Nabla

Parser
Team 9

Dive into Parser

- Defined expressions for LALR parsing
- Assigned tokens for various keywords and operators
- *Union* to define *int, float, char** data types
- Using Yacc on *grammar.y* to generate *y.tab.c*

More on Parser

- Grammar has reference to predefined functions such as trigonometric, exponential functions and backward
- Currently grammar prints the derivations of rule matched in reference to the test case provided
- Dangling *else* is tackled using *endif* keyword
- Grammar has preference of tensor operations over arithmetic operations

Parser Implementation

- LALR bottom up parsing is implemented
- Top of Stack is processed at each step
- If a non-terminal is encountered
    - Current terminal and parsing table give us rule to apply
    - Non-terminal is popped from stack
    - Replaced with right hand side of rule from right to left
- If terminal is encountered
    - Must match with current terminal
    - Cursor moves forward in the input string if it’s true

About Grammar

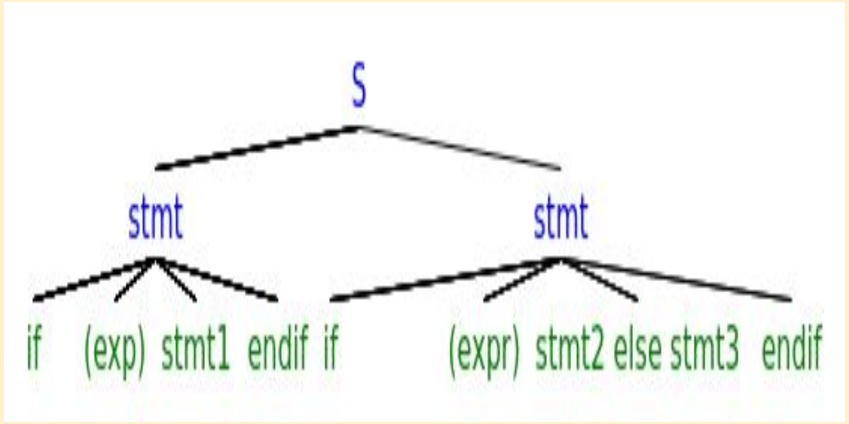
1) Grammar consists of set of terminal, Nonterminal and production rule.
2) Grammar of Nabla is little bit similar to C language grammar.
3) Nabla has a modified grammar which has no shift-reduce and reduce-reduce conflicts.
4) Grammar has inbuilt functions that we can use on the tensors i.e. sin, cos, exp, etc.
5) Nabla grammar eliminates dangling-else ambiguity.
6) Grammar of Nabla gives preference to tensor operations over arithmetic operations while performing operations.

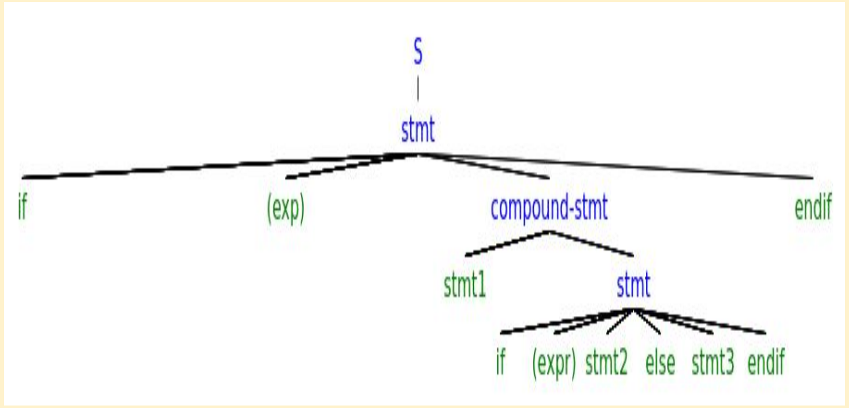
Ambiguity Resolution

Dangling ELSE Ambiguity: Every if statement is ended with corresponding endif statement.

if (exp) stmt1 endif if(expr) stmt2 else stmt endif

if (exp) stmt1 if(expr) stmt2 else stmt endif endif





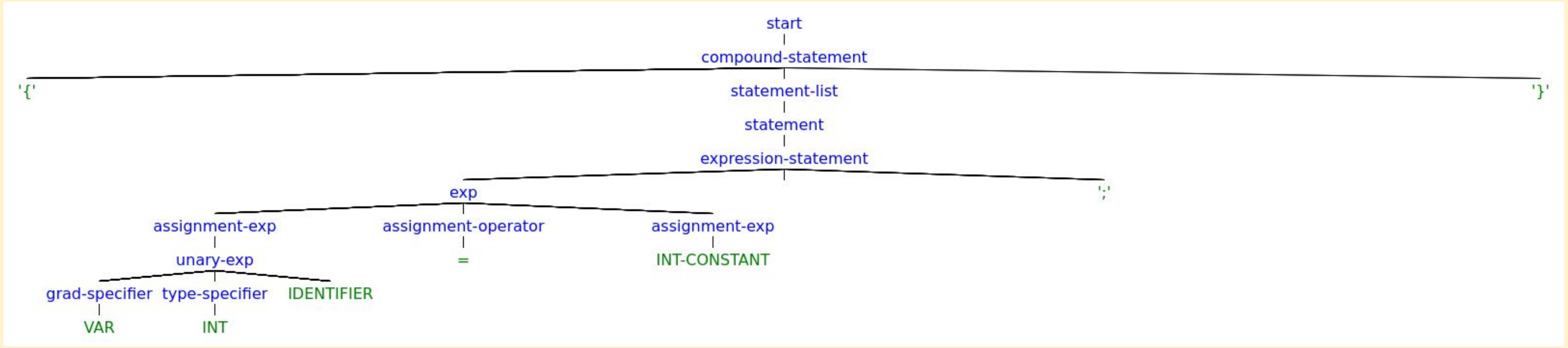
Tokens and union of grammar

- Tokens that are used in grammar are :

  IDENTIFIER CONSTANT STRING_LITERAL SIZEOF GRAD COS SIN EXP LOG BACKWARD INT_CONST FLOAT_CONST CHAR_CONST PRINT INC_OP DEC_OP LEFT_OP RIGHT_OP LE_OP GE_OP EQ_OP NE_OP AT_OP AND_OP OR_OP MUL_ASSIGN DIV_ASSIGN MOD_ASSIGN ADD_ASSIGN AT_ASSIGN SUB_ASSIGN LEFT_ASSIGN RIGHT_ASSIGN AND_ASSIGN XOR_ASSIGN OR_ASSIGN TYPE_NAME CHAR INT TENSOR FLOAT CNS VAR BOOL IF ELIF ELSE LOOP ENDIF

Programs and syntax trees

```
{
  var int num = 10;
}
```



*Note : Syntax trees are generated form http://mshang.ca/syntree/