Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Ejercicios

Compiladores

Autores:
Arroyo, Joaquín
Bolzan, Francisco

10 de diciembre de 2024

Índice

0.1. Práctica 1

Ejercicio 1. Implementar en PFC_0 las siguientes funciones: resta, exponente y factorial. (Tener en cuenta que **if** es **ifz**)

$resta = fun(n : \mathbb{N}).fun(m : \mathbb{N}).subt\ n\ m$

$$exp = fix(exp : \mathbb{N} \to \mathbb{N} \to \mathbb{N})(m : \mathbb{N}).\\ fun(m : \mathbb{N}).\\ \textbf{if } n \textbf{ then } 1\\ \textbf{else } (\textbf{if } subt\ n\ 1 \textbf{ then } m \textbf{ else } mul\ m\ (exp\ m\ (subt\ n\ 1)))$$

$$fact = fix(fact : \mathbb{N} \to \mathbb{N})(n : \mathbb{N}).\\ \textbf{if } n \textbf{ then } 1\\ \textbf{else } mul\ m\ (fact\ (subt\ n\ 1))$$

Donde *mul* es una función definida en el apunte de teoría.

Ejercicio 2. Definir representaciones para:

1. Booleanos. Esto requiere representar constructores true, false, y un ifthenelse.
   a) ¿Es posible representar ifthenelse simplemente como un término (es decir, una función ternaria), como en la codificación de Church?
   b) ¿Puede arreglarse la codificación, tal vez modificando como se codifican las ramas, para que ifthenelse sea simplemente un término?
   c) Considere el ejercicio anterior en un lenguaje con evaluación Lazy.
2. Pares. Esto requiere representar un constructor de pares pair, y proyecciones proj1 y proj2.

1. (a) Tomando $true = 0$ y $false = 1$ tenemos la siguiente representación de *ifthenelse*:
$ifthenelse = fun(b : \mathbb{N}).fun(t_1 : \mathbb{N}).fun(t_2 : \mathbb{N}).\textbf{if } b \textbf{ then } t_1 \textbf{ else } t_2$

El problema con esta representación es que, al PFC_0 ser *call by value*, los parámetros t_1 y t_2 se están ejecutando antes de ver cuál de los dos hay que ejecutar dependiendo el parámetro b.

1. (b) Para resolver el problema del punto **a)** podemos definir a *ifthenelse* de la siguiente manera:
$ifthenelse = fun(b : \mathbb{N}).fun(t_1 : \mathbb{N} \to \mathbb{N}).fun(t_2 : \mathbb{N} \to \mathbb{N}).\textbf{if } b \textbf{ then } t_1\,0 \textbf{ else } t_2\,0$

Tomando esta representación, al pasar a t_1 y t_2 como funciones, estas no van a ser previamente ejecutadas, si no que se va a ejecutar solamente uno de los dos caminos dependiendo el parámetro b. El problema con esto es que los términos $t1$ y $t2$ ahora son funciones.

1. (c) Considerando una evaluación *Lazy*, ya con la definición del punto **a)** nos es suficiente.

2.
$proj1 = fun(p : \mathbb{N} \to \mathbb{N}).p\,0$
$proj2 = fun(p : \mathbb{N} \to \mathbb{N}).p\,1$
$pair = fun(p_0 : \mathbb{N}).fun(p_1 : \mathbb{N}).fun(p : \mathbb{N}).\textbf{if } p \textbf{ then } p_1 \textbf{ else } p_2$

Ejercicio 3. Implementar el algoritmo de Euclides para calcular el máximo común divisor. Este algoritmo está dado por las siguientes ecuaciones:

$$\begin{aligned} gcd(m, 0) &= m \\ gcd(0, n) &= n \\ gcd(m, n) &= gcd(m - n, n) && m \geq n \\ gcd(m, n) &= gcd(m, n - m) && m < n \end{aligned}$$

$geq = fun(m : \mathbb{N}).fun(n : \mathbb{N}).\textbf{if } (subt\ n\ m) \textbf{ then } 0 \textbf{ else } 1$

$gcd = fix(gcd : \mathbb{N} \to \mathbb{N} \to \mathbb{N})(m : \mathbb{N}).fun(n : \mathbb{N}).$

if n **then** m
else if m **then** n
else $\mathit{ifthenelse}\,(\mathit{geq}\,m\,n)\,(\mathit{gcd}\,(\mathit{subt}\,m\,n)\,n)\,(\mathit{gcd}\,m\,(\mathit{subt}\,n\,m))$

Ejercicio 4. El sistema T de G¨odel visto en ALP tiene un operador de recursión R sobre los naturales. Definirlo en PCF_0 usando recursión general. Recordar que el operador $R_\tau\ z\ s$ define una función $f : \mathbb{N} \to \tau$ tal que:

$$f\,0 = z$$
$$f(\mathit{succ}\,n) = s\,(f\ n)\,n$$

$R_{\mathbb{N}} = \mathit{fun}(z : \mathbb{N}).\mathit{fun}(s : \mathbb{N} \to \mathbb{N} \to \mathbb{N}).$
$(\mathit{fix}(f : \mathbb{N} \to \mathbb{N})(n : \mathbb{N}).$
if n **then** z **else** $s\,(f\,(\mathit{subt}\,n\,1))(\mathit{subt}\,n\,1))0$

Ejercicio 5. Probar que PCF_0 es Turing-completo. Dado que por el ejercicio anterior sabemos que se pueden definir todas las funciones primitivas recursivas, solo falta probar que se puede definir un operador de minimización. Es decir un operador que, para una función $f : \mathbb{N} \to \mathbb{N}$, devuelve el mínimo $z : \mathbb{N}$ tal que $f\ z \Downarrow 0$.

$\mathit{min} = \mathit{fun}(f : \mathbb{N} \to \mathbb{N}).$
$(\mathit{fix}(r : \mathbb{N} \to \mathbb{N}).(z : \mathbb{N}).$
if $f\ z$ **then** z **else** $r\,(\mathit{succ}\,z))\,0$
Revisar como hacer para pasarle el caso base de $z = 0$ a *fix*.

Ejercicio 6. ¿Es la β-reducción una regla ecuacional válida en PFC_0? Es decir: ¿son los términos $(\mathit{fun}(x : \tau).t_1)t_2$ y $[t_2/x]t_1$ equivalentes?

Los términos no son equivalentes debido a que PFC_0 es *call by value*, es decir, en el primer término, t_2 va a ser evaluado a un valor v y luego se va a realizar el debido reemplazo sobre x en t_1 con dicho v. En cambio, en el segundo término, lo primero que se hace es reemplazar el/los término/s x por t_2 dentro de t_1, y luego, eventualmente, se evaluará a t_1, y dentro de él a t_2.

0.2. Practica 2

Ejercicio 1. ¿Cuál es el resultado de evaluar 2-2-2? ¿Por qué?

Puede dar 2 o 0 dependiendo de como este implementado la asociación del operador - (es decir si el parser asocia a izquierda o derecha).

Ejercicio 2. Luego de ejecutar una linea como:

$FD4 > \mathit{let}\ v\ =\ \mathit{print}\,'\mathit{Prueba}'\,(1+1)$

¿Qué cambios ocurren en el estado global? ¿Qué pasa cuando luego usamos v en otro término? Por ejemplo, ¿qué ocurre al evaluar $v + v$?

En el estado global se asocia a la variable v el valor 2 el cual es el resultado de la expresion, es decir, $\Gamma = \Gamma \cup \{v \mapsto 2\}$.
Cuando usamos v en otro termino, solamente se reemplaza v por el valor 2.
En el caso de evaluar la expresion $(v + v)$ nos devuelve 4.

$$\dfrac{\dfrac{(v \mapsto 2) \in \Gamma}{\Gamma \vdash v \Downarrow^{\emptyset} 2}\ \textbf{E-VAR} \quad \dfrac{(v \mapsto 2) \in \Gamma}{\Gamma \vdash v \Downarrow^{\emptyset} 2}\ \textbf{E-VAR}}{v + v \Downarrow^{\emptyset} 2 + 2}\ \textbf{E-ADD}$$

En el caso de la expresión $FD4 > \mathit{let}\,f\ =\ \mathit{fun}(x : \mathit{Nat}) \to \mathit{print}\,'\mathit{Prueba}\,2'\,(x + x)$, se guarda en la variable f la función sin aplicar, por lo que al ejecutar por ejemplo $f\,1$, se obtiene el valor 2, pero a su vez el mensaje $'\mathit{Prueba}\,22'$

En el caso de la expresión $f\,1 + f\,1$, esto nos devuelve el valor 4, pero antes imprime el mensaje $'\mathit{Prueba}\,22\,\mathit{Prueba}\,22'$.

$$\dfrac{\dfrac{(f \mapsto fun(x : Nat) \to print\,'Prueba\,2'\,(x + x)) \in \Gamma}{\Gamma \vdash f \Downarrow^{\emptyset} fun(x : Nat) \to print\,'Prueba\,2'\,(x + x)}\ \textbf{E-VAR} \quad \dfrac{}{1 \Downarrow^{\emptyset} 1}\ \textbf{E-NAT} \quad \dfrac{\dfrac{\dfrac{}{1 \Downarrow^{\emptyset} 1}\ \textbf{E-NAT} \quad \dfrac{}{1 \Downarrow^{\emptyset} 1}\ \textbf{E-NAT}}{(1+1) \Downarrow^{\emptyset} 2}\ \textbf{E-ADD}}{(print\,'Prueba\,2'\,(1+1)) \Downarrow^{\,'Prueba\,22'} 2}\ \textbf{E-PRINT}}{f\,1 \Downarrow^{\,'Prueba\,22'} 2}\ \textbf{E-APP}$$

$$\dfrac{\dfrac{}{f\,1 \Downarrow^{\,'Prueba\,22'} 2}\ \textbf{E-APP} \quad \dfrac{}{f\,1 \Downarrow^{\,'Prueba\,22'} 2}\ \textbf{E-APP}}{f\,1 + f\,1 \Downarrow^{\,'Prueba\,22\,Prueba\,22'} 4}\ \textbf{E-ADD}$$

Ejercicio 3. ¿Cuál es la salida al evaluar el siguiente fragmento? ¿Por qué?
$FD4 > print\,'Hola\,'\,(print\,'mundo!'\,2)$

Se va a evaluar primero la expresión interna, por lo que el programa imprime $'mundo!2'$ y luego la expresión externa, por lo que se imprime $'Hola\,2'$.

$$\dfrac{\dfrac{\dfrac{}{2 \Downarrow^{\emptyset} 2}\ \textbf{E-NAT}}{print\,'mundo!'\,2 \Downarrow^{mundo!\,2} 2}\ \textbf{E-PRINT}}{print\,'Hola\,'\,(print\,'mundo!'\,2) \Downarrow^{\,'mundo!2\,Hola\,2'} 2}\ \textbf{E-PRINT}$$

0.3. Práctica 3

Ejercicio 1. ¿A qué términos fully named corresponden los siguientes términos con índices de de Bruijn?

a) $\lambda\, 0 = \lambda\, x.x$
b) $\lambda\, 0\, \lambda\, 1\ 0 = \lambda\, x.x\, \lambda\, y.x\ y$
c) $\lambda\, \lambda\, \lambda\, 2\ 0\ 1 = \lambda\, x.\, \lambda\, y.\, \lambda\, z.x\ z\ y$
d) $\lambda(\lambda\, 0)(\lambda\, 1) = \lambda\, x.(\lambda\, y.y)(\lambda\, z.x)$

Ejercicio 2. Expresar los siguientes términos fully named con índices de de Bruijn.

a) $\lambda\, x.x(\lambda\, y.y) = \lambda\, 0\, \lambda\, 0$
b) $\lambda\, f.\, \lambda\, x.f\ x = \lambda\, \lambda\, 1\ 0$
c) $\lambda\, x.x\ x = \lambda 0\ 0$
d) $\lambda\, x.\, \lambda\, y.(\lambda\, z.x\ z)y = \lambda\, \lambda(\lambda\, 2\ 0)0$

Ejercicio 3. ¿Cuál es el resultado de las siguientes aplicaciones de open y close?

a) $open\ x\ 0 = x$
b) $open\ x\ (\lambda\, \lambda\, 2\ 0\ 1) = \lambda\, \lambda\, x\ 0\ 1$
c) $open\ x\ (\lambda\, 0) = \lambda\, 0$
d) $open\ x\ (0\, \lambda\, 0\ 1\, \lambda\, 1\ 2) = x\, \lambda\, 0\ x\, \lambda\, 1\ x$
e) $close\ x\ 0 = 0$
f) $close\ x\ (x(\lambda\, 0\ x)) = 0\, \lambda\, 0\ 1$
g) $close\ x\ (\lambda\, \lambda\, 0\ 1) = \lambda\, \lambda\, 0\ 1$
h) $close\ x\ \ \lambda(close\ y(\lambda\, y\ x)) = close\ x(\lambda\, \lambda\, 2\ x) = \lambda\, \lambda\, 2\ 3$

0.4. Práctica 4

Ejercicio 1. Traduzca los siguientes términos de FD4 a Core FD4 según las reglas de arriba:

```
a) let x : Nat = 2 in x + 1
b) fun (x:Nat) -> x
c) let id (x:Nat) : Nat = x in id 10
d) let app5 (f : Nat -> Nat) : Nat = f 5 in app5 (fun (x : Nat) -> x + x)
e) fun (x:Nat) (y:Nat) -> ifz x then y else 1
```

Ejercicio 2. Traduzca las siguientes declaraciones de FD4 a Core FD4 según las reglas de arriba:

```
a) let rec doble (x:Nat) : Nat = ifz x then 0 else 2 + (doble (x - 1))
b) let rec ack (m:Nat) (n:Nat) : Nat =
    ifz m
    then n + 1
    else (ifz n
        then ack (m - 1) 1
        else ack (m - 1) (ack m (n - 1)))
```

0.5. Práctica 6. (Bytecompile)

Ejercicio 5. Comparar performance de *ack* 3 11 entre la Machinna y un programa de C nativo.
Tiempos en programa nativo de C:

```
ack 3 11 = 16381

real    0m0.544s
user    0m0.544s
sys     0m0.000s
```

Tiempos en programa en la Machinna:

```
ack 3 11 = 16381

real    4m32.001s
user    3m40.972s
sys     1m48.327s
```