Semantic rules for Abstract Syntax Tree

COMPILER CONSTRUCTION

Batch #75

Aman Agarwal (2014A7PS042P)

Manik Bhandari (2014A7PS088P)

Rule No.	Grammar Rule	Abstract Syntax Tree Formation Rule
1.	<program> → <moduleDeclarations> <otherModules>$_1$ <driverModule> <otherModules>$_2$	<program>.nptr=makenode(<moduleDeclarations>.nptr, <otherModules>$_1$.nptr, <driverModule>.nptr, <otherModules>$_2$.nptr)
2.	<moduleDeclarations> → <moduleDeclaration> <moduleDeclarations>	<moduleDeclarations>$_1$.nptr = makenode(<moduleDeclaration>.nptr, <moduleDeclarations>$_2$.nptr)
3.	<moduleDeclarations> → e	<moduleDeclarations>.nptr = NULL
4.	<moduleDeclaration> → DECLARE MODULE ID SEMICOL	Symbol_Table(ID).type = MODULE <moduleDeclaration>.nptr = makeleaf(ID, Symbol_Table(ID).entry)
5.	<otherModules> → <module> <otherModules>	<otherModules>$_1$.nptr = makenode(<module>.nptr, <otherModules>$_2$.nptr)
6.	<otherModules> → e	<otherModules>.nptr = NULL
7.	<driverModule> → DRIVERDEF DRIVER PROGRAM DRIVERENDDEF <moduleDef>	<driverModule>.nptr = makenode(<moduleDef>.nptr)
8.	<module> → DEF MODULE ID ENDDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret> <moduleDef>	Symbol_Table(ID).type = MODULE ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <module>.nptr = makenode(ID.nptr, <input_plist>.nptr, <ret>.nptr, <moduleDef>.nptr)
9.	<ret> → RETURNS SQBO <output_plist> SQBC SEMICOL	<ret>.nptr = makenode(<output_plist>.nptr)
10.	<ret> → e	<ret>.nptr = NULL
11.	<input_plist> → ID COLON <dataType> <input_plist2>	Symbol_Table(ID).type = <dataType>.type ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <input_plist>.nptr = makenode(ID.nptr, <dataType>.nptr, <input_plist2>.nptr)
12.	<input_plist2>$_1$ → COMMA ID COLON <dataType> <input_plist2>$_2$	Symbol_Table(ID).type = <dataType>.type ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <input_plist2>$_1$.nptr = makenode(ID.nptr, <dataType>.nptr, <input_plist2>$_2$.nptr)
13.	<input_plist2> → e	<input_plist2>.nptr = NULL
14.	<output_plist> → ID COLON <type> <output_plist2>	Symbol_Table(ID).type = <type>.type ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <output_plist>.nptr = makenode(ID.nptr, <type>.nptr, <output_plist2>.nptr)
15.	<output_plist2>$_1$ → COMMA ID COLON <type> <output_plist2>$_2$	Symbol_Table(ID).type = <type>.type ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <output_plist>$_1$.nptr = makenode(ID.nptr, <type>.nptr, <output_plist2>$_2$.nptr)
16.	<output_plist2> → e	<output_list>.nptr = NULL
17.	<dataType> → INTEGER	<dataType>.type = integer
18.	<dataType> → REAL	<dataType>.type = real

19.	<dataType> → BOOLEAN	<dataType>.type = boolean
20.	<dataType> → ARRAY SQBO <range> SQBC OF <type>	<dataType>.nptr = makenode(<range>.nptr, <type>.nptr) <dataType>.type = <type>.type
21.	<type> → INTEGER	<type>.type = integer
22.	<type> → REAL	<type>.type = real
23.	<type> → BOOLEAN	<type>.type = boolean
24.	<moduleDef> → START <statements> END	<moduleDef>.nptr = <statements>.nptr
25.	$<statements_1>$ → <statement> $<statements_2>$	$<statements>_1$.nptr = makenode(<statement>.nptr, $<statements>_2$.nptr)
26.	<statements> → e	<statements>.nptr = NULL
27.	<statement> → <ioStmt>	<statement>.nptr = <ioStmt>.nptr
28.	<statement> → <simpleStmt>	<statement>.nptr = <simpleStmt>.nptr
29.	<statement> → <declareStmt>	<statement>.nptr = <declareStmt>.nptr
30.	<statement> → <condionalStmt>	<statement>.nptr = <condionalStmt>.nptr
31.	<statement> → <iterativeStmt>	<statement>.nptr = <iterativeStmt>.nptr
32.	<ioStmt> → GET_VALUE BO ID BC SEMICOL	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <ioStmt>.nptr = ID.nptr
33.	<ioStmt> → PRINT BO <var> BC SEMICOL	<ioStmt>.nptr = <var>.nptr
34.	<var> → ID <whichId>	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <var>.nptr = makenode(ID.nptr, <whichId>.nptr) <var>.type = ID.type
35.	<var> → NUM	NUM.nptr = makeleaf(NUM, NUM.value) <var>.nptr = NUM.nptr <var>.type = integer
36.	<var> → RNUM	RNUM.nptr = makeleaf(RNUM, RNUM.value) <var>.nptr = RNUM.nptr <var>.type = real
37.	<var> → TRUE	TRUE.nptr = makeleaf(TRUE, ‘TRUE’) <var>.nptr = TRUE.nptr <var>.type = boolean
38.	<var> → FALSE	FALSE.nptr = makeleaf(TRUE, ‘FALSE’) <var>.nptr = FALSE.nptr <var>.type = boolean
39.	<whichId> → SQBO ID SQBC	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <whichId>.nptr = ID.nptr

40.	<whichId> → e	<whichId>.nptr = NULL
41.	<simpleStmt> → <assignmentStmt>	<simpleStmt>.nptr = <assignmentStmt>.nptr
42.	<simpleStmt> → <moduleReuseStmt>	<simpleStmt>.nptr = <moduleReuseStmt>.nptr
43.	<assignmentStmt> → ID <whichStmt>	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <assignmentStmt>.nptr = makenode(ID.nptr, <whichStmt>.nptr) if(<whichStmt>.type != ID.type) => ERROR
44.	<whichStmt> → <lvalueIDStmt>	<whichStmt>.nptr = <lvalueIDStmt>.nptr <whichStmt>.type = <lvalueIDStmt>.type
45.	<whichStmt> → <lvalueARRStmt>	<whichStmt>.nptr = <lvalueARRStmt>.nptr <whichStmt>.type = <lvalueARRStmt>.type
46.	<lvalueIDStmt> → ASSIGNOP <expression> SEMICOL	<lvalueIDStmt>.nptr = <expression>.nptr <expression>.type = <lvalueIDStmt>.type
47.	<lvalueARRStmt> → SQBO <index> SQBC ASSIGNOP <expression> SEMICOL	<lvalueARRStmt>.nptr = makenode(<index>.nptr, <expression>.nptr) <expression>.type = <lvalueARRStmt>.type
48.	<index> → NUM	NUM.nptr = makeleaf(NUM, NUM.value) <index>.nptr = NUM.nptr <index>.type = integer
49.	<index> → ID	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <index>.nptr = ID.nptr <index>.type = ID.type
50.	<moduleReuseStmt> → <optional> USE MODULE ID WITH PARAMETERS <idList> SEMICOL	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <moduleReuseStmt>.nptr = makenode(<optional>.nptr, ID.nptr, <idList>.nptr)
51.	<optional> → SQBO <idList> SQBC ASSIGNOP	<optional>.nptr = <idList>.nptr
52.	<optional> → e	<optional>.nptr = NULL
53.	<idList> → ID <idList2>	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <idList>.nptr = makenode(ID.nptr, <idList2>.nptr)
54.	<idList2>$_1$ → COMMA ID <idList2>$_2$	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <idList2> $_1$.nptr = makenode(ID.nptr, <idList2> $_2$.nptr)
55.	<idList2> → e	<idList2>.nptr = NULL
56.	<expression> → <arithmeticOrBooleanExpr>	<expression>.nptr = <arithmeticOrBooleanExpr>.nptr if(<expression>.type != <arithmeticOrBooleanExpr>.type) => ERROR
57.	<expression> → MINUS <booleanOrNonBooleanArithmeticExpr>	<expression>.nptr = <booleanOrNonBooleanArithmeticExpr>.nptr if(<expression>.type == boolean || <expression>.type != <booleanOrNonBooleanArithmeticExpr>.type) => ERROR
58.	<booleanOrNonBooleanArithmeticExpr> → BO	<booleanOrNonBooleanArithmeticExpr>.nptr = <arithmeticExpr>.nptr

	<arithmeticExpr> BC	<booleanOrNonBooleanArithmeticExpr>.type = <arithmeticExpr>.type
59.	<booleanOrNonBooleanArithmeticExpr> → <arithmeticExpr>	<booleanOrNonBooleanArithmeticExpr>.nptr = <arithmeticExpr>.nptr <booleanOrNonBooleanArithmeticExpr>.type = <arithmeticExpr>.type

60.	<arithmeticOrBooleanExpr> → <anyTerm> <expressionWithLogOp>	<arithmeticOrBooleanExpr>.nptr = makenode(<anyTerm>.nptr, <expressionWithLogOp>.nptr)
		
		if(<expressionWithLogOp>.nptr != NULL)
		  <arithmeticOrBooleanExpr>.type = boolean
		else
		  <arithmeticOrBooleanExpr>.type = <anyTerm>.type

61.	<arithmeticOrBooleanExpr>$_1$ → BO <arithmeticOrBooleanExpr>$_2$ BC <arithmeticOrBooleanExpr2>	<arithmeticOrBooleanExpr>$_1$.nptr = makenode(<arithmeticOrBooleanExpr>$_2$.nptr,<arithmeticOrBooleanExpr2>.nptr) <arithmeticOrBooleanExpr>$_1$.type = <arithmeticOrBooleanExpr>$_2$.type
62.	<arithmeticOrBooleanExpr2> → <logicalOp> <arithmeticOrBooleanExpr>	<arithmeticOrBooleanExpr2>.nptr = makenode(<logicalOp>.nptr,<arithmeticOrBooleanExpr>.nptr)
63.	<arithmeticOrBooleanExpr2> → <relationalOp> <arithmeticOrBooleanExpr>	<arithmeticOrBooleanExpr2>.nptr = makenode(<relationalOp>.nptr,<arithmeticOrBooleanExpr>.nptr)
64.	<arithmeticOrBooleanExpr2> → <op1> <arithmeticOrBooleanExpr>	<arithmeticOrBooleanExpr2>.nptr = makenode(<op1>.nptr,<arithmeticOrBooleanExpr>.nptr)
65.	<arithmeticOrBooleanExpr2> → <op2> <arithmeticOrBooleanExpr>	<arithmeticOrBooleanExpr2>.nptr = makenode(<op2>.nptr,<arithmeticOrBooleanExpr>.nptr)
66.	<arithmeticOrBooleanExpr2> → e	<arithmeticOrBooleanExpr2>.nptr = NULL
67.	<expressionWithLogOp> → <logicalOp> <anyTerm2> <expressionWithLogOp>	<expressionWithLogOp>$_1$.nptr = makenode(<logicalOp>.nptr, <anyTerm2>.nptr, <expressionWithLogOp>$_2$.nptr) if(<anyTerm2>.type != boolean) => ERROR
68.	<expressionWithLogOp> → e	<expressionWithLogOp>.nptr = NULL

69.	<anyTerm> → <arithmeticExpr> <expressionWithRelOp>	<anyTerm>.nptr = makenode(<arithmeticExpr>.nptr, <expressionWithRelOp>.nptr)
		
		if(<expressionWithRelOp>.nptr != NULL)
		  <anyTerm>.type = boolean
		else
		  <anyTerm>.type = <arithmeticExpr>.type

70.	<anyTerm2> → BO <arithmeticOrBooleanExp> BC	<anyTerm2>.nptr= <arithmeticOrBooleanExp>.nptr <anyTerm2>.type= <arithmeticOrBooleanExp>.type
71.	<anyTerm2> → <anyTerm>	<anyTerm2>.nptr= <anyTerm>.nptr <anyTerm2>.type= <anyTerm>.type
72.	<expressionWithRelOp> → <relationalOp> <negOrPosArithmeticExpr> <expressionWithRelOp>	<expressionWithRelOp>$_1$.nptr = makenode(<relationalOp>.nptr, <negOrPosArithmeticExpr>.nptr, <expressionWithRelOp>$_2$.nptr) if(<negOrPosArithmeticExpr>.type != boolean) => ERROR
73.	<expressionWithRelOp> → e	<expressionWithRelOp>.nptr = NULL

74.	<negOrPosArithmeticExpr> → MINUS <booleanOrNonBooleanArithmeticExpr>	<negOrPosArithmeticExpr>.nptr = <booleanOrNonBooleanArithmeticExpr>.nptr
		
		if(<booleanOrNonBooleanArithmeticExpr>.type == boolean) => ERROR
		else
		  <negOrPosArithmeticExpr>.type = <booleanOrNonBooleanArithmeticExpr>.type

75.	<negOrPosArithmeticExpr> → <booleanOrNonBooleanArithmeticExpr>	<negOrPosArithmeticExpr>.nptr = <booleanOrNonBooleanArithmeticExpr>.nptr <negOrPosArithmeticExpr>.type = <booleanOrNonBooleanArithmeticExpr>.type
76.	<arithmeticExpr> → <term> <arithmeticExpr2>	<arithmeticExpr>.nptr = makenode(<term>.nptr, <arithmeticExpr2>.nptr) <arithmeticExpr>.type = <term>.type
77.	<arithmeticExpr2>$_1$ → <op1> <term> <arithmeticExpr2>$_2$	<arithmeticExpr2>$_1$.nptr = makenode(<op1>.nptr, <term>.nptr, <arithmeticExpr2>$_2$.nptr)
78.	<arithmeticExpr2> → e	<arithmeticExpr2>.nptr = NULL
79.	<term> → <factor> <term2>	<term2>.nptr = makenode(<factor>.nptr, <term2>.nptr) <term>.type = <factor>.type
80.	<term2>$_1$ → <op2> <factor> <term2>$_2$	<term2>$_1$.nptr = makenode(<op2>.nptr, <factor>.nptr, <term2>$_2$.nptr)
81.	<term2> → e	<term2>.nptr = NULL
82.	<factor> → <var>	<factor>.nptr = <var>.nptr <factor>.type = <var>.type
83.	<op1> → PLUS	PLUS.nptr = makeleaf(PLUS, ‘+’) <op1>.nptr = PLUS.nptr
84.	<op1> → MINUS	MINUS.nptr = makeleaf(MINUS, ‘-’) <op1>.nptr = MINUS.nptr
85.	<op2> → MUL	MUL.nptr = makeleaf(MUL, ‘*’) <op2>.nptr = MUL.nptr
86.	<op2> → DIV	DIV.nptr = makeleaf(DIV, ‘/’) <op2>.nptr = DIV.nptr
87.	<logicalOp> → AND	AND.nptr = makeleaf(AND, ‘AND’) <logicalOp>.nptr = AND.nptr
88.	<logicalOp> → OR	OR.nptr = makeleaf(OR, ‘OR’) <logicalOp>.nptr = OR.nptr
89.	<relationalOp> → LT	LT.nptr = makeleaf(LT, ‘LT’) <relationalOp>.nptr = LT.nptr
90.	<relationalOp> → LE	LE.nptr = makeleaf(LE, ‘LE’) <relationalOp>.nptr = LE.nptr
91.	<relationalOp> → GT	GT.nptr = makeleaf(GT, ‘GT’) <relationalOp>.nptr = GT.nptr
92.	<relationalOp> → GE	GE.nptr = makeleaf(GE, ‘GE’) <relationalOp>.nptr = GE.nptr

93.	<relationalOp> → EQ	EQ.nptr = makeleaf(EQ, 'EQ') <relationalOp>.nptr = EQ.nptr
94.	<relationalOp> → NE	NE.nptr = makeleaf(NE, 'NE') <relationalOp>.nptr = NE.nptr
95.	<declareStmt> → DECLARE <idList> COLON <dataType> SEMICOL	<declareStmt>.nptr = makenode(<idList>.nptr, <dataType>.nptr) <idList>.type = <dataType>.type
96.	<condionalStmt> → SWITCH BO ID BC START <caseStmts> <default> END	<condionalStmt>.nptr = makenode(<caseStmts>.nptr, <default>.nptr)
97.	<caseStmts> → CASE <value> COLON <statements> BREAK SEMICOL <caseStmt>	<caseStmts>.nptr = makenode(<value>.nptr, <statements>.nptr, <caseStmt>.nptr)
98.	$<caseStmt>_1$ → CASE <value> COLON <statements> BREAK SEMICOL $<caseStmt>_2$	$<caseStmt>_1$.nptr = makenode(<value>.nptr, <statements>.nptr, $<caseStmt>_2$.nptr)
99.	<caseStmt> → e	<caseStmt>.nptr = NULL
100.	<value> → NUM	NUM.nptr = makeleaf(NUM, NUM.value) <index>.nptr = NUM.nptr <var>.type = Integer
101.	<value> → TRUE	TRUE.nptr = makeleaf(NUM, 'TRUE') <index>.nptr = TRUE.nptr <var>.type = Boolean
102.	<value> → FALSE	FALSE.nptr = makeleaf(FALSE, 'FALSE') <index>.nptr = FALSE.nptr <var>.type = Boolean
103.	<default1> → DEFAULT COLON <statements> BREAK SEMICOL	<default1>.nptr = <statements>.nptr
104.	<default1> → e	<default1>.nptr = NULL
105.	<iterativeStmt> → FOR BO ID IN <range> BC START <statements> END	ID.nptr = makeleaf(ID,Symbol_Table(ID).entry) <iterativeStmt>.nptr = makenode(ID.nptr, <range>.nptr, <statements>.nptr)
106.	<iterativeStmt> → WHILE BO <arithmeticOrBooleanExpr> BC START <statements> END	<iterativeStmt>.nptr = makenode(<arithmeticOrBooleanExpr>.nptr, <statements>.nptr) If (<arithmeticOrBooleanExpr>.type != Boolean) => ERROR
107.	<range> → NUM_1 RANGEOP NUM_2	NUM_1.nptr = makeleaf(NUM_1, NUM_1.value) NUM_2.nptr = makeleaf(NUM_2, NUM_2.value) <range>.nptr = makenode(NUM_1.nptr, <NUM_2>.nptr)