Abstract Syntax Tree Rules
Group 33

Team Members:
- Akshit Khanna (2017A7PS0023P)
- Swadesh Vaibhav (2017A7PS0030P)
- Aryan Mehra (2017A7PS0077P)
- Vipin Baswan (2017A7PS0429P)

General options/tips/guidelines:
- We are using L-attributed grammar and AST generation techniques.
- Both inherited and synthetic attributes are used.

General functions and meanings:
- **makenode:** This function should take in the name of the node and the children of the node separated by comma
- **makeleafnode**: This function creates a leaf node
- **makelist**: This function takes one node as a parameter and creates a list with that node
- **concatenate**: This function takes a node and a list as parameters and inserts the node to the list
- **makelistnode**: This function takes a list as a parameter and returns a node with all the elements of the list as its children

---

The Abstract Syntax Tree generation rules for every grammar rule are as follows:

1. <program> ---- <moduleDeclarations> <otherModules> <driverModule> <otherModules>
   - 1.1. program.node = makenode('PROGRAM', moduleDeclarations.node, otherModules.node, driverModule.node, otherModules.node)

2. <moduleDeclarations> ---- <moduleDeclaration><moduleDeclarations>
   - 2.1. moduleDeclarations.node = makenode('MODULEDECLARATIONS', moduleDeclaration.node, moduleDeclarations.node)

3. <moduleDeclarations> ---- ε
   - 3.1. moduleDeclarations.node = NULL

4. <moduleDeclaration> ---- **DECLARE MODULE ID SEMICOL**
   - 4.1. moduleDeclaration.node = makenode('MODULEDEC', ID.node)

5. <otherModules> ---- <module> <otherModules>

5.1. otherModules.node = makenode('OTHERMODULES', module.node, otherModules.node)

6. <otherModules> ---- ***ε***

6.1. otherModule.node = NULL

7. <driverModule> ---- **DRIVERDEF DRIVER PROGRAM DRIVERENDDEF** <moduleDef>

7.1. driverModule.node = makenode('DRIVER', moduleDef.node)

8. <module> ---- **DEF MODULE ID ENDDEF TAKES INPUT SQBO** <input_plist> **SQBC SEMICOL** <ret><moduleDef>

8.1. module.node = makenode('MODULE', ID.node, input_plist.node, ret.node, moduleDef.node)

9. <ret> ---- **RETURNS SQBO** <output_plist> **SQBC SEMICOL**

9.1. ret.node = output_plist.node

10. <ret> ---- ***ε***

10.1. ret.node = NULL

11. <input_plist> ---- **ID COLON** <dataType> <input_plist2>

11.1. input_plist.node = makenode(ID.node, dataType.node, input_plist2.node)

12. <input_plist2> ---- **COMMA ID COLON** <dataType> <input_plist2>

12.1. Input_plist2.node = makenode('INPUT_LIST', ID.node, dataType.node, input_plist2.node)

13. <input_plist2> ---- ***ε***

13.1. Input_plist2.node = NULL

14. <output_plist> ---- **ID COLON** <type> <output_plist2>

14.1. output_plist.node = makenode('OUTPUT_LIST', ID.node, type.node, output_plist2.node)

15. <output_plist2> ---- **COMMA ID COLON** <type> <output_plist2>

15.1. output_plist2.node = makenode('OUTPUT_LIST', ID.node, type.node, output_plist2.node)

16. <output_plist2> ---- ***ε***

16.1. output_plist2.node = NULL

17. <dataType> ---- **INTEGER**

17.1. dataType.node = makeleafnode()
dataType.type = 'INTEGER'

18. <dataType> ---- **REAL**

18.1. dataType.node = makeleafnode()
dataType.type = 'REAL'

19. <dataType> ---- **BOOLEAN**

19.1. dataType.node = makeleafnode()
dataType.type = 'BOOL'

20. <dataType> ---- **ARRAY SQBO** <range1> **SQBC OF** <type>

20.1. dataType.node = makenode('ARRAY', range1.node, type.node)

21. <type> ---- **INTEGER**

21.1. type.node = makeleafnode()
type.node.type = 'INTEGER'

22. <type> ---- **REAL**

22.1. type.node = makeleafnode()
type.node.type = 'REAL'

23. <type> ---- **BOOLEAN**

23.1. type.node = makeleafnode()
type.node.type = 'BOOLEAN'

24. <moduleDef> ---- **START** <statements> **END**

24.1. //make a node here cause it will enable us to store a table with scope info
moduleDef.node = makenode('MODULEDEF', statements.node)

25. <statements> ---- <statement> <statements>

25.1. statements.node = makenode('STATEMENT', statement.node, statements.node)

26. <statements> ---- ***ε***

26.1. statements.node = NULL

27. <statement> ---- <ioStmt>

27.1. statement.node = ioStmt.node

28. <statement> ---- <simpleStmt>

28.1. statement.node = simpleStmt.node

29. <statement> ---- <declareStmt>

29.1. statement.node = declareStmt.node

30. <statement> ---- <conditionalStmt>

30.1. statement.node = conditionalStmt.node

31. <statement> ---- <iterativeStmt>

31.1. statement.node = iterativeStmt.node

32. <ioStmt> ---- **GET_VALUE BO ID BC SEMICOL**

32.1. ioStmt.node = makenode('GET_VAL', ID.node)

33. <ioStmt> ---- **PRINT BO** <varAndBool> **BC SEMICOL**
33.1. ioStmt.node = makenode('PRINT', varAndBool.node)

34. <varAndBool> ---- <var>
34.1. varAndBool.node = var.node

35. <varAndBool> ---- <boolConst>
35.1. varAndBool.node = boolConst.node

36. <var> ---- **ID** <whichId>
36.1. var.node = makenode('ID_ARR', ID.node, whichId.node)

37. <var> ---- **NUM**
37.1. var.node = newleafnode()
var.node.val = NUM.lexval
var.node.type = 'INTEGER'

38. <var> ---- **RNUM**
38.1. var.node = newleafnode()
var.node.val = RNUM.lexval
var.node.type = 'REAL'

39. <whichId> ---- **SQBO** <index> **SQBC**
39.1. whichId.node = index.node

40. <whichId> ---- ***ε***
40.1. whichId.node = NULL

41. <simpleStmt> ---- <assignmentStmt>
41.1. simpleStmt.node = assignStmt.node

42. <simpleStmt> ---- <moduleReuseStmt>
42.1. simpleStmt.node = moduleReuseStmt.node

43. <assignmentStmt> ---- **ID** <whichStmt>
43.1. whichStmt.inh = ID.node
43.2. assignmentStmt.node = whichStmt.node

44. <whichStmt> ---- <lvalueIDStmt>
44.1. lvalueIDStmt.inh = whichStmt.inh
44.2. whichStmt.node = lvalueIDStmt.node
44.3. whichStmt.type = 'ID'

45. <whichStmt> ---- <lvalueARRStmt>

45.1. lvalueARRStmt.inh = whichStmt.inh
45.2. whichStmt.node = lvalueARRStmt.node
45.3. whichStmt.type = 'ARR'

46. <lvalueIDStmt> ---- **ASSIGNOP** <expression> **SEMICOL**
46.1. lvalueIDStmt.node = makenode('ASSIGNOP', lvalueIDStmt.inh ,expression.node)

47. <lvalueARRStmt> ---- **SQBO** <index> **SQBC ASSIGNOP** <expression> **SEMICOL**
47.1. lvalueARRStmt.node = makenode('ASSIGNOPARR', lvalueARRStmt.inh, index.node, expression.node)

48. <index> ---- **NUM**
48.1. index.node = newleafnode()
Index.node.val = NUM.lexval
Index.node.type = 'INTEGER'

49. <index> ---- **ID**
49.1. Index.node = newleafnode()
index.node.node.type = ID.type
index.node.node.val = ID.lexval

50. <moduleReuseStmt> ---- <optional> **USE MODULE ID WITH PARAMETERS** <idList> **SEMICOL**
50.1. temp = makeNode('MODULECALL', ID.node, idList.node)
if(optional.node == NULL)
moduleReuseStmt.node = temp
else if (optional.node ! = NULL)
moduleReuseStmt.node = makenode('MODULEASSIGNOP',optional.node, temp)

51. <optional> ---- **SQBO** <idList> **SQBC ASSIGNOP**
51.1. optional.node = idList.node

52. <optional> ---- ***ε***
//simply shows that module can be evaluated and stored into variable (or not here)
52.1. optional.node = NULL

53. <idList> ---- **ID** <idList2>
// the list builds up in a classical top down manner with inherited and synthesized attributes.
53.1. idList2.inh = makelist(ID.node)
53.2. idList.syn = idList2.syn
53.3. idList.node = makelistnode(idList.syn)

54. <idList2> ---- **COMMA ID** <idList2>
54.1. idList2.inh = concatenate(ID.node, idList2.inh)
54.2. idList2.syn = idList2.syn

55. <idList2> ---- ***ε***

// base case for the list built up

55.1. idList2.syn = idList2.inh

56. <expression> ---- <expression2>

56.1. expression.node = expression2.node

57. <expression> ---- <unaryExprArithmetic>

57.1. expression.node = unaryArithmetic.node

58. <expression2> ---- <logicalExpr> <expression3>

58.1. expression3.inh = logicalExpr.node

58.2. expression2.node = expression3.node

59. <expression2> ---- <boolConst> <expression3>

59.1. expression3.inh = boolConst.node

59.2. expression2.node = expression3.node

60. <expression3> ---- <logicalOp> <expression2>

60.1. expression3.node = makenode(logicalOp.syn, expression3.inh, expression2.node)

61. <expression3> ---- ***ε***

61.1. expression3.node = expresion3.inh

62. <logicalExpr> ---- <arithmeticExpr> <logicalExpr2>

62.1. logicalExpr2.inh = arithmeticExpr.node

62.2. logicalExpr.node = logicalExpr2.node

63. <logicalExpr2> ---- <relationalOp> <arithmeticExpr>

63.1. logicalExpr2.node = makenode(relationalOp.syn, logicalExpr2.inh, arithmeticExpr.node)

64. <logicalExpr2> ---- ***ε***

64.1. logicalExpr2.node = logicalExpr2.inh

65. <arithmeticExpr> ---- <term> <arithmeticExpr2>

65.1. arithmeticExpr2.inh = term.node

65.2. arithmeticExpr.node = arithmeticExpr2.node

66. <arithmeticExpr2> ---- <op1> <arithmeticExpr>

66.1. arithmeticExpr2.node = makenode(op1.syn, arithmeticExpr2.inh, arithmeticExpr.node)

67. <arithmeticExpr2> ---- ***ε***

67.1. arithmeticExpr2.node = arithmeticExpr2.inh

68. <term> ---- <factor> <term2>

68.1. term2.inh = factor.node

68.2. term.node = term2.node

69. <term2> ---- <op2> <term>
69.1. term2.node = makenode(op2.syn, term2.inh, term.node)

70. <term2> ---- ***ε***
70.1. term2.node = term2.inh

71. <factor> ---- **BO** <expression2> **BC**
71.1. factor.node = expression2.node

72. <factor> ---- <var>
72.1. factor.node = var.node

73. <unaryExprArithmetic> ---- <op1> <myOptions>
73.1. unaryExprArithmetic.node = makeNode(op1.syn, myOptions.node)

74. <myOptions> ---- <var>
74.1. myOptions.node = var.node

75. <myOptions> ---- **BO** <arithmeticExprBoolnt> **BC**
75.1. myOptions.node = arithmeticExprBoolnt.node

76. <arithmeticExprBoolnt> ---- <termBoolnt> <arithmeticExpr2Boolnt>
76.1. arithmeticExpr2Boolnt.inh = termBoolnt.node
arithmeticExprBoolnt.node = arithmeticExpr2Boolnt.node

77. <arithmeticExpr2Boolnt> ---- <op1> <arithmeticExprBoolnt>
77.1. arithmeticExpr2Boolnt.node = makenode(op1.syn, arithmeticExpr2Boolnt.inh, arithmeticExprBoolnt.node)

78. <arithmeticExpr2Boolnt> ---- ***ε***
78.1. arithmeticExpr2Boolnt.node = arithmeticExpr2Boolnt.inh

79. <termBoolnt> ---- <factorBoolnt> <term2Boolnt>
79.1. term2Boolnt.inh = factorBoolnt.node
termBoolnt.node = term2Boolnt.node

80. <term2Boolnt> ---- <op2> <factorBoolnt>
80.1. term2Boolnt.node= makenode('op2.syn', term2BooInt.inh, factorBoolnt.node)

81. <term2Boolnt> ---- ***ε***
81.1. term2Boolnt.node = term2BooInt.inh

82. <factorBoolnt> ---- **BO** <arithmeticExprBoolnt> **BC**
82.1. factorBoolnt.node = arithemticExprBoolnt.node

83. <factorBoolnt> ---- <var>
  83.1. factorBoolnt.node = var.node

84. <op1> ---- **PLUS**
  84.1. op1.syn = 'PLUS'

85. <op1> ---- **MINUS**
  85.1. op1.syn = 'MINUS'

86. <op2> ---- **MUL**
  86.1. op2.syn = 'MUL'

87. <op2> ---- **DIV**
  87.1. op2.syn = 'DIV'

88. <relationalOp> ---- **LT**
  88.1. relationalOp.syn = 'LT'

89. <relationalOp> ---- **LE**
  89.1. relationalOp.syn = 'LE'

90. <relationalOp> ---- **GT**
  90.1. relationalOp.syn = 'GT'

91. <relationalOp> ---- **GE**
  91.1. relationalOp.syn = 'GE'

92. <relationalOp> ---- **EQ**
  92.1. relationalOp.syn = 'EQ'

93. <relationalOp> ---- **NE**
  93.1. relationalOp.syn = 'NE'

94. <logicalOp> ---- **AND**
  94.1. logicalOp.syn = 'AND'

95. <logicalOp> ---- **OR**
  95.1. logicalOp.syn = 'OR'

96. <boolConst> ---- **TRUE**
  96.1. boolConst.node = makeleafnode()
  boolConst.node.val = 'TRUE'
  boolConst.node.type = 'BOOL'

97. <boolConst> ---- **FALSE**
  97.1. boolConst.node = makeleafnode()

boolConst.node.val = 'FALSE'
boolConst.node.type = 'BOOL'

98. <declareStmt> ---- **DECLARE** <idList> **COLON** <dataType> **SEMICOL**
98.1. declareStmt.node = makenode('DECLARE', idList.node, dataType.node)

// Insert each element of idList in the symbol table
// Although this action is not strictly a part of the AST creation but rather the AST traversal
while(traversing(idList.syn))
insertInSymbolTable(idList.syn.element , dataType.node.type)

99. <conditionalStmt> ---- **SWITCH BO ID BC START** <caseStmt><default> **END**
99.1. conditionalStmt.node = makenode('SWITCH', ID.node, caseStmt.node, default.node)

100. <caseStmt> ---- **CASE** <value> **COLON** <statements> **BREAK SEMICOL** <caseStmts>
100.1. caseStmt.node = makenode('CASE', value.node, statements.node, caseStmts.node)

101. <caseStmts> ---- **CASE** <value> **COLON** <statements> **BREAK SEMICOL** <caseStmts>
101.1. caseStmts.node = makenode('CASE', value.node, statements.node, caseStmts.node)

102. <caseStmts> ---- ***ε***
102.1. caseStmts.node = NULL

103. <value> ---- **NUM**
103.1. value.node = makeleafnode()
value.node.val = NUM.lexval
value.node.type = 'INTEGER'

104. <value> ---- **TRUE**
104.1. value.node = makeleafnode()
value.node.type = 'BOOL'
value.node.val = 'TRUE'

105. <value> ---- **FALSE**
105.1. value.node = makeleafnode()
value.node.type = 'BOOL'
value.node.val = 'FALSE'

106. <default> ---- **DEFAULT COLON** <statements> **BREAK SEMICOL**
106.1. default.node = makenode('DEFAULT', statements.node)

107. <default> ---- ***ε***
107.1. default.node = NULL

108. <iterativeStmt> ---- **FOR BO ID IN** <range2> **BC START** <statements> **END**
108.1. iterativeStmt.node = makenode('FOR', ID.node, range2.node, statements.node)

109. <iterativeStmt> ---- **WHILE BO** <expression> **BC START** <statements> **END**

109.1. iterativeStmt.node = makenode('WHILE', expression.node, statements.node)

110. <range1> ---- **NUM RANGEOP ID**

110.1. range1.node = makenode('RANGEOP', NUM.node, ID.node)

111. <range1> ---- **ID RANGEOP** <index>

111.1. range1.node = makenode('RANGEOP', ID.node, index.node)

112. <range2> ---- **NUM RANGEOP NUM**

112.1. range2.node = makenode('RANGEOP', NUM.node , NUM.node)