Universidad del Valle

Universidad del Valle

Facultad de ingeniería

Ingeniería en sistemas

Cristian David Pacheco Torres
2227437
Juan Sebastian Molina Cuellar
2224491

Septiembre 2023

Taller 2

Abstract

Your abstract goes here functional programming

Contents

1 Introduction

A introduction a ver asdf

2 Taller 1 : Recursión

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 $< x, x_1, x_2, ..., x_n >$ si l es la lista que representa la secuencia $< x_1, x_2, ..., x_n >$.
- $l1 + +l2$ 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.

2.1 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.

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

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

Descripcion de la funcion.
La solucion propuesta se basa en el siguiente algoritmo (ver listing 2.)
Para ello se creo la funcion $tamI(l : List[Int]) : Int = \{\}$.

```
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 implementacion se utilizo una funcion auxiliar:
$tam(lst : List[Int], acc : Int) : Int$ (ver linea 2) la cual recibe como parametros una lista (l) y un acumulador (acc), el cual se encarga de contar el tamaño de la lista. En el caso base (ver linea 3), si la lista esta vacia, se retorna el acumulador, en caso contrario se llama a la funcion tam (ver linea 4) con la lista sin el primer elemento ($lst.tail$) y el acumulador incrementado en 1.
Para la inicializacion de la funcion $tamI(l : List[Int]) : Int$ se llama a la funcion tam (linea 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 funcion con parametro $List(12, 3, 1, 8, 4)$
$tamI(List(12, 3, 1, 8, 4))$
$\rightarrow tamI(List(12, 3, 1, 8, 4))$
$\rightarrow tam(List(12, 3, 1, 8, 4), 0)$
$\rightarrow if(List(12, 3, 1, 8, 4).isEmpty)\ 0\ else\ tam(List(12, 3, 1, 8, 4).tail, 0 + 1)$
$\twoheadrightarrow tam(List(3, 1, 8, 4), 1)$
$\rightarrow if(List(3, 1, 8, 4).isEmpty)\ 1\ else\ tam(List(3, 1, 8, 4).tail, 1 + 1)$
$\twoheadrightarrow tam(List(1, 8, 4), 2)$
$\rightarrow if(List(1, 8, 4).isEmpty)\ 2\ else\ tam(List(1, 8, 4).tail, 2 + 1)$
$\twoheadrightarrow tam(List(8, 4), 3)$
$\rightarrow if(List(8, 4).isEmpty)\ 3\ else\ tam(List(8, 4).tail, 3 + 1)$
$\twoheadrightarrow tam(List(4), 4)$
$\rightarrow if(List(4).isEmpty)\ 4\ else\ tam(List(4).tail, 4 + 1)$
$\twoheadrightarrow tam(List(), 5)$
$\rightarrow if(List().isEmpty)\ 5\ else\ tam(List().tail, 5 + 1)$
$\twoheadrightarrow 5$

Puesto que el proceso mantiene una forma constante, se puede decir que el proceso es **iterativo**.

2.1.2 Informe de corrección

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

2.2.1 Informe de procesos

```
  def noMenoresQue(l: List[Int], value : Int): List[Int
    ] = {

   if (l.isEmpty) Nil
   else if (l.head >= value) l.head :: noMenoresQue(l.
      tail, value)
   else noMenoresQue(l.tail, value)
 }
```

Listing 3: Calcula una lista construida con los valores menores de un valor pivote proporcionado

Este es un proceso de forma expansion contracion de tiempo lineal en funcion y es una funcion recursiva $menoresQue(List(9,\ 3,\ 1,\ 8,\ 7,\ 10,\ 0,\ 7),\ 8)$
$\rightarrow if\ (List(9,\ 3,\ 1,\ 8,\ 7,\ 10,\ 0,\ 7).isEmpty())\ [\,]$
$else\ if(9 < 8)\ 9 :: List(3,\ 1,\ 8,\ 7,\ 10,\ 0,\ 7)$
$else\ menoresQue(List(3,\ 1,\ 8,\ 7,\ 10,\ 0,\ 7),\ 8)$
$\twoheadrightarrow 3 :: menoresQue(List(1,\ 8,\ 7,\ 10,\ 0,\ 7),\ 8)$
$\twoheadrightarrow 3 :: 1 :: menoresQue(List(8,\ 7,\ 10,\ 0,\ 7),\ 8)$
$\twoheadrightarrow 3 :: 1 :: menoresQue(List(7,\ 10,\ 0,\ 7),\ 8)$
$\twoheadrightarrow 3 :: 1 :: 7 :: menoresQue(List(10,\ 0,\ 7),\ 8)$
$\twoheadrightarrow 3 :: 1 :: 7 :: menoresQue(List(0,\ 7),\ 8)$
$\twoheadrightarrow 3 :: 1 :: 7 :: 0 :: menoresQue(List(7),\ 8)$
$\twoheadrightarrow 3 :: 1 :: 7 :: 0 :: 7 :: menoresQue([\,],\ 8)$
$\rightarrow 3 :: 1 :: 7 :: 0 :: [7]$
$\rightarrow 3 :: 1 :: 7 :: [0,\ 7]$
$\rightarrow 3 :: 1 :: [7,\ 0,\ 7]$
$\rightarrow 3 :: [1,\ 7,\ 0,\ 7]$
$\rightarrow [3,\ 1,\ 7,\ 0,\ 7]$

2.2.2 Informe de corrección

Sea $L = \{l \mid l \text{ es un secuencia } < a_0, ...a_i, ..., a_n > \text{ para } 0 \leq i \leq n \in N^+ \cup [\,]\}$ un conjunto de secuencias de n elementos junto a un elemento que representa un lista vacia y sea $f : List[Int] \rightarrow List[Int]$ una funciñn definida en L para $k \geq 1$ número de elementos, y codominio en conjunto L, tal que $f(l) = m$ cumple $l \in L$ y $m \in L \wedge m \subseteq l \vee m = \emptyset$ y representa la secuencia de los elementos $a_i \in l$ que son menores a un k determinado.
Sea Pf el anterior programa realizado en $Scala$ que implementa f y al cúal se quiere demostrar su correctitud:

Se quiere probar

$$\forall \in N \setminus \{0\} : P_f(List(a_1, ..., a_i, ..., a_n)) = f(List(a_1, ..., a_i, ..., a_n))$$

- Caso base: Cuando $l = [\,]$. Utilizando el modelo de sustitución, se tiene:

  $P_f([\,]) \rightarrow si(l = [\,])\ [\,]$

  Se tiene en la funcion teorica f $f([]\ ) = [\,]$

  $\therefore\ P_f([\,]) = f([\,])$

- Caso de inducción: para $n = k + 1$ se debe tener que

  $Si\ P_f(List(b_1, ..., b_i, ..., b_k)) = f(List(b_1, ..., b_i, ..., b_k))$
  $\rightarrow P_f(List(b_1, ..., b_i, ..., b_{k+1})) = f(List(b_1, ..., b_i, ..., b_{k+1}))$
  Sea $m = List(a_1, ..., a_i, ..., a_{k+1})$ y p un valor comparar en el predicado $a_i \in m \wedge p \geq 1$, y por medio del modelo de sustitucion en $P_f(m)$ se tiene

  $P_f(m) \rightarrow if\ (m.isEmpty)\ [\,]\ else\ if(m.head < p)\ m.head ::$
  $P_f(\ List(a_2, ..., a_i, .., a_{k+1}))\ else\ P_f(List(a_2, ..., a_i, .., a_{k+1}))$

  Sea $\alpha = List(a_2, ..., a_i, ..., a_{k+1})$ la lista resultante en el primer paso de la iteracion y diferenciando aritméticamente los subíndices de los elementos extremos de las lista, se tiene
  $k + 1 - 2 = k\ -\ 1$
  Lo cual es tamaño obtenido para una lista de k elementos, así, por hipotésis de indución, para una lista de k elementos conlleva a

$$\therefore P_f(List(a_1, .., a_i, ..., a_n)) = f(List(a_1, .., a_i, ..., a_n)) \; \forall n \in N \setminus \{0\}$$

2.3 Calculando el k-ésimo elemento de una lista

2.3.1 Informe de procesos

```
def k_elem(l : List[Int], k : Int) : Int = {

    if(tamI(menoresQue(l, l.head)) == k - 1) l.head
    else if(tamI(menoresQue(l, l.head)) >= k) k_elem(
        menoresQue(l, l.head), k)
    else k_elem(noMenoresQue(l, l.head), k - tamI(
        menoresQue(l, l.head)) - 1)
 }
```

Listing 4: Calcula el k-ésimo elemento de una lista en el orden natural de los enteros positivos

$k\_elem(List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3),\ 6)$
$\to if(tamI(menoresQue(List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3)),\ 4))\ ==\ 6\ -\ 1)$
$\quad List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3).head$
$\quad else\ if(tamI(menoresQue(List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3), 4)) >=\ 6)$
$\quad k\_elem(menoresQue(List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3),$
$\quad List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3).head),\ 6)$
$\quad else\ k\_elem(noMenoresQue(List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3),$
$\quad List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3).head),$
$\quad 6 - tamI(menoresQue(List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3),$
$\quad List(4,\ 6,\ 1,\ 2,\ 7,\ 10,\ 8,\ 5,\ 3).head))\ -\ 1)$

$\twoheadrightarrow\ k\_elem(noMenoresQue(List(4,\ 6,\ 7,\ 10,\ 8,\ 5),$
$\quad List(4,\ 6,\ 7,\ 10,\ 8,\ 5).head),\ 6 - tamI(menoresQue(List(4,\ 6,\ 7,\ 10,\ 8,\ 5),$
$\quad List(4,\ 6,\ 7,\ 10,\ 8,\ 5).head))\ -\ 1)$

$\twoheadrightarrow\ k\_elem(noMenoresQue(List(4,\ 6,\ 7,\ 10,\ 8,\ 5),$
$\quad List(4,\ 6,\ 7,\ 10,\ 8,\ 5).head),\ 6 - 3 - 1)\ k\_elem(List(4,\ 6,\ 7,\ 10,\ 8,\ 5), 2)$

$\to\ k\_elem(List(\ 6,\ 7,\ 10,\ 8,\ 5), 1)$
$\quad if(tamI(menoresQue(List(\ 6,\ 7,\ 10,\ 8,\ 5)),\ 6))\ ==\ 1\ -\ 1)$
$\quad List(\ 6,\ 7,\ 10,\ 8,\ 5).head$
$\quad else\ if(tamI(menoresQue(List(\ 6,\ 7,\ 10,\ 8,\ 5), 4)) >=\ 1)$
$\quad k\_elem(menoresQue(List(6,\ 7,\ 10,\ 8,\ 5),\ \ List(\ 6,\ 7,\ 10,\ 8,\ 5).head),\ 1)$

$else\ k\_elem(noMenoresQue(List(\ 6,\ 7,\ 10,\ 8,\ 5),$
$List(\ 6\ 7,\ 10,\ 8,\ 5).head),\ 1 - tamI(menoresQue(List(\ 6,\ 7,\ 10,\ 8,\ 5),$
$List(\ 6,\ 7,\ 10,\ 8,\ 5).head))\ -\ 1)$
$\rightarrow\ 6$

2.3.2 Informe de corrección

- $s_o = (L, k)$ L la lista original, v el valor pivote
- $s = (L, k)$
- $s_f =\ k = f(L) \wedge tamI(menoresQue(L)) == k$
- $inv(s_i) = 0 \leq k \leq n \wedge k = f(L)$
- $T(s_i) = (L, k), m, v$ donde si $tamI(menoresQue(L)) \geq L = List(a_j, ..., L.head)$ sino $L = List(L.head, a_m) paraciertos 0 \leq m, j \geq n \wedge k = k - 1 - tamI(menoresQue(L.tail, L.head))$
- $Inv(s_0)$. Cumple con el invariante el estado inicial, $s_0 = (L, 0)$
- $s_i \neq s_f \wedge Inv(s_i) \rightarrow Inv(T(s_i))$ El nuevo estado cumple con la condicion invariante $s_i \neq s_f \wedge Inv(s_i) \rightarrow Inv(T(s_i))$
  $\equiv$
  $0 \leq k \leq n \wedge k = f(L) \rightarrow k = k \vee K = k - tamI(menoresQue(l, l.head)) - 1$
- $s_f =\ k = f(L) \wedge tamI(menoresQue(L)) == k$
- $inv(s_i) = 0 \leq k \leq n \wedge k = f(L)$
- $T(s_i) = (L, k), m, v$ donde si $tamI(menoresQue(L)) \geq L = List(a_j, ..., L.head)$ sino $L = List(L.head, a_m) paraciertos 0 \leq m, j \geq n \wedge k = k - 1 - tamI(menoresQue(L.tail, L.head))$

2.4 Ordenando una lista

2.4.1 Informe de procesos

2.4.2 Informe de corrección

3 Funciones de alto orden implementadas

A continuación, se presenta la funciones implementadas de alto orden, las cuales fueron utilizadas para instanciar otras funciones (funciones generadoras), a través de su paso como parámetro, ya sea referenciada (nominada) o

Funciones de alto orden		
Función	Forma de alto orden	Expresión donde aparece
Chip	Retorno	Retorno de funciones crearChipunario, crearChipBinario, half_adder, full_adder, adder
$(x : Int) \Rightarrow (x - 1)$	Lambda como argumento	$crearChipUnario((x : Int) \Rightarrow (x - 1)) : Chip$
$(x : Int, y : Int) => (x * y)$	Lambda como argumento	$crearChipBinario((x : Int, y : Int) => (x * y)) : Chip$
$(x : Int, y : Int) => (x + y) - (x * y)$	Lambda como argumento	$crearChipBinario((x : Int, y : Int) => (x + y) - (x * y)) : Chip$
$half\_adder$	Variable la cual se asigna una función de retorno	val $half\_adder = (operands : List[Int]) =>$ { ... }
$full\_adder$	Variable la cual se asigna una función de retorno	val $full\_adder = (operands : List[Int]) =>$ { ... }
adder	Variable la cual se asigna una función de retorno	val $adder = (operands : List[Int]) =>$ { ... }

Table 1: Funciones de alto orden realizadas en la implementación del circuito lógico.

como funcion anónima(inline), o como valor retorno de la misma.

4 Crear chip unario

4.1 Informe de procesos

Realiza una operación logica sobre un solo valor de entrada. A continuación, se presenta su implementación en *Scala*

```
def crearChipUnario( f: Int => Int ) : Chip = (arg: List[Int]) => { // Apply the f function on the head of current list and call recursively the crearChipUnarioHelper with function f, a accumulated list with new transformed value as its head, and the current list tail, until the empty list condition is reached.
  @tailrec
  def crearChipUnarioHelper(f: Int => Int, transformedList: List[Int], currentList: List[Int]): List[Int] =
```

```
    if (currentList.isEmpty) transformedList
    else crearChipUnarioHelper(f, f(currentList.head)
       ::transformedList, currentList.tail)

  // The initial state of the iteration
  crearChipUnarioHelper(f, List(), arg)
}
```
Listing 5: Aplica una operación binaria sobre una valor de entrada.

4.2 Informe de corrección

$val\ chip\_not = crearchipUnario(x => 1 - x)$

Caso 1:
$chip\_not((List(0)))$
$\rightarrow crearChipUnarioHelper(x => 1 - x, [\,],\ List(0))$
$\rightarrow if(List(0).isEmpty)\ [\,]$
$\quad else\ crearChipUnarioHelper(x => 1 - x, (1 - 0) :: [\,], [\,])$
$\rightarrow if(List().isEmpty)\ [1]$
$\quad else\ crearChipUnarioHelper(x => 1 - x, (1 - 1) :: [1], [\,])$
$\rightarrow [1]$

Caso 2:
$chip\_not((List(1)))$
$\rightarrow crearChipUnarioHelper(x => 1 - x, [\,], List(1))$
$\rightarrow if(List(1).isEmpty)\ [\,]$
$\quad else\ crearChipUnarioHelper(x => 1 - x, (1 - 1) :: [\,], [\,])$
$\rightarrow if(List().isEmpty)\ [0]$
$\quad else\ crearChipUnarioHelper(x => 1 - x, (1 - 0) :: [0], [\,])$
$\rightarrow [0]$

5 Conclusion

La conclusion

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