

Fundamentos de lenguajes de programación

Semántica de los Conceptos Fundamentales de Lenguajes de Programación

carlos.andres.delgado@correounivalle.edu.co

Carlos Andrés Delgado S. Carlos Alberto Ramírez

Facultad de Ingeniería. Universidad del Valle

Noviembre de 2016

Fundamentos
de lenguajes
de
programación

Carlos Andrés
Delgado S.
Carlos Alberto
Ramírez

Inferencia de
tipos

1 Inferencia de tipos

Fundamentos
de lenguajes
de
programación

Carlos Andrés
Delgado S.
Carlos Alberto
Ramírez

Inferencia de
tipos

1 Inferencia de tipos

- Algunos lenguajes de programación dejan que el compilador averigüe los tipos de todas las variables.
- Esto es llevado a cabo mediante la observación de cómo son usadas las variables y utilizando ayudas que el programador pueda aportar.
- Esta estrategia es llamada *inferencia de tipos*.
- Para el lenguaje en desarrollo, todas las expresiones de tipo son opcionales, y en donde no se coloquen, se pondrá el símbolo ?.

De esta manera, un programa puede verse como:

```
letrec
  (? even (? odd, int x) =
    if zero?(x) then 1 else (odd sub1(x))
in letrec
  bool odd(? x)
    if zero?(x) then 0 else (even odd sub1(x))
  in (odd 13)
```

Los tres símbolos de interrogación indican el lugar donde un tipo debe ser inferido.

Se agregan las siguientes reglas de producción a la gramática:

$$\langle \text{tipo-exp-opcional} \rangle ::= \langle \text{tipo-exp} \rangle$$

$\text{a-type-exp } (\text{texp})$

$$\langle \text{tipo-exp-opcional} \rangle ::= ?$$

$\text{no-type-exp } ()$




```

⟨expresión⟩ ::= letrec
    {⟨tipo-exp-opcional⟩ ⟨identificador⟩
      ( {⟨tipo-exp-opcional⟩ ⟨identificador⟩ }*(,) =
        ⟨expresión⟩ }*
    in ⟨expresión⟩
    letrec-exp
    (optional-result-texps proc-names
     optional-arg-texpss idss bodies
     letrec-body)

```

Se deben añadir las siguientes producciones a la especificación de la gramática:

```
( expression
  ("proc" "(" (separated-list tipo-exp-opcional
    identifier ",")
    ")"
    expression)
  proc-exp)
(expression
  ("letrec"
    (arbno tipo-exp-opcional identifier
      "(" (separated-list tipo-exp-opcional
        identifier ",") ")"
      "=" expression) "in" expression)
  letrec-exp)
```



Así mismo, se añaden las producciones correspondientes a las expresiones `tipo-exp-opcional` a la especificación de la gramática:

```
( tipo-exp-opcional ( tipo-exp ) a-type-exp )  
( tipo-exp-opcional ( "?" ) no-type-exp )
```

- Para operar con tipos desconocidos (expresados con los símbolos ?), se agregará una nueva clase de tipo llamada *variable de tipo*.
- Cada variable de tipo constará de un único número serial que lo identifica y un contenedor, el cual será un vector de tamaño 1.

- Una variable de tipo puede ser *vacía* (si almacena un `()`, que significa que no se conoce nada del tipo).
- Puede ser *llena* (almacena un tipo).
- Cuando una variable de tipo está llena, su contenido nunca cambia. Dicha variable de tipo es llamada de *asignación simple* o de *única escritura*.

El tipo de dato `type` es modificado, agregándole una nueva variante para las variables de tipo:

```
(define-datatype type type?
  (atomic-type
    (name symbol?))
  (proc-type
    (arg-types (list-of type?))
    (result-type type?))
  (tvar-type
    (serial-number integer?)
    (container vector?)))
```

Se define el procedimiento `fresh-tvar` que crea una variable de tipo, con un valor único global para su contador y con su vector inicializado en `()`.

```
(define fresh-tvar
  (let ((serial-number 0))
    (lambda ()
      → (set! serial-number (+ 1 serial-number))
        (tvar-type serial-number (vector '()))))))
```

- Se deben cambiar los llamados al procedimiento `expand-type-expression` por `expand-optional-type-expression` para estar acorde con los cambios hechos en la gramática.
- El procedimiento `expand-optional-type-expression` recibe una expresión de tipo opcional y un ambiente de tipos y se comporta de la siguiente manera:
 - Si encuentra una expresión de tipo (la expresión corresponde a la variante `a-type-exp`), llama a `expand-type-expression`.
 - Si se trata de una expresión de tipo opcional (denotada por `?`), emite una variable de tipo.

El procedimiento `expand-optional-type-expression` estará definido así:

```
(define expand-optional-type-expression
  (lambda (otexp tenv)
    (cases optional-type-exp otexp
      (no-type-exp () (fresh-tvar))
      (a-type-exp (texp) (expand-type-expression texp
        tenv))))))
```

- Para cada expresión posible en el lenguaje, se obtienen algunas ecuaciones entre tipos y variables de tipo.
- Por ejemplo, cuando se escribe una expresión condicional

if e_0 then e_1 else e_2 en $tenv$, se tiene:

$$\begin{aligned} &(\text{type-of-expression } \ll e_0 \gg tenv) = \text{bool} \\ &(\text{type-of-expression } \ll e_1 \gg tenv) \\ &= (\text{type-of-expression } \ll e_2 \gg tenv) \\ &= (\text{type-of-expression } \ll \text{if } e_0 \text{ then } e_1 \text{ else } e_2 \gg tenv) \end{aligned}$$

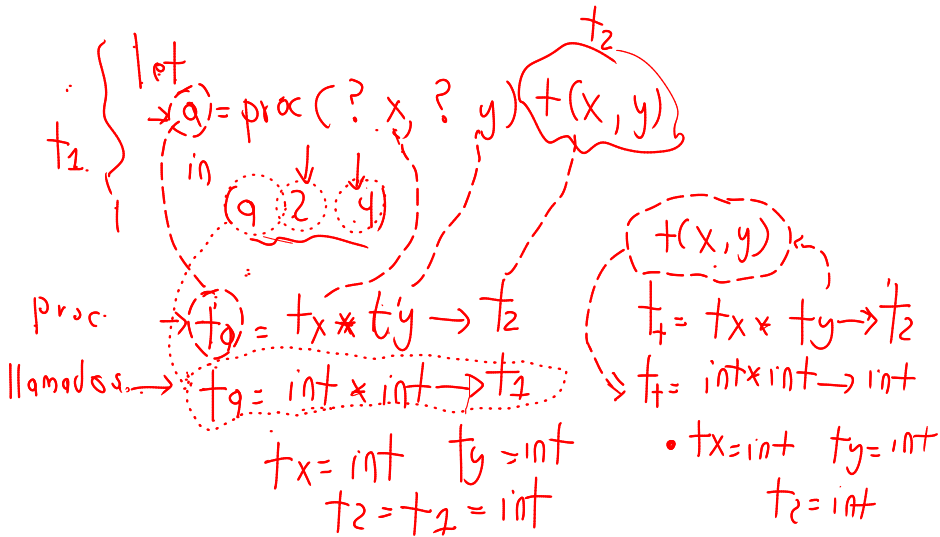
if x then ($p?$) else k

$t_x = \text{bool}$ $t_p = \text{int} \rightarrow t_1$ $t_1 = t_k$

Para las expresiones de aplicación de procedimientos (*rator* $rand_1 \dots rand_n$) en *tenv* se tiene:

```
(type-of-expression  $\ll rator \gg$  tenv) =  
((type-of-expression  $\ll rand_1 \gg$  tenv)  
 * ... *  
 (type-of-expression  $\ll rand_n \gg$  tenv)  
 ->  
 (type-of-expression  $\ll (rator \ rand_1 \dots rand_n) \gg$  tenv))
```

Lo anterior significa que en cada aplicación, el operador debe ser un procedimiento que asigna los tipos de los operandos al tipo de la aplicación entera.



Por último, cuando se escribe una expresión $\text{proc } (x_1 \dots x_n) \text{ exp}$ evaluada en el ambiente de tipos tenv , se tiene:

```
(type-of-expression  $\ll \text{proc } (x_1 \dots x_n) \text{ exp} \gg \text{tenv}$ ) =  
((type-of-expression  $x_1 \text{ tenv}_{\text{body}}$ )  
 *...*  
 (type-of-expression  $x_n \text{ tenv}_{\text{body}}$ )  
 ->  
 (type-of-expression  $\ll \text{exp} \gg \text{tenv}_{\text{body}}$ ))
```

donde $\text{tenv}_{\text{body}}$ es el ambiente de tipo en el cual el cuerpo exp será tipado.

Entonces, para deducir el tipo de una expresión:

- Se introduce una variable de tipo para cada variable ligada y cada aplicación, y
- se escribe una ecuación para cada componente de la expresión usando las reglas anteriores.

Ejemplos:

- Considere la expresión $\text{proc}(f, x) \mid (f + (1, x) \text{ zero?}(x))$.
- Primero se hace una tabla de todas las variables ligadas y aplicaciones de la expresión, y se asigna un tipo a cada una:

Expresión	Variable de Tipo
f	tf
x	tx
$(f + (1, x) \text{ zero?}(x))$	$t1$
• $+(1, x)$	$t2$
• $\text{zero?}(x)$	$t3$
$\text{proc}(f, x) \mid (f + (1, x) \text{ zero?}(x))$	tp

$$\left\{ \begin{array}{l} \text{proc} (? F, ? X) (F + (1, X) \text{ zero?}(X)) \\ t_p = t_F * t_X \rightarrow t_1 \end{array} \right.$$

$$t_1(1, X) = t_2$$

$$\text{int } t_X \ t_X \rightarrow t_2$$

$$\text{int} * \text{int} \rightarrow \text{int}$$

}

$$t_X = \text{int}$$

$$t_2 = \text{int}$$

$$\text{zero?}(X) = t_3$$

$$t_X \rightarrow t_3$$

$$\text{int} \rightarrow \text{bool}$$

$$t_X = \text{int} \checkmark$$

$$t_3 = \text{bool}$$

$$(F + \underline{(1, X)} \ \underline{\text{zero?}(X)}) = t_1$$

$$t_F = t_2 * t_3 \rightarrow t_1$$

$$t_F = \text{int} * \text{bool} \rightarrow t_1$$

Resolver

$$t_p = t_F * t_X \rightarrow t_1$$

$$t_p = (\text{int} * \text{bool} \rightarrow t_1) * \text{int} \rightarrow t_1$$

- Por la regla de los procedimientos, el tipo de toda la expresión tp es $(tf * tx \rightarrow t1)$.
- Por esto, se debe hallar los tipos tf , tx y $t1$.

- Ahora, para cada componente de la expresión, se puede deducir una ecuación de tipo:

Expresión	Ecuación de Tipo
$(f \ + (1, x) \ \text{zero?}(x))$	$tf = (t2 * t3 \rightarrow t1)$
$+ (1, x)$	$(\text{int} * \text{int} \rightarrow \text{int}) =$
	$(\text{int} * tx \rightarrow t2)$
$\text{zero?}(x)$	$(\text{int} \rightarrow \text{bool}) = (tx \rightarrow t3)$

- La primera ecuación muestra que el procedimiento f debe tomar un primer argumento del mismo tipo que $+ (1, x)$ y un segundo argumento del mismo tipo de $\text{zero?}(x)$, y su resultado debe ser del mismo tipo que la aplicación.

$> (3, 4) \Rightarrow \text{int} * \text{int} \rightarrow \text{bool}$

- Las otras ecuaciones son similares: en el lado izquierdo está el tipo del operador, y en el lado derecho el tipo construido de los tipos de los operandos y el tipo de la aplicación.

- Las otras ecuaciones son similares: en el lado izquierdo está el tipo del operador, y en el lado derecho el tipo construido de los tipos de los operandos y el tipo de la aplicación.
- Las tres ecuaciones de tipo:

$$tf = (t2 * t3 \rightarrow t1)$$
$$(int * int \rightarrow int) = (int * tx \rightarrow t2)$$
$$(int \rightarrow bool) = (tx \rightarrow t3)$$

se pueden resolver por inspección y sustitución (proceso denominado *unificación*).

- Se concluye por la segunda ecuación que:

`tx = int`

`t2 = int`

- Sustituyendo éstos valores en la primera y tercera ecuación se tiene:

`tf = (int * t3 -> t1)`

`(int -> bool) = (int -> t3)`

- De la última ecuación se deduce:

`t3 = bool`

- Sustituyendo en la primera ecuación:

`tf = (int * bool -> t1)`

Se han resuelto todas las variables de tipo, excepto `t1` y `tf`:

```
tf = (int * bool -> t1)
```

```
tx = int
```

```
t2 = int
```

```
t3 = bool
```

- Se tiene que el primer argumento del procedimiento, `f`, debe ser un procedimiento de dos argumentos:
 - El primero debe ser un `int` (correspondiente a `t2`).
 - El segundo debe ser un `bool` (correspondiente a `t3`).
- Así mismo, el segundo argumento del procedimiento, `x`, debe ser un `int` (variable de tipo `tx`).

- Luego, se tiene que el tipo de $\text{proc}(f, x) \ (f \rightarrow (1, x) \text{ zero?}(x))$ (representado por la variable de tipo tp) es $((\text{int} * \text{bool} \rightarrow t1) * \text{int} \rightarrow t1)$ para cualquier $t1$.
- El código funcionará para cualquier tipo $t1$. Se dice entonces que la expresión es polimórfica en $t1$.

Ejemplos:

- Considere la expresión del ejemplo anterior pero cambiando el + por cons, es decir `proc(f,x) (f cons(1,x) zero?(x))`.
- Las ecuaciones de tipo serían:

Expresión

`(f cons(1,x) zero?(x))`

`cons(1,x)`

`zero?(x)`

Ecuación de Tipo

`tf = (t2 * t3 -> t1)`

`(int * (list int) -> (list int))`
`= (int * tx -> t2)`

`(int -> bool) = (tx -> t3)`

Considere la expresión de ejemplo anterior pero cambiando el + por cons, es decir $\text{proc}(f, x)$ (f cons(1, x) zero?(x)).

F	t_f
x	t_x
proc(f x) ...	t_p
(f cons(1, x) zero?(x))	t_1
cons(1, x)	t_2
zero?(x)	t_3

$$\text{proc}(f, x) = t_p$$

$$t_p = t_f * t_x \rightarrow t_1$$

$$(f \text{ cons } 1, x) \text{ zero?}(x))$$

$$t_f = t_2 * t_3 \rightarrow t_1$$

$$(\text{cons } 1, x)$$

$$\text{int} * t_x \rightarrow t_2$$

$$\text{int} * \text{list} \rightarrow \text{list}$$

$$t_x = \text{list}$$

$$t_2 = \text{list}$$

$$\text{zero?}(x)$$

$$t_x \rightarrow t_3$$

$$\text{int} \rightarrow \text{bool}$$

$$t_3 = \text{bool}$$

$$t_x = \text{int}$$

Errol de tipos

- De la segunda ecuación se deduce:
`tx = (list int)`
`t2 = (list int)`
- Sustituyendo estos valores en la tercera ecuación, se tiene:
`(int -> bool) = ((list int) -> t3)`
- Pero no existe ningún valor para `t3` que haga igual a esos tipos.
- Para que fueran iguales se debería tener que `int = (list int)`, lo cual es falso.
- De allí que la expresión es rechazada y hay un error de tipos.

- En resumen, la inferencia de tipos (realizada por el procedimiento `check-equal-type!`) toma dos tipos `t1` y `t2`, y “revisa si ellos *pueden ser* el mismo”.
- Si lo son, ajusta el contenido de las variables de tipo para igualarlos.

El procedimiento es el siguiente:

- 1 Primero determina si t_1 y t_2 son el mismo valor en Scheme. Si lo son, tiene éxito y retorna un valor no específico.
- 2 Si t_1 es una variable de tipo, llama al procedimiento `check-tvar-equal-type!` con t_1 y t_2 , pasando `exp` para el reporte de error. De igual forma para t_2 .
- 3 Si t_1 y t_2 son tipos atómicos, determina si ellos tienen el mismo nombre; si no, no pueden ser igualados, y un error es reportado.
- 4 Si t_1 y t_2 son procedimientos de tipo, determina si tiene el mismo número de argumentos. Si lo tienen, se llama recursivamente con cada argumento y el tipo resultado.
- 5 De lo contrario, t_1 y t_2 no pueden ser igualados, por lo que se reporta un error.

El procedimiento `check-equal-type!` estará definido de la siguiente manera:

```
(define check-equal-type!  
  (lambda (t1 t2 exp)  
    (cond  
      ((eqv? t1 t2))  
      ((tvar-type? t1) (check-tvar-equal-type! t1 t2 exp))  
      ((tvar-type? t2) (check-tvar-equal-type! t2 t1 exp))  
      ((and (atomic-type? t1) (atomic-type? t2))  
       (if (not  
            (eqv?  
              (atomic-type->name t1)  
              (atomic-type->name t2)))  
           (raise-type-error t1 t2 exp)))  
      ...  
    )))
```

```
(define check-equal-type!  
  (lambda (t1 t2 exp)  
    (cond  
      ...  
      ((and (proc-type? t1) (proc-type? t2))  
       (let ((arg-types1 (proc-type->arg-types t1))  
             (arg-types2 (proc-type->arg-types t2))  
             (result-type1 (proc-type->result-type t1))  
             (result-type2 (proc-type->result-type t2)))  
         (if (not  
             (= (length arg-types1) (length arg-types2)))  
             (raise-wrong-number-of-arguments t1 t2 exp)  
             (begin  
               (for-each  
                (lambda (t1 t2)  
                  (check-equal-type! t1 t2 exp))  
                arg-types1 arg-types2)  
               (check-equal-type!  
                result-type1 result-type2 exp))))))  
      (else (raise-type-error t1 t2 exp))))
```

Ejemplos:

- Considere la expresión `let f = proc (? x) x in f.`
- Se utilizarán las siguientes variables de tipo:

Expresión	Variable de Tipo
<code>f</code>	<code>tf</code>
<code>x</code>	<code>tx</code>
<code>let f = proc (? x) x in f</code>	<code>t1</code>

- Luego, dado que f corresponde a un procedimiento, su tipo estará determinado por el tipo de sus parámetros formales y de su resultado.
- Por definición el tipo de la expresión `let` corresponde al tipo de su cuerpo (en este caso f).
- Por esta razón se tendrán las siguientes ecuaciones de tipo:
$$tf = (tx \rightarrow tx)$$
$$t1 = tf$$
- Donde el tipo de f y del `let` es $(tx \rightarrow tx)$ para cualquier tipo tx .

Ejemplos:

- Ahora considere la expresión `let f = proc (? x) x in let y = (f 5) in f.`
- Se utilizarán las siguientes variables de tipo:

Expresión	Variable de tipo
<code>f</code>	<code>tf</code>
<code>x</code>	<code>tx</code>
<code>y</code>	<code>ty</code>
<code>(f 5)</code>	<code>t2</code>
<code>let f = proc (? x) x in y = (f 5) in f</code>	<code>t1</code>

- Se tendrán las siguientes ecuaciones de tipo:

`tf = (tx -> tx)`

`tf = (int -> t2)`

`t1 = tf`

- De la segunda ecuación se puede inferir que:

`tx = int`

`t2 = tx`

- Luego el tipo de `f` y del `let` es `(int -> int)`.

Ejemplos:

- Ahora considere la expresión `let f = proc (? x) x in let y = (f 5) in let z = (f true) in f.`
- Se utilizarán las siguientes variables de tipo:

Expresión	Variable de Tipo
<code>f</code>	<code>tf</code>
<code>x</code>	<code>tx</code>
<code>y</code>	<code>ty</code>
<code>(f 5)</code>	<code>t2</code>
<code>(f true)</code>	<code>t3</code>
<code>let f = proc (? x) x in y = (f 5)</code>	
<code>in let z = (f true) in f</code>	<code>t1</code>

- Se tendrán las siguientes ecuaciones de tipo:

```
tf = (tx -> tx)
```

```
tf = (int -> t2)
```

```
tf = (bool -> t3)
```

```
t1 = tf
```

- De la segunda y tercera ecuación se puede inferir que:

```
tx = int = bool
```

- Lo que resulta en un error de tipo, dado que una variable de tipo no puede corresponder a dos tipos diferentes (`int` y `bool`).

Ejemplos:

- Considere la expresión:

```
let
  f = proc(? x, int y) if x then +(y,1) else -(y,1)
in
  let
    g = proc(? m, int n) (m true n)
  in
    let
      h = (g f 5)
    in
      g
```

- Inferir su tipo.

Ejercicio

- Considere la expresión:

```
let
  f = proc(? x, ? y, ? z)
        if (x y) then *(z,2) else z
in
  let
    g = proc(? m)
          if m then true else false
    k = 5
  in
    (f g true k)
```

- Inferir su tipo.

```

let
  f = proc(? x, ? y, ? z)
    if (x, y) then *(z, 2) else z
in
  let
    g = proc(? m)
      if m then true else false
    k = 5
  in
    (f g true k)

```

f	t_f	$(bool \rightarrow bool) \times bool \times int \rightarrow int$
x	t_x	$bool \rightarrow bool$
y	t_y	$bool$
z	t_z	int
g	t_g	$bool \rightarrow bool$
m	t_m	$bool$
k	t_k	int

if m
(f g true k)

t_4 bool
 t_5 int

let
in

if (x y)	t_1	<u>int</u>
(x y)	t_2	<u>bool</u>
*(z 2)	t_3	<u>int</u>

$t_p = t_5$

$t_f = (bool \rightarrow bool) \times bool \times int \rightarrow int$

$t_f = t_x \times t_y \times t_z \rightarrow t_1$

$t_f = t_g \times bool \times t_k \rightarrow t_5$



$t_x = t_g$

$t_y = bool$

$t_z = t_k = int$

$t_1 = t_5 = int$

$t_g = t_m \rightarrow t_4$

$t_g = bool \rightarrow bool$

$t_k = int$

$t_z \times int \rightarrow t_3$

$int \times int \rightarrow int$

$t_z = bool$

$t_3 = t_z$

$t_z = int$

$t_3 = t_z = t_1$

$t_3 = int$

$t_m = bool$

$t_y = bool$

$t_p = int$

o
o

Escriba expresiones en el lenguaje visto en el curso que sean:

1. $(\text{int} * (\text{int} \rightarrow \text{int}) * (\text{int} * \text{int} \rightarrow \text{int})) \rightarrow (\text{int} \rightarrow \text{int})$
2. $(\text{bool} * \text{int} * (\text{int} \rightarrow \text{int})) \rightarrow (\text{int} \rightarrow (\text{bool} \rightarrow \text{int}))$
3. $(\text{int} * \text{int} * (\text{int} \rightarrow \text{bool})) \rightarrow (\text{int} * (\text{int} \rightarrow \text{bool})) \rightarrow \text{int}$
4. $((\text{int} \rightarrow \text{bool}) * (\text{int} \rightarrow \text{int})) \rightarrow ((\text{int} \rightarrow \text{int}) * (\text{int} \rightarrow \text{int}) \rightarrow (\text{int} \rightarrow \text{bool}))$

• y de la expresión de:

```

1 let
2   x = proc(?a, ?b)
3       proc(?c) +(c, *(a, b))
4   y = proc(?e, ?f)
5       if >(((e 3 4) 5), f) then e
6           else
7               proc(?g) *(g, 2)
8   z = 4
9   in
  (y x z)

```

$$(\overbrace{\text{int} * (\text{int} \rightarrow \text{int})}^x * \underbrace{(\text{int} * \text{int} \rightarrow \text{int})}_y) \rightarrow \underbrace{(\text{int} \rightarrow \text{int})}_z$$

let

$$g = \text{proc}(\text{int } x, ?y, (\text{int} * \text{int} \rightarrow \text{int}) z)$$

$$\text{proc}(?x) + (x, z)$$

$g = \text{proc}(\text{int } x) \ x$

$b = \text{proc}(?j, ?i) + (j, i)$

in g

```

let
  x = proc(?a, ?b)
    proc(?c) + (c * (a, b)) |
  y = proc(?e, ?f)
    if >((e 3 4) 5), f) then e
    else
      proc(?g) *(g, 2)
    in
      z = 4
      (y x z)

```

$x \quad t_x (int \times int) \rightarrow (int \rightarrow int)$
 $9 \quad t_9 int$
 $6 \quad t_6 int$
 $y \quad t_y ((e \times int) \times int \rightarrow (int \times int))$
 $e \quad t_e int$
 $f \quad t_f int \times int \rightarrow (int \rightarrow int)$
 $if \quad t_0 (int \times int) \rightarrow (int \rightarrow int)$
 $let \quad t_p ((int \times int) \rightarrow (int \rightarrow int))$

$9 \quad t_9 int$
 $z \quad t_z int$
 $proc(?c) \quad t_1 (int \rightarrow int)$
 $+ (c, -) \quad t_2 int$
 $\times (a, b) \quad t_3 int$
 $> (e, f) \quad t_4 bool$
 $((e \ 3 \ 4) \ 5) \quad t_5 int$
 $(e \ 3 \ 4) \quad t_6 int \rightarrow int$
 $proc(?g) \quad t_7 int \rightarrow int$
 $\times (g, 2) \quad t_8 int$
 $(y \times z) \quad t_9 (int \times int) \rightarrow (int \rightarrow int)$

$t_p = t_9$
 $t_1 = t_c \rightarrow t_2$
 $t_c \times t_3 \rightarrow t_2$
 $int \times int \rightarrow int$
 $t_9 \times t_6 \rightarrow t_3$
 $int \times int \rightarrow int$

$t_x = t_9 \times t_6 \rightarrow t_1$
 $(int \times int) \rightarrow (int \rightarrow int)$
 $IF \begin{cases} t_4 = bool \\ t_e = t_7 = t_0 \end{cases}$
 $t_z = int$

$t_y = t_e \times t_f \rightarrow t_0$

$>$

$t_5 \times int \rightarrow t_4$
 $int \times int \rightarrow bool$

$(\underline{(e \ 3 \ 4)} \ 5) \quad t_5$
 $t_6 \times int \rightarrow t_5$

$\underline{(e \ 3 \ 4)} \quad t_6$
 $t_e = int \times int \rightarrow t_6 \quad t_6 = (int \rightarrow int)$
 $t_e = int \times int \rightarrow (int \rightarrow int)$

$\ast (\underline{g}, \underline{z}) \quad t_g \times int \rightarrow t_8 \quad t_8 = int$
 $int \times int \rightarrow int \quad t_g = int$
 $t_7 = t_g \rightarrow t_8$
 $t_7 = int \rightarrow int$

$\underline{(y \times z)} \quad t_9$
 $t_y = t_x \times t_z \rightarrow t_9$
 $t_y = t_e \times t_f \rightarrow t_0$
 $t_x = t_e \quad t_f = t_z$

$t_y = ((int \times int) \rightarrow (int \rightarrow int)) \times int \rightarrow (int \times int) \rightarrow (int \rightarrow int)$

Fundamentos
de lenguajes
de
programación

Carlos Andrés
Delgado S.
Carlos Alberto
Ramírez

Inferencia de
tipos

?

- Conceptos Fundamentales de la Programación Orientada a Objetos.