Aman Chauhan 1805158

① Closure Operation : If I is a set of items for a grammer G, then closure(I) is the set of items constructed from I by the two rules:

i) Initially every item in I is added to closure(I)

ii) If $A \to \alpha . B\beta$ is in closure(I) and $B \to \gamma$ is a production then add the item $B \to \gamma$ to I, if it is not already there. We apply this rule until no more items can be added to closure(I).

Example:

$S' \to S$

$S \to AA$

$A \to aA \mid b$

Closure($S' \to S$) = $S' \to .S$

$S \to .AA$

$A \to .aA \mid .b$

Closure($S \to A.A$) = $S \to A.A$

$A \to .aA \mid .b$

Closure($A \to .aA$) = $A \to a.A$

② GoTo Function : To model the transition that the parser would make from a given state on some grammer symbol x, the algorithm computes the set of items that would result from recognizing an x. To do so, the algorithm selects the subset of the current set of LR(1) items where precedes x and advances the fact the x in each of them.

Algorithm :

Finds state transitions

Aman Chauhan 1805158

s: a set of LR(1) items
x: a terminal or non-terminal symbol

```
goto(s, x)
    moved ← { }
    for each item i in s
        for i is like [A → B.xS, a] then
            moved ← moved ∪ {[A → Bx.S, a]}
    return closure(moved)
```

③ LR Algorithms

```
token = next_token()
repeat forever
    s = top of stack
    if action[s, token] = "shift si" then
        Push token
        Push si
        token = next_token()
    else if action[s, token] = "reduce A::=β" then
        POP 2 * |β| symbols
        s = top of stack
        Push A
        Push goto[s, A]
    else if action[s, token] = "accept" then
        return
    else
        error()
```

④ i) Handle Pruning :- It is the general approach used in shift and reduce

parsing. A handle is a substring that matches the body of a production. Handle reduction is a step in the reverse of rightmost derivation. A rightmost derivation in reverse can be obtained by handle pruning.

ii) Kernel Items & which include the initial item, $S' \rightarrow .S$ and all item whose dots are not at the left end.

iii) Non-kernel Items & which have their dots at the left end.

(5) LR parser is most powerful among bottom up parsers. It is a non-recursive, shift-reduce, bottom up parser. It uses a wide class of context-free grammer which makes it the most efficient syntax and analysis technique. LR parsers are also known as LR(k) parsers, where L stands for left to right scanning of the input stream, R stands for the construction of right-most derivation in reverse and k denotes the number of look ahead symbols to make decisions.

(6) Predictive Parser & i) It comes under the top-down parsing.

ii) Backtracking does not show its impact on predictive parser.

iii) The input string can be replaced by using the parser of recursive descent.

iv) Look ahead pointer is used for denoting the succeeding input symbols.

Aman Chauhan 1805158

Shift Reduce Parser :- i) It is the type of bottom up parsing.

ii) Shift step and reduce step are unique techniques in shift reducing parsing.

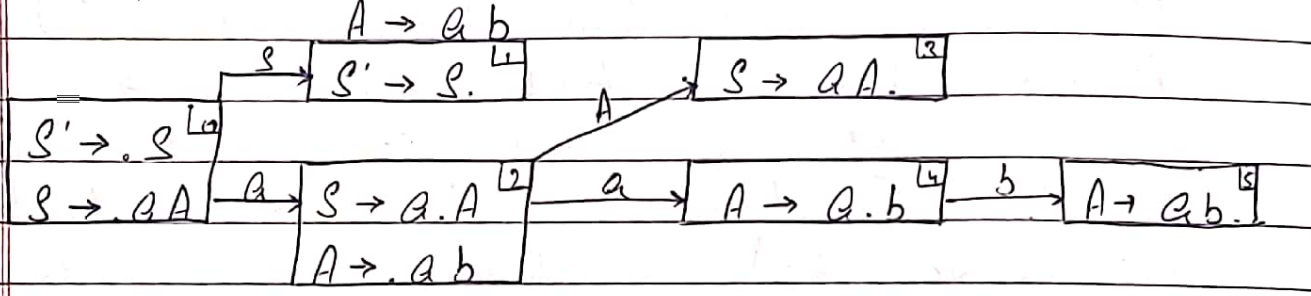
iii) The next input symbol is pointed by shift step technique.

iv) The complete grammer rule are tracked by reduce step technique.

(7) :) **LR(0) Item** & An LR(0) item is a production G with dot at some position on right side of the production. LR(0) item is useful to indicate that how much of the input has been scanned up to a given point in the process of parsing.

Example → $S \to aA$

$A \to ab$



State	Action a	Action b	Action $	GoTo S	GoTo A
0	S_2			1	
1			Accept		
2	S_4				3
3	r_1	r_1	r_1		
4		S_5			
5	r_2	r_2	r_2		

Aman Chauhan 1805158

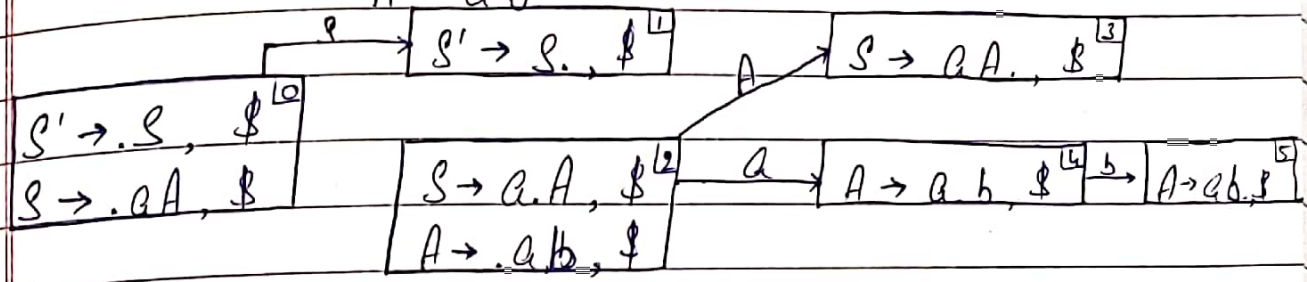
ii) **LR(1) Item** :- It is a collection of LR(0) items and a look ahead symbol.

LR(1) item = LR(0) item + look ahead

The look ahead is used to determine that where we place the final item. The look ahead always add $ symbol for the argument production.

Example → S → aA

A → ab



State	a	b	$	S	A
0	S_2			1	
1			Accept		
2	S_4				3
3			r_1		
4		S_5			
5			r_2		