Lenguajes de programación 2016-2
Ejercicio Semanal 2

Noé Salomón Hernández Sánchez
Albert M. Orozco Camacho
C. Moisés Vázquez Reyes

Facultad de Ciencias UNAM
Se entrega el 1 de marzo

Considera la siguiente gramática:

$$e := n \mid x \mid e + e \mid \mathsf{let}\ x = e\ \mathsf{in}\ e$$

su respectiva definición en Haskell es:

```
data Exp = Num Int | Var String | Suma Exp Exp | Let String Exp Exp
```

Implementa lo siguiente:

- `freeVars::Exp->[String]`

  Captura todas las variables libres dentro de una expresión.

- `sust::Exp->String->Exp->Exp`

  Implementa la sustitución textual de una variable por una expresión usando la definición vista en clase:

$$x[x := t] = t$$

$$y[x := t] = y \qquad x \neq y$$

$$(e_1 + e_2)[x := t] = e_1[x := t] + e_2[x := t]$$

$$(\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2)[y := t] = \mathsf{let}\ x = e_1[y := t]\ \mathsf{in}\ e_2[y := t] \qquad x \notin \{y\} \cup FV(t)$$

$$(\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2)[y := t] = \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \qquad x \in \{y\} \cup FV(t)$$

- `eval::Exp->Int`

  Implementa un evaluador que reciba una expresión y la reduzca a su respectivo valor.

  La regla de evaluación es la siguiente:

  `eval` n `=` n
  `eval` x `= error`
  `eval (`e_1`+`e_2`) = eval` e_1 `+ eval` e_2
  `eval (let` x`=`e_1 `in` e_2`) = eval` e_2`[`x`:=eval` e_1`]`

  Ejemplos:

  - `eval (Let "x" (Num 3) $ Suma (Suma (Num 2) (Num 3))`
    `(Suma (Num 3) (Num 10)))`
    devuelve 18.
  - `eval (Let "x" (Suma (Num 1) (Num 7)) $ Var "x")`
    devuelve 8.

- **EXTRA** (+5 pts)

  `evalreto::Exp->Int->(Int,Int)`

  Extiende el evaluador de tal manera que nos diga cuántas sumas se efectuaron al evaluar una expresión.

  La idea es recibir una expresión y un entero que representa el estado actual del evaluador, es decir, el entero simulará una variable que vaya contando el número de sumas efectuadas. El evaluador se ejecutará con un estado inicial cero.

  Ejemplos:

  - `evalreto (Let "x" (Num 3) $ Suma (Suma (Num 2) (Num 3))`
    `(Suma (Num 3) (Num 10))) 0`
    devuelve (18,3).
  - `evalreto (Let "x" (Suma (Num 1) (Num 7)) $ Var "x") 0`
    devuelve (8,1).
  - `evalreto (Suma (Num 1) (Num 2)) 2`
    devuelve (3,3)