Práctica 3: Cálculo lambda sin tipos

Luis Daniel Aragón Bermúdez Joshua Jair Pedrero Gómez

Miércoles 3 de octubre de 2018

Introducción

Descripción del programa

El programa es la implementación del cálculo lambda. El cálculo lambda (λ) consiste de tres términos y todas las combinaciones recursivas válidas de estos mismos.

- Var - Una variable.
- Lam - Una abstracción.
- App - Una aplicación.

Sintáxis (λ)

Se definen las expresiones del cálculo lambda del siguiente modo:

```
type Identifier = String

data Expr = Var Identifier
          | Lam Identifier Expr
          | App Expr Expr
```

Nosotros denotaremos la lambda por la diagonal (\$, el cuerpo con (->), y la aplicación con espacio encerrando las expresiones en un paréntesis ((e1 e2)). Las variables serán nombradas únicamente con caracteres alfanuméricos (la parte que es un número deberá ser precedida de la parte alfabética).

Se creó una instancia de la clase `Show` para las expresiones lambda.

```
instance Show Expr where
  show e = case e of
    Var x   -> x
    Lam x e -> "\\"++ x ++ " -> " ++ show e
    App x y -> "("++show x ++ " " ++ show y ++ ")"
```

Ejecución

Utilizando cabal de Cabal Development Team (2018) es sencillo cargar los ejemplos usando el siguiente comando:

```
$ cabal repl
```

Y una vez en la REPL, puede correr cada una de las funciones en el paquete.

Implementación

Sustitución y α-equivalencia

Sustitución

La evaluación de un término lambda $(\lambda x \epsilon) a$ consiste en sustituir todas las ocurrencias libres de x en ϵ por el argumento a. A este paso de la sustitución se le llama reducción. La sustitución se denota como $[x := a]$ y se define del siguiente modo:

$$x[x := a] = a$$

$$y[x := a] = y \succ x \neq y$$

$$ee'[x := a] = (e[x := a])(e'[x := a])$$

$$\lambda x \cdot e[x := a] = \lambda x \cdot e$$

$$\lambda y \cdot e[x := a] = \lambda y \cdot (e[x := a]) \; si \; x \neq y \wedge y \notin frVars(a)$$

Definimos el tipo de sustitución:

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

α-equivalencia

La alfa equivalencia es la propiedad de cambiar la variable ligada, junto con todas sus ocurrencias libres dentro del cuerpo sin cambiar el significado de la expresión.

$$\lambda x e \equiv^{\alpha} \lambda y (e[x := y])$$

En esta parte se implementaron las siguientes funciones:

```
-- | frVars
-- | Función que obtiene el conjunto de variables libres de una expresión.
frVars :: Expr -> [Identifier]

-- | lkVars
-- | Función que obtiene el conjunto de variables de una expresión.
lkVars :: Expr -> [Identifier]

-- | incrVar
-- | Función que dado un identificador,
-- | si este no termina en numero le agrega el sufijo 1,
-- | en caso contrario toma el valor del numero y lo incrementa en 1.
incrVar :: Identifier -> Identifier

-- | alphaExpr
-- | Función que toma una expresión lambda
-- | y devuelve una alfa-equivalencia utilizando la función incrVar
-- | hasta encontrar un nombre que no aparezca en el cuerpo.
alphaExpr :: Expr -> Expr
```

```haskell
-- | subst
-- | Función que aplica la sustitución a la expresión dada.
subst :: Expr -> Substitution -> Expr
```

β-reducción

La beta reducción es simplemente un paso de sustitución remplazando la variable ligada por una expresión lambda por el argumento de la aplicación.

$$(\lambda x a)y \rightarrow^{\beta} a[x := y]$$

Evaluación

La estrategia de evaluación de una expresión consistirá en aplicar la beta reducción hasta que ya no sea posible, usando las siguientes reglas:

$$\frac{t \rightarrow t'}{\lambda x t \rightarrow \lambda x t'} \ (Lam)$$

$$\frac{t_1 \rightarrow t_1'}{t_1 t_2 \rightarrow t_1' t_2'} \ (App1)$$

$$\frac{t_1 \rightarrow t_1'}{(\lambda x t) t_1 \rightarrow (\lambda x t) t_1'} \ (App2)$$

$$\frac{}{(\lambda x t)y \rightarrow^{\beta} t[x := y]} \ (Beta)$$

Y se implementaron las siguientes funciones:

```haskell
-------------------------------------------------
-------------   beta-reducción  ----------------
-------------------------------------------------

-- | beta.
-- | Función que aplica un paso de la beta reducción.
beta :: Expr -> Expr

-- | locked.
-- | Función que determina si una expresión esta bloqueada,
-- | es decir, no se pueden hacer mas beta reducciones.
locked :: Expr -> Bool

-- | eval.
-- | Función que evalúa una expresión lambda
-- | aplicando beta reducciones hasta quedar bloqueada.
eval :: Expr -> Expr
```

Bibliografía

Cabal Development Team. 2018. «Cabal: A framework for packaging Haskell software». http://hackage.haskell.org/package/Cabal.