Compiladores 2020-1 Facultad de Ciencias UNAM Práctica 6

Luis Eduardo Martinez Hernandez 314240028 Emmanuel Cruz Hernandez 314272588 Daniela Susana Vega Monroy 314582915

10 de noviembre de 2019

1. Solución a los ejercicios

Para realizar los ejercicios nos auxiliamos con las siguientes funciones:

- (uncurryLambda params cuerpo)
- (tablaSimbolos (make-hash))
- (tablita (make-hash))

(uncurryLambda params cuerpo) se utilizó para el primer ejercicio **uncurry**, (tablaSimbolos (make-hash)) para el segundo ejercicio **symbol-table-var** y (tablita (make-hash)) para el tercer ejercicio **assigment**.

2. Comentarios

A diferencia de otras prácticas, en esta solo se necesitó una función auxiliar por ejercicio, no fue la práctica más fácil sin embargo entendimos bien los ejercios el martes que se explicó la práctica y se terminó a bien tiempo.

3. Gramática

```
<expr> ::= <const>
       | <list>
       | <var>
       | <string>
       | (<prim> <const> <const>*)
       | (begin <expr> <expr>*)
       | (if <expr> <expr>)
       | (if <expr> <expr> <expr>)
       | (lambda ([<var> <type>]*) <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> | <car><digit>
<car> ::= a | b | c | ... | z
tist> ::= empty | (cons <const> tist>)
<string> ::= "" | " <char> <string> "
<char> ::= a | b | c | ... | z | ... | @ | # | $ | % | & | ...
<pri><pri>::= + | - | * | / | and | or | length | car | cdr
<type> ::= Bool | Int | Char | List | String
```

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

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

```
| (if <expr> <expr> <expr>)
        | (lambda ([<var> <type>]*) <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><digit> | <car><digit><var>
<car> ::= a | b | c | ... | z
<list> ::= empty | (cons <const> <list>)
<char> ::= a | b | c | ... | z | ... | @ | # | $ | % | & | ...
<pri>> ::= + | - | * | / | npr | length | and | or not | car | cdr
<type> ::= Bool | Int | Char | List
```

```
<expr> ::= <const>
       | <list>
       | <var>
       | (<prim> <const> <const>*)
       | (begin <expr> *)
       | (+ <expr> <expr>)
       | (- <expr> <expr>)
       | ( * <expr> <expr>)
       | (/ <expr> <expr>)
       | (primapp (<expr>)
       | (primapp (<expr>) <expr> <expr>)
       | (if <expr> <expr> <expr>)
       | (lambda ([<var> <type>]*) <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> | <car> <digit> | <car> <digit> < <car> <ii:= a | b | c | ... | z
<li><::= empty | (cons <const> <)
</pre>
<char> ::= a | b | c | ... | z | ... | @ | # | $ | % | & | ...
<prim> ::= length | car | cdr
<type> ::= Bool | Int | Char | List
```

```
<expr> ::= <const>
        | <list>
        | <var>
        | (<prim> <const> <const>*)
        | (begin <expr> *)
        | (+ <const> <const>)
        | (- <const> <const>)
        | ( * <const> <const>)
        | (/ <const> <const>)
        | (primapp (<expr>)
        | (primapp (<expr>) <expr> <expr>)
        | (if <expr> <expr> <expr>)
        | (lambda ([<var> <type>]*) <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>)
```

```
<expr> ::= <const>
       | <list>
        | <var>
        | (<prim> <const> <const>*)
        | (begin <expr> *)
        | (+ <const> <const>)
        | (- <const> <const>)
        | ( * <const> <const>)
        | (/ <const> <const>)
        | (primapp (<expr>)
        | (primapp (<expr>) <expr> <expr>)
        | (if <expr> <expr> <expr>)
        | (lambda ([<var> <type>]*) <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> ::= (kuote a) | (kuote b) | (kuote b) | ... | (kuote z) | ... | (koute @) |
   (kuote #) | (kuote $) | (kuote %) | (kuote &) | ...
<prim> ::= length | car | cdr
<type> ::= Bool | Int | Char | List
```

```
<expr> ::= <const>
        | <list>
        | <var>
        | (<prim> <const> <const>*)
        | (begin <expr> *)
        | (+ <const> <const>)
        | (- <const> <const>)
        | ( * <const> <const>)
        | (/ <const> <const>)
        | (primapp (<expr>)
        | (primapp (<expr>) <expr> <expr>)
        | (if <expr> <expr> <expr>)
        | (lambda ([<var> <type>]*) <expr>)
        | (letrec ([<var> <type> <expr>]*) <expr>)
        | (leftfun ([<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> ::= (kuote a) | (kuote b) | (kuote b) | ... | (kuote z) | ... | (koute @) |
    (kuote #) | (kuote $) | (kuote %) | (kuote &) | ...
<prim> ::= length | car | cdr
<type> ::= Bool | Int | Char | List
L9
<expr> ::= <const>
        | <list>
        | <var>
        | (<prim> <const> <const>*)
        | (begin <expr> <expr>*)
        | (+ <const> <const>)
        | (- <const> <const>)
        | ( * <const> <const>)
```

```
| (/ <const> <const>)
        | (primapp (<expr>)
        | (primapp (<expr>) <expr> <expr>)
        | (if <expr> <expr> <expr>)
        | (lambda ([<var> <type>]) <expr>)
        | (letrec ([<var> <type> <expr>]*) <expr>)
        | (leftfun ([<var> <type> <expr>]*) <expr>)
        | (<expr> <expr>*)
<const> ::= <boolean>
       | <integer>
        | <char>
<boolean> ::= (const bool #t) | (const bool #f)
<integer> ::= <digit> | <digit><integer>
<digit> ::= (const int 0) | (const int 1) | (const int 2) | (const int 3) | (const
   int 4) | (const int 5) | (const int 6) | (const int 7) | (const int 8) | (const
   int 9) | ... |
<var> ::= <car> | <car><var> | <car><digit> | <car><digit><var>
<car> ::= a | b | c | ... | z
<list> ::= empty | (cons <const> <list>)
<char> ::= (const Char a) | (const Char b) | (const Char b) | ... | (const Char z)
   | ... | (const Char @) | (const Char #) | (const Char $) | (const Char %) | (
   const Char &) | ...
<prim> ::= length | car | cdr
<type> ::= Bool | Int | Char | List
L10
<expr> ::= <const>
        | <list>
        | <var>
        | (<prim> <const> <const>*)
        | (begin <expr> *)
        | (+ <const> <const>)
        | (- <const> <const>)
        | ( * <const> <const>)
        | (/ <const> <const>)
        | (primapp (<expr>)
        | (primapp (<expr>) <expr> <expr>)
        | (if <expr> <expr> <expr>)
        | (lambda ([<var> <type>]) <expr>)
```

```
| (letrec ([<var> <type> <expr>]*) <expr>)
        | (leftfun ([<var> <type> <expr>]*) <expr>)
        | (<expr> <expr>*)
<const> ::= <boolean>
        | <integer>
        | <char>
<boolean> ::= (const bool #t) | (const bool #f)
<integer> ::= <digit> | <digit><integer>
<digit> ::= (const int 0) | (const int 1) | (const int 2) | (const int 3) | (const
   int 4) | (const int 5) | (const int 6) | (const int 7) | (const int 8) | (const
   int 9) | ... |
<var> ::= <car> | <car><digit> | <car><digit><var>
<car> ::= a | b | c | ... | z
<list> ::= empty | (cons <const> <list>)
<char> ::= (const Char a) | (const Char b) | (const Char b) | ... | (const Char z)
    | ... | (const Char @) | (const Char #) | (const Char $) | (const Char %) | (
   const Char &) | ...
<prim> ::= length | car | cdr
<type> ::= Bool | Int | Char | List
```

```
<expr> ::= <const>
       | <list>
       | <var>
       | (<prim> <const> <const>*)
       | (begin <expr> <expr>*)
       | (+ <const> <const>)
       | (- <const> <const>)
       | ( * <const> <const>)
       | (/ <const> <const>)
       | (primapp (<expr>)
       | (primapp (<expr>) <expr> <expr>)
       | (if <expr> <expr> <expr>)
       | (lambda ([<var> <type>]*) <expr>)
       | (letrec ([<var> <type> <expr>]*) <expr>)
       | (leftfun ([<var> <type> <expr>]*) <expr>)
       | (<expr> <expr>*)
<const> ::= <boolean>
  | <integer>
```

```
<expr> ::= <const>
        | <list>
        | <var>
        | (<prim> <const> <const>*)
        | (begin <expr> *)
        | (+ <const> <const>)
        | (- <const> <const>)
        | ( * <const> <const>)
        | (/ <const> <const>)
        | (primapp (<expr>)
        | (primapp (<expr>) <expr> <expr>)
        | (if <expr> <expr> <expr>)
        | (lambda ([<var> <type>]*) <expr>)
        | (let <expr>)
        | (letrec <expr>)
        | (leftfun <expr>)
        | (<expr> <expr>*)
<const> ::= <boolean>
        | <integer>
        | <char>
<boolean> ::= (const bool #t) | (const bool #f)
<integer> ::= <digit> | <digit><integer>
```

```
<digit> ::= (const int 0) | (const int 1) | (const int 2) | (const int 3) | (const
   int 4) | (const int 5) | (const int 6) | (const int 7) | (const int 8) | (const
   int 9) | ... |
<var> ::= <car> | <car><var> | <car><digit> | <car><digit><var>
<car> ::= a | b | c | ... | z
<list> ::= empty | (cons <const> <list>)
<char> ::= (const Char a) | (const Char b) | (const Char b) | ... | (const Char z)
    | ... | (const Char @) | (const Char #) | (const Char $) | (const Char %) | (
   const Char &) | ...
<prim> ::= length | car | cdr
<type> ::= Bool | Int | Char | List
(define-language LF
  (terminals
   (variable (x))
   (primitive (pr))
   (constant (c))
   (list (1))
   (string (s))
   (type (t)))
  (Expr (e body)
    х
    pr
    C.
    1
    (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 ...)))
(define-language L1
  (terminals
   (variable (x))
   (primitive (pr))
   (constant (c))
   (list (1))
   (string (s))
   (type (t)))
  (Expr (e body)
    Х
    pr
```

1

3

4

5 6

7

8

9 10

11

12

13

14 15

16

17

18

19

20 21

22

1

2

4

5

6

7 8

9

10

11

12

```
1
13
14
        (pr c* ... c)
15
16
        (begin e* ... e)
17
        (void (void))
        (if e0 e1 e2)
18
        (lambda ([x*t*] ...) body* ... body)
19
        (let ([x* t* e*] ...) body* ... body)
20
21
        (letrec ([x* t* e*] ...) body* ... body)
        (e0 e1 ...)))
22
   (define-language L2
1
     (terminals
2
3
      (variable (x))
       (primitive (pr))
4
       (constant (c))
5
6
      (list (1))
7
      (type (t)))
8
      (Expr (e body)
9
       Х
10
       pr
        С
11
        1
12
13
        t
14
        (pr c* ... c)
        (begin e* ... e)
15
        (void (void))
16
        (if e0 e1 e2)
17
        (lambda ([x*t*] ...) body* ... body)
18
        (let ([x* t* e*] ...) body* ... body)
19
20
        (letrec ([x* t* e*] ...) body* ... body)
        (e0 e1 ...)))
21
   (define-language L4
1
2
     (terminals
3
      (variable (x))
      (primitive (npr))
4
       (constant (c))
5
      (list (1))
6
      (string (s))
7
8
      (type (t)))
9
      (Expr (e body)
10
       Х
11
        Npr
12
        С
       1
13
14
        (pr c* ... c)
15
        (begin e* ... e)
16
17
        (not e)
        (and e0 e1)
18
19
        (Nor e0 e1)
        (void (void))
20
        (if e0 e1 e2)
21
```

```
(lambda ([x* t*] ...) body* ... body)
22
        (let ([x* t* e*] ...) body* ... body)
23
        (letrec ([x* t* e*] ...) body* ... body)
24
        (e0 e1 ...)))
25
1
   (define-language L5
     (terminals
2
       (variable (x))
3
       (constant (c))
4
       (list (1))
5
       (string (s))
6
7
      (type (t)))
8
     (Expr (e body)
9
       Х
10
       npr
        С
11
12
       1
13
14
        (npr c* ... c)
        (begin e* ... e)
15
16
        (not e)
        (and e0 e1)
17
        (or e0 e1)
18
        (+ e0 e1)
19
20
        (- e0 e1)
        (\* e0 e1)
21
        (/ e0 e1)
22
        (void (void))
23
^{24}
        (primapp (e0) e1)
25
        (primapp (e0) e1 e2))
26
        (if e0 e1 e2)
        (lambda ([x* t*] ...) body* ... body)
27
        (let ([x* t* e*] ...) body* ... body)
28
        (letrec ([x* t* e*] ...) body* ... body)
29
30
        (e0 e1 ...)))
   (define-language L5
1
2
      (terminals
       (variable (x))
3
       (constant (c))
4
       (list (1))
5
6
       (string (s))
7
       (type (t)))
8
      (Expr (e body)
9
        х
10
       npr
11
        С
12
       1
13
14
        (npr c* ... c)
        (begin e* ... e)
15
        (not e)
16
        (and e0 e1)
17
        (or e0 e1)
18
```

```
(+ e0 e1)
19
        (- e0 e1)
20
        (\* e0 e1)
21
        (/ e0 e1)
22
23
        (void (void))
        (primapp (e0) e1)
24
        (primapp (e0) e1 e2))
25
        (if e0 e1 e2)
26
27
        (lambda ([x*t*] ...) body* ... body)
        (let ([x* t* e*] ...) body* ... body)
28
29
        (letrec ([x* t* e*] ...) body* ... body)
        (e0 e1 ...)))
30
   (define-language L6
1
     (terminals
2
      (variable (x))
3
4
       (constant (c))
       (list (1))
5
6
       (string (s))
7
       (type (t)))
8
      (Expr (e body)
9
10
       npr
11
        С
12
       1
13
        (npr c* ... c)
14
        (begin e* ... e)
15
        (not e)
16
        (and e0 e1)
17
18
        (or e0 e1)
        (+ e0 e1)
19
        (-e0e1)
20
        (\* e0 e1)
21
        (/ e0 e1)
22
23
        (void (void))
        (primapp (e0) e1)
24
25
        (primapp (e0) e1 e2))
        (if e0 e1 e2)
26
27
        (lambda ([x*t*] ...) body* ... body)
28
        (let ([x* t* e*] ...) body* ... body)
        (letrec ([x* t* e*] ...) body* ... body)
29
        (e0 e1 ...)))
30
1
   (define-language L7
2
      (terminals
       (variable (x))
3
       (constant (c))
4
       (list (1))
5
6
       (string (s))
7
       (type (t)))
8
      (Expr (e body)
9
        Х
10
       npr
```

```
11
        1
12
13
        (npr c* ... c)
14
15
        (begin e* ... e)
        (\* e0 e1)
16
        (/ e0 e1)
17
        (primapp (e0) e1)
18
19
        (primapp (e0) e1 e2))
        (if e0 e1 e2)
20
21
        (lambda ([x*t*] ...) body* ... body)
        (let ([x* t* e*]) body* ... body)
22
        (letrec ([x* t* e*]) body* ... body)
23
        (letrec ([x t e]) body)
24
25
        (let ([x t e]) body)
        (e0 e1 ...)))
26
   (define-language L8
1
2
      (terminals
3
      (variable (x))
       (constant (c))
4
       (list (1))
5
6
      (string (s))
7
       (type (t)))
8
      (Expr (e body)
9
        Х
10
       npr
11
        С
       1
12
13
        (npr c* ... c)
14
15
        (begin e* ... e)
        (\* e0 e1)
16
        (/ e0 e1)
17
18
        (primapp (e0) e1)
19
        (primapp (e0) e1 e2))
        (if e0 e1 e2)
20
21
        (lambda ([x* t*] ...) body* ... body)
        (let ([x* t* e*]) body* ... body)
22
23
        (letrec ([x* t* e*]) body* ... body)
24
        (letrec ([x t e]) body)
        (let ([x t e]) body)
25
        (letfun ([x t e]) body)
26
        (e0 e1 ...)))
27
   (define-language L9
1
      (terminals
2
3
      (variable (x))
       (constant (c))
4
5
       (list (1))
6
      (string (s))
7
       (type (t)))
      (Expr (e body)
8
```

9

```
10
        npr
11
        С
12
        1
13
        t
14
        (npr c* ... c)
        (begin e* ... e)
15
16
        (\* e0 e1)
        (/ e0 e1)
17
18
        (primapp (e0) e1)
        (primapp (e0) e1 e2))
19
20
        (if e0 e1 e2)
        (lambda ([x t]) body ... body)
21
        (let ([x* t* e*]) body* ... body)
22
        (letrec ([x* t* e*]) body* ... body)
23
24
        (letrec ([x t e]) body)
25
        (let ([x t e]) body)
26
        (letfun ([x t e]) body)
        (e0 e1 ...)))
27
   (define-language L10
1
2
      (terminals
       (variable (x))
3
       (constant (c))
4
       (list (1))
5
6
       (string (s))
7
       (type (t)))
8
      (Expr (e body)
9
       Х
10
       npr
11
        С
12
       1
13
        (npr c* ... c)
14
15
        (begin e* ... e)
16
        (\* e0 e1)
17
        (/ e0 e1)
        (primapp (e0) e1)
18
19
        (primapp (e0) e1 e2))
        (if e0 e1 e2)
20
21
        (lambda ([x t]) body ... body)
        (let ([x* t* e*]) body* ... body)
22
        (letrec ([x* t* e*]) body* ... body)
23
        (letrec ([x t e]) body)
24
25
        (let ([x t e]) body)
26
        (letfun ([x t e]) body)
27
        (e0 e1 ...)))
   (define-language L11
1
      (terminals
2
3
       (variable (x))
```

```
(Expr (e body)
8
9
10
       npr
       С
11
12
       1
13
14
        (npr c* ... c)
        (begin e* ... e)
15
16
        (\* e0 e1)
        (/ e0 e1)
17
18
        (primapp (e0) e1)
        (primapp (e0) e1 e2))
19
        (if e0 e1 e2)
20
        (lambda ([x*t*] ...) body* ... body)
21
22
        (let ([x* t* e*]) body* ... body)
23
        (letrec ([x* t* e*]) body* ... body)
24
        (letrec ([x t e]) body)
        (let ([x t e]) body)
25
26
        (letfun ([x t e]) body)
27
        (e0 e1 ...)))
```

```
(define-language L12
1
     (terminals
2
      (variable (x))
3
4
      (constant (c))
      (list (1))
5
6
      (string (s))
      (type (t)))
7
     (Expr (e body)
8
9
       Х
10
       npr
11
        С
       1
12
13
14
        (npr c* ... c)
15
        (begin e* ... e)
        (\* e0 e1)
16
        (/ e0 e1)
17
        (primapp (e0) e1)
18
19
        (primapp (e0) e1 e2))
20
        (if e0 e1 e2)
        (lambda ([x*t*] ...) body* ... body)
21
        (let ([x* t* e*]) body* ... body)
22
        (letrec ([x* t* e*]) body* ... body)
23
        (letrec body)
24
25
        (let body)
        (letfun body)
26
        (e0 e1 ...)))
27
```