Date
Page No.

Assignment 02

Q.N.1 Differentiate between top down and bottom up parser.

Top down	Bottom up
1 Construct tree from root to leaves	Construct tree from leaves to root
2 Uses which RHS to substitute for nonterminal	Uses which rule to reduce terminal
3 produce left most derivation	produce reverse right most deriv
4) Recursive descent LL parser	Shift reduce LR, LALR etc
(5) ~~Recursive~~ Easy for human	Harder for human
(6) It is not accepting ambiguous Grammar	Its accepting Ambiguous gramm
7. It is simple to produce parser	It is difficult to produce pars
8 error detection is weak	error detection is strong
9 It starts with starting symbol of the grammar	It end with starting symbol of the grammar

Date
Page No.

Q.N2 Construct LR parsing table for the given context free grammar.

$S \rightarrow AA$

$A \rightarrow Aa / b$

Solution

Step 1 First augmented grammar the augmented grammar of the given grammar is

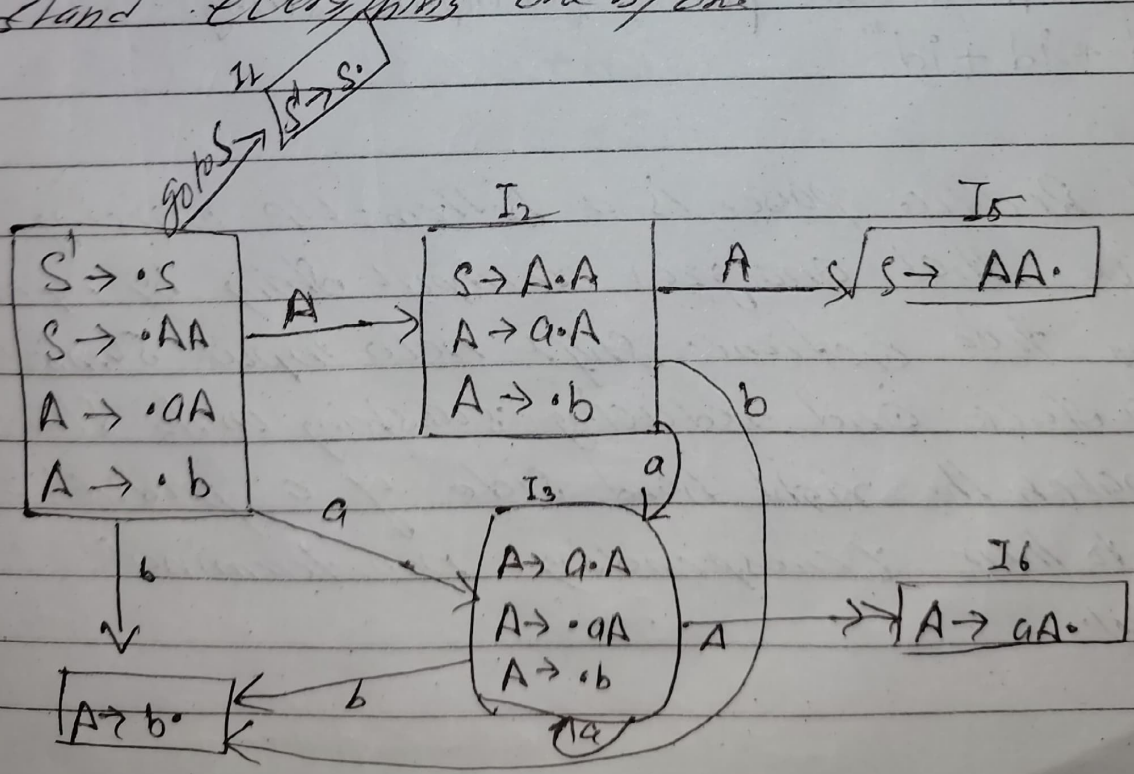
$S' \rightarrow \cdot S$ [0th production]

$S \rightarrow \cdot AA$ [1st production]

$A \rightarrow \cdot aA$ [2nd production]

$A \rightarrow \cdot b$ [3rd production]

Step 2 LR(0) Collection of items Below is the figure showing the LR(0) collection of item. We will understand everything one by one.



	Action			Goto	
	a	b	$	A	S
0	S_3	S_4		2	1
1			Accept		
2	S_3	S_4		5	
3	S_3	S_4		6	
4	R_3	S_3	R_3		
5			R_1		
6	R_2	R_2	R_2		

Q.13 Explain shift reduce parser and its working. Consider the grammar

$S \rightarrow S+S$

$S \rightarrow S*S$

$S \rightarrow id$

perform shift Reduce parsing for input string "id + id + id"

⇒ A shift Reduce parser is a bottom-up parsing technique that analyzes an input string by repeatedly performing two actions: shift new input symbol on to a stack and reducing substring on the stack that match the right hand side of a production rule to their corresponding non terminal symbol.

Date
Page No.

How it work:

Stack: The parser maintains a stack which stores the current parsing state, initially containing the start symbol of the grammar

Input buffer The input string is stored in an input buffer with the next token to be processed at the front

Parsing Table: A table that guides the parser actions based on the current stack symbol and the current input symbol indicating whether to shift reduce or error.

Stack	Input Buffer	Parsing Action
$	id+id+id$	Shift
$id	+id+id$	Reduce S→ id
$S	+id+id$	Shift
$S+	id+id$	Shift
$S+id	+id$	Reduce S→ id
$S+S	+id$	Reduce S→ S+S
$S	+id$	Shift
$S+	id$	Shift
$S+id	$	Reduce S→id
$S+S	$	Reduce S→S+S
$S	$	Accept

Date
Page No.

Q.N4 Construct predictive parsing table for

$S \rightarrow iEtSS_1 / a$

$S_1 \rightarrow eS / \epsilon$

$E \rightarrow b$

Soln First(S) = {i, a}

First(S_1) = {e, ε}

First(E) = {b}

Follow(S) = $

Follow(E) = First(tSS$_1$)

= {t}

Follow(S) = First(S_1)

= {e, ε}

= {e, ε} − ε ∪ follow(S)

follow(S) = {e, $}

follow(S_1) = follow(S)

{e, $}

follow(S) = follow(S_1)

= {e, $}

	$	a	i	b	e
S		S→a	S→iEtSS$_1$	~~E→b~~	
S_1					$S_1 \rightarrow eS$, $S_1 \rightarrow \epsilon$
E				E→b	

Since this is not (LL$_1$) grammar

Date
Page No.

Ans Consider the following grammar

$S \to AaAb \mid BbBa$

$A \to \epsilon$

$B \to \epsilon$

Test whether the grammar is LL(1) or not and Construct a predictive parsing table for it.

Sol: We Calculate first and follow of given production

first(S) = first(AaAb) U first(BbBa) — ①

first(A) = {ε}

first(B) = {ε}

first(S) first(A) - ε U first(aAb)

~~(ε)~~ - ε U {a}

{a}

first(S) = first(B) - ε U first(bBa)

~~(ε)~~ - ε U {b}

= {b}

first(S) = {a, b}

follow(S) = $

follow(A) = first(aAb)

= {a}

follow(A) = first(b)

= {b}

findfollow(A) = {a, b}

follow(B) = {a, b}

	$	a	b
S		S → AaAb	S → BbBa
A		A → ∈	A → ∈
B		B → ∈	B → ∈

There is no multiple entry so this is LL(1) grammar

Q6 what is predictive parsing. Explain Component of predictive parsing and its drawback

⇒ predictive parsing is a top-down parsing technique in Compiler design where the parser can predict which production rules apply next without backtracking essentially making a decision based solely on the current non terminal on the stack and the next input symbol, eliminating the need to reconsider choice later in the parsing process.

Component of predictive parsing.

- parsing Table: A table that stores the production rules to apply based on the current non-terminal on the stack and the next input symbol
- input Buffer: Holds the input string to be parsed usually with the end of string marker

Stack: A data structure that keeps track of the current state of the parse containing non terminal and sometime state information

Drawbacks of predictive parsing:

- Grammar Restrictions:
predictive parsing only work with grammar that are LL(1) which means they must be free from left recursion and left factoring limiting its applicability to certain language

- Error Recovery Difficult:
~~Predictive parsing only work with grammar~~ that are predictive parsers can struggle to recover from syntax errors gracefully as they typically don't have mechanism to backtrack and explore alternative parsing path

- Complexity for large Grammar:
Creating a parsing table for complex grammar can become very large and difficult to manage

Key point about predictive parsing:

No Backtracking:
The primary advantage of predictive parsing is that it does not need to backtrack making it efficient compared to other top-down parsing techniques.

LL(1) parsing: The most common implementation of predictive parsing is called LL(1) which stand for left to Right, left most derivation lookahead of one symbol

Compiler design Applications:
predictive parsing is widely used in Compiler design for syntax analysis where it help to determine the structure of a program by