Práctica 01

Galeana Araujo Emiliano
Miranda Sánchez Kevin Ricardo

Facultad de Ciencias, UNAM

Fecha de entrega: 26 de Agosto de 2018

1. Descripción del programa

En este semanal vemos una implementación de *Postfix*, que es una secuencia parentizada que consiste en una palabra reservada `Postfix`, seguida de un número natural que indica el número de argumentos que recibe el programa, seguido de cero o más comandos.

1.1. Postfix

```
data Command = I Int | ADD | DIV | Eq | EXEC | Gt | Lt
               | MUL | NGET | POP | REM | SEL | SUB | SWAP
               | ES [Command] deriving(Show, Eq)

data PF = POSTFIX deriving (Show, Eq)

type Program = (PF, Int, [Command])

type Stack = [Command]
```

Un ejemplo de un programa en Postfix:

```
(POSTFIX, 0, [ES[I 0, SWAP, SUB], I 7, SWAP, EXEC]) []
```

Las siguientes son funciones para la implementación de Postfix, algunas son recursivas.

```
-- | arithOperation. Funcion que realiza las operaciones de los comandos
--   aritmeticos. (add, div, eq, gt, lt, mul, rem, sub)
arithOperation :: Command → Command → Command → Command

-- | stackOperation. Funcion que realiza las operaciones de los comandos
--   que alteran la pila de valores.(ADD, DIV, Eq, Gt, Lt, MUL, REM, SUB).
stackOperation :: Stack → Command → Stack

-- | execOperation. Funcion que devuelve la lista de comandos y
--   la pila resultante de realizar la llamada a la operacion con exec.
execOperation :: [Command] → Stack → ( [ Command ] , Stack )

-- | validProgram. Funcion que determina si la pila de valores que
--   se desea ejecutar con un programa es valida.
validProgram :: Program → Stack → Bool

-- | executeCommands. Funcion que dada una lista de comandos y
--   una pila de valores obtiene la pila de valores resultant's despu s ejecutar
--   todos los comandos.
executeCommands :: [Command] → Stack → Stack

-- | executeProgram. Funci n que ejecuta cualquier programa en Postfix.
executeProgram :: Program → Stack → [Command]
```

2. Entrada y ejecución

El programa es interpretado por GCHI de la siguiente forma

```
~:ghci Practica1.hs
```

2.1. Postfix

Ya en el programa, los siguientes son ejemplos de la ejecución de las funciones para la implementación de Postfix.

```
*Practica1> arithOperation (I 1) (I 2) ADD
I 3

*Practica1> arithOperation (I 8) (I 0) DIV
*** Exception: Division entre 0.
```

```
*Practica1> stackOperation [I 1, I 5] SWAP
[I 5, I 1]

*Practica1> stackOperation [I 1, I 5] (ES [I 3, ADD, SWAP, I 2])
[ES [I 3, ADD, SWAP, I 2], I 1, I 5]

*Practica1> execOperation [ADD] [ES [I 1, ADD], I 2, I 3]
([I 1, ADD, ADD], [I 2, I 3])

*Practica1> execOperation [MUL] [ES [I 1, ADD], I 2, I 3]
([I 1, ADD, MUL], [I 2, I 3])

*Practica1> validProgram (POSTFIX, 0, [I 1, I 2, ADD]) []
True

*Practica1> validProgram (POSTFIX, 2, [ADD]) [I 3, ES[I 1, I 2, SUB]]
False

*Practica1> executeCommands seq1 []
[I 3]

*Practica1> executeCommands seq2 [I 7]
[I 14]

*Practica1> executeProgram (POSTFIX, 0, prg1) []
[I (-7)]

*Practica1> executeProgram (POSTFIX, 1, prg2) [I 4, I 5]
*** Exception: No es un programa v l i d o .
```

3. Conclusiones

Estuvo bien, el problema era ir a leer las notas e intentar entender como funcionaban algunas cosas, por ejemplo en `arithOperation` se supone que deberíamos restar el segundo elemento de la pila, menos el primero, pero para que los ejemplos del PDF pasen los cambiamos.
Otra cosa interesante fue todos los auxiliares que tuvimos que hacer para

por ejemplo verificar en `stackOperation` pues diferentes comandos usan diferentes tipos de pilas, por ejemplo no podíamos hacer `SWAP` de una lista con menos de dos elementos.

Referencias

[1] lp191n02.pdf, Archivero, curso de Lenguajes de Programacion 2019-1.