COMPILER DESIGN LAB EXAM : 26/11/2019
SET 1

For the Grammar G given, build an LL (1) parser which accepts the string id+id*(id+id)

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid id$

LL (1) is a Top-down parser, where the 1st **L** represents that the scanning of the Input will be done from Left to Right manner and second **L** shows that in this Parsing technique we are going to use Left most Derivation Tree. and finally the **1** represents the number of look ahead, means how many symbols are you going to see when you want to make a decision. Non-Recursive predictive parsing is a table-driven parser.

LL (1) parser consists of the following:
input buffer

- our string to be parsed. We will assume that its end is marked with a special symbol $.

output

- a production rule representing a step of the derivation sequence (left-most derivation) of the string in the input buffer.

stack

- contains the grammar symbols
- at the bottom of the stack, there is a special end marker symbol $.
- initially the stack contains only the symbol $ and the starting symbol S. ($S : initial stack)
- when the stack is emptied (i.e. only $ left in the stack), the parsing is completed.

parsing table

- a two-dimensional array M[A,a]
- each row is a non-terminal symbol
- each column is a terminal symbol or the special symbol $
- each entry holds a production rule.

Parser Actions

The symbol at the top of the stack (say X) and the current symbol in the input string (say a) determine the parser action. There are four possible parser actions.

- If X and a are $ → parser halts (successful completion)
- If X and a are the same terminal symbol (different from $) → parser pops X from the stack, and moves the next symbol in the input buffer.

- If X is a non-terminal → parser looks at the parsing table entry M[X,a]. If M[X,a] holds a production rule X ->$Y_1Y_2...Y_k$, it pops X from the stack and pushes $Y_k,Y_{k-1},...,Y_1$ into the stack. The parser also outputs the production rule X->$Y_1Y_2...Y_k$ to represent a step of the derivation.
- None of the above → error
  - All empty entries in the parsing table are errors.
  - If X is a terminal symbol different from a, this is also an error case.

Example:

For the Grammar is $S \rightarrow aBa$; $B \rightarrow bB \mid \varepsilon$ and the following LL(1) parsing table:

	a	**b**	**$**
S	$S \rightarrow aBa$		
B	$B \rightarrow \varepsilon$	$B \rightarrow bB$	

stack	*input*	*output*
$S	abba$	$S \rightarrow aBa$
$aBa	abba$	
$aB	bba$	$B \rightarrow bB$
$aBb	bba$	
$aB	ba$	$B \rightarrow bB$
$aBb	ba$	
$aB	a$	$B \rightarrow \varepsilon$
$a	a$	
$	$	accept, successful completion

Outputs: $S \rightarrow aBa$ $B \rightarrow bB$ $B \rightarrow bB$ $B \rightarrow \varepsilon$

Derivation (left-most): $S \Rightarrow aBa \Rightarrow abBa \Rightarrow abbBa \Rightarrow abba$

Constructing LL(1) parsing tables

- Two functions are used in the construction of LL(1) parsing tables -FIRST & FOLLOW
- Compute FIRST() and FOLLOW()

Algorithm for Constructing LL(1) Parsing Table:

- for each production rule $A \rightarrow \alpha$ of a grammar G
  - for each terminal a in FIRST(α) → add $A \rightarrow \alpha$ to M[A,a]
  - If ε in FIRST(α) → for each terminal a in FOLLOW(A) add $A \rightarrow \alpha$ to M[A,a]
  - If ε in FIRST(α) and $ in FOLLOW(A) → add $A \rightarrow \alpha$ to M[A,$]
- All other undefined entries of the parsing table are error entries.