7524 - Teoría de la programación TP Individual

Karina Alaya

Padron 75840

Ejercicio 1) - Procesamiento de listas Length Take Drop Append Member
<u>Position</u> <u>Ejercicio 2) - Referencias externas</u>
Ejercicio 3) - Ejemplo de ejecución Programa 1 Programa 2 Programa 3 Programa 4 Programa 5 Programa 6
<u>Ejercicio 4) - Case</u>
Ejercicio 5) - Recursividad 1. Traducción al lenguaje Kernel 2. "Tail Recursive" 3. Ejecución en la máquina abstracta Implementación básica Implementación Tail Recursive
Ejercicio 6) - Alto orden con listas FoldL FoldR Map Filter
Ejercicio 7) - Hilos 7.1 WaitSome 7.2 Maquina abstracta
<u>Ejercicio 8) - Evaluación perezosa</u> <u>8.1 Maquina Abstracta</u>

Lenguaje Kernel
Ejecución
8.2 Reverse

<u>Ejercicio 9) - Mensajes</u>

Ejercicio 10) - Servidor de filtros

Ejercicio 11) - Celdas de memoria

Ejercicio 1) - Procesamiento de listas

1. Length

a. Código Oz

```
% Length function
declare fun {Length Xs}
          case Xs of nil then 0
        [] H|T then 1 + {Length T}
        end
    end
```

b. Ejemplos de ejecución

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                                  X
File Edit Options Buffers Tools Oz Help
% Length function
declare fun {Length Xs}
           case Xs of nil then 0
           [] H|T then 1 + {Length T}
        end
% invocation examples Length
{Show 'Length examples'}
{Show {Length [1 2 10]}}
{Show {Length [1 2 10 'test']}}
{Show {Length ['a' 'b' 'c']}}
{Show {Length ['a' 'b' 'c' 4 5 6 7 8 9 10]}}
1\**- Oz
                      All L8
'Length examples'
4
3
10
1\**- *Oz Emulator*
                      Bot L9
                                 (Comint:run)
```

2. Take

a. Código Oz

```
% Take function
declare fun {Take Xs N}
    if N == 0 then nil
    else
        case Xs of nil then nil
        [] H|T then H|{Take T N-1}
        end
    end
end
```

```
X
Oz Programming Interface (emacs@KARINA-VAIO)
File Edit Options Buffers Tools Oz Help
                            X 1 1 1 1 1 1
% Take function
declare fun {Take Xs N}
           if N == 0 then nil
              case Xs of nil then nil
             [] H|T then H|{Take T N-1}
              end
           end
        end
% invocation examples Take
{Show 'Take examples'}
{Show {Take [1 2 10 15] 2}}
{Show {Take [1 2 10 15] 8}}
{Show {Take [a b c d] 3}}
{Show {Take [a b c d] 4}}
{Show {Take [a b c d] 1}}
{Show {Take [a b c d] 0}}
1\**- Oz
                      All L16
                                  (Oz)
'Take examples'
[1 2]
[1 2 10 15]
[a b c]
[abcd]
[a]
nil
1\**- *Oz Emulator*
                      All L8
                                 (Comint:run)
Beginning of buffer
```

3. Drop

a. Código Oz

```
% Drop function
declare fun {Drop Xs N}
    if N == 0 then Xs
    else
        case Xs of nil then nil
     [] H|T then {Drop T N-1}
        end
    end
    end
end
```

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                            X
File Edit Options Buffers Tools Oz Help
% Drop function
declare fun {Drop Xs N}
           if N == 0 then Xs
           else
              case Xs of nil then nil
              [] H|T then {Drop T N-1}
              end
           end
        end
% invocation examples Drop
{Show 'Drop examples'}
{Show {Drop [1 2 10 15] 2}}
{Show {Drop [1 2 10 15] 8}}
{Show {Drop [a b c d] 4}}
{Show {Drop [a b c d] 1}}
{Show {Drop [a b c d] 0}}
1\**- Oz
                       All L10
                                  (Oz)
'Drop examples'
[10 15]
nil
nil
[b c d]
[abcd]
1\**- *Oz Emulator*
                                   (Comint:run)
                      All L7
```

4. Append

a. Código Oz

```
% Append function
declare fun {Append Xs Ys}
    if Ys == nil then Xs
    else
        case Xs of nil then Ys
        [] H|T then H|{Append T Ys}
        end
    end
    end
end
```

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                             X
File Edit Options Buffers Tools Oz Help
% Append function
declare fun {Append Xs Ys}
           if Ys == nil then Xs
           else
              case Xs of nil then Ys
              [] H|T then H|{Append T Ys}
              end
           end
        end
% invocation examples Append
{Show 'Append examples'}
{Show {Append [1 2 10 15] [a b c d]}}
{Show {Append [1 2 10 15] nil}}
{Show {Append nil [a b]}}
{Show {Append [1 2 10 15] [a]}}
{Show {Append [1] [a b]}}
1\**- Oz
                       All L8
                                   (Oz)
'Append examples'
[1 2 10 15 a b c d]
[1 2 10 15]
[a b]
[1 2 10 15 a]
[1 a b]
1\**- *Oz Emulator*
                       All L7
                                   (Comint:run)
```

5. Member

a. Código Oz

```
% Member function
declare fun {Member Xs Y}
    case Xs of nil then false
    [] H|T then
        if H == Y then true
        else
        {Member T Y}
        end
        end
        end
        end
        end
```

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                           X
File Edit Options Buffers Tools Oz Help
                           * •
% Member function
declare fun {Member Xs Y}
           case Xs of nil then false
           [] H|T then
              if H == Y then true
              else
                 {Member T Y}
              end
           end
        end
s invocation examples Member
{Show 'Member examples'}
{Show {Member [1 2 10 15] 2}}
{Show {Member [1 2 10 15] 1}}
{Show {Member [1 2 10 15] 10}}
{Show {Member [1 2 10 a 15] 15}}
{Show {Member [1 2 10 a 15] a}}
{Show {Member [1 2 10 15] b}}
{Show {Member [1 2 10 15] 3}}
{Show {Member [1 2 10 15] 5}}
{Show {Member [1 2 10 15] nil}}
{Show {Member nil a}}
1\**- Oz
                    All L11
                                  (Oz)
'Member examples'
true
true
true
true
true
false
false
false
false
false
1\**- *Oz Emulator* All L11 (Comint:run)
```

6. Position

a. Código Oz

```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                             X
File Edit Options Buffers Tools Oz Help
% Position function
declare fun {Position Xs Y}
           case Xs of nil then 0
            [] H|T then
              if H == Y then 0
              else 1 + {Position T Y}
              end
           end
        end
% invocation examples Position
{Show 'Position examples'}
{Show {Position [1 2 10 15] 10}}
{Show {Position [1 2 10 a 15] 15}}
{Show {Position [1 2 10 a 15] a}}
{Show {Position [1 2 10 15] 1}}
{Show {Position [1 2 10 15] 3}}
1\**- Oz
                       All L5
                                   (Oz)
'Position examples'
3
0
4
<
1\**- *Oz Emulator*
                        All L7
                                    (Comint:run)
```

Ejercicio 2) - Referencias externas

1. $proc \{P X Y\} local Z in \{Q Z U\} end end$

Referencias externas: Q, U

2. $proc \{P X Y\} local Z in \{Q Z Y\} end end$

Referencias externas: Q

3. $proc \{P X Y\} local Z in \{P Z Y\} end end$

Referencias externas: P

Ejercicio 3) - Ejemplo de ejecución

Programa 1

Estado inicial

Stack	Store	E
local B in if B then Skip else skip end end	-	

=> Ejecución de declaración de variable

Stack	Store	Е
if B then skip else	b1	B-> b1
skip end		

=> Ejecución de condicional. Como E(B) no está determinado, el programa se suspende, a la espera que b1 tome algun valor.

Programa 2

Estado inicial

Stack	Store	Е
local B in B = false if B then skip else skip end	-	-
end		

=> Ejecución de declaración de variable y composición

Stack	Store	Е
B = false if B then skip else skip end	b1	B-> b1

=> Ejecución de binding de variables

Stack	Store	E
if B then skip else skip end	b1 = false	B-> b1

=> Ejecución de condicional, como B = false entonces se agrega en el stack la sentencia dentro del else

Stack	Store	Е
skip	b1 = false	B-> b1

=> Ejecución del skip (St6), no hay cambios en el store y se quita la sentencia.

Stack	Store	Е
-	b1 = false	B-> b1

No hay nada mas en el stack, entonces el programa finaliza.

Programa 3

Estado inicial

Stack	Store	Е
(local X Z A B P in proc {P X Y}	-	-

=> Ejecución de declaración de variables

Stack	Store	E'
(proc {P X Y} Y = X+Z end, E') (Z=7, E') (X=4, E') ({P X A}, E') ({P A B}, E')	x1 z1 a1 b1 p1	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

=> Ejecución de asignación de procedure value

Stack	Store	E'
(Z=7, E') (X=4, E') ({P X A}, E') ({P A B}, E')	x1 z1 a1 b1 p1 = (proc {\$ X Y} Y = X+Z end, CE = {Z->z1})	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

=> Ejecución de binding de variable

Stack	Store	E'
(X=4, E') ({P X A}, E') ({P A B}, E')	x1 z1 = 7 a1 b1 p1 = (proc {\$ X Y} Y = X+Z end, CE = {Z->z1})	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

=> Ejecución de binding de variable

Stack	Store	E'
({P X A}, E') ({P A B}, E')	x1 = 4 z1 = 7 a1 b1 p1 = (proc {\$ X Y}	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1

=> Aplicación de procedure value

Stack	Store	E'
(Y = X+Z, E") ({P A B}, E')	x1 = 4 z1 = 7 a1 b1 p1 = (proc {\$ X Y} Y = X+Z	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1
	end, CE = {Z->z1})	E" = CE + {X->E'(X); Y-> E'(A)}
		Z->z1 X->x1 Y->a1

=> Asignación de suma X + Z

Stack	Store	E'
({P A B}, E')	x1 = 4	X -> x1

z1 = 7 a1 = 11 b1 p1 = (proc {\$ X Y} Y = X+Z end, CE = {Z->z1})	Z-> z1 A -> a1 B -> b1 P-> p1
end, CE – {Z->21})	E" = CE + {X->E'(X); Y-> E'(A)}
	Z->z1 X->x1 Y->a1

=> Aplicación de procedure value

Stack	Store	E'
(Y = X+Z, E''')	x1 = 4 z1 = 7 a1 = 11 b1 p1 = (proc {\$ X Y} Y = X+Z	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1
	end, CE = {Z->z1})	E" = CE + {X->E'(X); Y-> E'(A)}
		Z->z1 X->x1 Y->a1
		E'"= CE + {X-> E'(A); Y-> E'(B)}
		Z->z1 X->a1 Y->b1

=> Asignación de suma X + Z

Stack	Store	E'
-	x1 = 4 z1 = 7 a1 = 11 b1 = 18 p1 = (proc {\$ X Y}	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1 E'' = CE + {X->E'(X); Y-> E'(A)}

	Z->z1 X->x1 Y->a1
	E'''= CE + {X-> E'(A); Y-> E'(B)}
	Z->z1 X->a1 Y->b1

Fin del programa

Programa 4

Estado inicial

Stack	Store	Е
local X Z A B P in proc {P X Y}	-	-
X=4 {P X A} {P A B}		
end end		

=> Ejecución de declaración de variables

Stack	Store	E
(proc {P X Y}	x1	X - > x1
Y = X+Z	z1	Z -> z1
end, E)	a1	A -> a1
(Z=10, E)	b1	B -> b1
(local Z in	p1	P -> p1
Z = 2		
X=4		
{P X A}		
{P A B}		
end, E)		

=> Declaración de procedure value

Stack	Store	Е
(Z=10, E)	x1	X - > x1
(local Z in	z1	Z -> z1
Z = 2	a1	A -> a1
X=4	b1	B -> b1
{P X A}	p1 = proc { \$ X Y }	P -> p1
{P A B}	Y = X+Z	
end, E)	end, CE = {Z -> z1}	

=> Ejecución de binding de variable

Stack	Store	E
(local Z in	x1	X -> x1
Z = 2 X=4	z1 = 10 a1	Z -> z1 A -> a1
{P X A} {P A B}	b1 p1 = proc {\$ X Y}	B -> b1 P -> p1
end, E)	Y = X + Z	Γ -> μ
	end, $CE = \{Z -> z1\}$	

=> Declaración de variable Z en un local

Stack	Store	Е
(Z = 2, E') (X=4, E') ({P X A}, E') ({P A B}, E')	x1 z1 = 10 a1 b1 p1 = proc {\$ X Y}	X - > x1 Z -> z1 A -> a1 B -> b1 P -> p1 E' = E + {Z -> nueva variable}, como Z existe en E, se reemplaza en este contexto. X - > x1 A -> a1 B -> b1 P -> p1 Z -> z2

=> Asignación Z = 2

Stack	Store	E
(X=4, E')	x1	X - > x1

({P X A}, E') ({P A B}, E')	z1 = 10 a1 b1 p1 = proc {\$ X Y} Y = X+Z end, CE = {Z -> z1} z2 = 2	Z -> z1 A -> a1 B -> b1 P -> p1 E' X -> x1 A -> a1 B -> b1 P -> p1 Z -> z2
--------------------------------	--	---

=> Asignación X = 4

Stack	Store	Е
({P X A}, E') ({P A B}, E')	x1 = 4 z1 = 10 a1 b1 p1 = proc {\$ X Y}	X - > x1 Z -> z1 A -> a1 B -> b1 P -> p1 E' X - > x1 A -> a1 B -> b1 P -> p1 Z -> z2

=> Aplicación procedure value

Stack	Store	Е
(Y = X+Z, E")	x1 = 4	Sin cambios
({P A B}, E')	Y = X+Z end, CE = {Z -> z1} z2 = 2	E'
		X - > x1 A -> a1 B -> b1 P -> p1 Z -> z2
		E" = CE + {X->E'(X); Y->E'(A)}
		Z-> z1

	X->x1 Y->a1
	1-741

=> **Ejecución Y = X + Z (en E")**

Stack	Store	Е
({P A B}, E')	x1 = 4 z1 = 10	Sin cambios
	a1 = 14	E'
	b1 p1 = proc {\$ X Y} Y = X+Z end, CE = {Z -> z1} z2 = 2	X - > x1 A -> a1 B -> b1 P -> p1 Z -> z2
		E"
		Z-> z1 X->x1 Y->a1

=> Aplicación procedure value

Stack	Store	E
(Y = X+Z, E''')		Sin cambios E' X - > x1 A -> a1 B -> b1
		P -> p1 Z -> z2
		E"
		Sin cambios
		E'''= CE + {X -> E'(A); Y-> E'(B)}
		Z-> z1 X-> a1 Y-> b1

=> Ejecución de Y =X+Z en E'"

Stack	Store	E
-	x1 = 4	Sin cambios
	z1 = 10 a1 = 14	E'
	b1 = 24 p1 = proc {\$ X Y} Y = X+Z end, CE = {Z -> z1} z2 = 2	X - > x1 A -> a1 B -> b1 P -> p1 Z -> z2 E" Sin cambios
		Z-> z1 X-> a1 Y-> b1

Fin del programa

Programa 5

Estado inicial

Stack	Store	E
local X Y Z P Q in	-	-
X=6		
Y=4		
proc {P A B}		
proc {B U V}		
local F in		
F=A+1		
V=U+F		
end		
end		
end		
{P X Q}		
{Q Y Z}		
end		

=> Declaración de variables

Stack	Store	E
(X=6, E)	x1	X -> x1
(Y=4, E)	y1	Y -> y1
(proc {P A B}	z1	Z -> z1
proc {B U V}	p1	P -> p1
local F in	q1	Q -> q1
F=A+1		
V=U+F		
end		
end		
end , E)		
((P X Q), E)		
((Q Y Z) , E)		

=> Asignación de variable X = 6

Stack	Store	E
(Y=4, E)	x1 = 6	X -> x1
(proc {P A B}	y1	Y -> y1
proc {B U V}	z1	Z -> z1
local F in	p1	P -> p1
F=A+1	q1	Q -> q1
V=U+F		
end		
end		
end, E)		
({P X Q}, E)		
({Q Y Z}, E)		

=> Asignación de variable Y = 4

Store	E
x1 = 6	X -> x1
y1 = 4	Y -> y1
z1	Z -> z1
p1	P -> p1
	Q -> q1
	x1 = 6 y1 = 4

=> Declaración procedure value

Stack	Store	E
({P X Q}, E)	x1 = 6	X -> x1
((Q Y Z), E)	y1 = 4	Y -> y1
	z1	Z -> z1
	p1 = (proc { \$ A B }	P -> p1
	proc {B U V}	Q -> q1
	local F in	
	F=A+1	
	V=U+F	
	end	
	end	
	end, CE ={})	
	q1	

=> Aplicación procedure value ({P X Q}, E)

Stack	Store	E
proc {B U V} local F in F=A+1 V=U+F end	x1 = 6 y1 = 4 z1 p1 = (proc {\$ A B} proc {B U V}	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1
end, E') ({Q Y Z}, E)	local F in F=A+1 V=U+F	E' = CE + {A-> E(X); B-> E(Q)}
	end end, CE ={}} q1	A->x1 B->q1

=> Declaración procedure value

Stack	Store	E
({Q Y Z}, E)	x1 = 6 y1 = 4 z1 p1 = (proc {\$ A B} proc {B U V} local F in F=A+1 V=U+F end end, CE ={})	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1 E' A->x1 B->q1

q1 = (proc {\$ U V} local F in F=A+1 V=U+F	
end end, CE2 = {A->x1})	

=> Ejecución procedure value ({Q Y Z}, E)

Stack	Store	E
(local F in F=A+1 V=U+F end, E")	x1 = 6 y1 = 4 z1 p1 = (proc {\$ A B} proc {B U V}	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1
	local F in F=A+1 V=U+F end end	E' A->x1 B->q1
	end, CE ={}) q1 = (proc {\$ U V} local F in F=A+1	E" = CE2 + {U-> E(Y); V-> E(Z)} A->x1
	V=U+F end end, CE2 = {A->x1})	U->y1 V->z1

=> Ejecución declaración local F

Stack	Store	E
(F=A+1, E'") (V=U+F, E"")	x1 = 6 y1 = 4 z1 p1 = (proc {\$ A B} proc {B U V} local F in F=A+1 V=U+F end end end, CE ={}) q1 = (proc {\$ U V} local F in F=A+1	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1 E' A->x1 B->q1 E"

V=U+F end	U->y1 V->z1
end, CE2 = {A->x1}) f1	E'" = E" +{F->f1 nueva variable}
	A->x1 U->y1 V->z1 F-> f1

=> Ejecución asignación F = A +1

Stack	Store	E
(V=U+F, E'")	x1 = 6 y1 = 4 z1 p1 = (proc {\$ A B} proc {B U V} local F in F=A+1 V=U+F end end, CE ={}) q1 = (proc {\$ U V} local F in F=A+1 V=U+F end end, CE2 = {A->x1}) f1 = 7	X -> x1 Y -> y1 Z -> z1 P -> p1 Q -> q1 E' A->x1 B->q1 E" A->x1 U->y1 V->z1 E"' A->x1 U->y1 V->z1 F-> f1

=> Ejecución asignación V = U + F

Stack	Store	Е
-	x1 = 6 y1 = 4 z1 = 11 p1 = (proc {\$ A B}	X -> x1 Y -> y1 Z -> z1 P -> p1
	proc {B U V}	Q -> q1

local F in F=A+1 V=U+F end end, CE ={}) q1 = (proc {\$ U V} local F in F=A+1 V=U+F end end, CE2 = {A->x1})	E' A->x1 B->q1 E" A->x1 U->y1 V->z1 E"'
V=U+F end	

Fin del programa

Programa 6

Estado inicial

Stack	Store	E
local X Y Z in	-	-

=> Declaración de variables

Stack	Store	E
(X = Y, E) try (X = 1, E) (Y = 2, E) (Z = 3, E) catch Exception then (skip, E)	x1 y1 z1	X -> x1 Y -> y1 Z -> z1

end ({Browse X#Y#Z}, E)	
([DIOWSC 7(# 1 # 2]; L)	

=> Asignación de variables

Stack	Store	E
try	x1 = y1	X -> x1
(X = 1, E)	z1	Y -> y1
(Y = 2, E)		Z -> z1
(Z = 3, E)		
catch Exception then		
(skip, E)		
end		
({Browse X#Y#Z}, E)		

=> Ejecución de X = 1 dentro del try

Stack	Store	E
(Y = 2, E) (Z = 3, E) catch Exception then (skip, E) end ({Browse X#Y#Z}, E)	x1 = y1 = 1 z1	X -> x1 Y -> y1 Z -> z1

=> Ejecución de Y = 2 dentro del try -> no puedo porque y1 ya tiene valor, desapilamos todo hasta la excepcion

Stack	Store	E
catch Exception then (skip, E) end ({Browse X#Y#Z}, E)	x1 = y1 = 1 z1	X -> x1 Y -> y1 Z -> z1

=> Declaración Exception

Stack	Store	Е
(skip, E') ({Browse X#Y#Z}, E)	x1 = y1 = 1 z1 e1 = Invalid	X -> x1 Y -> y1 Z -> z1
		E' = E + Declaracion excepcion

	X -> x1 Y -> y1 Z -> z1
	Exception -> e1

=> Ejecución Skip

Stack	Store	Е
({Browse X#Y#Z}, E)	x1 = y1 = 1 z1 e1 = Invalid	X -> x1 Y -> y1 Z -> z1
		X -> x1 Y -> y1 Z -> z1 Exception -> e1

=> Ejecución Browse

Stack	Store	E
	x1 = y1 = 1 z1 e1 = Invalid	X -> x1 Y -> y1 Z -> z1
		X -> x1 Y -> y1 Z -> z1 Exception -> e1

Como el valor de Z no está determinado el browse mostrará un '_' indicando que aún no se definió, en otras operaciones podría suspenderse la ejecución, pero no ocurre con Browse. La ejecución mostrará:

1#1#_

Fin del programa

Ejercicio 4) - Case

{Test [b c a]}

Predicción: 'case'(4)

Ejecución: 'case'(4)

La lista no empieza con a como primer elemento, ni es un record, es una lista pero el primer y elemento no son iguales, luego es una lista, entonces case 4.

{Test f(b(3))}

Predicción: 'case'(5)

Ejecución: 'case'(5)

No es lista que empiece con a, si bien es un tupla llamada f, su valor no es a sino b(3), tampoco es una lista, con lo cual saltamos al caso 5 que cumple, dado que es un tupla llamada f y Y tomará el valor b(3)

{Test f(a)}

Predicción: 'case'(2)

Ejecución: 'case'(2)

No es lista, entonces analiza el case 2 donde coincide el nombre de la tupla y el elemento que contiene.

{Test f(a(3))}

Predicción: 'case'(5)

Ejecución: 'case'(5)

No es lista, luego es una tupla llamada f pero el elemento no es a, luego caso 3 y 4 no cumple por no ser una lista, en el caso 5 satisface la condición porque se llama f y el valor de la variable Y será a(3)

{Test f(d)}

Predicción: 'case'(5)

Ejecución: 'case'(5)

No es lista, luego es una tupla llamada f pero el elemento no es a, luego caso 3 y 4 no cumple por no ser una lista, en el caso 5 satisface la condición porque se llama f y el valor de la variable Y será d

{Test [a b c]}

Predición: 'case'(1)

Ejecución: 'case'(1)

Es una lista que empieza con el valor a, cumple la primer condición.

{Test [c a b]}

Prediccion: 'case'(4)

Ejecución: case (4)

No es una lista que comience con el valor a, luego no es una tupla, luego no es una lista con los primeros dos elementos iguales. Finalmente es una lista por lo que entra en el caso 4.

{Test ala}

Predicción: 'case'(1)

Ejecución: 'case'(1)

Es una lista que comienza con el valor a.

{Test '|'(v b)}

Predicción: 'case'(6)

Ejecución: 'case'(4)

No es una lista que comience con a, tampoco es tupla, no es una lista con dos elementos iguales al inicio, pero si es una lista donde el valor de Zes b. El motivo por el cual mi predicción fue incorrecta es que pensé que el formato aceptado era una cabeza con una cola, o sea que debía ser 'l'(v [b]), pero es incorrecto.

{Test '|'(a a)}

Predicción: 'case'(6)

Ejecución: 'case'(1)

Es una lista que comienza con a, por el mismo motivo que el punto anterior me equivoqué en la predicción.

{Test '|'(b b)}

Predicción 'case'(6)

Ejecución: 'case'(3)

No es una lista que comienza con a, luego tampoco es tupla, luego si es una lista con ambos elementos primero y segundo iguales.

{Test '|'(a b c)}

Predicción 'case'(6)

Ejecución: 'case'(6)

Al tener 3 elementos no coincide con una lista iniciando con a, luego no es tupla ni lista con ambos elementos iguales, tampoco es lista en la 4ta opcion, tampoco tupla llamada f, y queda como única opción el caso 6.

{Test 'l'(a [b c]}

Predicción: 'case'(1)

Ejecución: 'case'(1)

En este caso si crea con la cabeza y la cola la lista con 3 elementos que empiezan con a, por lo tanto coincide la primer condición.

Ejercicio 5) - Recursividad

1. Traducción al lenguaje Kernel

```
local Length in
      Length = proc {$ Xs N}
                  case Xs of nil then
                        N = 0
                  else
                        case Xs of _|T then
                              local U in
                                    {Length T U}
                                    N = U + 1
                              end
                        else
                              skip
                        end
                  end
            end
      local K in
            {Length [1 2 3 4] K}
            {Show K}
      end
end
```

2. "Tail Recursive"

```
local Length in
   Length = proc {$ Xs A N}
                        case Xs of nil then
                              N = A
                        else
                              case Xs of _|T then
                                    local X in
                                          X = A + 1
                                          {Length T X N}
                                    end
                              else
                                    skip
                              end
                        end
                  end
      local K in
```

```
{Length [1 2 3 4] 0 K}
{Show K}
end
end
```

En el primer caso no es la llamada recursiva lo último que se ejecuta, entonces voy a tener las operaciones que siguen acumuladas en el stack y hasta que no se termine la invocación de la última llamada recursiva no voy a poder liberar el stack. Al hacerlo tail recursive no tengo en el stack operaciones pendientes, el cual me queda claramente más pequeño. Este es el motivo por el cual siempre debemos tratar de hacer la invocación a la recursividad al final. Esto se verá reflejado en el punto 3.

3. Ejecución en la máquina abstracta

Implementación básica

Estado inicial

Stack	Store	E
local Length in Length = proc {\$ Xs N}	-	-
end end		

=> Declaración de variable Length

Stack	Store	Е
<pre>(Length = proc {\$ Xs N}</pre>	1	Length -> 1

```
end
else
skip
end
end
end, E)
(local K in
{Length [1 2 3 4] K}
{Show K}
end, E)
```

=> Asignacion de stored procedure

Stack	Store	E
(local K in {Length [1 2 3 4] K} {Show K} end, E)	<pre>11 = (proc {\$ Xs N}</pre>	Length -> 11

=> Declaración de variable local K

Stack	Store	E
({Length [1 2 3 4] K}, E') ({Show K}, E')	l1 = (proc {\$ Xs N} case Xs of nil then	Length -> 11
((Show K), E)	N = 0 else	E' = E + {K->k1}
	case Xs of _ T then local U in {Length T U} N = U + 1	Length -> 11 K-> k1
	end else skip	
	end end, CE={Length -> I1}) k1	

=> Ejecución de procedure value

Stack	Store	E
(case Xs of nil then N = 0	I1 = (proc {\$ Xs N} case Xs of nil then	Length -> I1
else case Xs of _IT then	N = 0	E'

local U in {Length T U} N = U + 1	else case Xs	s of _IT then local U in	Length -> I1 K-> k1
end else skip end, E") ({Show K}, E')	else end end end, CE={Length -> I1}) k1 =n1 xs1 = [1 2 3 4]	{Length T U} N = U + 1 end skip	E"= CE + {Xs -> E'([1 2 3 4]); N -> E'(K)} Xs-> xs1 N-> n1

=> Ejecución de case

Stack	Store	E
(case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end, E'') ({Show K}, E')	11 = (proc {\$ Xs N}	Length -> I1
	case Xs of nil then $N = 0$ else	E'
	case Xs of _IT then local U in {Length T U} N = U + 1	Length -> I1 K-> k1
	end else	E"
	skip end end end, CE={Length -> I1}) k1 = n1 xs1 = [1 2 3 4]	Xs-> xs1 N-> n1

=> Ejecución de case

Stack	Store	E
(local U in {Length T U}	I1 = (proc {\$ Xs N} case Xs of nil then	Length -> I1
N = U + 1 end, E'") ({Show K}, E')	N = 0	E'
	case Xs of _IT then local U in {Length T U} N = U + 1	Length -> I1 K-> k1
	end else	E"
	skip end end	Xs-> xs1 N-> n1
	end, CE={Length -> I1}) k1 = n1	E'''
	xs1 = [1 2 3 4] t1 = [2 3 4]	Xs-> xs1 N-> n1 T-> t1

=> Declaración de variable local U

Stack	Store	E
({Length T U}, E'''') (N = U + 1, E'''')	I1 = (proc {\$ Xs N} case Xs of nil then	Length -> I1
({Show K}, E')	N = 0 else	E,
	case Xs of _IT then local U in {Length T U} N = U + 1	Length -> I1 K-> k1
	end else	E"
	skip end end, CE=(Length -> I1)) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1	Length -> I1 Xs-> xs1 N-> n1 E''' Length -> I1 Xs-> xs1 N-> n1 T-> t1
		E"" Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1

=> Ejecución stored procedure

Stack	Store	E
(case Xs of nil then	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end end end, CE=(Length -> I1)) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4]	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"" Length -> I1 U->u1
E5 = CE + {Xs->E''''(T); N->E''''(U)}		
Length -> I1 Xs-> xs2 N-> n2		

Stack	Store	Е
(case Xs of _IT then local U in {Length T U}	I1 = (proc {\$ Xs N} case Xs of nil then	Length -> I1
{Length U}	N = 0 else case Xs of _IT then local U in {Length T U} N = U + 1 end else skip end end, CE={Length -> I1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4]	E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1
		E"" Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5		
Length -> I1 Xs-> xs2 N-> n2		

Stack	Store	E
(local U in {Length T U} N = U + 1 end, E6) (N = U + 1, E''') ({Show K}, E')	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end, CE=(Length -> I1)) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4]	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"" Length -> I1 T-> t1 U->u1
E5 Length -> I1 Xs-> xs2 N-> n2	E6 Length -> I1 Xs-> xs2 N-> n2 T-> t2	

=> Declaración variable local U

Stack	Store	E
((Length T U), E7) (N = U + 1, E7) (N = U + 1, E"") ({Show K}, E')	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end, CE={Length -> I1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"" Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5 Length -> I1 Xs-> xs2 N-> n2	E6 Length -> I1 Xs-> xs2 N-> n2 T-> t2	E7 Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2

=> Ejecución de procedure value

Stack	Store	E
(case Xs of nil then N = 0	I1 = (proc {\$ Xs N} case Xs of nil then	Length -> I1
else case Xs of _ T then	N = 0 else	E'
local U in {Length T U} N = U + 1 end	case Xs of _ T then local U in {Length T U} N = U + 1	Length -> I1 K-> k1
else skip	end else	E"
end end, E8) (N = U + 1, E7) (N = U + 1, E"") ({Show K}, E')	skip end end, CE={Length -> I1}) k1 = n1 xs1 = [1 2 3 4]	Length -> I1 K-> k1 Xs-> xs1 N-> n1
	t1 = [2 3 4]	E'''
	u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 = n3	Length -> I1 Xs-> xs1 N-> n1 T-> t1
		E''''
		Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8 = CE + {Xs->E7(T) ; N->E7(U)}		
Length -> I1 Xs->t2 N->n3		

Stack	Store	E
case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end, E8 (N = U + 1, E7) (N = U + 1, E"") ({Show K}, E')	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end end, CE={Length -> 1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 = n3	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"" Length -> I1 J-> t1 U-> u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8		
Length -> I1 Xs->t2 N->n3		

Stack	Store	E
local U in {Length T U} N = U + 1 end, E9) (N = U + 1, E7) (N = U + 1, E"") ({Show K}, E')	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U}	Length -> I1 E' Length -> I1 K-> k1
	N = U + 1 end else skip end end, CE={Length -> I1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 = n3 t3 = [4]	E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E" Length -> I1 Xs-> xs1 N-> n1 T-> t1
		E''' Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	

=> Declaración de local U

Stack	Store	E
((Length T U), E10) (N = U + 1, E10) (N = U + 1, E''') (N = U + 1, E'''') ({Show K}, E')	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end end, CE=(Length -> 1]) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 = n3 t3 = [4] u3	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"" Length -> I1 J-> u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3

=> Ejecución de procedure value

Stack	Store	Е
(case Xs of nil then	I1 = (proc (\$ Xs N)	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"" Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3
E11 = CE + {Xs->E10(T); N->E10(U)}		
Length-> I1 Xs->t3 N->n4		

Stack	Store	E
(case Xs of _IT then local U in	I1 = (proc {\$ Xs N} case Xs of nil then	Length -> I1
{Length T U} N = U + 1	N = 0 else	E'
end else skip	case Xs of _IT then local U in {Length T U}	Length -> I1 K-> k1
end, E11) (N = U + 1, E10) (N = U + 1, E7)	N = U + 1 end	E"
(N = U + 1, E'''') ({Show K}, E')	else skip end	Length -> I1 K-> k1 Xs-> xs1
	end end, CE={Length -> I1}) k1 = n1	N-> n1
	xs1 = [1 2 3 4] t1 = [2 3 4]	E'''
	u1 = n2 xs2 = [2 3 4] t2 = [3 4]	Length -> I1 Xs-> xs1 N-> n1 T-> t1
	u2 =n3 t3 = [4]	E'''
	u3 =n4	Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3
E11		
Length-> I1 Xs->t3 N->n4		

Stack	Store	E
(local U in {Length T U} N = U + 1 end, E12) (N = U + 1, E10) (N = U + 1, E7) (N = U + 1, E"") ({Show K}, E')	I1 = (proc {\$ Xs N}	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1
	end end end, CE={Length -> I1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 = n3 t3 = [4] u3 = n4 t4 = nil	K-> k1 Xs-> xs1 N-> n1 E"' Length -> l1 Xs-> xs1 N-> n1 T-> t1 E"'' Length -> l1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3
E11	E12	
Length-> I1 Xs->t3 N->n4	Length-> I1 Xs->t3 N->n4 T-> t4	

=> Ejecución de local U

Stack	Store	E
((Length T U), E13) (N = U + 1, E13) (N = U + 1, E10) (N = U + 1, E"") ({Show K}, E')	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end, CE=(Length -> I1)) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] t1 = n2 xs2 = [2 3 4] t2 = n3 t3 = [4] u3 = n4 t4 = nil u4	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"'' Length -> I1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3
E11	E12	E13
Length-> I1 Xs->t3 N->n4	Length-> I1 Xs->t3 N->n4 T-> t4	Length-> I1 Xs->t3 N->n4 T-> t4 U->u4

=> Ejecucion procedure value

Stack	Store	E
(case Xs of nil then N = 0 else	I1 = (proc {\$ Xs N} case Xs of nil then N = 0	Length -> I1
case Xs of _IT then local U in {Length T U} N = U + 1 end else skip end end, E14) (N = U + 1, E13) (N = U + 1, E10) (N = U + 1, E7) (N = U + 1, E") (Show K}, E')		Length -> I1 K-> k1 E" Length -> I1 K-> k1 X-> k1 X-> x1 N-> n1 E" Length -> I1 X-> x2 In X-> x3 In X-> x1 In X-> x3 In X-> x1 In X-> x1 In X-> x3 In X-> x1 In X-> x3 In X-> x1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3
E11	E12	E13
Length-> I1 Xs->t3 N->n4	Length-> I1 Xs->t3 N->n4 T-> t4	Length-> I1 Xs->t3 N->n4 T-> t4 U->u4

E14= CE + {Xs->E13(T); N->E13(U)}	
Length-> I1 Xs->t4 N->n5	

Stack	Store	E
(N = 0, E14) (N = U + 1, E13) (N = U + 1, E10) (N = U + 1, E7) (N = U + 1, E'''') ({Show K}, E')	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end, CE={Length -> 1]) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] t1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 = n3 t3 = [4] u3 = n4 t4 = nil u4 = n5	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"'' Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5 Length -> I1 Xs-> xs2 N-> n2	E6 Length -> I1 Xs-> xs2 N-> n2 T-> t2	E7 Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3
E11	E12	E13

Length-> I1 Xs->t3 N->n4	Length-> I1 Xs->t3 N->n4 T-> t4	Length-> I1 Xs->t3 N->n4 T-> t4 U->u4
E14		
Length-> I1 Xs->t4 N->n5		

=> Asignación N =0

Stack	Store	E
(N = U + 1, E13) (N = U + 1, E10) (N = U + 1, E7)	I1 = (proc {\$ Xs N} case Xs of nil then N = 0	Length -> I1
(N = U + 1, E''') ({Show K}, E')	else case Xs of _IT then local U in {Length T U}	Length -> I1 K-> k1
	N = U + 1 end else	E" Length -> I1
	skip end end end, CE={Length -> I1})	K-> k1 Xs-> xs1 N-> n1
	k1 = n1 xs1 = [1 2 3 4]	E'''
	t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 = n3	Length -> I1 Xs-> xs1 N-> n1 T-> t1
	t3 = [4] u3 = n4	E''''
	t4 =nil u4 =n5 = 0	Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2	Length -> I1 Xs->t2	Length -> I1 Xs->t2

N->n3	N->n3 T-> t3	N->n3 T-> t3 U->u3
E11	E12	E13
Length-> I1 Xs->t3 N->n4	Length-> I1 Xs->t3 N->n4 T-> t4	Length-> I1 Xs->t3 N->n4 T-> t4 U->u4
E14		
Length-> I1 Xs->t4 N->n5		

=> Asignación N =U+1 en E13

Stack	Store	E
(N = U + 1, E10) (N = U + 1, E7) (N = U + 1, E'"') ({Show K}, E')	I1 = (proc (\$ Xs N)	Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E"' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"" Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3
E11	E12	E13
Length-> I1 Xs->t3 N->n4	Length-> I1 Xs->t3 N->n4 T-> t4	Length-> I1 Xs->t3 N->n4 T-> t4 U->u4

=> Asignación N =U+1 en E10

Stack	Store	E
(N = U + 1, E7) (N = U + 1, E"") ({Show K}, E')	I1 = (proc {\$ Xs N} case Xs of nil then N = 0	Length -> I1
	else case Xs of _IT then local U in {Length T U} N = U + 1 end else skip end end, CE={Length -> I1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 = n3 = 2 t3 = [4] u3 = n4 = 1 t4 = nil u4 = n5 = 0	Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E" Length -> I1 Xs-> xs1 N-> n1 T-> t1 E"" Length -> I1 T-> t1 U->u1
E5	E6	E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2	Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	E9	E10
Length -> I1 Xs->t2 N->n3	Length -> I1 Xs->t2 N->n3 T-> t3	Length -> I1 Xs->t2 N->n3 T-> t3 U->u3

=> Asignación N =U+1 en E7

Stack	Store	E
(N = U + 1, E"")	I1 = (proc {\$ Xs N}	Length -> I1
({Show K}, E')	case Xs of nil then N = 0 else	E'

	end end, CE={Length - k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 = 3 xs2 = [2 3 4] t2 = [3 4] u2 = n3 = 2 t3 = [4] u3 = n4 = 1 t4 =nil u4 =n5 = 0	else end	end skip	Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E''' Length -> I1 Xs-> xs1 N-> n1 T-> t1 E'''' Length -> I1 Xs-> xs1 N-> n1 T-> t1 U->u1
E5	E6			E7
Length -> I1 Xs-> xs2 N-> n2	Length -> I1 Xs-> xs2 N-> n2 T-> t2			Length -> I1 Xs-> xs2 N-> n2 T-> t2 U->u2

=> Asignación N =U+1 en E'''

Store	E
I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end, CE=(Length -> 1]) k1 = n1 = 4 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 = 3	E Length -> I1 E' Length -> I1 K-> k1 E" Length -> I1 K-> k1 Xs-> xs1 N-> n1 E''' Length -> I1 Xs-> xs1
	I1 = (proc {\$ Xs N} case Xs of nil then N = 0 else case Xs of _ T then local U in {Length T U} N = U + 1 end else skip end end end, CE={Length -> I1}) k1 = n1 = 4 xs1 = [1 2 3 4] t1 = [2 3 4]

t3 = [4 u3 = n	E'''	
t4 = nil u4 = ns	Length -> I1	

=> Ejecución de Show K

Stack	Store	E
({Show K}, E')	I1 = (proc {\$ Xs N}	Length -> I1
	case Xs of nil then	
	N = 0	E'
	else case Xs of _IT then local U in {Length T U} N = U + 1 end else skip end end, CE={Length -> I1}) k1 = n1 = 4 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 = 3 xs2 = [2 3 4] t2 = [3 4] u2 = n3 = 2 t3 = [4] u3 = n4 = 1 t4 = nil u4 = n5 = 0	Length -> I1 K-> k1

Muestra el valor de E'(K) = 4

Fin del programa

Implementación Tail Recursive

Estado inicial

Stack	Store	E
local Length in Length = proc {\$ Xs A N}	-	

=> Declaración de variable

Stack	Store	E
(Length = proc {\$ Xs A N} case Xs of nil then N = A	ıı	Length -> I1
else		
case Xs of _IT then local X in		
X = A + 1 {Length T X N}		
end		
else		
skip		
end		
end		
end, E)		
(local K in		
{Length [1 2 3 4] 0 K} {Show K}		
end, E)		

=> Asignación de procedure value

Stack	Store	E
(local K in {Length [1 2 3 4] 0 K} {Show K} end, E)	I1= (proc {\$ Xs A N}	Length -> I1

=> Declaración local K

Stack	Store	E
((Length [1 2 3 4] 0 K), E1)	I1= (proc {\$ Xs A N}	Length -> I1
((Show K), E1)	case Xs of nil then N = A	E1
	else	Length -> I1 K->k1

=> Ejecución procedure value

Stack		Store	E
(case Xs of nil the	nen	I1= (proc {\$ Xs A N} case Xs of nil then	Length -> l1
else	a of IT these	N = A	E1
case X	s of _ T then local X in X = A + 1 {Length T X N}	else case Xs of _ T then local X in X = A + 1	Length -> I1 K->k1
else	end	{Length T X N} end	E2
end end, E2) ({Show K}, E1)	skip	else skip end end	Length-> I1 Xs->xs1 A->a1 N->n1
		end, CE={Length-> 1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0	143111

Stack	Store	E
(case Xs of _ T then	I1= (proc {\$ Xs A N} case Xs of nil then N = A else case Xs of _ T then local X in X = A + 1 {Length T X N} end else skip end end, CE=(Length-> 1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0	Length -> I1 E1 Length -> I1 K->k1 E2 Length-> I1 Xs->xs1 A->a1 N->n1

Stack	Store	E
local X in	I1= (proc {\$ Xs A N}	Length -> I1 E1 Length -> I1 K->k1 E2 Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1

=> Declaración local X

Stack	Store	E
(X = A + 1, E4)	l1= (proc (\$ Xs A N) case Xs of nil then	Length -> I1
((Length T X N), E4) ((Show K), E1)	N = A else	E1
	case Xs of _IT then local X in X = A + 1	Length -> I1 K->k1
	{Length T X N}	E2
	else skip end end end, CE={Length->I1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1	Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1

=> Suma X = A+1 en E4

Stack	Store	E
((Length T X N), E4)	I1= (proc {\$ Xs A N}	Length -> I1
({Show K}, E1)	case Xs of nil then N = A else	E1
	case Xs of _IT then local X in $X = A + 1$	Length -> I1 K->k1
	{Length T X N}	E2
		Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1

=> Aplicación de procedure value

Stack	Store	E
(case Xs of nil then N = A	I1= (proc {\$ Xs A N} case Xs of nil then	Length -> I1
else case Xs of _IT then	N = A else	E1
local X in X = A + 1 {Length T X N}	case Xs of _IT then local X in X = A + 1	Length -> I1 K->k1
end	{Length T X N}	E2
skip end end, E5) ({Show K}, E1)	else skip end end end, CE=(Length->I1)) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1	Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5		
Length->l1 Xs-> t1 A-> x1 N-> n1		

Stack	Store	E
(case Xs of _IT then local X in	I1= (proc {\$ Xs A N} case Xs of nil then	Length -> I1
X = A + 1 {Length T X N}	N = A else	E1
end else skip	case Xs of _IT then local X in X = A + 1	Length -> I1 K->k1
end, E5) ((Show K), E1)	{Length T X N}	E2
(CHOW IV), LI)	ellu else skip end end end, CE={Length->I1})	Length-> I1 Xs->xs1 A->a1 N->n1
	k1 = n1 xs1 = [1 2 3 4]	E3
	a1 = 0 t1 = [2 3 4] x1 = 1	Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5		
Length->I1 Xs-> t1 A-> x1 N-> n1		

Stack	Store	E
(local X in	I1= (proc {\$ Xs A N}	Length -> I1
X = A + 1	case Xs of nil then	
{Length T X N}	N = A	E1
end, E6)	else	
({Show K}, E1)	case Xs of _IT then	Length -> I1
	local X in	K->k1
	X = A + 1	
	{Length T X N}	E2
	end	
	else	Longth > I1
	skip	Length-> I1
	SKIP	Xs->xs1

	end end end, CE={Length->I1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4]	A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 E4 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 N->n1 T->t1 X->x1
E5	E6	
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2	

=> Declaración local X

Stack	Store	E
(X = A + 1, E7) ((Length T X N), E7) ((Show K), E1)	I1= (proc {\$ Xs A N}	Length -> I1
		E1
		Length -> l1 K->k1
		E2
		Length-> I1 Xs->xs1 A->a1 N->n1
		E3
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2

=> Ejecución X = A +1 en E7

Stack	Store	E
((Length T X N), E7) ((Show K), E1)	I1= (proc {\$ Xs A N} case Xs of nil then	Length -> I1
	N = A else	E1
	case Xs of _IT then local X in X = A + 1	Length -> I1 K->k1
	{Length T X N}	E2
else skip end end end, CE={Length->I1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2	else skip end end	Length-> I1 Xs->xs1 A->a1 N->n1
	k1 = n1	E3
	a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4]	Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2

=> Ejecución procedure value

Stack	Store	E
(case Xs of nil then N = A	I1= (proc {\$ Xs A N} case Xs of nil then	Length -> I1
else case Xs of _IT then	N = A else	E1
local X in X = A + 1 {Length T X N}	case Xs of _IT then local X in X = A + 1	Length -> I1 K->k1
end else	{Length T X N}	E2
skip end end, E8) ((Show K), E1)	else	Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 E4 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 N->n1 T->t1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8		
Length->l1 Xs-> t2 A->x2 N->n1		

Stack	Store	E
(case Xs of _ T then local X in	I1= (proc {\$ Xs A N} case Xs of nil then	Length -> I1
X = A + 1 {Length T X N}	N = A else	E1
end else skip	case Xs of _IT then local X in X = A + 1	Length -> I1 K->k1
end, E8)	{Length T X N}	E2
((Show K), E1)	else skip end end end, CE=(Length->I1)) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2	Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8		
Length->I1 Xs-> t2 A->x2 N->n1		

Stack	Store	Е
(local X in	I1= (proc {\$ Xs A N}	Length -> I1
		Length -> I1 K->k1
	{Length T X N} end	E2
	ellu else skip end end end, CE={Length->I1})	Length-> I1 Xs->xs1 A->a1 N->n1
	k1 = n1 xs1 = [1 2 3 4]	E3
	a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2 t3 = [4]	Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	
Length->I1 Xs-> t2 A->x2 N->n1	Length->I1 Xs-> t2 A->x2 N->n1 T->t3	

=> Declaración local X

Stack	Store	Е
(X = A + 1, E10) ((Length T X N), E10) ((Show K), E1)	I1= (proc {\$ Xs A N}	Length -> I1
		E1
		Length -> I1 K->k1
		E2
		Length-> I1 Xs->xs1 A->a1 N->n1
	k1 = n1 xs1 = [1 2 3 4]	E3
a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2	t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2 t3 = [4]	Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
	x3	E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->I1 Xs-> t2 A->x2 N->n1	Length->l1 Xs-> t2 A->x2 N->n1 T->t3	Length->l1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3

=> **Ejecución X = A +1 en E10**

Stack	Store	Е
((Length T X N), E10) ((Show K), E1)	I1= (proc {\$ Xs A N}	Length -> I1
		E1
		Length -> l1 K->k1
		E2
		Length-> I1 Xs->xs1 A->a1 N->n1
	k1 = n1 xs1 = [1 2 3 4]	E3
	a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2 t3 = [4] x3 = 3	Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->I1 Xs-> t2 A->x2 N->n1	Length->I1 Xs-> t2 A->x2 N->n1 T->t3	Length->l1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3

=> Ejecución procedure value

Stack	Store	E
(case Xs of nil then	I1= (proc {\$ Xs A N}	Length -> I1 E1 Length -> I1 K->k1 E2 Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 E4 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 Xs->xs1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->I1 Xs-> t2 A->x2 N->n1	Length->l1 Xs-> t2 A->x2 N->n1 T->t3	Length->l1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3
E11		
Length->l1 Xs-> t3 A->x3 N->n1		

=> Ejecución case

Stack	Store	Е
(case Xs of _ T then local X in X = A + 1 {Length T X N} end else skip end, E11) ((Show K), E1)	I1= (proc {\$ Xs A N}	Length -> I1 E1 Length -> I1 K->k1 E2 Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 E4 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 T->t1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1
E5	E6	E7
Length->l1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->l1 Xs-> t2 A->x2 N->n1	Length->I1 Xs-> t2 A->x2 N->n1 T->t3	Length->l1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3
E11		
Length->I1 Xs-> t3 A->x3 N->n1		

=> Ejecución case

Stack	Store	E
local X in	If= (proc {\$ Xs A N} case Xs of nil then N = A else case Xs of _ T then local X in X = A + 1 {Length T X N} end else skip end end end, CE=(Length-> 1) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2 t3 = [4] x3 = 3 t4 = nil	Length -> I1 E1 Length -> I1 K->k1 E2 Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 E4 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 Xs->xs1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->l1 Xs-> t2 A->x2 N->n1 E11 Length->l1	Length->I1 Xs-> t2 A->x2 N->n1 T->t3 E12 Length->I1	Length->l1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3
Xs-> t3 A->x3 N->n1	Xs-> t3 A->x3 N->n1 T->t4	

=> Declaración local X

Stack	Store	E
(X = A + 1, E13) ((Length T X N), E13)	I1= (proc {\$ Xs A N} case Xs of nil then	Length -> I1
((Show K), E1)	N = A else	E1
	case Xs of _IT then local X in X = A + 1	Length -> I1 K->k1
	{Length T X N} end	E2
	else skip end end end, CE={Length->I1})	Length-> I1 Xs->xs1 A->a1 N->n1
	k1 = n1 xs1 = [1 2 3 4]	E3
	a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2 t3 = [4]	Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
	x3 = 3 t4 = nil	E4
x4	Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1	
E5	E6	E7
Length->l1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->l1 Xs-> t2 A->x2 N->n1	Length->l1 Xs-> t2 A->x2 N->n1 T->t3	Length->I1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3
E11	E12	E13
Length->l1 Xs-> t3 A->x3 N->n1	Length->l1 Xs-> t3 A->x3 N->n1 T->t4	Length->l1 Xs-> t3 A->x3 N->n1 T->t4 X-> x4

=> Asignación X = A+1 en E13

Stack	Store	E
((Length T X N), E13) ((Show K), E1)	If= (proc {\$ Xs A N}	Length -> I1 E1 Length -> I1 K->k1 E2 Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 E4 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 Length-> I1 Xs->xs1 X->xs1 X->xs1 X->xs1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->l1 Xs-> t2 A->x2 N->n1	Length->I1 Xs-> t2 A->x2 N->n1 T->t3	Length->l1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3
E11	E12	E13
Length->l1 Xs-> t3 A->x3 N->n1	Length->I1 Xs-> t3 A->x3 N->n1 T->t4	Length->l1 Xs-> t3 A->x3 N->n1 T->t4 X-> x4

=> Ejecución procedure value

Stack	Store	E
(case Xs of nil then N = A else case Xs of _IT then local X in X = A + 1 {Length T X } end else skip end end, E14) ((Show K), E1)	I1= (proc {\$ Xs A N} case Xs of nil then N = A else case Xs of _IT then local X in	Length -> I1 E1 Length -> I1 K->k1 E2 Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 E4 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 X->xs1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->I1 Xs-> t2 A->x2 N->n1	Length->I1 Xs-> t2 A->x2 N->n1 T->t3	Length->I1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3
E11	E12	E13
Length->l1 Xs-> t3 A->x3 N->n1	Length-> 1 Xs-> t3 A->x3 N->n1 T->t4	Length->l1 Xs-> t3 A->x3 N->n1 T->t4 X-> x4

E14	
Length->l1 Xs->t4 A->x4 N->n1	

=> Ejecución case

Stack	Store	E
(N = A, E14)	I1= (proc {\$ Xs A N}	Length -> I1
((Show K), E1)	case Xs of nil then N = A else	E1
	case Xs of _IT then local X in X = A + 1	Length -> I1 K->k1
	{Length T X N}	E2
	else	Length-> I1 Xs->xs1 A->a1 N->n1 E3 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 E4 Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 Xs->xs1 A->a1 N->n1 Xs->xs1 A->a1 N->n1 Xs->xs1 A->a1 N->n1 Xs->xs1 A->a1 N->n1 Xs->xs1
E5	E6	E7
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1 T-> t2	Length-> 1 Xs-> t1 A-> x1 N-> n1 T-> t2 X->x2
E8	E9	E10
Length->I1 Xs-> t2 A->x2 N->n1	Length->l1 Xs-> t2 A->x2 N->n1	Length->l1 Xs-> t2 A->x2 N->n1

	T->t3	T->t3 X-> x3
E11	E12	E13
Length->l1 Xs-> t3 A->x3 N->n1	Length->l1 Xs-> t3 A->x3 N->n1 T->t4	Length->l1 Xs-> t3 A->x3 N->n1 T->t4 X-> x4
E14		
Length->l1 Xs->t4 A->x4 N->n1		

=> Ejecución N = A en E14

Stack	Store	E		
((Show K), E1)	I1= (proc {\$ Xs A N} case Xs of nil then	Length -> I1		
	N = A else	E1		
	case Xs of _IT then local X in X = A + 1	Length -> l1 K->k1		
	{Length T X N}	E2		
	else skip end end end, CE={Length->I1})	Length-> I1 Xs->xs1 A->a1 N->n1		
	k1 = n1 = 4 xs1 = [1 2 3 4]	E3		
	xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2 t3 = [4] x3 = 3 t4 = nil x4 = 4	a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2 t3 = [4] x3 = 3 t4 = nil	t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2 t3 = [4]	Length-> I1 Xs->xs1 A->a1 N->n1 T->t1
			E4	
		Length-> I1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1		
E5	E6	E7		
Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1	Length->I1 Xs-> t1 A-> x1 N-> n1		

	T-> t2	T-> t2 X->x2
E8	E9	E10
Length->I1 Xs-> t2 A->x2 N->n1	Length->I1 Xs-> t2 A->x2 N->n1 T->t3	Length->I1 Xs-> t2 A->x2 N->n1 T->t3 X-> x3
E11	E12	E13
Length->l1 Xs-> t3 A->x3 N->n1	Length->l1 Xs-> t3 A->x3 N->n1 T->t4	Length->l1 Xs-> t3 A->x3 N->n1 T->t4 X-> x4
E14		

=> Show E1(K) = 4, imprime un 4. Fin del programa

Stack	Store	E
-	I1= (proc {\$ Xs A N}	Length -> I1
	case Xs of nil then N = A	E1
	else	Length -> I1 K->k1

Ejercicio 6) - Alto orden con listas

FoldL

a. Código Oz

```
@ emacs@KARINA-VAIO
                                                                   X
File Edit Options Buffers Tools Oz Help
% FoldL function
declare fun {FoldL L F U}
           case L of nil then U
           [] H|T then {F {FoldL T F U} H}
           end
       end
declare fun {Multiplica X Y}
          X * Y
        end
{Show {FoldL [2 2 3] Multiplica 1}}
                      All L10
1\**- Oz
                               (Oz)
12
1\**- *Oz Emulator* All L2 (Comint:run)
menu-bar Oz Feed Buffer
```

FoldR

a. Código Oz

```
@ emacs@KARINA-VAIO
                                                                    X
File Edit Options Buffers Tools Oz Help
% FoldR function
declare fun {FoldR L F U}
           case L of nil then U
           [] H|T then {F H {FoldR T F U}}
        end
declare fun {Suma X Y}
           X + Y
        end
{Show {FoldR [1 2 3] Suma 0}}
1\**- Oz
                      All L4
                                  (Oz)
1\**- *Oz Emulator* All L2 (Comint:run)
menu-bar Oz Feed Buffer
```

Map

a. Código Oz

```
@ emacs@KARINA-VAIO
                                                                    X
File Edit Options Buffers Tools Oz Help
                                          a
% Map function
declare fun {Map L F }
           case L of nil then nil
           [] H|T then {F H}| {Map T F}
           end
        end
declare fon {Potencia X}
           X * X
{Show {Map [2 1 3 6] Potencia}}
1\--- 6Map
                      All L7
                                  (Oz)
[4 1 9 36]
1\**- *Oz Emulator*
                      All L2 (Comint:run)
Wrote c:/Users/Karina/Facu/7524/TPIndividual/6Map
```

Filter

a. Código Oz

```
@ emacs@KARINA-VAIO
                                                                    X
File Edit Options Buffers Tools Complete In/Out Signals Help
            X
% Filter function
declare fun {Filter L F}
           case L of nil then nil
           [] H|T then
              if {F H} then
                 H | {Filter T F}
              else
                 {Filter T F}
              end
           end
        end
declare fun {Condicion X}
           X > 2
        end
{Show {Filter [2 1 3 6] Condicion}}
1\--- 6Filter
                     A11 L10
                                 (Oz)
[3 6]
1\**- *Oz Emulator* All L2 (Comint:run)
```

Ejercicio 7) - Hilos

7.1 WaitSome

a. Código Oz

```
end
}
{Show 'Esperando por algun valor a bindear'}
{Wait Y}
{Show 'Al menos uno fue bindeado'}

end

end
{Show 'Primera lista los 3 valores bindeados'}
{WaitSome [1 2 3]}
{Show 'Segunda lista solo uno esta bindeado'}
{WaitSome [_ 2 _]}
{Show 'Tercera lista ninguno esta bindeado'}
{WaitSome [_ 2 _]}
end
```

```
@ emacs@KARINA-VAIO
                                                                             X
File Edit Options Buffers Tools Complete In/Out Signals Help
                  end
   {Show 'Primera lista los 3 valores bindeados'}
   {WaitSome [1 2 3]}
   {Show 'Segunda lista solo uno esta bindeado'}
   {WaitSome [_ 2 _]} {Show 'Tercera lista ninguno esta bindeado'}
   {WaitSome [_ _ ]}
end
1\**- Oz
                      Bot L16
'Primera lista los 3 valores bindeados'
'Esperando por algun valor a bindear'
'Valor Bindeado:'
'Valor Bindeado:'
'Valor Bindeado:'
'Al menos uno fue bindeado'
'Segunda lista solo uno esta bindeado'
'Esperando por algun valor a bindear'
'Valor Bindeado:'
'Al menos uno fue bindeado'
'Tercera lista ninguno esta bindeado'
'Esperando por algun valor a bindear'
1\**- *Oz Emulator* All L3
                                   (Comint:run)
```

7.2 Maquina abstracta

Estado inicial

Stack		Store	E
local A B C in thread		-	-
tilleda	if A then		
	B=true		
	else		
	B=false		
	end		
end			
thread			
	if B then		
	C=false		
	else		
	C=true		
end	end		
A=false			
end			

=> Declaración variables

Stack		Store	E
(thread		a1	A-> a1
	if A then	b1	B-> b1
	B=true else	c1	C-> c1
	B=false		
	end		
end, E) (thread			
•	if B then		
	C=false		
	else		
	C=true		
,	end		
end, E) (A=false	, E)		

=> Ejecución de thread, un nuevo stack de instrucciones es generado

Stack	Stack1	Store	E
(thread if B then C=false else C=true end end, E) (A=false, E)	(if A then B=true else B=false end, E)	a1 b1 c1	A-> a1 B-> b1 C-> c1

=> En este punto, el scheduler decidirá si continúa con la ejecución de Stack o de Stack1, vamos a seguir con Stack, ejecución de la otra sentencia thread, que creará un nuevo stack de ejecución.

Stack	Stack1	Stack2	Store	E
(A=false, E)	(if A then B=true else B=false end, E)	(if B then C=false else C=true end, E)	a1 b1 c1	A-> a1 B-> b1 C-> c1

=> En este punto, el scheduler decidirá si continúa con la ejecución de Stack, Stack1 o Stack2. En caso que continúe con Stack1, la sentencia if suspenderá su ejecución porque A no está determinado. Si continúa con Stack2 pasará lo mismo debido a la indeterminación de B. El Scheduler no tendrá otra opción que ejecutar Stack

=> Asignación A = false

Stack	Stack1	Stack2	Store	E
-	(if A then B=true else B=false end, E)	(if B then C=false else C=true end, E)	a1 = false b1 c1	A-> a1 B-> b1 C-> c1

=> El Scheduler determinará que puede continuar con la ejecución de Stack1 porque A ahora está determinado. Ejecución de sentencia if

Stack	Stack1	Stack2	Store	E
-	(B=false, E)	(if B then C=false else C=true end, E)	a1 = false b1 c1	A-> a1 B-> b1 C-> c1

=> El Scheduler determinará que sólo puede continuar con Stack1. Binding the variable B

Stack	Stack1	Stack2	Store	E
-	-	(if B then C=false else C=true end, E)	a1 = false b1 = false c1	A-> a1 B-> b1 C-> c1

=> Finalmente podemos seguir ejecutando Stack2. Ejecutamos el If

Stack	Stack1	Stack2	Store	E
-	-	(C=true, E)	a1 = false b1 = false c1	A-> a1 B-> b1 C-> c1

=> Ejecución de C=true

Stack	Stack1	Stack2	Store	E
-	-	-	b1 = false	A-> a1 B-> b1 C-> c1

FIN DE EJECUCIÓN

Ejercicio 8) - Evaluación perezosa

8.1 Maquina Abstracta

Lenguaje Kernel

```
local Ints L A B C in
       proc (Ints N L1)
              {ByNeed
                      proc {$ L2}
                             local L3 in
                                     {Ints N+1 L3}
                                     L2 = NIL3
                              end
                      end
              L1}
       end
       {Ints 1 L}
       A = L.2.2.1
       B = L.1
       C = A+B
       {Browse C}
end
```

Ejecución

Estado inicial

Stack	Store	E
local Ints L A B C in proc {Ints N L1} {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3	-	-
end L1) end {Ints 1 L} A = L.2.2.1 B = L.1 C = A+B {Browse C} end		

=> Declaración variables

Stack		Store	E	
(proc {Ints N L1} {ByNeed L1} end, E) ({Ints 1 L}, E)	d proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end end	Store i1 I0 a1 b1 c1	E Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1	
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ({Browse C}, E)				

=> Declaración procedure value

Stack	Store	E
((Ints 1 L), E) (A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E)	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end end L1} end, CE={Ints->i1}) IO a1 b1 c1	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1

=> Ejecución procedure value

Stack	Store	E
((ByNeed proc (\$ L2) local L3 in {Ints N+1 L3} L2 = NiL3 end end L1), E1) (A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E)	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end end L1} end, CE={Ints->i1}) IO=I1 a1 b1 c1 n1 = 1	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1 E1 Ints-> i1 N->n1 L1->I1

=> Ejecución ByNeed (creación de un nuevo stack como un thread)

Stack	Stack 1	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ({Browse C}, E)	((proc {\$ L2} local L3 in {Ints N+1 L3} L2 = N L3 end end L1}, E1)	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3}	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
		L2 = NIL3 end end L1} end, CE={Ints->i1}) I0=I1 a1 b1 c1 n1 = 1	Ints-> i1 N->n1 L1->l1

=> Se ejecutará el Stack 1 para obtener el valor de L.2.2.1 (primero obtendrá L.2)

Stack	Stack 1	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ({Browse C}, E)	(local L3 in {Ints N+1 L3} L2 = NIL3 end, E2)	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
		{Ints N+1 L3}	E1
		L2 = NIL3 end end	Ints-> i1 N->n1 L1->l1
		L1} end, CE={Ints->i1})	E2
		IO=I1 a1 b1 c1 n1 = 1	Ints->i1 N->n1 L2->i2

=> Stack 1 => Definición L3

Stack	Stack 1	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ({Browse C}, E)	((Ints N+1 L3), E3) (L2 = NIL3, E3)	i1= (proc {\$ N L1}	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
		{Ints N+1 L3}	E1
		L2 = NIL3 end end	Ints-> i1 N->n1 L1->l1
		L1} end, CE={Ints->i1})	E2
		IO=I1 a1 b1 c1	Ints->i1 N->n1 L2->l2
		n1 = 1 I3	E3
			Ints->i1 N->n1 L2-> 2 L3-> 3

=> Stack 1 => Ejecución Ints

Stack	Stack 1	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E)	({ByNeed proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
	end	{Ints N+1 L3}	E1
	L1), E4) (L2 = N L3, E3)	L2 = NIL3 end	Ints-> i1 N->n1 L1->l1
		end L1} end, CE={Ints->i1})	E2
		O=Id, CE={Inits->njj} O=I1 a1 b1	Ints->i1 L2->l1
		c1 n1 = 1	E3
		I3 n2 = 2	Ints->i1 N->n1 L2->l2 L3-> l3
E4			
Ints->i1 N->n2 L1->l3			

=> Stack 1 => Ejecución ByNeed (creacion de nuevo thread)

Stack	Stack 1	Stack2	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E)	(L2 = NIL3, E3)	((proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end	i1= (proc (\$ N L1) {ByNeed proc (\$ L2) local L3 in {Ints N+1 L3}	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
		end, E4)	L2 = NIL3 end end	E1
			L1} end, CE={Ints->i1}) I0=I1 a1	Ints-> i1 N->n1 L1->l1
			b1 c1	E2
			n1 = 1 l3 n2 = 2	Ints->i1 L2->l1
				E3
				Ints->i1 N->n1 L2->l2 L3-> l3
E4				
Ints->i1 N->n2 L1->l3				

=> Stack 2 => Ejecución procedure

Stack	Stack 1	Stack2	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ({Browse C}, E)	(L2 = NIL3, E3)	(local L3 in {Ints N+1 L3} L2 = NIL3 end, E5)	i1= (proc (\$ N L1) {ByNeed proc (\$ L2) local L3 in {Ints N+1 L3} L2 = NIL3	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
			end end	E1
			L1) end, CE=(Ints->i1)) I0=I1	Ints-> i1 N->n1 L1->l1
			a1 b1	E2
			c1 n1 = 1	Ints->i1 L2->l1
			n2 = 2	E3
				Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5			
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2			

=> Stack 2 => Declaración Local L3

Stack	Stack 1	Stack2	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E)	(L2 = NIL3, E3)	((Ints N+1 L3), E6) (L2 = N L3, E6)	i1= (proc (\$ N L1) {ByNeed proc (\$ L2) local L3 in {Ints N+1 L3} L2 = NIL3	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
			end end	E1
			L1} end, CE={Ints->i1}) I0=I1	Ints-> i1 N->n1 L1->l1
			I3 a1	E2
			b1 c1 n1 = 1	Ints->i1 L2->l1
			n2 = 2	E3
				Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5	E6		
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2	Ints->i1 N->n2 L2->l2 L3-> l3		

=> Stack 2 => Ejecución Ints

Stack	Stack 1	Stack2	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E)	(L2 = NIL3, E3)	((ByNeed proc (\$ L2) local L3 in {Ints N+1 L3} L2 = N L3 end	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
		end L1], E7)	end end	E1
		(L2 = NIL3, E6)	L1} end, CE={Ints->i1}) I0=I1 I2	Ints-> i1 N->n1 L1->l1
			I3 a1	E2
			b1 c1 n1 = 1	Ints->i1 L2->l1
			n2 = 2 n3 = 3	E3
				Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5	E6	E7	
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2	Ints->i1 N->n2 L2->l2 L3-> l3	Ints->i1 N->n3 L1->l3	

=> Stack 2 => Ejecución ByNeed (new thread)

(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E) ((L2 = NIL3, E3)) (L2 = NIL3, E6) ((Browse C), E) ((L2 = NIL3, E6)) ((L2 = NIL3, E6)) ((L2 = NIL3, E6)) ((Ints N+1 L3) (ByNeed proc {\$ L2} (Ints N+1 L3)	Stack	Stack 1	Stack2	Stack3	Store	E
	(B = L.1, E) (C = A+B, E)	(L2 = NIL3, E3)	(L2 = NIL3, E6)	local L3 in {Ints N+1 L3} L2 = NIL3 end	{ByNeed proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end end L1} end, CE={Ints->i1}) l0=l1 l2 l3 a1 b1 c1	L->IO A-> a1 B-> b1 C-> c1 E1 Ints-> i1 N->n1 L1->I1 E2 Ints->i1 L2->I1

				n3 = 3	Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5	E6	E7		
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2	Ints->i1 N->n2 L2->i2 L3-> i3	Ints->i1 N->n3 L1->l3		

=> Stack 2 => El Stack 3 no necesita ejecutarse porque no necesito esos valores. Se ejecuta el Stack2

Stack	Stack 1	Stack2	Stack3	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E)	(L2 = NIL3, E3)	-	((proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end end}, E7)	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
			enu ₃ , E7)	end end	E1
				L1) end, CE={Ints->i1}) I0=I1 I2= 2 I3	Ints-> i1 N->n1 L1->l1
				13 = 3 _ a1	E2
				b1 c1 n1 = 1	Ints->i1 L2->l1
				n2 = 2 n3 = 3	E3
					Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5	E6	E7		
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2	Ints->i1 N->n2 L2->l2 L3-> l3	Ints->i1 N->n3 L1->l3		

=> Stack 1 => L2=N|L3 en E3

Stack	Stack 1	Stack2	Stack3	Store	E
(A = L.2.2.1, E) (B = L.1, E) (C = A+B, E) ((Browse C), E)	(E) local L3 in {ByNeed proc {\$ L2}} (E) {Ints N+1 L3} proc {\$ L2} (E) L2 = NIL3 proc {\$ L2} local L3 in {Ints N+1 L3}	{ByNeed L- proc {\$ L2} A- local L3 in B- {Ints N+1 L3} C-	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1		
			end}, E7)	L2 = NIL3 end end	E1
				L1) end, CE={Ints->i1}) I0=I 1=1 2 I2= 2 3	Ints-> i1 N->n1 L1->l1
	I2= 2 3 3 =3 _ a1 b1 c1 n1 = 1	I3 =3I_	E2		
				b1 c1	Ints->i1 L2->l1
				n2 = 2 n3 = 3	E3
					Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5	E6	E7		
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2	Ints->i1 N->n2 L2->l2 L3-> l3	Ints->i1 N->n3 L1->l3		

=> Stack => Ejecución A=L.2.2.1 en E

Stack	Stack 1	Stack2	Stack3	Store	E
(B = L.1, E) (C = A+B, E) ((Browse C), E)	-	-	((proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3}	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
			end}, E7)	L2 = NIL3 end end	E1
			end L1} end, CE={Ints->i1}) I0=I1=1II2 I2= 2II3	Ints-> i1 N->n1 L1->l1	
				12 - 2 5 13 = 3 _ a1 = 3	E2
			b1 c1 n1 = 1	Ints->i1 L2->l1	
				n2 = 2	E3

				n3 = 3	Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5	E6	E7		
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2	Ints->i1 N->n2 L2->i2 L3-> i3	Ints->i1 N->n3 L1->l3		

=> Stack => B = L.1

Stack	Stack 1	Stack2	Stack3	Store	E
(C = A+B, E) ((Browse C), E)	-	-	((proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end	local L3 in {Ints N+1 L3}	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
			end}, E7)	L2 = NIL3 end end	E1
				end, CE={Ints->i1}) l0=l1=1 2 l2= 2 3	Ints-> i1 N->n1 L1->l1
				13 a1 = 3	E2
				b1 = 1 c1 n1 = 1	Ints->i1 L2->l1
				n2 = 2 n3 = 3	E3
					Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5	E6	E7		
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2	Ints->i1 N->n2 L2->l2 L3-> l3	Ints->i1 N->n3 L1->l3		

=> Stack => C= A + B

Stack	Stack 1	Stack2	Stack3	Store	E
({Browse C}, E)	-	-	((proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3 end end}, E7)	i1= (proc (\$ N L1) {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3} L2 = NIL3	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1

				end end	E1
				end L1} end, CE={Ints->i1}) I0=I1=1 2 I2= 2 3	Ints-> i1 N->n1 L1->l1
				12 - 2 3 13 a1 = 3	E2
				b1 = 1 c1 = 4 n1 = 1	Ints->i1 L2->l1
				n2 = 2 n3 = 3	E3
					Ints->i1 N->n1 L2->l2 L3-> l3
E4	E5	E6	E7		
Ints->i1 N->n2 L1->l3	Ints->i1 N->n2 L2->l2	Ints->i1 N->n2 L2->l2 L3-> l3	Ints->i1 N->n3 L1->l3		

Stack => {Browse C}

Stack	Stack 1	Stack2	Stack3	Store	E	
-	-	-	((proc (\$ L2) local L3 in {Ints N+1 L3} L2 = NIL3 end	i1= (proc {\$ N L1} {ByNeed proc {\$ L2} local L3 in {Ints N+1 L3}	{ByNeed L- proc {\$ L2} A local L3 in B {Ints N+1 L3} C	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
			end}, E7)	L2 = NIL3 end	E1	
				end L1} end, CE={Ints->i1}) I0=I1=1II2 I2= 2II3	Ints-> i1 N->n1 L1->l1	
				13 a1 = 3	E2	
				b1 = 1 c1 = 4 n1 = 1	Ints->i1 L2->l1	
				n2 = 2 n3 = 3	E3	
					Ints->i1 N->n1 L2->l2 L3-> l3	
E4	E5	E6	E7			
Ints->i1 N->n2	Ints->i1 N->n2	Ints->i1 N->n2	Ints->i1 N->n3			

|--|

El programa finaliza imprimiendo el valor 4, que es el valor actual de la variable C.

8.2 Reverse

- La diferencia en comportamiento es que la función Rev al ser lazy en la segunda implementación, solo va a ser ejecutada, cuando realmente sea necesario y no antes.
 Lo que ocurre es que como a su vez se encuentra dentro de otra funcion lazy, tambien va a ser ejecutada cuando necesitemos los valores de la lista y eso va a ocurrir al mismo tiempo, la necesidad de ejecución de una, en este caso fuerza a la ejecución de la otra
- Ambas funciones retornan el mismo resultado, recién se ejecutan la función recursiva al hacer uso de los elementos de la lista. A continuación un ejemplo de ejecución:

Codigo oz

```
local Reverse1 Reverse2 A B C in
   fun lazy {Reverse1 S}
      fun {Rev S R}
       case S of nil then
          {Show R}
          R
       [] X|S2 then
          {Show X}
          \{Rev S2 X | R\}
       end
      end
   in {Rev S nil} end
   fun lazy {Reverse2 S}
      fun lazy {Rev S R}
       case S of nil then
          {Show R}
          R
       [] X|S2 then
          {Show X}
          \{Rev S2 X | R\}
       end
      end
```

```
in {Rev S nil} end
  {Show 'Comenzamos'}
  A = [a b c]
  {Show 'Antes de llamar reverse1'}
  B = {Reverse1 A}
  {Show 'Antes del Show B'}
 % {Show {Reverse1 A}}
 % {Show A}
   {Show B}
  {Show 'Antes del case'}
  case B of K|_ then
     {Show K}
  end
  {Show 'Antes de llamar reverse2'}
  C = {Reverse2 A}
  {Show 'Antes de Show C'}
  %{Show {Reverse2 A}}
  {Show C}
  {Show 'Antes del case'}
  case C of M|_ then
     {Show M}
  end
end
```

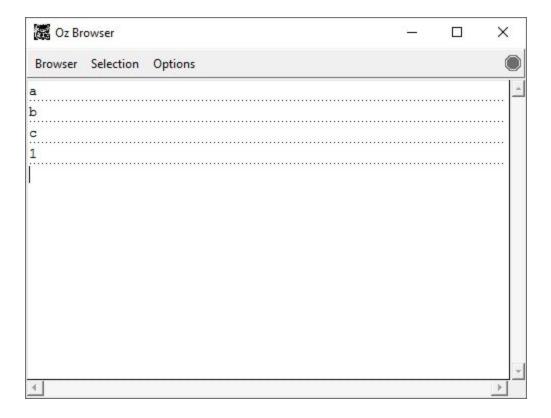
```
Oz Programming Interface (emacs@KARINA-VAIO)
                                                                             X
File Edit Options Buffers Tools Oz Help
local Reversel Reverse2 A B C in
   fun lazy {Reverse1 S}
      fun {Rev S R}
         case S of nil then
            {Show R}
            R
1\--- 8Reverse
                       Top L1
                                   (Oz)
'Comenzamos'
'Antes de llamar reverse1'
'Antes del Show B'
<optimized>
'Antes del case'
C
[c b a]
'Antes de llamar reverse2'
'Antes de Show C'
<optimized>
'Antes del case'
b
C
[c b a]
1\**- *Oz Emulator* All L16
                                   (Comint:run)
```

Ejercicio 9) - Mensajes

a. Código Oz

```
declare P in
local S in
  {NewPort S P}
  thread {ForAll S proc {$ M} {Browse M} end} end
  {Send P a}
```

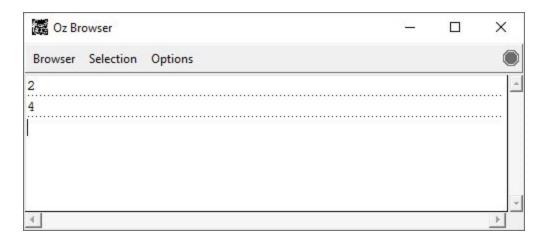
```
{Send P b}
{Send P c}
{Send P 1}
end
```



Ejercicio 10) - Servidor de filtros

a. Código Oz

```
%funcion unaria
declare fun {EsPar A}
         if (A \mod 2 == 0) then
            true
         else
            false
         end
      end
%P1 puerto donde envio todos los elementos
%P2 puerto donde envio solo los elementos que pasan el filtro
declare P1 P2 in
local S1 S2 in
   {NewPort S1 P1}
   {NewPort S2 P2}
   thread {ForAll S1 proc {$ M}
                  if {EsPar M} then
                     {Send P2 M}
                  end
                 end} end
   thread {ForAll S2 proc {$ M} {Browse M} end} end
   {Send P1 1}
   {Send P1 2}
   {Send P1 3}
   {Send P1 4}
end
```



Ejercicio 11) - Celdas de memoria

Explicación linea a linea

<pre>declare X={NewCell 0}</pre>	Declara la celda X y le asigna el valor inicial 0.
{Assign X 5}	A la celda X le asigna el nuevo valor 5
Y=X	A la celda Y le asigna X, ambas variables son la misma y tienen el mismo valor.
{Assign Y 10}	A la celda Y e asigna el valor 10, también cambiará el valor de X porque es la misma celda que puedo acceder con dos identificadores.
{Browse {Access X}==10}	Pregunta si el valor de X es 10, lo cual es verdadero. Imprime true
{Browse X==Y}	Pregunta si la celda X es igual a la celda Y, dado que son la misma celda, el valor es verdadero. Imprime true
Z={NewCell 10}	Declara una nueva celda Z, le asigna el valor 10
{Browse Z==Y}	Pregunta si la celda Z es igual a la celda Y, como son dos celdas diferentes la comparación es falsa. Imprime false
{Browse @X==@Z}	Pregunta si el valor de la celda X, que es 10 es igua al valor de la celda Z que también es 10 son iguales, la condicion es verdadera. Imprime true