Sistemes de Tipus

Albert Rubio

Llenguatges de Programació, FIB, UPC

Primavera 2016

Sistemes de tipus

Continguts

Introducció

Usos principals dels tipus en els llenguatges de programació:

- Donar nom i organitzar conceptes.
- Assegurar-se que els bits de memòria s'interpreten consistentment.
- Donar informació al compilador sobre les dades manipulades pel programa.

Introducció

És important tenir en compte que:

- No hi ha llenguatges de programació sense tipus
- Sempre hi ha alguna forma de tipus.
  No necessàriament explícits.

El fet d'exitir tipus implica l'aparició dels "errors de tipus"

Continguts

Type safety

Un llenguatge és "type safe" si cap programa pot donar errors de tipus en temps d'execució.

Alguns errors típics en execució:

- Type Cast: usar un valor d'un tipus en un altre tipus.
  Per exemple, en C un enter pot ser usat com a funció (com a adreça), però pot saltar on no hi ha una funció (generant un error).

- Aritmètica de punters.
  Per exemple, a C si tenim
  `A* p;`
  llavors `*(p+i)` té tipus `A`, però el que hi ha a `p+i` pot ser qualsevol cosa i pot causar un error de tipus.

Type safety

- Alliberament explícit de memòria (deallocate/delete).
  Per exemple, a Pascal usar un apuntador alliberat pot donar errors de tipus.
- En llenguatges OO (antics). No existència d'un mètode (degut a l'herència).

Exemples de llenguatges:

- "Type safe": Haskell, Java, ...
- "Type unsafe": C, C++, ...

Tipat fort/feble

Els llenguatges amb tipat fort són els que imposen restriccions que eviten barrejar valors de diferents tipus (per exemple, amb conversions implícites).

Per exemple, amb un tipat feble podem tenir:

```
a = 2
b = "2"

a + b       # JavaScript retorna "22"
a + b       # Visual Basic retorna 4
```

Exemples de llenguatges:

- Tipat fort: C++, Java, Python, Haskell, ...
- Tipat feble: Basic, JavaScript, Perl, ...

Comprovació Estàtica/Dinàmica

- Comprovació estàtica: en temps de compilació.

  Conservadora.
  Codi més eficient.

- Comprovació dinàmica: en temps d'execució.

En llenguatges OO (sense comprovació estàtica) com ara Ruby, es poden donar excepcions d'execució: "Method undefined".

Pot implicar, restringir construccions com l'herència, per garantir que sempre tot serà tiplable (type safe).

Continguts

Sistema de tipus del Haskell

Usarem un estil de regles d'inferència (com al λ-calcul):

Variable:
$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

Declaració:
$$\frac{f :: \tau \in \Gamma}{\Gamma \vdash f : \tau\xi}$$

Abstracció:
$$\frac{\Gamma \cdot \{x : \sigma\} \vdash t : \tau}{\Gamma \vdash (\backslash x.t) : \sigma \to \tau}$$

Aplicació:
$$\frac{\Gamma \vdash s : \sigma \to \tau \quad \Gamma \vdash t : \sigma}{\Gamma \vdash (s\ \ t) : \tau}$$

Let:
$$\frac{\Gamma \vdash s : \sigma \quad \Gamma \cdot \{x : \sigma\} \vdash t : \tau}{\Gamma \vdash \text{let } x = s \text{ in } t : \tau}$$

Context declaration:
$$\frac{C[\alpha] => f :: \tau[\alpha] \in \Gamma \quad C[\sigma] \text{ es satisfà a } \Gamma}{\Gamma \vdash f : \tau[\sigma]\xi}$$
on ξ es una substitució de tipus

Sistema de tipus del Java

Usarem un estil de regles d'inferència (com al λ-calcul):

Constants:

$$\frac{}{\vdash 1 : \texttt{Int}} \qquad \ldots$$

Variables:

$$\frac{x : T \in E}{\vdash x : T}$$

Aplicació de funció:

$$\frac{E \vdash e_1 : T_1 \quad \ldots \quad E \vdash e_n : T_n}{E \vdash obj.f(e_1, \ldots, e_n) : T}$$

Si $obj : Class\{\ldots T\ f(T_1, \ldots, T_n) \ldots\} \in E$

Sistema de tipus del Java

Exemple de derivació de tipus

$$\dfrac{\dfrac{x : \mathit{Object}\{\ldots \mathit{String}\ \mathit{toString}()\ldots\} \in E}{E \vdash x.\mathit{toString}() : \mathit{String}}}{E \vdash \mathit{System.out.println}(x.\mathit{toString}()) : \mathit{void}}$$

ja que

$$\mathit{System.out} : \mathit{PrintStream}\{\ldots\ \mathit{void}\ \mathit{println}(\mathit{String})\ldots\} \in E$$

és a dir, *System.out* és un objecte de la classe *PrintStream*

- Cal afegir la noció de subtipus (herència)
- Cal extendre les regles y afegir-ne de noves.

Sistema de tipus del Java amb herència

Aplicació de funció:

$$\frac{E \vdash e_1 : U_1 \quad \ldots \quad E \vdash e_n : U_n \quad E \vdash U_1 \leq T_1 \quad \ldots \quad E \vdash U_n \leq T_n}{E \vdash obj.f(e_1, \ldots, e_n) : T}$$

Si $obj : Class\{\ldots T\ f(T_1, \ldots, T_n) \ldots\} \in E$

Regles pel subtipat:

Reflexivity:

$$\frac{}{\vdash T \leq T}$$

Object:

$$\frac{T \text{ no és un tipus primitiu}}{\vdash T \leq Object}$$

Implements:

$$\frac{\texttt{class}\ U\ \texttt{implements}\ T \in E}{E \vdash U \leq T}$$

En general, si $U \leq T$ apareix directament a E amb un `implements` o un `extends`

Sistema de tipus del Java amb herència

Més regles pel subtyping

Arrays:

$$\frac{E \vdash U \leq T}{E \vdash U[\,] \leq T[\,]}$$

Transitivitat:

$$\frac{E \vdash U \leq T \qquad E \vdash T \leq S}{E \vdash U \leq S}$$

Cal definir de qui s'hereten els mètodes quan hi ha herència i redefinició de mètodes.

Solució: del més proper!

Problemes amb l'herència múltiple.

Sistema de tipus del Java amb herència

Suposeu que tenim

$$D \leq B \leq A$$
$$D \leq C \leq A$$

Aquí D presenta herència multiple de B i C.

Si D crida a un mètode d'A que ha estat redefinit per B i per C de maneres diferents.

De qui ha d'heretar el mètode?

És un problema de tipus!

Java no admet herència múltiple.

Per exemple, C++ i Python sí ho admeten però de formes diferents.

Inferència de tipus

Mecanisme (re)inventat per Robin Milner

(Curry i Hindley havien desenvolupat idees similars independentment en el context del lambda-calcul)

Donat un programa en un llenguatge de programació

trobar el tipus més general pel programa (y totes les seves expressions) dins del sistema de tipus del llenguatge.

L'algorisme és similar a la "unificació".

- Sempre present als llenguatges funcionals.
- S'ha estès a altres llenguatges.
  Per exemple Visual Basic, C#, C++, ...

Inferència de tipus a C++

A la versió més recent de l'estàndar de C++ (C++0x/C++11) apareix la inferència de tipus.

Paraula clau "auto"

```
auto x = 0;
```

També hi ha la Paraula clau decltype

Obté el tipus d'una expressió.

```
decltype(u+1) m =0;
```

No necessariament genera el mateix tipus que auto. Vegeu l'exemple (`exem.cpp`)

Inferència de tipus a Haskell

- En la majoria de casos no cal definir res.
- Algunes situacions estranyes.
- Problemes amb l'anomenada "Monomorphism restriction"

  Sovint no es pot sobrecarregar una funció si no es dona una declaració explicita de tipus (veure exemples).

Es poden demanar els tipus inferits.

Continguts

L'algorisme de Milner

Noms:

- Hindley–Milner
- Damas–Milner
- Damas–Hindley–Milner

Propietats:

- Complet
- Computa el tipus més general possible sense necessitat d'anotacions
- Eficient: gairebé lineal (inversa de la funció d'Ackermann)
  L'eficiència depend de l'algorisme d'unificació que s'apliqui.

L'algorisme de Milner

1. S'assigna un tipus a l'expressió i a cada subexpressió.
   - Si el tipus 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

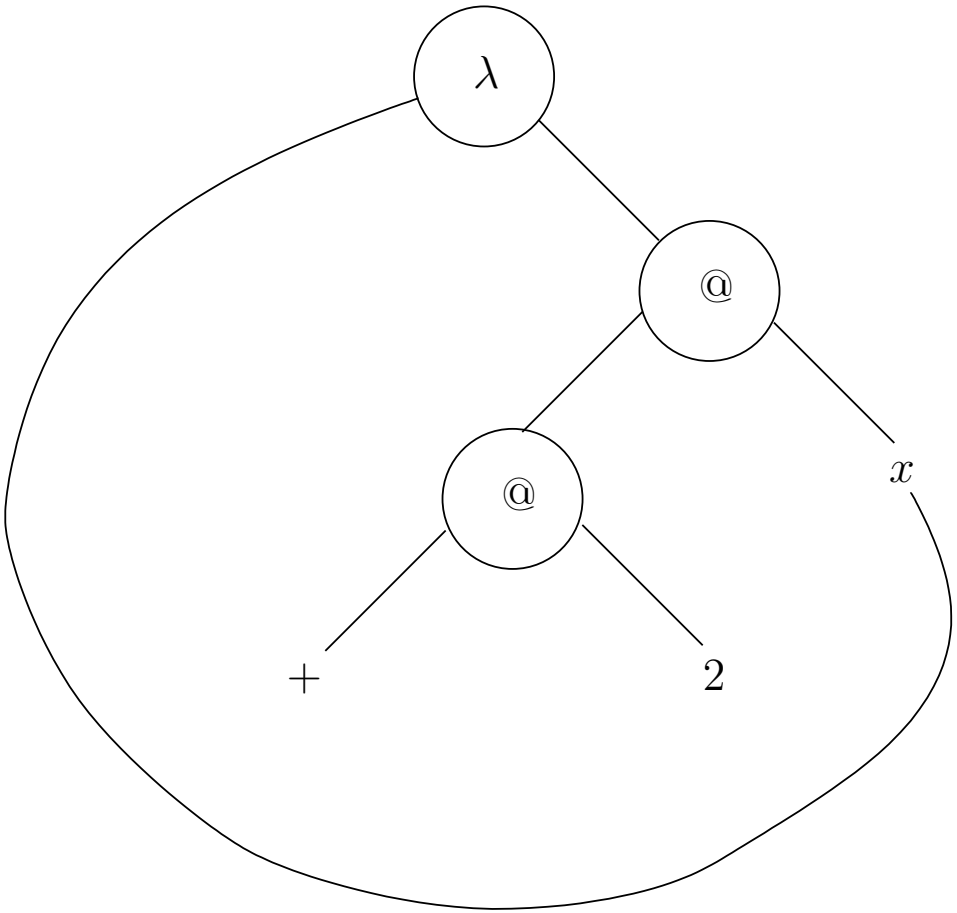
- Considereu l'expressió: $(+\ 2\ x)$

L'algorisme de Milner

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

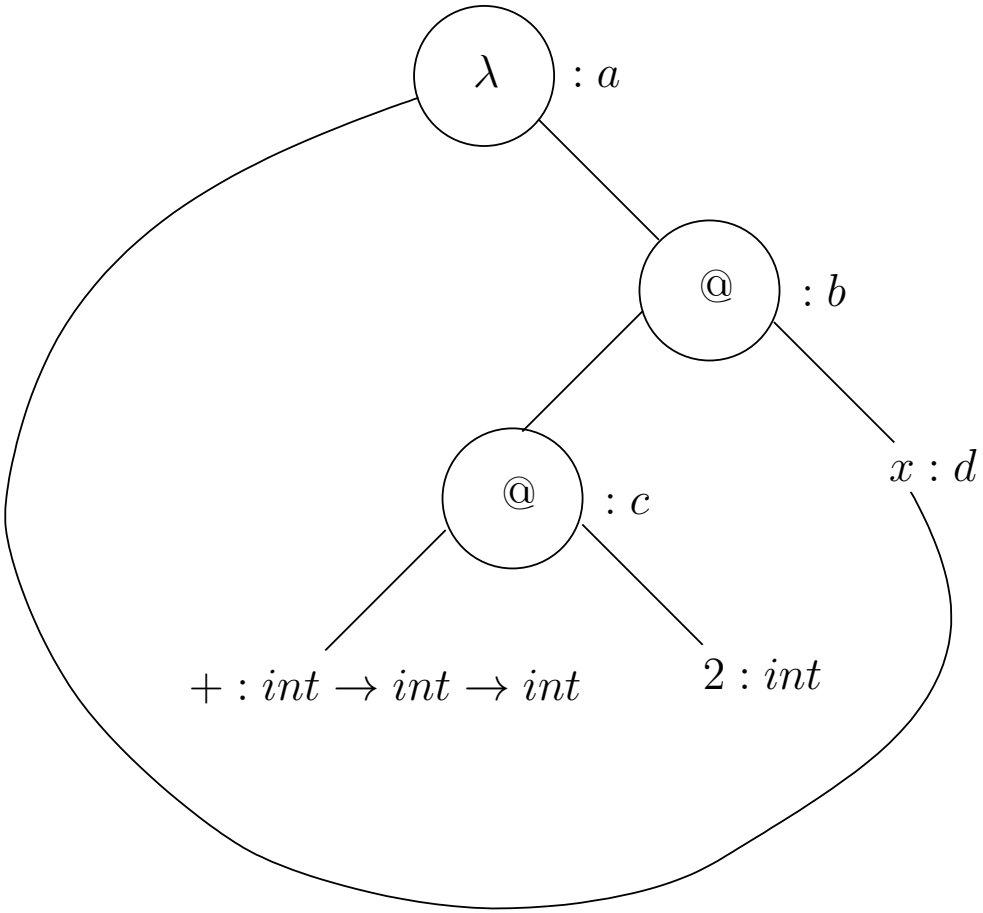
L'algorisme de Milner

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



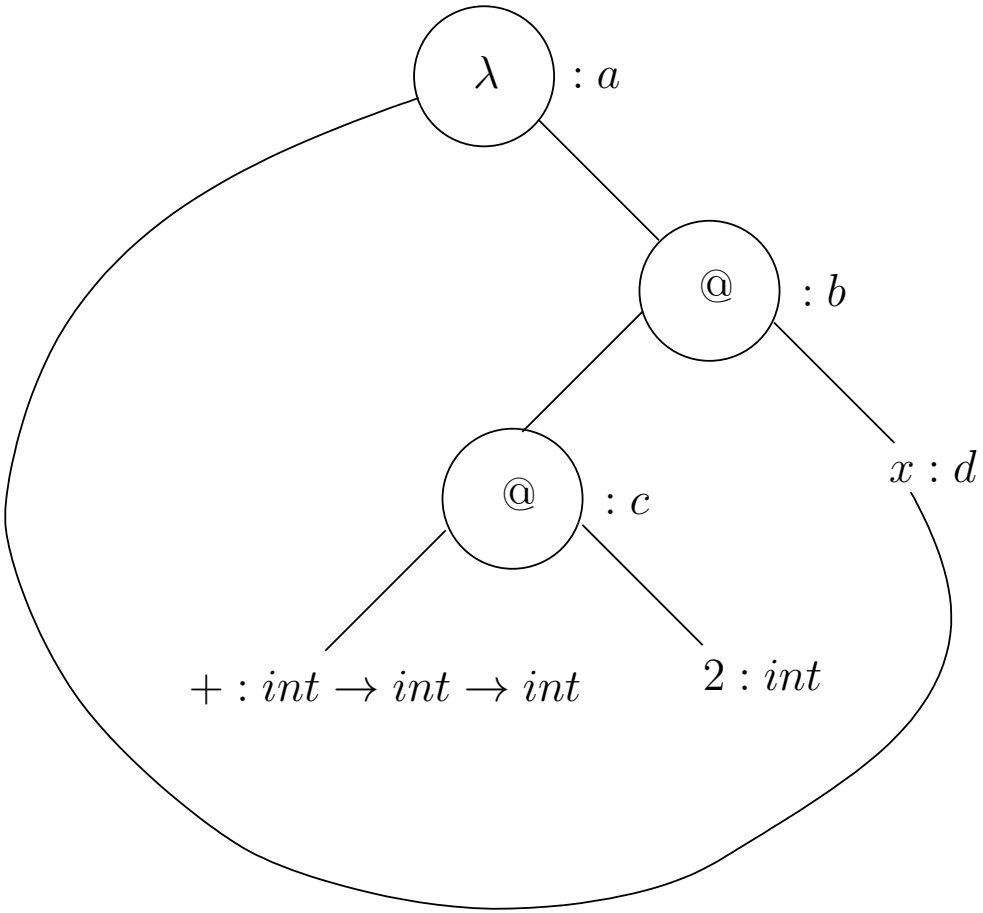
L'algorisme de Milner

1. Assignem tipus a totes les expressions



L'algorisme de Milner

2. Generem les restriccions/equacions



Equacions:

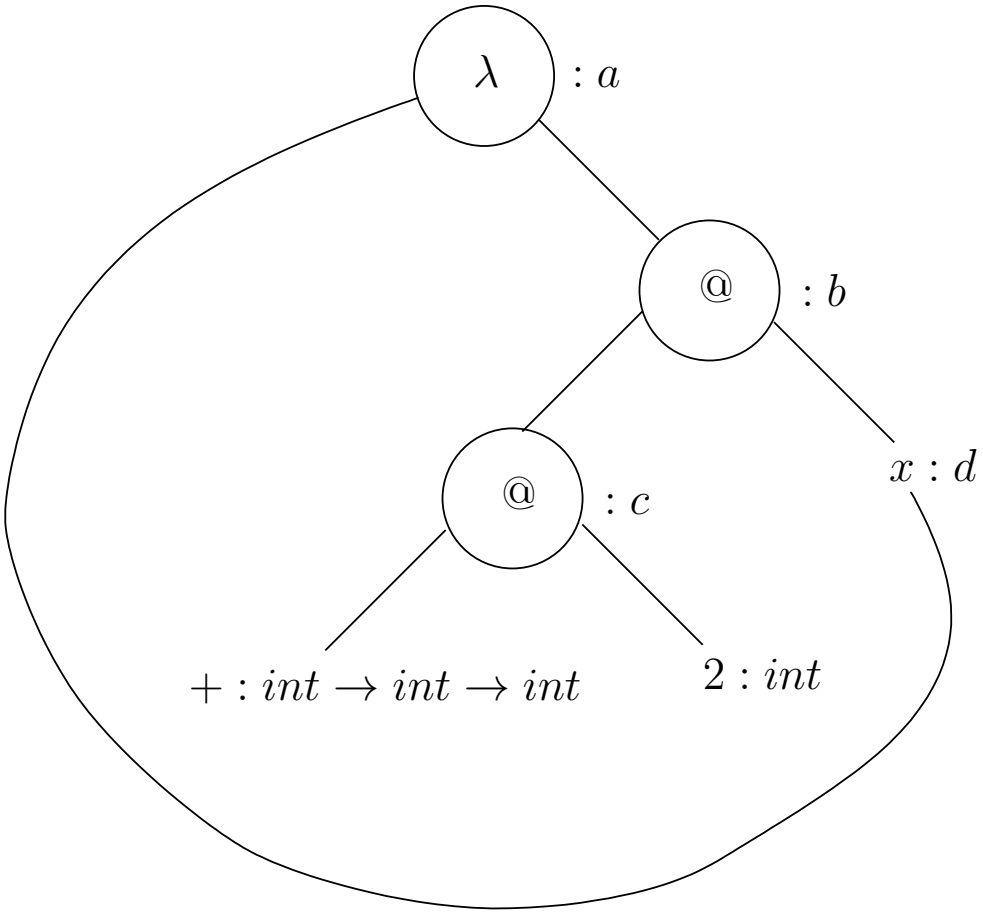
$a = d \to b$

$c = d \to b$

$int \to int \to int = int \to c$

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:

$a = int \to int$

$b = int$

$c = int \to int$

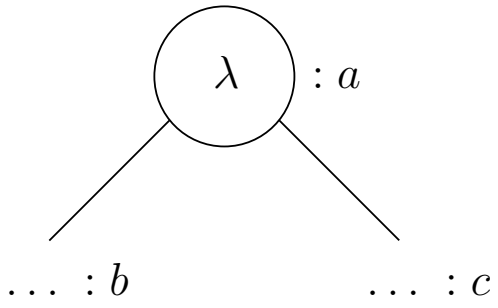
$d = int$

$\boxed{int \to int}$

L'algorisme de Milner

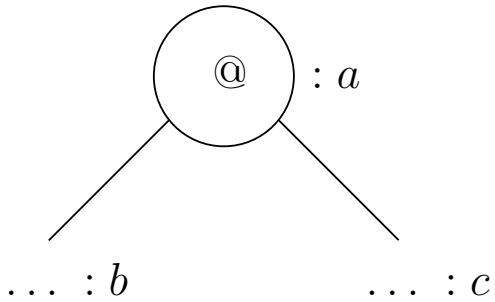
Aquestes són les regles per generar les equacions:

Abstracció:



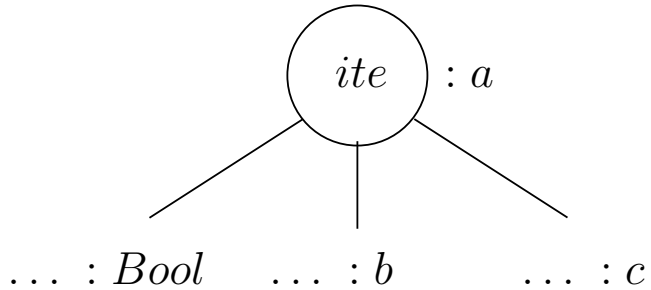
$$a = b \to c$$

Aplicació:



$$b = c \to a$$

IfThenElse:



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

L'algorisme de Milner

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

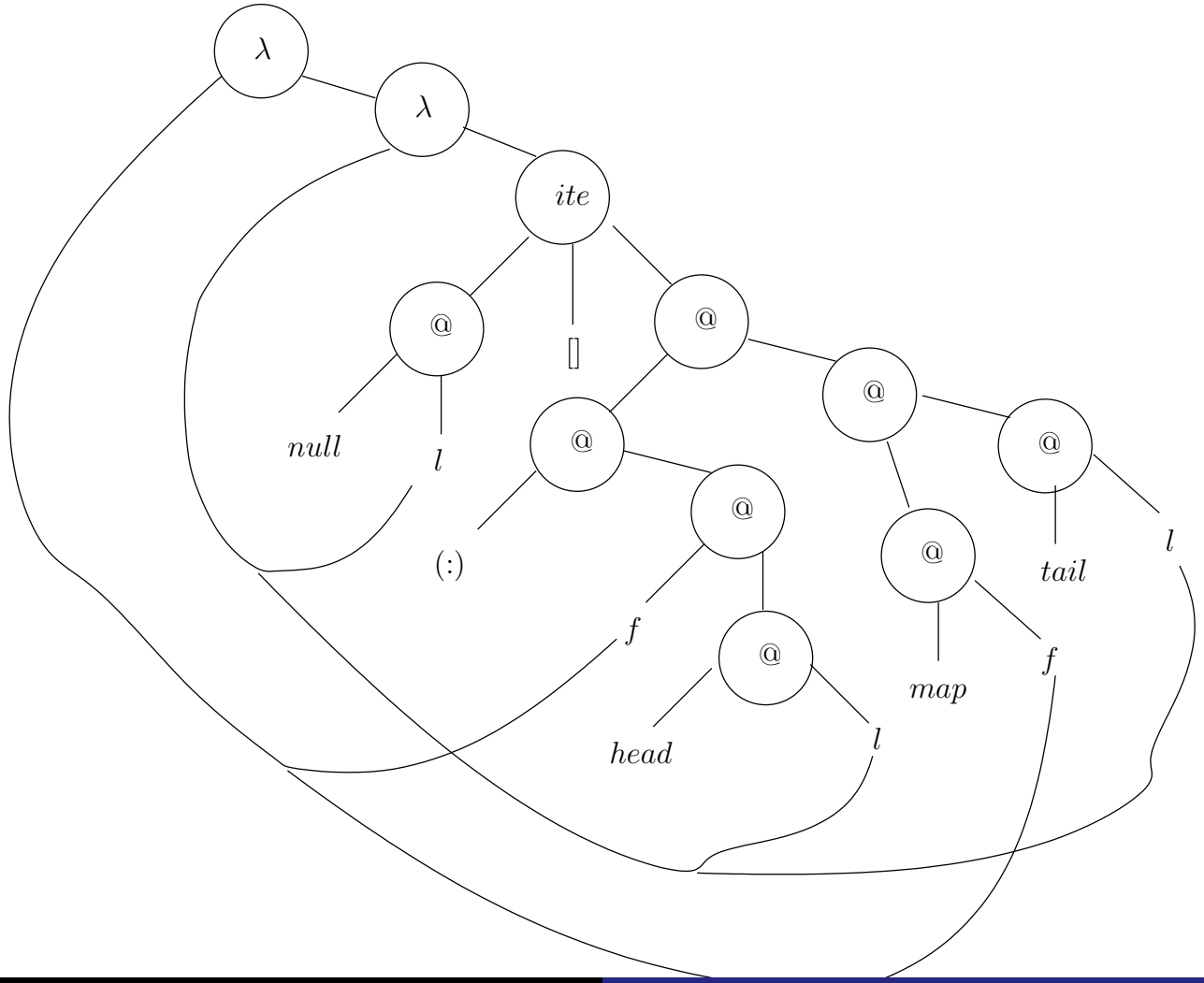
Podem entendre una definició, com una funció que aplicada als paràmetres ens torna la part dreta de la definició.

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

Noteu que el tipus de if_then_else és:
$if\_then\_else : Bool \rightarrow a \rightarrow a \rightarrow a$

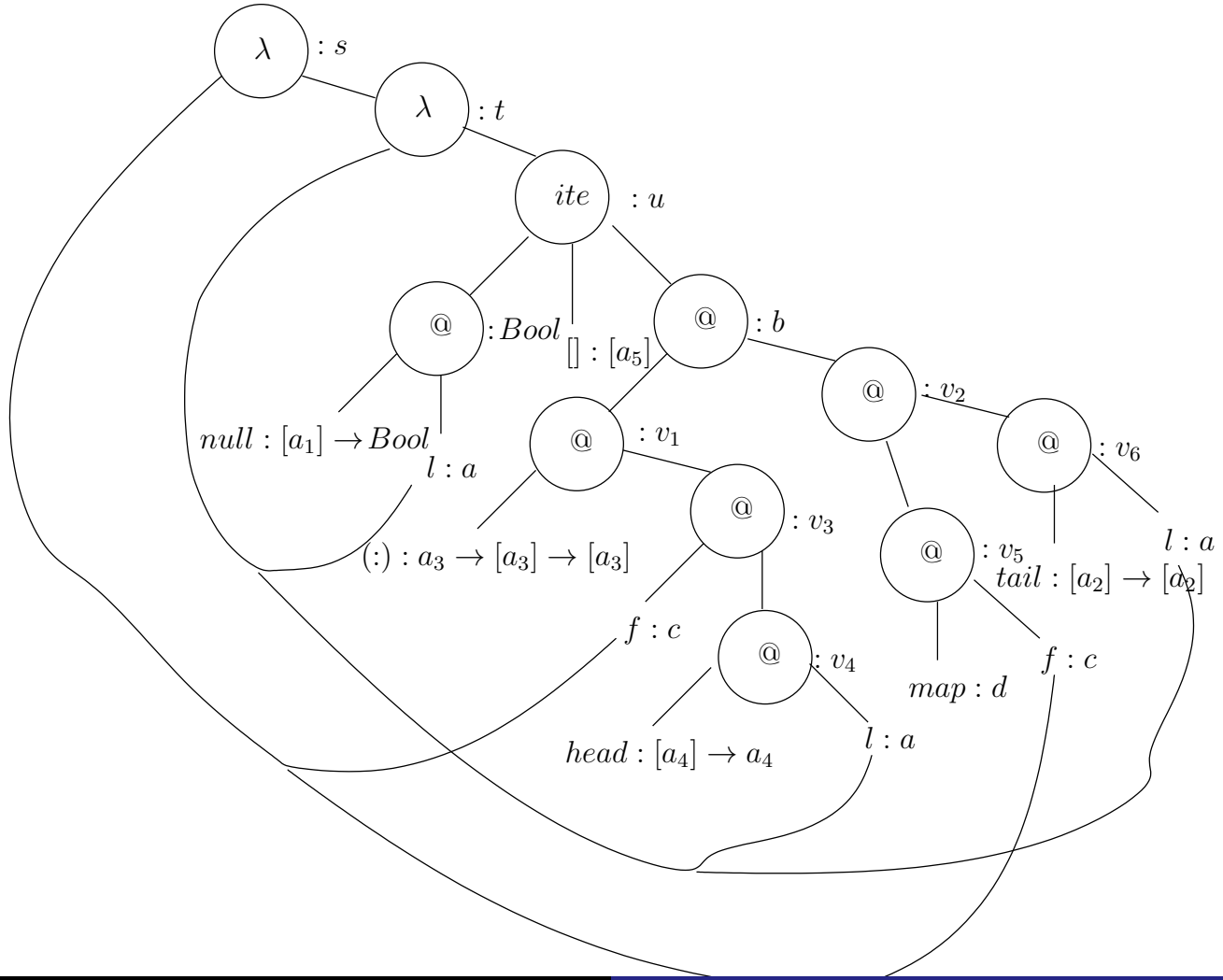
L'algorisme de Milner

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



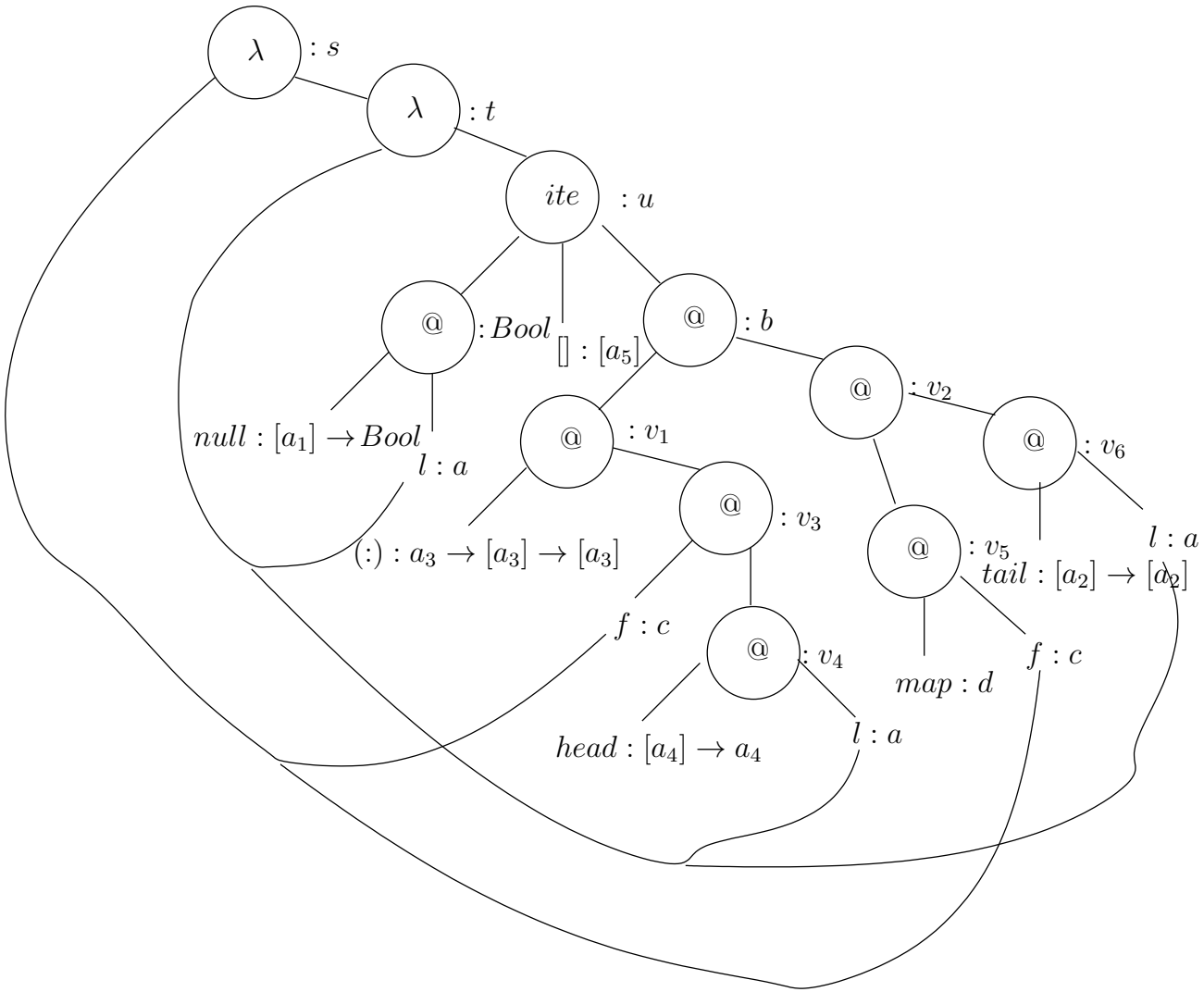
L'algorisme de Milner

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



L'algorisme de Milner

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



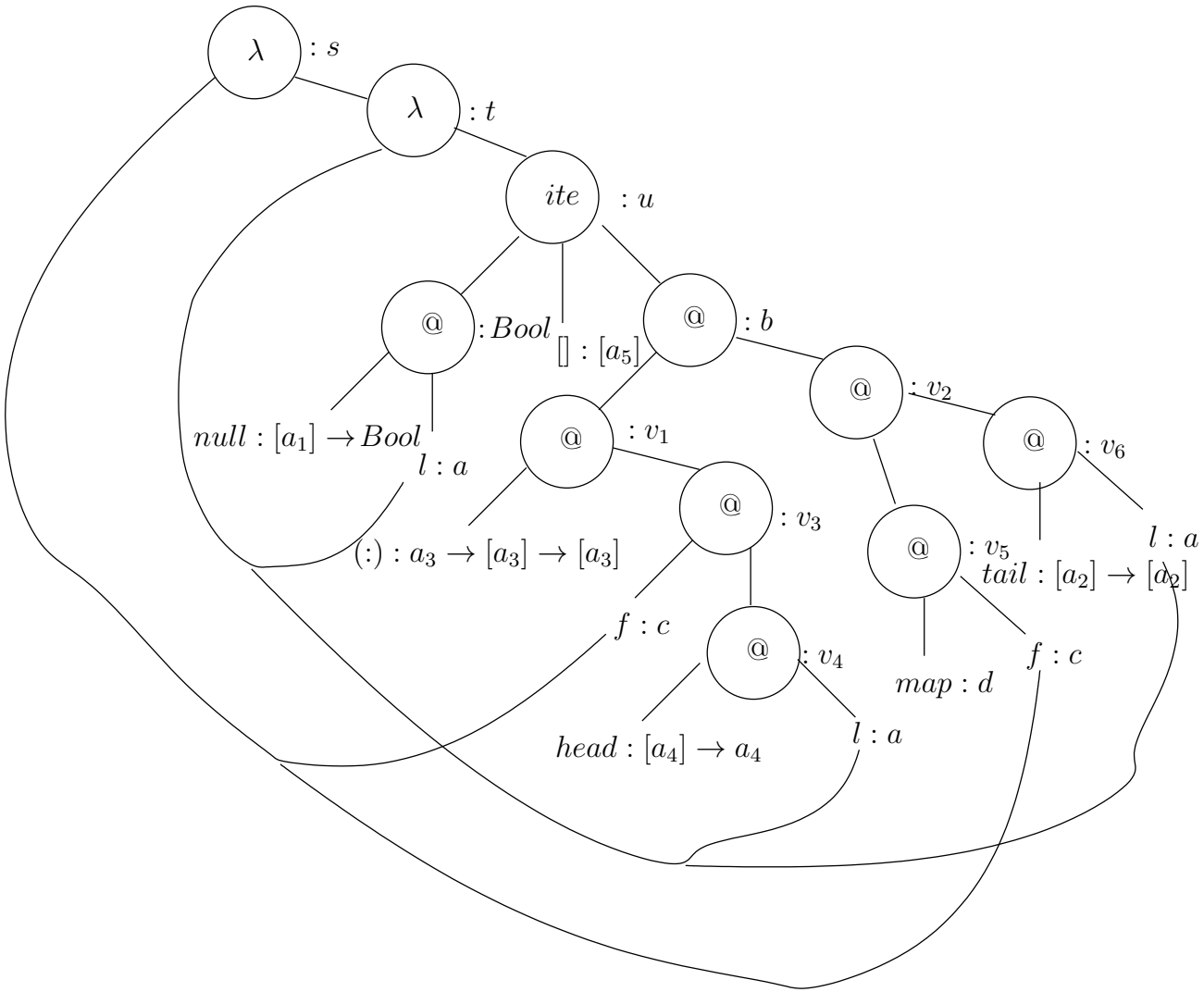
Equacions:

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

$s = d$

L'algorisme de Milner

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



Solució:

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

$$\boxed{(a_1 \to a_3) \to [a_1] \to [a_3]}$$

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:

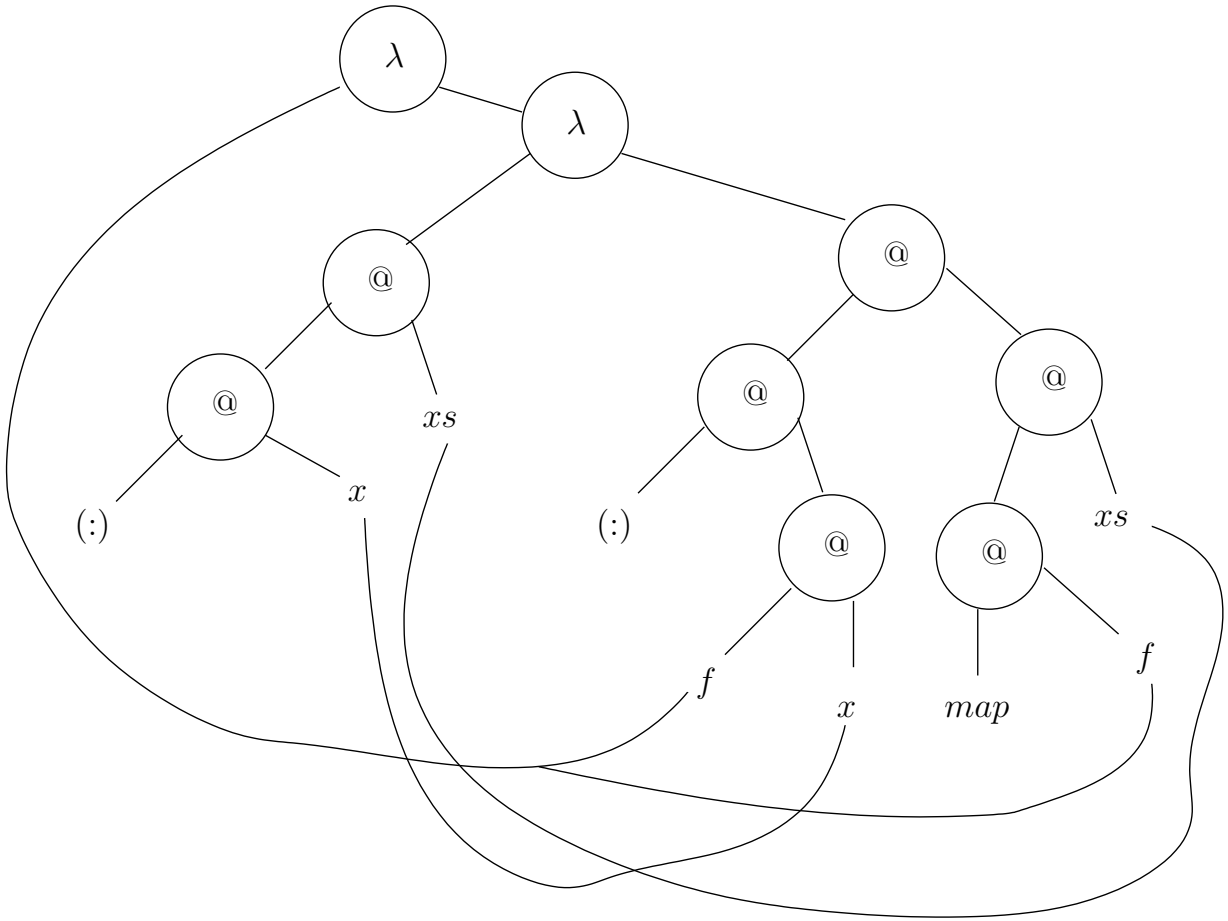
$\lambda\ f.\ \lambda\ (x : xs).\ (f\ x) : (\text{map}\ f\ xs)$

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.

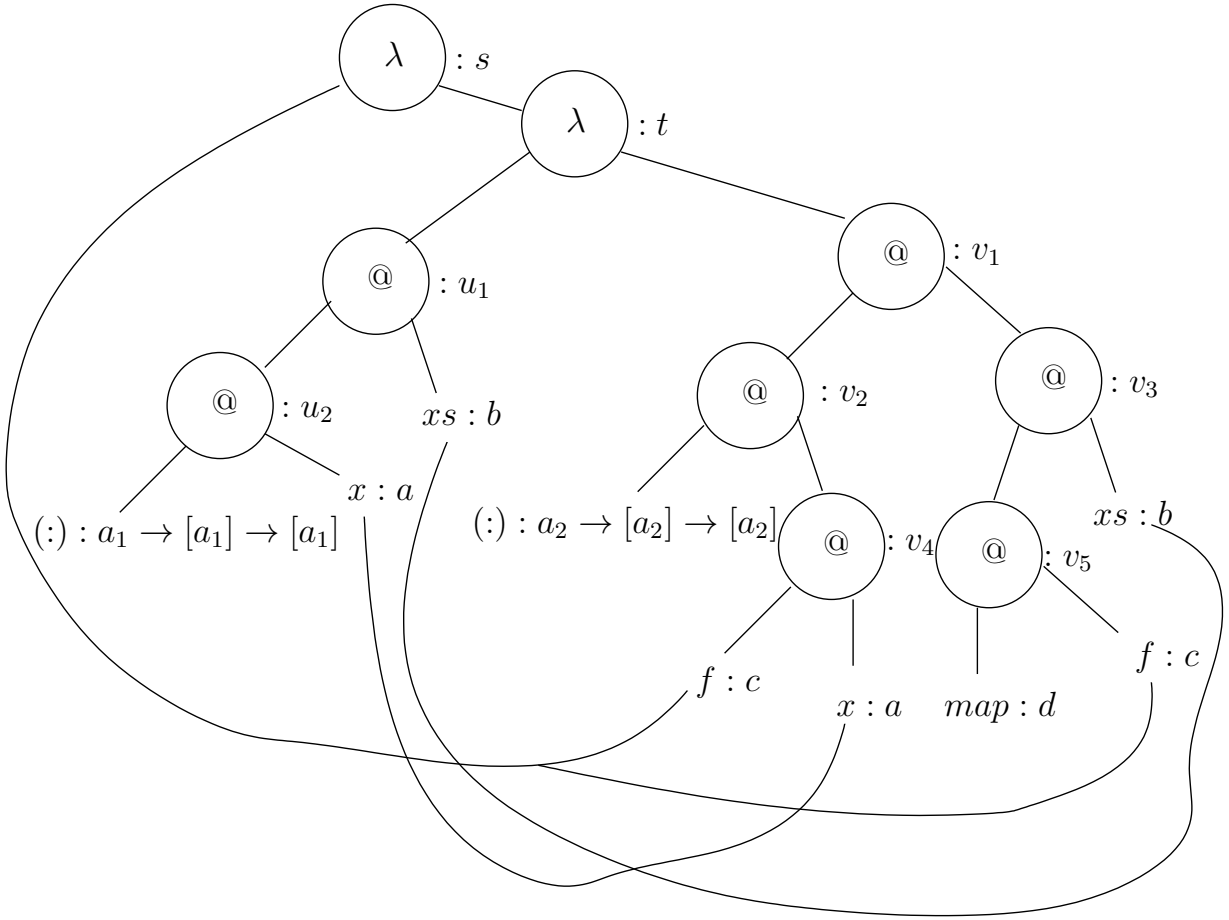
L'algorisme de Milner

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



L'algorisme de Milner

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



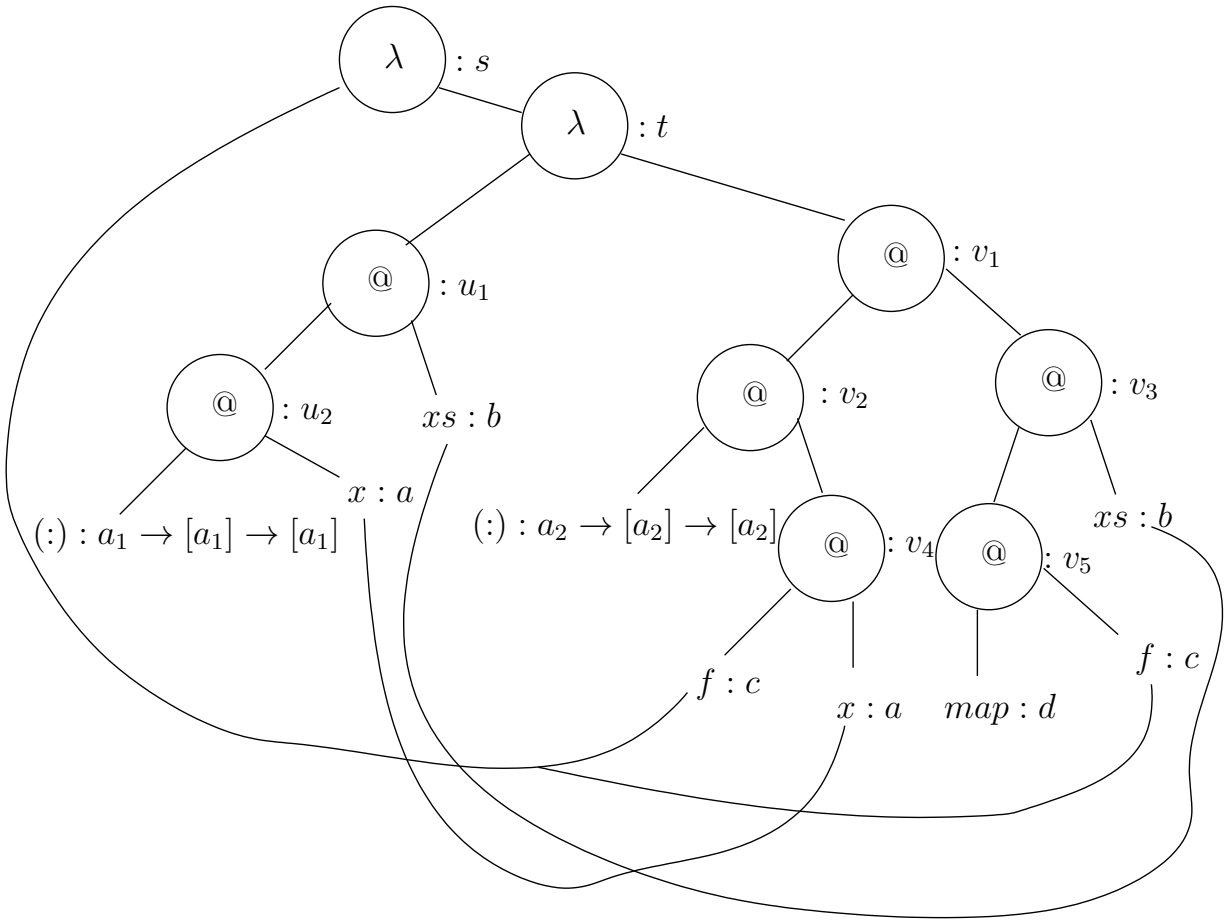
Equacions:

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

$$s = d$$

L'algorisme de Milner

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



Solució:

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

$$\boxed{(a \to a_2) \to [a] \to [a_2]}$$

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

$(+) :: \mathit{Num}\ a \Rightarrow a \to a \to a$
$(>) :: \mathit{Ord}\ \ a \Rightarrow a \to a \to \mathit{Bool}$

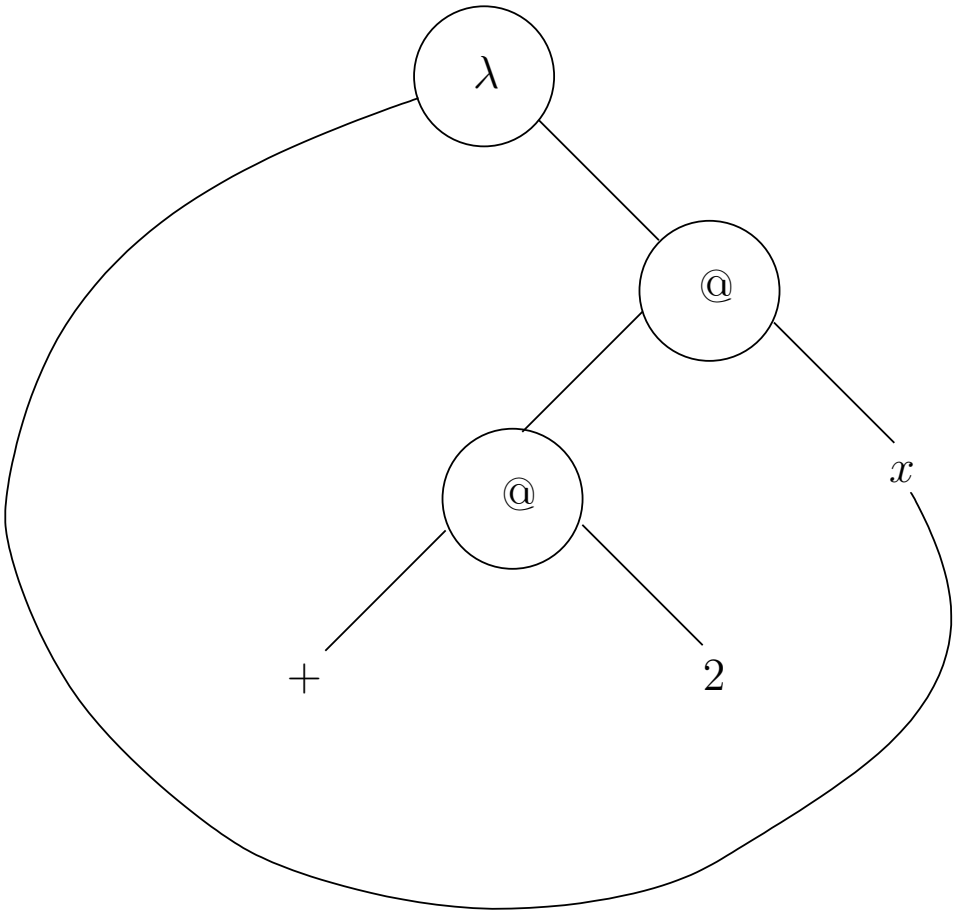
Això introdueix un nou tipus de restricció *de context*.

Per tant les solucions també

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

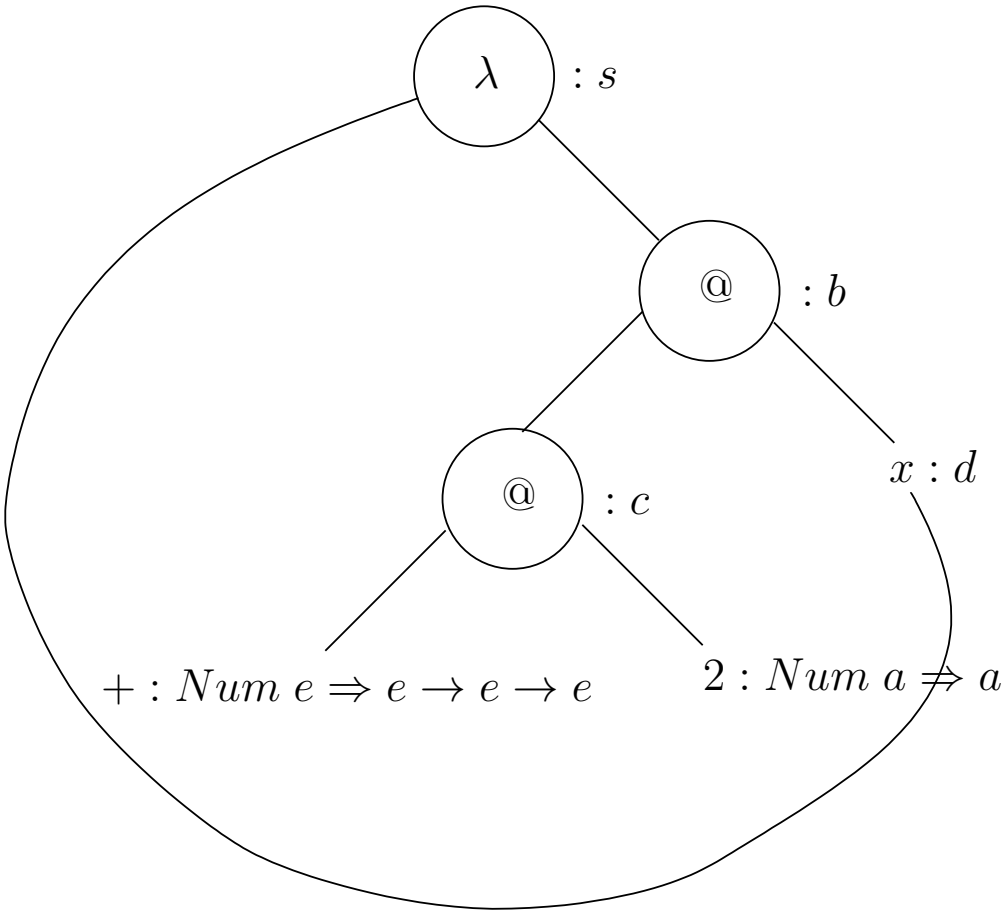
L'algorisme de Milner (Classes)

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



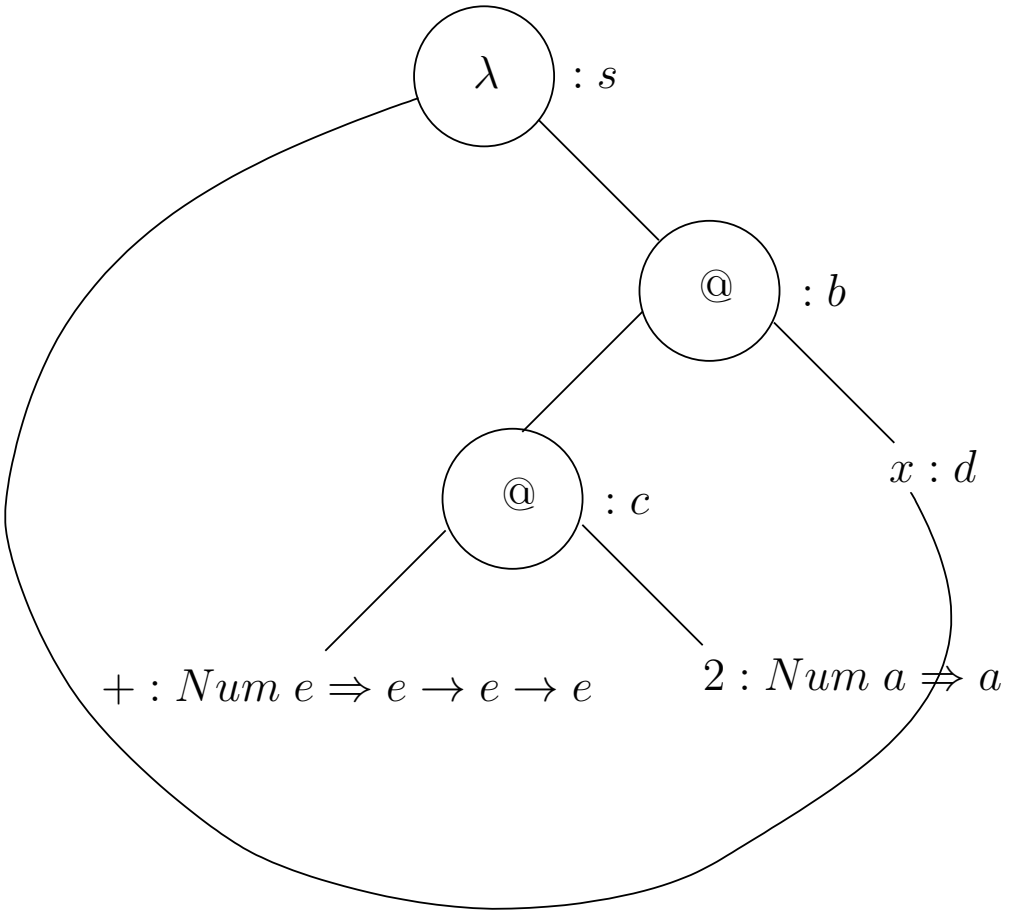
L'algorisme de Milner (Classes)

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



L'algorisme de Milner (Classes)

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



Equacions:

$s = d \to b$

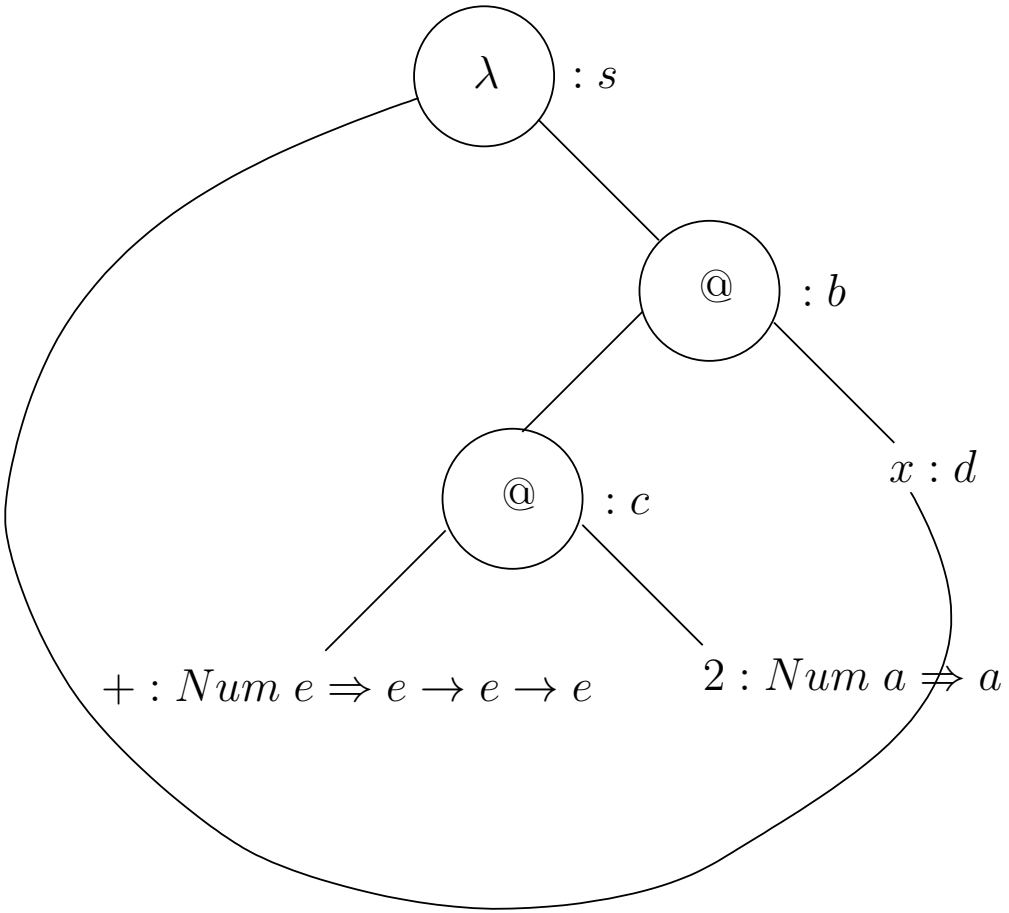
$c = d \to b$

$e \to e \to e = a \to c$

$Num(e),\ Num(a)$

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:

$s = a \to a$

$b = a$

$c = a \to a$

$d = a$

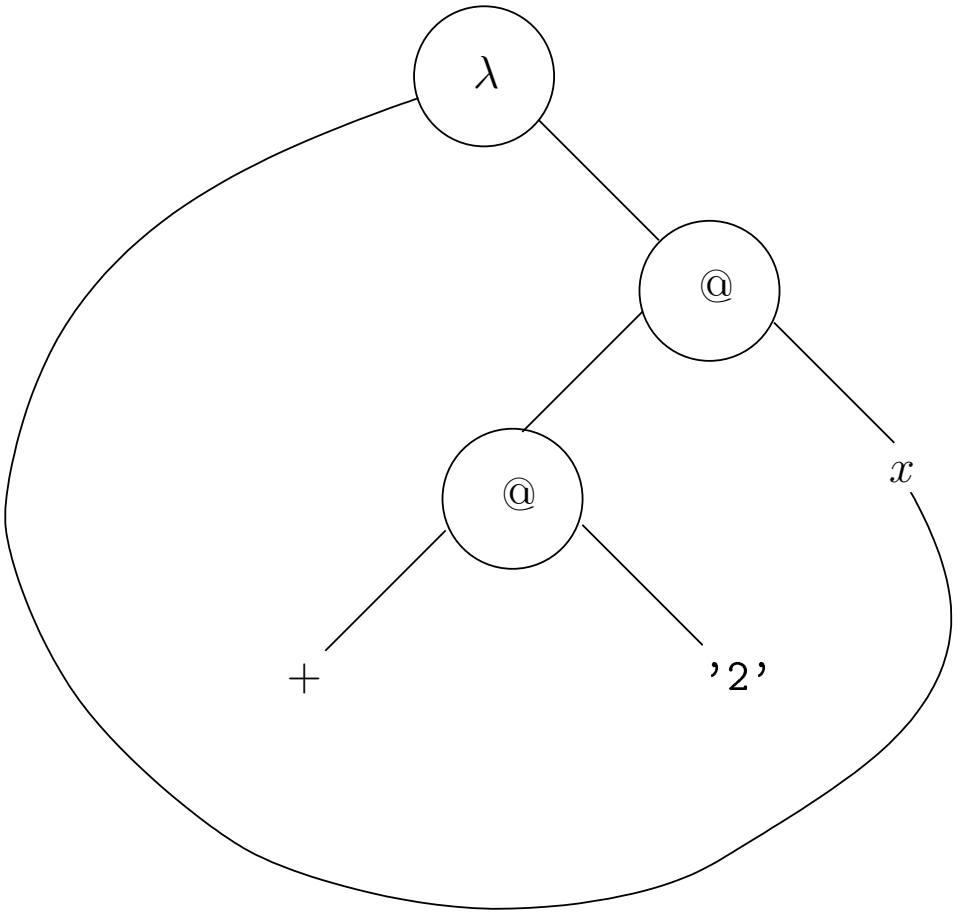
$e = a$

$Num\ a$

$$\boxed{Num\ a \Rightarrow a \to a}$$

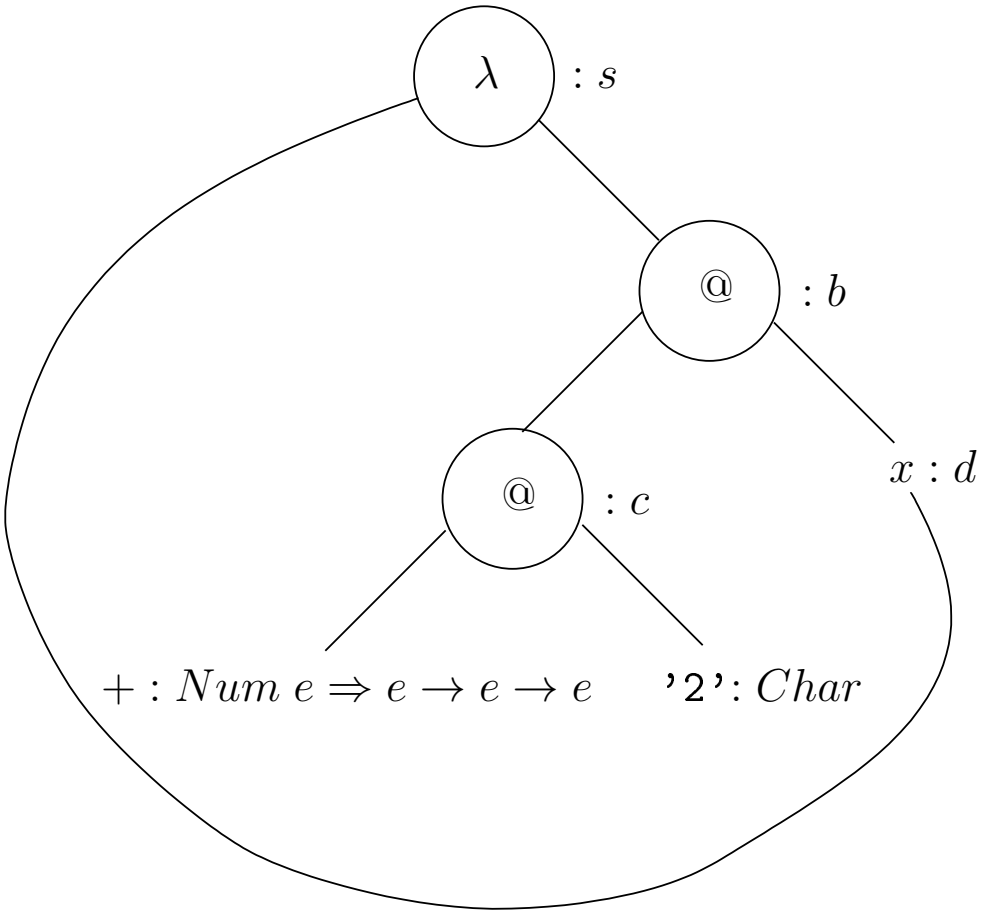
L'algorisme de Milner (Classes)

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



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`
   ```