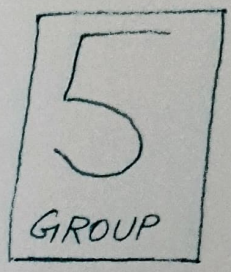
GROUP - 5

RULES FOR AST

NAME	ID
• ANTRIKSH SHARMA	2020A7PS1691P
• KAUSTAB CHOUDHURY	2020A7PS0013P
• HRISHIKESH HARSH	2020A7PS0313P
• HARSH PRIYADARSHI	2020A7PS0110P
• SHASHANK SHREEDHAR BHATT	2020A7PS0078P



LEGEND

TRAVERSAL

- ALL FREE( ) - UPWARD TRAVERSAL
- & - ADDRESS OF NODE(S)

ATTRIBUTES

INHERITED ATTRIBUTES

- head , • inh_addr

SYNTHESIZED ATTRIBUTES

- addr , • syn_addr , • heads , • index , • index If Array

1. program : moduleDeclarations otherModules driverModule otherModules,
2. moduleDeclarations : moduleDeclaration moduleDeclarations,
3. moduleDeclarations : EPSILON
4. moduleDeclaration : DECLARE MODULE ID SEMICOL
5. otherModules : module otherModules,
6. otherModules : EPSILON
7. driverModule : DRIVERDEF DRIVER PROGRAM DRIVERENDDEF moduleDef
8. module : DEF MODULE ID ENDDEF TAKES INPUT SQBO input_plist SQBC SEMICOL ret moduleDef
9. ret : RETURNS SQBO output_plist SQBC SEMICOL
10. ret : EPSILON
11. input_plist : ID COLON dataType n1
12. n1 : COMMA ID COLON dataType n1,
13. n1 : EPSILON
14. output_plist : ID COLON type_ n2
15. n2 : COMMA ID COLON type_ n2,
16. n2 : EPSILON
17. dataType : ARRAY SQBO range_arrays SQBC OF type_
18. dataType : INTEGER
19. dataType : REAL
20. dataType : BOOLEAN
21. range_arrays : index_arr RANGEOP index_arr,
22. index_arr : sign new_index
23. sign : PLUS
24. sign : MINUS
25. sign : EPSILON
26. new_index : NUM
27. new_index : ID
28. type_ : INTEGER
29. type_ : REAL
30. type_ : BOOLEAN
31. moduleDef : START statements END
32. statements : statement statements,
33. statements : EPSILON
34. statement : ioStmt
35. statement : simpleStmt
36. statement : declareStmt
37. statement : condionalStmt
38. statement : iterativeStmt
39. ioStmt : GET_VALUE BO ID BC SEMICOL
40. ioStmt : PRINT BO var_print BC SEMICOL
41. boolConstt : TRUE
42. boolConstt : FALSE
43. var_print : ID p1
44. var_print : NUM
45. var_print : RNUM
46. var_print : boolConstt
47. p1 : SQBO index_arr SQBC
48. p1 : EPSILON
49. simpleStmt : assignmentStmt
50. simpleStmt : moduleReuseStmt
51. assignmentStmt : ID whichStmt
52. whichStmt : lvalueIDStmt
53. whichStmt : lvalueARRStmt
54. lvalueIDStmt : ASSIGNOP expression SEMICOL
55. lvalueARRStmt : SQBO element_index_with_expressions SQBC ASSIGNOP expression SEMICOL
56. moduleReuseStmt : optional USE MODULE ID WITH PARAMETERS actual_para_list SEMICOL
57. optional : SQBO idList SQBC ASSIGNOP
58. optional : EPSILON
59. actual_para_list : sign par_print n_12
60. n_12 : COMMA sign par_print n_12,
61. n_12 : EPSILON
62. par_print : ID n_11
63. par_print : NUM
64. par_print : RNUM
65. par_print : boolConstt
66. idList : ID n3
67. n3 : COMMA ID n3,
68. n3 : EPSILON
69. expression : arithmeticOrBooleanExpr
70. expression : u
71. u : unary_op new_NT
72. new_NT : BO arithmeticExpr BC

73. new_NT : var_id_num
74. var_id_num : ID
75. var_id_num : NUM
76. var_id_num : RNUM
77. unary_op : PLUS
78. unary_op : MINUS
79. arithmeticOrBooleanExpr : anyTerm n7
80. n7 : logicalOp anyTerm n7
81. n7 : EPSILON
82. anyTerm : arithmeticExpr n8
83. n8 : relationalOp arithmeticExpr
84. n8 : EPSILON
85. arithmeticExpr : term n4
86. n4 : op1 term n4
87. n4 : EPSILON
88. term : factor n5
89. n5 : op2 factor n5
90. n5 : EPSILON
91. factor : BO arithmeticOrBooleanExpr BC
92. factor : NUM
93. factor : RNUM
94. factor : boolConstt
95. factor : ID n_11
96. n_11 : SQBO element_index_with_expressions SQBC
97. n_11 : EPSILON
98. element_index_with_expressions : arrExpr
99. element_index_with_expressions : uarr
100. uarr : unary_op new_NT_arr
101. new_NT_arr : BO arrExpr BC
102. new_NT_arr : var_id_num
103. arrExpr : arrTerm arr_N4
104. arr_N4 : op1 arrTerm arr_N4
105. arr_N4 : EPSILON
106. arrTerm : arrFactor arr_N5
107. arr_N5 : op2 arrFactor arr_N5
108. arr_N5 : EPSILON
109. arrFactor : ID
110. arrFactor : NUM
111. arrFactor : boolConstt
112. arrFactor : BO arrExpr BC
113. op1 : PLUS
114. op1 : MINUS
115. op2 : MUL
116. op2 : DIV
117. logicalOp : AND
118. logicalOp : OR
119. relationalOp : LT
120. relationalOp : LE
121. relationalOp : GT
122. relationalOp : GE
123. relationalOp : EQ
124. relationalOp : NE
125. declareStmt : DECLARE idList COLON dataType SEMICOL
126. condionalStmt : SWITCH BO ID BC START caseStmts default_ END
127. caseStmts : CASE value COLON statements BREAK SEMICOL n9
128. n9 : CASE value COLON statements BREAK SEMICOL n9
129. n9 : EPSILON
130. default_ : DEFAULT COLON statements BREAK SEMICOL
131. default_ : EPSILON
132. value : NUM
133. value : TRUE
134. value : FALSE
135. iterativeStmt : FOR BO ID IN range_for_loop BC START statements END
136. iterativeStmt : WHILE BO arithmeticOrBooleanExpr BC START statements END
137. range_for_loop : index_for_loop RANGEOP index_for_loop
138. index_for_loop : sign new_index_for_loop
139. new_index_for_loop : NUM

1. program.addr = CREATE_NODE ("PROGRAM", moduleDeclarations.head, otherModules.head, driverModule.addr, otherModules$_1$.head) //UP
   FREE (moduleDeclarations, otherModules, driverModule, otherModules$_1$)
2. INSERT_AT_END(moduleDeclarations.head, moduleDeclaration.addr) //DOWN
   moduleDeclarations$_1$.head = moduleDeclarations.head //DOWN
   FREE (moduleDeclaration, moduleDeclarations$_1$)
3. FREE (EPSILON)
4. moduleDeclaration.addr = &ID //UP
   FREE (DECLARE, MODULE, SEMICOL)
5. INSERT_AT_END (otherModules.head, module.addr) //DOWN
   otherModules$_1$.head = otherModules.head //DOWN
   FREE (module, otherModules$_1$)
6. FREE (EPSILON)
7. driverModule.addr = moduleDef.addr //UP
   FREE (DRIVERDEF, MODULE, PROGRAM, DRIVERENDDEF)
8. module.addr = CREATE_NODE ("MODULE", &ID, input_plist.head, ret.addr, moduleDef.addr) //UP
   FREE (DEF, MODULE, SEMICOL, ENDDEF, TAKES, INPUT, SQBO, SQBC, input_plist, ret, moduleDef)
9. ret.addr = output_plist.head //UP
   FREE (RETURNS, SQBO, SQBC, SEMICOL, output_plist)
10. ret.addr = NULL //UP
    FREE (EPSILON)
11. INSERT_AT_END (input_plist.head, CREATE_NODE ("I_PARAM", &ID, dataType.addr)) //DOWN
    n1.head = input_plist.head //DOWN
    FREE (COLON, dataType, n1)
12. INSERT_AT_END (n1.head, CREATE_NODE ("I_PARAM", &ID, dataType.addr)) //DOWN
    n1$_1$.head = n1.head //DOWN
    FREE (COLON, COMMA, dataType, n1$_1$)

28. type_.addr = &INTEGER. //UP

13. FREE(EPSILON)

14. INSERT_AT_END(output_plist.head, CREATE_NODE("O_PARAM", &ID, type_.addr)) //DOWN

n2.head = output_plist.head //DOWN

FREE(COLON, type_, n2)

15. INSERT_AT_END(n2.head, CREATE_NODE("O_PARAM", &ID, type_.addr)) //DOWN

$n2_1$.head = n2.head //DOWN

FREE(COLON, COMMA, type_, $n2_1$)

16. FREE(EPSILON)

17. DataType.addr = CREATE_NODE("ARRAY", range_arrays.addr, type_.addr) //UP

FREE(ARRAY, SQBO, SQBC, OF, range_arrays, type_)

18. DataType.addr = &INTEGER //UP

19. DataType.addr = &REAL //UP

20. DataType.addr = &BOOLEAN //UP

21. range_arrays.addr = CREATE_NODE("RANGE", index_arr.addr, $index\_arr_1$.addr) //UP

FREE(RANGEOP, index_arr, $index\_arr_1$)

22. if (sign.addr != NULL) { //UP

ADD_RIGHT_CHILD_TO_NODE(sign.addr, new_index.addr) //UP

index_arr.addr = sign.addr //UP

} else {

index_arr.addr = new_index.addr //UP

FREE(sign)

}

FREE(new_index)

23. sign.addr = &PLUS //UP

24. sign.addr = &MINUS //UP

25. sign.addr = NULL //UP

FREE(EPSILON)

26. new_index.addr = &NUM //UP

27. new_index.addr = &ID //UP

28. type_.addr = &INTEGER. //UP
29. type_.addr = &REAL // UP
30. type_.addr = &BOOLEAN //UP
31. moduleDef.addr = statements.head //UP
    FREE(START)
    FREE(statements)
    FREE(END)
32. INSERT_AT_END(statements_.head, statement.addr) //DOWN
    statements$_1$.head = statements.head //DOWN
    FREE(statement)
    FREE(statements)
33. FREE(EPSILON)
34. statement.addr = ioStmt.addr //UP
    FREE(ioStmt)
35. statement.addr = simpleStmt.addr //UP
    FREE(simpleStmt)
36. statement.addr = declareStmt.addr //UP
    FREE(declareStmt)
37. statement.addr = conditionalStmt.addr //UP
    FREE(conditionalStmt)
38. statement.addr = iterativeStmt.addr //UP
    FREE(iterativeStmt)
39. ioStmt.addr = CREATE_NODE("GET_VALUE", &ID) //UP
    FREE(GET_VALUE)
    FREE(BO)
    FREE(BC)
    FREE(SEMICOL)
40. ioStmt.addr = CREATE_NODE("PRINT", var_print.addr) //UP
    FREE(PRINT)
    FREE(BO)
    FREE(BC)
    FREE(SEMICOL)
41. boolConstt.addr = &TRUE //UP
42. boolConstt.addr = &FALSE //UP
43. var_print.addr = &ID //UP
    var_print.indexIfArray = p1.index //UP
    FREE(p1)

44. var_print.addr = &NUM //UP

45. var_print.addr = &RNUM //UP

46. var_print.addr = boolConstt.addr //UP
    FREE (boolConstt)

47. p1.index = index_arr.addr //UP
    FREE (SQBO)
    FREE (SQBC)
    FREE (index_arr)

48. p1.index = NULL //UP
    FREE (EPSILON)

49. simpleStmt.addr = assignmentStmt.addr //UP
    FREE (assignmentStmt)

50. simpleStmt.addr = moduleReuseStmt.addr //UP
    FREE (moduleReuseStmt)

51. assignmentStmt.addr = CREATE_NODE ("ASSIGN",
    &ID, NULL, NULL) //UP
    whichStmt.inh_addr = assignmentStmt.addr //DOWN
    FREE (whichStmt)

52. lvalueIDStmt.inh_addr = whichStmt.inh_addr //DOWN
    FREE (lvalueIDStmt)

53. lvalueARRStmt.inh_addr = whichStmt.inh_addr //DOWN
    FREE (lvalueARRStmt)

54. ADD_THIRD_CHILD_TO_NODE (lvalueIDStmt.inh_addr,
    expression.syn_addr) //UP
    FREE (expression)
    FREE (ASSIGNOP)
    FREE (SEMICOL)

55. ADD_SECOND_CHILD_TO_NODE (lvalueARRStmt.inh_addr,
    element_index_with_expressions.addr) //UP
    ADD_THIRD_CHILD_TO_NODE (lvalueARRStmt.inh_addr,
    expression.syn_addr) //UP
    FREE (ASSIGNOP)
    FREE (SEMICOL)
    FREE (SQBO)
    FREE (SQBC)
    FREE (element_index_with_expressions)
    FREE (expression)

56 → moduleReuseStmt.addr = CREATE_NODE("MODULE_USE", &ID, actual_para_list.head, optional.head)
↳ // UP
FREE (optional)
FREE (actual_para_list)

57 → optional.head = idList.head // UP
FREE (SQBO)
FREE (SQBC)
FREE (ASSIGNOP)
FREE (idList)

58 → optional.head = NULL // UP
FREE (EPSILON)

59 → if (sign.addr != NULL) {
ADD_RIGHT_CHILD_TO_NODE (sign.addr, par_print.addr) // UP
INSERT_AT_END (actual_para_list.head, sign.addr) // DOWN
}
else {
INSERT_AT_END (actual_para_list.head, par_print.addr) // DOWN
FREE (sign)
}
n_12.head = actual_para_list.head // DOWN
FREE (n_12)
FREE (par_print)

60 → if (sign.addr != NULL) {
ADD_RIGHT_CHILD_TO_NODE (sign.addr, par_print.addr) // UP
INSERT_AT_END (n_12.head, sign.addr) // DOWN
}
else {
INSERT_AT_END (n_12.head, par_print.addr) // DOWN
FREE (sign)
}
n_12'.head = n_12.head // DOWN
FREE (n_12')
FREE (COMMA)
FREE (par_print)

61 → FREE (EPSILON)

62 → par_print.addr = &ID // UP

par_print.indexIfArray = n_11.index // UP

FREE(n_11)

63 → par_print.addr = &NUM // UP

64 → par_print.addr = &RNUM // UP

65 → par_print.addr = boolConstt.addr // UP

FREE (boolConstt)

66 → INSERT_AT_END(idList.head, &ID) //DOWN

n3.head = idList.head //DOWN

FREE(n3)

67 → INSERT_AT_END(n3.head, &ID) //DOWN

$n3_1$.head = n3.head // DOWN

FREE (COMMA)

FREE ($n3_1$)

68 → FREE (EPSILON)

69 → expression.syn-addr = arithmeticOrBooleanExpr.syn-addr //UP

FREE (arithmeticORBooleanExpr)

70 → expression.syn-addr = u.addr // UP

FREE (u)

71 → u.addr = unary_op.addr // UP

ADD_RIGHT_CHILD_TO_NODE(unary_op.addr, NULL, new_NT.syn-addr) //UP

FREE(unary_op)

FREE(new_NT)

72 → new_NT.syn_addr = arithmeticExpr.syn-addr // UP

FREE(BO)

FREE(BC)

FREE (arithmeticExpr)

73 → new-NT.syn_addr = var_id_num.addr //UP

FREE (var_id_num)

74 → var_id_num.addr = &ID // UP

75 → var_id_num.addr = &NUM // UP

76 → var_id_num.addr = &RNUM // UP

77 → unary_op.addr = &PLUS // UP

78 → unary_op.addr = &MINUS // UP

79 → arithmetic OR Boolean Expr.syn_addr = n7.syn_addr // UP
arithmetic OR Boolean Expr.addr = anyTerm.addr // UP
n7.inh_addr = arithmetic OR Boolean Expr.addr // DOWN
FREE (anyTerm)
FREE (n7)

80 → ADD_LEFT_CHILD_TO_NODE (logicalOp.addr, n7.inh_addr) // DOWN
ADD_RIGHT_CHILD_TO_NODE (logicalOp.addr, anyTerm.addr) // UP
$n7_1$.inh_addr = logicalOp.addr // DOWN
n7.Synaddr = $n7_1$.syn_addr // UP
FREE (anyTerm)
FREE (logicalOp)
FREE ($n7_1$)

81 → n7.syn_addr = n7.inh_addr // UP
FREE (EPSILON)

82 → anyTerm.addr = arithmetic Expr.addr // UP
n8.inh_addr = anyTerm.addr // DOWN
anyTerm.syn_addr = n8.syn_addr // UP
FREE (arithmetic Expr)
FREE (n8)

83 → ADD_LEFT_CHILD_TO_NODE (relationalOp.addr, n8.inh_addr) // DOWN
ADD_RIGHT_CHILD_TO_NODE (relationalOp.addr, arithmetic Expr.addr) // UP
$n8_1$.inh_addr = relationalOp.addr // DOWN
n8.syn_addr = $n8_1$.syn_addr // UP
FREE (arithmetic Expr)
FREE (relationalOp)
FREE ($n8_1$)

84 → n8.syn_addr = n8.inh_addr // UP
FREE (EPSILON)

85 → arithmeticExpr.addr = term.addr // UP
n4.inh_addr = arithmetic.addr // DOWN
arithmeticExpr.syn_addr = n4.syn_addr // UP
FREE (term)
FREE (n4)

86 → ADD-LEFT_CHILD_TO_NODE (op1.addr, n4.inh_addr) // UP
ADD-RIGHT_CHILD_TO_NODE (op1.addr, term.addr) // UP
n4-1.inh_addr = op1.addr // DOWN
n4.syn_addr = n4-1.syn_addr // UP
FREE (term)
FREE (op1)
FREE (n4-1)

87 → n4.syn_addr = n4.inh_addr // UP
FREE (EPSILON)

88 → term.addr = factor.addr // UP
n5.inh_addr = term.addr // DOWN
term.synaddr = n5.syn_addr // UP
FREE (factor)
FREE (n5)

89 → ADD_LEFT_CHILD_TO_NODE (op2.addr, n5.inh_addr) // UP
ADD-RIGHT_CHILD_TO_NODE (op2.addr, factor.addr) // UP
n5-1.inh_addr = op2.addr // DOWN
n6.syn_addr = n5-1.syn_addr // UP
FREE (factor)
FREE (op2)
FREE (n5-1)

90 → n5.syn_addr = n5.inh_addr // UP
FREE (EPSILON)

91 → factor.addr = arithmeticOrBooleanExpr.syn_addr // UP
FREE (BO)
FREE (BC)
FREE (arithmeticOrBooleanExpr)

92 → factor.addr = &NUM // UP

93 → factor.addr = &RNUM // UP

94 → factor.addr = boolConstt.addr // UP

FREE (boolConstt)

95 →

factor.addr = &ID // UP

factor.index1fArray = n_11.index // UP

FREE (n_11)

96 → n_11.index = element_index_with_expressions.addr // UP

FREE (SCOBO)

FREE (SCBC)

FREE (element_index_with_expressions)

97 → n_11.index = NULL // UP

FREE (EPSILON)

98 → element_index_with_expressions.addr = arrExpr.addr // UP

FREE(arrExpr)

99 → element_index_with_expressions.addr = uarr.addr // UP

FREE (uarr)

100 → uarr.addr = ADD_RIGHT_CHILD_TO_NODE (unary_op.addr , new_NT_addr.addr) // UP

FREE (unary_op)

FREE (new_NT_arr)

101 → new_NT_arr.addr = arrExpr.addr // UP

FREE (BO)

FREE (BC)

FREE (arrExpr)

102 → new_NT_arr.addr = var_id_num.addr // UP

FREE (var_id_num)

103 → arrExpr.syn_addr = arr_N4.syn_addr // UP

arrExpr.addr = arrTerm.addr // UP

arr_N4.inh_addr = arrExpr.addr // DOWN

FREE (arrTerm)

FREE (arr_N4)

104 → ADD_LEFT_CHILD_TO_NODE (op1.addr , aexr_N4.inh_addr) // UP

ADD_RIGHT_CHILD_TO_NODE (op1.addr, aexrTerm.addr) // UP

aexr_N4-1.inh_addr = aexr_N4.addr //DOWN

aexr_N4.syn_addr = aexr_N4-1.syn_addr //UP

FREE (aexrTerm)

FREE (op1)

FREE (aexr_N4-1)

105 → aexr_N4.syn_addr = aexr_N4.inh_addr // UP

FREE (EPSILON)

106 → aexrTerm.syn_addr = aexr_N5.syn_addr // UP

aexrTerm.addr = aexrFactor.addr // UP

aexr_N5.inh_addr = aexrTerm.addr // DOWN

FREE (aexrFactor)

FREE (aexr_N5)

107 → ADD_LEFT_CHILD_TO_NODE (op2.addr, aexr_N5.inh_addr) // UP

ADD_RIGHT_CHILD_TO_NODE (op2.addr, aexrFactor.addr) // UP

aexr_N5-1.inh=addr = aexr_N5.addr // DOWN

aexr_N5.syn_addr = aexr_N5-1.syn_addr // UP

FREE (aexrFactor)

FREE (op2)

FREE (aexr_N5-1)

108 → aexr_N5.syn_addr = aexr_N5.inh_addr // UP

FREE (EPSILON)

109 → aexrFactor.addr = &ID // UP

110 → aexrFactor.addr = &NOM // UP

111 → aexrFactor.addr = boolConst.addr // UP

FREE (boolConst)

112 → arrFactor.addr = arrExpr.addr // UP
FREE(arrExpr)
FREE(BO)
FREE(BC)

113 → op1.addr = &PLUS // UP

114 → op1.addr = &MINUS // UP

115 → op2.addr = &MUL // UP

116 → op2.addr = &DIV // UP

117 → logicalOp.addr = &AND // UP

118 → logicalOp.addr = &OR // UP

119 → relationalOp.addr = &LT // UP

120 → relationalOp.addr = &LE // UP

121 → relationalOp.addr = &GT // UP

122 → relationalOp.addr = &GE // UP

123 → relationalOp.addr = &EQ // UP

124 → relationalOp.addr = &NE // UP

125 → declareStmt.addr = CREATE_NODE("DECLARE", idList.head, dataType.addr) // UP
FREE(DECLARE)
FREE(COLON)
FREE(SEMICOL)
FREE(idList)
FREE(dataType)

126 → conditionalStmt.addr = CREATE_NODE("SWITCH", &ID, caseStmts.head, default_.addr) // UP
FREE(SWITCH)
FREE(BO)
FREE(BC)
FREE(START)
FREE(END)
FREE(caseStmts)
FREE(default_)

127 → INSERT_AT_END(caseStmts.head, CREATE_NODE("CASE", value.addr, statements.head)) // DOWN
n9.head = caseStmts.head // DOWN
FREE(CASE)
FREE(COLON)
FREE(BREAK)
FREE(SEMICOL)
FREE(value)
FREE(statements)
FREE(n9)

128 → INSERT_AT_END(n9.head, CREATE_NODE("CASE", value.addr, statements.head)) // DOWN
n9_1.head = n9.head // DOWN
FREE(CASE)
FREE(COLON)
FREE(BREAK)
FREE(SEMICOL)
FREE(value)
FREE(statements) // FREE(n9_1)

129 → FREE (EPSILON)

130 → default_ . addr = statements . ~~addr~~ head // UP
FREE (DEFAULT)
FREE (COLON)
FREE (BREAK)
FREE (SEMICOL)
FREE (statements)

131 → default_ . addr = NULL // UP
FREE (EPSILON)

132 → value . addr = & NUM // UP

133 → value . addr = & TRUE // UP

134 → value . addr = & FALSE // UP

135 → iterative Stmt . addr = CREATE_NODE ("FOR", & ID, range_for_loop . addr, statements . head) // UP
FREE(FOR)
FREE(IN)
FREE (BO)
FREE (BC)
FREE(END)
FREE(START)
FREE(range_for_loop)
FREE(statements)

136 → iterative Stmt . addr = CREATE_NODE ("WHILE", arithmetic Or BooleanExpr . syn_addr, statements . head) // UP
FREE (WHILE)
FREE (BO)
FREE (BC)
FREE (START)
FREE (END)
FREE (statements)
FREE (arithmetic Or Boolean Expr)

137 → range_for_loop . addr = CREATE_NODE ("RANGE_FOR", index_for_loop . addr, index_for_loop$_{-1}$. addr) // UP
FREE (index_for_loop)
FREE (index_for_loop$_{-1}$)
FREE (RANGEOP)

138 → if (sign . addr != NULL) {
ADD_RIGHT_CHILD_TO_NODE (sign . addr, new_index_for_loop . addr) // UP
index_for_loop . addr = sign . addr // UP
}
else {
index_for_loop . addr = new_index_for_loop . addr // UP
FREE(sign)
}
FREE(new_index_for_loop)

139 → new_index_for_loop . addr = & NUM // UP