Práctica Extra

Implementación de la semántica operacional de paso grande

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

Lunes 3 de diciembre de 2018

Fecha de entrega: Domingo 9 de diciembre de 2018 a las 23:59:59.

Reglas:

Los lineamientos de entrega serán los mismos de las prácticas anteriores salvo los siguientes:

1. La calificación de esta práctica sustituirá a la menor calificación del laboratorio, siempre y cuando sea **aprobatoria**.
2. El desarrollo y la entrega deberá ser de manera **individual**.
3. La fecha de entrega es definitiva. **No habrá retardos ni prorrogas**.

A lo largo del curso la semántica dinámica se definió a través de la semántica operacional de paso pequeño, la cuál consiste en evaluar las expresiones un paso a la vez. Como alternativa a esta, existe otro estilo de evaluar las expresiones que consiste en definir de forma completa cómo se evalúa una expresión hasta llegar a un valor. A este estilo se le denomina semántica operacional de paso grande, o semántica natural, y se denota con el símbolo $\Downarrow$.

1 Sintaxis

Usaremos las expresiones del lenguaje `EAB`.

```
type Identifier = String

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

Pero añadiremos los pares:

```
...
| Pair Expr Expr
| Fst Expr | Snd Expr
```

y las funciones lambda:

```
...
| Lam Identifier Expr
| App Expr Expr
```

Implementa las siguientes funciones:

1. (0.5 puntos) Crea una instancia de la clase `Show` en sintaxis abstracta.

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

2. (1 punto) `frVars`. Obtiene el conjunto de variables libres de una expresión.

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

   Ejemplo:

```
*Main> frVars (Snd (Pair (V "x") (V "y")))
["x", "y"]
*Main> frVars (Let "x" (Add (V "x") (I 10))
(Mul (I 0) (Add (V "x") (V "y"))))
["x", "y"]
```

3. (1 punto) `subst`. Aplica la sustitución a la expresión dada en caso de ser posible.

   Recuerda la definición de la sustitución: `type Substitution = ( Identifier, Expr`.

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

   Ejemplo:

```
*Main> subst (Fst (Pair (V "x") (V "x"))) ("x", I 1)
Pair (I 1) (I 1)
*Main> subst (Let "x" (Pred (V "x")) (Add (I 1) (V "x"))) ("x", I 90)
Let "x" (Pred (I 90)) (Add (I 1) (V "x"))
*Main> subst (Lam "x" (Or (V "x") (V "z"))) ("z", And (B True) (V "x"))
*** Exception: Could not apply the substitution.
```

2 Semántica

Antes de definir las reglas de semántica, primero hay que definir qué expresiones representarán a los valores. Comúnmente a estas expresiones se les dice que están en forma canónica (C).

- Los enteros están en forma canónica, *i.e.* $n \in C$.
- Los booleanos están en forma canónica, *i.e.* $b \in C$.
- Los pares de formas canónicas están forma canónica, *i.e.* $(c_1, c_2) \in C$ si $c_1 \in C$ y $c_2 \in C$.
- Las abstracciones cerradas están en forma canónica, *i.e.* $\lambda x.t \in C$ si $\lambda x.t$ está cerrada. **Nota**: Una expresión está cerrada si no contiene variables libres.

Ahora podemos dar las reglas de evaluación de la forma $e \Downarrow c$, donde e es una expresión cerrada y c está en forma canónica.

Formas Canónicas.

$$\frac{c \in C}{c \Downarrow c}$$

Operadores aritméticos (Unarios).

$$\frac{e \Downarrow n}{\circ e \Downarrow \circ n}$$

Operadores aritméticos y de relación (Binarios):

$$\frac{e_1 \Downarrow n_1 \quad e_2 \Downarrow n_2}{e_1 \diamond e_2 \Downarrow n_1 \diamondsuit n_2}$$

Operadores booleanos (Unarios).

$$\frac{e \Downarrow b}{\circ e \Downarrow \circ b}$$

Operadores booleanos (Binarios):

$$\frac{e_1 \Downarrow b_1 \quad e_2 \Downarrow b_2}{e_1 \diamond e_2 \Downarrow b_1 \diamondsuit b_2}$$

Condicional:

$$\frac{e_0 \Downarrow false \quad e_1 \Downarrow c_1}{if\ e_0\ then\ e_1\ else\ e_2\ \Downarrow c_1}$$

$$\frac{e_0 \Downarrow true \quad e_2 \Downarrow c_2}{if\ e_0\ then\ e_1\ else\ e_2\ \Downarrow c_2}$$

Expresión `Let`:

$$\frac{e_1 \Downarrow c_1 \quad e_2[x := c_1] \Downarrow c_2}{let\ x \Leftarrow e_1\ in\ e_2\ \Downarrow c_2}$$

Pares:

$$\frac{e_1 \Downarrow c_1 \quad e_2 \Downarrow c_2}{(e_1, e_2) \Downarrow (c_1, c_2)}$$

$$\frac{e \Downarrow (c_1, c_2)}{fst(e) \Downarrow c_1}$$

$$\frac{e \Downarrow (c_1, c_2)}{snd(e) \Downarrow c_2}$$

Funciones lambda:

$$\frac{e_1 \Downarrow \lambda x.e_1' \quad e_2 \Downarrow c_2 \quad e_1'[x := c_2] \Downarrow c}{(e_1 e_2) \Downarrow c}$$

Implementa las siguientes funciones:

1. (4 puntos) `evals`. Devuelve la evaluación de una expresión implementando las reglas anteriores.

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

   Ejemplo:

   ```
   *Main> evals (Pair (I 0) (B False))
   Pair (I 0) (B False)
   *Main> evals (Add (I 10) (Mul (Pred (I 2)) (Succ (I 0))))
   I 11
   *Main> evals (Let "c" (Eq (I 10) (I 2)) (If (V "c")
   (And (B True) (V "c")) (Or (B False) (V "c"))))
   B False
   ```

2. (3 puntos) `vt`. Verifica el tipado de una expresión tal que `vt Γ e T = True` syss $\Gamma \vdash e : T$.

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

Las definiciones de los tipos y el contexto son:

```
data Type = Integer
          | Boolean
          | Prod Type Type
          | Func Type Type

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

Ejemplo:

```
*Main> ["x", Boolean] (Pair (I 10) (V "x"))
(Prod Integer Boolean)
True
*Main> ["x", Integer] (Lam "x" (Eq (I 0) (V "x")))
(Func Integer Boolean)
True
*Main> [] (If (Eq (I 0) (I 0)) (B False) (I 10))
Integer
False
```

3. (0.5 puntos) `eval`. Devuelve la evaluación de una expresión sólo si esta bien tipada.

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

Ejemplo:

```
*Main> eval (If (Eq (I 0) (I 0)) (B False) (I 10))
*** Exception: Invalid expression.
*Main> eval (Let "c" (Eq (I 10) (I 2)) (If (V "c")
(And (B True) (V "c")) (Or (B False) (V "c"))))
B False
```

4. (1 punto) Agrega la expresión `Rec Identifier Expr` al lenguaje. Esta expresión representará a las funciones recursivas ($rec\ y.(\lambda x.t)$).

   Ejemplo:

```
Rec "fact" (Lam "x" (If (Eq (V "x") (I 0)) (I 1)
(Mul (V "x") (App (V "fact") (Pred (V "x"))))))
```

   Debes investigar cómo definir su semántica operacional de paso grande.

¡Suerte!