Grammar - Compiler Construction

Group - 29
Shaurya Marwah - 2019B3A70794P
Ruchir Kumbhare - 2019B5A70650P
Ashwin Murali - 2019B2A70957P
Hari Sankar - 2019B3A70564P
Dilip Venkatesh - 2020A7PS1203P

1.

<program'> → <program> $

First(<program'>) = {DRIVERDEF, DEF, DECLARE}

2.

~~<moduleDeclarations> →~~

<program> → <moduleDeclarations> <otherModules> <driverModule> <otherModules>

First(<program>) = {DRIVERDEF, DEF, DECLARE}

3.

<moduleDeclarations> → <moduleDeclaration> <moduleDeclarations> | ∈

First(<moduleDeclarations>) = {DECLARE, ∈}

Follow(<moduleDeclarations>) = {DEF, DRIVERDEF}

4.

<moduleDeclaration> → DECLARE MODULE ID SEMICOL

First(<moduleDeclaration>) = {DECLARE}

~~Follow(<moduleDeclaration>) = {}~~

5.

<otherModules> → <module> <otherModules> | ∈

First(<otherModules>) = {DEF, ∈}

Follow(<otherModules>) = {DRIVERDEF}

6.

<driverModule> → DRIVERDEF DRIVERPROGRAM DRIVERENDDEF <moduleDef>

First(<driverModule>) = {DRIVERDEF}

7.
<module> → DEF MODULE ID ENDEF TAKES INPUT SQBO <input_plist> SQBC SEMICOL <ret> <moduleDef>
First(<module>) = {DEF}

8.
<input_plist> → ID COLON <dataType> <new_input_plist>
First(<input_plist>) = {ID}

9.
<new_input_plist> → COMMA ID COLON <dataType> <new_output_plist> | ϵ
First(<new_input_plist>) = {COMMA, ϵ}
Follow(<new_input_plist>) = {SQBC}

10.
<output_plist> → ID COLON <dataType> <new output_plist> | ϵ
First(<output_plist>) = {ID}

11.
<new_output_plist> → COMMA ID COLON <dataType> <new_output_plist> | ϵ
First(<new_output_plist>) = {COMMA, ϵ}
Follow(<new_output_plist>) = {SQBC}

12.
<dataType> → INTEGER | REAL | BOOLEAN | ARRAY SQBO <range> SQBC OF <type>
First(<dataType>) = {INTEGER, REAL, BOOLEAN, ARRAY}

13.

<type> → INTEGER | REAL | BOOLEAN

First (<type>) = { INTEGER, REAL, BOOLEAN }

14.

<moduleDef> → START <statements> END

First (<moduleDef>) = { START }

15.

<statements> → <statement> <statements> | ∈

First (<statements>) = { GET_VALUE, PRINT, ID, SQBO, USE, DECLARE, FOR, WHILE, SWITCH, ∈ }

FOLLOW (<statements>) = { BREAK, END }

16.

<statement> → <ioStmt> | <simpleStmt> | <declareStmt> | <conditionalStmt> | <iterativeStmt>

First (<statement>) = { GET_VALUE, PRINT, ID, SQBO, USE, DECLARE, FOR, WHILE, SWITCH }

17.

<ioStmt> → GET_VALUE BO ID BC SEMICOL | PRINT BO <var/AllBool> BC SEMICOL

First (<ioStmt>) = { GET_VALUE, PRINT }

18

<var> → ID <whichId> | NUM | RNUM

First (<var>) = { ID, NUM, RNUM }

19.

<bool> → TRUE | FALSE

First (<bool>) = { TRUE, FALSE }

20.
<var AndBool> → <var> | <bool>

First(<var AndBool>) = { ID, NUM, RNUM, TRUE, FALSE }

21.
<whichId> → SQBO ID SQBC | ∈

First(<whichId>) = { SQBO, ∈ }

Follow(<whichId>) = { BC }

22.
<simpleStmt> → <assignmentStmt> | <moduleReuseStmt>

First(<simpleStmt>) = { ID, SQBO, USE }

23.
<assignmentStmt> → ID <whichStmt>

First(<assignmentStmt>) = { ID }

24.
<whichStmt> → <lvalueIDStmt> | <lvalueARRStmt>

First(<whichStmt>) = { ASSIGNOP, SQBO }

25.
<lvalueIdStmt> → ASSIGNOP <expression> SEMICOL

First(<lvalueIdStatement>) = { ASSIGNOP }

26.
<lvalueARRStmt> → SQBO <index> SQBC ASSIGNOP <expression> SEMICOL

First(<lvalueARRStmt>) = { SQBC }

27.
<index> → NUM | ID

First(<index>) = { NUM, ID }

28.

< moduleReuseStmt > → < optional > USE MODULE ID WITH PARAMETERS < idList > SEMICOL

First (< moduleReuseStmt >) = { SQBO, USE }

29.

< optional > → SQBO < idList > SQBC ASSIGNOPLE

First (< optional >) = { SQBO, ϵ }

Follow (< optional >) = { USE }

30.

< idList > → ID < idList' >

First (< idList >) = { ID }

31.

< idList' > → COMMA ID < idList' > | ϵ

First (< idList' >) = { COMMA, ϵ }

Follow (< idList' >) = { SEMICOL, SQBC, COLON }

32.

< expression > → PLUS < expression' > | MINUS < expression' > | < boolConjunctArithmetic >

First (< expression >) → { PLUS, MINUS, BO, ID, RNUM, NUM }

33.

< expression' > → < var > | BO < arithmetExpr > BC

First (< expression' >) = { BO, ID, NUM, RNUM }

< factor' > → < bool >

First (< factor' >) = { BO, ID, NUM, RNUM }

34.

< boolConjunctArithmetic > → < factor' > < boolConjunctArithmetic' >

first (< boolConjunctArithmetic >) = { BO, ID, RNUM, NUM }

35.

<boolConjunctArithmetic'> → <logicalOp><factor'><boolConjunctArithmetic'>

First(<boolConjunctArithmetic'>) = {E, AND, OR}

Follow(<boolConjunctArithmetic'>) = {SEMICOL

36.

<factor'> → <ArithmeticExpr><factor''> | <bool><factor''> | <bool>

First(<factor'>) = {BO, ID, NUM, RNUM}

37.

<factor''> → <relationalOp><arithmeticExpr><factor''> | ∈

First(<factor''>) = {LT, LE, GT, GE, EQ, NE, ∈}

Follow(<factor''>) = {SEMICOL}

38.

<arithmeticExpr> → BO <boolConjunctArithmetic> BC | <term1><arithmeticExpr'>

First(<arithmeticExpr>) = {BO, ID, NUM, RNUM}

39.

<arithmeticExpr'> → <op1><term1><arithmeticExpr'> | ∈

First(<arithmeticExpr'>) = {PLUS, MINUS, ∈}

Follow(<arithmeticExpr'>) = {~~COMMA~~ LT, LE, GT, GE, EQ, NE, AND, OR, SEMICOL, BC}

40.

<term1> → <op2><term1'>

first(<term1>) = {BO, ID, NUM, RNUM}

41.

<term1'> → <op1'><op2><term1'> | ∈

first(<term1'>) = {MUL, DIV, ∈}

follow(<term1'>) = {PLUS, MINUS, COMMA, LT, LE, GT, GE, EQ, NE, AND, OR, SEMICOL, BC}

42.
<op1> → PLUS | MINUS

First(<op1>) = {PLUS, MINUS}

43.
<op21'> → MUL | DIV

First(<op21'>) = {MUL, DIV}

44.
<op2> → BO <arithmeticExpr> BC | ID | NUM | RNUM

First(<op2>) = {BO, ID, NUM, RNUM}

45.
<logicalOp> → AND | OR

First(<logicalOp>) → {AND, OR}

46.
<relationalOp> → LT | LE | GT | GE | EQ | NE

First(<relationalOp>) = {LT, LE, GT, GE, EQ, NE}

47.
<declareStmt> → DECLARE <idList> COLON <dataType> SEMICOL

first(<declareStmt>) = {DECLARE}

48.
<conditionalStmt> → SWITCH BO ID BC START <caseStmt> <default> END

first(<conditionalStmt>) = {SWITCH}

49.
<caseStmt> → CASE <value> COLON <statements> BREAK SEMICOL
<caseStmt'>

First(<caseStmt>) = {CASE}

50.
<caseStmt'> → <caseStmt> | ϵ

First(<caseStmt'>) = {CASE, ϵ}

Follow(<caseStmt'>) = {DEFAULT, END}

51.
<value> → NUM | TRUE | FALSE

First(<value>) = { NUM, TRUE, FALSE }

52.
<default> → DEFAULT COLON <elements> BREAK SEMICOL | ε

first(<default>) = { DEFAULT, ε }

follow(<default>) = { END }

53.
<iterativeStmt> → FOR BO ID IN NUM RANGEOP NUM BC START <elements>
END | WHILE BO <boolExpr> BC START
<elements> END

first(<iterativeStmt>) = { FOR, WHILE }

54.
<range> → ID RANGEOP <range'> | NUM RANGEOP <range'>

first(<range>) = { ID, NUM }

55.
<range'> → ID | NUM

first(<range'>) = { ID, NUM }

Note:

1. Precedence order considered for expressions - Arithmetic > Relational > logical. We have accordingly introduced new non-terminals to incorporate this precedence

2. The points that have been taken into ~~cond~~ consideration for modifying the grammar are as follows -
   a. Left Recursion
   b. Left Factoring
   c. Precedence
   d. First Sets
   e. Follow sets
   f. LL1 compatibility

3. Since a boolean expression can include an arithmetic expression, we have introduced a new non-terminal boolconjunct arithmetic to resolve left factoring in the following rule

   < expression > → < arithmeticExpr > / @ < booleanExpr >

4. The arithmetic operators '*' and '/' have more precedence over '+' and '-' have been incorporated using op1 and op1' as the new non-terminals

5. In the original grammar, case statement non terminal did not have an ending condition. We have introduced a new non-terminal ~~caseStmnt~~ caseStatement' so that infinite no. of case statements are not generated along with preserving the condition that atleast one case statement must exist.

6. Since a range statement can derive expressions with identifier, as well we have accordingly modified the rules for range non-terminal and remove left factoring

7. All the rules that have been mentioned in the mails as well as language specifications report have been thoroughly studied and incorporated