

# TP FINAL - ANALISIS DE LENGUAJES DE PROGRAMACIÓN

Tomás Maiza (M-7116/1)

15/04/2025

## Descripción del Proyecto

El proyecto consiste en un DSL para funciones recursivas de listas (FRL). Se define una **lista** como una secuencia ordenada de cero o más elementos pertenecientes a los  $N_0$ . Llamando  $L$  al conjunto de todas las listas, las **funciones de lista** son funciones que van de  $L$  en  $L$ .

Las listas admiten diferentes formas y operaciones que se representan en el lenguaje, como:

- Listas vacías
- Operación concatenación
- Funciones base:
  - **Cero a derecha** (inserta un 0 como último elemento de la lista)
  - **Cero a izquierda** (inserta un 0 como primer elemento de la lista)
  - **Borrar a derecha** (borra el último elemento de la lista)
  - **Borrar a izquierda** (borra el primer elemento de la lista)
  - **Sucesor a izquierda** (incrementa en una unidad el primer elemento de la lista)
  - **Sucesor a derecha** (incrementa en una unidad el último elemento de la lista)
- **Operador composición** (permite la aplicación de una sucesión de funciones)
- **Operador repetición** (permite iterar una función sobre una lista)

Además, incluye algunas funciones básicas formadas a partir de las funciones base:

- **Pasar a izquierda** (mueve el primer elemento al último lugar de la lista)
- **Pasar a derecha** (mueve el último elemento al primer lugar de la lista)
- **Duplicar a izquierda** (duplica el primer elemento de la lista)
- **Duplicar a derecha** (duplica el último elemento de la lista)
- **Intercambiar extremos** (intercambia el primer y el último elemento)

## Modo de uso

El proyecto puede ejecutarse en dos entornos: estático e interactivo.

Para ejecutar un programa estático se debe lanzar alguna de las siguientes líneas de comandos en la carpeta del proyecto:

```
cabal run tp -- pathToFile
cabal run tp -- pathToFile -s
```

El archivo cargado tiene una secuencia de comandos (separados por ‘;’) que se correrán sin mostrar la traza de cada ejecución.

La bandera **-s** explicita que se va a utilizar el modo estático y si no se ponen banderas se asume que se utilizará el modo estático.

Para ejecutar el modo interactivo se pueden lanzar los siguientes comandos en la carpeta del proyecto:

```
cabal run tp -- -i
cabal run tp -- pathToFile -i
```

La bandera **-i** indica que se quiere usar el modo interactivo. En el primer caso se abre el entorno interactivo y permite ejecutar comandos uno a uno mostrando la ejecución paso a paso. En el segundo caso se carga el archivo indicado, se muestra la traza de la ejecución de sus comandos y se procede a entrar en el entorno interactivo.

Dentro del entorno interactivo se cuenta con los comandos **:l** (para cargar archivos que pueden contener definiciones) y **:q** (para finalizar su ejecución).

Para cargar archivos debemos escribir:

```
> :l ejemplos/archivo.frl
```

Para finalizar la ejecución del entorno interactivo simplemente escribimos:

```
> :q
```

Se cuenta con los siguientes símbolos que representan funciones de listas:

- **Od, Oi** (0 a derecha y 0 a izquierda)
- **Sd, Si** (Sucesor a derecha y Sucesor a izquierda)
- **Bd, Bi** (Borrar a derecha y Borrar a izquierda)
- **Dd, Di** (Duplicar a derecha y Duplicar a izquierda)
- **(->), (<-)** (Mover a derecha y Mover a izquierda)
- **(<->)** (Intercambiar extremos)

Para escribir listas debemos iniciarla con '[' y terminarla con ']', y entre ellos escribir sus elementos (números naturales) separados por coma.

*Ejemplos de listas válidas:* [], [1], [7, 4, 3, 6]

Para aplicar una función la escribimos a la izquierda de una lista.

*Ejemplos:* Oi [], Sd [1, 2, 3]

Las funciones pueden componerse escribiendo una al lado de la otra, ejecutándose primer la función ubicada más a la izquierda.

*Ejemplos:* Od Sd [1, 2]

Podemos escribir una **repetición** encerrando una función entre los símbolos '<' y '>'.

*Ejemplos:* <Si> [0, 3], Od <Sd> Bi [7, 6, 2, 4]

El lenguaje soporta notación de potencia, escribiendo la potencia al lado de la función (con espacio de por medio). Podemos aplicar la potencia a una composición escribiéndola entre los caracteres '{' y '}'.

*Ejemplos:* Od 2 [1, 2], Od 2 Sd 3 [1, 2, 3], Od {Sd Di} 3 Dd [7]

Podemos asignar nombres a una función, una lista o al resultado de aplicar una función a una lista con el símbolo '='. Las listas deben llevar nombres de letras mayúsculas y las funciones deben llamarse con palabras que empiecen en minúscula.

*Ejemplos:* X = [1, 2], foo = Sd (->) Di, Y = foo [12, 8], Z = Od Bd X, H = [X, Y]

Con la función 'print' podemos ver el contenido de una variable de lista o a qué función refiere una variable.

*Ejemplo:* print X, print foo

Podemos determinar si el resultado de una aplicación es una lista en particular (o no) con los símbolos '==' y '!='.

*Ejemplos:* Od Oi [1] == [0, 1, 0] (indica True), Od X != [] (indica True), Od [] == [] (indica False)

En un archivo podemos escribir varios comandos colocando el símbolo ';' al final de cada línea (menos la última).

*Ejemplo:*

```
foo = Od (<->);  
X = [];  
Sd [1, 3];  
Od X
```

Con `'/'` podemos hacer un comentario de una línea. Los comentarios que abarquen más líneas se abren con `'/'` y cierran con `'/'`.

## Organización de los archivos

La carpeta del proyecto cuenta con la siguiente estructura:

```
.
|---- tp.cabal
|---- informe.pdf
|---- app/
|   |---- Test.hs
|   |---- Main.hs
|---- ejemplos/
|   |---- variablesListas.frl
|   |---- variablesFuncionesListas.frl
|   |---- test.frl
|---- src/
|   |---- Eval.hs
|   |---- Monads.hs
|   |---- PPfrl.hs
|   |---- Parser.hs
|   |---- AST.hs
```

## Características del proyecto

Se tomaron como referencia para la organización del código los trabajos prácticos realizados durante el cursado, principalmente el TP 3.

Elegí un enfoque **deep embedding** para representar los elementos, almacenando el AST en los tipos de datos y evaluando todo en una misma función (el evaluador).

### Sintaxis Abstracta

```
elems ::=  $\epsilon$  | nat | nat, elems
fun ::= op fun' | fun' | varFun fun' ::=  $\epsilon$  | fun fun'
op ::= Oi | Od | Si | Sd | Bi | Bd | <-> | Di | Dd | -> | <-
list ::= [elems] | [varList] | var
varList ::= cl | cl, varList varFun ::= var
```

Donde `nat` es el conjunto de los números naturales, `var` el conjunto de identificadores de variables y `cl` es el conjunto de caracteres en mayúscula.

```
comm ::= skip
      | var = list
      | comm; comm
      | fun list
      | var = fun list
      | fun list == list
      | fun list != list
      | print varList
```

## Representación en Haskell

La representación del AST en Haskell se puede encontrar en `src/AST.hs`. Se tiene un tipo para listas con 5 constructores:

- **Nil** (para listas vacías `[]`)
- **Unit** (para listas de un único elemento `[x]`)
- **Cons** (para listas de dos o más elementos `[x, y]`)
- **Concat** (para concatenar variables de listas `[X, Y]`. Además es el constructor utilizado para construir las listas parseadas)
- **Var** (para almacenar variables de listas `X`)

El constructor `Cons` se definió de manera que permita el acceso eficiente al primer y último elemento de la lista (ya que las funciones de listas siempre trabajan sobre estos). La desventaja de este enfoque es que es ineficiente construir la lista en base a este constructor al parsearla, por lo que todas las listas de dos o más elementos se construyen con `Concat` y a la hora de evaluar alguna función en la lista, el evaluador la transforma a una construida a base de `Cons`.

El tipo de las funciones cuenta con 4 constructores:

- **Op** (para operadores básicos como `LeftZero`, `RightSucc`, `Swap`, etc)
- **Repeat** (para representar una repetición)
- **Comp** (para representar la composición de funciones)
- **FunVar** (para almacenar nombres de funciones)

También se cuenta con un tipo para representar los comandos con los siguientes constructores:

- **App** (representa la aplicación de una función a una lista, `f X`)
- **LetList** (representa la asignación de una lista a una variable, `X = [x]`)
- **LetListFun** (representa la asignación de una aplicación a una variable, `X = f [x]`)
- **LetFun** (representa la asignación de un nombre a una función, `f = 0d Si`)
- **Seq** (representa una secuencia de comandos)
- **Eq** (representa la comparación de igualdad entre una aplicación y una lista, `f X == Y`)
- **NEq** (representa la comparación de desigualdad entre una aplicación y una lista, `f X != Y`)
- **Print** (representa la salida por pantalla del valor asociado a una variable, `print X`)
- **Skip** (representa un comando que no hace nada)

## Reglas de Evaluación

### Concat

$$\frac{}{\text{Concat } ls \text{ Nil} \rightarrow ls} \text{CONCAT}_1$$

$$\overline{\text{Concat Nil } ls \rightarrow ls} \text{ CONCAT}_2$$

$$\overline{(\text{Unit } x) (\text{Unit } y) \rightarrow \text{Cons } x \text{ Nil } y} \text{ CONCAT}_3$$

$$\overline{(\text{Unit } x) (\text{Cons } x' ls y') \rightarrow \text{Cons } x (\text{Concat } (\text{Unit } x') ls) y'} \text{ CONCAT}_4$$

$$\overline{(\text{Cons } x ls y) (\text{Unit } z) \rightarrow \text{Cons } x (\text{Concat } ls (\text{Unit } y)) z} \text{ CONCAT}_5$$

$$\overline{(\text{Cons } x ls y) (\text{Cons } x' ls' y') \rightarrow \text{Cons } x (\text{Concat } (\text{Concat } ls (\text{Unit } y)) (\text{Concat } (\text{Unit } x') ls')) y'} \text{ CONCAT}_6$$

## LeftZero

$$\overline{\text{LeftZero Nil} \rightarrow \text{Unit } 0} \text{ L-ZERO}_1$$

$$\overline{\text{LeftZero } (\text{Unit } x) \rightarrow \text{Cons } 0 \text{ Nil } x} \text{ L-ZERO}_2$$

$$\overline{\text{LeftZero } (\text{Cons } x ls y) \rightarrow \text{Cons } 0 (\text{Concat } (\text{Unit } x) ls) y} \text{ L-ZERO}_3$$

## RightZero

$$\overline{\text{RightZero Nil} \rightarrow \text{Unit } 0} \text{ R-ZERO}_1$$

$$\overline{\text{RightZero } (\text{Unit } x) \rightarrow \text{Cons } x \text{ Nil } 0} \text{ R-ZERO}_2$$

$$\overline{\text{RightZero } (\text{Cons } x ls y) \rightarrow \text{Cons } x (\text{Concat } ls (\text{Unit } y)) 0} \text{ R-ZERO}_3$$

## LeftDel

$$\overline{\text{LeftDel Nil} \rightarrow \text{DomainErr}} \text{ L-DEL}_1$$

$$\overline{\text{LeftDel } (\text{Unit } x) \rightarrow \text{Nil}} \text{ L-DEL}_2$$

$$\overline{\text{LeftDel } (\text{Cons } x ls y) \rightarrow \text{Concat } ls (\text{Unit } y)} \text{ L-DEL}_3$$

## RightDel

$$\overline{\text{RightDel Nil} \rightarrow \text{DomainErr}} \text{ R-DEL}_1$$

$$\overline{\text{RightDel } (\text{Unit } x) \rightarrow \text{Nil}} \text{ R-DEL}_2$$

$$\overline{\text{RightDel } (\text{Cons } x ls y) \rightarrow \text{Concat } (\text{Unit } x) ls} \text{ R-DEL}_2$$

## LeftSucc

$$\frac{}{LeftSucc\ Nil \rightarrow DomainErr} \text{L-SUCC}_1$$

$$\frac{}{LeftSucc\ (Unit\ x) \rightarrow Unit\ (Succ\ x)} \text{L-SUCC}_2$$

$$\frac{}{LeftSucc\ (Cons\ x\ ls\ y) \rightarrow Cons\ (Succ\ x)\ ls\ y} \text{L-SUCC}_3$$

## RightSucc

$$\frac{}{RightSucc\ Nil \rightarrow DomainErr} \text{R-SUCC}_1$$

$$\frac{}{RightSucc\ (Unit\ x) \rightarrow Unit\ (Succ\ x)} \text{R-SUCC}_2$$

$$\frac{}{RightSucc\ (Cons\ x\ ls\ y) \rightarrow Cons\ x\ ls\ (Succ\ y)} \text{R-SUCC}_3$$

## MoveLeft

$$\frac{}{MoveLeft\ Nil \rightarrow DomainErr} \text{L-MOVE}_1$$

$$\frac{}{MoveLeft\ (Unit\ x) \rightarrow Unit\ x} \text{L-MOVE}_2$$

$$\frac{}{MoveLeft\ (Cons\ x\ ls\ y) \rightarrow Concat\ (Unit\ y)\ (Concat\ (Unit\ x)\ ls)} \text{L-MOVE}_3$$

## MoveRight

$$\frac{}{MoveRight\ Nil \rightarrow DomainErr} \text{R-MOVE}_1$$

$$\frac{}{MoveRight\ (Unit\ x) \rightarrow Unit\ x} \text{R-MOVE}_2$$

$$\frac{}{MoveRight\ (Cons\ x\ ls\ y) \rightarrow Concat\ (Concat\ ls\ (Unit\ y))\ (Unit\ x)} \text{R-MOVE}_3$$

## DupLeft

$$\frac{}{DupLeft\ Nil \rightarrow DomainErr} \text{L-DUP}_1$$

$$\frac{}{DupLeft\ (Unit\ x) \rightarrow Cons\ x\ Nil\ x} \text{L-DUP}_2$$

$$\frac{}{DupLeft\ (Cons\ x\ ls\ y) \rightarrow Cons\ x\ (Concat\ (Unit\ x)\ ls)\ y} \text{L-DUP}_3$$

## DupRight

$$\frac{}{\text{DupRight } Nil \rightarrow \text{DomainErr}} \text{R-DUP}_1$$

$$\frac{}{\text{DupRight } (Unit\ x) \rightarrow \text{Cons } x\ Nil\ x} \text{R-DUP}_2$$

$$\frac{}{\text{DupRight } (\text{Cons } x\ ls\ y) \rightarrow \text{Cons } x\ (\text{Concat } ls\ (Unit\ y))\ y} \text{R-DUP}_3$$

## Swap

$$\frac{}{\text{Swap } Nil \rightarrow \text{DomainErr}} \text{SWAP}_1$$

$$\frac{}{\text{Swap } (Unit\ x) \rightarrow Unit\ x} \text{SWAP}_2$$

$$\frac{}{\text{Swap } (\text{Cons } x\ ls\ y) \rightarrow \text{Cons } y\ ls\ x} \text{SWAP}_3$$

## Comp

$$\frac{}{\text{Comp } f\ g\ ls \rightarrow g\ (f\ ls)} \text{COMP}$$

## Repeat

$$\frac{}{\text{Repeat } f\ Nil \rightarrow \text{DomainErr}} \text{REPEAT}_1$$

$$\frac{}{\text{Repeat } f\ (Unit\ x) \rightarrow \text{DomainErr}} \text{REPEAT}_2$$

$$\frac{x = y}{\text{Repeat } f\ (\text{Cons } x\ ls\ y) \rightarrow \text{Cons } x\ ls\ y} \text{REPEAT}_3$$

$$\frac{x \neq y}{\text{Repeat } f\ (\text{Cons } x\ ls\ y) \rightarrow \text{Repeat } f\ (f\ (\text{Cons } x\ ls\ y))} \text{REPEAT}_4$$

## Bibliografía

- Slides y trabajos prácticos de la cátedra
- Slides funciones recursivas de listas de la cátedra de Lenguajes Formales y Computabilidad
- Libro “Temas de Teoría de la Computación”