Ejercicio Semanal 2

El lenguaje `EAB`. (Sintaxis)

Favio E. Miranda Perea (favio@ciencias.unam.mx)
Diego Carrillo Verduzco (dixego@ciencias.unam.mx)
Pablo G. González López (pablog@ciencias.unam.mx)

Sábado 25 de agosto de 2018

Fecha de entrega: Sábado 1 de septiembre de 2018 a las 23:59:59.
El lenguaje de Expresiones Aritmético Booleanas de define del siguiente modo:

```
e ::= x | n | true | false
    | e + e | e * e | succ e | pred e
    | not e | and e e | or e e
    | lt e e | gt e e | eq e e
    | if e then e else e end | let x = e in e end
```

1 Sintaxis

Primero renombraremos el tipo de datos `String` a `Identifier` para representar al conjunto de nombres de variables como cadenas de texto.

```
type Identifier = String
```

Ahora habrá que definir la sintaxis de las expresiones `EAB`.

```
data Exp = V Identifier | I Int | B Bool
         | Add Exp Exp | Mul Exp Exp | Succ Exp | Pred Exp
         | Not Exp | And Exp Exp | Or Exp Exp
         | Lt Exp Exp | Gt Exp Exp | Eq Exp Exp
         | If Exp Exp Exp
         | Let Identifier Exp Exp deriving (Eq)
```

Para poder visualizar las expresiones de una forma más sencilla:

1. (1 punto) Crea una instancia de la clase `Show`.

```
instance Show Exp where
    show e = case e of
        (V x) -> "V[" ++ x ++ "]"
        (I n) -> "N[" ++ (show n) ++ "]"
        (B b) -> "B[" ++ (show b) ++ "]"
        ...
```

2 Sustitución y α-equivalencia

Definiremos el tipo sustitución del siguiente modo:

```
type Substitution = (Identifier, Exp)
```

Implementa las siguientes funciones:

1. (3 puntos) `frVars`. Obtiene el conjunto de variables libres de una expresión.

   ```
   frVars :: Exp -> [Identifier]
   ```

   Ejemplo:

   ```
   *Main> frVars (Add (V "x") (I 5))
   ["x"]
   *Main> frVars (Let "x" (I 1) (V "x"))
   []
   ```

2. (3 puntos) `subst`. Aplica la sustitución a la expresión dada en caso de ser posible.

   ```
   subst :: Exp -> Substitution -> Exp
   ```

   Ejemplo:

   ```
   *Main> subst (Add (V "x") (I 5)) ("x", I 10)
   Add (I 10) (I 5)
   *Main> subst (Let "x" (I 1) (V "x")) ("y", Add (V "x") (I 5))
   *** Exception: Could not apply the substitution.
   ```

3. (3 puntos) `alphaEq`. Determina si dos expresiones son alfa equivalentes.

   ```
   alphaEq :: Exp -> Exp -> Bool
   ```

   Ejemplo:

   ```
   *Main> alphaEq (Let "x" (I 1) (V "x")) (Let "y" (I 1) (V "y"))
   True
   *Main> alphaEq (Let "x" (I 1) (Add (V "x") (V "x")))
                  (Let "y" (I 1) (Add (V "y") (V "z")))
   False
   ```

¡Suerte!