7524 - Teoría de la programación

TP Individual

Karina Alaya

Padron 75840

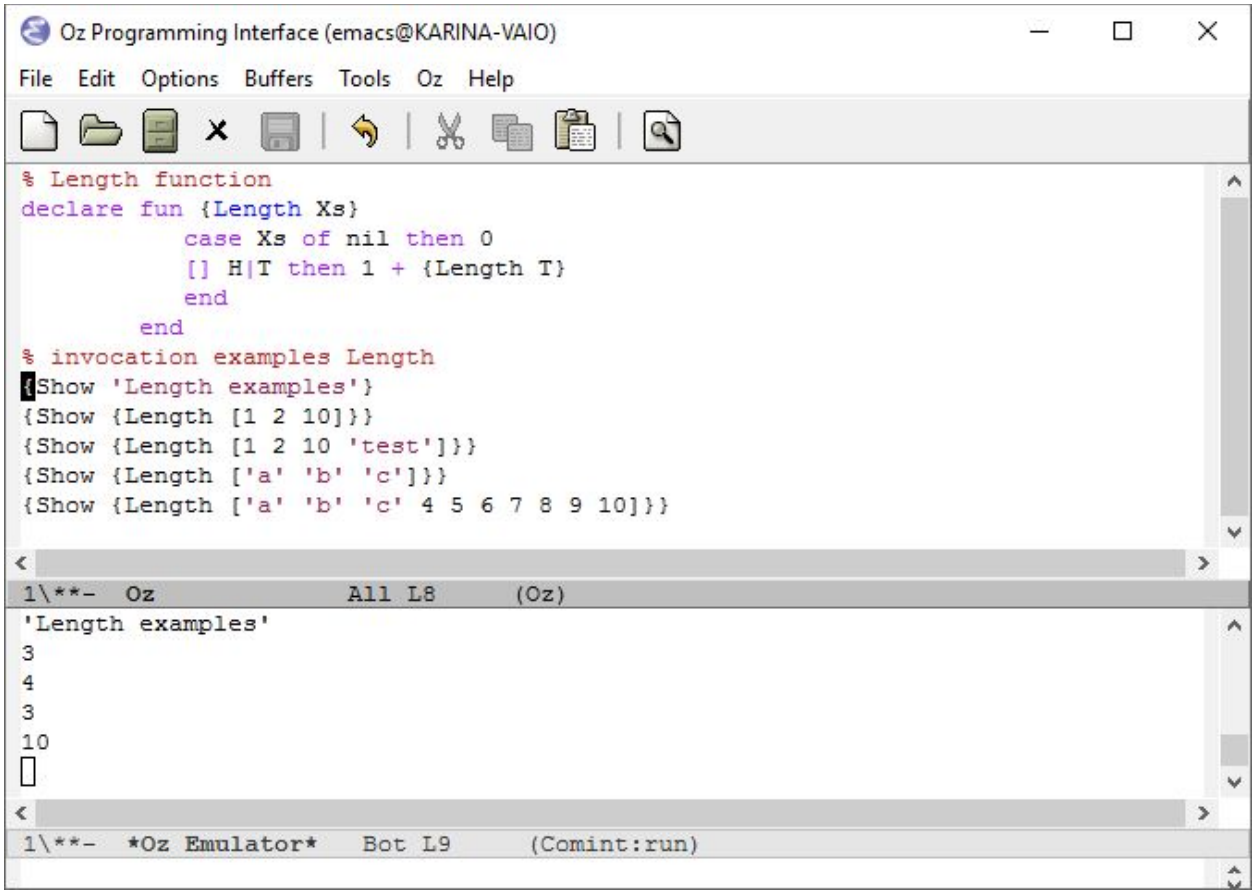
Ejercicio 1) - Procesamiento de listas

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

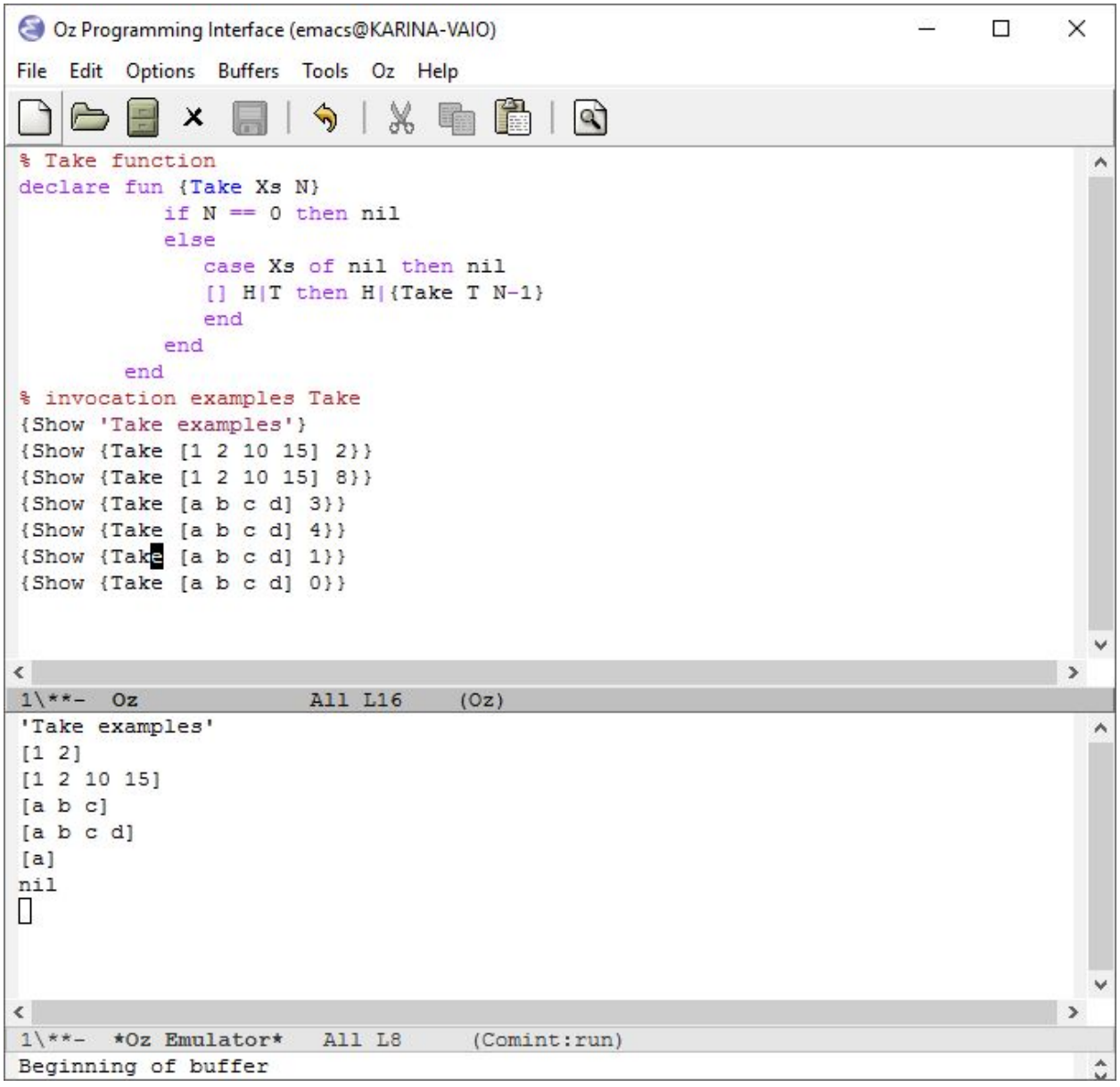


2. Take

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
```

b. Ejemplos de ejecución

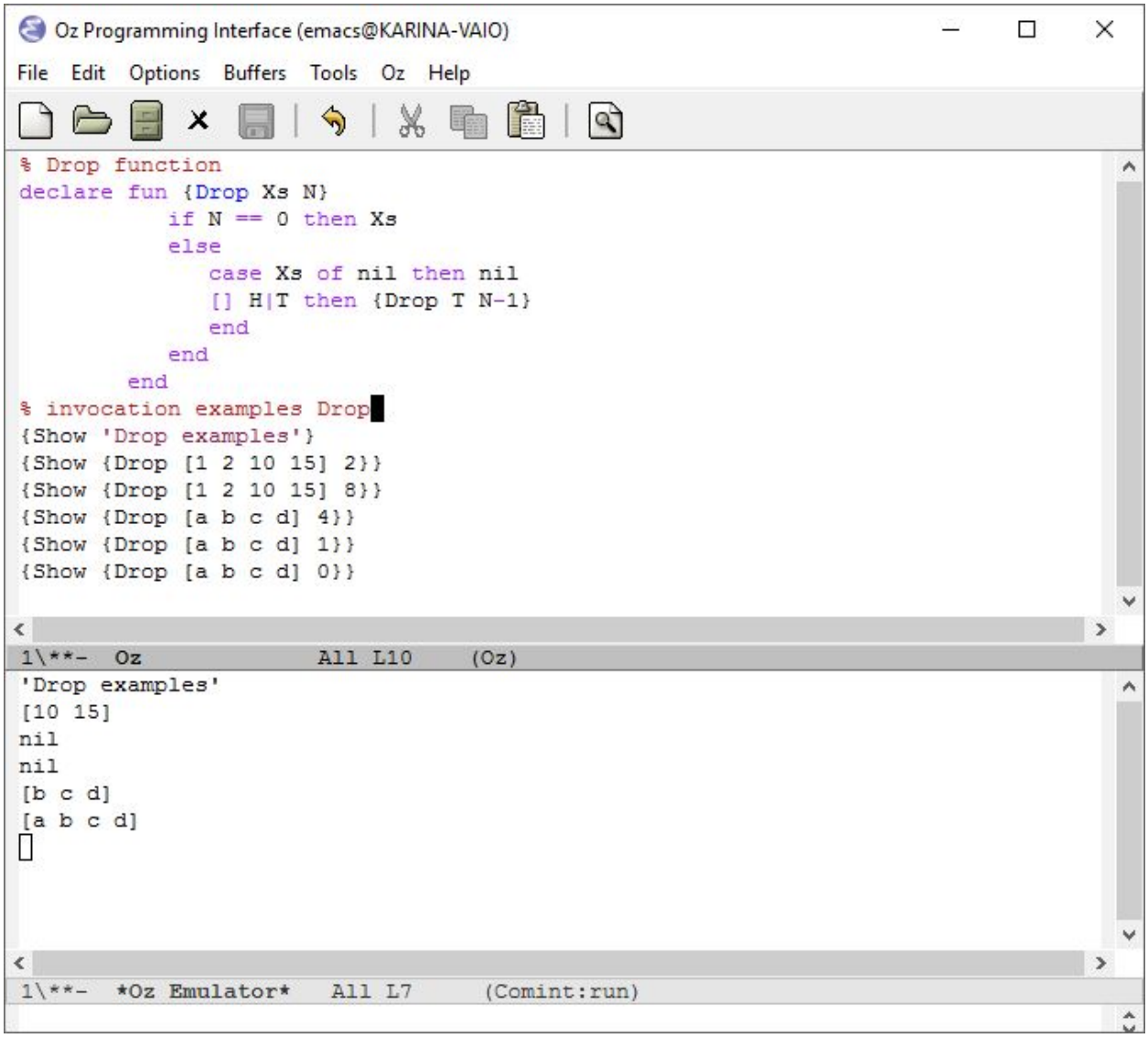


3. Drop

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
```

b. Ejemplos de ejecución

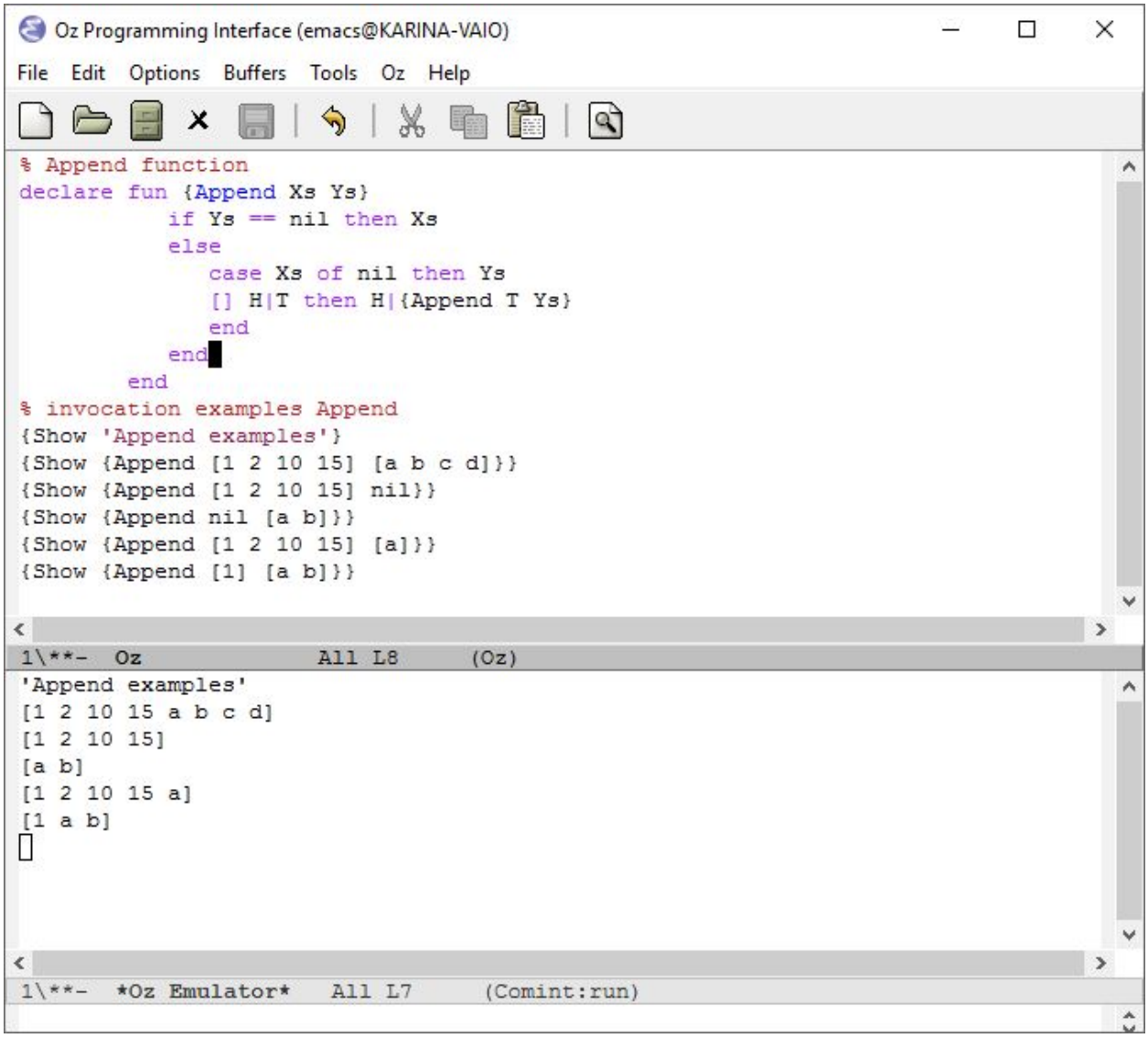


4. Append

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
```

b. Ejemplos de ejecución

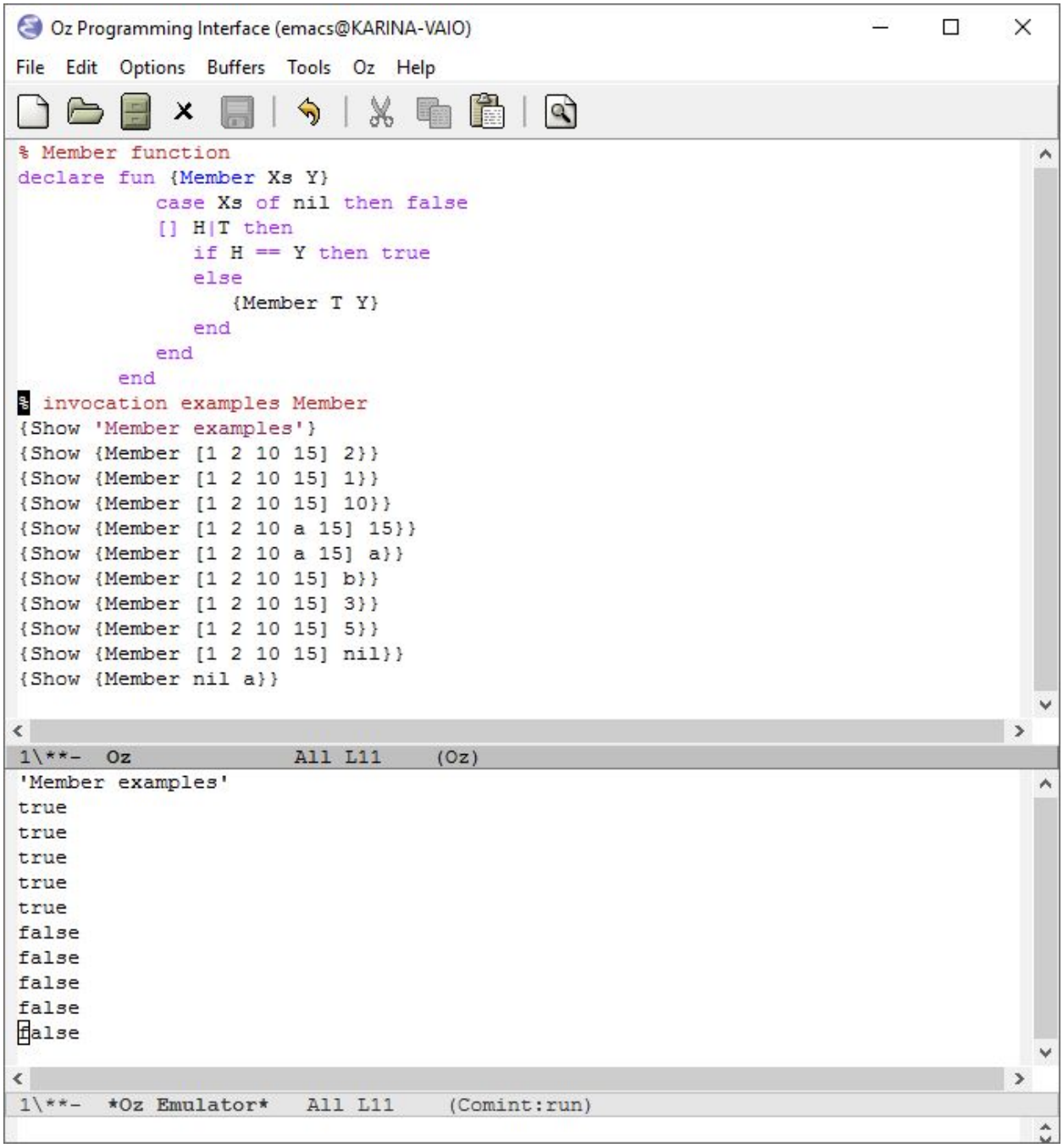


5. Member

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
```

b. Ejemplos de ejecución

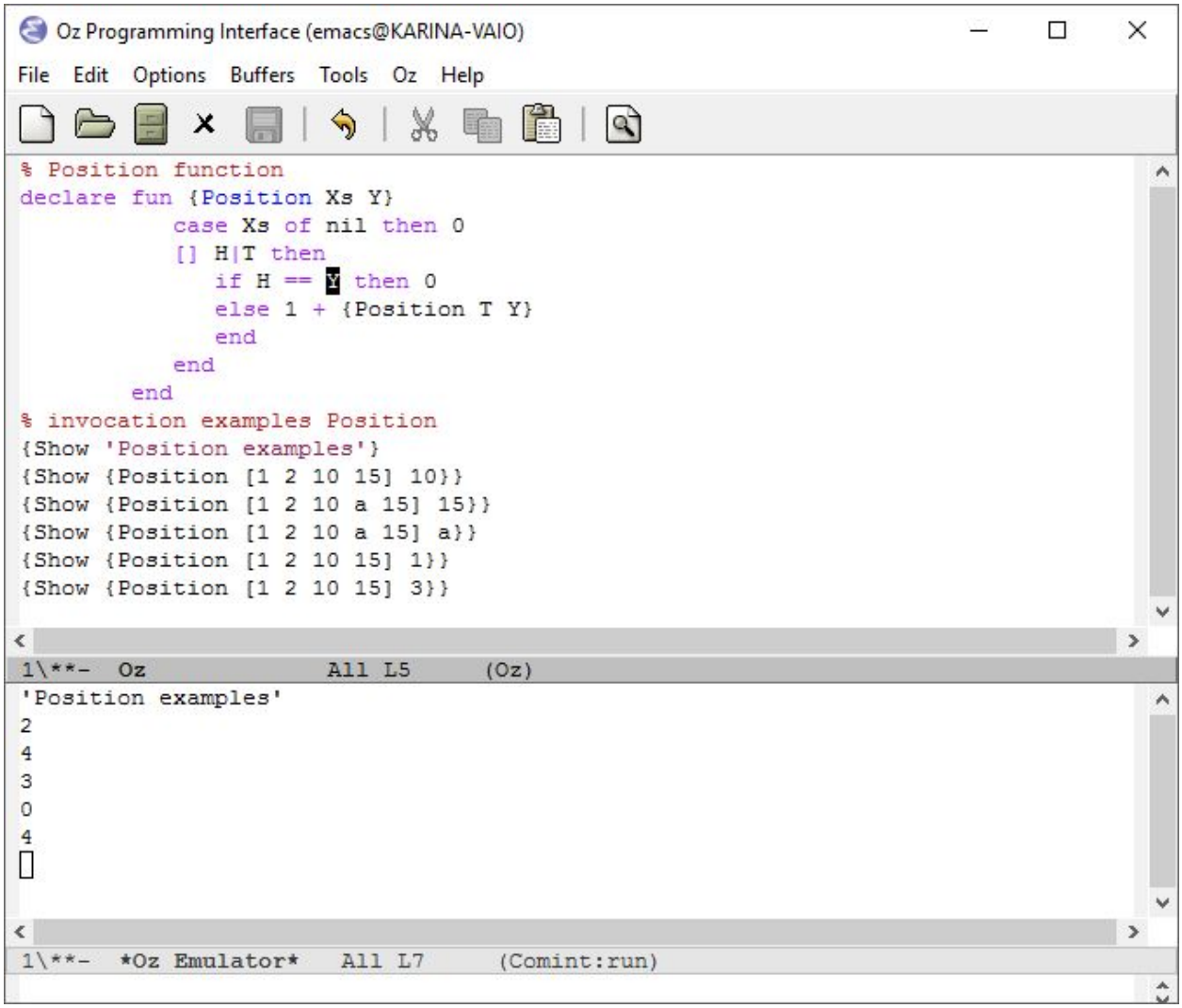

Oz Programming Interface (emacs@KARINA-VAIO)
File Edit Options Buffers Tools Oz Help
% Member function
declare fun {Member Xs Y}
case Xs of nil then false
[] H|T then
if H == Y then true
else
{Member T Y}
end
end
end
% invocation examples Member
{Show 'Member examples'}
{Show {Member [1 2 10 15] 2}}
{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

```
% 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
```

b. Ejemplos de ejecución



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**	**E**

if B then
skip
else
skip
end
	b1	**B-> b1**

=> 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**	**E**

local B in
B = false
if B then
skip
else
skip
end
end
	-	**-**

=> Ejecución de declaración de variable y composición

Stack	**Store**	**E**

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**	**E**
skip	b1 = false	B-> b1

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

Stack	**Store**	**E**
-	b1 = false	B-> b1

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

Programa 3

Estado inicial

Stack	**Store**	**E**
(local X Z A B P in proc {P X Y} Y = X+Z end Z=7 X=4 {P X A} {P A B} end, E)	-	**-**

=> 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} Y = X+Z end, CE = {Z->z1})	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 end, CE = {Z->z1})	X -> x1 Z-> z1 A -> a1 B -> b1 P-> p1
		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
		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 end, CE = {Z->z1})	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**

=> Asignación de suma X + Z

Stack	**Store**	**E'**
-	x1 = 4 z1 = 7 a1 = 11 **b1 = 18** p1 = (proc {$ X Y} Y = X+Z end, CE = {Z->z1})	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**	**E**
local X Z A B P in proc {P X Y} Y = X+Z end Z=10 local Z in Z = 2 X=4 {P X A} {P A B} end end	-	**-**

=> Ejecución de declaración de variables

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

=> Declaración de procedure value

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

=> Declaración de variable Z en un local

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

=> Ejecución Y = X + Z (en E'')

Stack	**Store**	**E**
({P A B}, E')	x1 = 4 z1 = 10 a1 = **14** b1 p1 = proc {$ X Y} Y = X+Z end, CE = {Z -> z1} z2 = 2	Sin cambios
		E'
		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''')	x1 = 4 z1 = 10 a1 = 14 b1 p1 = proc {$ X Y} Y = X+Z end, CE = {Z -> z1} z2 = 2	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 z1 = 10 a1 = 14 b1 = **24** p1 = proc {$ X Y} Y = X+Z end, CE = {Z -> z1} z2 = 2	Sin cambios
		E'
		X - > x1 A -> a1 B -> b1 P -> p1 Z -> z2
		E''
		Sin cambios
		E'''
		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)** (Y=4**, E)** (proc {P A B} proc {B U V} local F in F=A+1 V=U+F end end end**, E)** ({P X Q}**, E)** ({Q Y Z}**, E)**	**x1** **y1** **z1** **p1** **q1**	**X -> x1** **Y -> y1** **Z -> z1** **P -> p1** **Q -> q1**

=> Asignación de variable X = 6

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

=> Asignación de variable Y = 4

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

=> Declaración procedure value

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

	V=U+F end end, CE2 = {A->x1}) **f1**	U->y1 V->z1 **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 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**	**E**
-	x1 = 6 y1 = 4 z1 = **11** 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 end, CE ={}) q1 = (proc {$ U V} local F in F=A+1 V=U+F end end, CE2 = {A->x1}) f1 = 7	**E'** A->x1 B->q1 **E''** A->x1 U->y1 V->z1 **E'''** A->x1 U->y1 V->z1 F-> f1

Fin del programa

Programa 6

Estado inicial

Stack	**Store**	**E**
local X Y Z in X = Y try X = 1 Y = 2 Z = 3 catch Exception then skip end {Browse X#Y#Z} end	-	**-**

=> 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)**		

=> Asignación de variables

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

=> 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**	**E**
(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**	**E**
({Browse X#Y#Z}, E)	x1 = y1 = 1 z1 e1 = Invalid...	X -> x1 Y -> y1 Z -> z1
		E'
		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
		E'
		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 a|a}

Predicción: 'case'(1)

Ejecución: 'case'(1)

Es una lista que comienza con el valor a.

{Test 'l'(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 'l'(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 'l'(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 'l'(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}`
`  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`
	-	-

=> Declaración de variable Length

Stack	Store	E

`(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`
	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)**	**l1 = (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 -> l1}**)**	**Length -> l1**

=> 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 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 -> l1}) **k1**	Length -> l1 **E' = E + {K->k1}** **Length -> l1** **K-> k1**

=> Ejecución de procedure value

Stack	Store	E
(case Xs of nil then N = 0 else case Xs of _|T then	l1 = (proc {$ Xs N} case Xs of nil then N = 0	Length -> l1 **E'**

local U in {Length T U} N = U + 1 end else skip end, **E'')** **({Show K}, E')**	else case Xs of _|T then local U in {Length T U} N = U + 1 end else skip end end end, CE={Length -> l1}) **k1 =n1** **xs1 = [1 2 3 4]**	Length -> l1 K-> k1 **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')**	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4]	Length -> l1 **E'** Length -> l1 K-> k1 **E''** Xs-> xs1 N-> n1

=> Ejecución de case

Stack	Store	E
(local U in {Length T U} N = U + 1 end, **E'')** **({Show K}, E')**	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4]	Length -> l1 **E'** Length -> l1 K-> k1 **E''** Xs-> xs1 N-> n1 **E'''** **Xs-> xs1** **N-> n1** **T-> t1**

=> Declaración de variable local U

Stack	Store	E
({Length T U}, **E””)** (N = U + 1, **E””)** **({Show K}, E’)**	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] **u1**	Length -> l1 **E’** Length -> l1 K-> k1 **E”** Length -> l1 Xs-> xs1 N-> n1 **E”’** Length -> l1 Xs-> xs1 N-> n1 T-> t1 **E””** **Length -> l1** **Xs-> xs1** **N-> n1** **T-> t1** **U->u1**

=> Ejecución stored procedure

Stack	Store	E
(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, **E5)** (N = U + 1, **E’’’’)** **({Show K}, E’)**	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] **u1 = n2** **xs2 = [2 3 4]**	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 = CE + {Xs->E’’’’(T); N->E’’’’(U)}		
Length -> l1 Xs-> xs2 N-> n2		

=> Ejecución case

Stack	**Store**	**E**
(case Xs of _|T then local U in {Length T U} N = U + 1 end else skip end, **E5)** (N = U + 1, **E’’’’)** **({Show K}, E’)**	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4]	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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		
Length -> l1 Xs-> xs2 N-> n2		

=> Ejecución case

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr><td rowspan="10">(local U in
{Length T U}
N = U + 1
end, E6)
(N = U + 1, E’’’’)
({Show K}, E’)</td><td rowspan="10">l1 = (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 -> l1})
k1 = n1
xs1 = [1 2 3 4]
t1 = [2 3 4]
u1 = n2
xs2 = [2 3 4]
t2 = [3 4]</td><td>Length -> l1</td></tr>
<tr><td>E’</td></tr>
<tr><td>Length -> l1
K-> k1</td></tr>
<tr><td>E’’</td></tr>
<tr><td>Length -> l1
K-> k1
Xs-> xs1
N-> n1</td></tr>
<tr><td>E’’’</td></tr>
<tr><td>Length -> l1
Xs-> xs1
N-> n1
T-> t1</td></tr>
<tr><td>E’’’’</td></tr>
<tr><td>Length -> l1
Xs-> xs1
N-> n1
T-> t1
U->u1</td></tr>
<tr><td></td></tr>
<tr><td>E5</td><td>E6</td><td></td></tr>
<tr><td>Length -> l1
Xs-> xs2
N-> n2</td><td>Length -> l1
Xs-> xs2
N-> n2
T-> t2</td><td></td></tr>
</table>

=> Declaración variable local U

Stack	**Store**	**E**
({Length T U}, **E7)** (N = U + 1, **E7)** (N = U + 1, E’’’’) ({Show K}, E’)	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] **u2**	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	**Length -> l1** **Xs-> xs2** **N-> n2** **T-> t2** **U->u2**

=> Ejecución de procedure value

Stack	**Store**	**E**
(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, **E8)** (N = U + 1, **E7)** (N = U + 1, E’’’’) ({Show K}, E’)	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] **u2 =n3**	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8 = CE + {Xs->E7(T) ; N->E7(U)}		
Length -> l1 Xs->t2 N->n3		

=> Ejecución de case

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’)	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 =n3	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8		
Length -> l1 Xs->t2 N->n3		

=> Ejecución de case

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’)	l1 = (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 -> l1}) k1 = n1 xs1 = [1 2 3 4] t1 = [2 3 4] u1 = n2 xs2 = [2 3 4] t2 = [3 4] u2 =n3 **t3 = [4]**	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	
Length -> l1 Xs->t2 N->n3	**Length -> l1** **Xs->t2** **N->n3** **T-> t3**	

=> Declaración de local U

Stack	**Store**	**E**
({Length T U}, **E10)** (N = U + 1, **E10)** (N = U + 1, E7) (N = U + 1, E’’’’) ({Show K}, E’)	l1 = (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 -> l1}) 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 -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	**Length -> l1** **Xs->t2** **N->n3** **T-> t3** **U->u3**

=> Ejecución de procedure value

Stack	**Store**	**E**
(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**, E11)** (N = U + 1, E10) (N = U + 1, E7) (N = U + 1, E’’’’) ({Show K}, E’)	l1 = (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 -> l1}) 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**	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	Length -> l1 Xs->t2 N->n3 T-> t3 U->u3
E11 = CE + {Xs->E10(T); N->E10(U)}		
Length-> l1 Xs->t3 N->n4		

=> Ejecución de case

Stack	Store	E
(case Xs of _|T then **local U in** **{Length T U}** **N = U + 1** **end** **else** **skip** **end, E11)** (N = U + 1, E10) (N = U + 1, E7) (N = U + 1, E’’’’) ({Show K}, E’)	l1 = (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 -> l1}) 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	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	Length -> l1 Xs->t2 N->n3 T-> t3 U->u3
E11		
Length-> l1 Xs->t3 N->n4		

=> Ejecución de case

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’)	l1 = (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 -> l1}) 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**	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	Length -> l1 Xs->t2 N->n3 T-> t3 U->u3
E11	**E12**	
Length-> l1 Xs->t3 N->n4	**Length-> l1** **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, E7) (N = U + 1, E’’’’) ({Show K}, E’)	l1 = (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 -> l1}) 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 **u4**	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	Length -> l1 Xs->t2 N->n3 T-> t3 U->u3
E11	**E12**	**E13**
Length-> l1 Xs->t3 N->n4	Length-> l1 Xs->t3 N->n4 T-> t4	**Length-> l1** **Xs->t3** **N->n4** **T-> t4** **U->u4**

=> Ejecucion procedure value

Stack	Store	E
(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, E14)** (N = U + 1, E13) (N = U + 1, E10) (N = U + 1, E7) (N = U + 1, E’’’’) ({Show K}, E’)	l1 = (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 -> l1}) 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 **u4 =n5**	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	Length -> l1 Xs->t2 N->n3 T-> t3 U->u3
E11	**E12**	**E13**
Length-> l1 Xs->t3 N->n4	Length-> l1 Xs->t3 N->n4 T-> t4	Length-> l1 Xs->t3 N->n4 T-> t4 U->u4

E14= CE + {Xs->E13(T); N->E13(U)}		
Length-> l1 **Xs->t4** **N->n5**		

=> Ejecución case

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’)	l1 = (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 -> l1}) 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 u4 =n5	Length -> l1 **E’** Length -> l1 K-> k1 **E’’** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	Length -> l1 Xs->t2 N->n3 T-> t3 U->u3
E11	**E12**	**E13**

Length-> l1 Xs->t3 N->n4	Length-> l1 Xs->t3 N->n4 T-> t4	Length-> l1 Xs->t3 N->n4 T-> t4 U->u4
E14		
Length-> l1 Xs->t4 N->n5		

=> Asignación N =0

Stack	**Store**	**E**
(N = U + 1, E13) (N = U + 1, E10) (N = U + 1, E7) (N = U + 1, E'''') ({Show K}, E')	l1 = (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 -> l1}) 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 **u4 =n5 = 0**	Length -> l1 **E'** Length -> l1 K-> k1 **E''** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2	Length -> l1 Xs->t2	Length -> l1 Xs->t2

N->n3	N->n3 T-> t3	N->n3 T-> t3 U->u3
E11	**E12**	**E13**
Length-> l1 Xs->t3 N->n4	Length-> l1 Xs->t3 N->n4 T-> t4	Length-> l1 Xs->t3 N->n4 T-> t4 U->u4
E14		
Length-> l1 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')	l1 = (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 -> l1}) 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 = 1** t4 =nil u4 =n5 = 0	Length -> l1 **E'** Length -> l1 K-> k1 **E''** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	Length -> l1 Xs->t2 N->n3 T-> t3 U->u3
E11	**E12**	**E13**
Length-> l1 Xs->t3 N->n4	Length-> l1 Xs->t3 N->n4 T-> t4	Length-> l1 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')	l1 = (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 -> l1}) 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 -> l1
		E'
		Length -> l1 K-> k1
		E''
		Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2
E8	**E9**	**E10**
Length -> l1 Xs->t2 N->n3	Length -> l1 Xs->t2 N->n3 T-> t3	Length -> l1 Xs->t2 N->n3 T-> t3 U->u3

=> Asignación N =U+1 en E7

Stack	Store	E
(N = U + 1, E'''') ({Show K}, E')	l1 = (proc {$ Xs N} case Xs of nil then N = 0 else	Length -> l1
		E'

	case Xs of _|T then local U in {Length T U} N = U + 1 end else skip end end end, CE={Length -> l1}) 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	Length -> l1 K-> k1 **E''** Length -> l1 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 -> l1 Xs-> xs2 N-> n2	Length -> l1 Xs-> xs2 N-> n2 T-> t2	Length -> l1 Xs-> xs2 N-> n2 T-> t2 U->u2

=> Asignación N =U+1 en E''''

Stack	**Store**	**E**
({Show K}, E')	l1 = (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 -> l1}) **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	Length -> l1 **E'** Length -> l1 K-> k1 **E''** Length -> l1 K-> k1 Xs-> xs1 N-> n1 **E'''** Length -> l1 Xs-> xs1 N-> n1 T-> t1

	t3 = [4] u3 = n4 = 1 t4 =nil u4 =n5 = 0	**E’’’’** Length -> l1 Xs-> xs1 N-> n1 T-> t1 U->u1

=> Ejecución de Show K

Stack	**Store**	**E**
({Show K}, E’)	l1 = (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 -> l1}) 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 -> l1 **E’** Length -> l1 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}
  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
	-	-

=> Declaración de variable

Stack	Store	E

(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, **E)**
(local K in
  {Length [1 2 3 4] 0 K}
  {Show K}
end, **E)**
	l1	**Length -> l1**

=> Asignación de procedure value

Stack	Store	E
(local K in {Length [1 2 3 4] 0 K} {Show K} end**, E)**	**l1= (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->l1})**	Length -> l1

=> Declaración local K

Stack	Store	E
({Length [1 2 3 4] 0 K}, E1) **({Show K}, E1)**	l1= (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->l1}) **k1**	Length -> l1 **E1** **Length -> l1** **K->k1**

=> Ejecución procedure value

Stack	Store	E
(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, E2)** ({Show K}, E1)	l1= (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->l1}) **k1 = n1** **xs1 = [1 2 3 4]** **a1 = 0**	Length -> l1 **E1** Length -> l1 K->k1 **E2** **Length-> l1** **Xs->xs1** **A->a1** **N->n1**

=> Ejecución case

Stack	Store	E
(case Xs of _|T then **local X in** **X = A + 1** **{Length T X N}** **end** **else** **skip** **end, E2)** ({Show K}, E1)	l1= (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->l1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0	Length -> l1 **E1** Length -> l1 K->k1 **E2** Length-> l1 Xs->xs1 A->a1 N->n1

=> Ejecución case

Stack	**Store**	**E**
local X in **X = A + 1** **{Length T X N}** **end, E3)** ({Show K}, E1)	l1= (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->l1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 **t1 = [2 3 4]**	Length -> l1 **E1** Length -> l1 K->k1 **E2** Length-> l1 Xs->xs1 A->a1 N->n1 **E3** **Length-> l1** **Xs->xs1** **A->a1** **N->n1** **T->t1**

=> Declaración local X

Stack	**Store**	**E**
(X = A + 1, E4) **({Length T X N}, E4)** ({Show K}, E1)	l1= (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->l1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] **x1**	Length -> l1 **E1** Length -> l1 K->k1 **E2** Length-> l1 Xs->xs1 A->a1 N->n1 **E3** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 **E4** **Length-> l1** **Xs->xs1** **A->a1** **N->n1** **T->t1** **X-> x1**

=> Suma X = A+1 en E4

Stack	Store	E
({Length T X N}, E4) ({Show K}, E1)	l1= (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->l1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] **x1 = 1**	Length -> l1 **E1** Length -> l1 K->k1 **E2** Length-> l1 Xs->xs1 A->a1 N->n1 **E3** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 **E4** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1

=> Aplicación de procedure value

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr><td rowspan="10">(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, E5)
({Show K}, E1)</td><td rowspan="10">l1= (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->l1})
k1 = n1
xs1 = [1 2 3 4]
a1 = 0
t1 = [2 3 4]
x1 = 1</td><td>Length -> l1</td></tr>
<tr><td>E1</td></tr>
<tr><td>Length -> l1
K->k1</td></tr>
<tr><td>E2</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1</td></tr>
<tr><td>E3</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1
T->t1</td></tr>
<tr><td>E4</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1
T->t1
X-> x1</td></tr>
<tr><td>E5</td><td></td><td></td></tr>
<tr><td>Length->l1
Xs-> t1
A-> x1
N-> n1</td><td></td><td></td></tr>
</table>

=> Ejecución case

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

=> Ejecución case

Stack	Store	E
(local X in **X = A + 1** **{Length T X N}** **end, E6)** ({Show K}, E1)	l1= (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	Length -> l1 **E1** Length -> l1 K->k1 **E2** Length-> l1 Xs->xs1

	end end end, CE={Length->l1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1 **t2 = [3 4]**	A->a1 N->n1
		E3
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	**E6**	
Length->l1 Xs-> t1 A-> x1 N-> n1	**Length->l1** **Xs-> t1** **A-> x1** **N-> n1** **T-> t2**	

=> Declaración local X

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr><td rowspan="10">(X = A + 1, E7)
({Length T X N}, E7)
({Show K}, E1)</td><td rowspan="10">l1= (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->l1})
k1 = n1
xs1 = [1 2 3 4]
a1 = 0
t1 = [2 3 4]
x1 = 1
t2 = [3 4]
x2</td><td>Length -> l1</td></tr>
<tr><td>E1</td></tr>
<tr><td>Length -> l1
K->k1</td></tr>
<tr><td>E2</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1</td></tr>
<tr><td>E3</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1
T->t1</td></tr>
<tr><td>E4</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1
T->t1
X-> x1</td></tr>
<tr><td></td></tr>
<tr><td>E5</td><td>E6</td><td>E7</td></tr>
<tr><td>Length->l1
Xs-> t1
A-> x1
N-> n1</td><td>Length->l1
Xs-> t1
A-> x1
N-> n1
T-> t2</td><td>Length->l1
Xs-> t1
A-> x1
N-> n1
T-> t2
X->x2</td></tr>
</table>

=> Ejecución X = A +1 en E7

Stack	**Store**	**E**
({Length T X N}, E7) ({Show K}, E1)	l1= (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->l1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] **x2 = 2**	Length -> l1
		E1
		Length -> l1 K->k1
		E2
		Length-> l1 Xs->xs1 A->a1 N->n1
		E3
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	**E6**	**E7**
Length->l1 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 procedure value

Stack	Store	E
(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, E8)** ({Show K}, E1)	l1= (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->l1}) k1 = n1 xs1 = [1 2 3 4] a1 = 0 t1 = [2 3 4] x1 = 1 t2 = [3 4] x2 = 2	Length -> l1
		E1
		Length -> l1 K->k1
		E2
		Length-> l1 Xs->xs1 A->a1 N->n1
		E3
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	**E6**	**E7**
Length->l1 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**		

=> Ejecución case

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

=> Ejecución case

<table>
<tr><th>Stack</th><th>Store</th><th>E</th></tr>
<tr><td rowspan="10">(local X in
X = A + 1
{Length T X N}
end, E9)
({Show K}, E1)</td><td rowspan="10">l1= (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->l1})
k1 = n1
xs1 = [1 2 3 4]
a1 = 0
t1 = [2 3 4]
x1 = 1
t2 = [3 4]
x2 = 2
t3 = [4]</td><td>Length -> l1</td></tr>
<tr><td>E1</td></tr>
<tr><td>Length -> l1
K->k1</td></tr>
<tr><td>E2</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1</td></tr>
<tr><td>E3</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1
T->t1</td></tr>
<tr><td>E4</td></tr>
<tr><td>Length-> l1
Xs->xs1
A->a1
N->n1
T->t1
X-> x1</td></tr>
<tr><td>E5</td><td>E6</td><td>E7</td></tr>
<tr><td>Length->l1
Xs-> t1
A-> x1
N-> n1</td><td>Length->l1
Xs-> t1
A-> x1
N-> n1
T-> t2</td><td>Length->l1
Xs-> t1
A-> x1
N-> n1
T-> t2
X->x2</td></tr>
<tr><td>E8</td><td>E9</td><td></td></tr>
<tr><td>Length->l1
Xs-> t2
A->x2
N->n1</td><td>Length->l1
Xs-> t2
A->x2
N->n1
T->t3</td><td></td></tr>
</table>

=> Declaración local X

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

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

=> Declaración local X

Stack	Store	E
(X = A + 1, E13) **({Length T X N}, E13)** ({Show K}, E1)	l1= (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->l1}) 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 **x4**	Length -> l1 **E1** Length -> l1 K->k1 **E2** Length-> l1 Xs->xs1 A->a1 N->n1 **E3** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 **E4** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	**E6**	**E7**
Length->l1 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->l1 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	**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)	l1= (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->l1}) 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 **x4 = 4**	Length -> l1 **E1** Length -> l1 K->k1 **E2** Length-> l1 Xs->xs1 A->a1 N->n1 **E3** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 **E4** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	**E6**	**E7**
Length->l1 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->l1 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	**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

=> Ejecución procedure value

Stack	Store	E
(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, E14)** ({Show K}, E1)	l1= (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->l1}) 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 x4 = 4	Length -> l1 **E1** Length -> l1 K->k1 **E2** Length-> l1 Xs->xs1 A->a1 N->n1 **E3** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 **E4** Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	**E6**	**E7**
Length->l1 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->l1 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	**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 case

Stack	**Store**	**E**
(N = A, E14) ({Show K}, E1)	l1= (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->l1}) 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 x4 = 4	Length -> l1
		E1
		Length -> l1 K->k1
		E2
		Length-> l1 Xs->xs1 A->a1 N->n1
		E3
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	**E6**	**E7**
Length->l1 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->l1 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)	l1= (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->l1}) **k1 = n1 = 4** 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	Length -> l1
		E1
		Length -> l1 K->k1
		E2
		Length-> l1 Xs->xs1 A->a1 N->n1
		E3
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1
		E4
		Length-> l1 Xs->xs1 A->a1 N->n1 T->t1 X-> x1
E5	**E6**	**E7**
Length->l1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1	Length->l1 Xs-> t1 A-> x1 N-> n1

	T-> t2	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->l1 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		
Length->l1 Xs->t4 A->x4 N->n1		

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

Stack	**Store**	**E**
-	l1= (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->l1}) k1 = n1 = 4 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	Length -> l1 **E1** Length -> l1 K->k1

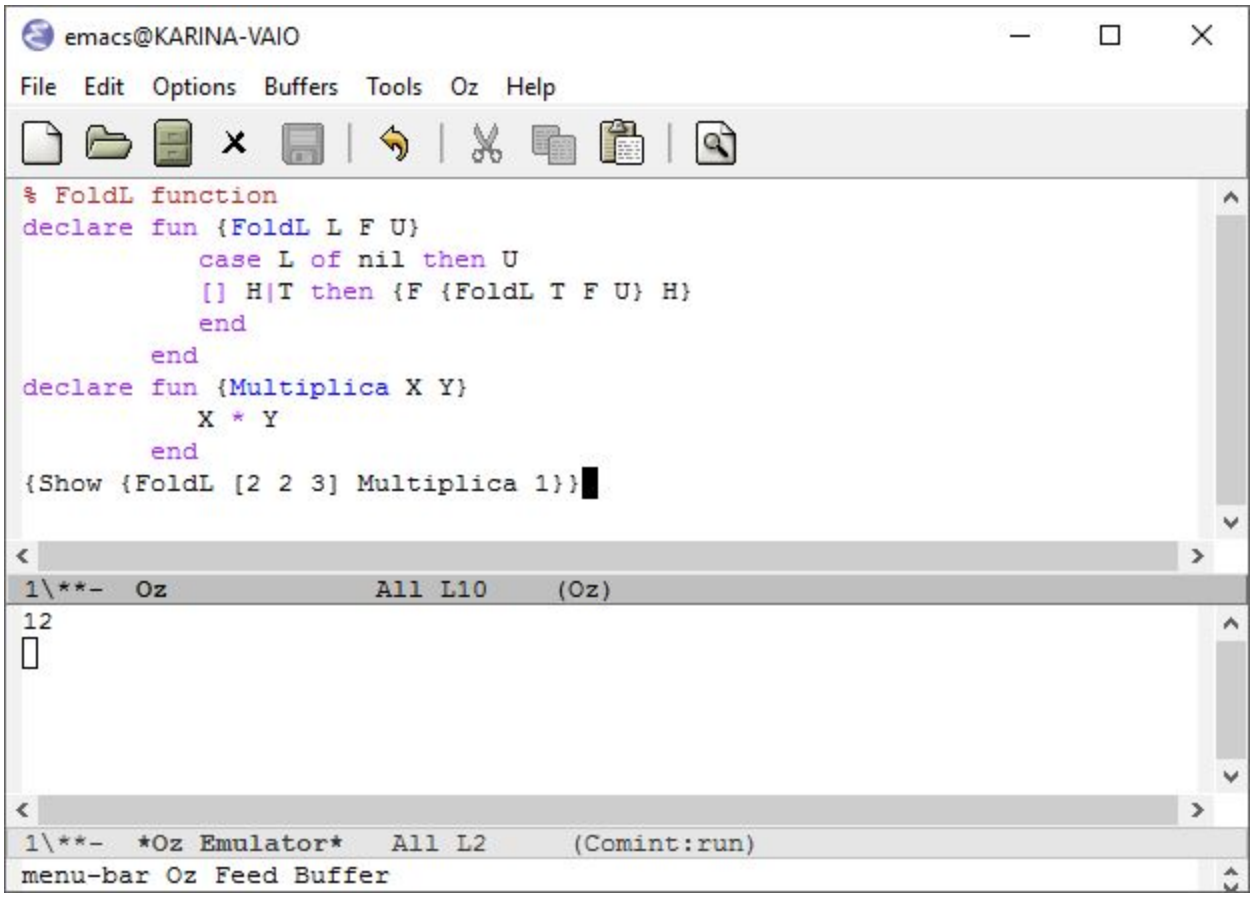
Ejercicio 6) - Alto orden con listas

FoldL

a. Código Oz

```
% 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}}
```

b. Ejemplo de ejecución

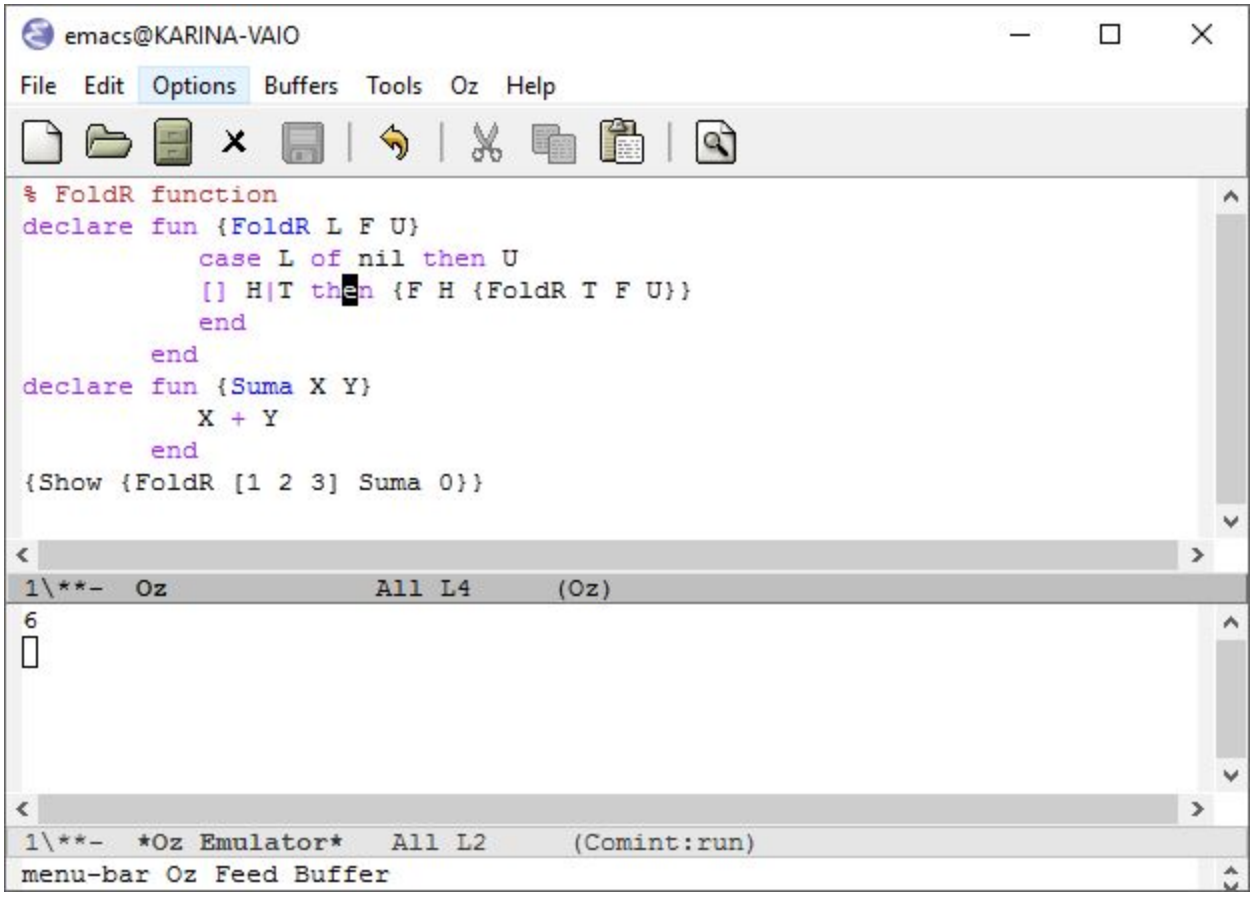


FoldR

a. Código Oz

```
% FoldR function
declare fun {FoldR L F U}
         case L of nil then U
         [] H|T then {F H {FoldR T F U}}
         end
      end
declare fun {Suma X Y}
         X + Y
      end
{Show {FoldR [1 2 3] Suma 0}}
```

b. Ejemplo de ejecución

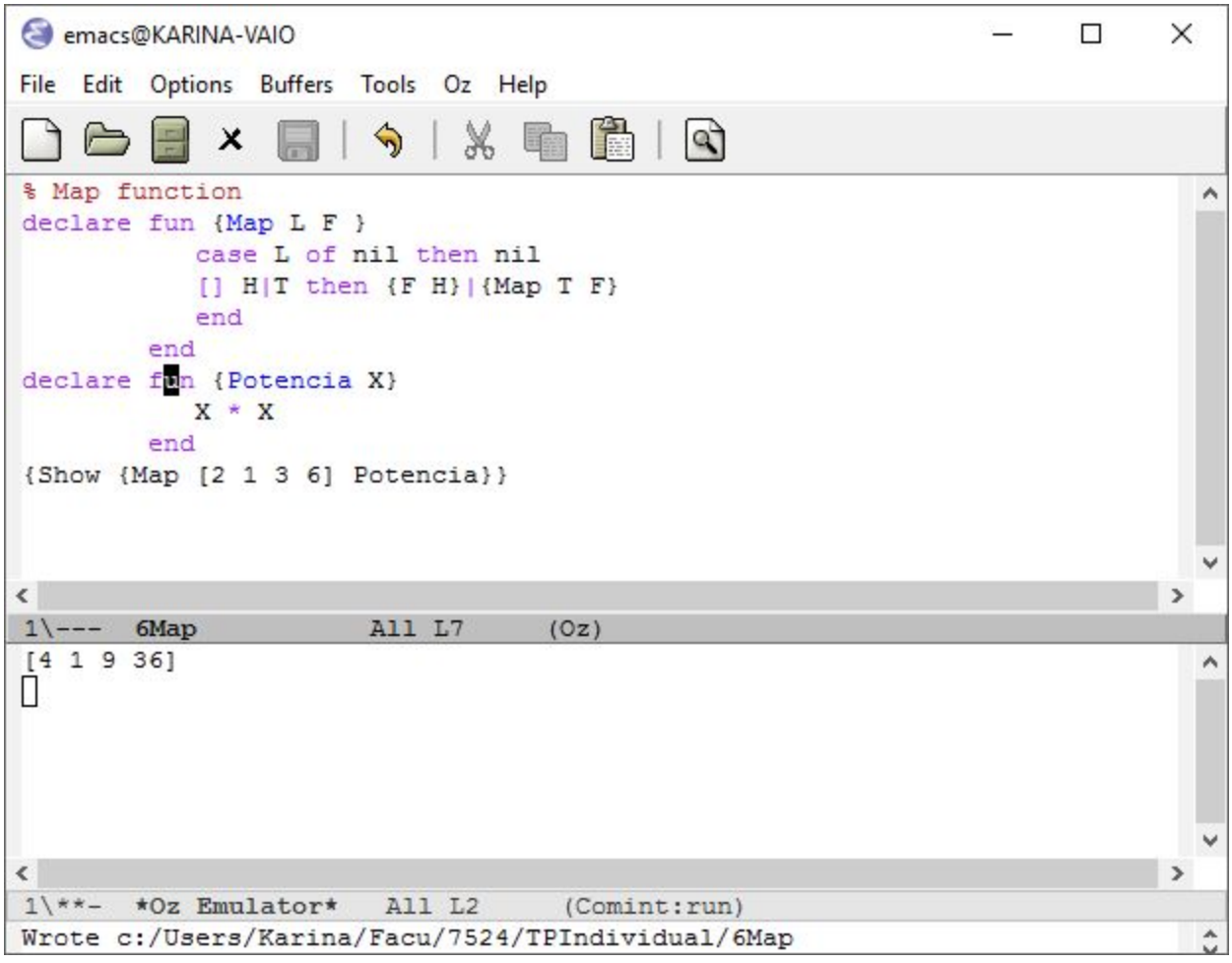


Map

a. Código Oz

```
% Map function
declare fun {Map L F }
         case L of nil then nil
         [] H|T then {F H}|{Map T F}
         end
      end
declare fun {Potencia X}
         X * X
      end
{Show {Map [2 1 3 6] Potencia}}
```

b. Ejemplo de ejecución

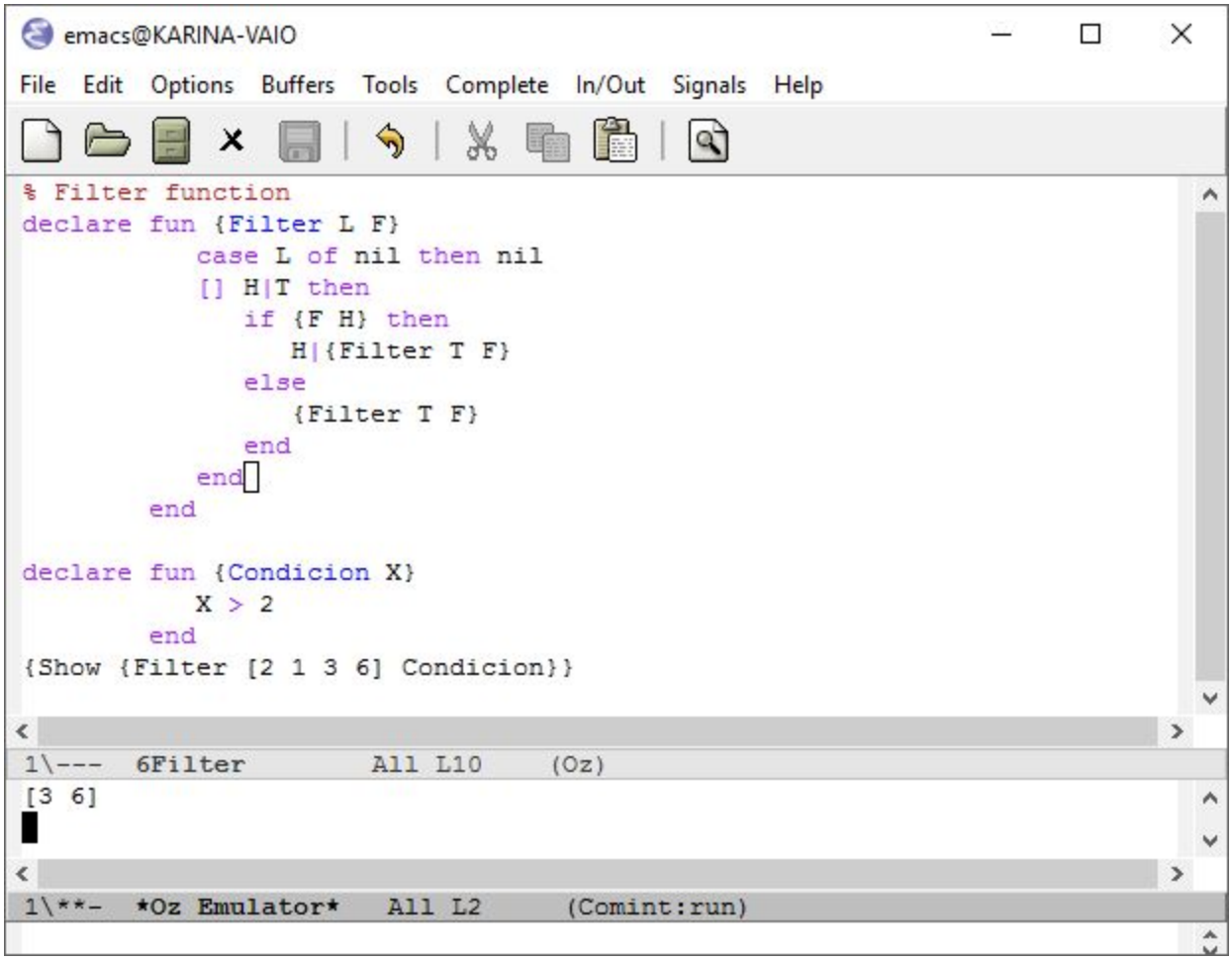


Filter

a. Código Oz

```
% 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}}
```

b. Ejemplo de ejecución



Ejercicio 7) - Hilos

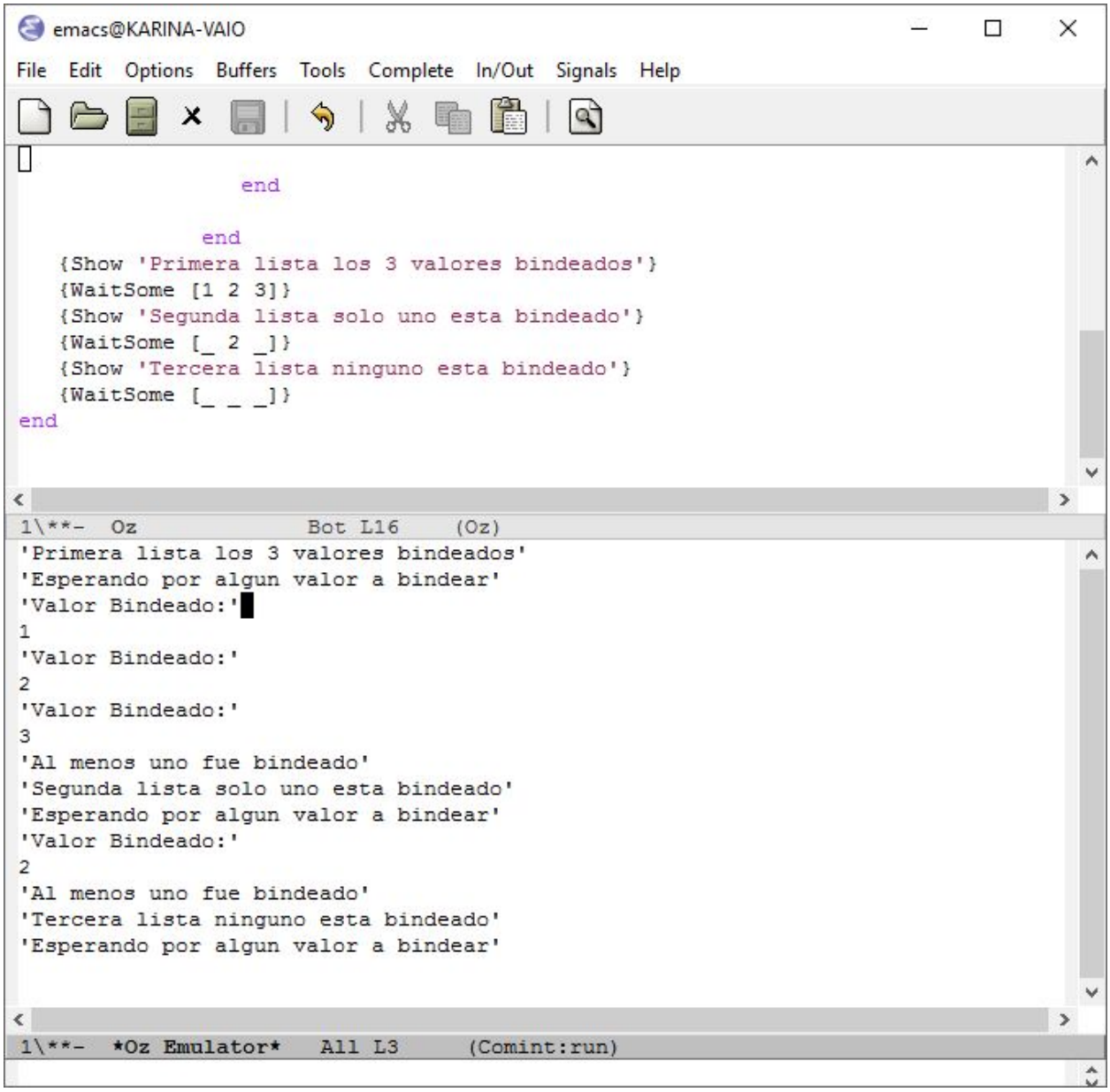
7.1 WaitSome

a. Código Oz

```
local WaitSome in
   WaitSome = proc {$ Xs}
              local Y in
                 {ForAll Xs
                  proc {$ X}
                   thread {Wait X}
                      Y = true
                      {Show 'Valor Bindeado:'}
                      {Show X}
                   end
```

```
                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 [_ _ _]}
end
```

b. Ejemplo de ejecución


emacs@KARINA-VAIO
File Edit Options Buffers Tools Complete In/Out Signals Help
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 [_ _ _]}
end
1**- Oz Bot L16 (Oz)
'Primera lista los 3 valores bindeados'
'Esperando por algun valor a bindear'
'Valor Bindeado:'
1
'Valor Bindeado:'
2
'Valor Bindeado:'
3
'Al menos uno fue bindeado'
'Segunda lista solo uno esta bindeado'
'Esperando por algun valor a bindear'
'Valor Bindeado:'
2
'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
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
if A then
B=true
else
B=false
end
end, E)
(thread
if B then
C=false
else
C=true
end
end, E)
(A=false, E)
	a1
	b1
	c1
		A-> a1
		B-> b1
		C-> c1

=> 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
-	-	**-**	a1 = false b1 = false **c1 = true**	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 = N|L3
                                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 = N|L3 end 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** **proc {$ L2}** **local L3 in** **{Ints N+1 L3}** **L2 = N|L3** **end** **end** **L1}** **end, E)** **({Ints 1 L}, E)** **(A = L.2.2.1, E)** **(B = L.1, E)** **(C = A+B, E)** **({Browse C}, E)**	**i1** **l0** **a1** **b1** **c1**	**Ints-> i1** **L->l0** **A-> a1** **B-> b1** **C-> c1**

=> 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 = N|L3** **end** **end** **L1}** **end, CE={Ints->i1})** l0 a1 b1 c1	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1

=> Ejecución procedure value

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

=> 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 = N|L3** **end, E2)**	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=l1 a1 b1 c1 n1 = 1	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1 **E1** Ints-> i1 N->n1 L1->l1 **E2** **Ints->i1** **N->n1** **L2->l2**

=> 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 = N|L3, E3)**	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=l1 a1 b1 c1 n1 = 1 **l3**	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1 **E1** Ints-> i1 N->n1 L1->l1 **E2** Ints->i1 N->n1 L2->l2 **E3** **Ints->i1** **N->n1** **L2->l2** **L3-> l3**

=> 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 = N|L3** **end** **end** **L1}, E4)** (L2 = N|L3, E3)	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=l1 a1 b1 c1 n1 = 1 l3 **n2 = 2**	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1
			E1
			Ints-> i1 N->n1 L1->l1
			E2
			Ints->i1 L2->l1
			E3
			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 = N|L3, E3)	**({proc {$ L2}** **local L3 in** **{Ints N+1 L3}** **L2 = N|L3** **end** **end, E4)**	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=l1 a1 b1 c1 n1 = 1 l3 n2 = 2	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1
				E1
				Ints-> i1 N->n1 L1->l1
				E2
				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 = N|L3, E3)	**(local L3 in** **{Ints N+1 L3}** **L2 = N|L3** **end, E5)**	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) **l0=l1** **l2** a1 b1 c1 n1 = 1 l3 n2 = 2	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1
				E1
				Ints-> i1 N->n1 L1->l1
				E2
				Ints->i1 L2->l1
				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 = N|L3, 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 = N|L3 end end L1} end, CE={Ints->i1}) **I0=I1** **I2** **I3** a1 b1 c1 n1 = 1 n2 = 2	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
				E1
				Ints-> i1 N->n1 L1->I1
				E2
				Ints->i1 L2->I1
				E3
				Ints->i1 N->n1 L2->I2 L3-> I3
E4	**E5**	**E6**		
Ints->i1 N->n2 L1->I3	Ints->i1 N->n2 L2->I2	Ints->i1 N->n2 L2->I2 L3-> I3		

=> 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 = N|L3, E3)	**({ByNeed** **proc {$ L2}** **local L3 in** **{Ints N+1 L3}** **L2 = N|L3** **end** **end** **L1}, E7**) **(L2 = N|L3, E6)**	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) I0=I1 I2 I3 a1 b1 c1 n1 = 1 n2 = 2 **n3 = 3**	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
				E1
				Ints-> i1 N->n1 L1->I1
				E2
				Ints->i1 L2->I1
				E3
				Ints->i1 N->n1 L2->I2 L3-> I3
E4	**E5**	**E6**	**E7**	
Ints->i1 N->n2 L1->I3	Ints->i1 N->n2 L2->I2	Ints->i1 N->n2 L2->I2 L3-> I3	**Ints->i1** **N->n3** **L1->I3**	

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

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 = N|L3, E3)	(L2 = N|L3, E6)	**({proc {$ L2}** **local L3 in** **{Ints N+1 L3}** **L2 = N|L3** **end** **end}, E7**)	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) I0=I1 I2 I3 a1 b1 c1 n1 = 1 n2 = 2	Ints-> i1 L->I0 A-> a1 B-> b1 C-> c1
					E1
					Ints-> i1 N->n1 L1->I1
					E2
					Ints->i1 L2->I1
					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->l2 L3-> l3	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 = N|L3, E3)	-	({proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end}, E7)	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=l1 **l2= 2|l3** l3 =3|_ a1 b1 c1 n1 = 1 n2 = 2 n3 = 3	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1
					E1
					Ints-> i1 N->n1 L1->l1
					E2
					Ints->i1 L2->l1
					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)	-	-	({proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end}, E7)	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=**l1=1||2** l2= 2||3 l3 =3|_ a1 b1 c1 n1 = 1 n2 = 2 n3 = 3	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1
					E1
					Ints-> i1 N->n1 L1->l1
					E2
					Ints->i1 L2->l1
					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 = N|L3 end end}, E7)	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=l1=1||2 l2= 2||3 l3 =3|_ **a1 = 3** b1 c1 n1 = 1 n2 = 2	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1
					E1
					Ints-> i1 N->n1 L1->l1
					E2
					Ints->i1 L2->l1
					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->l2 L3-> l3	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 = N|L3 end end}, E7)	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=l1=1|l2 l2= 2|l3 l3 a1 = 3 **b1 = 1** c1 n1 = 1 n2 = 2 n3 = 3	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1 **E1** Ints-> i1 N->n1 L1->l1 **E2** Ints->i1 L2->l1 **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 = N|L3 end end}, E7)	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1

				end end L1} end, CE={Ints->i1}) l0=l1=1|l2 l2= 2|l3 l3 a1 = 3 b1 = 1 **c1 = 4** n1 = 1 n2 = 2 n3 = 3	**E1** Ints-> i1 N->n1 L1->l1 **E2** Ints->i1 L2->l1 **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 = N|L3 end end}, E7)	i1= (proc {$ N L1} {ByNeed proc {$ L2} local L3 in {Ints N+1 L3} L2 = N|L3 end end L1} end, CE={Ints->i1}) l0=l1=1|l2 l2= 2|l3 l3 a1 = 3 b1 = 1 c1 = 4 n1 = 1 n2 = 2 n3 = 3	Ints-> i1 L->l0 A-> a1 B-> b1 C-> c1 **E1** Ints-> i1 N->n1 L1->l1 **E2** Ints->i1 L2->l1 **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		

L1->I3	L2->I2	L2->I2 L3-> I3	L1->I3		

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

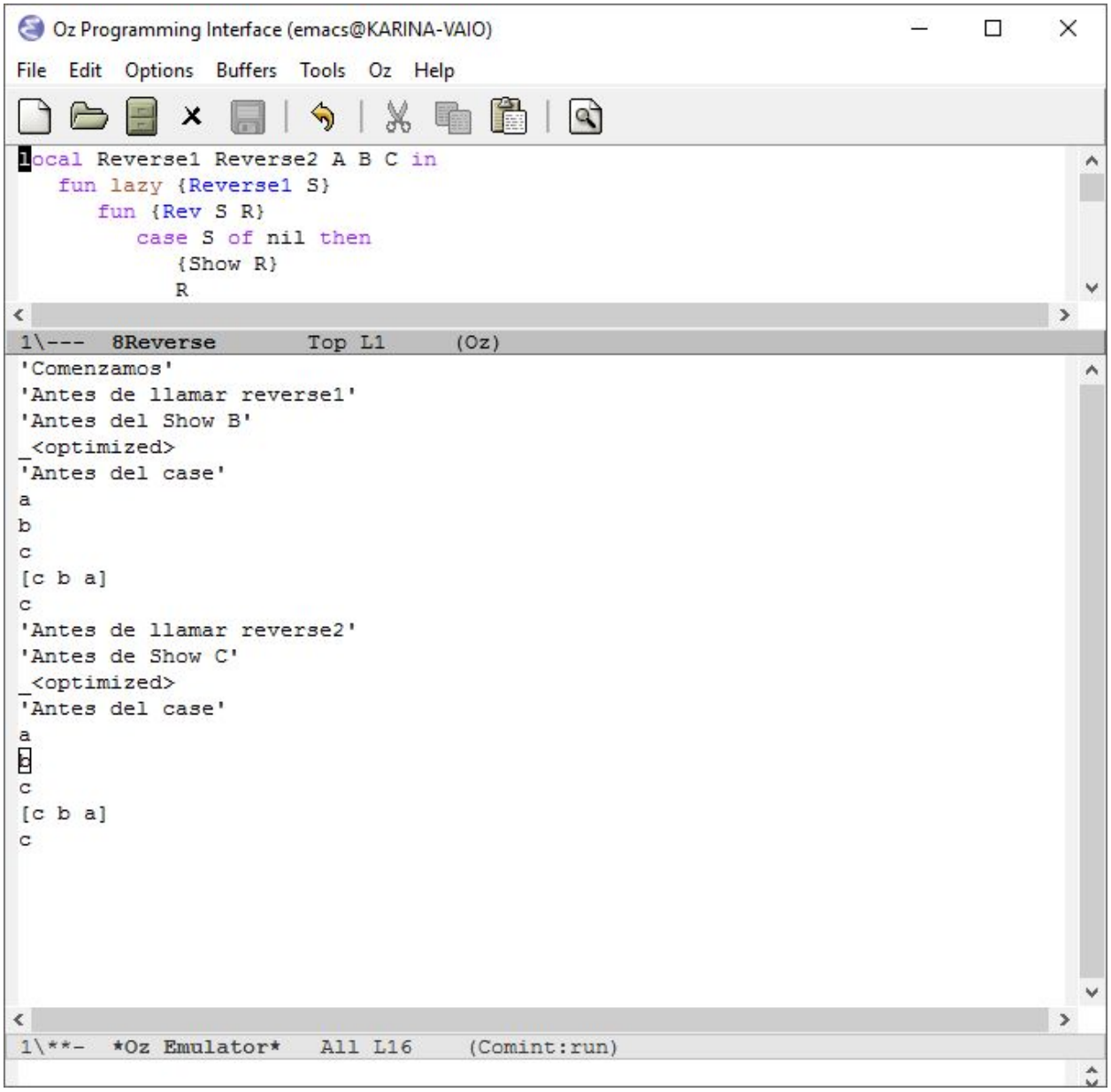
8.2 Reverse

- 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
```



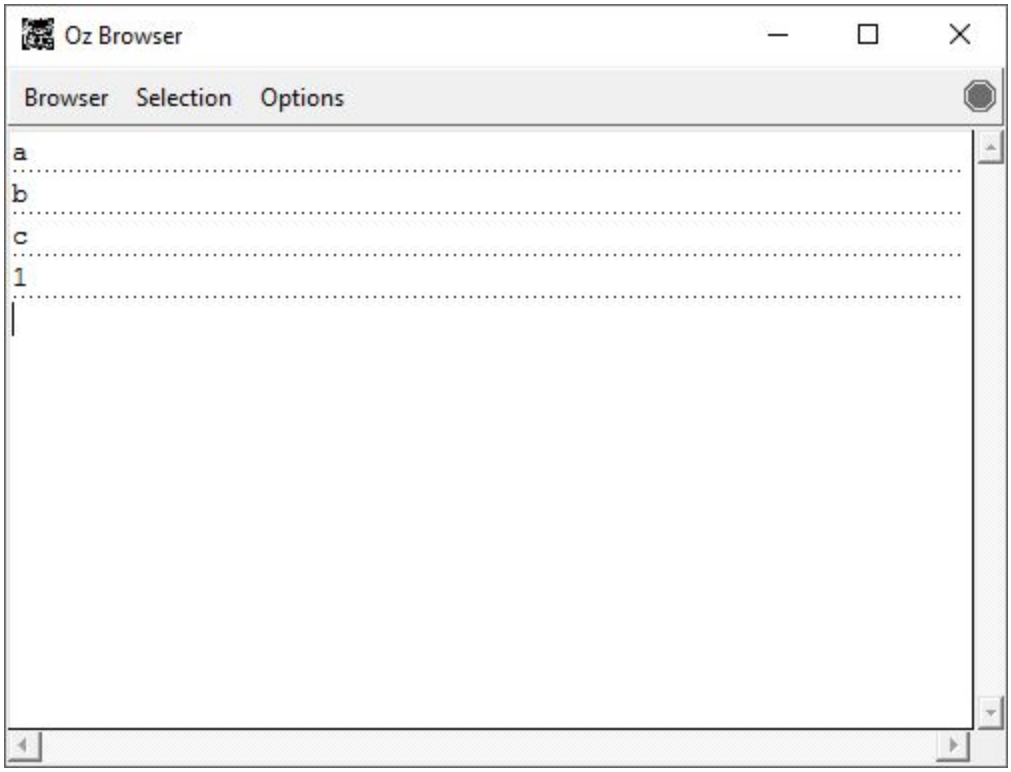
Ejercicio 9) - Mensajes

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
```

b. Ejemplo de ejecución

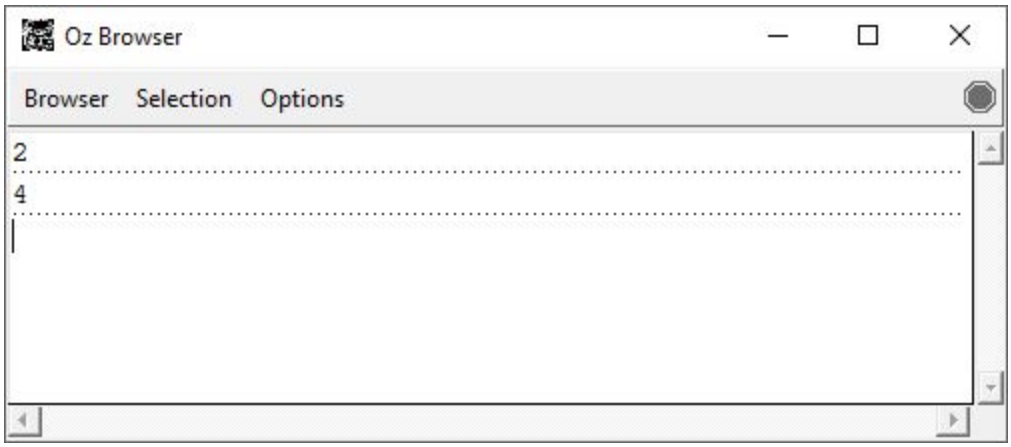


Ejercicio 10) - Servidor de filtros

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
```

b. Ejemplo de ejecución



Ejercicio 11) - Celdas de memoria

Explicación linea a linea

`declare` `X={NewCell 0}`	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