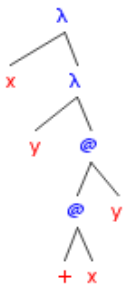
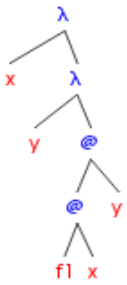
Linguagens de Programação

Inferência de Tipos
– Soluções –

1. (a) `fun f1 x y = x + y;`
   `f1` : $int \rightarrow int \rightarrow int$



   (b) `fun f2 x y = f1 x y;`
   `f2` : $int \rightarrow int \rightarrow int$



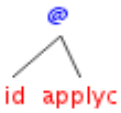
   (c) `fun f3 x = f1 x;`
   `f3` : $int \rightarrow int \rightarrow int$

   (d) `fun id x = x;`
   `id` : `'a` $\rightarrow$ `'a`

   (e) `fun apply (f, x) = f x;`
   `apply` : `('a` $\rightarrow$ `'b) * 'a` $\rightarrow$ `'b`

   (f) `fun applyc f x = f x;`
   `applyc` : `('a` $\rightarrow$ `'b)` $\rightarrow$ `'a` $\rightarrow$ `'b`

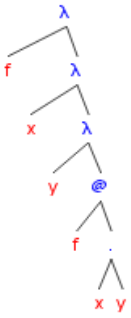
(g) `id applyc;`
`id applyc : ('a → 'b) → 'a → 'b`



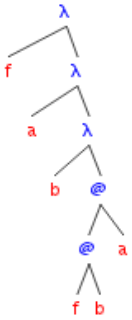
(h) `applyc id;`
`applyc id : 'a → 'a`

(i) `apply (f1, 5);`
`apply (f1, 5)` : $int \to int$

(j) `fun curry f x y = f (x, y);`
`curry` : `('a * 'b -> 'c) -> 'a -> 'b -> 'c`



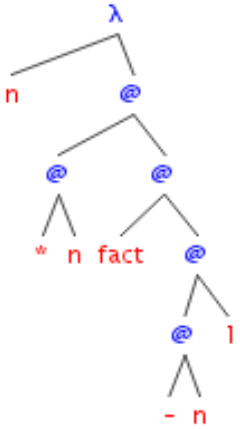
(k) `fun flip f a b = f b a;`
`flip` : `('a -> 'b -> 'c) -> 'b -> 'a -> 'c`

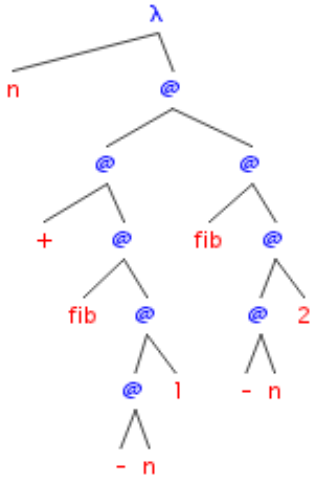


(l) `fun fact 0 = 1`
`| fact n = n * fact (n - 1);`
`fact` : $int \to int$

(m) `fun fib n = fib (n - 1) + fib (n - 2);`
`fib` : $int \to int$

(n) `fun f4 (m, n) = f4 (m + n, m);`
`f4` : $int * int \to$ `'a`





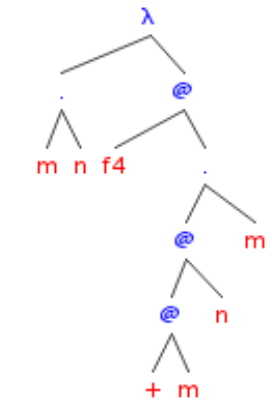
(o) `fun f5 (h,x) = h (f5 (h, x - 1), x);`
`f5` : `('a *` $int \rightarrow$ `'a) *` $int \rightarrow$ `'a`

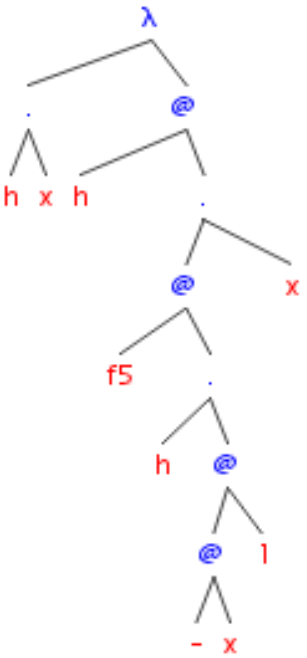
(p) `fun f6 h x = h (f6 h (x - 1)) x;`
`f6` : `('a`$\rightarrow$ $int \rightarrow$ `'a)`$\rightarrow$ $int \rightarrow$ `'a`

(q) `fun f7 g a = g (g a);`
`f7` : `('a` $\rightarrow$ `'a)` $\rightarrow$ `'a` $\rightarrow$ `'a`

(r) `fun e1 f = f (e1 0);`
erro de tipos!!!

(s) `fun e2 f = f f + 2;`





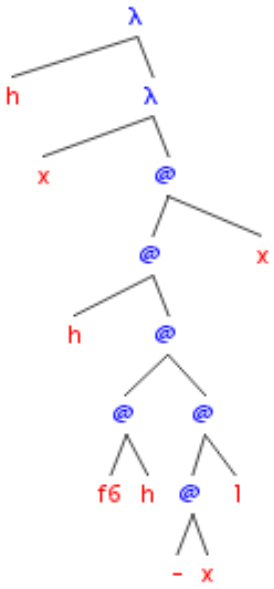
erro de tipos!!!

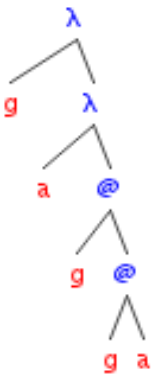
(t) `fun o f g x = f (g x);`
`o` : `('a` → `'b)` → `('c` → `'a)` → `'c` → `'b`

(u) `fun ou (f, g, x) = f (g x);`
`ou` : `('a` → `'b)` * `('c` → `'a)` * `'c` → `'b`

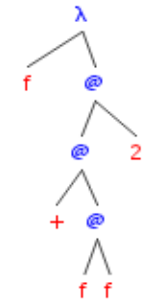
(v) `fun pr (f, a, x) = f (x, pr (f, a, a x));`
`pr` : `('a` * `'b` → `'b)` * `('a` → `'a)` * `'a` → `'b`

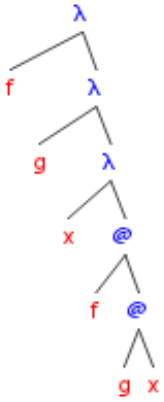
(w) `fun len l = if l = [] then 0 else 1 + len (tl l);`
`len` : `'a` *list* → *int*

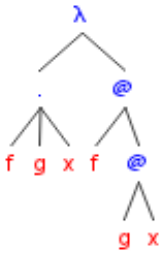




(x) `fun ape l m = if l = [] then m else ape (tl l) m;`
Como `ape` : `'a` *list* $\rightarrow$ `'b` $\rightarrow$ `'b`
e `ape` : `'a` *list* $\rightarrow$ `'b` $\rightarrow$ `'c`
Temos que `ape` : `'a` *list* $\rightarrow$ `'b` $\rightarrow$ `'b`



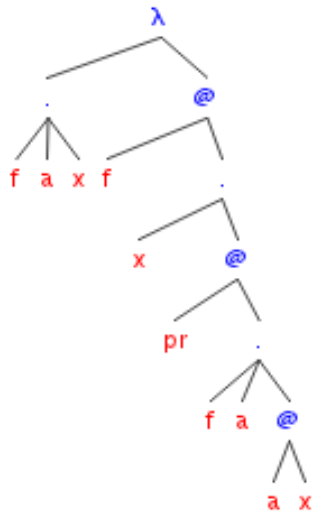


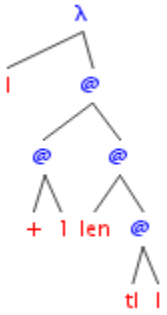


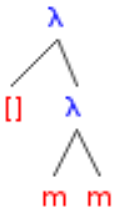
(y) `fun app l m = if l = [] then m else hd l :: app (tl l) m;`
Como `app` : `'a` *list* → `'a` → `'a`
e `app` : `'a` *list* → `'a` *list* → `'a` *list*
Temos que `app` : `'a` *list* → `'a` *list* → `'a` *list*

(z) `fun rev [] = []`
`| rev l = app (rev (tl l)) (hd l :: []);`
`rev` : `'a` *list* → `'a` *list*

2. (a) Como `fun f x y = M; f 4 5`
no qual `M = let fun g x = 3 * x + y in g (x - 1);`
é equivalente a `let f =` λ`x.`λ`y.M in f 4 5;`
Sabendo que `let x = N in M` é equivalente a `(`λ`x.M) N`





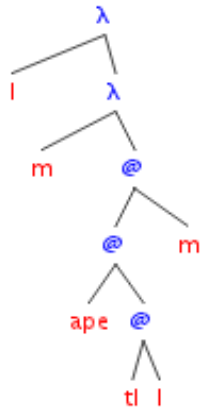


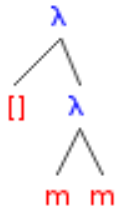
Teremos então $(\lambda$`f.f 4 5)(`λ`x.`λ`y.(`λ`g.g (x-1))(`λ`x.3*x+y))`

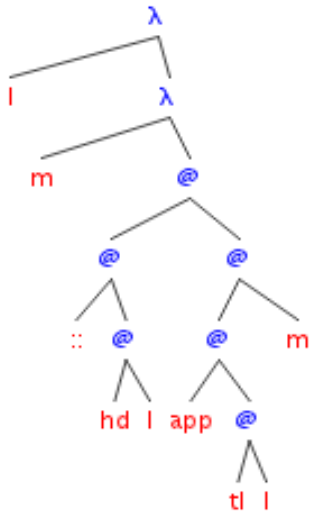
(b) $3 * (4 - 1) + 5 = 3 * 3 + 5 = 14$

(c) `f` : $int \to int \to int$

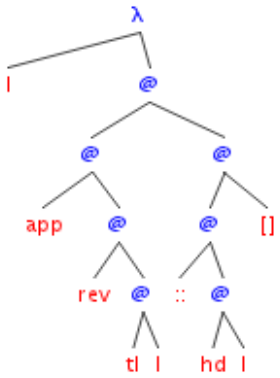
3. (a) Árvore Sintáctica:

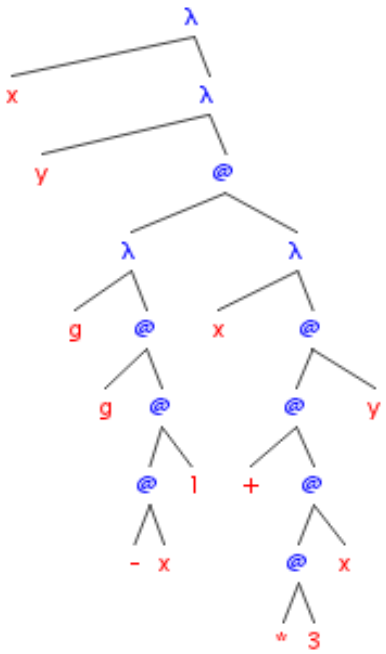


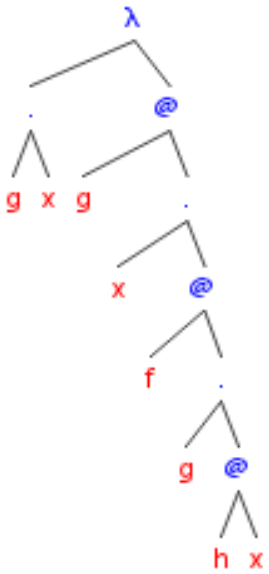




(b) `f`: $(\texttt{'a} * \texttt{'b} \to \texttt{'b}) * \texttt{'a} \to \texttt{'b}$


λ
l
@
@
@
app
@
@
[]
rev
@
::
@
tl
l
hd
l


λ
x
λ
y
@
λ
λ
g
@
x
@
g
@
@
y
@
l
+
@
-
x
@
x
*
3


λ
.
@
g
x
g
.
x
@
f
.
g
@
h
x