Práctica 1

Primer intérprete de `EAB`

Favio E. Miranda Perea (favio@ciencias.unam.mx)
Javier Enríquez Mendoza (javierem_94@ciencias.unam.mx)
Pablo G. González López (pablog@ciencias.unam.mx)

Miércoles 7 de agosto de 2019

Fecha de entrega: Miércoles 21 de agosto de 2019 a las 23:59:59.

El objetivo de esta práctica será implementar un pequeño intérprete de un lenguaje de expresiones aritmético-booleanas.

Las representación de las expresiones en *Haskell* de nuestro lenguaje serán las siguientes:

```
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
```

Para implementar el intérprete, compilaremos estas expresiones a secuencias de instrucciones que se ejecutarán en una pila para obtener el resultado.

1 Instrucciones

1.1 Sintaxis

Las instrucciones podrán tener alguna de las siguientes formas:

- Una literal entera.
- Una literal booleana.
- Una de las siguientes palabras reservadas: `add`, `and`, `div`, `eq`, `exec`, `get`, `gt`, `lt`, `mul`, `not`, `pop`, `rem`, `sel`, `sub`, `swap`.

- Una secuencia ejecutable. Una secuencia parentizada de instrucciones que sirve como una subrutina.

Para representar estas instrucciones en *Haskell* usaremos la siguiente definición:

```
data Instruction = I Int | B Bool
                 | ADD | AND | DIV | Eq | EXEC | GET| Gt
                     | Lt
                 | MUL | NOT | POP | REM | SEL | SUB |
                   SWAP
                 | ES [Instruction]
```

1.2 Semántica

El significado de una secuencia de instrucciones se determina ejecutando cada una en orden, manipulando una pila de valores.

La pila de valores la representaremos del siguiente modo:

```
type Stack = [Instruction]
```

Hay que tener en mente que durante la ejecución de las instrucciones, si se esta trabajando con una pila inapropiada, se pueden generar errores. Estos errores deben ser manejados por la función que los genera y deben informar explícitamente al usuario cuál fue la causa de ese error.

1. (1 punto) `arithOperation`. Función que realiza las operaciones de las instrucciones aritméticas (`add`, `div`, `mul`, `rem`, `sub`). Los primeros dos argumentos corresponden a los operandos, y el tercero al operador. La función devuelve una literal entera.

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

   Ejemplo:

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

2. (0.5 puntos) `bboolOperation`. Función que realiza las operaciones de las instrucciones booleanas binarias (`and`). Los primeros dos argumentos corresponden a los operandos, y el tercero al operador. La función devuelve una literal booleana.

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

   Ejemplo:

   ```
   *Main> bboolOperation (B True) (B False) AND
   B False
   ```

3. (0.5 puntos) `uboolOperation`. Función que realiza las operaciones de las instrucciones booleanas unarias (`not`). El primer argumento corresponde al operando, y el segundo al operador. La función devuelve una literal booleana.

```
uboolOperation :: Instruction -> Instruction
   -> Instruction
```

Ejemplo:

```
*Main> uboolOperation (B True) NOT
B False
```

4. (0.5 puntos) `relOperation`. Función que realiza las operaciones de las instrucciones relaciones (`eq`, `gt`, `lt`). Los primeros dos argumentos corresponden a los operandos, y el tercero al operador. La función devuelve una literal booleana.

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

Ejemplo:

```
*Main> relOperation (I 1) (I 2) Eq
B False
```

5. (1.5 puntos) `stackOperation`. Función que realiza las operaciones de las instrucciones que alteran la pila de valores (literal entera, literal booleana, `get`, `pop`, `sel`, `swap`, secuencia ejecutable).

```
stackOperation :: Stack -> Instruction ->
   Stack
```

Ejemplo:

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

6. (0.5 puntos) `execOperation`. Función que devuelve la secuencia de instrucciones y la pila resultante de realizar la llamada a la operación `exec`.

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

Ejemplo:

```
*Main> execOperation [ADD] [ES [I 1, ADD], I
    2, I 3] EXEC
([I 1, ADD, ADD], [I 2, I 3])
```

Ahora hay que implementar la ejecución de una secuencia de instrucciones en la pila de valores. A estas secuencias de instrucciones las llamaremos programas, y los representaremos del siguiente modo:

```
type Program = [Instruction]
```

1. (2 puntos) `executeProgram`. Función que dada una secuencia de instrucciones y una pila de valores, obtiene la pila de valores resultante después ejecutar todas las instrucciones.

   ```
   executeProgram :: Program -> Stack -> Stack
   ```

   Ejemplo:

   ```
   *Main> prg = [I (-1), I 2, ADD, I 3, MUL]
   *Main> executeProgram prg []
   [I 3]
   *Main> prg = [ES [I 2, MUL], EXEC]
   *Main> executeProgram prg [I 7]
   [I 14]
   *Main> prg = [I 4, MUL, ADD]
   *Main> executeProgram prg [I 3]
   *** Exception: Not enough numbers to add.
   *Main> prg = [I 4, SUB, DIV]
   *Main> executeProgram prg [I 4, I 5]
   *** Exception: Divide by zero.
   ```

2 Compilador

Finalmente brindaremos al usuario un compilador, para permitirle escribir expresiones aritmético-booleanas en lugar de programas.

1. (3 puntos) `compile`. Función que dada una expresión aritmético-booleana, obtiene la secuencia de instrucciones que permite evaluarla.

   ```
   compile :: Expr -> Program
   ```

   Ejemplo:

   ```
   *Main> compile ((Succ (N 1)) :+ (N 3))
   [I 1, I 1, ADD, I 3, ADD]
   ```

2. (0.5 puntos) `execute`. Función que dada una expresión aritmético-booleana, realiza la evaluación del programa resultante y devuelve una literal entera o una literal booleana.

```
execute :: Expr -> Instruction
```

Ejemplo:

```
*Main> execute ((Succ (N 1)) :+ (N 3))
I 5
*Main> execute ((Not F) :| T)
B True
```

3 Anexo: Semántica de los comandos

- I: Agrega la literal entera N a la pila.
- B: Agrega la literal booleana B a la pila.
- `sub`: Si el primer (tope) y segundo elementos de la pila son v_1, v_2 respectivamente, eliminarlos de la pila y agregar $v_2 - v_1$ en su lugar. Si hay menos de dos valores en la pila o los dos primeros valores no son números enteros, lanzar un error. Los demás operadores aritméticos (`add` (suma), `mul` (multiplicación), `div` (división entera), y `rem` (residuo de la división entera)) se comportan de manera similar. `div` y `rem` lanzan un error si v_1 es cero.
- `not`: Si el primer (tope) elemento de la pila es v_1, eliminarlo de la pila y agregar $\neg v_1$ en su lugar. Si hay menos de un valor en la pila o el valor no es un booleano, lanzar un error.
- `and`: Si el primer (tope) y segundo elementos de la pila son v_1, v_2 respectivamente, eliminarlos de la pila y agregar $v_2 \wedge v_1$ en su lugar. Si hay menos de dos valores en la pila o los dos primeros valores no son booleanos, lanzar un error.
- `lt`: Si el primer (tope) y segundo elementos de la pila son v_1, v_2 respectivamente, eliminarlos de la pila. Si $v_2 < v_1$ agregar $true$ a la pila, en otro caso agregar $false$ a la pila. Los demás operadores de comparación (`eq` (igualdad), `gt` (mayor que)) se comportan de manera similar. Si hay menos de dos valores en la pila o los dos primeros valores no son números enteros, lanzar un error.
- `pop`: Elimina el primer elemento de la pila. Lanza un error si esta es vacía.
- `swap`: Intercambia los primeros dos valores de la pila. Lanza un error si la pila tiene menos de dos valores.

- `sel`: Si el primer (tope), segundo y tercer elemento de la pila son v_1, v_2, v_3 respectivamente, eliminarlos de la pila. Si v_3 es la literal booleana $false$, agregar v_1 a la pila; si v_3 es la literal booleana $true$, agregar v_2 a la pila. Lanza error si la pila no contiene al menos tres valores, o si v_3 no es una literal booleana.

- `get`: Llamemos v_{index} al tope de la pila y v_1,...,v_n a los elementos restantes en orden siendo v_n el que está en el fondo. Eliminar v_{index} de la pila. Si v_{index} es un número i tal que $1 \leq i \leq n$, agregar v_i a la pila. Lanza un error si la pila no contiene al menos un valor, si v_{index} no es un número, o si i no esta en el rango $[1..n]$.

- $(C_1$ ... $C_n)$: Agrega la secuencia ejecutable como un solo valor en la pila. Las secuencias ejecutables se usan en conjunción con `exec`.

- `exec`: Eliminar la secuencia ejecutable del tope de la pila y agregar en orden sus comandos al inicio de la secuencia de comandos actualmente en ejecución. Lanza un error si la pila es vacía o si el primer valor no es una secuencia de comandos.

¡Suerte!