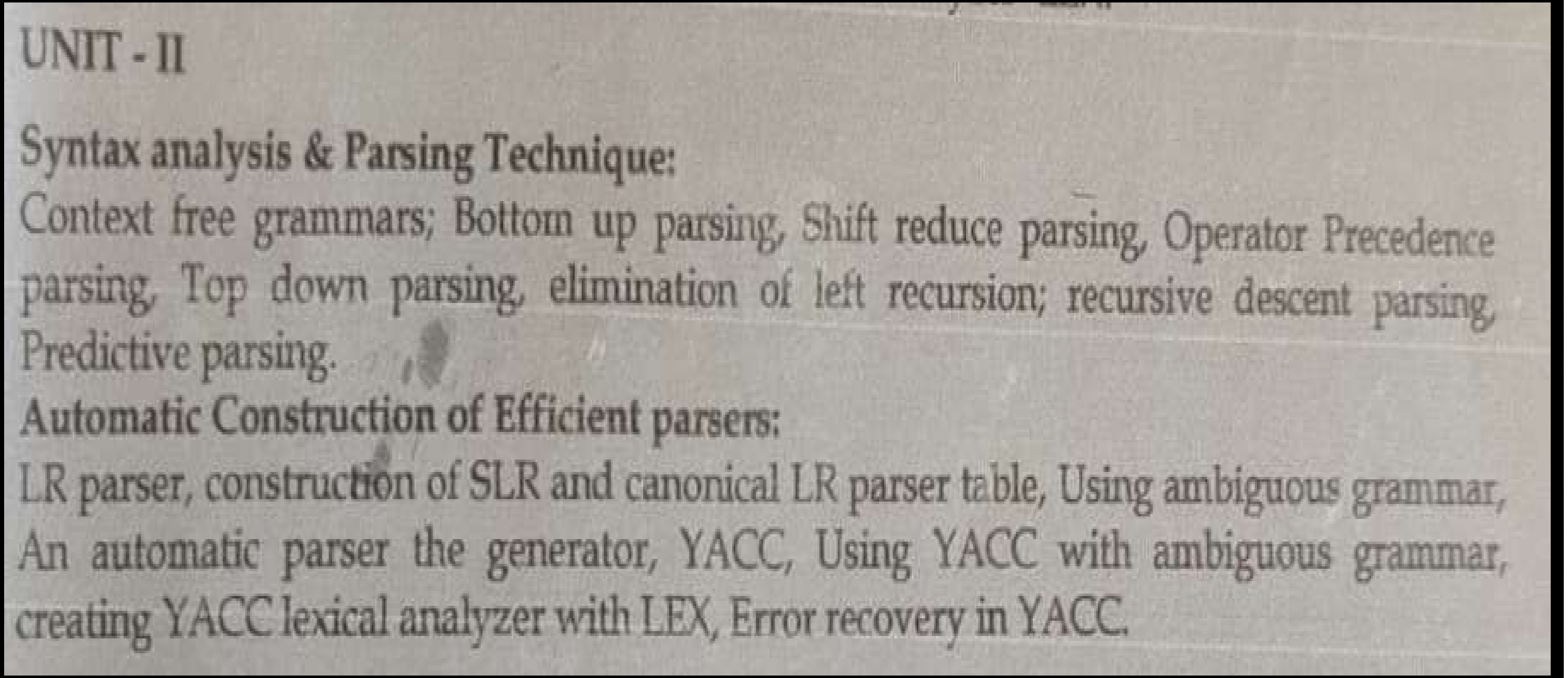
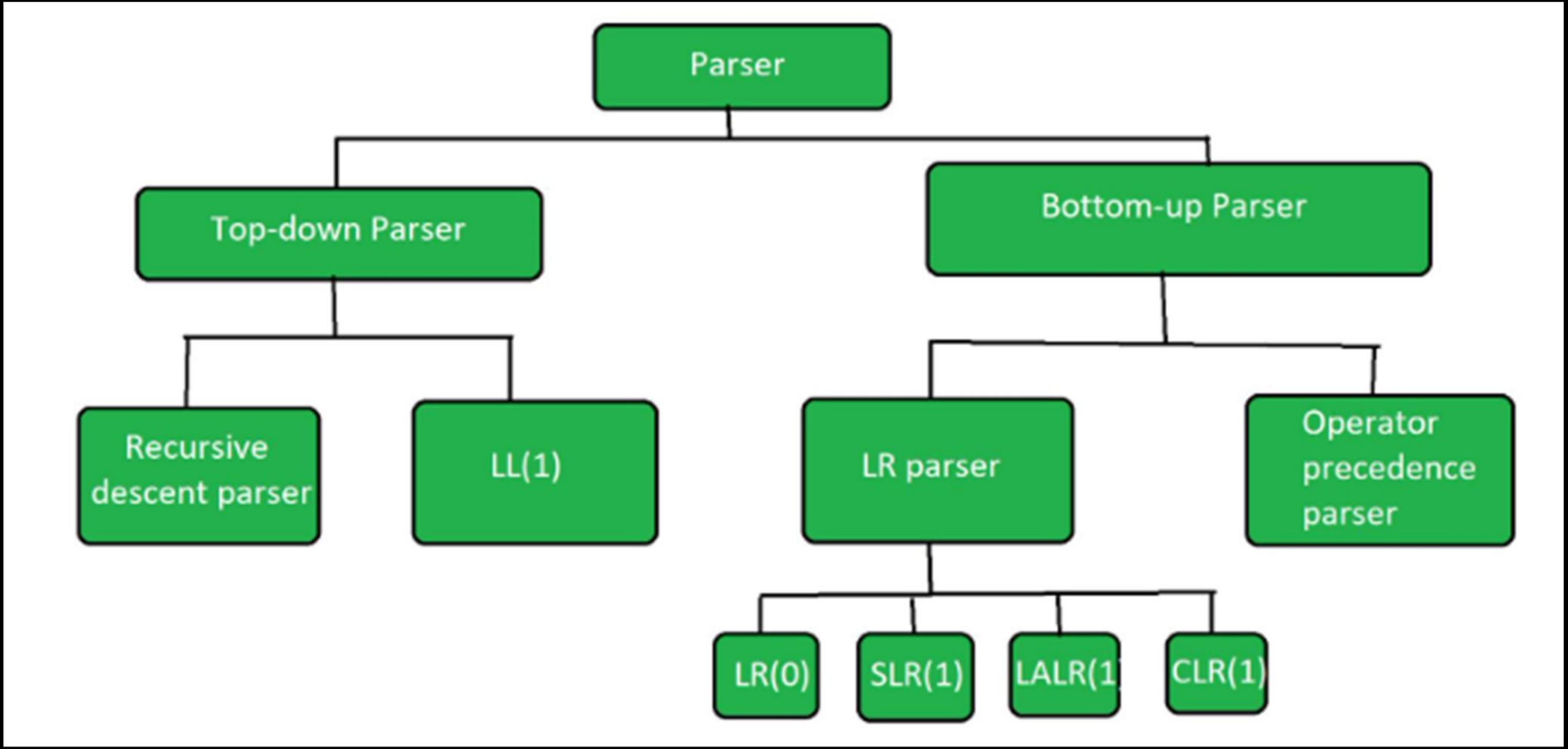
Compiler Design

Durgesh Kumar Keshar
SoS Computer Application
Shaheed Mahendra Karma University

UNIT- II
Automatic Construction Of Efficient Parsers


UNIT - II

Syntax analysis & Parsing Technique:
Context free grammars; Bottom up parsing, Shift reduce parsing, Operator Precedence parsing, Top down parsing, elimination of left recursion; recursive descent parsing, Predictive parsing.
Automatic Construction of Efficient parsers:
LR parser, construction of SLR and canonical LR parser table, Using ambiguous grammar, An automatic parser the generator, YACC, Using YACC with ambiguous grammar, creating YACC lexical analyzer with LEX, Error recovery in YACC.


Parser
Top-down Parser
Bottom-up Parser
Recursive descent parser
LL(1)
LR parser
Operator precedence parser
LR(0)
SLR(1)
LALR(1
CLR(1)

Top Down Parsers

- The top-down parser is the parser that **generates parse for the given input string** with the help of grammar productions by expanding the non-terminals.
- It starts from the start symbol and ends on the terminals.
- It uses left most derivation.
- **Types:-**

  A recursive descent parser

  Non-recursive descent parser.
- Recursive descent parser is also known as the **Brute force parser or the backtracking parser.** It basically generates the parse tree by using brute force and backtracking.
- **Non-recursive descent parser** is also known as **LL(1)** parser or predictive parser or without backtracking parser or dynamic parser. It uses a parsing table to generate the parse tree instead of backtracking.

Bottom Up - Parsers

- Bottom-up Parser is the parser that generates the parse tree for the given input string with the help of grammar productions by compressing the non-terminals.
- It starts from non-terminals and ends on the start symbol.
- It uses the reverse of the rightmost derivation.
- **Further Bottom-up parser is classified into two types:**

1) LR parser
2) Operator precedence parser

LR Parsers

- LR parser is the bottom-up parser that generates the parse tree for the given string by using unambiguous grammar.
- It follows the reverse of the rightmost derivation.
- **LR parser is of four types:**
- (1)LR(0)
- (2)SLR(1)
- (3)LALR(1)
- (4)CLR(1)

Operator Precedence Parsers

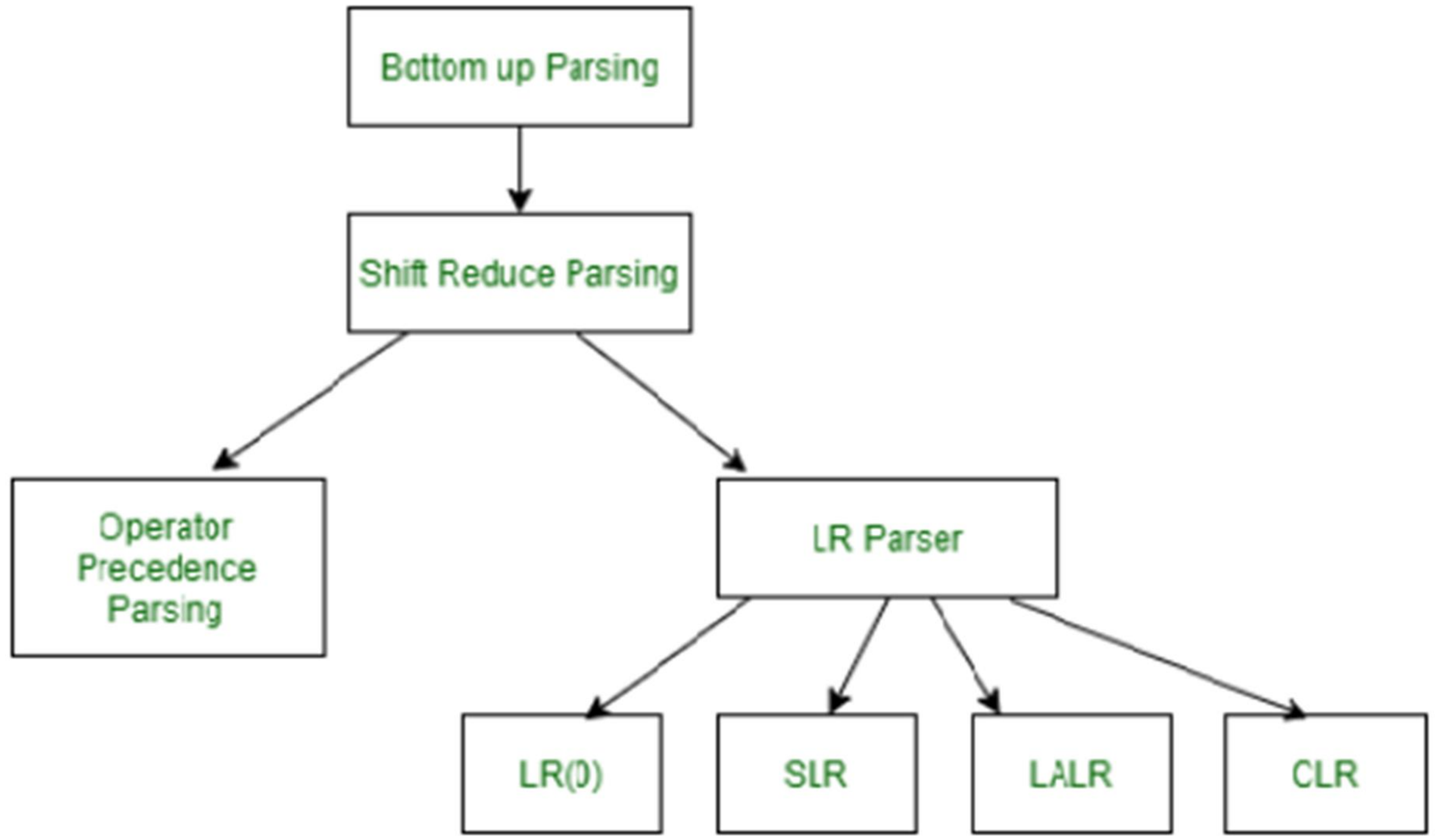
- Operator precedence parser generates the parse tree from given grammar and string but the only condition is two consecutive non-terminals and epsilon never appears on the right-hand side of any production.
- The operator precedence parsing techniques can be applied to **Operator grammars.**
- **Operator grammar:** A grammar is said to be operator grammar if there does not exist any production rule on the right-hand side.

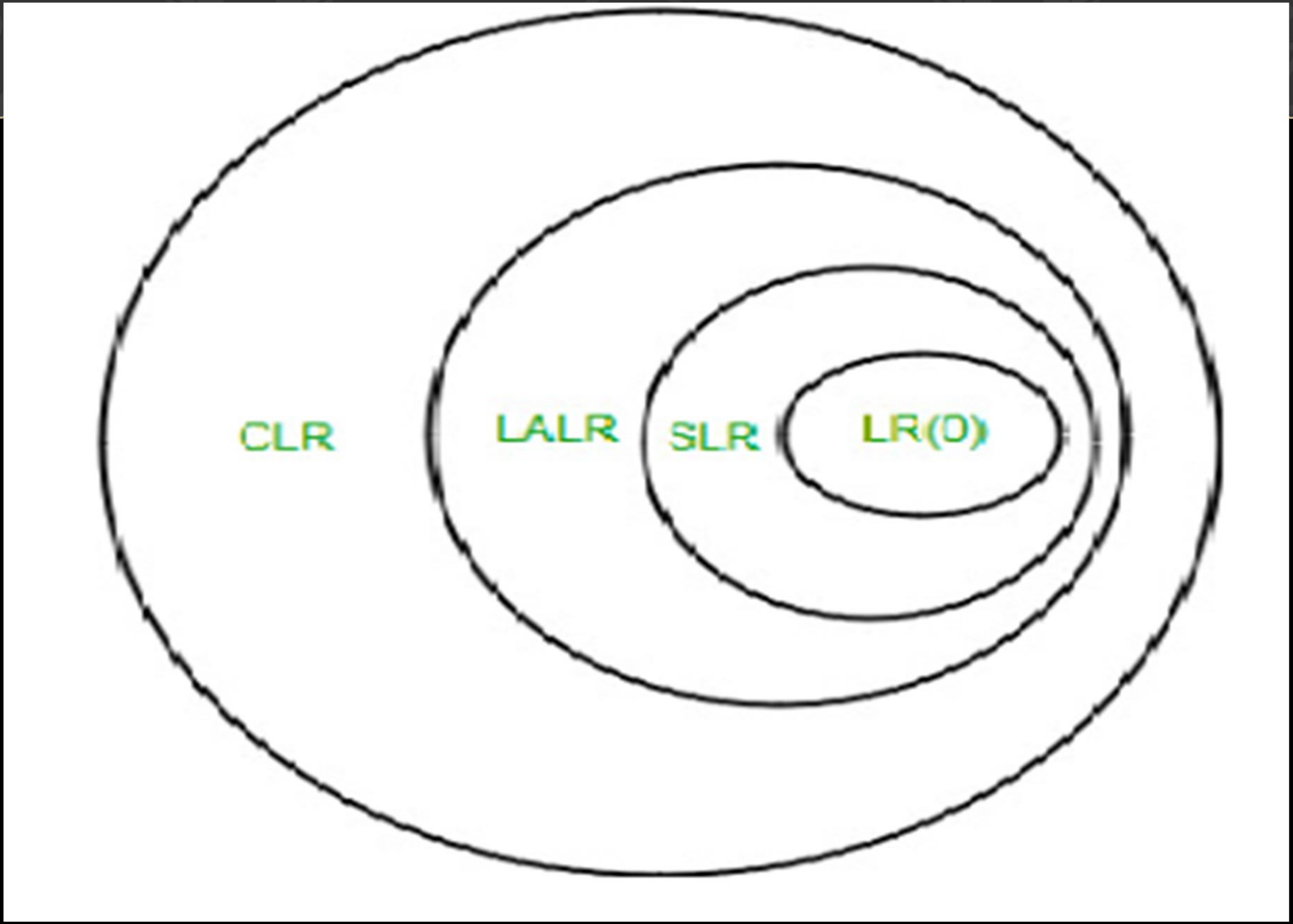
  1. as ε(Epsilon)

  2. Two non-terminals appear consecutively, that is, without any terminal between them operator precedence parsing is not a simple technique to apply to most the language constructs.

LR Parsers

- LR parser is a bottom-up parser for context-free grammar that is very generally used by computer programming language compiler and other associated tools.
- LR parser reads their input from left to right and produces a right-most derivation.
- It is called a Bottom-up parser because it attempts to reduce the top-level grammar productions by building up from the leaves.
- LR parsers are the most powerful parser of all deterministic parsers in practice.
- The term parser LR(k) parser, here the L refers to the left-to-right scanning, R refers to the rightmost derivation in reverse and k refers to the number of unconsumed “look ahead” input symbols that are used in making parser decisions


Bottom up Parsing
Shift Reduce Parsing
Operator Precedence Parsing
LR Parser
LR(0)
SLR
LALR
CLR


CLR
LALR
SLR
LR(0)

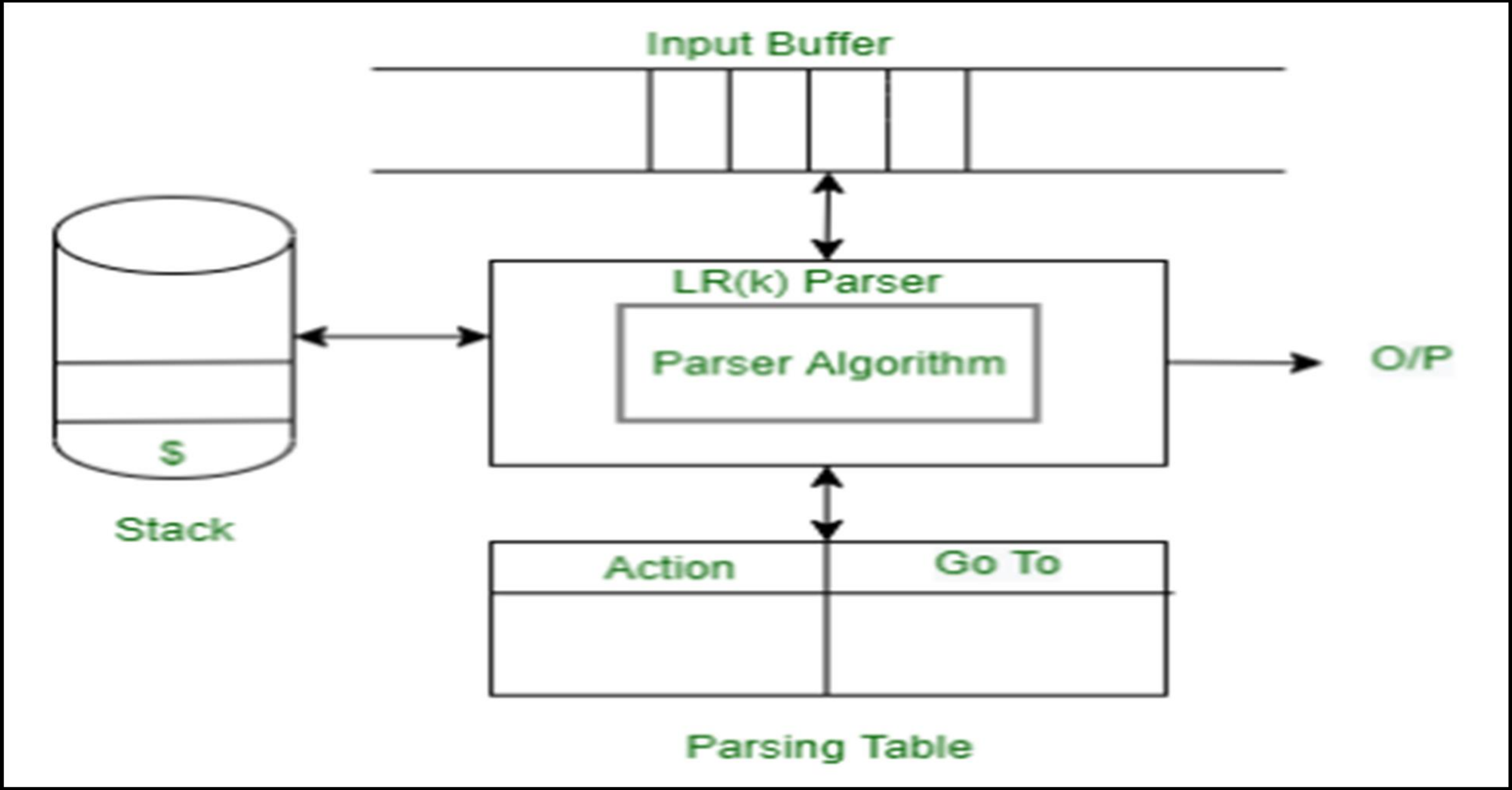
Rules of LR Parsers

- The first item from the given grammar rules adds itself as the first closed set.
- If an object is present in the closure of the form A→ α. β. γ, where the next symbol after the symbol is non-terminal, add the symbol's production rules where the dot precedes the first item.
- Repeat steps (B) and (C) for new items added under (B).
- **Input Buffer –**
  It contains the given string, and it ends with a $ symbol.
- **Stack –**
  The combination of state symbol and current input symbol is used to refer to the parsing table in order to take the parsing decisions.

Parsing Table

- Parsing table is divided into two parts- **Action table and Go-To** table.
- The action table gives a grammar rule to implement the given current state and current terminal in the input stream.
- There are four cases used in action table :-
- **Shift Action-** In shift action the present terminal is removed from the input stream and the state n is pushed onto the stack, and it becomes the new present state.
- **Reduce Action-** The number m is written to the output stream.
- The symbol m mentioned in the left-hand side of rule m says that state is removed from the stack.
- The symbol m mentioned in the left-hand side of rule m says that a new state is looked up in the goto table and made the new current state by pushing it onto the stack.
- **An accept - the string is accepted No action - a syntax error is reported**
- The **go-to table** indicates which state should proceed.

LR Parser Diagram


Input Buffer
LR(k) Parser
Parser Algorithm
O/P
$
Stack
Action
Go To
Parsing Table