Práctica 3

Marco Antonio Orduña Avila y Enrique Sanchez Lara

Facultad de Ciencias, UNAM

2 de Octubre de 2019

1. Descripción del programa

El programa lo que hara es extender el lenguaje de EAB con la implementación del cálculo lambda y hacerlo un lenguaje de programación funcional, habra 3 cosas que tomar en cuenta, la Sintaxis, la Semántica y la Funciones Recursivas, para el de sintaxis cambiamos la abtracción lambda a la que llamaremos función, para la Semantica tenemos que definir la Beta reducción y para las Funciones recursivas tenemos que implementar el llamado combinador de punto fijo.

2. Entrada y ejecución

Para ejecutar el programa solo basta entrar al archivo donde se ubica main.hs y usar el intérprete de haskell, pues main.hs importa las demas clases que se necesitan, para las funciones:

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

Prelude> frVars (Fn "f" (App (App (V "f" ) (Fn "x" (App (App (V "f" ) (V "x" ) )
debe regresar [ ]

incrVar :: Identifier -> Identifier
Prelude> incrVar  "x97"
Debe regresar "x98"

alphaExpr :: Expr -> Expr

Prelude> alphaExpr (Fn "x" (Fn "x1" (App (V "x" ) (V "x1" ))))
debe de regresar fn ( x1.fn( x2.app (V[x1],V[x2])))

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

```
Prelude> subst(Fn "x" (V "y" ) ) ( "y" , V "x" )
debe de regresar fn (x1.V[x])

eval1 :: Expr -> Expr

Prelude> eval1 ( Lt (( App (Fn "x" (Add (V "x" ) ( I
20 )))( I 10 ))) (I 20))
debe de regresar lt(add(I[10] ,I[20]),I[20])



evals :: Expr -> Expr

Prelude> evals (App(App(Fn "x"(If(Or(V "x" ) (B False))(Add(I1)(I1))(Succ(Pred(I
)))(B False))(V "x"))
debe de regresar app (if(or(B[False],B[False]),add(I[1],I[1]),succ(pred(N[0]))),

evale :: Expr -> Expr

Prelude> evale(Let"f"(Fn"x"(If(Or(V"x")(B False ))(Add(I 1)(I 1))(Succ(Pred(I 0)
))(App(V "f")(B False)))
debe de regresar I [0]
```

3. Conclusiones

Es facil ver que para definir funciones recursivas basta usar un operador de punto fijo. La propiedad de terminación no es válida en el cálculo lambda puro, al existir expresiones e que no cuentan con una forma normal.
El cálculo lambda tiene una naturaleza no determinista, el uso de expresiones λ si bien proporciona una notación lineal simple para definir funciones.
puede resultar dificil de manejar para el programador. Para evitar su uso directo podemos introducir una definición que se asemeje a la declaración de funciones en lenguajes de programación actuales.
Por ejemplo podemos agregar a la sintaxis concreta una declaración de la forma $fun(x : T) \Rightarrow e$ para denotar a la función x $\longrightarrow$ e. Es decir, el uso de la declaración fun es simplemente una abreviatura:
Este proceso de definición se conoce como azúcar sintáctica