Universidad Nacional de Rosario
Facultad de Ciencias Exactas,
Ingeniería y Agrimensura
Departamento de Ciencias de la
Computación

Análisis de Lenguajes de Programación

Trabajo Práctico 3

Cerruti Lautaro, Poma Lucas

Octubre de 2022

Ejercicio 1

La función infer retorna un tipo Either String Type ya que de esta forma puede devolver errores con un Left. Los errores posibles son

- matchError: Este error ocurre cuando el tipo inferido es distinto al tipo especificado
- notfunError: Este error ocurre cuando el tipo de t no es una función cuando se esta evaluando una aplicación t u.
- notfoundError: Este error ocurre cuando el tipo de una variable libre no se encuentra en el entorno.

El operador $\gg=$ lo que hace es que si el valor v es Left e, devuelve Left e propagando el error e, pero si el operador es un Right v1 le aplica f a v1 y devuelve este resultado.

Ejercicio 4

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x : E \in \{x : E\}}{\{x : E\} \vdash x : E}\, T-VAR}{\vdash \lambda x : E.x : E \to E}\, T-ABS}{\vdash (\lambda x : E.x)\ as\ E \to E : E \to E}\, T-ASCRIBE \qquad \dfrac{z : E \to E \in \{z : E \to E\}}{\{z : E \to E\} \vdash z : E \to E}\, T-VAR}{\vdash\ let\ z = ((\lambda x : E.x)\ as\ E \to E)\ in\ z : E \to E}\, T-LET}{\vdash\ (let\ z = ((\lambda x : E.x)\ as\ E \to E)\ in\ z)\ as\ E \to E : E \to E}\, T-ASCRIBE$$

Ejercicio 6

Agregamos las siguientes cuatro reglas:

$$\frac{t_1 \to t_1'}{(t_1, t_2) \to (t_1', t_2)}\, E-TUPLE1$$

$$\frac{t_2 \to t_2'}{(v, t_2) \to (v, t_2')}\, E-TUPLE2$$

$$\frac{}{fst(v_1, v_2) \to v_1}\, E-FST$$

$$\frac{}{snd(v_1, v_2) \to v_2}\, E-SND$$

Ejercicio 8

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash unit : Unit}\,T-UNIT}{\vdash unit\ as\ Unit : Unit}\,T-ASCRIBE \qquad \dfrac{\dfrac{\dfrac{x : (E,E) \in \{x : (E,E)\}}{\{x : (E,E)\} \vdash x : (E,E)}\,T-VAR}{\{x : (E,E)\} \vdash snd\ x : E}\,T-SND}{\vdash \lambda x : (E,E).snd\ x : (E,E) \to E}\,T-ABS}{\vdash (unit\ as\ Unit, \lambda x : (E,E).snd\ x) : (Unit, (E,E) \to E)}\,T-PAIR}{\vdash fst(unit\ as\ Unit, \lambda x : (E,E).snd\ x) : Unit}\,T-FST
$$

Ejercicio 10

Vamos a hacer la transformación de la funcion de Ackermann a Lambda tipado partiendo desde la implementación en Haskell.

```
Ack : Nat -> Nat -> Nat
Ack 0 n = n + 1
Ack m 0 = Ack (m - 1) 1
Ack m n = Ack (m - 1) (Ack m (n - 1))
```

Abstraemos n en los 3 casos de la función y en los 2 casos donde m es distinto de 0 lo agrupamos llamando a una función auxiliar $auxAck$.

```
Ack 0 = \n:Nat. suc n
Ack m = \n:Nat. auxAck n

auxAck 0 = Ack (m - 1) 1
auxAck n = Ack (m - 1) (Ack m (n - 1))
```

Parametrizamos en $auxAck$, la llamada $Ack(m-1)$, pasándolo como argumento desde la función Ack

```
Ack 0 = \n:Nat. suc n
Ack m = \n:Nat. auxAck (Ack (m - 1)) n

auxAck f 0 = f 1
auxAck f n = f (Ack m (n - 1))
```

Como podemos ver en la funcion Ack, `Ack m = \n:Nat. auxAck (Ack (m - 1)) n`, entonces reemplazamos la aparición de `Ack m` en $auxAck$ por `\n:Nat. auxAck (Ack (m - 1)) n`

```
Ack 0 = \n:Nat. suc n
Ack m = \n:Nat. aux (Ack (m - 1)) n

auxAck f 0 = f 1
auxAck f n = f ((\n:Nat. auxAck (Ack (m - 1)) n) (n - 1))
```

Como podemos ver, en este nuevo termino tenemos una aparición de $Ack(m-1)$ que es el valor que pasamos por parámetro como f, por lo que lo reemplazamos.

```
Ack 0 = \n:Nat. suc n
Ack m = \n:Nat. aux (Ack (m - 1)) n

```

```
auxAck f 0 = f 1
auxAck f n = f ((\n:Nat. auxAck f n) (n - 1))
```

En el termino $(\backslash n : Nat.auxAckfn)(n-1)$ podemos aplicar la regla $E - APPABS$

```
Ack 0 = \n:Nat. suc n
Ack m = \n:Nat. auxAck (Ack (m - 1)) n

auxAck f 0 = f 1
auxAck f n = f (auxAck f (n - 1))
```

Luego de este último paso podemos llevar la implementación *Ack* a una expresión equivalente haciendo uso del operador *R*.

```
Ack = \m:Nat. R (\n:Nat. suc n) (\x:Nat->Nat.\i:Nat. auxAck x) m

auxAck f 0 = f 1
auxAck f n = f (auxAck f (n - 1))
```

Ahora podemos realizar lo mismo con *auxAck*

```
Ack = \m:Nat. R (\n:Nat. suc n) (\x:Nat->Nat.\i:Nat. auxAck x) m

auxAck f = \n:Nat. R (f (suc 0)) (\x:Nat.\ii:Nat. f x) n
```

Y por ultimo abstraemos *f* en *auxAck*, Y la funcion de Ackermann queda definida de la siguiente manera:

```
Ack = \m:Nat. R (\n:Nat. suc n) (\x:Nat->Nat.\i:Nat. auxAck x) m

auxAck =\f:Nat->Nat. \n:Nat. R (f (suc 0)) (\x:Nat.\ii:Nat. f x) n
```

Bibliografia

Higher-Order Recursion Abstraction