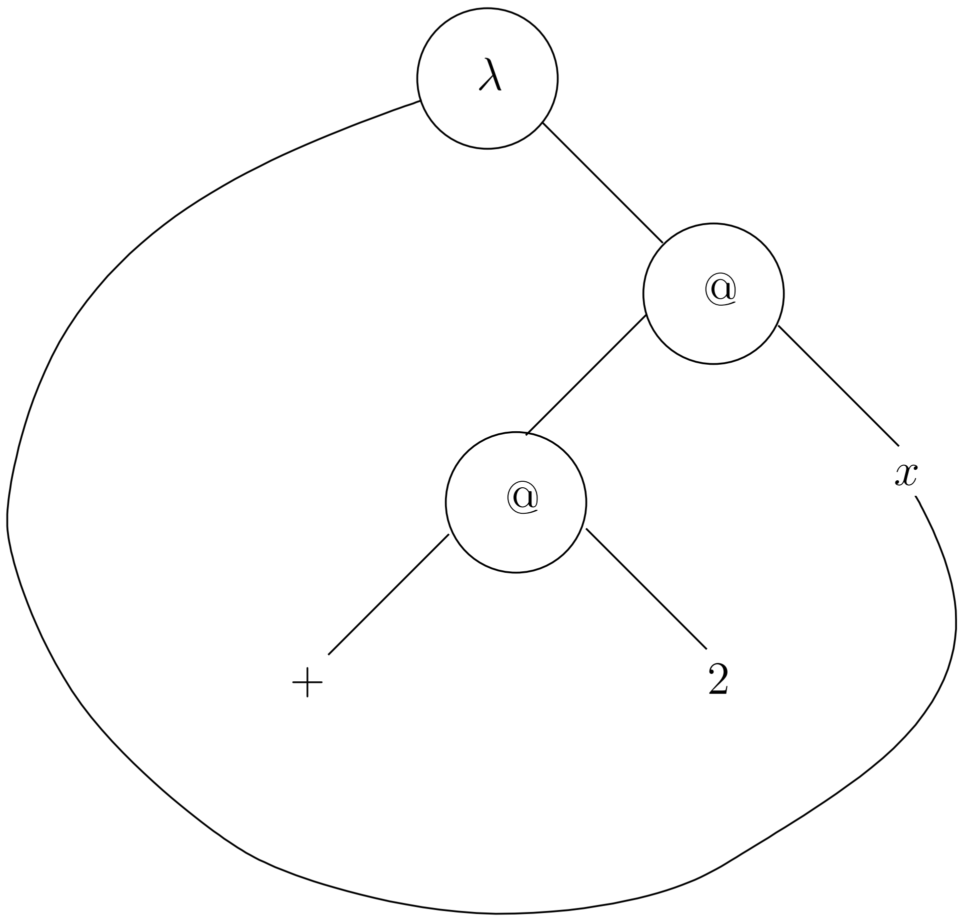
- Considereu l'expressió: $(+\ 2\ x)$

L'algorisme de Milner

- Lliguem les variables lliures amb lambdas: $\lambda x.(+\ 2\ x)$

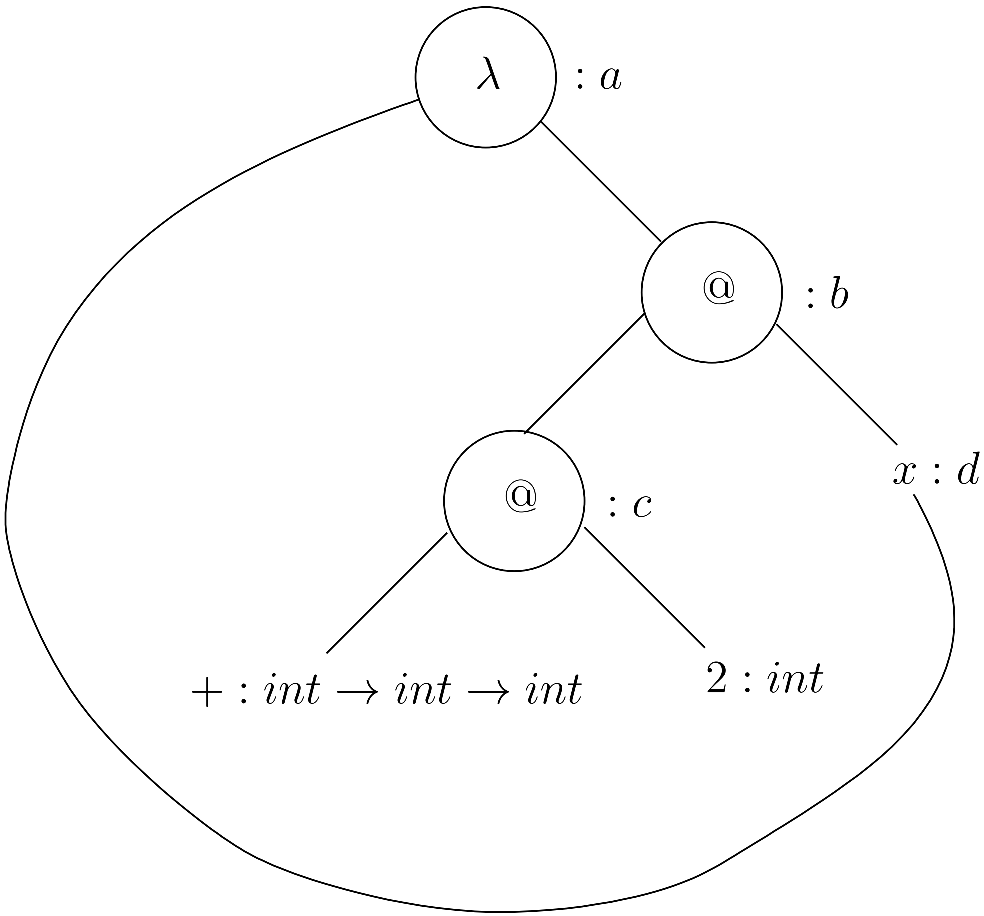
L'algorisme de Milner

- Lliguem les variables lliures amb lambdas: $\lambda x.(+\ 2\ x)$



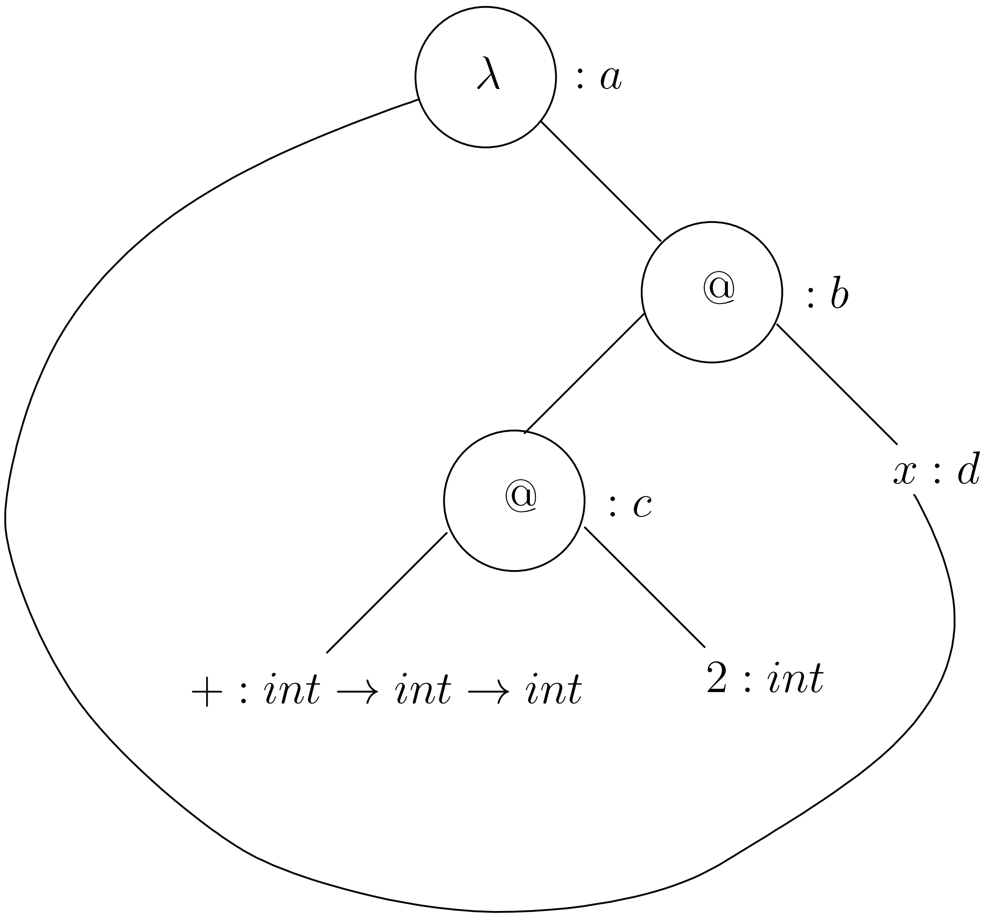
L'algorisme de Milner

1. Assignem tipus a totes les expressions



L'algorisme de Milner

2. Generem les restriccions/equacions

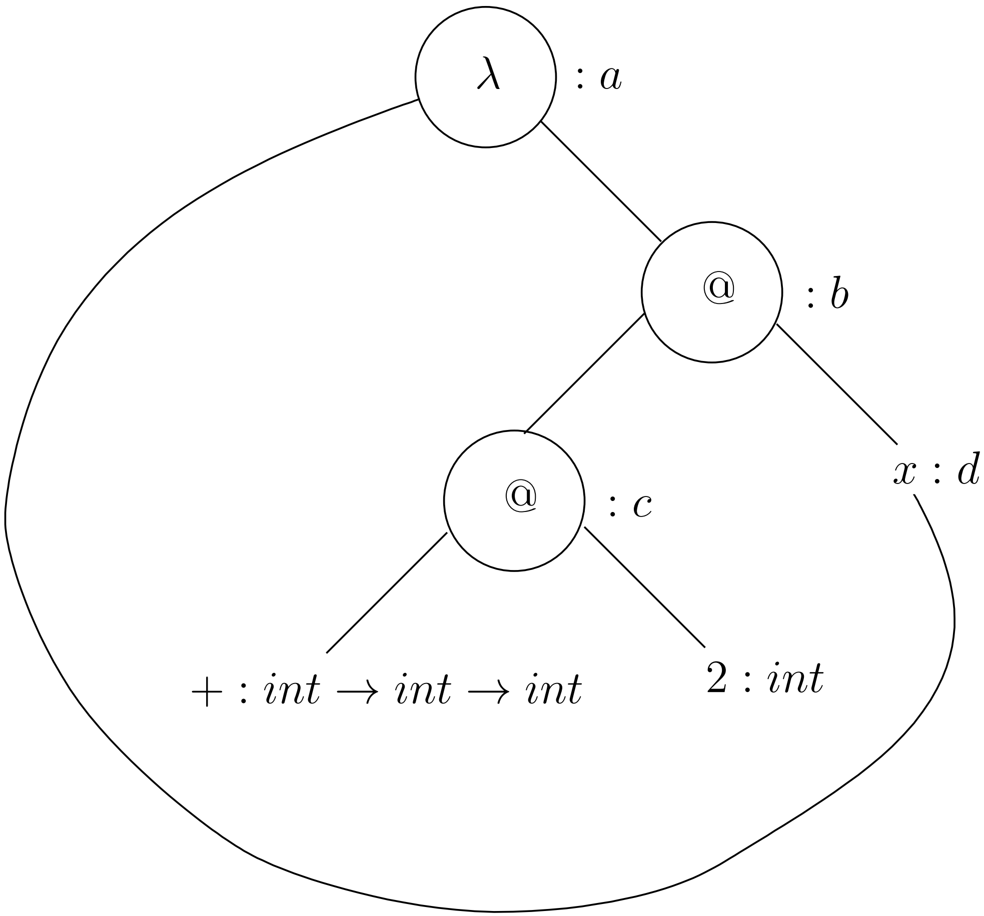


Equacions:

$$
\begin{aligned}
a &= d \rightarrow b \\
c &= d \rightarrow b \\
int \rightarrow int \rightarrow int &= int \rightarrow c
\end{aligned}
$$

L'algorisme de Milner

3. Solucionem les equacions amb unificació



Equacions:

$$a = d \to b$$
$$c = d \to b$$
$$int \to int \to int = int \to c$$

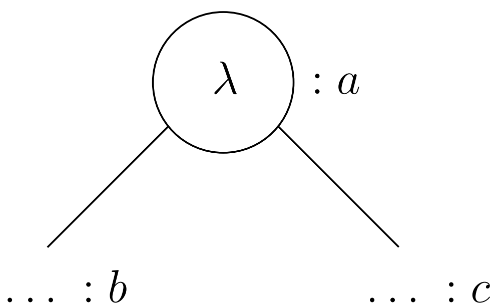
Solució:

$$a = int \to int$$
$$b = int$$
$$c = int \to int$$
$$d = int$$

L'algorisme de Milner

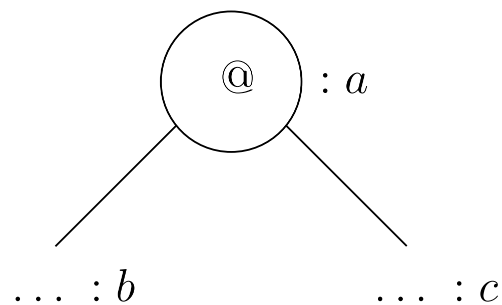
Aquestes són les regles per generar les equacions:

Abstracció:



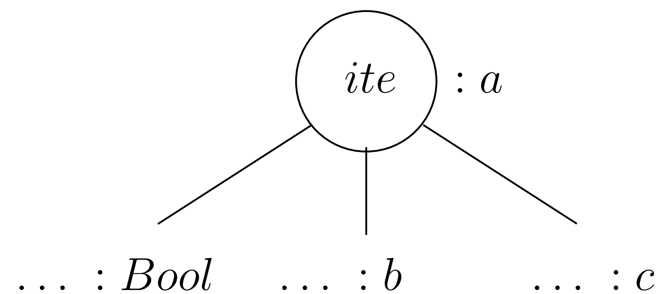
$$a = b \to c$$

Aplicació:



$$b = c \to a$$

IfThenElse:



$$a = c$$
$$a = b$$

L'algorisme de Milner

Considerem ara:

$$map\ f\ l\ \ =\ \ if\ (null\ l)\ then\ [\,]\ else\ f\ (head\ l) : map\ f\ (tail\ l)$$

Podem entendre una definició, com una funció que aplicada als paràmetres ens torna la part dreta de la definició.

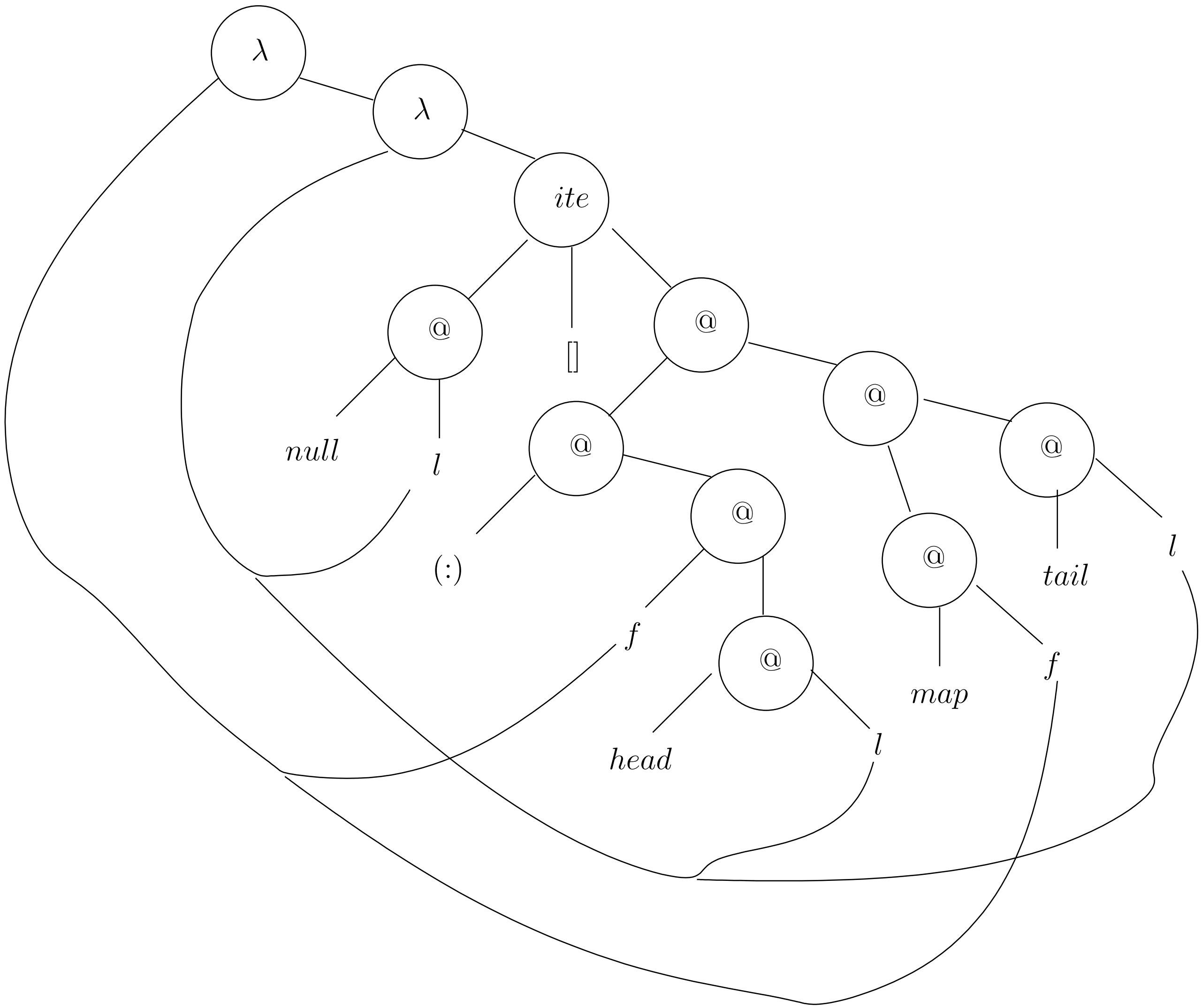
$$\lambda\ f.\ \lambda\ l.\ if\ (null\ l)\ then\ [\,]\ else\ f\ (head\ l) : map\ f\ (tail\ l)$$

Noteu que el tipus de if_then_else és:

$$if\_then\_else : Bool\ \rightarrow\ a\ \rightarrow\ a\ \rightarrow\ a$$

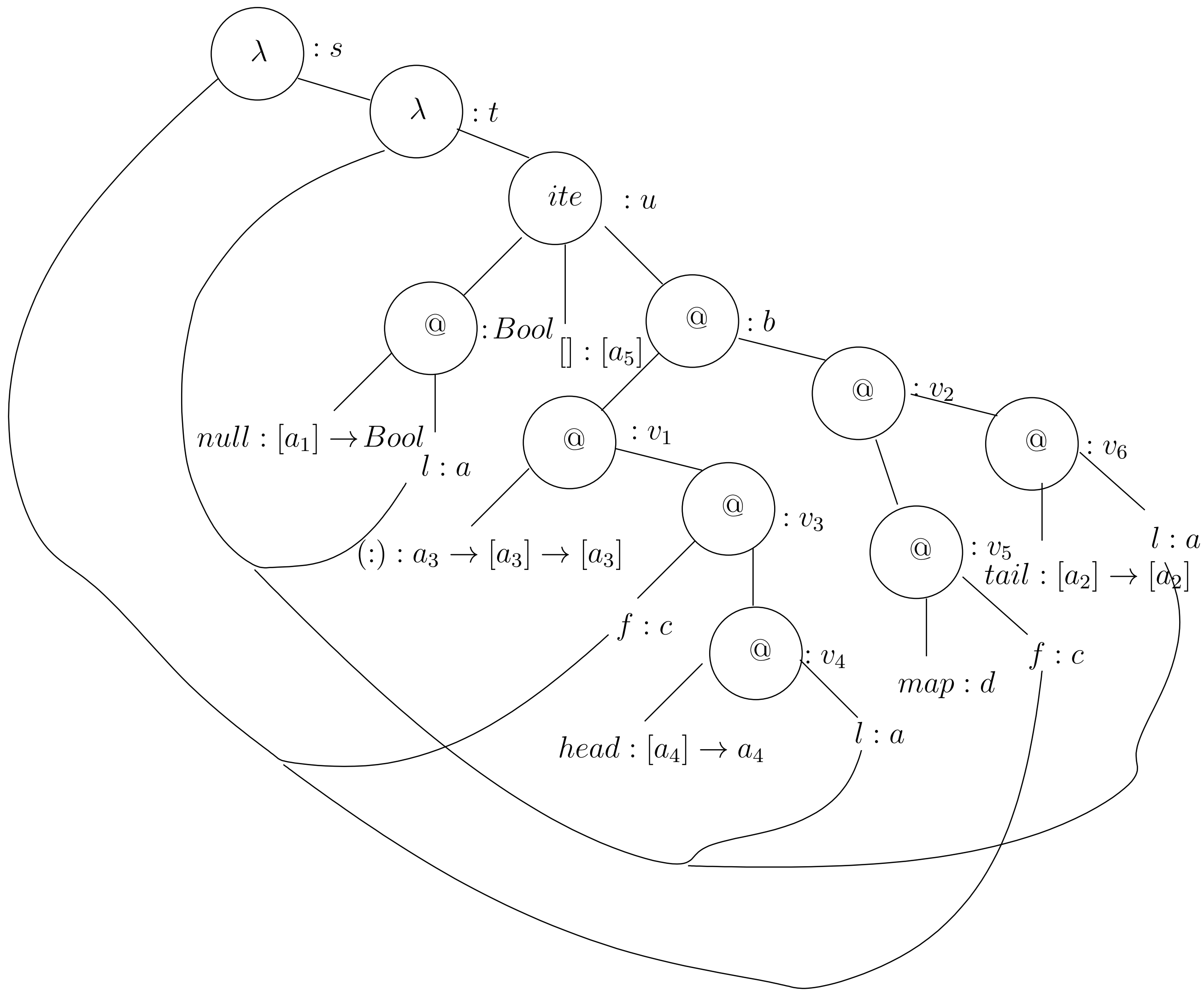
L’algorisme de Milner

$$\lambda\, f.\ \lambda\, l.\ if\ (null\ l)\ then\ [\,]\ else\ f\ (head\ l) : map\ f\ (tail\ l)$$



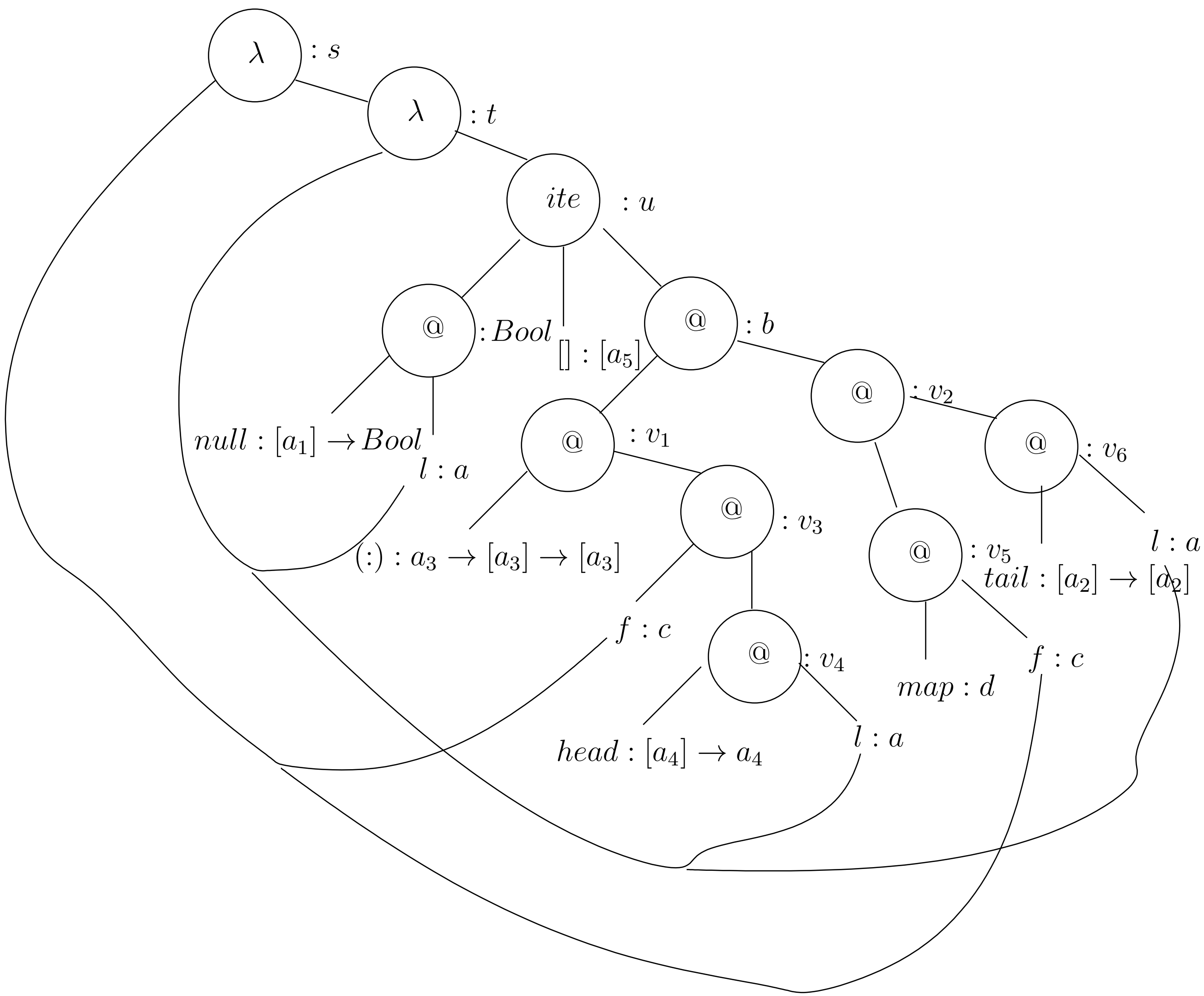
L'algorisme de Milner

$$\lambda\, f.\, \lambda\, l.\; if\ (null\ l)\ then\ [\,]\ else\ f\ (head\ l) : map\ f\ (tail\ l)$$



L'algorisme de Milner

$$\lambda\, f.\, \lambda\, l.\ if\ (null\ l)\ then\ [\,]\ else\ f\ (head\ l) : map\ f\ (tail\ l)$$

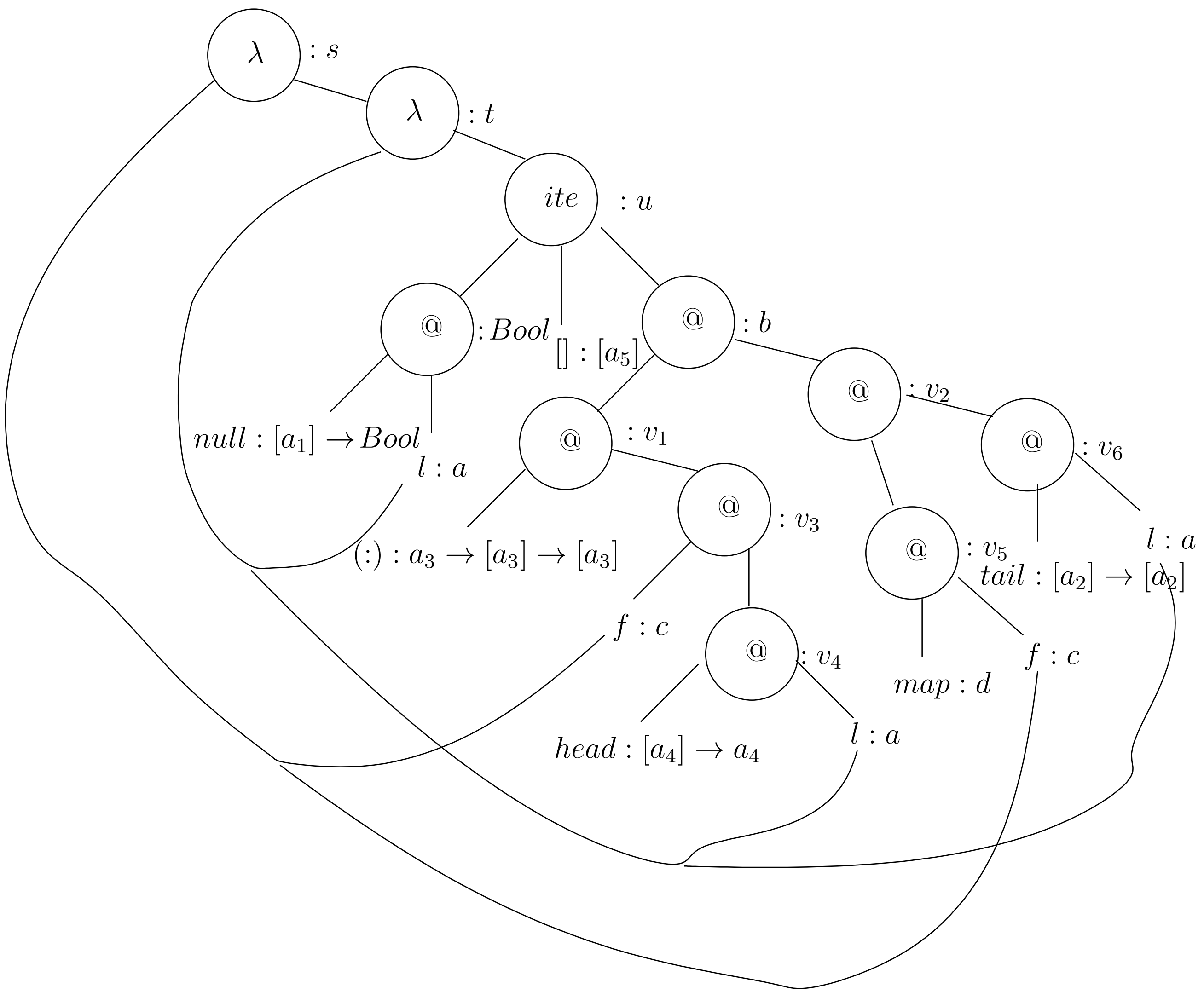


Equacions:

$$
\begin{aligned}
s &= c \to t \\
t &= a \to u \\
u &= [a_5] \\
u &= b \\
[a_1] \to Bool &= a \to Bool \\
v_1 &= v_2 \to b \\
a_3 \to [a_3] \to [a_3] &= v_3 \to v_1 \\
c &= v_4 \to v_3 \\
[a_4] \to a_4 &= a \to v_4 \\
v_5 &= v_6 \to v_2 \\
d &= c \to v_5 \\
[a_2] \to [a_2] &= a \to v_6 \\
\\
s &= d
\end{aligned}
$$

L'algorisme de Milner

$$\lambda\, f.\, \lambda\, l.\; if\ (null\ l)\ then\ [\,]\ else\ f\ (head\ l) : map\ f\ (tail\ l)$$



Solució:

$$\begin{aligned}
a &= [a_1] \\
a_2 &= a_1 \\
a_4 &= a_1 \\
a_5 &= a_3 \\
b &= [a_3] \\
c &= a_1 \to a_3 \\
d &= (a_1 \to a_3) \to [a_1] \to [a_3] \\
s &= (a_1 \to a_3) \to [a_1] \to [a_3] \\
t &= [a_1] \to [a_3] \\
u &= [a_3] \\
v_1 &= [a_3] \to [a_3] \\
v_2 &= [a_3] \\
v_3 &= a_3 \\
v_4 &= a_1 \\
v_5 &= [a_1] \to [a_3] \\
v_6 &= [a_1]
\end{aligned}$$

L'algorisme de Milner

Considerem ara:
$map\ f\ (x : xs) = (f\ x) : (\text{map}\ f\ xs)$

És a dir, una definició amb *pattern matching*

En aquest cas la introducció de lambdas és una mica diferent, ja que tractem els patrons com si fossin vairables lliures:

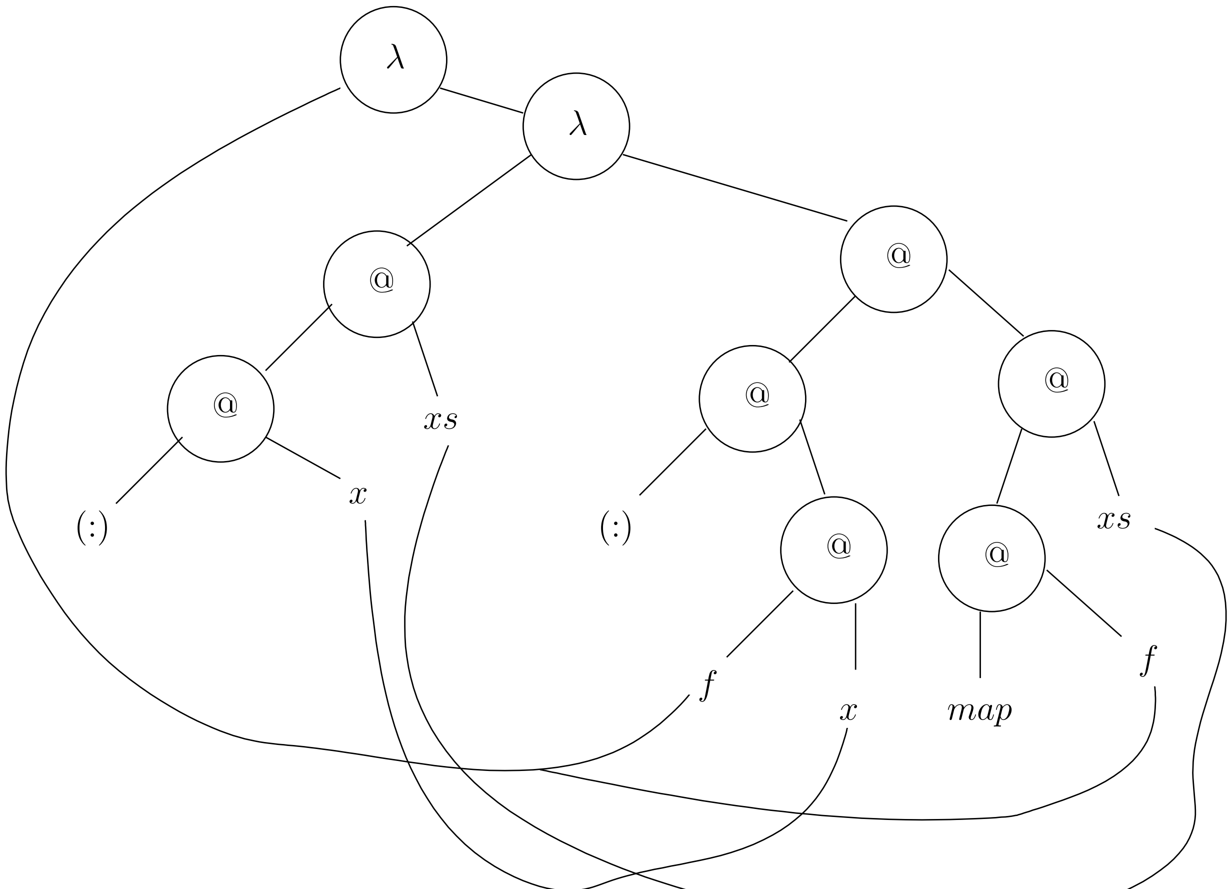
$\lambda\ f.\ \lambda\ (x : xs).\ (f\ x) : (\text{map}\ f\ xs)$

Noteu que ara hem de considerar que el primer argument de lambda pot ser una expressió, que tractarem igual que les demés.

Totes les variables del *pattern* queden lligades per la lambda.

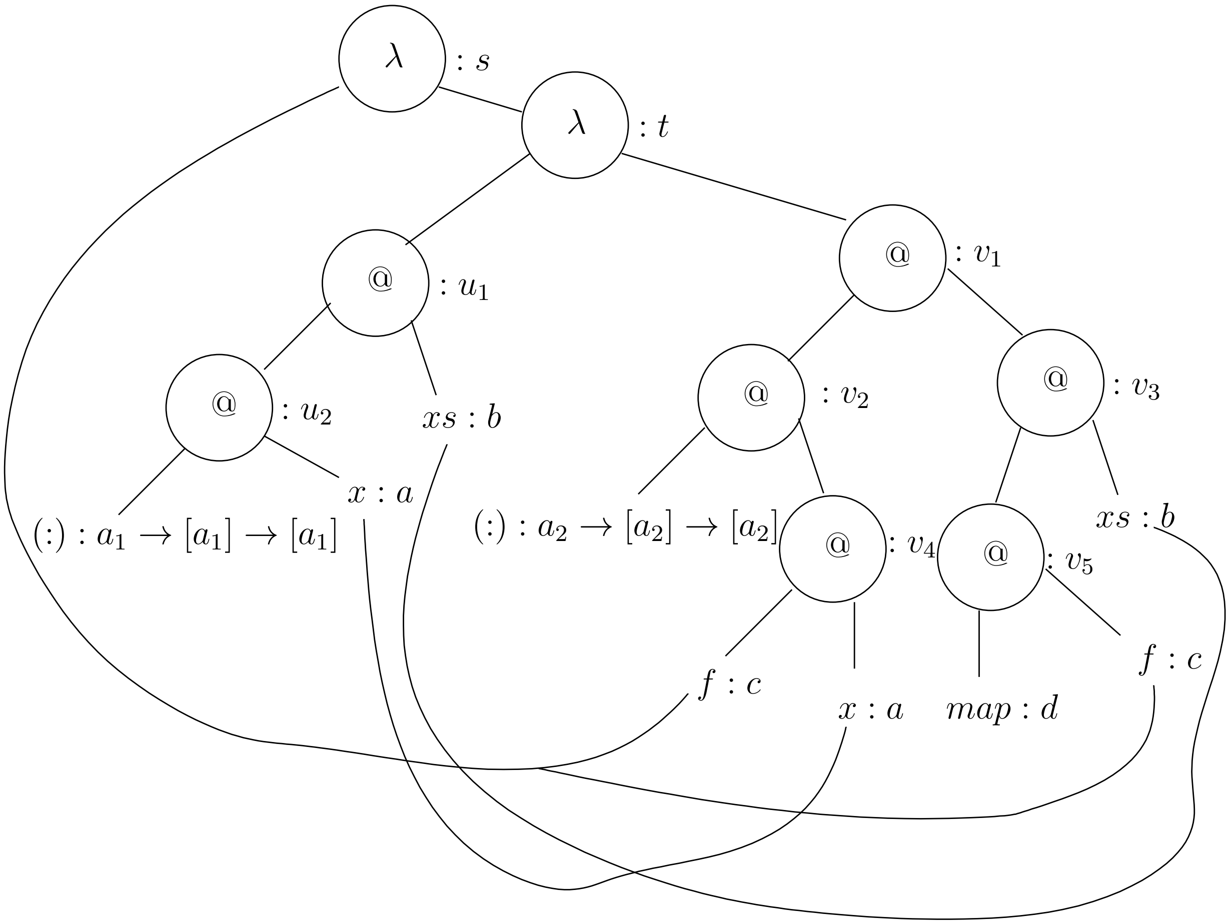
L'algorisme de Milner

$\lambda\ f.\ \lambda\ (x : xs).\ (f\ x) : (\mathsf{map}\ f\ xs)$



L'algorisme de Milner

$\lambda\ f.\ \lambda\ (x : xs).\ (f\ x) : (\mathsf{map}\ f\ xs)$

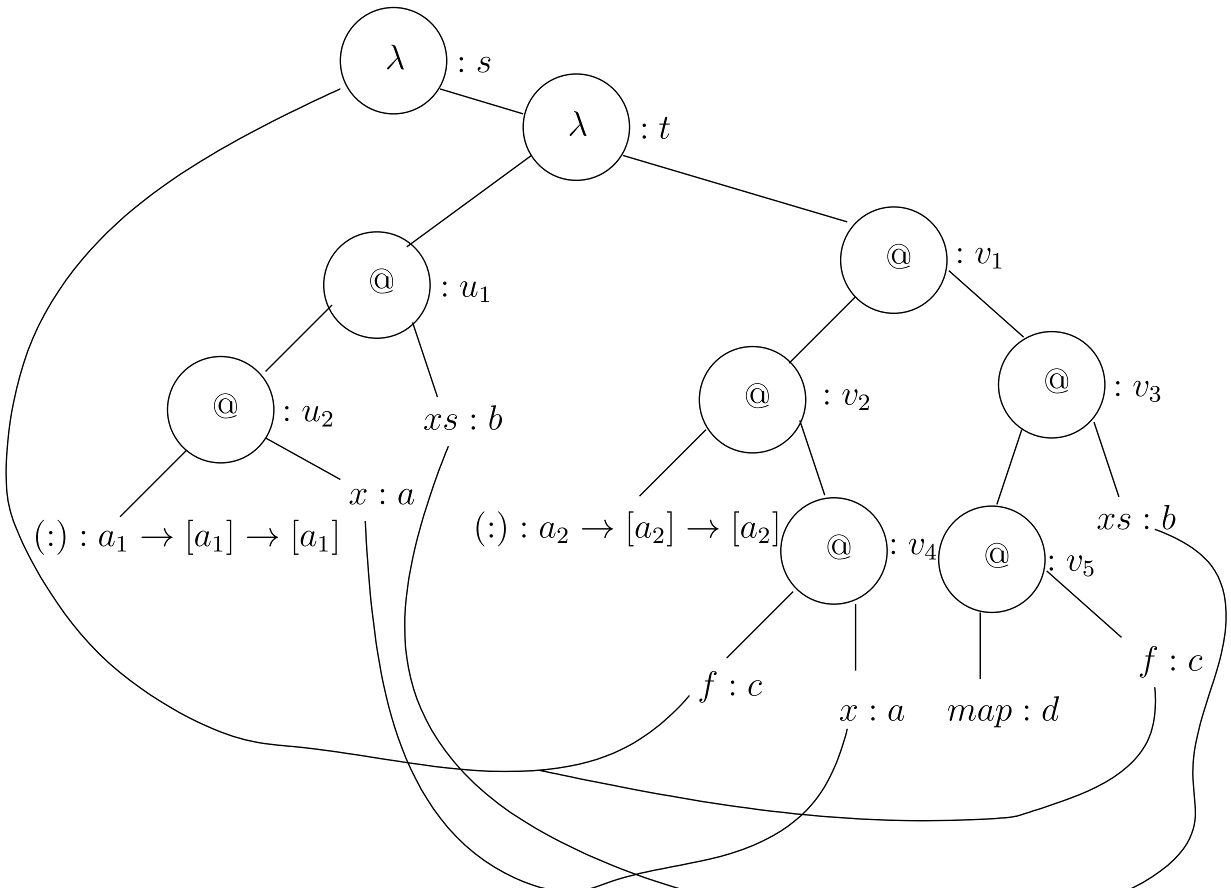


Equacions:

$$
\begin{aligned}
s &= c \to t \\
t &= u_1 \to v_1 \\
u_2 &= b \to u_1 \\
a_1 \to [a_1] \to [a_1] &= a \to u_2 \\
v_2 &= v_3 \to v_1 \\
a_2 \to [a_2] \to [a_2] &= v_4 \to v_2 \\
c &= a \to v_4 \\
v_5 &= b \to v_3 \\
d &= c \to v_5 \\
\\
s &= d
\end{aligned}
$$

L’algorisme de Milner

$\lambda\ f.\ \lambda\ (x : xs).\ (f\ x) : (\text{map}\ f\ xs)$



Solució:

$$
\begin{aligned}
a_1 &= a \\
b &= [a] \\
c &= a \to a_2 \\
d &= (a \to a_2) \to [a] \to [a_2] \\
s &= (a \to a_2) \to [a] \to [a_2] \\
t &= [a] \to [a_2] \\
u_1 &= [a] \\
u_2 &= [a] \to [A] \\
v_1 &= [a_2] \\
v_2 &= [a_2] \to [a_2] \\
v_3 &= [a_2] \\
v_4 &= a_2 \\
v_5 &= [a] \to [a_2]
\end{aligned}
$$

L'algorisme de Milner

Obviament podem tractar abstraccions del Haskell

```
\x -> E
```

tal com ho hem fet amb $\lambda x.\ E$

Les construccions *let* (o *where*) es poden expressar per a la inferencia de tipus amb abstraccions i aplicacions

Per exemple

```
let x=L in E
```

es tracta com $(\lambda x.E\ L)$

Les *guardes* es tracten com un *if*

El *case* es tracta com una definició per pattern matching

....

L'algorisme de Milner (Classes)

Considerem ara que tenim classes de tipus.

És a dir, que tenim definicions com ara

$$(+) :: Num\ a \Rightarrow a \rightarrow a \rightarrow a$$
$$(>) :: Ord\ \ a \Rightarrow a \rightarrow a \rightarrow Bool$$

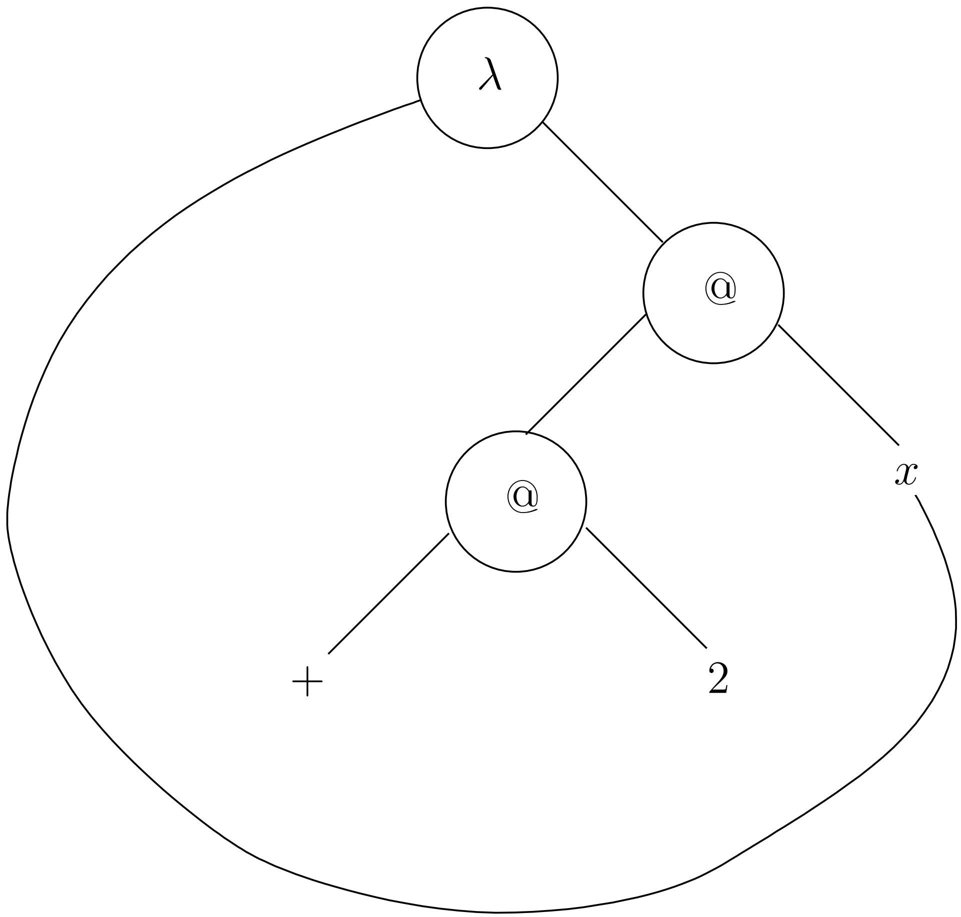
Això introdueix un nou tipus de restricció *de context*.

Per tant les solucions també

- han de satisfer les condicions de classe.
- contindran condicions de classe.

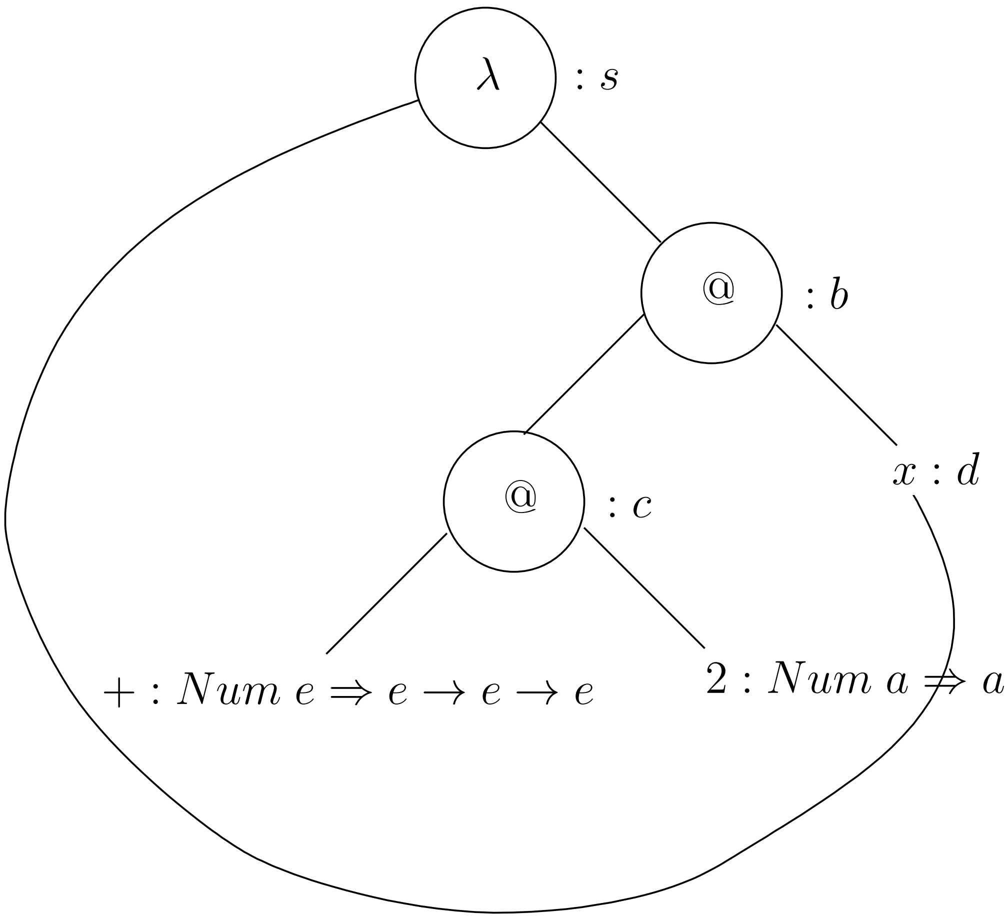
L'algorisme de Milner (Classes)

Reconsiderem l'exemple inicial: $\lambda x.(+\ 2\ x)$



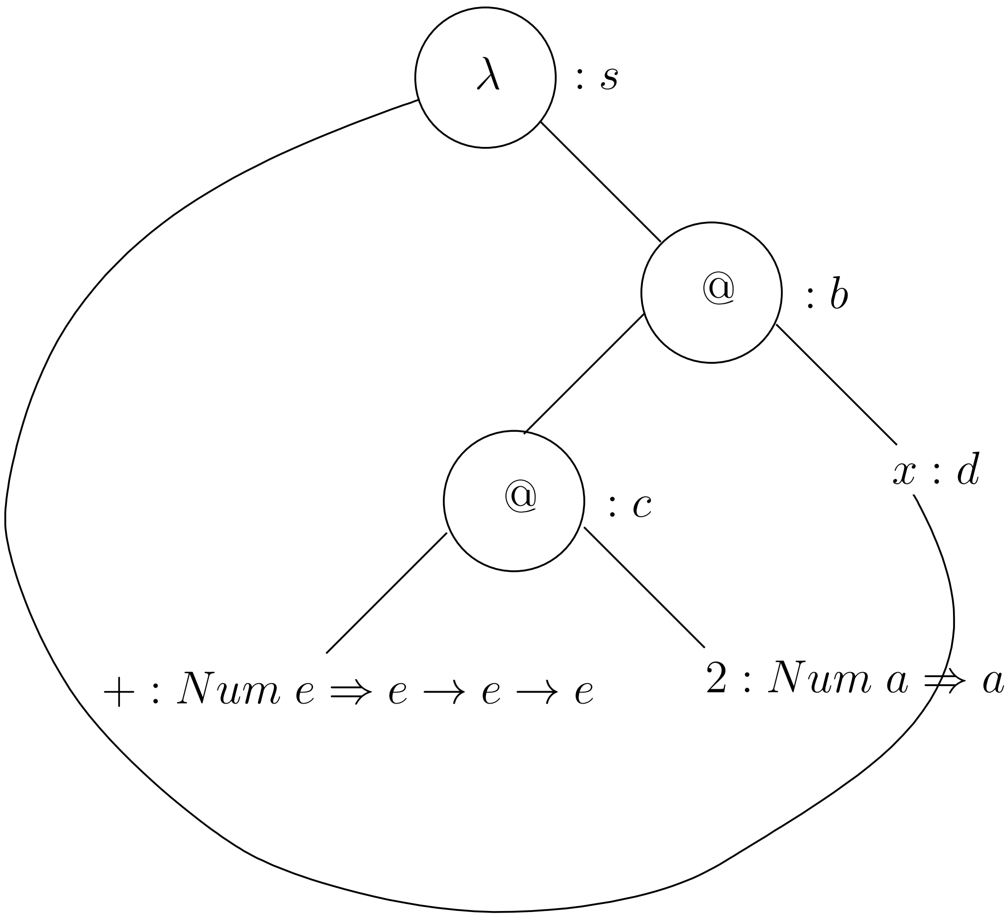
L'algorisme de Milner (Classes)

Reconsiderem l'exemple inicial: $\lambda x.(+\ 2\ x)$



L'algorisme de Milner (Classes)

Reconsiderem l'exemple inicial: $\lambda x.(+\ 2\ x)$

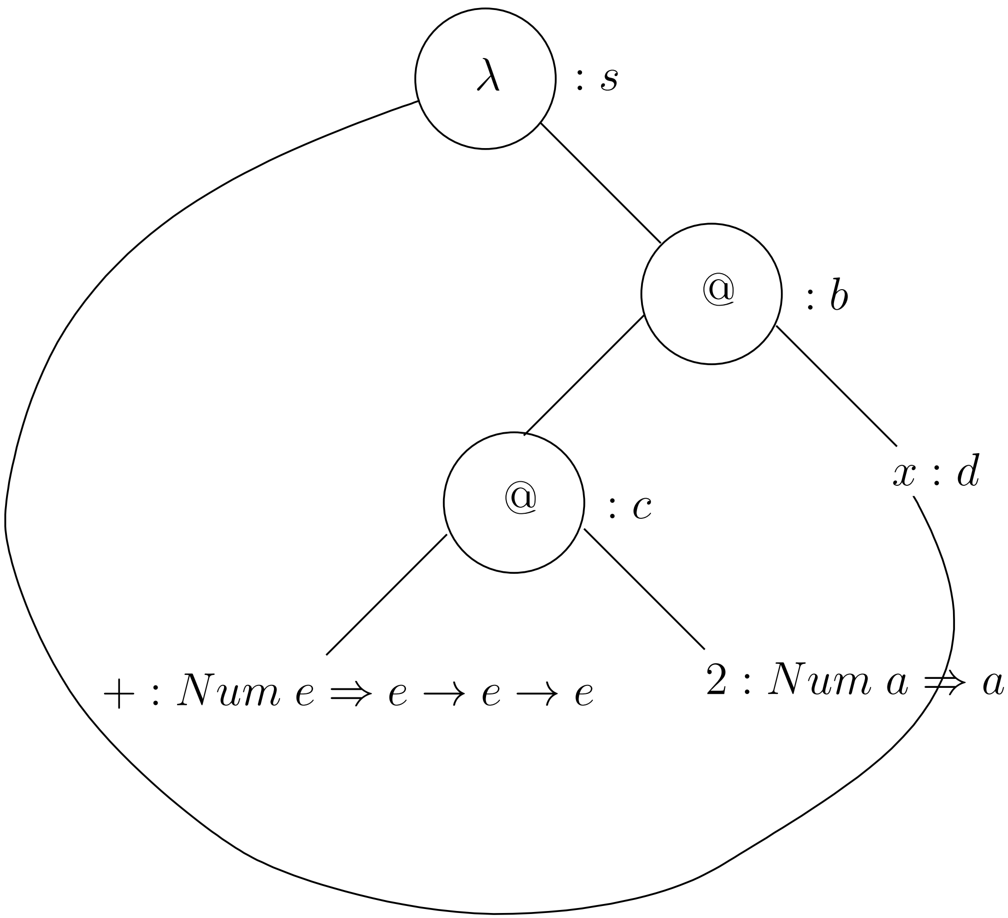


Equacions:

$$s = d \to b$$
$$c = d \to b$$
$$e \to e \to e = a \to c$$
$$Num(e),\ Num(a)$$

L'algorisme de Milner (Classes)

Reconsiderem l'exemple inicial: $\lambda x.(+\ 2\ x)$



Equacions:

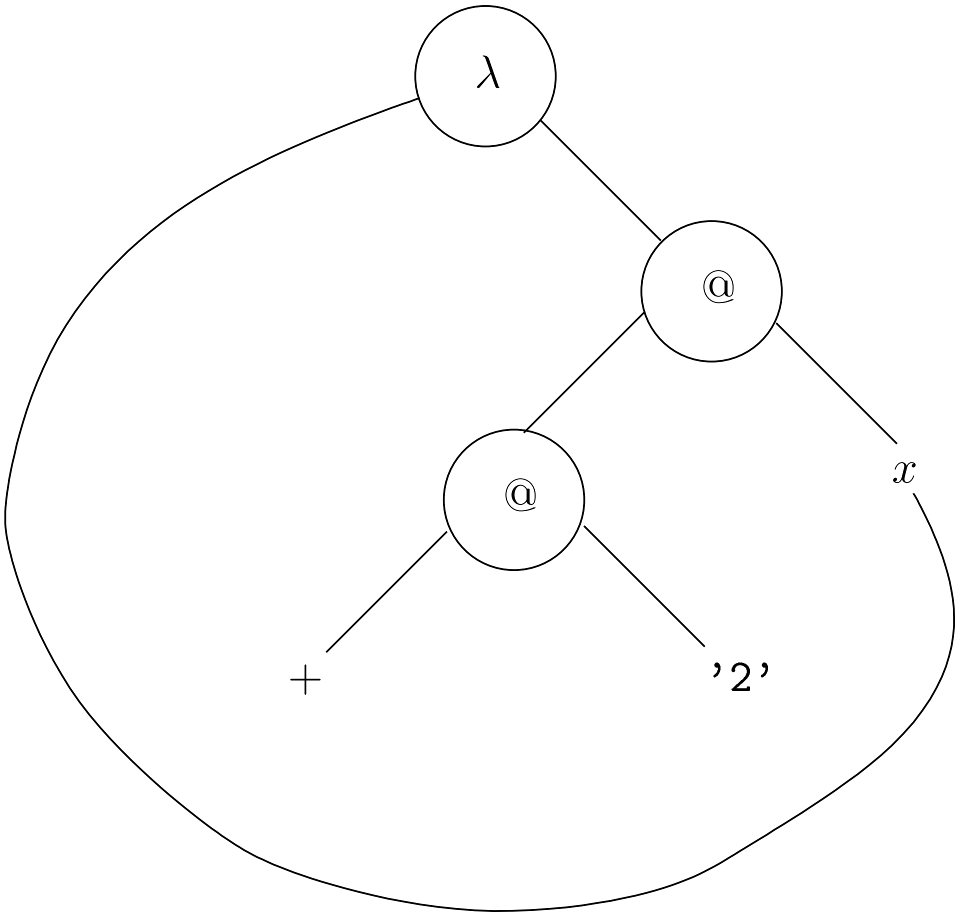
$$s = d \to b$$
$$c = d \to b$$
$$e \to e \to e = a \to c$$
$$Num(e),\ Num(a)$$

Solució:

$$s = a \to a$$
$$b = a$$
$$c = a \to a$$
$$d = a$$
$$e = a$$
$$Num\ a$$

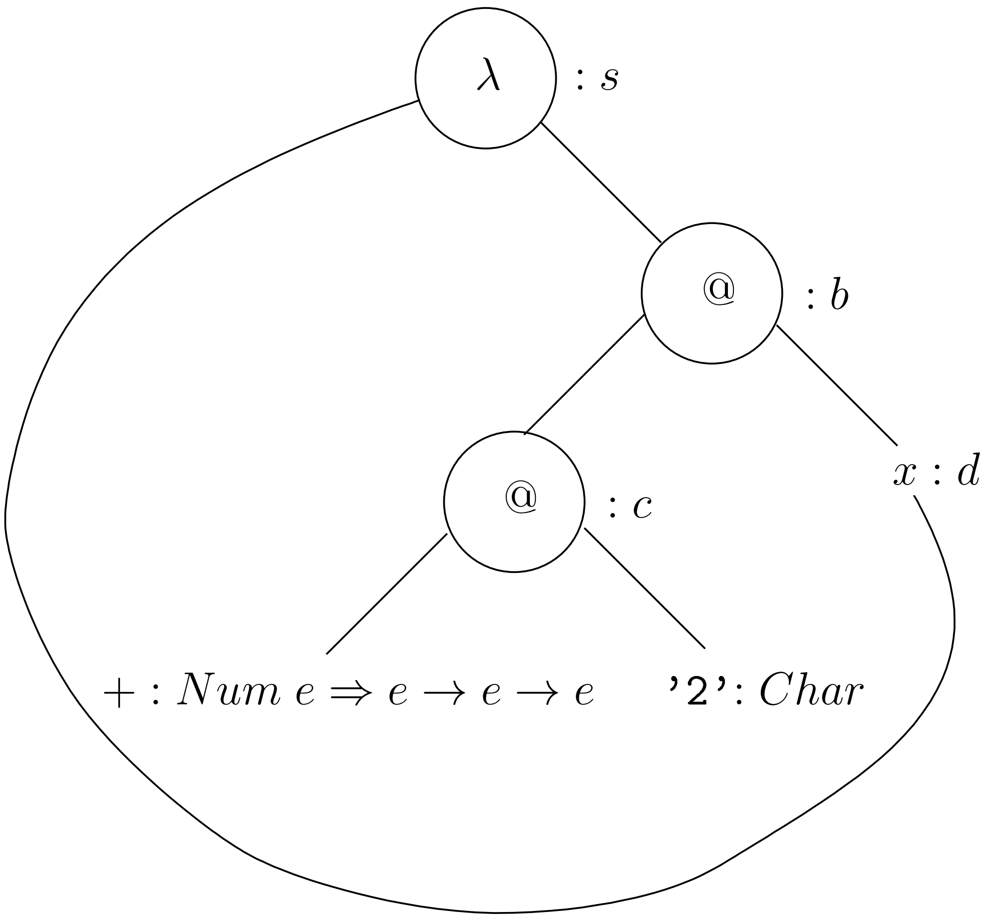
L'algorisme de Milner (Classes)

Suposeu ara: $\lambda x.(+\ \texttt{'2'}\ x)$



L'algorisme de Milner (Classes)

Suposeu ara: $\lambda x.(+\ \text{'2'}\ x)$



Equacions:

$$s = d \to b$$
$$c = d \to b$$
$$e \to e \to e = a \to c$$
$$Num(e)$$

Solució:

$$s = Char \to Char$$
$$b = Char$$
$$c = Char \to Char$$
$$d = Char$$
$$e = Char$$
$$Num\ Char$$

ERROR!!!

L'algorisme de Milner (Exercicis)

1. `even x = if (rem x 2) == 0 then True else False`
   amb `rem :: int -> int -> int`

2. `last [x] = x`
   Recordeu que `[x]` és `x:[]`

3. `foldr f z (x : xs) = f x (foldr f z xs)`

4. ```
   delete x (y:ys) = if x==y
                     then ys
                     else y:(delete x ys)
   ```
   amb `(==) :: Eq a => a -> a -> Bool`
   ```