UNIVERSIDADE FEDERAL DO PARANÁ

DEPARTAMENTO DE INFORMÁTICA

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

CI211 - CONSTRUÇÃO DE COMPILADORES

Trabalho 2 - Analisador Sintático

Nome: Henrique Colodetti Escanferla - GRR20135427

ALTERAÇÕES DO ANALISADOR LÉXICO

Retirado: ATR => [=] Adicionado: = => [=] const => [c][o][n][s][t]

main => [m][a][i][n] function => [f][u][n][c][t][i][o][n]

return => [r][e][t][u][r][n]

GRAMÁTICA AUMENTADA

Obs.: red nº da redução (nº de pop da pilha)

As regras de NUM, TIPO, OP\_REL e OP\_LOG foram descartadas pois não são necessárias. O analisador léxico trata como itens léxicos.

Adicionado o lexical "function" em FUNCAO para evitar conflito entre DECL\_VAR e FUNCAO e gerar mínimas modificações ao autômato.

Adicionado os lexicais “then” e “do” em IF e WHILE respectivamente.

S => PROG # PROG => DECL\_VAR CONST FUNCOES MAIN - red 1 (4)

DECL\_VAR => type id ; DECL\_VAR - red 2 (4) DECL\_VAR => € - red 38 (0)

CONST => const id = num ; CONST - red 3 (6) CONST => € - red 39 (0)

FUNCOES => FUNCAO FUNCOES - red 4 (2)

FUNCOES => € - red 40 (0)

FUNCAO => function type id ( ) { DECL\_VAR CMDS return id ; } - red 5 (12)

MAIN => main { DECL\_VAR CMDS } - red 6 (5)

CMDS => CMD CMDS - red 7 (2) CMDS => CMD - red 8 (1)

CMD => IF - red 9 (1) CMD => WHILE - red 10 (1)

CMD => FOR - red 11 (1) CMD => ATR - red 12 (1)

CMD => CH\_FUNC - red 13 (1) CMD => INCR ; - red 14 (2)

IF => if ( TESTE ) then { CMDS } - red 15 (8)

IF => if ( TESTE ) then { CMDS } else { CMDS } - red 16 (12)

WHILE => while ( TESTE ) do { CMDS } - red 17 (8)

FOR => for ( ATR\_FOR ; TESTE ; INCR ) { CMDS } - red 18 (11)

ATR => id = num ; - red 19 (4) ATR => id = EXPR\_MAT ; - red 20 (4)

ATR => id = ch ; - red 21 (4) ATR => id = str ; - red 22 (4)

ATR => id = CH\_FUNC - red 23 (3) ATR => id = id ; - red 24 (4)

CH\_FUNC => id ( ) ; - red 25 (4) INCR => id inc - red 26 (2)

ATR\_FOR => id = num - red 27 (3)

EXPR\_MAT => num op\_art num - red 28 (3)

EXPR\_MAT => id op\_art id - red 29 (3)

EXPR\_MAT => id op\_art num - red 30 (3)

EXPR\_MAT => num op\_art id - red 31 (3)

TESTE => EXPR\_REL - red 32 (1)

TESTE => EXPR\_REL op\_log TESTE - red 33 (3)

EXPR\_REL => num op\_rel num - red 34 (3)

EXPR\_REL => id op\_rel id - red 35 (3)

EXPR\_REL => id op\_rel num - red 36 (3)

EXPR\_REL => num op\_rel id - red 37 (3)

ITENS LR(0)

S => . PROG # | PROG . # | PROG # .

PROG => . DECL\_VAR CONST FUNCOES MAIN |

DECL\_VAR . CONST FUNCOES MAIN |

DECL\_VAR CONST . FUNCOES MAIN |

DECL\_VAR CONST FUNCOES . MAIN |

DECL\_VAR CONST FUNCOES MAIN .

DECL\_VAR => . | . type id ; DECL\_VAR | type . id ; DECL\_VAR |

type id . ; DECL\_VAR | type id ; . DECL\_VAR | type id ; DECL\_VAR .

CONST => . | . const id = num ; CONST | const . id = num ; CONST |

const id . = num ; CONST | const id = . num ; CONST |

const id = num . ; CONST | const id = num ; . CONST |

const id = num ; CONST .

FUNCOES => . | . FUNCAO FUNCOES | FUNCAO . FUNCOES |

FUNCAO FUNCOES .

FUNCAO => . function type id ( ) { DECL\_VAR CMDS return id ; } |

function . type id ( ) { DECL\_VAR CMDS return id ; } |

function type . id ( ) { DECL\_VAR CMDS return id ; } |

function type id . ( ) { DECL\_VAR CMDS return id ; } |

function type id ( . ) { DECL\_VAR CMDS return id ; } |

function type id ( ) . { DECL\_VAR CMDS return id ; } |

function type id ( ) { . DECL\_VAR CMDS return id ; } |

function type id ( ) { DECL\_VAR . CMDS return id ; } |

function type id ( ) { DECL\_VAR CMDS . return id ; } |

function type id ( ) { DECL\_VAR CMDS return . id ; } |

function type id ( ) { DECL\_VAR CMDS return id . ; } |

function type id ( ) { DECL\_VAR CMDS return id ; . } |

function type id ( ) { DECL\_VAR CMDS return id ; } .

MAIN => . main { DECL\_VAR CMDS } | main . { DECL\_VAR CMDS } |

main { . DECL\_VAR CMDS } | main { DECL\_VAR . CMDS } |

main { DECL\_VAR CMDS . } | main { DECL\_VAR CMDS } .

CMDS => . CMD | CMD . | . CMD CMDS | CMD . CMDS | CMD CMDS .

CMD => . IF | IF . | . WHILE | WHILE . | . FOR | FOR . | . ATR | ATR . |

. CH\_FUNC | CH\_FUNC . | . INCR ; | INCR . ; | INCR ; .

IF => . if ( TESTE ) then { CMDS } | if . ( TESTE ) then { CMDS } |

if ( . TESTE ) then { CMDS } | if ( TESTE . ) then { CMDS } |

if ( TESTE ) . then { CMDS } | if ( TESTE ) then . { CMDS } |

if ( TESTE ) then { . CMDS } | if ( TESTE ) then { CMDS . } |

if ( TESTE ) then { CMDS } . | . if ( TESTE ) then { CMDS } else { CMDS } |

if . ( TESTE ) then { CMDS } else { CMDS } |

if ( . TESTE ) then { CMDS } else { CMDS } |

if ( TESTE . ) then { CMDS } else { CMDS } |

if ( TESTE ) . then { CMDS } else { CMDS } |

if ( TESTE ) then . { CMDS } else { CMDS } |

if ( TESTE ) then { . CMDS } else { CMDS } |

if ( TESTE ) then { CMDS . } else { CMDS } |

if ( TESTE ) then { CMDS } . else { CMDS } |

if ( TESTE ) then { CMDS } else . { CMDS } |

if ( TESTE ) then { CMDS } else { . CMDS } |

if ( TESTE ) then { CMDS } else { CMDS . } |

if ( TESTE ) then { CMDS } else { CMDS } .

WHILE => . while ( TESTE ) do { CMDS } | while . ( TESTE ) do { CMDS } |

while ( . TESTE ) do { CMDS } | while ( TESTE . ) do { CMDS } |

while ( TESTE ) . do { CMDS } | while ( TESTE ) do . { CMDS } |

while ( TESTE ) do { . CMDS } | while ( TESTE ) do { CMDS . } |

while ( TESTE ) do { CMDS } .

FOR => . for ( ATR\_FOR ; TESTE ; INCR ) { CMDS } |

for . ( ATR\_FOR ; TESTE ; INCR ) { CMDS } |

for ( . ATR\_FOR ; TESTE ; INCR ) { CMDS } |

for ( ATR\_FOR . ; TESTE ; INCR ) { CMDS } |

for ( ATR\_FOR ; . TESTE ; INCR ) { CMDS } |

for ( ATR\_FOR ; TESTE . ; INCR ) { CMDS } |

for ( ATR\_FOR ; TESTE ; . INCR ) { CMDS } |

for ( ATR\_FOR ; TESTE ; INCR . ) { CMDS } |

for ( ATR\_FOR ; TESTE ; INCR ) . { CMDS } |

for ( ATR\_FOR ; TESTE ; INCR ) { . CMDS } |

for ( ATR\_FOR ; TESTE ; INCR ) { CMDS . } |

for ( ATR\_FOR ; TESTE ; INCR ) { CMDS } .

ATR => . id = num ; | id . = num ; | id = . num ; | id = num . ; | id = num ; . |

. id = EXPR\_MAT ; | id . = EXPR\_MAT ; | id = . EXPR\_MAT ; |

id = EXPR\_MAT . ; | id = EXPR\_MAT ; . | . id = ch ; | id . = ch ; | id = . ch ; |

id = ch . ; | id = ch ; . | . id = str ; | id . = str ; | id = . str ; | id = str . ; |

id = str ; . | . id = CH\_FUNC | id . = CH\_FUNC | id = . CH\_FUNC |

id = CH\_FUNC . | . id = id ; | id . = id ; | id = . id ; | id = id . ; | id = id ; .

CH\_FUNC => . id ( ) ; | id . ( ) ; | id ( . ) ; | id ( ) . ; | id ( ) ; .

INCR => . id inc | id . inc | id inc .

ATR\_FOR => . id = num | id . = num | id = . num | id = num .

EXPR\_MAT => . num op\_art num | num . op\_art num | num op\_art . num |

num op\_art num . | . id op\_art id | id . op\_art id | id op\_art . id |

id op\_art id . | . num op\_art id | num . op\_art id | num op\_art . id |

num op\_art id . | . id op\_art num | id . op\_art num | id op\_art . num |

id op\_art num .

TESTE => . EXPR\_REL | EXPR\_REL . | . EXPR\_REL op\_log TESTE |

EXPR\_REL . op\_log TESTE | EXPR\_REL op\_log . TESTE |

EXPR\_REL op\_log TESTE .

EXPR\_REL => . num op\_rel num | num . op\_rel num | num op\_rel . num |

num op\_rel num . | . id op\_rel id | id . op\_rel id | id op\_rel . id |

id op\_rel id . | . num op\_rel id | num . op\_rel id | num op\_rel . id |

num op\_rel id . | . id op\_rel num | id . op\_rel num | id op\_rel . num |

id op\_rel num .

ESTADOS DO AUTÔMATO SLR(1)

e0: empilha PROG, empilha DECL\_VAR, empilha type ou red 38

S => . PROG #

PROG => . DECL\_VAR CONST FUNCOES MAIN

DECL\_VAR => . | . type id ; DECL\_VAR

e1: aceita a entrada se ler # logo não acusa erros

S => PROG . #

e2: red 1

PROG => DECL\_VAR CONST FUNCOES MAIN .

e3: empilha id

DECL\_VAR => type . id ; DECL\_VAR

e4: empilha ;

DECL\_VAR => type id . ; DECL\_VAR

e5: empilha DECL\_VAR, empilha type ou red 38

DECL\_VAR => type id ; . DECL\_VAR

DECL\_VAR => . | . type id ; DECL\_VAR

e6: empilha id

CONST => const . id = num ; CONST

e7: empilha =

CONST => const id . = num ; CONST

e8: empilha num

CONST => const id = . num ; CONST

e9: empilha ;

CONST => const id = num . ; CONST

e10: empilha CONST, empilha const ou red 39

CONST => const id = num ; . CONST

CONST => . | . const id = num ; CONST

e11: empilha FUNCOES, empilha FUNCAO, empilha function ou red 40

FUNCOES => FUNCAO . FUNCOES

FUNCOES => . | . FUNCAO FUNCOES

FUNCAO => . function type id ( ) { DECL\_VAR CMDS return id ; }

e12: empilha )

FUNCAO => function type id ( . ) { DECL\_VAR CMDS return id ; }

e13: empilha {

FUNCAO => function type id ( ) . { DECL\_VAR CMDS return id ; }

e14: empilha DECL\_VAR, empilha type ou red 38

FUNCAO => function type id ( ) { . DECL\_VAR CMDS return id ; }

DECL\_VAR => . | . type id ; DECL\_VAR

e15: empilha return

FUNCAO => function type id ( ) { DECL\_VAR CMDS . return id ; }

e16: empilha id

FUNCAO => function type id ( ) { DECL\_VAR CMDS return . id ; }

e17: empilha ;

FUNCAO => function type id ( ) { DECL\_VAR CMDS return id . ; }

e18: empilha }

FUNCAO => function type id ( ) { DECL\_VAR CMDS return id ; . }

e19: red 5

FUNCAO => function type id ( ) { DECL\_VAR CMDS return id ; } .

e20: empilha {

MAIN => main . { DECL\_VAR CMDS }

e21: empilha DECL\_VAR, empilha type ou red 38

MAIN => main { . DECL\_VAR CMDS }

DECL\_VAR => . | . type id ; DECL\_VAR

e22: empilha }

MAIN => main { DECL\_VAR CMDS . }

e23: red 6

MAIN => main { DECL\_VAR CMDS } .

e24: empilha CMDS, empilha CMD, empilha IF, empilha WHILE, empilha FOR, empilha ATR, empilha CH\_FUNC, empilha INCR, empilha if, empilha while, empilha for, empilha id ou red 8

CMDS => CMD . | CMD . CMDS

CMDS => **. CMD** | **. CMD** CMDS

CMD => . IF | . WHILE | . FOR | . ATR | . CH\_FUNC | . INCR ;

IF => **. if** ( TESTE ) then { CMDS } |

**. if** ( TESTE ) then { CMDS } else { CMDS }

WHILE => . while ( TESTE ) do { CMDS }

FOR => . for ( ATR\_FOR ; TESTE ; INCR ) { CMDS }

ATR => **. id** = num ; | **. id** = EXPR\_MAT ; | **. id** = ch ; | **. id** = str ; |

**. id = CH\_FUNC** | **. id** = id ;

CH\_FUNC => **. id** ( ) ; - INCR => **. id** inc

e25: red 7

CMDS => CMD CMDS .

e26: red 9

CMD => IF .

e27: red 10

CMD => WHILE .

e28: red 11

CMD => FOR .

e29: red 12

CMD => ATR .

e30: red 13

CMD => CH\_FUNC .

e31: empilha ;

CMD => INCR . ;

e32: red 14

CMD => INCR ; .

e33: empilha (

IF => if **. (** TESTE ) then { CMDS } | if **. (** TESTE ) then { CMDS } else { CMDS }

e34: empilha TESTE, empilha EXPR\_REL, empilha num ou empilha id

IF => if ( **. TESTE** ) then { CMDS } | if ( **. TESTE** ) then { CMDS } else { CMDS }

TESTE => **. EXPR\_REL** | **. EXPR\_REL** op\_log TESTE

EXPR\_REL => **. num** op\_rel num | **. id** op\_rel id | **. num** op\_rel id |

**. id** op\_rel num

e35: empilha )

IF => if ( TESTE **. )** then { CMDS } | if ( TESTE **. )** then { CMDS } else { CMDS }

e36: empilha then

IF => if ( TESTE ) **. then** { CMDS } | if ( TESTE ) **. then** { CMDS } else { CMDS }

e37: empilha {

IF => if ( TESTE ) then **. {** CMDS } |

if ( TESTE ) then **. {** CMDS } else { CMDS }

e38: empilha CMDS, empilha CMD, empilha IF, empilha WHILE, empilha FOR,

empilha ATR, empilha CH\_FUNC, empilha INCR, empilha if, empilha while,

empilha for ou empilha id

IF => if ( TESTE ) then { **. CMDS** } |

if ( TESTE ) then { **. CMDS** } else { CMDS }

CMDS => **. CMD** | **. CMD** CMDS

CMD => . IF | . WHILE | . FOR | . ATR | . CH\_FUNC | . INCR ;

IF => **. if** ( TESTE ) then { CMDS } |

**. if** ( TESTE ) then { CMDS } else { CMDS }

WHILE => . while ( TESTE ) do { CMDS }

FOR => . for ( ATR\_FOR ; TESTE ; INCR ) { CMDS }

ATR => **. id** = num ; | **. id** = EXPR\_MAT ; | **. id** = ch ; | **. id** = str ; |

**. id = CH\_FUNC** | **. id** = id ;

CH\_FUNC => **. id** ( ) ;

INCR => **. id** inc

e39: empilha }

IF => if ( TESTE ) then { CMDS **. }** |

if ( TESTE ) then { CMDS **. }** else { CMDS }

e40: empilha else ou red 15

IF => if ( TESTE ) then { CMDS } . |

if ( TESTE ) then { CMDS } . else { CMDS }

e41: empilha {

IF => if ( TESTE ) then { CMDS } else . { CMDS }

e42: empilha CMDS, empilha CMD, empilha IF, empilha WHILE, empilha FOR, empilha ATR, empilha CH\_FUNC, empilha INCR, empilha if, empilha while, empilha for ou empilha id

IF => if ( TESTE ) then { CMDS } else { . CMDS }

CMDS => **. CMD** | **. CMD** CMDS

CMD => . IF | . WHILE | . FOR | . ATR | . CH\_FUNC | . INCR ;

IF => **. if** ( TESTE ) then { CMDS } |

**. if** ( TESTE ) then { CMDS } else { CMDS }

WHILE => . while ( TESTE ) do { CMDS }

FOR => . for ( ATR\_FOR ; TESTE ; INCR ) { CMDS }

ATR => **. id** = num ; | **. id** = EXPR\_MAT ; | **. id** = ch ; | **. id** = str ; |

**. id = CH\_FUNC** | **. id** = id ;

CH\_FUNC => **. id** ( ) ; - INCR => **. id** inc

e43: empilha }

IF => if ( TESTE ) then { CMDS } else { CMDS . }

e44: red 16

IF => if ( TESTE ) then { CMDS } else { CMDS } .

e45: empilha (

WHILE => while . ( TESTE ) do { CMDS }

e46: empilha TESTE, empilha EXPR\_REL, empilha num ou empilha id

WHILE => while ( . TESTE ) do { CMDS }

TESTE => **. EXPR\_REL** | **. EXPR\_REL** op\_log TESTE

EXPR\_REL => **. num** op\_rel num | **. id** op\_rel id | **. num** op\_rel id |

**. id** op\_rel num

e47: empilha )

WHILE => while ( TESTE . ) do { CMDS }

e48: empilha do

WHILE => while ( TESTE ) . do { CMDS }

e49: empilha {

WHILE => while ( TESTE ) do . { CMDS }

e50: empilha CMDS, empilha CMD, empilha IF, empilha WHILE, empilha FOR, empilha ATR, empilha CH\_FUNC, empilha INCR, empilha if, empilha while, empilha for ou empilha id

WHILE => while ( TESTE ) do { . CMDS }

CMDS => **. CMD** | **. CMD** CMDS

CMD => . IF | . WHILE | . FOR | . ATR | . CH\_FUNC | . INCR ;

IF => **. if** ( TESTE ) then { CMDS } | **. if** ( TESTE ) then { CMDS } else { CMDS }

WHILE => . while ( TESTE ) do { CMDS }

FOR => . for ( ATR\_FOR ; TESTE ; INCR ) { CMDS }

ATR => **. id** = num ; | **. id** = EXPR\_MAT ; | **. id** = ch ; | **. id** = str ; |

**. id = CH\_FUNC** | **. id** = id ; - CH\_FUNC => **. id** ( ) ; - INCR => **. id** inc

e51: empilha }

WHILE => while ( TESTE ) do { CMDS . }

e52: red 17

WHILE => while ( TESTE ) do { CMDS } .

e53: empilha (

FOR => for . ( ATR\_FOR ; TESTE ; INCR ) { CMDS }

e54: empilha ATR\_FOR ou empilha id

FOR => for ( . ATR\_FOR ; TESTE ; INCR ) { CMDS } - ATR\_FOR => . id = num

e55: empilha ;

FOR => for ( ATR\_FOR . ; TESTE ; INCR ) { CMDS }

e56: empilha TESTE, empilha EXPR\_REL, empilha num ou empilha id

FOR => for ( ATR\_FOR ; . TESTE ; INCR ) { CMDS }

TESTE => **. EXPR\_REL** | **. EXPR\_REL** op\_log TESTE

EXPR\_REL => **. num** op\_rel num | **. id** op\_rel id | **. num** op\_rel id |

**. id** op\_rel num

e57: empilha ;

FOR => for ( ATR\_FOR ; TESTE . ; INCR ) { CMDS }

e58: empilha INCR ou empilha id

FOR => for ( ATR\_FOR ; TESTE ; . INCR ) { CMDS } - INCR => . id inc

e59: empilha )

FOR => for ( ATR\_FOR ; TESTE ; INCR . ) { CMDS }

e60: empilha {

FOR => for ( ATR\_FOR ; TESTE ; INCR ) . { CMDS }

e61: empilha CMDS, empilha CMD, empilha IF, empilha WHILE, empilha FOR, empilha ATR, empilha CH\_FUNC, empilha INCR, empilha if, empilha while, empilha for ou empilha id

FOR => for ( ATR\_FOR ; TESTE ; INCR ) { . CMDS }

CMDS => **. CMD** | **. CMD** CMDS

CMD => . IF | . WHILE | . FOR | . ATR | . CH\_FUNC | . INCR ;

IF => **. if** ( TESTE ) then { CMDS } | **. if** ( TESTE ) then { CMDS } else { CMDS }

WHILE => . while ( TESTE ) do { CMDS }

FOR => . for ( ATR\_FOR ; TESTE ; INCR ) { CMDS }

ATR => **. id** = num ; | **. id** = EXPR\_MAT ; | **. id** = ch ; | **. id** = str ; |

**. id = CH\_FUNC** | **. id** = id ; - CH\_FUNC => **. id** ( ) ; - INCR => **. id** inc

e62: empilha }

FOR => for ( ATR\_FOR ; TESTE ; INCR ) { CMDS . }

e63: red 18

FOR => for ( ATR\_FOR ; TESTE ; INCR ) { CMDS } .

e64: empilha =, empilha ( ou empilha inc

ATR => id **. =** num ; | id **. =** EXPR\_MAT ; | id **. =** ch ; | id **. =** str ; |

id **. =** CH\_FUNC | id **. =** id ; - CH\_FUNC => id . ( ) ; - INCR => id . inc

e65: empilha EXPR\_MAT, empilha CH\_FUNC, empilha num, empilha ch, empilha str ou empilha id

ATR => id = **. num** ; | id = . EXPR\_MAT ; | id = . ch ; | id = . str ; |

id = . CH\_FUNC | id = **. id** ; - CH\_FUNC => **. id** ( ) ;

EXPR\_MAT => **. num** op\_art num | **. id** op\_art id | **. num** op\_art id |

**. id** op\_art num

e66: empilha op\_art ou empilha ;

ATR => id = num . ; - EXPR\_MAT => num **. op\_art** num | num **. op\_art** id

e67: red 19

ATR => id = num ; .

e68: empilha ;

ATR => id = EXPR\_MAT . ;

e69: red 20

ATR => id = EXPR\_MAT ; .

e70: empilha ;

ATR => id = ch . ;

e71: red 21

ATR => id = ch ; .

e72: empilha ;

ATR => id = str . ;

e73: red 22

ATR => id = str ; .

e74: red 23

ATR => id = CH\_FUNC .

e75: empilha op\_art, empilha ( ou empilha ;

ATR => id = id . ; - CH\_FUNC => id . ( ) ;

EXPR\_MAT => id **. op\_art** num | id **. op\_art** id

e76: red 24

ATR => id = id ; .

e77: empilha )

CH\_FUNC => id ( . ) ;

e78: empilha ;

CH\_FUNC => id ( ) . ;

e79: red 25

CH\_FUNC => id ( ) ; .

e80: red 26

INCR => id inc .

e81: empilha =

ATR\_FOR => id . = num

e82: empilha num

ATR\_FOR => id = . num

e83: red 27

ATR\_FOR => id = num .

e84: empilha num ou empilha id

EXPR\_MAT => num op\_art . num |

num op\_art . id

e85: red 28

EXPR\_MAT => num op\_art num .

e86: red 31

EXPR\_MAT => num op\_art id .

e87: empilha num ou empilha id

EXPR\_MAT => id op\_art . num | id op\_art . id

e88: red 30

EXPR\_MAT => id op\_art num .

e89: red 29

EXPR\_MAT => id op\_art id .

e90: empilha op\_log ou red 32

TESTE => EXPR\_REL . | EXPR\_REL . op\_log TESTE

e91: empilha TESTE, empilha EXPR\_REL, empilha num ou empilha id

TESTE => EXPR\_REL op\_log . TESTE | **. EXPR\_REL** |

**. EXPR\_REL** op\_log TESTE

EXPR\_REL => **. num** op\_rel num | **. id** op\_rel id | **. num** op\_rel id |

**. id** op\_rel num

e92: red 33

TESTE => EXPR\_REL op\_log TESTE .

e93: empilha op\_rel

EXPR\_REL => num **. op\_rel** num | num **. op\_rel** id

e94: empilha num ou empilha id

EXPR\_REL => num op\_rel . num | num op\_rel . id

e95: red 34

EXPR\_REL => num op\_rel num .

e96: red 37

EXPR\_REL => num op\_rel id .

e97: empilha op\_rel

EXPR\_REL => id **. op\_rel** num | id **. op\_rel** id

e98: empilha num ou empilha id

EXPR\_REL => id op\_rel . num | id op\_rel . id

e99: red 36

EXPR\_REL => id op\_rel num .

e100: red 35

EXPR\_REL => id op\_rel id .

e101: empilha type

FUNCAO => function . type id ( ) { DECL\_VAR CMDS return id ; }

e102: empilha id

FUNCAO => function type . id ( ) { DECL\_VAR CMDS return id ; }

e103: empilha (

FUNCAO => function type id . ( ) { DECL\_VAR CMDS return id ; }

e104: red 2

DECL\_VAR => type id ; DECL\_VAR .

e105: red 3

CONST => const id = num ; CONST .

e106: red 4

FUNCOES => FUNCAO FUNCOES .

e107: empilha CONST, empilha const ou red 39

PROG => DECL\_VAR . CONST FUNCOES MAIN

CONST => . | . const id = num ; CONST

e108: empilha FUNCOES, empilha FUNCAO, empilha function ou red 40

PROG => DECL\_VAR CONST . FUNCOES MAIN

FUNCOES => . | . FUNCAO FUNCOES

FUNCAO => . function type id ( ) { DECL\_VAR CMDS return id ; }

e109: empilha MAIN ou empilha main

PROG => DECL\_VAR CONST FUNCOES . MAIN

MAIN => . main { DECL\_VAR CMDS }

e110: empilha CMDS, empilha CMD, empilha IF, empilha WHILE, empilha FOR, empilha ATR, empilha CH\_FUNC, empilha INCR, empilha if, empilha while, empilha for ou empilha id

FUNCAO => function type id ( ) { DECL\_VAR . CMDS return id ; }

CMDS => **. CMD** | **. CMD** CMDS

CMD => . IF | . WHILE | . FOR | . ATR | . CH\_FUNC | . INCR ;

IF => **. if** ( TESTE ) then { CMDS } | **. if** ( TESTE ) then { CMDS } else { CMDS }

WHILE => . while ( TESTE ) do { CMDS }

FOR => . for ( ATR\_FOR ; TESTE ; INCR ) { CMDS }

ATR => **. id** = num ; | **. id** = EXPR\_MAT ; | **. id** = ch ; | **. id** = str ; |

**. id = CH\_FUNC** | **. id** = id ; - CH\_FUNC => **. id** ( ) ; - INCR => **. id** inc

e111: empilha CMDS, empilha CMD, empilha IF, empilha WHILE, empilha FOR, empilha ATR, empilha CH\_FUNC, empilha INCR, empilha if, empilha while, empilha for ou empilha id

MAIN => main { DECL\_VAR . CMDS } CMDS => **. CMD** | **. CMD** CMDS

CMD => . IF | . WHILE | . FOR | . ATR | . CH\_FUNC | . INCR ;

IF => **. if** ( TESTE ) then { CMDS } |

**. if** ( TESTE ) then { CMDS } else { CMDS }

WHILE => . while ( TESTE ) do { CMDS }

FOR => . for ( ATR\_FOR ; TESTE ; INCR ) { CMDS }

ATR => **. id** = num ; | **. id** = EXPR\_MAT ; | **. id** = ch ; | **. id** = str ; |

**. id = CH\_FUNC** | **. id** = id ; - CH\_FUNC => **. id** ( ) ; - INCR => **. id** inc

CONJUNTOS PRIMEIRO

P(PROG) = { type, const, function, main } P(DECL\_VAR) = { type, € }

P(FUNCOES) = { function, € } P(CONST) = { const, € }

P(FUNCAO) = { function } P(MAIN) = { main }

P(CMDS) = P(CMD) = { if, while, for, id } P(IF) = { if } P(WHILE) = { while }

P(FOR) = { for } P(ATR) = P(CH\_FUNC) = P(INCR) = P(ATR\_FOR) = { id }

P(EXPR\_MAT) = P(EXPR\_REL) = P(TESTE) = { num, id }

CONJUNTOS SEQUÊNCIA

S(PROG) = S(MAIN) = { # }

S(DECL\_VAR) = { const, function, main, if, while, for, id }

S(FUNCOES) = { main } S(CONST) = S(FUNCAO) = { function, main }

S(CMDS) = { return, } } S(ATR\_FOR) = S(EXPR\_MAT) = { ; }

S(TESTE) = S(INCR) = { ), ; }

S(CMD) = S(IF) = S(WHILE) = S(FOR) = S(CH\_FUNC) = S(ATR) =

{ if, while, for, id, return, } }

S(EXPR\_REL) = { ), ;, op\_log }

COMENTÁRIOS SOBRE A IMPLEMENTAÇÃO

1. Foi definido enums de constantes inteiras para as linhas e colunas da tabela alem dos itens contidos nela. Nos enums, existe o numero da regra e o numero de pop quando aplicável. Também existe representação do lexical em string para relacionar com o respectivo enum. Exceto a string mencionada, todo o resto foi declarado como inteiros e usados para indexar a matriz de inteiros que representa a tabela de análise sintática.
2. O algoritmo que percorre a tabela é bastante simples. Existe uma funçao que retira até o ponto-e-virgula do item lexico retornado pelo analizador lexico. A partir disso, é feita uma leitura da posição atual na tabela e, por meio de alguns if-elses, é decidido o que fazer. Este passo define qual o próximo item da tabela e o que é feito com o lexical lido. Podemos empilhar, reduzir, aceitar finalizando a execução ou imprimir mensagem de erro e terminar a execução.
3. Quando ocorre uma redução, desempilhamos a quantidade indicada pela constante declarada e escrevemos a constante que indica o simbolo reduzido. Logo em seguida, lemos tal simbolo e a posição atual da tabela e empilhamos o estado indicado na tabela. Isso foi feito para que não houvesse necessidade de perguntar se existe simbolo reduzido para ler no inicio do algoritmo que percorre a tabela afinal sempre podemos assumir que precisamos ler o simbolo reduzido e empilhar o próximo estado após fazer uma redução.