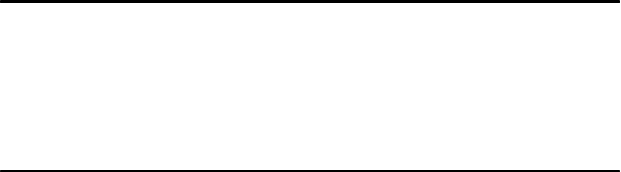


**Compiler Project 2024**

**SLR Parser**





|  |  |
| --- | --- |
| Subject | 컴파일러 03분반 |
| Team Number | Team 02 |
| Team Members | 20202721 신현우 20212735 이진경 |

Content

[1, Non-ambiguous CFG 3](#_Toc168240114)

[2. SLR parsing table 4](#_Toc168240115)

[3. Change in the CFG 5](#_Toc168240116)

[4. How syntax analyzer works 6](#_Toc168240117)

1. Non-ambiguous CFG

00 : CODE’ → CODE

01 : CODE → VDECL CODE | FDECL CODE | ε

02 : VDECL → vtype id semi | vtype ASSIGN semi

03 : ASSIGN → id assign RHS

04 : RHS → EXPR | literal | character | boolstr

05 : EXPR → EXPR addsub TERM | TERM

06 : TERM → TERM multdiv FACTOR | FACTOR

07 : FACTOR → lparen EXPR rparen | id | num

08 : FDECL → vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace

09 : ARG → vtype id MOREARGS | ε

10 : MOREARGS → comma vtype id MOREARGS | ε

11 : BLOCK → STMT BLOCK | ε

12 : STMT → VDECL | ASSIGN semi

13 : STMT → if lparen COND rparen lbrace BLOCK rbrace ELSE

14 : STMT → while lparen COND rparen lbrace BLOCK rbrace

15 : COND → COND comp PRED | boolstr

16 : PRED → boolstr

17 : ELSE → else lbrace BLOCK rbrace | ε

18 : RETURN → return RHS semi

# 2. SLR parsing table

The size of the parsing table was so large that I saved it as a separate Excel file

rather than including it in the document.

# 3. Change in the CFG

The existing CFG was ambiguous because in productions 5 and 14, the non-terminal on the RHS repeatedly appeared on the LHS. Therefore, by modifying this grammar, it could be transformed into a non-ambiguous CFG. The method we employed involved introducing new non-terminals to eliminate ambiguity.

[1]

<Before>

05 : EXPR → EXPR addsub EXPR | EXPR multidiv EXPR

06 : EXPR → lparen EXPR rparen | id | num

We create non-terminal TERM and FACTOR

<After>

05 : EXPR → EXPR addsub TERM | TERM

06 : TERM → TERM multdiv FACTOR | FACTOR

07 : FACTOR → lparen EXPR rparen | id | num

[2]

<Before>

14 : COND → COND comp COND | boolstr

We create a non-terminal PRED

<After>

15 : COND → COND comp PRED | boolstr

16 : PRED -> boolstr

# 4. How syntax analyzer works

Overall procedures for input token

: vtype id semi vtype id lparen vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

For

| vtype id semi vtype id lparen vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 0

Input Symbol : vtype

Decision : Shift & Goto 5

Stack : 0 5

For

vtype | id semi vtype id lparen vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 5

Input Symbol : id

Decision : Shift & Goto 8

Stack : 0 5 8

For

vtype id | semi vtype id lparen vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 7

Input Symbol : semi

Decision : Shift & Goto 10

Stack : 0 5 8 10

For

vtype id semi | vtype id lparen vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 10

Input Symbol : vtype

Decision : Reduce by (4) VDECL -> vtype id semi & GOTO 2

Stack : 0 2

For

VDECL | vtype id lparen vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 2

Input Symbol : vtype

Decision : Shift & Goto 5

Stack : 0 2 5

For

VDECL vtype | id lparen vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 5

Input Symbol : id

Decision : Shift & Goto 8

Stack : 0 2 5 8

For

VDECL vtype id | lparen vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 8

Input Symbol : lparen

Decision : Shift & Goto 11

Stack : 0 2 5 8 11

For

VDECL vtype id lparen | vtype id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 11

Input Symbol : vtype

Decision : Shift & Goto 15

Stack : 0 2 5 8 11 15

For

VDECL vtype id lparen vtype | id ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 15

Input Symbol : id

Decision : Shift & Goto 28

Stack : 0 2 5 8 11 15 28

For

VDECL vtype id lparen vtype id | ϵ rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 28

Input Symbol : ϵ

Decision : Shift & Goto 35

Stack : 0 2 5 8 11 15 28 35

For

VDECL vtype id lparen vtype id ϵ | rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 39

Input Symbol : rparen

Decision : Reduce by (22) MOREARGS -> ϵ & Goto 33

Stack : 0 2 5 8 11 15 28 33

For

VDECL vtype id lparen vtype id MOREARGS | rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 33

Input Symbol : rparen

Decision : Reduce by (19) ARG -> vtype id MOREARGS & Goto 14

Stack : 0 2 5 8 11 14

For

VDECL vtype id lparen ARG | rparen lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 14

Input Symbol : rparen

Decision : Shift & Goto 27

Stack : 0 2 5 8 11 14 27

For

VDECL vtype id lparen ARG rparen | lbrace ϵ return boolstr semi rbrace ϵ $

Start State : 27

Input Symbol : lbrace

Decision : Shift & Goto 32

Stack : 0 2 5 8 11 14 27 32

For

VDECL vtype id lparen ARG rparen lbrace | ϵ return boolstr semi rbrace ϵ $

Start State : 32

Input Symbol : ϵ

Decision : Shift & Goto 41

Stack : 0 2 5 8 11 14 27 32 41

For

VDECL vtype id lparen ARG rparen lbrace ϵ | return boolstr semi rbrace ϵ $

Start State : 41

Input Symbol : return

Decision : Reduce by (24) BLOCK -> ϵ & Goto 39

Stack : 0 2 5 8 11 14 27 32 39

For

VDECL vtype id lparen ARG rparen lbrace BLOCK | return boolstr semi rbrace ϵ $

Start State : 39

Input Symbol : return

Decision : Shift & Goto 50

Stack : 0 2 5 8 11 14 27 32 39 50

For

VDECL vtype id lparen ARG rparen lbrace BLOCK return | boolstr semi rbrace ϵ $

Start State : 50

Input Symbol : boolstr

Decision : Shift & Goto 21

Stack : 0 2 5 8 11 14 27 32 39 50 21

For

VDECL vtype id lparen ARG rparen lbrace BLOCK return boolstr | semi rbrace ϵ $

Start State : 21

Input Symbol : semi

Decision : Reduce by (10) RHS -> boolstr & Goto 58

Stack : 0 2 5 8 11 14 27 32 39 50 58

For

VDECL vtype id lparen ARG rparen lbrace BLOCK return RHS | semi rbrace ϵ $

Start State : 58

Input Symbol : semi

Decision : Shift & Goto 62

Stack : 0 2 5 8 11 14 27 32 39 50 58 62

For

VDECL vtype id lparen ARG rparen lbrace BLOCK return RHS semi | rbrace ϵ $

Start State : 62

Input Symbol : rbrace

Decision : Reduce by (33) RETURN -> return RHS semi & Goto 49

Stack : 0 2 5 8 11 14 27 32 39 49

For

VDECL vtype id lparen ARG rparen lbrace BLOCK RETURN | rbrace ϵ $

Start State : 49

Input Symbol : rbrace

Decision : Shift & Goto 57

Stack : 0 2 5 8 11 14 27 32 39 49 57

For

VDECL vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace | ϵ $

Start State : 56

Input Symbol : ϵ

Decision : Reduce by (18) FDECL -> vtype id lparen ARG rparen lbrace BLOCK RETURN rbrace & Goto 3

Stack : 0 2 3

For

VDECL FDECL | ϵ $

Start State : 3

Input Symbol : ϵ

Decision : Shift & Goto 4

Stack : 0 2 3 4

For

VDECL FDECL ϵ | $

Start State : 4

Input Symbol : $

Decision : Reduce by (3) CODE -> ϵ & Goto 7

Stack : 0 2 3 7

For

VDECL FDECL CODE | $

Start State : 7

Input Symbol : $

Decision : Reduce by (2) CODE -> FDECL CODE & Goto 6

Stack : 0 2 6

For

VDECL CODE | $

Start State : 6

Input Symbol : $

Decision : Reduce by (1) CODE -> VDECL CODE & Goto 1

Stack : 0 1

For

CODE | $

Start State : 1

Input Symbol : $

Decision : accept

Data Structures & Conversion

1. Parsing Table and CFG Conversion:

- The parsing table was converted into an Excel file.

- Tokens and actions were sequentially stored in a list as key-value pairs.

- The converted Context-Free Grammar (CFG) was stored using a dictionary like {num : production}.

- The parsing tree utilized the 'anytree' data structure from the Python library.

2. Input Text Handling:

- Input text was received as a file, saved into a list, and passed to right\_sub\_string.

- right\_sub\_string was designed to pop input tokens from the front and pass them to left\_sub\_string using the deque data structure.

- An object of the SLRParser class was created, and parsing began by executing the parsing method of the object.

3. Parsing Method:

- The parsing method repeats reading right\_sub\_string until it is read to the end.

- It starts by inserting the initial state [0] into the stack.

- The first token from right\_sub\_string is received, and the current state is set as the end state of the stack.

Detailed Parsing Process

1. Token and State Handling:

- If the input symbol received during parsing is not included in the key value of current\_state in the parsing\_table, it immediately returns "reject".

- If the token matches the key value in the parsing table, the value of that key (i.e., the action) is divided into the action and the new state.

2. Shift Action:

- If the action is shift, the value in right\_sub\_string is popped and stored in left\_sub\_string.

- The new state is added to state\_stack, and a node is added to the parsing tree.

3. Reduce Action:

- If the action is reduce, the value whose key is the new state is received from cfg.

- This value represents the current production to be used and is divided into LHS (Left-Hand Side) and RHS (Right-Hand Side), which are stored in the list.

- Symbols included in the RHS are popped from left\_sub\_string and state\_stack, child nodes are formed, and they are added to the parsing tree.

- After the reduce action, goto\_state is obtained using the LHS as the key value from the parsing\_table and added to state\_stack.

4. Acceptance:

- Even if the input symbol is not read to the end, if "acc" (accept) is entered in the action state, it is immediately accepted, and the parsing tree created so far is output.