Practica Extra 1

Emiliano Galeana Araujo

Facultad de Ciencias, UNAM

Fecha de entrega: 9 de Diciembre de 2018

1. Descripción del programa

Dada una expresión lambda, representaremos variables ligadas apuntando directamente al símbolo lambda que la liga en el árbol de sintaxis abstracta correspondiente, es decir, mediante el número de lambdas que es necesario "saltar" hasta encontrar la lambda que liga a la variable en cuestión. Estos números, son conocidos como los números de `Brujin`.

```
type Identifier = String

type Index = Int

type Substitution = (Index, ExprB)

data ExprL = VL Identifier
           | LL Identifier ExprL
           | AL ExprL ExprL deriving (Show)

data ExprB = IB Index
           | LB ExprB
           | AB ExprB ExprB deriving (Show)
```

Las siguientes son funciones recursivas para numeros de `Brujin`.

```
-- | ctx. Funcion que obtiene el contexto canonico de una expresion.
ctx :: ExprL → [Identifier]
```

```
-- | qn. Dado un contexto de indices y una expresion lambda obtiene su
-- |     representacion anonima.
qn :: ([Identifier], ExprL) → ExprB

-- | newVar. Dado un contexto de nombres, obtiene una nueva variable y la
-- |         agrega al contexto.
newVar :: [Identifier] → [Identifier]

-- | pn. Dado un contexto de nombres y una expresion anonima devuelve su
-- |     representacion correspondiente en el calculo lambda con nombres.
pn :: ([Identifier], ExprB) → ExprL

-- | shift. Desplaza los indices de una expresion anonima dado un parametro de
-- |        corte.
shift :: (Int, Int, ExprB) → ExprB

-- | subst. Aplica la substitucion a la expresion anonima.
subst :: ExprB → Substitution → ExprB

-- | eval1. Aplica un paso de la reduccion de una expresion anonima.
eval1 :: ExprB → ExprB

-- | locked. Determina si una expresion anonima esta bloqueada es decir, no se
-- |         pueden hacer mas reducciones.
locked :: ExprB → Bool

-- | eval. Evalua una expresion anonima hasta quedar bloqueada.
eval :: ExprB → ExprB
```

2. Entrada y ejecución

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

```
~:ghci Extra01.hs
```

Ya en el programa, los siguientes son ejemplos de ejecución.

```
*Extra01> ctx (LL "x" (LL "y" (AL (VL "u" ) (AL (VL "x" )
              (AL (VL "y" ) (AL (VL "␣z␣" ) (AL (VL "␣z␣" )
              (AL (VL "y" ) (VL "v" ) ) ) ) ) ) ) ) )
[‘‘v’’’, ‘‘z’’’, ‘‘u’’’]
```

```
*Extra01> ctx (AL (VL "u" ) (AL (VL "v" ) (AL (VL "x" )
              (AL (VL "y" )(AL (VL "z" ) (AL (VL "x" )
              (AL (VL "x" ) (VL "v" ) ) ) ) ) ) ) )
[‘‘v’’’, ‘‘x’’’, ‘‘z’’’, ‘‘y’’’, ‘‘u’’’]

*Extra01> qn ( [ "x" ] , LL "␣z␣" (AL (VL "␣z␣" ) (AL (VL "x" )
             (LL "y" (AL (VL "␣z␣" )(AL (VL "x" ) (VL "y" ) ) ) ) ) ) )
LB (AB ( IB 0 ) (AB ( IB 1 ) (LB (AB ( IB 1 ) (AB ( IB 2 ) ( IB 0 ) ) ) ) ) )

*Extra01> qn ( [ "y" , "x" ] , LL "␣z␣" (AL (VL "␣z␣" )
             (AL (VL "x" ) (VL "y" ) ) ) )
LB (AB ( IB 0 ) (AB ( IB 2 ) ( IB 1 ) ) )

*Extra01> newVar [‘‘x’’’, ‘‘z’’’’’
[‘‘x’’’, ‘‘y’’’, ‘‘z’’’]

*Extra01> newVar [‘‘x’’’, ‘‘x2’’’, ‘‘x1’’’]
[‘‘x3’’’, ‘‘x’’’, ‘‘x2’’’, ‘‘x1’’’]

*Extra01> pn ( [ ‘‘x’’’ , LB (AB ( IB 0 ) (AB ( IB 1 )
             (LB (AB ( IB 1 ) (AB ( IB 2 )( IB 0 ) ) ) ) ) ) )
LL‘x1’’’ (AL (VL  "x1"  ) (AL (VL "x" ) (LL "x2""␣(AL
␣␣␣␣␣␣␣␣␣␣␣␣(VL␣"x1"␣)␣(AL␣(VL␣"x"␣)␣(VL␣␣"x2""  ) ) ) ) ) )

*Extra01> pn ( [ "y" , "x" ] , LB (AB ( IB 0 ) (AB ( IB 2 ) ( IB 1 ) ) ) )
LL "x1" (AL (VL "x1" ) (AL (VL "x" ) (VL "y" ) ) )

*Extra01> shift ( 1 , 0 , LB (AB ( IB 0 ) ( IB 2 ) ) )
LB (AB ( IB 0 ) ( IB 3 ) )

*Extra01> shift ( 1 , 2 , LB (AB ( IB 0 ) (AB ( IB 2 ) ( IB 1 ) ) ) )
LB (AB ( IB 0 ) (AB ( IB 2 ) ( IB 1 ) ) )

*Extra01> subst (LB (AB ( IB 0 ) (AB ( IB 2 ) ( IB 1 ) ) ) )
                           ( 1 , LB (AB ( IB 0 )( IB 2 ) ) )
LB (AB ( IB 0 ) (AB (LB (AB ( IB 0 ) ( IB 3 ) ) ) ( IB 1 ) ) )

*Extra01> eval1 (AB (LB (AB (LB ( IB 1 ) ) ( IB 0 ) ) )
                (LB (AB ( IB 2 ) (AB ( IB 1 )( IB 0 ) ) ) ) )
```

```
AB (LB (LB (AB ( IB 3 ) (AB ( IB 2 ) ( IB 0 ) ) ) ) )
(LB (AB ( IB 2 ) (AB ( IB 1 )( IB 0 ) ) ) )

*Extra01> locked ( IB 1 )
True

*Extra01> locked (LB (AB (LB ( IB 0 ) ) (LB ( IB 0 ) ) ) )
False
```

3. Conclusiones

Estuvo bien, al principio creí que no me saldría, pues no entendía bien qué había que hacer, pero después de leer el PDF de *deBrujin* entendí y ya fue fácil hacerla. Además de que ayudó haber implementado algunas funciones en otras prácticas, como la 3, que tenía que ver con lambdas, me ayudó a hacer el eval, locked y eval1.

Referencias

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