SEMANTIC RULES

for

AST CREATION

of

ERPLAG

GROUP NO.: 15

Mithil Shah 2020A7PS0980P

Shivam Pande 2020A7PS0124P

Tanveer Singh 2020A7PS0084P

Kshitij Garg 2020A7PS0120P

Utkarsh Darolia 2020A7PS0981P

Semantic Rules

1. { Post_Order : program.node_syn = create_node("program", moduleDeclarations.list_head_syn, otherModules_1.list_head_syn, driverModule.node_syn, otherModules_2.list_head_syn);
}

2. { Post_Order : moduleDeclarations_1.list_head_syn = push_front(moduleDeclarations_2.list_head_syn, moduleDeclaration.node_syn)
}

3. { Post_Order : moduleDeclarations.list_syn_head = NULL;
}

4. { Post_Order : moduleDeclaration.node_syn = ID.addr;
}

5. { Post_Order : otherModules_1.list_head_syn = push_front(otherModules_2.list_head_syn, module.node_syn);
}

6. { Post_Order : otherModules.list_head_syn = NULL;
}

7. { Post_Order : driverModule.node_syn = create_node("driverModule", moduleDef.node_syn);
}

8. { Post_Order : module.node_syn = create_node("module", ID.addr, input_plist.list_head_syn, ret.node_syn, moduleDef.node_syn)
}

9. { Post_Order : ret.node_syn = create_node("ret", output_plist.list_head_syn);
}

10. { Post_Order : ret.node_syn = create_node("ret", NULL);
}

11. { In_Order : new_list = create_list (dataType.node_syn);
N1.list_head_inh = push_front(new_list, ID.addr);
Post_Order: input_plist.list_head_syn = N1.list_head_syn;
}

12. { Pre_Order : N1_2.list_head_inh = push_back (N1_1.list_head_inh, ID.addr)
Post_Order : N1_1.list_head_syn = N1_2.list_head_syn;
}

13. { Post_Order: N1.list_head_syn = N1.list_head_inh;
}

14. { Pre_Order : new_list = create_list (type.node_syn);
N2.list_head_inh = push_back (new_list, ID.addr);
Post_Order : output_plist.list_head_syn = N2.list_head_syn.
}

15. { Pre_Order : N2_2.list_head_inh = push_back (N2_1.list_head_inh, ID.addr);
Post_Order: N2_1.list_head_syn = N2_2.list_head_syn;
}

16. { Post_Order : N2_list_head_syn = N2.list_head_inh;
}

17. { Post_Order : dataType.node_syn = create_node ("array", type.node_syn,
ranges_array.node_syn);
}

18. { Post_Order : ranges_arrays.node_syn = create_node ("range_array",
index_arr_1.node_syn, index_arr_2.node_syn);
}

19. { Post_Order: dataType.node_syn = INTEGER;
}

20. { Post_Order : dataType.node_syn = REAL;
}

21. { Post-Order : dataType.node-syn = BOOLEAN;
}

22. { Post-Order : type.node-syn = INTEGER;
}

23. { Post-Order : type.node-syn = REAL;
}

24. { Post-Order : type.node-syn = BOOLEAN;
}

25. { Post-Order : moduleDef.node-syn = create ("statements", statements.list-head
}

26. { Post-Order : statements_1.list-head-syn = push-front(
statements_2.list-head-syn, statement.node-syn);
}

27. { Post-Order : statements.list-head-syn = NULL;
}

28. { Post-Order : statement.node-syn = ioStmt.node-syn;
}

29. { Post-Order : statment.node-syn = simpleStmt.node-syn;
}

30. { Post-Order : statement.node-syn = declareStmt.node-syn;
}

31. { Post-Order : statement.node_syn = conditionalStmt.node-syn';

}

32. { Post_ Order : statement.node-syn = iterativeStmt.node-syn';

}

33. { Post_Order : ioStmt.node_syn = Create_node ("GET-VALUE", ID.addr);

}

34. { Post_Order : ioStmt_node-syn = create_node ("PRINT", var_print.node-syn);

}

35. { Post_Order : N14.node-Syn = NUM.addr;

}

36. { Post-Order : N14.node-syn = RNUM.addr;

}

37. { Post-Order : var-print.node-Syn = create_node ("pws_var-print", N14.node-syn);

}

38. { Post-Order : var-print.node-Syn = create_node ("minus-var-print", N14.node-syn);

}

39. { Post_ Order : var-print.node-syn = N14.node-syn;

}

40. { Pre_Order : P1.node_inh = ID.addr;
Post_Order : var-print.node-syn = P1.node-syn;

}

41. { Post-Order : var-print• node-syn = boolconstt.node-syn;
    }

42. { Post-Order : P1.node-syn = create_node ("index-arr", P1.node-inh,
                    index-arr.node-syn);
    }

43. { Post-Order : P1.node-syn = P1.node-inh;
    }

44. { Post-Order : simpleStmt.node-syn = assignmentStmt.node-syn;
    }

45. { Post-Order : simpleStmt.node-syn = moduleReuseStmt.node-syn;
    }

46. { Pre-Order : whichStmt.node-inh = ID.addr;
      Post-Order : assignmentStmt.node-syn = whichstmt.node-syn;
    }

47. { Pre-Order : lvalueIDstmt.node-inh = whichstmt.node-inh;
      Post-Order : whichStmt.node-syn = lvalueIDstmt.node-syn;
    }

48. { Pre-Order : lvalueARRstmt.node-inh = whichStmt.node-inh;
      Post-Order : whichstmt.node-syn = lvalueARRstmt.node-syn;
    }

49. { Post-Order : lvalueIDstmt.node-syn = create-node ("expression",
                    lvalueIDstmt.node-inh, expression.node-syn);
    }

50. { In-Order : expression.node-inh = create-node ("expression",
                  lvalueARRstmt.node-inh, element-index-with-expressions.nodesyn
      Post-Order : lvalueARRstmt.node-syn = create-node ("arr-expression",
                    lvalue ARR stmt.node-inh,
                    element-index-with-expressions.node-syn, expression.node-syn)
    }

51. { Post-Order: index_arr.node_syn = create_node("index_arr",
sign.node_syn, new_index.node_syn);
}

52. { Post-Order: new_index.node_syn = NUM.addr;
}

53. { Post-Order: new_index.node_syn = ID.addr;
}

54. { Post-Order: sign.node_syn = PLUS;
}

55. { Post-Order: sign.node_syn = MINUS;
}

56. { Post-Order: sign.node_syn = NULL;
}

57. { Post-Order: moduleReusestmt.node_syn = create_node("moduleReusestm
ID.addr, optional.list_head_syn,
actual_para_list.list_head_syn);
}

58. { Post_Order: actual_para_list.list_head_syn = push_front(
N12.list_head_syn, singular_para_list.node_syn);
}

59. { Post_Order: N12_1.list_head_syn = push_front(N12_2.list_head_syn,
singular_para_list.node_syn);
}

60. { Post-Order: N12.list_head_syn = create_list("singular_paralist");
}

61. { Post_Order: singular_para_list.list_head_syn = create_node (
"singular_para_list", boolConstt.node_syn);
}

62. { Post_Order: singular_para_list.list_head_syn = create_node(
"singular_para_list", N13.node_syn, PLUS);
}

63. { Post_Order: singular_para_list.node_syn = create_node (
"singular_para_list", N13.node_syn, MINUS);
}

64. { Post_Order: singular_para_list.node_syn = create_node (
"singular_para_list", N13.node_syn, NULL);
}

65. { Post_Order: N13.node_syn = NUM.addr;
}

66. { Post_Order: N13.node_syn = RNUM.addr;
}

67. { Pre_Order: N11.node_inh = ID.addr;
Post_Order:
}

68. { Pre_Order: element_index_with_expressions.node_inh = N11.node_inh;
Post_Order: N11.node_syn = element_index_with_expressions.node_syn
}

69. { Post_Order: N11.node_syn = NULL;
}

70. { Post_Order: optional.list_head_syn = idList.list_head_syn;
}

71. { Post-Order: optional.list_head_syn = NULL;
}

72. { Pre-Order: new_list = create_list(ID.addr);
N3.list_head_inh = new_list;
Post-Order: idList.list_head_syn = N3.list_head_syn;
}

73. { Pre-Order: N3-2.list_head_inh = push_back(N3-1.list_head_inh,
ID.addr);
Post-Order: N3-1.list_head_syn = N3-2.list_head_syn;
}

74. { Post-Order: N3.list_head_syn = N3.list_head_inh;
}

75. { Post-Order: expression.node_syn = arithmeticOrBooleanExp.node_syn;
}

76. { Pre-Order: U.node_inh = expression.node_inh;
Post-Order: expression.node_syn = U.node_syn;
}

77. { Post-Order: U.node_syn = create_node(op1.node_syn, U.node_inh,
new_NT.node_syn);
}

78. { Post-Order: new_NT.node_syn = var_id_num.node_syn;
}

79. { Post-Order: new_NT.node_syn = arithmeticExpr.node_syn;
}

80. { Pre-Order: N7.node_inh = AnyTerm.node_syn;
Post-Order: arithmeticOrBooleanExpr.node_syn = N7.node_syn;
}

81. { In_Order : N7-2.node_inh = create_node (logical Op.node-syn,
N7-1.node_inh, Anyterm.node-syn);
Post-Order : N7-1.node-syn = N7-2.node_syn;
}

82. { Post-order : N7.node_syn = N7.node_inh;
}

83. { Pre_Order : N8.node_inh = arithmeticExpr.node-syn;
Post_Order : Anyterm.node-syn = N8.node-syn;
}

84. { Post_Order : boolConstt.node-syn = TRUE;
}

85. { Post_Order : boolcoonstt.node_syn = FALSE;
}

86. { Post_Order : N8.node-syn = create-node ("N8", relationalOp.node-syn,
N8.node_inh, arithmeticExpr.node-syn);
}

87. { Post-Order : N8.node_syn = N8.node_inh;
}

88. { Pre-Order : N4.node_inh = term.node_Sinh;
Post-Order : arithmeticExpr.node-syn = N4.node_syn;
}

89. { In_Order : N4-2.node_inh = create_node ("N4", opl.node-syn,
N4-1.node_inh, term.node-syn);
Post-Order : N4-1.node-syn = N4-2.node-syn;
}

90. { Post_Order : N4.node-syn = N4.node_inh;
}

91. { Pre_Order : N5.node_inh = factor.node_syn;
Post-Order : term.node-syn = N5.node_syn;
}

92. { In-Order : N5_2.node_inh = create_node ("N4", opl.node_syn,
N5_1.node_inh, term.node_syn);
Post_Order : N5_1.node_syn = N5_2.node-syn;
}

93. { Post_Order : N5.node_syn = N5.node_inh;
}

94. { Post_Order : factor.node_syn = NUM.addr;
}

95. { Post_Order : factor.node_syn = BoolConstt.node_syn;
}

96. { Pre_Order : N11.node_inh = ID.addr;
Post_Order : factor.node_syn = N11.node-syn;
}

97. { Post-Order : factor.node_syn = RNUM.addr;
}

98. { Post_Order : factor.node-syn = boolconstt.node-syn;
}

99. { Pre_Order : element_index_with-expressions.node_inh = ID.addr;
Post-Order : array_element.node_syn = element_index_with_
expressions.node_syn.
}

100. { Pre_Order : arrExpr.node_inh = element-index-with_expression.addr;
Post_Order : element_index_with_expressions.node_syn =
arrExpr.node_syn;
}

101. { Post_Order : element_index_with_expressions.node_syn = create_node
"element_index_with_expressions", PLUS, new_index.node_syn
}

102. { Post_Order : element_index_with_expressions.node_syn = create_node
"element_index_with_expressions", MINUS, new_index.node_syn);
}

103. { Pre_Order : arr_N4.node_inh = arrTerm.node_syn;
Post_Order : arrExpr.node_syn = arr_N4.node.syn;
}

104. { In_Order : arr_N4_2.node_inh = create_node ("arr_N4", op1.node_syn,
arr_N4_1.node_inh, arrTerm.node_syn);
Post_Order : arr_N4_1.node_syn = arr_N4_2.node_syn;
}

105. { Post_Order : arr_N4_node_syn = arr_N4.node_inh;
}

106. { Pre_Order : arr_N5.node_inh = arrFactor.node_syn;
Post_Order : arrTerm.node_syn = arr_N5.node_Syn;
}

107. { In_Order : arr_N52node_inh = create_node ("arr_N4", op2.node_syn,
arr_N5_1.node_inh, arrFactor.node_syn);

Post_Order : arr_N5_1.node_syn = arr_N5_2.node_syn;
}

108. { Post_Order : arr_N5.node_syn = arr_N5.node_inh;
}

109. { Post_Order : arrFactor.node_syn = ID.addr;
}

110. { Post_Order : arrFactor.node_syn = NUM.addr;
}

111. { Post_Order: arrFactor.node-syn = boolConst1.node-syn;
}

112. { Post_Order: arrFactor.node-syn = arrExpr.node-syn;
}

113. { Post_Order: var_id_num.node-syn = ID.addr;
}

114. { Post_Order: var_id_num.node-syn = NUM.addr;
}

115. { Post_Order: var_id_num.node.syn = RNUM.addr;
}

116. { Post_Order: op1.node-syn = PLUS;
}

117. { Post_Order: op1.node-syn = MINUS;
}

118. { Post_Order: op2.node-syn = MUL;
}

119. { Post_Order: op2.node-syn = DIV;
}

120. { Post_Order: logicalop.node-syn = AND;
}

121. { Post-Order: logical Op. node-syn = OR;
}

122. { Post-Order: relational Op. node-syn = LT;
}

123. { Post-Order: relational Op. node-syn = LE;
}

124. { Post-Order: relational op. node-syn = GT;
}

125. { Post-Order: relational Op. node-syn = GE;
}

126. { Post-Order: relational op node-syn = EQ;
}

127. { Post-Order: relational op. node-syn = NE;
}

128. { Post-Order: declarestmt. node-syn = create_node("DECLARE", dataType.
node-syn, idList. list-head-syn);
}

129. { Post-Order: conditional Stmt. node-syn = create_node("conditionalStmt",
ID. addr, case Stmts. list-head-syn, default. node-syn);
}

130. { Post-Order: new-case-statement = create_list("CASE", value. node-syn,
statements. list-head-syn);
caseStmts. list-head-syn = push-front(N9. list-head-syn.
new-case-statement);
}

```
131. { Post_Order: new_case_statement = create_list("CASE", value.node_syn,
                   statements.list_head_syn);
                   N9_1.list_head_syn = push_front(N9_2.list_head_syn,
                   new_case_statement);
     }
132. { Post_Order: N9.node_syn = create_list("statements");
     }
133. { Post_Order: value.node_syn = NUM.addr;
     }
134. { Post_Order: value.node_syn = TRUE;
     }
135. { Post_Order: value.node_syn = FALSE;
     }
136. { Post_Order: default.node_syn = create_node("default", statements.list_head_syn);
     }
137. { Post_Order: default.node_syn = NULL;
     }
138. { Post_Order: iterativeStmt.node_syn = create_node("FOR", ID.addr,
                   range_for_loop.node_syn, statements.list_head_syn);
     }
139. { Post_Order: iterativeStmt.node_syn = create_node("WHILE",
                   arithmeticOrBooleanExp.node_syn, statements.list_head_syn);
     }
140. { Post_Order: range_for_loop.node_syn = create_node("range_for_loop",
                   index_for_loop_1.node_syn, index_for_loop_2.node_syn);
     }
```

141. {Post_Order: index_for_loop.node_syn = create_node(
"index_for_loop", sign_for_loop.node_syn,
new_index_for_loop.node_syn);
}

142. {Post_Order: new_index_for_loop.node_syn = NUM.addr;
}

143. {Post_Order: sign_for_loop.node_syn = PLUS;
}

144. {Post_Order: sign_for_loop.node_syn = MINUS;
}

145. {Post_Order: sign_for_loop.node_syn = NULL;
}

Note: ALL PARSE TREE NODES ON THE RIGHT SIDE OF THE RULES ARE FREED AFTER EXECUTION OF THAT RULE.