Práctica 2

El lenguaje `EAB`. (Semántica)

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

Viernes 30 de agosto de 2019

Fecha de entrega: Miércoles 11 de septiembre de 2019 a las 23:59:59.

Recordemos la definición del lenguaje de Expresiones Aritmético Booleanas que usamos en el ejercicio semanal anterior.

```
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 Semántica Dinámica

La semántica dinámica es la que determina cuál es el valor de la evaluación de un programa. Como se vio en clase existen varios estilos para definirla, sin embargo seguiremos utilizando la semántica operacional para definir el comportamiento de los programas a través de un sistema de transiciones.

1. (1 punto) Agrega en el *README* la definición de la semántica dinámica de los operadores booleanos y de relación.

   Sugerencia: Utiliza el comando `\infer` del paquete `proof` para escribir los juicios.

   \infer[Nombre del juicio]{Conclusión}{Premisas}

Implementa las siguientes funciones:

Nota: Para realizar esta práctica deberán importar el módulo `Sintax.hs` de alguno de los integrantes del equipo.

1. (2 puntos) `eval1`. Devuelve la transición tal que `eval1 e = e'` syss $e \to e'$.

```
eval1 :: Expr -> Expr
```

Ejemplo:

```
*Main> eval1 (Add (I 1) (I 2))
I 3
*Main> eval1 (Let "x" (I 1) (Add (V "x") (I 2)))
Add (I 1) (I 2)
```

2. (2 puntos) `evals`. Devuelve la transición tal que `evals e = e'` syss $e \to^* e'$ y e' está bloqueado.

```
evals :: Expr -> Expr
```

Ejemplo:

```
*Main> evals (Let "x" (Add (I 1) (I 2)) (Eq (V "x
    ") (I 0)))
B False
*Main> evals (Add (Mul (I 2) (I 6)) (B True))
Add (I 12) (B True)
```

3. (1 punto) `evale`. Devuelve la evaluación de un programa tal que `evale e = e'` syss $e \to^* e'$ y e' es un valor. En caso de que e' no esa un valor deberá mostrar un mensaje de error particular del operador que lo causó.

```
evale :: Expr -> Expr
```

Ejemplo:

```
*Main> evale (Add (Mul (I 2) (I 6)) (B True))
*** Exception: [Add] Expects two Integer.
*Main> evale (Or (Eq (Add (I 0) (I 0)) (I 0)) (Eq
     (I 1) (I 10)))
B True
```

2 Semántica Estática

La semántica estática es la que determina cuándo un programa está bien definido mediante criterios sintácticos que son sensibles al contexto, requiriendo que cada variable sea declarada antes de usarse.

La verificación de la correctud estática de un programa se hace a través del sistema de tipos. Un sistema de tipos consiste en una colección de reglas de inferencia que imponen ciertas restricciones en la formación de programas.

Para las expresiones `EAB` definiremos los siguientes tipos:

```
data Type = Integer | Boolean
```

1. (1 punto) Agrega en el *README* la definición de la semántica estática de los operadores booleanos y de relación.

Ahora modelaremos el contexto como una lista de pares que almacenen el nombre de las variables junto con su tipo.

```
type Decl = (Identifier, Type)
type TypCtxt = [Decl]
```

Implementa la siguiente función:

1. (2 puntos) `vt`. Verifica el tipado de un programa tal que `vt Γ e T = True` syss $\Gamma \vdash e : T$.

   ```
   vt :: TypCtxt -> Expr -> Type -> Bool
   ```

   Ejemplo:

   ```
   *Main> vt [("x",Boolean)] (If (B True) (B False)
       (Var "x")) Boolean
   True
   *Main> vt [] (Let "x" (Add (I 1) (I 2))
   (Eq (Mul (Add (V "x") (I 5)) (I 0)) (Add (V "x")
       (I 2))))
   Boolean
   True
   ```

2. (1 punto) `eval`. Verifica el tipado de un programa, y en caso de ser correcto lo evalúa hasta obtener un valor.

   ```
   eval :: Expr -> Type -> Expr
   ```

   Ejemplo:

   ```
   *Main> eval (Let "x" (B True) (If (B True) (B
       False) (Var "x"))) Boolean
   B False
   *Main> eval (Add (Mul (I 2) (I 6)) (B True))
       Integer
   *** Exception: Type error.
   ```

3 Evaluador de `EAB`

Ya que hemos creado el núcleo del evaluador de programas en `EAB`, lo integraremos con un analizador sintáctico el cual nos permitirá escribir programas en texto plano y poder evaluarlos.

Para hacer esto descarga el archivo `BAE.zip` del sitio del curso. Este último tiene la siguiente estructura:

```
BAE
 |--- demo
 |      |-- Example.bae
```

```
|--- Makefile
|--- src
     |-- BAE
     |    |-- Parser.hs
     |-- Main.hs
```

- `Example.bae`. Contiene un ejemplo de un programa en `EAB`.
- `Parser.hs`. Módulo de `Haskell` que contiene la implementación del analizador sintáctico de programas en `EAB`.
- `Main.hs`. Módulo de `Haskell` que contiene la descripción del `main`, el cuál, en términos generales, recibirá una ruta a un archivo con un programa en `EAB`, lo intentará evaluar y mostrará en pantalla su resultado.
- `Makefile`. Define un conjunto de tareas para compilar todos los recursos y generar un archivo ejecutable.

1. (Requisito de entrega) Exporta este módulo como `Semantic` y agrégalo junto con el anterior (`Sintax`) al directorio `src/BAE`. Compila el evaluador ejecutando `make` desde el directorio raíz. Se generará un archivo ejecutable llamado `BAEi`.

   Ejecuta `make run`. Si todo funciona como debería obtendrás una salida como la siguiente:

   ```
   $ make run

   mkdir -p build
   ghc src/Main.hs -isrc -outputdir build/ -o BAEi
   ./BAEi ./demo/goodExample.bae
   Program:
   let(mul(add(N[1], N[2]), succ(pred(N[10]))), x1.
       let(eq(N[1], N[1]), y.if(or(and(B[True], gt(V
       [x1], N[-4])), B[False]), or(not(V[y]), lt(V[
       x1], N[31])), lt(V[x1], N[-4])))) : Boolean
   Evaluation:
   B[True]
   ```

Atención

Será un **requisito de entrega** integrar correctamente sus módulos `Sintax` y `Semantic` al analizador sintáctico.

¡Suerte!