Compiladores 2020-1 Facultad de Ciencias UNAM Práctica 4

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

7 de octubre de 2019

1. Solución a los ejercicios

Para la práctica usamos diferentes funciones auxiliares, para el primer ejercicio se usó currificaLet y get-list. Para el tercer ejercicio definimos la función concatena y para el último ejercicio usamos la función arity? y lista? .

2. Comentarios

Si bien no era una práctica tan sencilla, con la ayudantía pasada see aclararon muchos conceptos y muchos problemas que teníamos con los ejercicios, gracias a la explicación más a fondo de los ejercicios se pudo resolver un poco más rápido que las otras. También al haber ya usado nanopass en 3 prácticas, se nos hizo más fácil su uso, pues al principio no entendíamos bien su funcionamiento.

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>
```

```
<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> *)
        | (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> ::= a | b | c | ... | z | ... | @ | # | $ | % | & | ...
<prim> ::= + | - | * | / | and | or | length | car | cdr
<type> ::= Bool | Int | Char | List
```

```
<expr> ::= <const>
       | <list>
       | <var>
       | (<prim> <const> <const>*)
       | (begin <expr> <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> <var> 

<car> ::= a | b | c | ... | z
<char> ::= a | b | c | ... | z | ... | @ | # | $ | % | & | ...
<pri><pri><pri><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>)
        | (<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> ::= (kuote a) | (kuote b) | (kuote b) | ... | (kuote z) | ... | (koute @) |
   (kuote #) | (kuote $) | (kuote %) | (kuote &) | ...
<prim> ::= length | car | cdr
```

```
L7
<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
L8
<expr> ::= <const>
        | <list>
        | <var>
        | (<prim> <const> <const>*)
```

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

```
| (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
1 (define-language LF
     (terminals
      (variable (x))
3
      (primitive (pr))
4
      (constant (c))
5
6
      (list (1))
      (string (s))
7
8
      (type (t)))
     (Expr (e body)
9
10
      х
11
      pr
12
       C.
13
       1
14
15
       (pr c* ... c)
      (begin e* ... e)
16
```

```
(if e0 e1)
17
        (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 L1
1
     (terminals
2
3
      (variable (x))
      (primitive (pr))
4
5
      (constant (c))
6
      (list (1))
7
      (string (s))
      (type (t)))
8
     (Expr (e body)
9
10
       X
11
       pr
12
       С
       1
13
14
       t
       (pr c* ... c)
15
        (begin e* ... e)
16
17
        (void (void))
18
        (if e0 e1 e2)
        (lambda ([x* t*] ...) body* ... body)
19
        (let ([x* t* e*] ...) body* ... body)
20
        (letrec ([x* t* e*] ...) body* ... body)
21
22
        (e0 e1 ...)))
   (define-language L2
1
2
     (terminals
      (variable (x))
3
      (primitive (pr))
4
      (constant (c))
5
6
      (list (1))
7
      (type (t)))
8
     (Expr (e body)
9
       х
10
       pr
11
       С
12
13
14
       (pr c* ... c)
       (begin e* ... e)
15
       (void (void))
16
17
        (if e0 e1 e2)
        (lambda ([x* t*] ...) body* ... body)
18
19
        (let ([x* t* e*] ...) body* ... body)
        (letrec ([x* t* e*] ...) body* ... body)
20
        (e0 e1 ...)))
21
  (define-language L4
2 (terminals
```

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

```
(define-language L5
1
2
      (terminals
3
      (variable (x))
       (constant (c))
4
       (list (1))
5
       (string (s))
6
7
       (type (t)))
8
      (Expr (e body)
9
        X
10
        npr
11
        С
        1
12
13
        t
14
        (npr c* ... c)
        (begin e* ... e)
15
        (not e)
16
        (and e0 e1)
17
        (or e0 e1)
18
        (+ e0 e1)
19
20
        (- e0 e1)
        (\* e0 e1)
21
22
        (/ e0 e1)
        (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 L6
1
      (terminals
2
       (variable (x))
3
4
       (constant (c))
       (list (1))
5
6
      (string (s))
       (type (t)))
7
8
      (Expr (e body)
9
        Х
10
        npr
11
        С
        1
12
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)
28
        (let ([x* t* e*] ...) body* ... body)
        (letrec ([x* t* e*] ...) body* ... body)
29
30
        (e0 e1 ...)))
   (define-language L7
      (terminals
2
       (variable (x))
3
       (constant (c))
4
       (list (1))
5
6
      (string (s))
       (type (t)))
7
      (Expr (e body)
8
9
10
       npr
11
        С
       1
12
13
        (npr c* ... c)
14
        (begin e* ... e)
15
        (not e)
16
17
        (and e0 e1)
        (or e0 e1)
18
        (+ e0 e1)
19
        (-e0e1)
20
        (\* e0 e1)
21
        (/ e0 e1)
22
23
        (void (void))
        (primapp (e0) e1)
24
25
        (primapp (e0) e1 e2))
26
        (if e0 e1 e2)
        (lambda ([x*t*] ...) body* ... body)
27
28
        (let ([x* t* e*] ...) body* ... body)
        (letrec ([x* t* e*] ...) body* ... body)
29
30
        (letrec ([x t e]) body)
        (let ([x t e]) body)
31
32
        (e0 e1 ...)))
1
   (define-language L8
2
     (terminals
      (variable (x))
3
4
       (constant (c))
5
       (list (1))
      (string (s))
6
7
      (type (t)))
      (Expr (e body)
8
9
       Х
10
        npr
11
        С
       1
12
13
```

```
(npr c* ... c)
14
       (begin e* ... e)
15
        (not e)
16
        (and e0 e1)
17
18
        (or e0 e1)
19
        (+ e0 e1)
       (- 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
        (lambda ([x*t*] ...) body* ... body)
27
28
        (let ([x* t* e*] ...) body* ... body)
       (letrec ([x* t* e*] ...) body* ... body)
29
30
       (letrec ([x t e]) body)
        (let ([x t e]) body)
31
32
       (letfun ([x t e]) body)
        (e0 e1 ...)))
33
```