Fundamentos de lenguajes de programación

Robinson Duque, M.Eng, Ph.D

Universidad del Valle

robinson.duque@correounivalle.edu.co

Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Este documento es una adaptación del material original de los profesores
Carlos Andres Delgado y Carlos Alberto Ramírez

Overview

1 Inferencia de tipos

Inferencia de tipos

- Algunos lenguajes de programación dejan que el compilador averigue 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 ?.

Inferencia de tipos

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.

Inferencia de tipos

Se agregan las siguientes reglas de producción a la gramática:

⟨tipo-exp-opcional⟩ ::= ⟨tipo-exp⟩

```
a-type-exp (texp)
```

⟨tipo-exp-opcional⟩ ::= ?

```
no-type-exp ()
```

Inferencia de tipos

Para usar las expresiones de tipo opcionales en expresiones normales, se deben cambiar las reglas de producción para `proc-exp` y `letrec-exp`:

⟨expresión⟩ ::= `proc` ({⟨tipo-exp-opcional⟩ ⟨identificador⟩}$^{*(,)}$) ⟨expresión⟩

```
proc-exp (optional-arg-texps ids body)
```

⟨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)
```

Inferencia de tipos

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)
```

Inferencia de tipos

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)
```

Inferencia de tipos

- 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.

Inferencia de tipos

- 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*.

Inferencia de tipos

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?)))
```

Inferencia de tipos

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 '())))))
```

Inferencia de tipos

- 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.

Inferencia de tipos

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)))))
```

Inferencia de tipos

- 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:

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

Inferencia de tipos

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

```
(type-of-expression ≪rator≫ tenv) =
((type-of-expression ≪rand_1≫ tenv)
 *...*
 (type-of-expression ≪rand_n≫ tenv)
 ->
 (type-of-expression ≪(rator rand_1 ... rand_n)≫ 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.

Inferencia de tipos

Por último, cuando se escribe una expresión `proc` $(x_1 \ldots x_n)$ *exp* evaluada en el ambiente de tipos *tenv*, se tiene:

```
(type-of-expression ≪proc (x1 ... xn) exp≫ tenv) =
((type-of-expression x1 tenv_body)
 *...*
 (type-of-expression xn tenv_body)
 ->
 (type-of-expression ≪exp≫ tenv_body))
```

donde $tenv_{body}$ es el ambiente de tipo en el cual el cuerpo *exp* será tipado.

Inferencia de tipos

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.

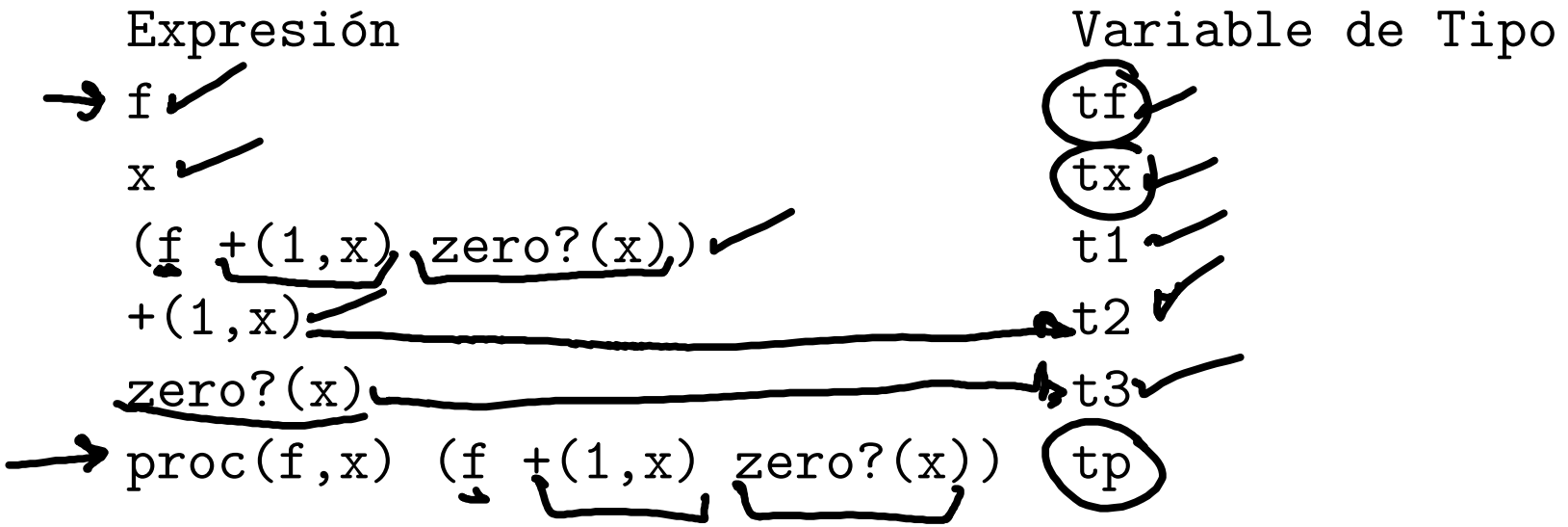
```
(type-of-program (scan&parse "
let
 g = proc(int r) *(2,r)
 k = proc (int q, bool e) if e then q else *(-1,q)
 x = proc(? y, int w, (int*bool->int) z) y
in
 (x g 5 (k 5 true) )
"))
```

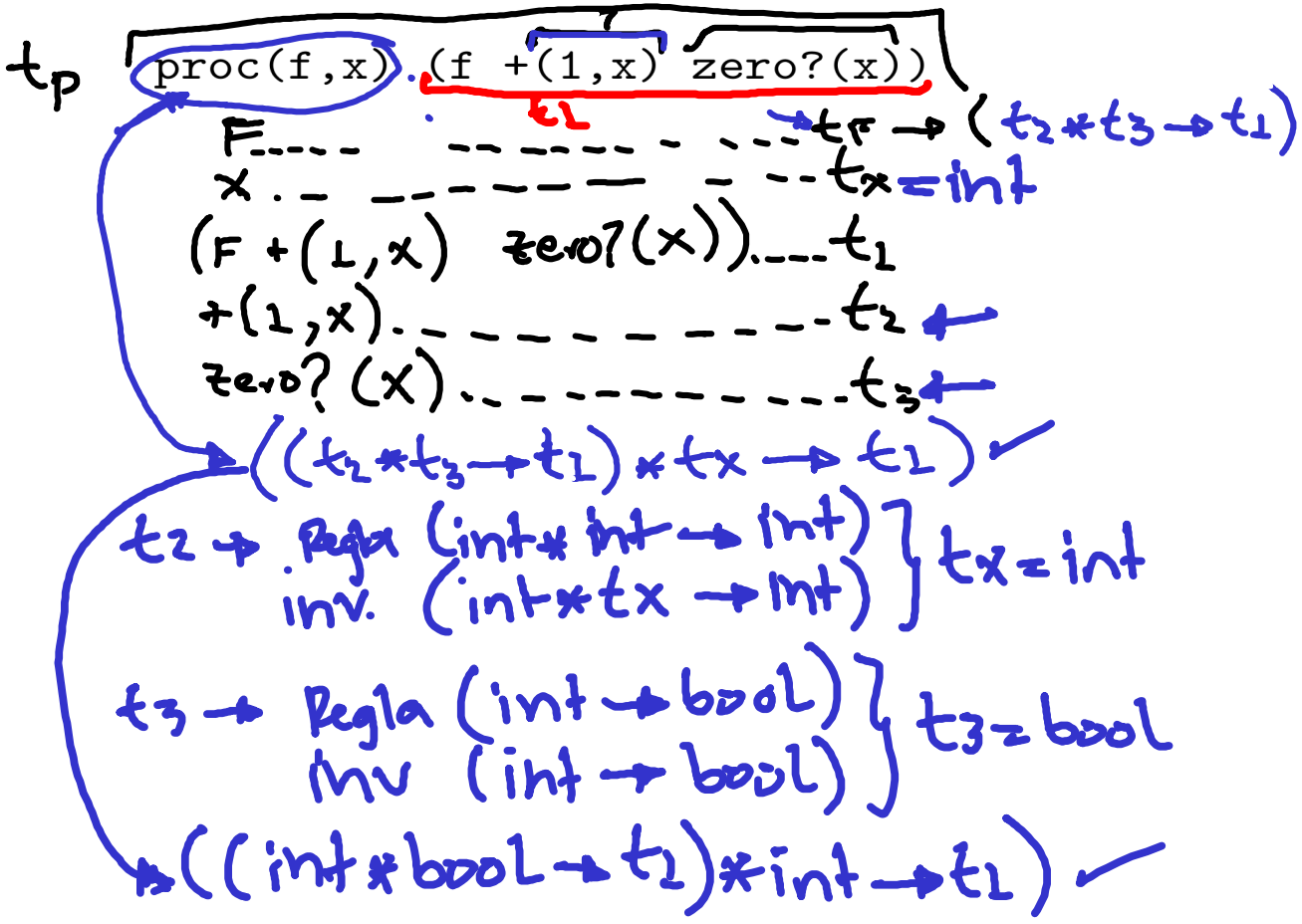
```
(type-of-program (scan&parse "
let
 g = proc(int r) *(2,r)
 k = proc (int q, bool e) if e then q else *(-1,q)
 x = proc(? y, int w, (int*bool->int) z) y
in
 (x g 5 k)
"))
```

Inferencia de tipos
Ejemplos

Ejemplos:

- Considere la expresión `proc(f,x) (f +(1,x) 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:




t_p
proc(f,x) (f +(1,x) zero?(x))
t_1
F ..... $t_F \to (t_2 * t_3 \to t_1)$
x ..... $t_x = int$
(F +(1,x) zero?(x)) ..... t_1
+(1,x) ..... t_2
zero?(x) ..... t_3
$((t_2 * t_3 \to t_1) * t_x \to t_1)$
$t_2 \to$ Regla (int * int → int)
inv. (int * t_x → int) } $t_x = int$
$t_3 \to$ Regla (int → bool)
inv (int → bool) } $t_3 = bool$
((int * bool → t_1) * int → t_1)

Inferencia de tipos
Ejemplos

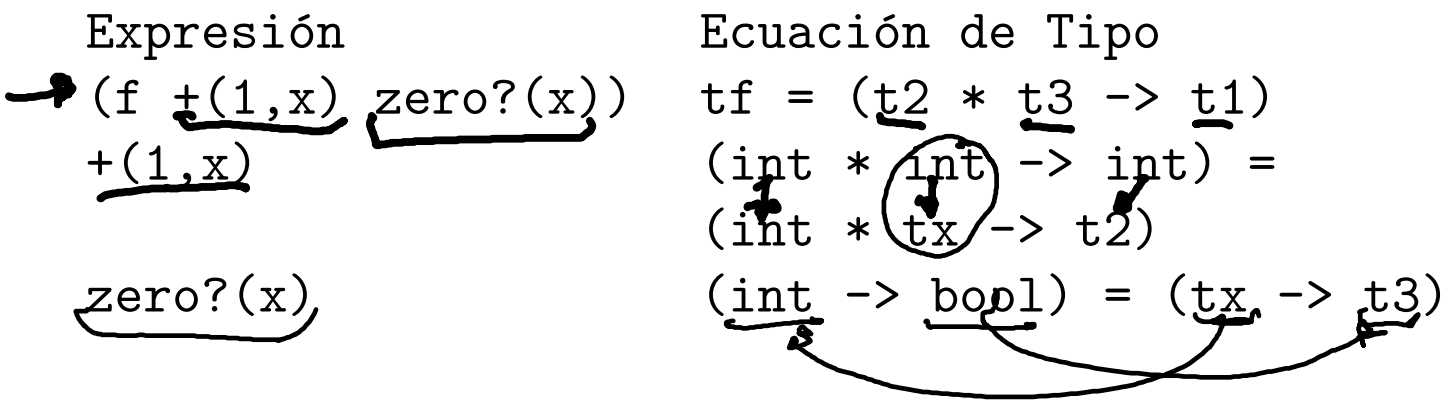
```
proc(f,x) (f +(1,x) zero?(x))
```

(tf * tx → t1)

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

Inferencia de tipos
Ejemplos

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



- 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 `zero?(x)`, y su resultado debe ser del mismo tipo que la aplicación.

Inferencia de tipos
Ejemplos

- 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 -> t1)
(int * int -> int) = (int * tx -> t2)
(int -> bool) = (tx -> t3)
```

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

Inferencia de tipos
Ejemplos

- 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 -> t1)
(int * int -> int) = (int * tx -> t2)
(int -> bool) = (tx -> t3)
```

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

Inferencia de tipos
Ejemplos

- 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)`

Inferencia de tipos
Ejemplos

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

```
tf = (int * bool -> t1)
tx = int
t2 = int
t3 = bool
```

Inferencia de tipos
Ejemplos

- 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`).

Inferencia de tipos
Ejemplos

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

Inferencia de tipos
Ejemplos

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:

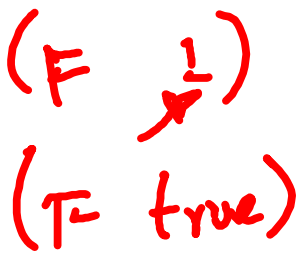
Expresión	Ecuación de Tipo
`(f cons(1,x) zero?(x))`	`tf = (t2 * t3 -> t1)`
`cons(1,x)`	`(int * (list int) -> (list int))` `= (int * tx -> t2)`
`zero?(x)`	`(int -> bool) = (tx -> t3)`

Inferencia de tipos
Ejemplos

- 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.

Inferencia 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.



Inferencia de tipos

El procedimiento es el siguiente:

1. Primero determina si `t1` y `t2` son el mismo valor en Scheme. Si lo son, tiene éxito y retorna un valor no específico.
2. Si `t1` es una variable de tipo, llama al procedimiento `check-tvar-equal-type!` con `t1` y `t2`, pasando `exp` para el reporte de error. De igual forma para `t2`.
3. Si `t1` y `t2` son tipos atómicos, determina si ellos tienen el mismo nombre; si no, no pueden ser igualados,y un error es reportado.
4. Si `t1` y `t2` 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, `t1` y `t2` no pueden ser igualados, por lo que se reporta un error.

Inferencia de tipos

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)))
      ...
      )))
```

Inferencia de tipos

```
(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)))))
```

Inferencia de tipos
Ejemplos

Ejemplos:

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

Expresión	Variable de Tipo
`f`	`tf`
`x`	`tx`
`let f = proc (? x) x in f`	`t1`

$t_f = (t_x \rightarrow t_x)$

$t_1 = (t_x \rightarrow t_x)$

(type-to-external-form (type-of-program (scan&parse "let f=proc(? x) x in f")))

Inferencia de tipos
Ejemplos

- 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 razon se tendrán las siguientes ecuaciones de tipo:

```
tf = (tx -> tx)
t1 = tf
```

- Donde el tipo de `f` y del `let` es `(tx -> tx)` para cualquier tipo `tx`.

Inferencia de tipos
Ejemplos

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
`f`	`tf`
`x`	`tx`
`y`	`ty`
`(f 5)`	`t2`
`let f = proc (? x) x in y = (f 5) in f`	`t1`

- 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
`f`	`tf`
`x`	`tx`
`y`	`ty`
`(f 5)`	`t2`
`let f = proc (? x) x in y = (f 5) in f`	`t1`

$t_F = (t_x \to t_x)$

$t_y =$ regla $(t_x \to t_x)$, inv $(int \to int)$ ⟹ $t_x = int$, $t_y = int$

$t_F = (int \to int)$

$t_2 = int$

$t_1 = t_F \to (int \to int)$

Inferencia de tipos
Ejemplos

- 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)`.

Inferencia de tipos
Ejemplos

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
`f`	`tf`
`x`	`tx`
`y`	`ty`
`(f 5)`	`t2`
`(f true)`	`t3`
`let f = proc (? x) x in y = (f 5) in let z = (f true) in f`	`t1`

- 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
`f`	`tf`
`x`	`tx`
`y`	`ty`
`(f 5)`	`t2`
`(f true)`	`t3`
`let f = proc (? x) x in y = (f 5) in let z = (f true) in f`	`t1`

$t_f = (t_x \to t_x)$

$t_y =$ Regla $(t_x \to t_x)$ } $t_x = int$
eval $(int \to int)$ } $t_y = int$

$t_f = (int \to int)$

$t_z =$ Regla $(int \to int)$ } error $int \neq bool$
inv $(bool \to bool)$ }

(type-to-external-form (type-of-program (scan&parse "let f=proc(? x) x in let y= (f 5) in let z = (f true) in f")))

Inferencia de tipos
Ejemplos

- Se tendrán las siguientes ecuaciones de tipo:

```
tf = (tx -> tx)
tf = (int -> t2)
tf = (bool -> t3)
t1 = tf
```

- De la segunday 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`).

Inferencia de tipos
Ejemplos

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.

- 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.

$t_f = (t_x * int \to int)$

$t_x = bool$

$t_f = (bool * int \to int)$

$t_y = int$

$t_2 = $ Regla $(int * int \to int)$, inv $(int * int \to int)$ $\}\ t_2 = int$

$t_3 = $ Regla $(int * int \to int)$, inv $(int * int \to int)$ $\}\ t_3 = int$

$t_2 = t_3$ ✓ OK

$t_1 = int$

$t_m = (bool * int \to t_4)$ ✓

$t_g = ((bool * int \to t_4) * . int \to t_4)$

$t_1 = $ if ...

$\begin{cases} t_2 = +(y,1) \\ t_3 = -(y,1) \end{cases}$

$t_4 = (m\ true\ n)$ ✓

$t_5 = (g\ f\ 5)$

$t_5 = $ Regla $((bool * int \to t_4) * . int \to t_4)$

inv $((bool * int \to int) * int \to t_4)$

$t_m = t_f$ entonces

$t_4 = int$

$t_h = int$

$\begin{cases} t_m = (bool * int \to t_4) \\ t_f = (bool * int \to int) \end{cases}$

$t_g = ((bool * int \to t_4) * . int \to t_4)$

$((bool * int \to int) * int \to int)$

```
( type-to-external-form (type-of-program (scan&parse "
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")))
```

Inferencia de tipos

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.

Preguntas

?

Próxima sesión

- Conceptos Fundamentales de la Programación Orientada a Objetos.