Práctica 1: Primer intérprete de Expresiones Aritmético-Booleanas

Lenguajes de Programación I

López Soto Ramses Antonio
Quintero Villeda Erik

21 de agosto de 2019

Introducción

Objetivo

Implementar un intérprete de expresiones artimético-booleanas (**EAB**) a través del lenguaje de programación *Haskell*.

PostFix se trata de un lenguaje simple basado en una pila e inspirado en el lenguaje gráfico PostScript, el lenguaje de programación Forth y las calculadoras HP.

Desarrollo

La implementación del intérprete se realizo por medio del lenguaje de programación *Haskell* a manera de simular el mismo comportamiento del lenguaje *Postfix*

Implementación

Para la implementación del intérprete se utilizó lo siguientes:

```
data Instruction = I Int | B Bool
                 | ADD | AND | OR | DIV | EQQ | EXEC
                 | GET | Gt | Lt | POW | MAX | MIN | FACT | MUL
                 | NOT | POP | REM | SEL | SUB
                 | SWAP | ES [Instruction] deriving (Eq, Show)
type Stack = [Instruction]
type Program = [Instruction]
```

```
data Expr = N Int | T | F | Succ Expr | Pred Expr
                  | Expr :+ Expr | Expr : Expr | Expr : Expr
                  | Expr :/ Expr | Expr :% Expr | Not Expr
                  | Expr : Expr | Expr :| Expr
                  | Expr :> Expr | Expr :< Expr | Expr := Expr
                  | Expr :^ Expr| Max Expr Expr | Min Expr Expr
                  | Fact Expr deriving (Eq,Show)
```

Las funciones implementadas en el archivo *practica1.hs* fueron Los siguientes:

- *Función que se encarga de realizar todas las operaciones aritméticas haciendo uso de instrucciones.*

  arithOperation :: Instruction -> Instruction -> Instruction -> Instruction

- *Función que se encarga de mostrar la valuación del operador binario AND a modo de instrucción*

  bboolOperation :: Instruction -> Instruction -> Instruction -> Instruction

- *Función que se encarga de mostrar la valuación del operador unario NOT a modo de instrucción*

  uboolOperation :: Instruction -> Instruction -> Instruction

- *Función que se encarga de mostrar el resultado luego de comparar si dos números son iguales o uno es mayor o menor que otro.*

  relOperation :: Instruction -> Instruction -> Instruction -> Instruction

- *Función que se encarga de realizar las operaciones de una pila usando las instrucciones definidas*

  stackOperation :: Stack -> Instruction -> Stack

- *Función que devuelve la lista de instrucciones y la pila*

  execOperation :: [Instruction] -> Stack -> Instruction -> ([Instruction], Stack)

- *Función que se encarga de de realizar la evaulación de un programa y una lista de instrucciones*

  executeProgram :: Program -> Stack -> Stack

- *Función que se encarga de traducir una istrucción a un programa*

  compile:: Expr -> Program

- *Función que se encarga de traducir una expresión a una instrucción*

  execute :: Expr -> Instruction

A lo largo de la práctica se hizo uso de la recursión para definir todas y cada una de las funciones requieridas.

Conclusión

La implementación del intérprete fue un tanto complicada. Acabamos de implementar el intérprete con tiempo, pero al verificarla, notamos que algunas funciones se encontraban mal implementadas, ya que, nosotros definimos las funciones de una forma, la cual, no era correcta. Por lo que tuvimos que volver a implementar y arreglar algunas cosas.

Simular el comportamiento de PostFix a través de este intérprete nos ayudó a comprender de una manera mucho más clara su comportamiento, y además, a tener el cuidado de verificar si la implementación es la correcta o se debe de corregir y/o mejorar, si es posible.

En conclusión, lo que esta práctica tiene como fin, es que el lenguaje PostFix sea comprendido de manera más fácil por medio de una implementación, la cual no es idéntica al lenguaje pero si cercana para comprender su comportamiento.