SNS COLLEGE OF TECHNOLOGY

Coimbatore-36.
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

COURSE NAME : 19CSB301&Automata Theory and Compiler Design

III YEAR/ V SEMESTER

UNIT – III SYNTAX ANALYSIS AND SEMANTIC ANALYSIS

Topic: Bottom Up Parser_LR Parser

Dr.B.Vinodhini
Associate Professor
Department of Computer Science and Engineering

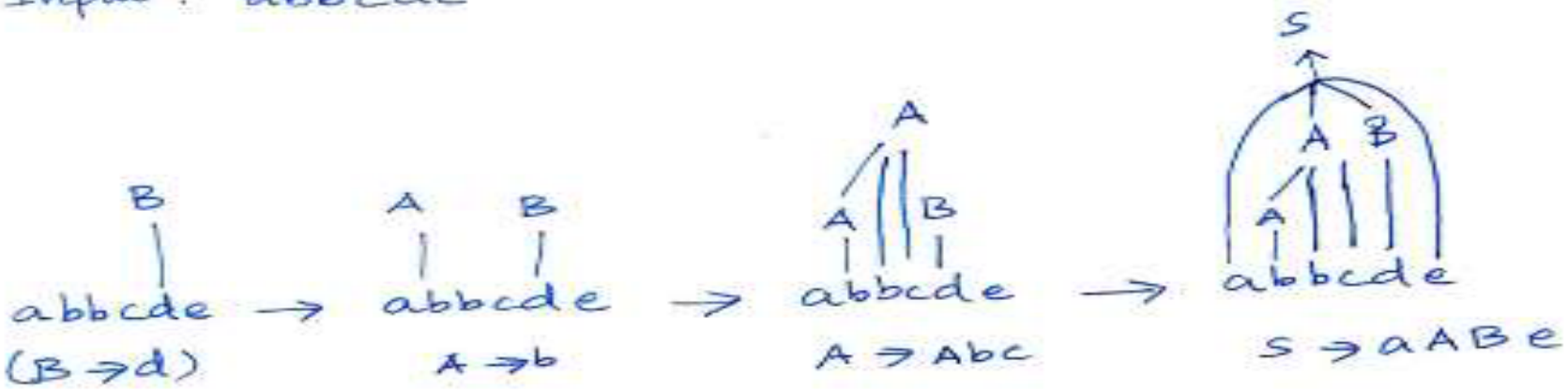
Bottom up parsing (Right most Derivations)

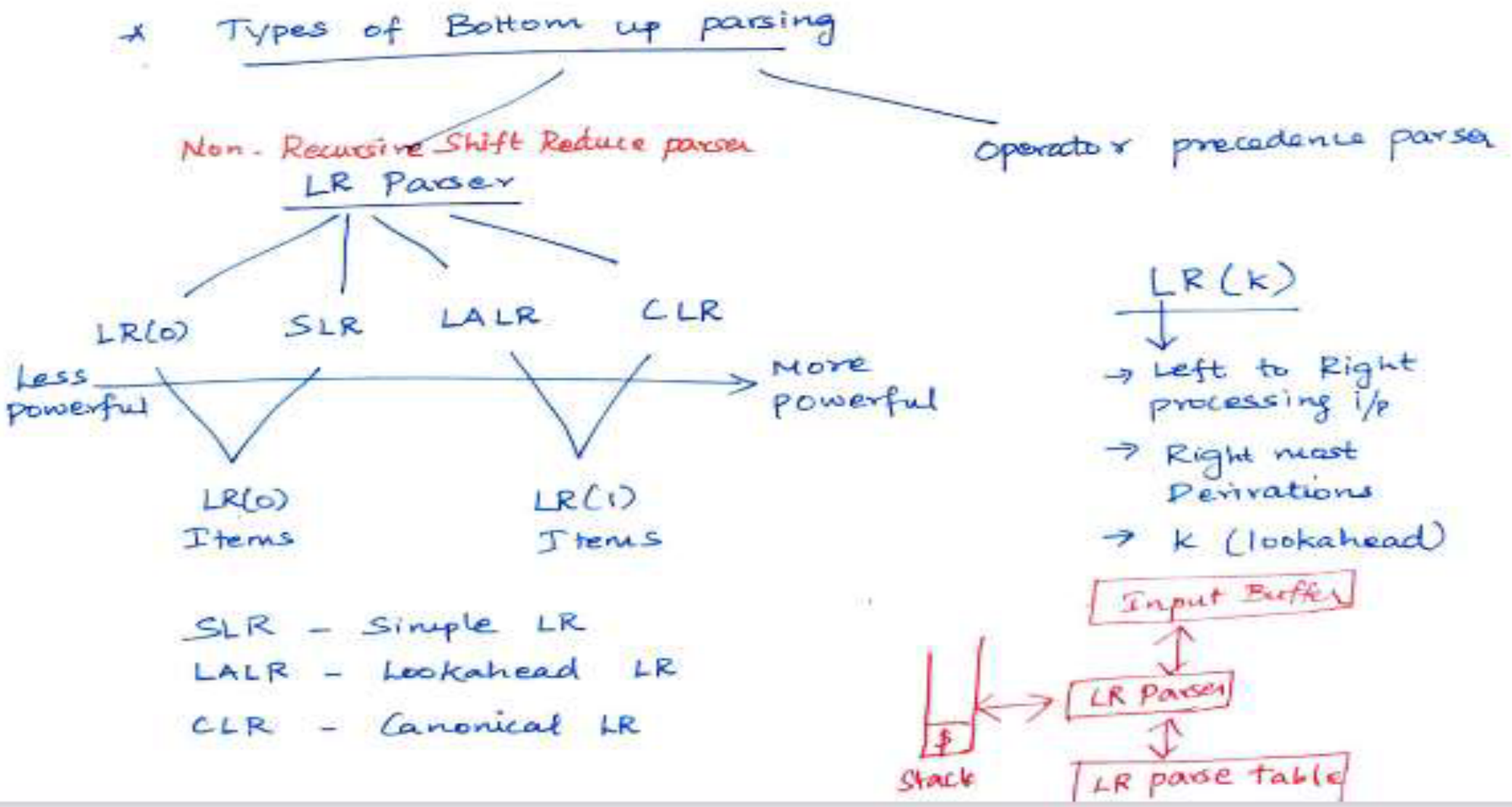
* Terminals → Non-Terminals
  leaf node → Root Node (Start Symbol)

* Example 1:

  S → aABe
  A → Abc | b
  B → d

  Input : abbcde




Types of Bottom up parsing
Non. Recursive Shift Reduce parser
LR Parser
Operator precedence parser
LR(0)
SLR
LALR
CLR
Less powerful
More powerful
LR(0) Items
LR(1) Items
SLR - Simple LR
LALR - Lookahead LR
CLR - Canonical LR
LR(k)
→ Left to Right processing i/p
→ Right most Derivations
→ k (lookahead)
Input Buffer
LR Parser
$
Stack
LR parse table

SHIFT REDUCE PARSING

Shift-reduce parsing is a type of bottom-up parsing that attempts to construct a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top).

Example:

Consider the grammar:

S → aABe
A → Abc | b
B → d

The sentence to be recognized is abbcde.

REDUCTION (LEFTMOST)		RIGHTMOST DERIVATION
abbcde	(A → b)	S → aABe
aAbcde	(A → Abc)	→ aAde
aAde	(B → d)	→ aAbcde
aABe	(S → aABe)	→ abbcde
S		

The reductions trace out the right-most derivation in reverse.

Handles

Handles:

A handle of a string is a substring that matches the right side of a production, and whose reduction to the non-terminal on the left side of the production represents one step along the reverse of a rightmost derivation.

Example:

Consider the grammar:

$E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow (E)$
$E \rightarrow id$

$E \rightarrow \underline{\mathbf{E+E}}$
$\rightarrow E+\underline{\mathbf{E*E}}$
$\rightarrow E+E*\underline{\mathbf{id_3}}$
$\rightarrow E+\underline{\mathbf{id_2}}*id_3$
$\rightarrow \underline{\mathbf{id_1}}+id_2*id_3$

In the above derivation the underlined substrings are called **handles.**

Handle pruning:

A rightmost derivation in reverse can be obtained by "**handle pruning**".
(i.e.) if w is a sentence or string of the grammar at hand, then $w = \gamma_n$, where γ_n is the n^{th} right-sentinel form of some rightmost derivation.

Stack Implementation of Shift Reduce Parsing

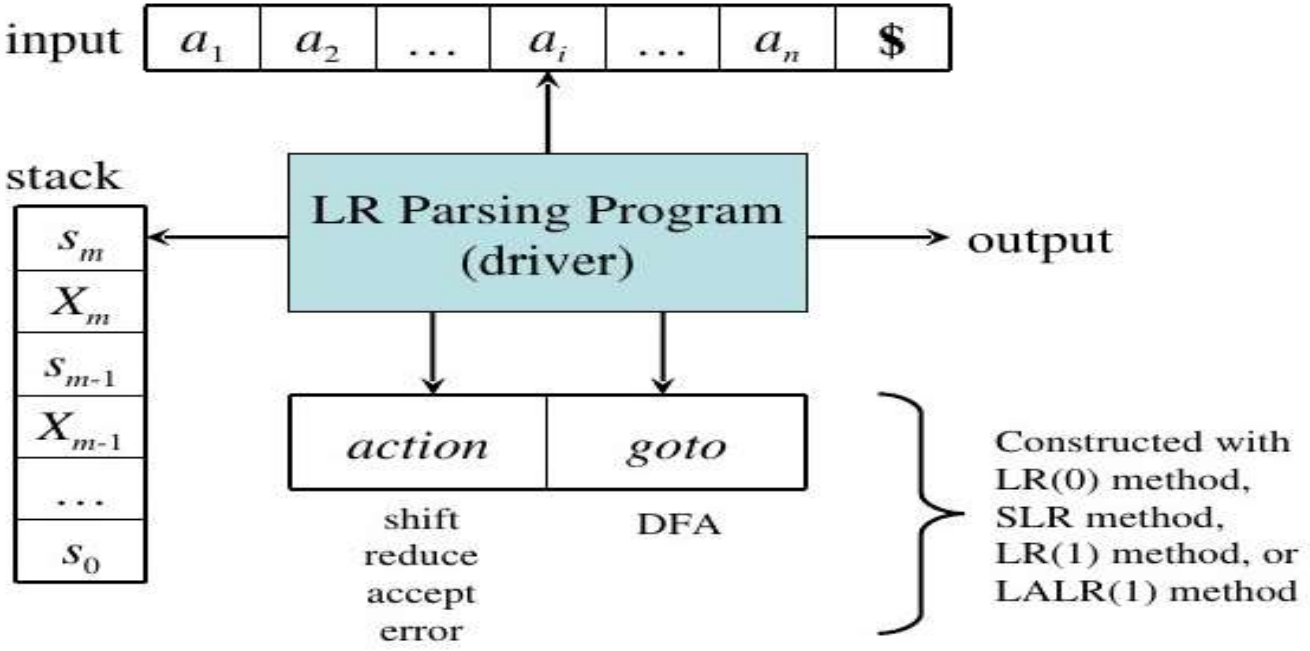
Stack	Input	Action
$	$id_1+id_2*id_3$ $	shift
$ id_1	$+id_2*id_3$ $	reduce by E→id
$ E	$+id_2*id_3$ $	shift
$ E+	id_2*id_3 $	shift
$ E+id_2	$*id_3$ $	reduce by E→id
$ E+E	$*id_3$ $	shift
$ E+E*	id3 $	shift
$ E+E*id3	$	reduce by E→id
$ E+E*E	$	reduce by E→ E *E
$ E+E	$	reduce by E→ E+E
$ E	$	accept

Actions in shift-reduce parser:

- shift – The next input symbol is shifted onto the top of the stack.
- reduce – The parser replaces the handle within a stack with a non-terminal.
- accept – The parser announces successful completion of parsing.
- error – The parser discovers that a syntax error has occurred and calls an error recovery routine.

LR Parsing Method

Model of an LR Parser



LR Parsing Algorithm

Input: An input string *w* and an LR parsing table with functions *action* and *goto* for grammar G.

Output: If *w* is in L(G), a bottom-up-parse for *w*; otherwise, an error indication.

Method: Initially, the parser has s_0 on its stack, where s_0 is the initial state, and *w*\$ in the input buffer. The parser then executes the following program :

```
set ip to point to the first input symbol of w$;
repeat forever  begin
        let s be the state on top of the stack and
            a  the symbol pointed to by ip;
        if action[s, a] = shift s' then begin
            push a then s' on top of the stack;
            advance ip to the next input symbol
        end
        else if action[s, a] = reduce A→β then begin
              pop 2* | β | symbols off the stack;
              let s' be the state now on top of the stack;
              push A then goto[s', A] on top of the stack;
              output  the production A→ β
        end
        else if action[s, a] = accept then
              return
        else error( )
end
```

LR(0) parser

Q. $S \rightarrow AA$

$A \rightarrow aA \mid b$

Step 1: Augmented Grammar

$S' \rightarrow S$

$S \rightarrow AA$

$A \rightarrow aA \mid b$

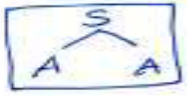
Step 2: Canonical LR(0) items

LR(0) items → productions →

$S \rightarrow .AA$ (yet to read)

$S \rightarrow A.A$ (want to read A)

$S \rightarrow AA.$ (Completing read).



No. The production

1 $S \rightarrow AA$

2 $A \rightarrow aA$

3 $A \rightarrow b$

Step 3: parse Table

	Goto (Non-Termi)		Action (Terminals		
	A	S	a	b	$
I_0	2	1	S_3	S_4	
I_1					Accept
I_2	5		S_3	S_4	
I_3	6		S_3	S_4	
I_4			r_3	r_3	r_3
I_5			r_1	r_1	r_1
I_6			r_2	r_2	r_2

State $\xrightarrow{T}$ Shift

State $\xrightarrow{N\text{-}T}$ Goto

Final State → Red

STACK	INPUT	ACTION
$0	aabb$	
$0a3	abb$	Shift a3
$0a3a3	bb$	Shift a3
$0a3a3b4	b$	Shift b4
$0a3a3A6	b$	Reduce (R_3) 3rd production (3A → b)
$0a3A6	b$	
$0A2	b$	Reduce (R_2) 2. A → aA
$0A2b4	$	Reduce (R_2)
$0A2A5	$	Shift b4
$0S1	$	Reduce (R3) (3. A → b)
		Reduce (R_1) (1. S → AA)
		Accept.

3 → A → 6


Thank
you