AST Rules

Compiler Group-29

Shaurya Marwah - 2019B3A70459P
Ruchir Kumbhare - 2019B5A70650P
Ashwin Murali - 2019B2A70957P
Mari Sankar - 2019B3A70564P
Dilip Venkatesh - 2020A7PS1203P

1.

a) Initialize othermodules(2).inh_list = NULL
b) Initialize othermodules(1).inh_list = NULL
c) Initialize moduleDeclarations.inh_list = NULL
// ↓

d) Program.addr = mk_node(label: program, mk_node(label: 'moduleDeclarations', moduleDeclarations.addr), mk_node(label: 'othermodules', othermodules(2).addr), mk_node(label: 'othermodules', othermodules(1).addr), mk_node(label: 'drivermodule', drivermodule.addr), mk_node(label: 'othermodules', othermodules(1).addr)) // ↑

e) free(moduleDeclarations, otherModules(2), driverModule, otherModules(1)) // ↑

2.

a) moduleDeclarations(1).inh_list = append_at_end(moduleDeclarations.inh_list, moduleDeclarations.addr) // ↓

b) moduleDeclarations.syn_list = moduleDeclarations(1).syn_list // ↑

c) free(moduleDeclaration, moduleDeclarations(1)) // ↑

3

a) moduleDeclarations.syn_list = moduleDeclarations.inh_list

b) free(EPSILON)

4

a) moduleDeclarations.addr = mk_node(label: 'moduleDeclaration', ID.addr)
b) free(DECLARE, MODULE, SEMICOL)

5

a) othermodules(1).snh_list = append_at_end(othermodules.inh_list, module.addr) //↓
b) Othermodules.syn_list = othermodules(1).syn_list //↑
c) free(module, otherModules(1)) //↑

6

a) Othermodules.syn_list = othermodules(1).syn_list
b) free(EPSILON)

7

a) Drivermodule.addr = module.addr
b) free(DRIVERDEF, DRIVER, PROGRAM, DRIVERENDDEF, moduleDef)

8

a) Module.addr = mk_node(label: 'module', ID.addr, input_plist.addr, ret.addr, moduleDef.addr)
b) free(DEF, MODULE, ENDDEF, TAKES, INPUT, SQBO, input_plist, SQBC, SEMICOL, ret, moduleDef)

9

a) Ret.addr = ouput_plist.addr
b) free(RETURNS, SQBO, output_plist, SQBC, SEMICOL)

10)

a) Ret.addr = NULL
b) free(EPSILON)

Compiler Group-29 Shaurya, Ruchir, Ashwin, Hari, Dilip

a) Initialize input_plist.head = NULL //↓

b) input_plist.node = mk_linkedlist_node(ID.addr, dataType.addr) //↓

c) Input_plist.head = append_at_end(input_plist.head, input_plist.node, NULL) //↓

d) nl.inhList = input_plist.head //↑↓

e) input_plist.synList = nl.synList //↑

f) Input_plist.addr = mk_node(label: 'input plist', NULL, input_plist.synList) //↑

g) free(colon, nl, dataType) //↑

12.

a) nl.node = mk_linkedlist_node(ID.addr, dataType.type) //↓

b) nl.inhList = append_at_end(nl.inhList, nl.node, NULL) //↓

c) nl(1).inhList = nl.inhList //↓

d) nl.synList = nl(1).synList //↑

e) free(COMMA, COLON, nl(1), dataType) //↑

13)

a) nl.synList = nl.inhList

b) free(EPSILON)

14)

a) Initialize output_plist.head = NULL //↓

b) output_plist.node = mk_linkedlist_node(ID.addr, type.type) //↓

c) ouput_plist.head = append_at_end(output_plist.head, output_plist.node, NULL)

//↓

d) n1.inhList = ouput_plist.head // ↓

e) output_plist.synList = n1.synList // ↑

f) output_plist.addr = mk_node (label: 'ouput list', NULL, output_plist.synList) // ↑

g) free (COLON, n2#, type) // ↑

15.

a) n2.node = mk_linkedlist_node (ID.addr, type.type) // ↓

b) n2.inhList = append_at_end (n2.inhList, n2.node, NULL) // ↓

c) n2(1).inhList = n2.inhList // ↓

d) n2.synList = n2(1).synList // ↑

e) free(COMMA, COLON, n2(1)#, type) // ↑

16

a) n2.synList = n2.inhList

b) free (EPSILON)

17

a) dataType.addr = INTEGER.addr

~~free (INTEGER)~~

18

a) dataType.addr = REAL.addr

~~free (REAL)~~

19

a) dataType.addr = BOOLEAN.addr

~~free (BOOLEAN)~~

20

a) Datatype1.addr = mk_node (label : 'array type', range_arrays.addr, type.addr)

b) free(ARRAY, SQBO, range_arrays, SQBC, OF, type)

Compiler Group-29 Shaunya, Rucher, Ashwin, Hari, Dilip

21. ~~Range_arrays~~

a) range : arrays . addr = mk_node (label: 'range', index_arr1.addr, index_arr2.addr)

b) free(index_arr, RANGEOP, index_arr)

22.

a) type . addr = INTEGER.addr

~~b) free (INTEGER)~~

23.

a) type . addr = REAL.addr

~~b) free (REAL)~~

24.

a) type . addr = BOOLEAN.addr

~~free (BOOLEAN)~~

25.

a) Slaements . inh_list = NULL; // ↓

b) module Def. slaements - list = statements . syn_list // ↑

c) module Def . addr = mk - node (label : 'slaements', slaements . syn - list, NULL) // ↑

d) free (START, statements, END) // ↑

26.

a) slaements (2) . inh_list = append_linkedlist_node (statement.addr, NULL) // ↓

b) statements (1). inh_list = slaements (2) . inh_list // ↓

c) slaements . syn_list = slaements (1) . syn_list // ↑

d) free (statements(1), statement) // ↑

27

a) Slaements . syn_list = slaements . inh_list

b) free (EPSILON)

28

a) Slaements . addr = ioStmt . addr

b) free (iostmt)

29.

a) Statements.addr = simpleStmt.addr
b) free(simpleStmt)

30

a) Statements.addr = declareStmt.addr
b) free(declareStmt)

31

a) Statements.addr = conditionalStmt.addr
b) free(conditionalStmt)

32

a) Statements.addr = iterativeStmt.addr
b) free(iterativeStmt)

33.

a) ioStmt.addr = mk_node("label: ioStmt", ID.addr, NULL)
b) free(get_value, LO, BC, semicol ?, var_print)

34

a) ioStmt.addr = var_print.addr
b) free(PRINT, BO, var_print, BC, SEMICOL)

35

a) boolConstt.addr = TRUE.addr
~~free(TRUE)~~

36

a) boolConstt.addr = FALSE.addr
~~b) free(FALSE)~~

37

a) id_num_rnum.addr = ID.addr

38

a) ~~id_num~~ id_num_rnum.addr = NUM.addr
~~b) free(NUM)~~

39

a) id_num_rnum.addr = RNUM.addr ~~b) free(RNUM)~~

Compiler group-29

0. ~~var_print~~

a) if(var_print_dash.addr = NULL)

var_print.addr = ID.addr

else

var_print.addr = mk_node(label: array_element, ID.addr, var_print_dash.addr)

b) free(var_print_dash)

4).

a) var_print = NUM.addr

~~b) free(NUM)~~

4) a) var_print = RNUM.addr

43 ~~b) free(RNUM)~~

a) var_print = boolConst.addr

44 b) free(boolConst)

a) var_print_dash.addr = p1.addr

45 b) free(p1)

a) var_print_dash.addr = index_arr.addr

46 b) free(SqBO, index_arr, SqBC)

a) ~~p1.addr~~ p1.addr = new_index.addr

47 b) free(SqBO, ~~index~~ new_index, SqBC)

a) p1.addr = NULL

48 b) free(EPSILON)

a) simpleStmt.addr = assignmentStmt.addr

49 b) free(assignmentStmt)

49.a) simpleStmt.addr = moduleReuseStmt.addr

b) free(moduleReuseStmt)

50

a) whichStmt.inh = ID.addr //↓

b) assignmentStmt.addr = mk_node(':=', whichStmt.syn, whichStmt.addr) //↑

c) free(whichStmt) //↑

Compiler group - 29

51.

a) whichStmt.addr = lvalueIDStmt.addr

b) whichStmt.syn = whichStmt.inh

c) free(lvalueIDStmt)

52.

a) lvalueARRStmt.inh = whichStmt.inh // ↓

b) whichStmt.syn = lvalueARRStmt.syn // ↑

c) whichStmt.addr = lvalueARRStmt.addr // ↑

d) free(lvalueARRStmt) // ↑

54.

a) lvalueIDStmt.addr = expression.addr

b) free(ASSIGNOP, expression, SEMICOL)

55.

a) lvalueARRStmt.syn = mk_node(label: 'array', lvalueARRStmt.inh, element_index_with_expressions.addr)

b) lvalueARRStmt.addr = expression.addr

c) free(SABO, element_index_with_expressions, SABC, ASSIGNOP, expression, SEMICOL)

56.

a) index_arr.addr = sign + new_index.addr // sign is concatenated with whatever lexeme new_index.addr has

b) free(sign, new_index)

57.

a) new_index.addr = NUM.addr, ~~b) free(NUM)~~

58

a) new_index.addr = ID.addr

59.

a) sign.sign = PLUS. ~~addr~~, b) free(PLUS)

60.

a) sign.sign = MINUS ~~addr~~ b) free(MINUS)

61.

a) sign.sign = ~~EPSILON.addr~~ NULL

b) free(EPSILON)

Compiler group-29

2.

a) inherited moduleReuseStmt.head = NULL // ↓

b) Actual_para_list.inh_list = moduleReuseStmt.head // ↓

c) moduleReuseStmt.addr = mk_node(label : "moduleReuseStmt", ID.addr, optional.addr, actual_para_list.addr) // ↑

d) free(optional, USE, MODULE, WITH, PARAMETERS, actual_para_list, SEMICOL) // ↑

63.

a) actual_para_list_dash.inh_list = appendat_end(actual_para_list.inh_list, (depending on sign) K.addr, NULL) // ↓

b) actual_para_list.addr = actual_para_list_dash.syn_list // ↑

c) free(sign, K, actual_para_list_dash)

64.

a) actual_para_list_dash.inh = appendat_end(actual_para_list.inh_list, boolConst.addr, NULL) // ↓

b) actual_para_list.addr = actual_para_list_dash.syn_list // ↑

c) free(boolConstt, actual_para_list_dash) // ↑

65

a) actual_para_list_dash(1).inh_list = appendat_end(actual_para_list.inh_list, boolConst.addr, NULL) // ↓

b) actual_para_list.addr = actual_para_list_dash.syn_list // ↑

c) free(COMMA, Kbool, actual_para_list_dash(1)) // ↑

66

a) kbool.addr = (Depending on sign) K.addr // sign is concatenated with whatever lexeme K.addr passes

b) free(sign, K)

67

a) kbool.addr = boolConstt.addr

b) free(boolConstt)

68

a) actual_para_list_dash.syn_list = actual_para_list_dash.inh_list

b) free(EPSILON)

Compiler group-29

69.

a) k.addr = NUM.addr

b) ~~free(NUM)~~

70.

a) k.addr = RNUM.addr

71 b) ~~free (RNUM)~~

a) if (n11.addr == NULL

k.addr = ID.addr

else

k.addr = mk_node (label: 'array_element', ID.addr, n_11.addr

b) free(n11)

72.

a) n11.addr = element_index_with_expressions.addr

b) free (SQBO, element_index_with_expressions, SQBC)

73

~~a) A11.addr11~~

a) n11.addr = NULL

b) free (EPSILON)

74.

a) optional.addr = idlist.synList

b) free (SQBO, idList, SQBC, ASSIGNOP)

75

a) optional.addr = NULL

b) free (EPSILON)

76.

a) idList.inhList = append_at_end (idList.inhList, ID, NULL);

b) n3.inhList = idList.inhList

c) idList.synList = n3.synList

d) free (n3)

Compiler Group - 29

77.

a) n3(2).inhList = append_at_end(n3(2).inhList, ID, NULL)

b) n3(1).inhList = n3(2).inhList

c) n3(2).synList = n3(1).synList

d) free(COMMA, n3(1))

78.

a) n3.synList = n3.inhList

b) free(EPSILON)

79.

a) Expression.addr = arithematicOrBooleanExpr.addr

80.

a) Expression.addr = u.addr

81.

82.

a) new_NT.addr = arithmeticExpr.addr

83.

a) new_NT.addr = var_id_num.addr

84.

a) var_id_num.addr = ID.addr

b) free(ID)

85.
a) var_id_num.addr = NUM.addr
b) ~~free(NUM)~~

86.
a) var_id_num.addr = RNUM.addr
~~b) free(RNUM)~~

87.
a) unary_op.addr = PLUS ~~addr~~
~~b) free(PLUS)~~ b) free(PLUS)

88.
a) unary_op.addr = MINUS ~~addr~~
~~b) free(MINUS)~~ b) free(MINUS)

89.
a) N7.inh = anyterm.addr
b) arithmetic or BooleanExpr.addr = n7.addr
c) free(anyterm, n7)

90.
a) n7(1).inh = anyterm.addr // ↓
b) n7(2).addr = mk_node(logOp.whatever_op, n7(2).inh, n7(1).addr) // ↑
c) free(logOp.anyterm, n7(1)) // ↑

91.
a) n7.addr = n7.inh
b) free(EPSILON)

92.
a) n8.inh = arithmeticExpr.addr // ↓
b) anyterm.addr = n8.addr // ↑
c) free(arithmeticExpr, ~~anyterm~~ n8) // ↑

93.
a) anyterm.addr = boolConst.addr
b) free(boolConst)

Compiler group-29

94.

a) n8.addr = mk_node(relationalOp.whatever_op, n8.inh, arithmeticExpr.addr)

b) free(relationalOp, arithmeticOp)

95.

a) n8.addr = n8.inh

b) free(EPSILON)

96.

a) n4.inh = term.addr // ↓

b) if(n4.addr != n4.inh)

i) arithmeticExpr.addr = n4.addr

c) else

i) arithmeticExpr.addr = term.addr

// ↑

d) free(term, n4) // ↑

97.

a) n4(1).inh = term.addr // ↓

b) if(n4(1).inh != n4(1).addr)

n4(2).addr = mk_node(op1.whatever_op, n4(2).inh, n4(1).addr)

c) else

n4(2).addr = mk_node(op1.whatever_op, n4(2).inh, term.addr)

// ↑

d) free(op1, n4, term) // ↑

99.

a) n4.addr = n4.inh

b) free(EPSILON)

100.

a) n5.inh = factor.addr // ↓

b) if(n5.addr != n5.inh)

term.addr = n5.addr

c) else
    term.addr = factor.addr } //↑

a) free(factor, n5) //↑

101.

a) n5(1).inh = factor.addr //↓

b) n5(2).addr = mk_node(op2.whatever_op, n5(2).inh, n5(1).addr) //↑

c) free(op2, factor, n5(1)) //↑

102.

a) n5.addr = n5.inh

b) free(EPSILON)

103.

a) factor.addr = arithmeticOrBooleanExpr.addr

b) free(arithmeticOrBooleanExpr)

104.

a) factor.addr = NUM.addr

b) ~~free(NUM)~~

105.

a) factor.addr = RNUM.addr

b) ~~free(RNUM)~~

106.

a) factor.addr = boolConst.addr

b) free(boolConst)

107.

a) if(array_element.addr == NULL)
    factor.addr = ID.addr

b) else
    factor.addr = mk_node(label: array_element, ID.addr, array_element.addr)

c) free(array_element)

108.

a) array_element.addr = element_index_with_expressions.addr

b) free(element_index_with_expression)

Compiler Group-29

109.
a) array_element.addr = NULL
b) free(EPSILON)

110.
a) element_index_with_expression.addr = sign + n10.addr // sign is concatenated with whatever lexeme n10 passes
b) free(sign, n10)

111.
a) element_index_with_expression.addr = arrExpr.addr
b) free(arrExpr)

112.
a) element_index_with_expression.addr = boolConst.addr
b) free(boolConst)

113.
a) n10.addr = new_index.addr
b) free(new_index)

114.
a) n10.addr = arrExpr.addr
b) free(arrExpr)

115.
a) arr_n4.inh = arrTerm.addr // ↓
b) arrExpr.addr = arr_n4.addr // ↑
c) free(arrTerm, arr_n4) // ↑

116.
a) arrExpr.addr = boolConst.addr
b) free(boolConst)

117.
a) arr_n4(1).inh = arrTerm.addr // ↓
b) arr_n4(2).addr = mk_node(op1.whatever_op, arr_n4(2).inh, arr_n4(1).addr) // ↑
c) free(arrTerm, arr_n4(1)) // ↑

118.
a) arr_n4.addr = arr_n4.inh
b) free(EPSILON)

119.
a) arr_n5.inh = arrFactor.addr // ↓
b) arrTerm.addr = arr_n5.addr // ↑
c) free(arrFactor, arr_n5) // ↑

120.
a) arr_n5(1).inh = arrTerm.addr // ↓
b) arr_n5(2).addr = mk_node(op2.whatever_op, arr_n5(2).inh, arr_n5(1).addr) // ↑
c) free(arr_n5(1), arrFactor, op2.) // ↑

121.
a) arr_n5.addr = arr_n5.inh
b) free(EPSILON)

122.
a) arrFactor.addr = ID.addr
~~b) free(ID)~~

123.
a) arrFactor.addr = NUM.addr
~~b) free(NUM)~~

~~125~~

125.
a) arrFactor.addr = arrExpr.addr
b) free(BO, BC, arrExpr)

126.
a) op1.addr = PLUS ~~addr~~
b) free(PLUS)

127.
a) op1.addr = MINUS ~~addr~~
b) free(MINUS)

128.
a) op2.addr = MUL ~~addr~~
b) free(MUL)

129.
a) op2.addr = DIV ~~addr~~
b) free(DIV)

Compiler Group-29

130.
a) logicalOp.addr = AND.~~addr~~
b) free(AND)

131.
a) logicalOp.addr = OR.~~addr~~
b) free(OR)

132.
a) relationalOp.addr = LT.~~addr~~
b) free(LT)

133.
a) relationalOp.addr = LE.~~addr~~
b) free(LE)

134.
a) relationalOp.addr = GT.~~addr~~
b) free(GT)

135.
a) relationalOp.addr = GE.~~addr~~
b) free(GE)

136.
a) relationalOp.addr = EQ.~~addr~~
b) free(EQ)

137.
a) relationalOp.addr = NE.~~addr~~
b) free(NE)

138.
a) initialize declareStmt.head = NULL // ↓
b) idList.inhList = declareStmt.head // ↓
c) declareStmt.synList = idList.synList // ↑
d) declareStmt.type = datatype.~~type~~ addr // ↑
e) declareStmt.addr = mk_node(label:'declare', declareStmt.type, declareStmt.synList) // ↑
f) free(DECLARE, COLON, SEMICOL~~ON~~, datatype, idList) // ↑

139.
a) initialize conditionalStmt.list_head = NULL // ↓
b) caseStmts.inh_list = conditionalStmt.list_head // ↓
c) conditionalStmt.addr = mk_node(label:'conditionalStmt', ID.addr, caseStmts.syn_list, default_addr)//↑
d) free(SWITCH, BO, BC, START, caseStmts, default_, END) //↑

140.
a) caseStmts.inh_list = append_at_end(mk_node(label:'caseStmt', value.addr, statements.addr)) //↓
b) n9.inh_list = caseStmts.inh_list //↓
c) caseStmts.syn_list = n9.syn_list //↑
d) free(CASE, value, COLON, statements, BREAK, SEMICOL, n9)//↑

141.
a) n9(2).inh_list = append_at_end(mk_node(label:'caseStmt', value.addr, statements.addr)) //↓
b) n9(1).inh_list = n9(2).inh_list //↓
c) n9(2).syn_list = n9(1).syn_list //↑
d) free(CASE, value, COLON, statements, BREAK, SEMICOL, n9(1))//↑

142.
a) n9.syn_list = n9.inh_list
b) free(EPSILON)

143.
a) value.addr = NUM.addr
b) ~~free(NUM)~~

144.
a) value.addr = TRUE.addr
b) ~~free(TRUE)~~

145.
a) value.addr = FALSE.addr
b) ~~free(FALSE)~~

146.
a) default_.addr = mk_node(label:'default', NULL, statements.addr)
b) free(DEFAULT, COLON, statements, BREAK, SEMICOL)

147.
a) default_.addr = NULL
b) free(EPSILON)

148.
a) iterativeStmt.addr = mk_node(label:'for', range_for_loop.addr, statements.addr, ID.addr
b) free(FOR, BO, IN, range_for_loop, BC, START, statements, END)

Compiler group-29

149. a) iterativeStmt.addr = mk_node(label:'while', arithmeticOrBooleanExpr.addr, statements.addr)

b) free(WHILE, BO, arithmeticOrBooleanExpr, BC, START, statements, END)

150.

a) if (sign_for_loop.sign == EPSILON || sign_for_loop.sign == PLUS)

index_for_loop.index = new_index_for_loop.num

b) else

index_for_loop.index = -new_index_for_loop.num

c) free(sign_for_loop, new_index_for_loop)

151.

a) new_index_for_loop.num = NUM.addr

b) ~~free(NUM)~~

152.

a) sign_for_loop.sign = PLUS ~~.addr~~

b) free(PLUS)

153.

a) sign_for_loop.sign = MINUS ~~.addr~~

b) free(MINUS)

154.

a) sign_for_loop.sign = ~~EPSILON.addr~~ NULL

b) free(EPSILON)

155.

a) range_for_loop.addr = mk_node(label:'range', index_for_loop1.index, index_for_loop2.index)

b) ~~free(rangeop)~~,

b) free(RANGEOP, index_for_loop1, index_for_loop2)