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

Luis Martínez Hernández 314240028 Emmanuel Cruz Hernández 314272588
Daniela Susana Vega Monroy 314582915

7 de septiembre de 2019

1. Soluciones a los ejercicios:

1. Ejercicio 1: El ejercicio se solucionó de la siguiente manera, para cada terminal,definimos una función que revisara el tipo de ese terminal,esto con ayuda de las funciones ya implementadas por racket.
2. Ejercicio 2: Sin solucionar
3. Ejercicio 3: El ejercicio se solucionó de la siguiente manera: Para empezar se definió un nuevo lenguaje que quitara la expresión if de dos parámetros y que además agregara la expresión void,despues se definió un pass que cuando se tenía un if de dos parámetros que perteneciera a el lenguaje fuente lo cambiamos por un if que tuviera tres parámetros dónde el último parámetro sería un void.
4. Ejercicio 4: El ejercicio se solucionó de la siguiente manera: Para empezar se definio un nuevo lenguaje que quitara las cadenas,y después se definio un pass que cuando se tenea una cadena en el lenguaje fuente la cambiara por esa misma cadena pero en forma de lista, cabe mencionar que hicimos uso de la función de racket string->list para convertir la cadena en una lista.

2. Comentarios

Tuvimos problemas, como la mayoría de los equipos, en el ejercicio 2 pues la recursión para que se fuera asignando x0,x1,x2, etc no nos quedó claro en racket, solo logramos que se asignará x0 a las susticiones y solo a partir del segundo paréntesis. Los demás ejercicios, si bien no estaban tan difíciles si tomó más tiempo de lo esperado.

3. Gramática

```
Gram. L1
<programa> ::= <expr>

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

```
         | (<expr> <expr>*)

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

```
Gram. L2
<programa> ::= <expr>

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

<const> ::=  <boolean>
         | <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>)


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

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

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

4. Lenguajes

```
(define-language L1
  (terminals
   (variable (x))
   (primitive (pr))
   (constant (c))
   (list (l))
   (string (s))
   (type (t)))
  (Expr (e body)
    x
    pr
    c
    l
    t
    (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 ...)))
```

```
(define-language L2
  (terminals
   (variable (x))
   (primitive (pr))
   (constant (c))
   (list (l))
   (type (t)))
  (Expr (e body)
    x
    pr
    c
    l
    t
    (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 ...)))
```