Ejercicio Semanal 4

Implementación de la máquina $\mathcal{K}$

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

Miércoles 7 de octubre de 2018

Fecha de entrega: Miércoles 14 de noviembre de 2018 a las 23:59:59.

Como ya se discutió en clase, la técnica para definir la semántica dinámica de un lenguaje que se ha usado hasta ahora es bastante conveniente para poder probar distintas propiedades de los lenguajes, sin embargo, resulta poco favorable para usarla como guía de la implementación.

Recordemos que su definición se da a través de un sistema de transiciones que utiliza reglas que determinan dónde aplicar la siguiente instrucción, pero sin especificar cómo encontrarla dentro de la expresión. Para hacer este proceso explícito, se introduce el mecanismo llamado pila de control, que registrará el trabajo que queda por hacer después de que se ejecuta una instrucción. Usando esta pila se elimina la necesidad de contar con premisas en las reglas de transición, de tal modo que el sistema de transiciones define una máquina abstracta cuyos pasos están determinados por la información que se encuentra en el estado, tal como lo haría cualquier computadora. En este ejercicio semanal implementaremos la máquina abstracta $\mathcal{K}$ para evaluar expresiones `EAB`.

1 La máquina $\mathcal{K}$

1.1 Expresiones y Marcos

Las expresiones de `EAB` para este ejercicio semanal son las siguientes:

```
data Expr = V Identifier | I Int | B Bool
  | Add Expr Expr | Mul Expr Expr | Succ Expr | Pred Expr
  | And Expr Expr | Or Expr Expr | Not Expr
  | Lt Expr Expr | Gt Expr Expr | Eq Expr Expr
  | If Expr Expr Expr
  | Let Identifier Expr Expr
  | Error

type Identifier = String
```

Los marcos son esqueletos estructurales que registran los cómputos pendientes de manera explícita. Se definen del siguiente modo:

- Operadores binarios

  $\overline{op(-, e_2) \text{ marco}}$

  $\overline{op(v_1, -) \text{ marco}}$

- Operadores unarios

  $\overline{op(-) \text{ marco}}$

- Condicional booleano

  $\overline{if(-, e_1, e_2) \text{ marco}}$

- Instrucción `let`

  $\overline{let(-, x, e) \text{ marco}}$

El espacio marcado con un guión indica la posición correspondiente al valor devuelto en la evaluación actual, el cual corresponde al lugar donde se está llevando a cabo la evaluación.

Modelaremos esto del siguiente modo:

```
type Pending = ()

data Frame = AddL Pending Expr
           | AddR Pending Expr
           | MulL Pending Expr
           | MulR Pending Expr
           | SuccF Pending
           | PredF Pending
           ...
```

Implementa las siguientes funciones:

1. (1/3 punto) Crea una instancia de la clase `Show` para las expresiones `EAB` de acuerdo a su **sintaxis abstracta**.

2. (1/3 punto) Crea una instancia de la clase `Show` para los marcos de acuerdo a la sintaxis descrita en la nota 11 del curso.

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

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

   Ejemplo:

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

4. (1 punto) `subst`. Realiza la sustitución en una expresión `EAB`.

   ```
   subst :: Expr -> Substitution -> Expr
   ```

   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.
   ```

   Recuerda que la definición de sustitución es:

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

1.2 Estados, transiciones y pilas de control

Un estado s de la máquina $\mathcal{K}$ consiste de una pila de control k y una expresión cerrada e que toma alguna de las siguientes formas:

1. Una *evaluación* de la forma $k \triangleright e$ que corresponde a la evaluación de la expresión cerrada e sobre la pila de control k.
2. Un *retorno* de la forma $k \triangleleft e$, donde e es un valor, que corresponde a la evaluación de la pila de control k sobre la expresión cerrada e.

Modelaremos esto del siguiente modo:

```
State = E (Stack, Expr) | R (Stack, Expr)
```

Donde la pila de control será una lista que almacenará los marcos de las expresiones.

```
Stack = [Frame]
```

Finalmente, las transiciones se definen inductivamente del siguiente modo:

- Valores

$$\frac{}{k \triangleright v \rightarrow_{\mathcal{K}} k \triangleleft v}$$

- Operadores binarios

$$\frac{}{k \triangleright op(e_1, e_2) \rightarrow_{\mathcal{K}} k; op(-, e_2) \triangleright e_1}$$

$$\frac{}{k; op(-, e_2) \triangleleft v \rightarrow_{\mathcal{K}} k; op(v, -) \triangleright e_2}$$

$$\frac{}{k; op(v_1, -) \triangleleft v_2 \rightarrow_{\mathcal{K}} k \triangleleft op_p(v_1, v_2)}$$

op_p es el operador primitivo.

- Operadores unarios

$$\frac{}{k \triangleright op(e) \rightarrow_{\mathcal{K}} k; op(-) \triangleright e}$$

$$\frac{}{k; op(-) \triangleleft v \rightarrow_{\mathcal{K}} k \triangleleft op_p(v)}$$

op_p es el operador primitivo.

- Condicional booleano

$$\frac{}{k \triangleright if(e_1, e_2, e_3) \rightarrow_{\mathcal{K}} k; if(-, e_2, e_3) \triangleright e_1}$$

$$\frac{}{k; if(-, e_2, e_3) \triangleleft true \rightarrow_{\mathcal{K}} k \triangleright e_2}$$

$$\frac{}{k; if(-, e_2, e_3) \triangleleft false \rightarrow_{\mathcal{K}} k \triangleright e_3}$$

- Instrucción `let`

$$\frac{}{k \triangleright let(e_1, x.e_2) \rightarrow_{\mathcal{K}} k; let(-, x.e_2) \triangleright e_1}$$

$$\frac{}{k; let(-, x.e_2) \triangleleft v \rightarrow_{\mathcal{K}} k \triangleright e_2[x := v]}$$

Como podemos observar evaluar cualquier valor x es simplemente regresarlo. Para evaluar cualquier operador $op(e)$, agregamos el marco correspondiente a la pila de control y evaluamos la expresión e; cuando esta se regresa como e', regresamos $op(e')$ a la pila de control original.

Implementa las siguientes funciones:

1. (1/3 punto) Crea una instancia de la clase `Show` para los estados de acuerdo a la sintaxis descrita anteriormente.

2. (4 puntos) `eval1`. Recibe un estado de la máquina $\mathcal{K}$ y devuelve un paso de la transición. Es decir:

   `eval1 (E (s, e)) = E (s', e')` syss $s \triangleright e \rightarrow_{\mathcal{K}} s' \triangleright e'$

   `eval1 (R (s, e)) = R (s', e')` syss $s \triangleleft e \rightarrow_{\mathcal{K}} s' \triangleleft e'$

   `eval1 (E (s, e)) = R (s', e')` syss $s \triangleright e \rightarrow_{\mathcal{K}} s' \triangleleft e'$

   `eval1 (R (s, e)) = E (s', e')` syss $s \triangleleft e \rightarrow_{\mathcal{K}} s' \triangleright e'$

   ```
   eval1 :: State -> State
   ```

   Ejemplo:

```
*Main> eval1 (E ([], Add (I 1) (I 2)))
E ([AddL Pending (I 2)], I 1)
*Main> eval1 (R ([OrR Pending (B True), AndL Pending (B False)],
B False))
R ([AndL Pending (B False)], B True)
*Main> eval1 (R ([Succ Pending], B False))
R ([], Error)
```

Nota: En caso de no poder evaluar la expresión se deberá devolver $\emptyset \triangleleft$ *error*.

3. (2 puntos) `evals`. Recibe un estado de la máquina $\mathcal{K}$ y devuelve un estado derivado de evaluar varias veces hasta obtener la pila vacía. Es decir:

   `evals (E (s, e)) = R ([], e')` syss $s \triangleright e \rightarrow_{\mathcal{K}}^{*} \emptyset \triangleleft e'$

   ```
   evals :: State -> State
   ```

   Ejemplo:

   ```
   *Main> evals (E ([], Let "x" (I 2) (Mul (Add (I 1) (V "x")) (V "x"))))
   R ([], I 6)
   *Main> evals (E ([], Let "x" (B True) (If (V "x") (V "x") (B False))))
   R ([], B True)
   *Main> evals (E ([], Add (B True) (I 1)))
   R ([], Error)
   ```

4. (1 punto) `eval`. Recibe una expresión `EAB`, la evalúa con la máquina $\mathcal{K}$ y devuelve un valor si, iniciando con la pila vacía esta devuelve un valor a la pila vacía. Es decir:

   `eval e = e'` syss $\emptyset \triangleright e \rightarrow_{\mathcal{K}}^{*} \emptyset \triangleleft e'$ y e' es un valor.

   En caso contrario manda un mensaje de error.

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

   Ejemplo:

   ```
   *Main> eval (Let "x" (I 1) (If (Gt (V "x") (I 0)) (Eq (V "x") (I 0))
   (B True)))
   B False
   *Main> eval (Not (I 3))
   *** Exception: Error.
   ```

¡Suerte!