Practica Extra 2

Emiliano Galeana Araujo

Facultad de Ciencias, UNAM

Fecha de entrega: 9 de Diciembre de 2018

1. Descripción del programa

Hemos estudiado la semántica operacional de paso pequeño, la cual consiste en evaluar 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, llamamos a esto semántica operacional de paso grande.

```
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
          | Pair Expr Expr
          | Fst Expr | Snd Expr
          | Lam Identifier Expr
          | App Expr Expr
          | Rec Identifier Expr deriving (Eq)
```

Las siguientes son funciones recursivas para el tipo de dato `Expr`.

```
-- | frVars. Obtiene el conjunto de variables libres de una expresion.
frVars :: Expr → [Identifier]

-- | subst. Aplica una sustitucion.
subst :: Expr → Substitution → Expr
```

```
-- | evals. Devuelve la evaluacion de una expresion implementando las reglas
-- |        anteriores(PDF).
evals :: Expr → Expr

-- | vt. Funcion que verifica el tipado de un programa.
vt :: TypCtx → Expr → Type → Bool

-- | eval. Funcion que devuelve la evaluacion de una expresion solo si esta
-- |       bien tipada.
eval :: Expr → Expr
```

2. Entrada y ejecución

El programa es interpretado por `GCHI` de la siguiente forma

```
~:ghci Extra02.hs
```

Ya en el programa, los siguientes son ejemplos de ejecución de paso grande `Snoc`.

```
*Extra02> frVars (Snd (Pair (V ‘‘x’’’) (V ‘‘y’’’)))
 [‘‘x’’’, ‘‘y’’’]
*Extra02> frVars (Let ‘‘x’’’ (Add (V ‘‘x’’’) (I 10))
                             (Mul (I 0) (Add (V ‘‘x’’’) (V ‘‘y’’’))))
 [‘‘x’’’, ‘‘y’’’]
*Extra02> subst (Fst (Pair (V ‘‘x’’’) (V ‘‘x’’’))) (‘‘x’’’, I 1)
Pair (I 1) (I 1)
*Extra02> subst (Lam ‘‘x’’’ (Or (V ‘‘x’’’) (V ‘‘z’’’)))
                                          (‘‘z’’’, (And (B True)(V ‘‘x’’’)))
 *** Exception: Could not apply the substitution.
*Extra02> evals (Pair (I 0) (B False))
 Pair (I 0) (B False)
*Extra02> evals (Add (I 10) (Mul (Pred (I 2)) (Succ (I 0))))
 I 11
*Extra02> vt [(‘‘x’’’, Integer)] (Lam ‘‘x’’’ (Eq (I 0) (V ‘‘x’’’)))
                                  (Func Integer Boolean)
 True
*Extra02> vt [] (If (Eq (I 0) (I 0)) (B False) (I 10)) Integer
 False
*Extra02> eval (Let ‘‘c’’’ (Eq (I 10) (I 2)) (If (V ‘‘c’’’) (And (B True) (V‘‘c’’’)
```

```
                                                    (Or (B False)(V‘‘c’’’))))
  B False
```

3. Conclusiones

Creo que implementar la semántica operacional de paso grande es más sencillo, al menos por la parte donde no es necesario escribir todos los pasitos. Pero creo que para mostrar errores sería mejor con la otra, ya que aquí los podemos mostrar, pero no como lo implementamos en la otra.

Referencias

[1] Archivero, curso de Lenguajes de Programación 2019-1