Compiladores 2020-1
Facultad de Ciencias UNAM
Práctica 6: Tablas de Símbolos

Lourdes del Carmen González Huesca Alejandra Krystel Coloapa Díaz
Javier Enríquez Mendoza

Integrantes del equipo

Ángeles Martínez Ángela Janín	314201009
García Landa Valeria	314033008
Martínez Monroy Emily	314212391
Rebollar Pérez Ailyn	314322164

Desarrollo de los Ejercicios

Ejercicio 1
Para este ejercicio lo que hicimos fue definir un nuevo lenguaje que se llama L11, donde las lambdas vuelven a ser multiparamétricas.
A continuación el lenguaje L11:

```
(define-language L11
  (extends L10)
  (Expr (e body)
        (- (lambda ([x t]) body))
        (+ (lambda ([x* t*] ...) body* ... body))))
```

Sucesivamente, definimos el proceso de uncurry, lo que hicimos fue cuando tuviéramos una lambda entonces mandar llamar una función auxiliar que definimos en definitions que se llama *descurrifica* la cual recibe una expresión y una lista, donde en la lista llevaremos todos los parámetros anidados pero para que respetará la sintáxis del lenguaje tuvimos que construir los parámetros con cons mientras que en expr iremos pasando el cuerpo de la lambda, llegará un momento en el que el cuerpo de una lambda ya no sea una lambda y es cuando entrará en el caso del else es cuando regresamos el constructor de la lambda con la lista de parámetros y el cuerpo.

Ejercicio 2
Para éste ejercicio seguimos el hint que se nos indicó en la práctica e hicimos una función usando el nanopass-case donde en los casos donde cazábamos let, letrec y letfun agregabamos en nuestra tabla de símbolos las variables con su valor de las asignaciones y luego regresábamos la tabla de símbolos, pero teníamos algunos casos base donde no teníamos que agregar cosas como las variables, las constantes y la lista vacía pero había otros donde sólo hicimos recursión en las expresiones por si caían en alguno de los casos de let, letrec y letfun.
Además usamos la función de las Hash-Tables de racket que se llama hash-union! para que en los casos donde tengamos más de una expresión podamos unir sus tablas de símbolos y se volviera una sola, para ello tuvimos que poner la línea *(require racket/hash)*

Ejercicio 3
Para este ejercicio lo que hicimos fue definir un nuevo lenguaje que se llama L12, donde los constructores let, letrec y lefun ya no hay asignaciones, sino que ahora reciben una expresión.
A continuación el lenguaje L12:

```
(define-language L12
  (extends L11)
  (Expr (e body)
        (- (let ([x t e]) body)
           (letrec ([x t e]) body)
           (letfun ([x t e]) body))
        (+ (let e)
           (letrec e)
           (letfun e))))
```

Luego lo que hicimos fue hacer un proceso auxiliar al cual llamamos *change-constructor* el cual recibe una expresión en el lenguaje L11 y les quitamos las asignaciones a los constructores mencionados, sin embargo para el proceso principal que es assigment, lo que hicimos fue mandar llamar la función auxiliar de change-constructor y al del ejercicio 2 ya que teníamos que devolver la expresión y la tabla de símbolos que genera cada expresión.