



Universidad del Valle
Facultad de ingeniería
Ingeniería en sistemas

Cristian David Pacheco Torres
2227437

Juan Sebastian Molina Cuellar
2224491

Septiembre 2023

Taller 1

Abstract

Your abstract goes here functional programming

Contents

1	Introduction	4
2	Taller 1 : Recursión	4
2.1	Calcular el tamaño de una lista con un proceso iterativo . . .	4
2.1.1	Informe de procesos	4
2.1.2	Informe de corrección	6
2.2	Dividiendo una lista en dos sublistas a partir de un pivote . .	7
2.2.1	Informe de procesos	7
2.2.2	Informe de corrección	7
2.3	Calculando el k-ésimo elemento de una lista	7
2.3.1	Informe de procesos	7
2.3.2	Informe de corrección	7
2.4	Ordenando una lista	7
2.4.1	Informe de procesos	7
2.4.2	Informe de corrección	7
3	Conclusion	7

1 Introduction

Para el desarrollo de este taller, se utilizaron las siguientes funciones en scala:

- *isEmpty*: Boolean (Devuelve si una lista esta vacia).
- *head*: Int (devuelve si una lista *l* esta vacia).
- *tail*: List[Int] (devuelve la lista sin el primer elemento *l*).
- *x :: l*: devuelve la lista que representa la secuencia $\langle x, x_1, x_2, \dots, x_n \rangle$ si *l* es la lista que representa la secuencia $\langle x_1, x_2, \dots, x_n \rangle$.
- *l1 ++ l2* devuelve la lista que representa la concatenacion de las secuencias representadas por *l1* y *l2*.

Se da por hecho de que las funciones anteriores estan argumentadas y demostradas, por lo tanto se procede a la resolucion de los ejercicios.

2 Taller 1 : Recursión

2.1 Calcular el tamaño de una lista con un proceso iterativo

En el listing 1 se puede apreciar la funcion *tamR(l)* que a traves de un proceso recursivo calcula el tamaño de una lista.

```
1  def tamR(l : List[Int] ) : Int = {  
2    if (l.isEmpty) 0  
3    else 1 + tamR(l.tail )  
4  }
```

Listing 1: Calcula el tamaño de una lista con un proceso recursivo

El problema a solucionar es hacer una funcion *tamI(l)* tal que:
 $tamR(l) == tamI(l)$

2.1.1 Informe de procesos

Descripción de la función.

La solución propuesta se basa en el siguiente algoritmo (ver listing 2.)
Para ello se creó la función $tamI(l : List[Int]) : Int = \{\}$.

```

1  def tamI(l : List[Int]): Int = {
2      def tam(lst : List[Int], acc : Int): Int = {
3          if (lst.isEmpty) acc
4          else tam(lst.tail, acc + 1)
5      }
6      tam(l, 0)
7  }

```

Listing 2: Calcula el tamaño de una lista con un proceso iterativo

Para esta implementación se utilizó una función auxiliar:

$tam(lst : List[Int], acc : Int) : Int$ (ver línea 2) la cual recibe como parámetros una lista (l) y un acumulador (acc), el cual se encarga de contar el tamaño de la lista. En el caso base (ver línea 3), si la lista está vacía, se retorna el acumulador, en caso contrario se llama a la función tam (ver línea 4) con la lista sin el primer elemento ($lst.tail$) y el acumulador incrementado en 1.

Para la inicialización de la función $tamI(l : List[Int]) : Int$ se llama a la función tam (línea 6) con la lista y un acumulador inicializado en 0.

Tipo de proceso.

Se pretende de que el tipo de proceso es **iterativo** para ello evaluamos la función con parámetro $List(12, 3, 1, 8, 4)$

```

tamI(List(12, 3, 1, 8, 4))
→ tamI(List(12, 3, 1, 8, 4))
→ tam(List(12, 3, 1, 8, 4), 0)
→ if(List(12, 3, 1, 8, 4).isEmpty) 0 else tam(List(12, 3, 1, 8, 4).tail, 0 + 1)
→ tam(List(3, 1, 8, 4), 1)
→ if(List(3, 1, 8, 4).isEmpty) 1 else tam(List(3, 1, 8, 4).tail, 1 + 1)
→ tam(List(1, 8, 4), 2)
→ if(List(1, 8, 4).isEmpty) 2 else tam(List(1, 8, 4).tail, 2 + 1)
→ tam(List(8, 4), 3)
→ if(List(8, 4).isEmpty) 3 else tam(List(8, 4).tail, 3 + 1)
→ tam(List(4), 4)
→ if(List(4).isEmpty) 4 else tam(List(4).tail, 4 + 1)
→ tam(List(), 5)
→ if(List().isEmpty) 5 else tam(List().tail, 5 + 1)
→ 5

```

Aunque la función utiliza la recursión en su estructura, el proceso que genera es **iterativo**, ya que el proceso mantiene su forma constante.

2.1.2 Informe de corrección

Argumentacion sobre la corrección.

Se desea calcular el tamaño de una lista de enteros, tal que si $List(x_1, x_2, x_3, \dots, x_n) \rightarrow n$, donde $n \in \mathbb{N}_{\geq 0}$ es el tamaño de la lista. Si la lista es vacia entonces $n = 0$

Sea $L = List(a_1, a_2, a_3, \dots, a_n)$, $n \geq 0$.

Sea $f : L \rightarrow \mathbb{N}_{\geq 0}$, $f(L) = n$

Y sea P_f (ver Listing 2) la propiedad que se desea demostrar, $P_f(L) : f(L) = n$.

- Un estado $s = (l, acc)$ donde $l = List(a_i, a_{i+1}, \dots, a_n)$ es una cola de L .
- El estado inicial $s_0 = (L, 0) = (List(x_1, x_2, x_3, \dots, x_n), 0)$.
- $s = (l, acc)$ es final si l es vacia.
- $Inv(l, acc) \equiv l = List(a_i, a_{i+1}, \dots, a_n) \wedge acc = f(List(a_1, a_2, \dots, a_{i-1}))$.
- $transformar(l, acc) = (l', acc')$ donde $l' = l.tail$ y $acc' = acc + 1$.

Demostracion.

1. $Inv(s_0)$: el estado inicial cumple la condición invariante. Como $s_0 = (L, 0)$, entonces $l = L$ y $acc = 0$, por lo tanto $l = List(a_1, a_2, \dots, a_n)$ y $acc = f(List()) = 0$, por lo tanto $Inv(s_0)$ se cumple.
2. ()

2.2 Dividiendo una lista en dos sublistas a partir de un pivote

2.2.1 Informe de procesos

2.2.2 Informe de corrección

2.3 Calculando el k-ésimo elemento de una lista

2.3.1 Informe de procesos

2.3.2 Informe de corrección

2.4 Ordenando una lista

2.4.1 Informe de procesos

2.4.2 Informe de corrección

3 Conclusion

La conclusion

$$a = \sum F \dot{m} = \frac{dv}{dt}$$