

Universidad Nacional Autónoma De México
Facultad de Ciencias

SEMANTAL 7

Lenguajes de programacion

Integrantes:

Emiliano Luna Campos: 320292084



Jose Luis Flores Gutierrez: 320060087



Oscar Tapia Sanchez: 320266076



1. Ejercicio 1

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

→ Error variable above "sum" Ydef λf. (λx. f (xx)) (λx. f (xx))

sum := λ(n): if0 n then 0
      else (+ n (sum (n-1)))

(Y sum) 3 = def [ (λf. (λx. f (xx)) (λx. f (xx)) sum ] 3
→ 0 (λx. sum (xx)) (λx. sum (xx)) 3
→ 0 (sum [(λx. sum (xx)) (λx. sum (xx))]) 3 } (Y sum)

sum =def λ(f): λ(n): if0 n then 0
      else (+ n (f (n-1)))
```

```
(λn: if0 n then 0
  else (+ n (Y sum) (n-1)) 3

→ 0 (λn. ((λsum. λn. if0 n 0 (+ n (sum (- n 1))))) (xx)) (λx. ((λsum. λn. if0 n 0 (+ n (sum (- n 1))))) (xx))) (xx)

Ahora puede invocarse a si misma sum

→ 0 (λn. if0 n 0 (+ n (sum (- n 1)) 3

Primero, se verifica la condición if0 n 0. Como n es 3, no es 0, por lo que ejecutamos
el segundo caso: (+ n (sum (- n 1)))
```

```
(sum 2) = (λn. if0 n 0 (+ n (sum (- n 1))) 2

De nuevo, n no es 0, así que se convierte en: 2 + (sum 1)

(sum 1) = (λn. if0 n 0 (+ n (sum (- n 1))) 1
Otra vez, n no es 0 así que se convierte en: 1 + sum(0)
(sum 0) = (λn. if0 n 0 (+ n (sum (- n 1))) 0

sum(0) = 0
sum(1) = 1 + 0 = 1
sum(2) = 2 + 1 = 3
sum(3) = 3 + 3 = 6
```

2. Ejercicio 2

2. λ define $c \# f$

$$\lambda (+1 (+2 (+3 (+ \text{let/cc } K (\text{set! } c\ K) 4) 5))))$$

$$\lambda (c\ 10)$$

$$(+1 (+2 (+3 (+ \text{let/cc } K (\text{set! } c\ K) 4) 5))))$$

(+ valor de c 5)

(+ 3 resultado) $\rightarrow (+3 (+4 5)) = (+3 9)$

(+ 2 resultado) $\rightarrow (+2 12)$

(+ 1 resultado) $\rightarrow (+1 14)$

La segunda parte es una invocación a $c = 10$

$$(+3 (+ \text{valor } 5))$$

$$(+3 (+10 5)) \rightarrow (+3 15) = 18$$

$$(+1 (+2 18)) \rightarrow (+1 20) = 21$$

$$\lambda \uparrow_v (+1 (+2 (+3 (+ v 5))))$$

3. Ejercicio 3

3. a)

ocurrenciasElementos $\therefore \text{Eg } a \Rightarrow [a] \rightarrow [a] \rightarrow [(a, \text{Int})]$

ocurrenciasElementos $[\] = [\]$

ocurrenciasElementos $xs\ (y:ys) = (y, \text{repetidoaux } y\ xs) : \text{ocurrenciasElementos } xs\ ys$

repetidoaux $\therefore \text{Eg } a \Rightarrow [a] \rightarrow a \rightarrow \text{Int}$

repetidoaux $[\] = 0$

repetidoaux $(x:xs)\ y$

$\quad | x == y = 1 + \text{repetidoaux } xs\ y$

$\quad | \text{otherwise} = \text{repetidoaux } xs\ y$

b) ocurrenciasElementos $[1,2,3]$ $[1,2]$
 L_1 L_2

1- Agregamos el 1 de L_2 y llamamos a repetidoAux para que vea cuantas veces se repite en L_1

repetido aux 1 $[1,2,3]$ primero se compara $1==1$ y es un True #T, as que agrega 1.

Luego se compara 1 $[2,3]$ $1==2$ es #f y continúa

Al final solo se compara con el 3, $1==3$ es #f y termina

Agregamos el 2 de L_2 y llamamos otra vez a repetidoAux para que vea cuantas veces se repite en L_1

Primero llama repetidoaux 2 $[1,2,3]$ $2==1$ es false #f y continúa, luego otra repetidoaux 2 $[2,3]$ y $2==2$ es True añade o le suma 1 y luego prueba 2 $[3]$ $2==3$ es false termina.

Llama a ocurrenciasElementos y como ya utilizamos todos los elementos de L_2 entonces ocurrenciasElementos $[1,2,3]$ $[]$

La salida sería $[(1,1), (2,1)]$

c) repetido aux :: $Eg a \Rightarrow [a] \rightarrow a \rightarrow Int \rightarrow Int$
 repetido aux $[]$ = acc = acc
 repetido aux $(x:xs)$ = acc
 $1 \text{ if } x == x = \text{repetido aux } xs \text{ if } (acc+1)$
 $1 \text{ otherwise} = \text{repetido aux } xs \text{ if } acc$

repetido Ocu :: $Eg a \Rightarrow [a] \rightarrow [a] \rightarrow [a, Int] \rightarrow [a, Int]$
 repetido Ocu $[]$ acc = reverse acc
 repetido Ocu $xs (y:ys)$ acc = repetidoOcu xs ys ((y repetidoaux xs y 0) : acc)

ocurrenciasElementos :: $Eg a \Rightarrow [a] \rightarrow [a] \rightarrow [a, Int]$
 ocurrenciasElementos xs ys = ocurrenciasElementos xs ys $[]$

d) Es evaluado ocurrenciasElementos $[1,2,3]$ $[1,2]$
 L_1 L_2

Manda a llamar a repetidoOcu $[1,2,3]$ $[1,2]$ $[]$

Despues esa manda a llamar a repetidoAux $[1,2,3]$ 1 0
 \uparrow elem L_2

Compara $1==1$ es True y manda a llamar a repetido aux $[2,3]$ 1 1.

Luego:

Compara $1==2$ es False y manda a llamar a repetido aux $[3]$ 1 1.

Despues

Compara $1==3$ es False y retorna 1

La función repetidoOcu se llama a si misma y tiene $[(1,1)]$

Luego: Mandamos a llamar a repetido aux $[1,2,3]$ 2 0

Luego: Mandamos a llamar a `repetido aux [1,2,3] 2 0`
Compara `1 == 1` es `False`, así que llama otra vez `repetido aux [2,3] 2 0`
Compara `2 == 2` es `True`, llama otra vez `repetido aux [3] 2 1`.
Compara `2 == 3` es `False` al final retorna el ace que es `1`.
Y se llama `repetido Ocu` a si misma con `[(2,1), (1,1)]` pero para eso era
el reverse devolviendonos al final `[(1,1), (2,1)]`