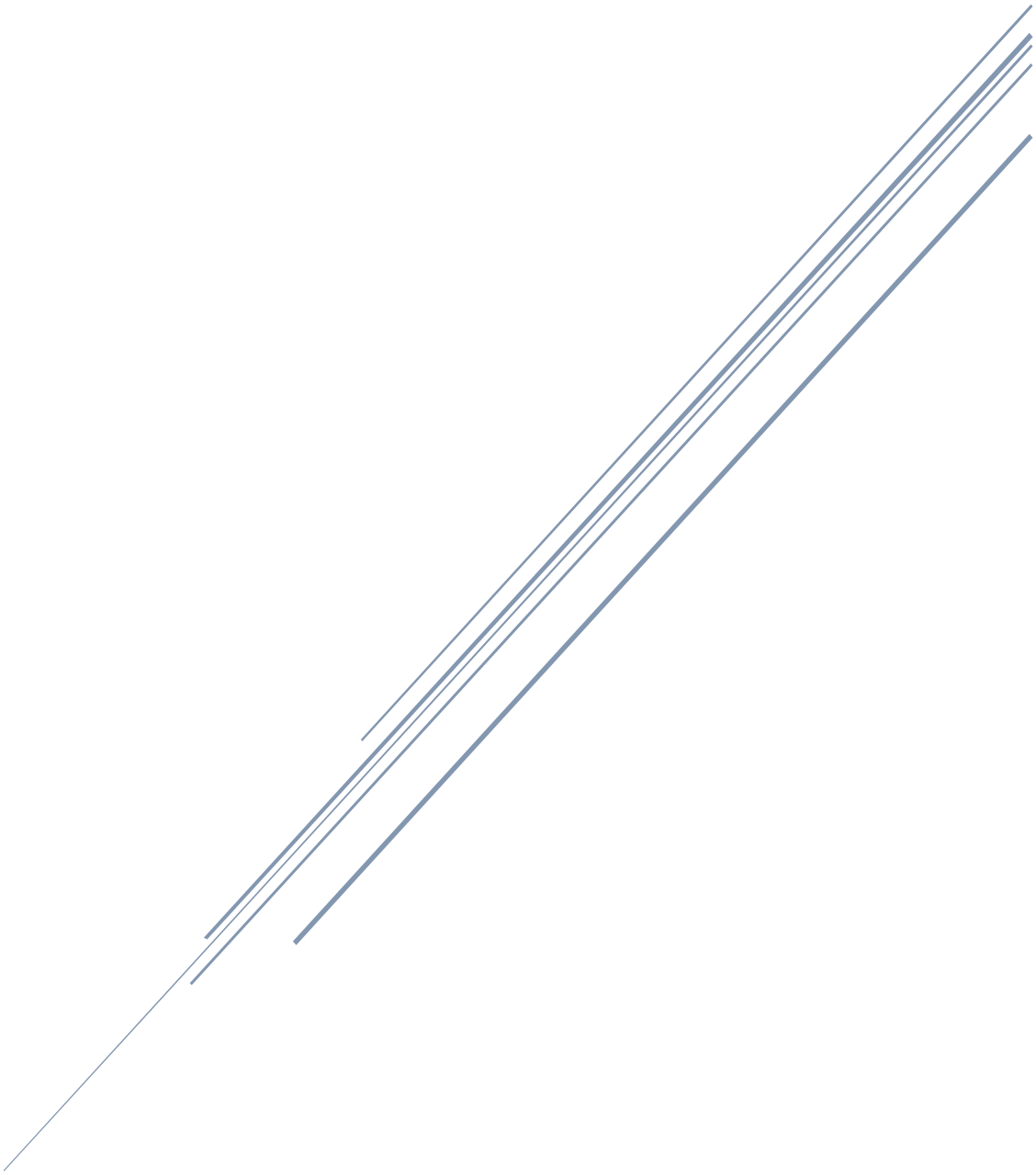
# LABORATORIO 4: SEMÁNTICA LÓGICA

## MFC - Modelos Formales de Computación



MITSS
Sergi Sanz Carreres

# 2. Ejercicios básicos a resolver

Dada la codificación de la función de longitud de una lista mostrada anteriormente, indica con detalle el resultado de la ejecución de cada uno de los siguientes programas.

## 2.1. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f([])
) =>! X:[Value] .
```

Devuelve un 0 debido a que la lista se encuentra vacía.

```
Maude>  search evalLog(('x1,'x2,'x3,'x4), let rec f(l) = if null?(l) then 0 els
 1 + f(cdr(l)) in f([])) =>! X:[Value] .
search in LOGFUN-TEST : evalLog('x1,'x2,'x3,'x4, let rec f l = if null?(l) ther
    0 else 1 + f cdr(l) in f []) =>! X:[ValueList] .

Solution 1 (state 0)
states: 1  rewrites: 44 in 0ms cpu (0ms real) (~ rewrites/second)
X:[ValueList] --> int(0)

No more solutions.
states: 1  rewrites: 44 in 0ms cpu (0ms real) (~ rewrites/second)
Maude>
```

## 2.2. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(cons(1,cons(2,[])))
) =>! X:[Value] .
```

El siguiente programa devuelve una lista con 2 elementos, donde cons coge una lista vacía y añade un 2, posteriormente cons vuelve a coger la lista y añade un 1.

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
>  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
>  in f(cons(1,cons(2,[])))
> ) =>! X:[Value] .
search in LOGFUN-TEST : evalLog('x1,'x2,'x3,'x4, let rec f l = if null?(l) then
    0 else 1 + f cdr(l) in f cons(1, cons(2, [])) =>! X:[ValueList] .

Solution 1 (state 0)
states: 1  rewrites: 126 in 0ms cpu (0ms real) (~ rewrites/second)
X:[ValueList] --> int(2)

No more solutions.
```

## 2.3. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f([1,2])
) =>! X:[Value] .
```

En esta ocasión se devuelve un dos debido a que se le pasa directamente una lista con dos elementos (1 y 2)

```
Maude>  search evalLog(('x1,'x2,'x3,'x4),
>
>  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
>
>  in f([1,2])
>
> ) =>! X:[Value] .
search in LOGFUN-TEST : evalLog('x1,'x2,'x3,'x4, let rec f l = if null?(l) then
    0 else 1 + f cdr(l) in f [1,2]) =>! X:[ValueList] .

Solution 1 (state 0)
states: 1  rewrites: 120 in 0ms cpu (0ms real) (~ rewrites/second)
X:[ValueList] --> int(2)

No more solutions.
```

## 2.4. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f([ivar(a),ivar(b)])
) =>! X:[Value] .
```

Devuelve dos elementos, pero los dos elementos que devuelve no se sabe el valor de los elementos, de forma que esta ejecución es más genérica que la anterior.

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
>  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
>  in f([ivar(a),ivar(b)])
> ) =>! X:[Value] .
search in LOGFUN-TEST : evalLog('x1,'x2,'x3,'x4, let rec f l = if null?(l) then
    0 else 1 + f cdr(l) in f [ivar(a),ivar(b)]) =>! X:[ValueList] .

Solution 1 (state 0)
states: 1  rewrites: 120 in 1ms cpu (0ms real) (120000 rewrites/second)
X:[ValueList] --> int(2)

No more solutions.
states: 1  rewrites: 120 in 1ms cpu (0ms real) (120000 rewrites/second)
Maude>
```

## 2.5. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(lvar(l))
) =>! VL:ValueList,int(0) .
```

Devuelve el resultado nil, por que la lista necesaria para que su tamaño sea 0 es una lista vacía.

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
>  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
>  in f(lvar(l))
> ) =>! VL:ValueList,int(0) .
search in LOGFUN-TEST : evalLog('x1,'x2,'x3,'x4, let rec f l = if null?(l) then
    0 else 1 + f cdr(l) in f lvar(l)) =>! VL:ValueList,int(0) .

Solution 1 (state 3)
states: 5  rewrites: 70 in 1ms cpu (0ms real) (70000 rewrites/second)
VL:ValueList --> l |-> {nil}

No more solutions.
```

## 2.6. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(lvar(l))
) =>! VL:ValueList,int(4) .
```

Devuelve una lista de cuatro elementos donde la cabeza l y cola x1 a su vez es una lista con cabeza 1 y cola x2, que a su vez es una lista con cabeza 1 y cola x3, que es una lista con cabeza 1 y cola x4, la cual es una lista vacía, por tanto, devuelve una lista de cuatro elementos que satisface la condición de que la función devuelva 4.

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
>  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
>  in f(lvar(l))
> ) =>! VL:ValueList,int(4) .
search in LOGFUN-TEST : evalLog('x1,'x2,'x3,'x4, let rec f l = if null?(l) then
    0 else 1 + f cdr(l) in f lvar(l)) =>! VL:ValueList,int(4) .

Solution 1 (state 22)
states: 23  rewrites: 293 in 2ms cpu (2ms real) (146500 rewrites/second)
VL:ValueList --> ('x4 |-> {nil}),('x3 |-> {int(1),lval('x4)}),('x2 |-> {int(1),
    lval('x3)}),('x1 |-> {int(1),lval('x2)}),l |-> {int(1),lval('x1)}

No more solutions.
```

## 2.7. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(lvar(l))
) =>! VL:ValueList,int(5) .
```

Devuelve que no tiene solución por que no tenemos suficientes variables auxiliares debido a que necesita 5 y solo tenemos 4.

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
>  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
>  in f(lvar(l))
> ) =>! VL:ValueList,int(5) .
search in LOGFUN-TEST : evalLog('x1,'x2,'x3,'x4, let rec f l = if null?(l) then
    0 else 1 + f cdr(l) in f lvar(l)) =>! VL:ValueList,int(5) .

No solution.
```

## 2.8. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(cons(1,lvar(l)))
) =>! VL:ValueList,int(5) .
```

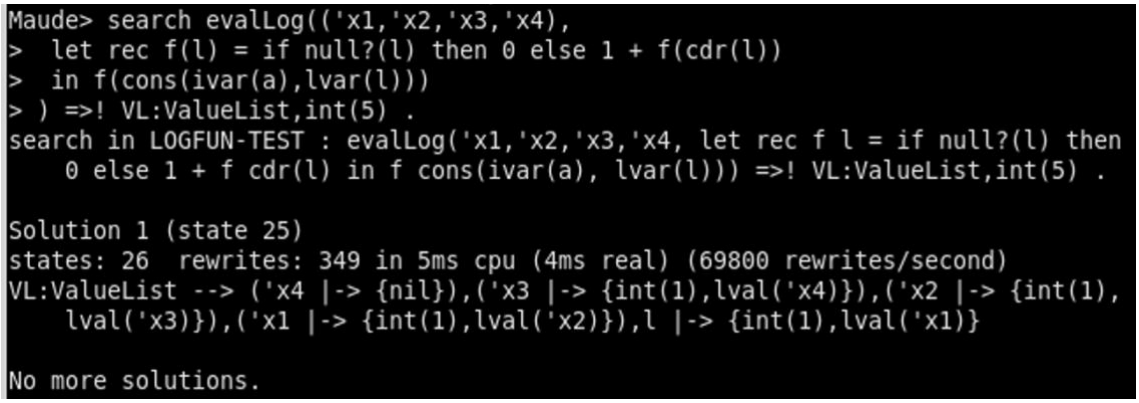
A diferencia del ejercicio anterior, en este caso se añade un elemento al principio de la lista de manera que puede utilizarse las variables auxiliares de las que disponemos, devolviendo una lista con cinco elementos.

```
Maude>  search evalLog(('x1,'x2,'x3,'x4),
>
>  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
>
>  in f(cons(1,lvar(l)))
>
> ) =>! VL:ValueList,int(5) .
search in LOGFUN-TEST : evalLog('x1,'x2,'x3,'x4, let rec f l = if null?(l) then
    0 else 1 + f cdr(l) in f cons(1, lvar(l))) =>! VL:ValueList,int(5) .

Solution 1 (state 25)
states: 26  rewrites: 349 in 4ms cpu (4ms real) (87250 rewrites/second)
VL:ValueList --> ('x4 |-> {nil}),('x3 |-> {int(1),lval('x4)}),('x2 |-> {int(1),
    lval('x3)}),('x1 |-> {int(1),lval('x2)}),l |-> {int(1),lval('x1)}

No more solutions.
```

## 2.9. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(cons(ivar(a),lvar(l)))
) =>! VL:ValueList,int(5) .
```

Devuelve una lista con cinco elementos, pero se diferencia del ejercicio anterior en que en vez de un número concreto le pasa un elemento cualquiera.




## 2.10. ¿Qué devuelve el siguiente programa y por qué?

```
Maude> search evalLog(('x1,'x2,'x3,'x4),
  let rec f(l) = if null?(l) then 0 else 1 + f(cdr(l))
  in f(cons(1,lvar(l)))
) =>! VL:ValueList,int(6) .
```

No tiene solución al igual que el ejercicio 2.7 debido a que no tiene suficientes variables auxiliares ya que necesitaríamos 5 en lugar de 4