



Universidad Nacional Autónoma de México  
Facultad de Ciencias  
Lenguajes de Programación  
SEMANTIC 7

Comas Castañeda Mauricio Santiago	320215988
Robledo Ramírez Isaac	320140655
Sánchez Pérez Ricardo	315153327



- Dada la siguiente expresión en MiniLisp.

```
(let (sum (lambda (n) (if0 n 0 (+ n (sum (- n 1))))))  
sum 5))
```

- Ejecutarla y explicar el resultado.  
Al ejecutar esta expresión nos arroja como resultado un error de variable libre, siendo la variable que queda libre sum.
- Modificarla usando el combinador de punto fijo Y, volver a ejecutarla y explicar el resultado.  
La version modificada con el combinador y basandonos en las notas quedaria como sigue:

```
let ( Y ( lambda ( f ) (( lambda ( x ) ( f ( x x ) ) ) ( lambda ( x ) ( f ( x x ) ) ) ) ) )  
  ( let ( sum ( Y ( lambda ( sum ) ( lambda ( n ) ( if0 n 0 (+ n ( sum (- n 1) ) ) ) ) ) ) ) ( sum 5) )
```

Donde n es 5 y al hacer la ejecución tomamos la definición de nuestro combinador Y, la aplicamos a nuestra funcion de suma, con lo que esto quedaria:

```
(lambda(Y)(lambda(sum))5)
```

donde evaluamos  $Y$  y sustituimos en  $f$  por nuestro parametro  $\text{lambda}(\text{sum})$  y se hacen las betareducciones en nuestra evaluacion de la siguiente manera:

```
((lambda(x) (lambda(sum) (xx))) (lambda(x) (lambda(sum) (xx)))) 5)
beta reduce: ((lambda(sum)) (lambda(x) (lambda(sum) (xx))) (lambda(x) (lambda(sum) (xx)))) 5)
```

Ahora tenemos en nuestra beta reduccion la definicion de nuestro combinador  $Y$  por lo que esto se puede ver como:

```
(lambda (sum) (lambda (Y) (lambda (sum)) 5)
beta reduce: (lambda (n) (if0 n 0 (+ n ((Y)(sum)) (-5 1)) 5)).
eval: (+ 5 (((Y)(sum)) (- 5 1))).
eval: (+ 5 (((Y)(sum)) (4)))
```

Tenemos ahora nuestra  $Y$  sum con 4, entonces repitiendo los pasos anteriores con  $Y$  sum  $n-1$ , y simulando una recursion solamente con lets dentro de nuestro minilisp, nuestro resultado final sera 15. que sera la suma de la evaluacion final que tenemos al llegar a 0 que seria  $0+1+2+3+4+5=15$ .

2. Evaluar la siguiente expresión en Racket, explicar su resultado y dar la continuación asociada a evaluar utilizando la notación  $\lambda \uparrow$ .

```
(define c #f)
(+ 1 (+ 2 (+ 3 (+ let/cc k (set! c k) 4) 5))))
(c 10)
```

La expresión imprime 15 y 21

1. Se inicializa una variable  $c$  con el valor  $\#f$ .
2. Se evalúa la suma  $(+1(+2(+3$  evaluando el primer operando y el segundo.
3. Al llegar al operando  $(+let/cc(set!ck)$ ,  $let/cc$  guarda la continuación  $k$  de la evaluación en la variable previamente definida como  $c$ .



4. Se termina de evaluar la suma con los operandos 4 (puesto que no se llamó la continuación) y 5, siendo  $1 + 2 + 3 + 4 + 5 = 15$  y se imprime el resultado.
5. Se llama la continuación  $c$  con 10, lo que sustituye a 4, con lo que ahora la suma se evalúa como  $1 + 2 + 3 + 10 + 5 = 21$  y se imprime el resultado.

La continuación asociada con  $\lambda \uparrow$  sería:

$$(\lambda \uparrow (u)(+1(+2(+3(+u5)))))$$

3. ■ Definir la función recursiva `ocurrenciasElementos` que toma como argumentos dos listas y devuelve una lista de parejas, en donde cada pareja contiene en su parte izquierda un elemento de la segunda lista y en su parte derecha el número de veces que aparece dicho elemento en la primera lista.

(Le llamaremos `ocurrenciasElementosRc`).

```
1 {- ocurrenciasElementosRC: Funcion RECURSIVA que toma como argumentos dos
2    listas y devuelve una lista de parejas, en donde cada pareja contiene en su
3    parte izquierda un elemento de la segunda lista y en su parte derecha el
4    numero de veces que aparece dicho elemento en la primera lista. -}
5 ocurrenciasElementosRc :: forall a. (Eq a, Show a) => [a] -> [a] -> [(a,Int)]
6 ocurrenciasElementosRc xs [] = []
7 ocurrenciasElementosRc xs (y:ys) = (y, length (filter (== y) xs)) : ocurrenciasElementosRc xs ys
```

- Mostrar los registros de activación generados por la función definida en el ejercicio anterior con la llamada `ocurrenciasElementos [1,2,3] [1,2]`.



Función	Cuerpo	Args	Resultado
ocurrenciasElementos xs []	[]	[1,2,3] []	[]
(==) x y	False	2 3	False
(==) x y	x == x	2 2	True
(==) x y	False	2 1	False
filter p xs	[x   x ← xs, p x]	(==2) [1,2,3]	[2]
length l	length []	[]	1
length l	length (x:xs)	[1]	1
length l	length (x:xs)	(filter(==y)xs)	1
(:) a [a] / cons a [a]	a:[a]	length(filter(==y)xs):ocurrenciasElementosRc xs ys	[(2,1)]
ocurrenciasElementos xs (y:ys)	(y, length(filter(==y)xs)):ocurrenciasElementosRc xs ys	[1,2,3] [2]	[(2,1)]
(==) x y	False	1 3	False
(==) x y	False	1 2	False
(==) x y	x == x	1 1	True
filter p xs	[x   x ← xs, p x]	(==1) [1,2,3]	[1]
length l	length []	[]	1
length l	length (x:xs)	[2]	1
length l	length (x:xs)	(filter(==y)xs)	1
(:) a [a] / cons a [a]	a:[a]	length(filter(==y)xs):ocurrenciasElementosRc xs ys	[(1,1),(2,1)]
ocurrenciasElementos xs (y:ys)	(y, length(filter(==y)xs)):ocurrenciasElementosRc xs ys	[1,2,3] [1,2]	[(1,1),(2,1)]

- Optimizar la función definida usando recursión de cola. Deben transformar todas las funciones auxiliares que utilicen.

```

1 {- ocurrenciasElementos: Funcion que toma como argumentos dos listas y
2    devuelve una lista de parejas, en donde cada pareja contiene en su parte
3    izquierda un elemento de la segunda lista y en su parte derecha el numero
4    de veces que aparece dicho elemento en la primera lista. -}
5 ocurrenciasElementos :: forall a. (Eq a, Show a) => [a] -> [a] -> [(a,Int)]
6 ocurrenciasElementos xs ys = ocurrenciasElementosTR xs (reverseTR ys []) []
7 -- Uso de reverse para que la salida sea identica.
8   where

```

```
9      {- ocurrenciasElementosTR: Funcion basada en RECURSION DE COLA que toma como
10         argumentos dos listas y devuelve una lista de parejas, en donde cada pareja
11         contiene en su parte izquierda un elemento de la segunda lista y en su
12         parte derecha el numero de veces que aparece dicho elemento en la primera
13         lista.
14         Debe iniciarse con ocurrenciasElementosTR _ _ [] para su correcto
15         funcionamiento. -}
16      ocurrenciasElementosTR :: forall a. (Eq a, Show a) => [a] -> [a] -> [(a,Int)] -> [(a,Int)]
17      ocurrenciasElementosTR _ [] acc = acc
18      ocurrenciasElementosTR xs (y:ys) acc = ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)
19
20  {- occurencesTR: Funcion basada en recursion de cola que toma como argumentos
21     un elemento y una lista y devuelve la cantidad de veces que aparece el
22     elemento en la lista.
23     Debe iniciarse con occurencesTR _ _ 0, para su correcto funcionamiento. -}
24  occurencesTR :: Eq a => a -> [a] -> Int -> Int
25  occurencesTR _ [] acc = acc
26  occurencesTR x (y:ys) acc
27      | x == y = occurencesTR x ys (acc + 1)
28      | otherwise = occurencesTR x ys acc
29
30  {- reverseTR: Funcion basada en recursion de cola que toma como
31     argumento una lista y devuelve la lista invertida.
32     Debe iniciarse con reverseTR _ [] para su correcto funcionamiento. -}
33  reverseTR :: [a] -> [a] -> [a]
34  reverseTR [] acc = acc
35  reverseTR (x:xs) acc = reverseTR xs (x:acc)
```

Usamos `ocurrenciasElementos` para pasar de una entrada de dos argumentos, a una con 3. De igual manera, para procurar una lista idéntica al ejemplo, usamos `reverseaTR`, igual adaptada al uso de recursión de cola; para que la obtención de elementos pueda obtenerse de atrás para adelante y las duplas aparezcan en el orden correcto c:.

- Mostrar los registros de activación generados por la versión de cola con la misma llamada.



Función	Cuerpo	Args	Resultado
reverseTR ys acc	reverseTR xs (x:acc)	[1,2] []	
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] _	

↓

Función	Cuerpo	Args	Resultado
reverseTR ys acc	reverseTR xs (x:acc)	[2] [1]	
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] _	_

↓

Función	Cuerpo	Args	Resultado
reverseTR ys acc	reverseTR xs (x:acc)	[] [2,1]	[2,1]
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] _	_

↓

Función	Cuerpo	Args	Resultado
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	_

↓

Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	2 [1,2,3] 0	_
(:) a l	a:l	(2, occurencesTR 2 [1,2,3] 0) []	_
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [2,1] []	_
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	_



↓

Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	2 [2,3] 0	—
(:) a l	a:l	(2, occurencesTR 2 [1,2,3] 0) []	—
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [2,1] []	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	2 [3] 1	—
(:) a l	a:l	(2, occurencesTR 2 [1,2,3] 0) []	—
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [2,1] []	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	2 [] 1	1
(:) a l	a:l	(2, occurencesTR 2 [1,2,3] 0) []	—
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [2,1] []	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓





Función	Cuerpo	Args	Resultado
$(:)$ a l	a:l	(2, 1) []	(2,1):[] = [(2,1)]
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [2,1] []	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [1] [(2,1)]	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	1 [1,2,3] 0	—
$(:)$ a l	a:l	(1, occurencesTR 1 [1,2,3] 0) [(2,1)]	—
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [1] [(2,1)]	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	1 [2,3] 1	—
$(:)$ a l	a:l	(1, occurencesTR 1 [1,2,3] 0) [(2,1)]	—
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [1] [(2,1)]	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓



Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	1 [3] 1	—
(:) a l	a:l	(1, occurencesTR 1 [1,2,3] 0) [(2,1)]	—
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [1] [(2,1)]	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	1 [] 1	—
(:) a l	a:l	(1, occurencesTR 1 [1,2,3] 0) [(2,1)]	—
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [1] [(2,1)]	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
occurencesTR y xs acc	x == y = occurencesTR x ys (acc + 1)   otherwise = occurencesTR x ys acc	1 [] 1	1
(:) a l	a:l	(1, occurencesTR 1 [1,2,3] 0) [(2,1)]	—
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [1] [(2,1)]	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
(:) a l	a:l	(1,1) [(2,1)]	(1,1):[(2,1)] = [(1,1),(2,1)]
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR xs ys ((y, occurencesTR y xs 0):acc)	[1,2,3] [1] [(2,1)]	—
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—



↓

Función	Cuerpo	Args	Resultado
ocurrenciasElementosTR xs (y:ys) acc	ocurrenciasElementosTR _ [] acc	[1,2,3] [] [(1,1),(2,1)]	[(1,1),(2,1)]
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	—

↓

Función	Cuerpo	Args	Resultado
ocurrenciasElementos xs ys	ocurrenciasElementosTR xs (reverseTR ys []) []	[1,2,3] [2,1]	[(1,1),(2,1)]

\*Espero se entienda porque de verdad me costó hacer esto :P. Momento L<sup>A</sup>T<sub>E</sub>X.