Compiladores 2020-1
Facultad de Ciencias UNAM
Práctica 2: Definición y preprocesamiento del lenguaje fuente

Lourdes del Carmen González Huesca Alejandra Krystel Coloapa Díaz

Javier Enríquez Mendoza*

9 de septiembre de 2019

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 éste ejercicio lo que hicimos fue fijarnos en cómo estaba definida la grámatica del lenguaje fuente LF para poder usar algunos predicados de las primitivas en racket, pero para los que no tuvimos que utilizar quote para tomar los símbolos y checar si eran igual a los que queríamos recibir.

Gramática

```
<programa> ::= <expr>

<expr> ::= <const>
         | <list>
         | <var>
         | <string>
         | (<prim> <const> <const>*)
         | (begin <expr> <expr>*)
         | (if <expr> <expr>)
         | (if <expr> <expr> <expr>)
         | (lambda ([<var> <type>]*) <expr>)
         | (let ([<var> <type> <expr>]*)  <expr>)
         | (letrec ([<var> <type> <expr>]*) <expr>)
         | (<expr> <expr>*)

<const> ::=  <boolean>
```

*javierem_94@ciencias.unam.mx

```
          | <integer>
          | <char>

<boolean> ::= #t | #f

<integer> ::= <digit> | <digit><integer>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<var> ::= <car> | <car><var> | <car><digit> | <car><digit><var>

<car> ::= a | b | c | ... | z

<list> ::= empty | (cons <const> <list>)

<string> ::= "" | " <char> <string> "

<char> ::=  a | b | c | ... | z | ... | @ | # | $ | % | & | ...

<prim> ::= + | - | * | / | and | or | length | car | cdr

<type> ::= Bool | Int | Char | List | String
```

Código

```
(define-language LF
  (terminals
   (variable (x))
   (primitive (pr))
   (constant (c))
   (list (l))
   (string (s))
   (type (t)))
  (Expr (e body)
    x
    pr
    c
    l
    s
    t
    (pr c* ... c)
    (begin e* ... e)
    (if e0 e1)
    (if e0 e1 e2)
    (lambda ([x* t*] ...) body* ... body)
    (let ([x* t* e*] ...) body* ... body)
    (letrec ([x* t* e*] ...) body* ... body)
    (e0 e1 ...)))
```

Ejercicio 2

Para éste ejercicio fue ayudarnos de lo que sabemos de racket, por lo que decidimos hacer las funciones auxiliares en la parte de **definitions** del proceso.

En ésta sección definimos una hash table que llamamos **sustituciones** y ahí almacenamos las variables que utilizamos como la llave y su valor era la variable por la que la ibamos a renombrar. Una función auxiliar para almacenar las variables ligadas la llamamos **guarda-variablesDL** donde identificamos los constructores de nuestro lenguaje que tenían variables deligadas. Otra de nuestras funciones auxiliares se llama **sustituye** en la cual ya teniendo las sustituciones o renombramientos correspondientes se encarga de ir cambiando el renombre.

Ejercicio 3

Para resolver este ejercicio, extendimos nuestro lenguaje fuente LF al nuevo lenguaje LFV en donde agregamos el terminal void para poder recibir (if e0 e1) y que siempre regrese un if con cuerpo en else, es decir (if e0 e2 (void))

```
(define-language LFV
  (terminals
   (variable (x))
   (primitive (pr))
   (constant (c))
   (list (l))
   (string (s))
   (type (t))
   (void (v)))
  (Expr (e body)
    x
    pr
    c
    l
    s
    t
    v
    (pr c* ... c)
    (begin e* ... e)
    (if e0 e1 e2)
    (lambda ([x* t*] ...) body* ... body)
    (let ([x* t* e*] ...) body* ... body)
    (letrec ([x* t* e*] ...) body* ... body)
    (e0 e1 ...)))
```

Ejercicio 4

Para resolver este ejercicio, extendimos el lenguaje LF al nuevo lenguaje LFVS. Para el nuevo lenguaje que elimina las cadenas como elementos terminales, cada que es detectada un s, la convertimos a lista con una función nativa de racket.

```
(define-language LFVS
  (terminals
   (variable (x))
   (primitive (pr))
   (constant (c))
   (list (l))
   (type (t))
   (void (v)))
  (Expr (e body)
    x
    pr
    c
    l
    t
    v
    (pr c* ... c)
```

```
    (begin e* ... e)
    (if e0 e1 e2)
    (lambda ([x* t*] ...) body* ... body)
    (let ([x* t* e*] ...) body* ... body)
    (letrec ([x* t* e*] ...) body* ... body)
    (e0 e1 ...)))
```