

Compiladores 2020-1

Facultad de Ciencias UNAM

Práctica 7

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

25 de noviembre de 2019

1. Solución a los ejercicios

Para realizar los ejercicios nos auxiliamos con las siguientes funciones:

- (createList expresiones)
- (createTypeList expresiones)
- (contexto (list (cons 'x 'Int)))
- (cType x)
- (stringType x)
- (traduceParametros parametros tipos)

(createList) y (createTypeList) fueron usados para el primer ejercicio **list-to-array**. Y (contexto (list (cons 'x 'Int))), (cType x), (stringType x), (traduceParametros parametros tipos) para **c**.

2. 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>
        | <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

```

L1

```

<programa> ::= <expr>

<expr> ::= <const>
          | <list>
          | <var>
          | <string>
          | <void>
          | (<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

```

L2

```

<programa> ::= <expr>

<expr> ::= <const>
          | <list>
          | <var>
          | (<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

```

L4

```

<programa> ::= <expr>

<expr> ::= <const>
          | <list>
          | <var>
          | (<prim> <const> <const>*)
          | (begin <expr> <expr>*)
          | (and <expr> <expr>)
          | (or <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> ::= + | - | * | / | npr | length | and | or | not | car | cdr

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

```

L5

```

<programa> ::= <expr>

<expr> ::= <const>
          | <list>
          | <var>
          | (<prim> <const> <const>*)
          | (begin <expr> <expr>*)
          | (+ <expr> <expr>)
          | (- <expr> <expr>)
          | ( * <expr> <expr>)
          | (/ <expr> <expr>)
          | (primapp (<expr>) <expr>)
          | (primapp (<expr>) <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> ::= length | car | cdr

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

```

L6

```

<programa> ::= <expr>

<expr> ::= <const>
          | <list>
          | <var>
          | (<prim> <const> <const>*)
          | (begin <expr> <expr>*)
          | (+ <const> <const>)
          | (- <const> <const>)
          | ( * <const> <const>)
          | (/ <const> <const>)
          | (primapp (<expr>) <expr>)
          | (primapp (<expr>) <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> ::= (quote a) | (quote b) | (quote b) | ... | (quote z) | ... | (quote @) |
          (quote #) | (quote $) | (quote %) | (quote &) | ...

<prim> ::= length | car | cdr

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

```

L7

```

<programa> ::= <expr>

<expr> ::= <const>
          | <list>
          | <var>
          | (<prim> <const> <const>*)
          | (begin <expr> <expr>*)
          | (+ <const> <const>)
          | (- <const> <const>)
          | ( * <const> <const>)
          | (/ <const> <const>)
          | (primapp (<expr>) <expr>)
          | (primapp (<expr>) <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> ::= (quote a) | (quote b) | (quote b) | ... | (quote z) | ... | (quote @) |
          (quote #) | (quote $) | (quote %) | (quote &) | ...

<prim> ::= length | car | cdr

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

```

L8

```

<programa> ::= <expr>

```

```

<expr> ::= <const>
        | <list>
        | <var>
        | (<prim> <const> <const>*)
        | (begin <expr> <expr>*)
        | (+ <const> <const>)
        | (- <const> <const>)
        | ( * <const> <const>)
        | (/ <const> <const>)
        | (primapp (<expr>) <expr>)
        | (primapp (<expr>) <expr> <expr>)
        | (if <expr> <expr> <expr>)
        | (lambda ([<var> <type>]*) <expr>)
        | (let ([<var> <type> <expr>]*) <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) | ... | (kuote @) |
          (kuote #) | (kuote $) | (kuote %) | (kuote &) | ...

<prim> ::= length | car | cdr

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

```

L9

```

<programa> ::= <expr>

<expr> ::= <const>
        | <list>
        | <var>
        | (<prim> <const> <const>*)
        | (begin <expr> <expr>*)
        | (+ <const> <const>)
        | (- <const> <const>)
        | ( * <const> <const>)
        | (/ <const> <const>)

```

```

| (primapp (<expr>) <expr>)
| (primapp (<expr>) <expr> <expr>)
| (if <expr> <expr> <expr>)
| (lambda ([<var> <type>]) <expr>)
| (let ([<var> <type> <expr>]*) <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

```

<programa> ::= <expr>

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

```

L11

```

<programa> ::= <expr>

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

```

L12

```

<programa> ::= <expr>

<expr> ::= <const>
  | <list>
  | <var>
  | (<prim> <const> <const>*)
  | (begin <expr> <expr>*)
  | (+ <const> <const>)
  | (- <const> <const>)
  | ( * <const> <const>)
  | (/ <const> <const>)
  | (primapp (<expr>) <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

```

L13

```

<programa> ::= <expr>

<expr> ::= <const>
          | <var>
          | (<prim> <const> <const>*)
          | (begin <expr> <expr>*)
          | (+ <const> <const>)
          | (- <const> <const>)
          | ( * <const> <const>)
          | (/ <const> <const>)
          | (primapp (<expr>) <expr>)
          | (primapp (<expr>) <expr> <expr>)
          | (if <expr> <expr> <expr>)
          | (lambda ([<var> <type>]*) <expr>)
          | (let <expr>)
          | (letrec <expr>)
          | (leftfun <expr>)
          | (<expr> <expr>*)
          | (c (array len t [e* ...]))

<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

```

```

<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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

1 (define-language L5
2   (terminals
3     (variable (x))
4     (constant (c))
5     (list (l))

```

```

6  (string (s))
7  (type (t)))
8  (Expr (e body)
9    x
10   npr
11   c
12   l
13   t
14   (npr c* ... c)
15   (begin e* ... e)
16   (not e)
17   (and e0 e1)
18   (or e0 e1)
19   (+ e0 e1)
20   (- e0 e1)
21   (\* e0 e1)
22   (/ e0 e1)
23   (void (void))
24   (primapp (e0) e1)
25   (primapp (e0) e1 e2))
26   (if e0 e1 e2)
27   (lambda ([x* t*] ...) body* ... body)
28   (let ([x* t* e*] ...) body* ... body)
29   (letrec ([x* t* e*] ...) body* ... body)
30   (e0 e1 ...)))

```

```

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

```

```

29 (letrec ([x* t* e*] ...) body* ... body)
30 (e0 e1 ...)))

```

```

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

```

```

1 (define-language L7
2   (terminals
3     (variable (x))
4     (constant (c))
5     (list (l))
6     (string (s))
7     (type (t)))
8   (Expr (e body)
9     x
10    npr
11    c
12    l
13    t
14    (npr c* ... c)
15    (begin e* ... e)
16    (\* e0 e1)
17    (/ e0 e1)
18    (primapp (e0) e1)
19    (primapp (e0) e1 e2))
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 (letrec ([x t e]) body)
25 (let ([x t e]) body)
26 (e0 e1 ...)))

```

```

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

```

```

1 (define-language L9
2   (terminals
3     (variable (x))
4     (constant (c))
5     (list (l))
6     (string (s))
7     (type (t)))
8   (Expr (e body)
9     x
10    npr
11    c
12    l
13    t
14    (npr c* ... c)
15    (begin e* ... e)
16    (\* e0 e1)
17    (/ e0 e1)
18    (primapp (e0) e1)
19    (primapp (e0) e1 e2))

```



```

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 (letrec ([x t e]) body)
25 (let ([x t e]) body)
26 (letfun ([x t e]) body)
27 (e0 e1 ...)))

```

```

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

```

```

1 (define-language L11
2   (terminals
3     (variable (x))
4     (constant (c))
5     (list (l))
6     (string (s))
7     (type (t)))
8   (Expr (e body)
9     x
10    npr
11    c
12    l
13    t
14    (npr c* ... c)
15    (begin e* ... e)
16    (\* e0 e1)
17    (/ e0 e1)

```

```

18 (primapp (e0) e1)
19 (primapp (e0) e1 e2))
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 (letrec ([x t e]) body)
25 (let ([x t e]) body)
26 (letfun ([x t e]) body)
27 (e0 e1 ...)))

```

```

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

```

```

1 (define-language L13
2   (terminals
3     (variable (x))
4     (constant (c))
5     (string (s))
6     (type (t)))
7   (Expr (e body)
8     x
9     npr
10    c
11    l
12    t
13    (npr c* ... c)
14    (begin e* ... e)
15    (\* e0 e1)

```

```
16  (/ e0 e1)
17  (primapp (e0) e1)
18  (primapp (e0) e1 e2))
19  (if e0 e1 e2)
20  (lambda ([x* t*] ...) body* ... body)
21  (let ([x* t* e*]) body* ... body)
22  (letrec ([x* t* e*]) body* ... body)
23  (letrec body)
24  (let body)
25  (letfun body)
26  (c (array len t [e* ...]))
27  (e0 e1 ...)))
```