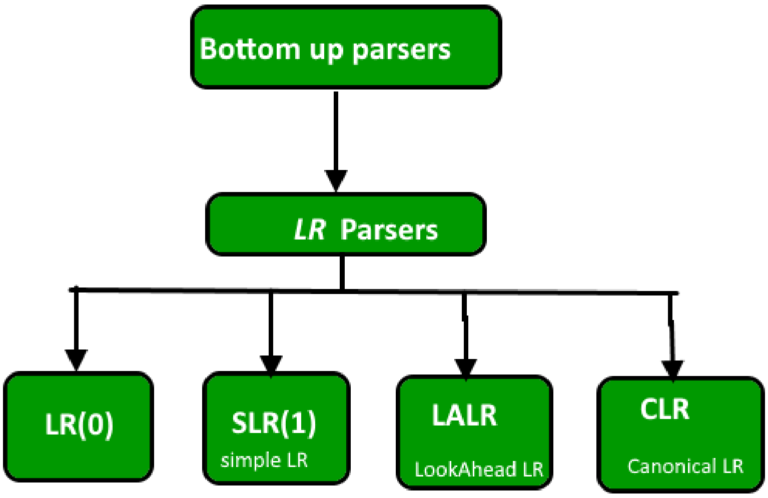
Syntax Analysis- III

Bottom-Up Parsing

Introduction:

It is the process of reducing the input string to the start symbol. Here parse tree is constructed from leaves to the root. Also known as ***Shift-Reduce Parsing.*** The goal of bottom-up parser is to construct a derivation in reverse.

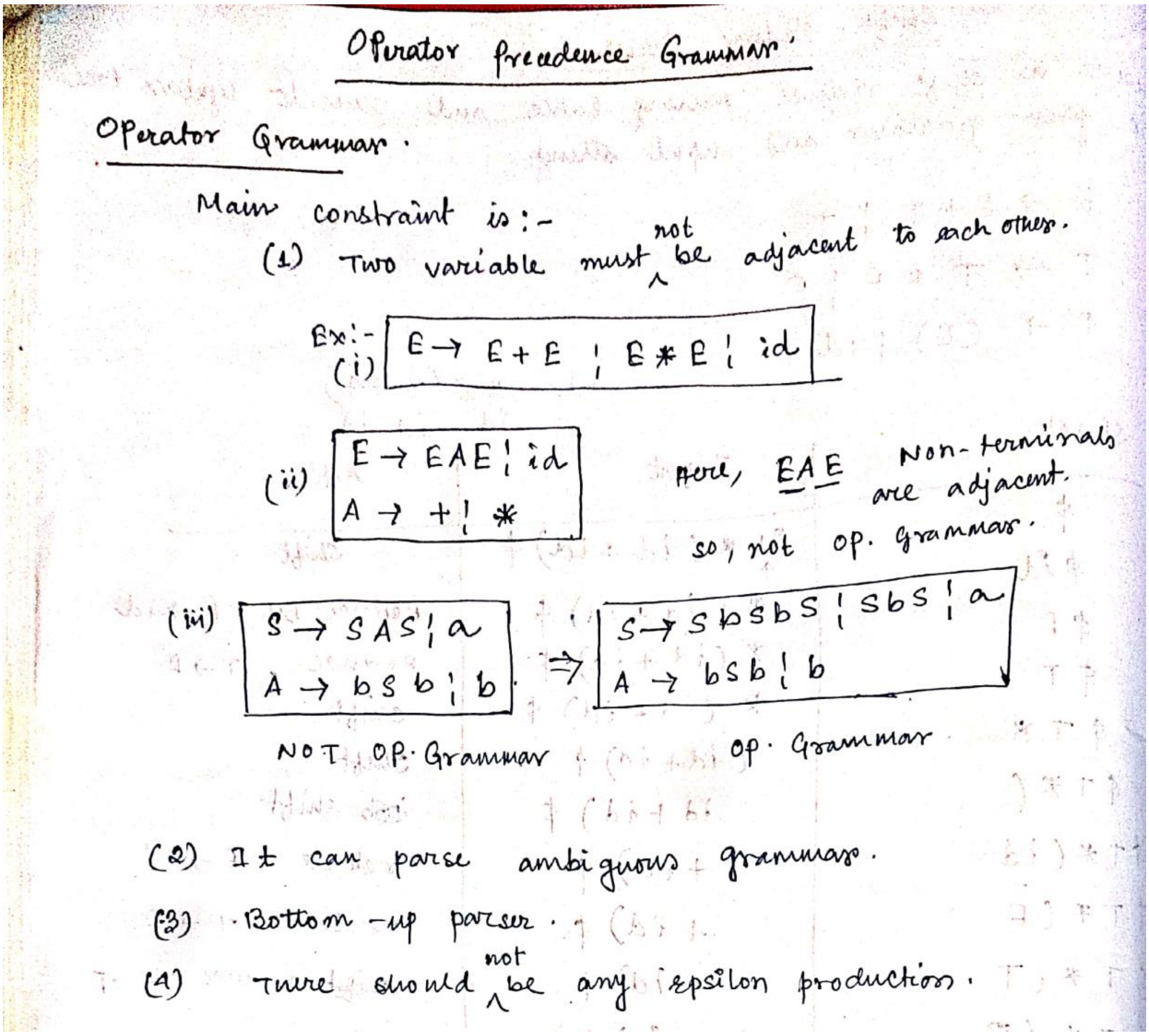
- Bottom-up parsing starts from the leaf nodes of a tree and works in upward direction till it reaches the root node.
- We start from a sentence and then apply production rules in reverse manner in order to reach the start symbol.



Operator Precedence Parsers

Operator precedence parsing is another type of bottom up parser. It works only for 'Operator' grammar. Operator grammar is a grammar where a production rule does not have two consecutive non-terminals.

In such grammars, we apply an operator precedence such as **<•** = or **•>** between consecutive operators, when input is being scanned into a stack which operator precedence symbol will be applied is determined by the operator precedence table.



Operator precedence Grammar.

Operator Grammar.

Main constraint is:-

(1) Two variable must not be adjacent to each other.

Ex:- (i) $\boxed{E \rightarrow E+E \mid E*E \mid id}$

(ii) $\boxed{\begin{array}{l} E \rightarrow EAE \mid id \\ A \rightarrow + \mid * \end{array}}$ Here, EAE Non-terminals are adjacent. so, not op. grammar.

(iii) $\boxed{\begin{array}{l} S \rightarrow SAS \mid a \\ A \rightarrow bSb \mid b \end{array}} \Rightarrow \boxed{\begin{array}{l} S \rightarrow SbSbS \mid SbS \mid a \\ A \rightarrow bSb \mid b \end{array}}$

NOT OP. Grammar   op. Grammar

(2) It can parse ambiguous grammars.

(3) Bottom-up parser.

(4) There should not be any epsilon production.

Advantages:

1) These are simple and easy to operate.

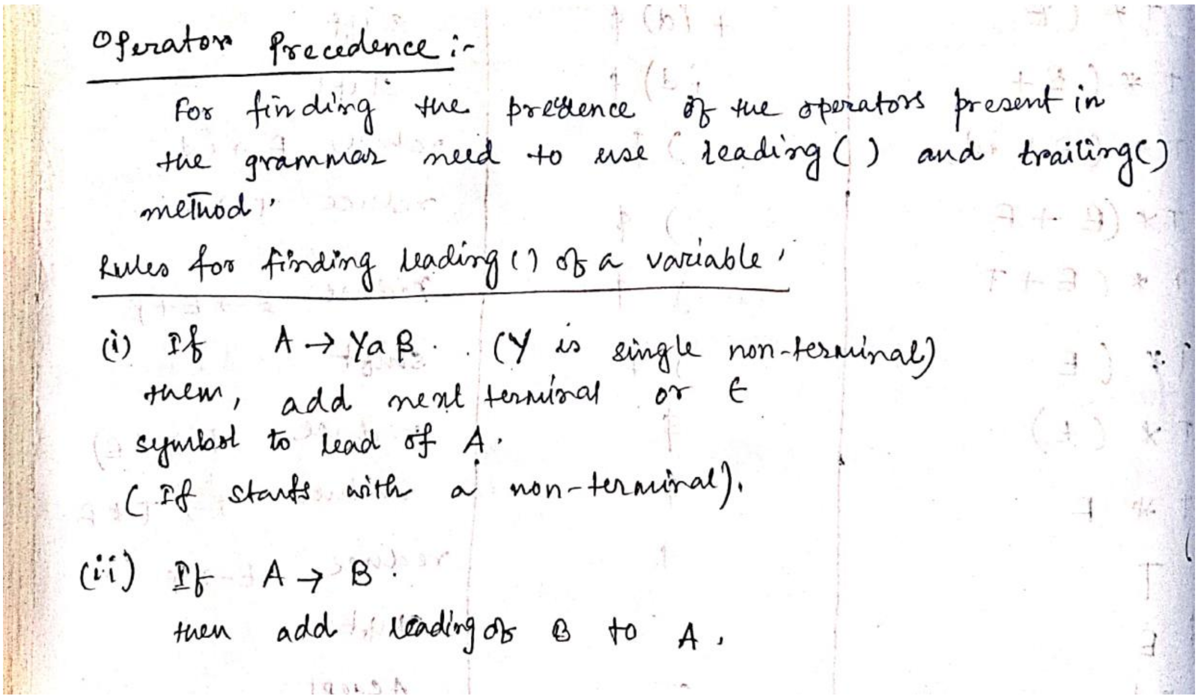
2) Handles are easy to find since a handle is always enclosed by **<●** and **●> .**

Disadvantages:

1) Finding the correct precedence for all operators is difficult.

2) Often it is not possible to find the proper precedence for a pair of operators.

Use:- This is usually used for parsing arithmetic expressions because operator precedence for such expressions is easy to understand.



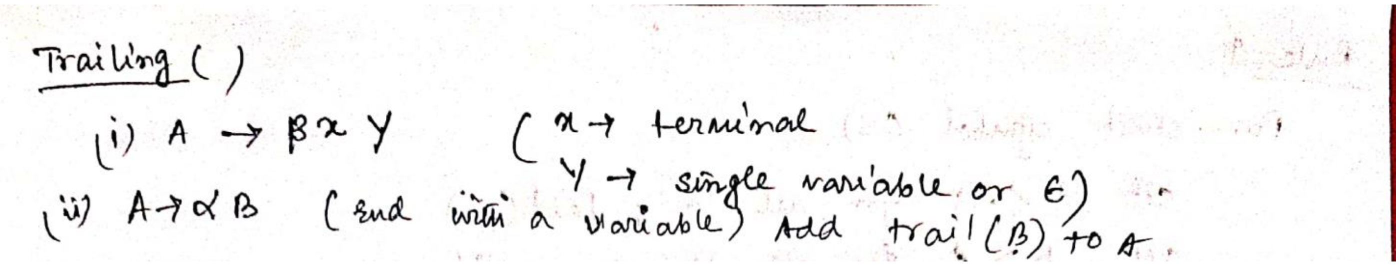
Operator Precedence :-

For finding the predence of the operators present in the grammar need to use leading( ) and trailing() method.

Rules for finding leading() of a variable.

(i) If $A \rightarrow Ya\beta$. (Y is single non-terminal) them, add next terminal or ϵ symbol to lead of A.
(If starts with a non-terminal).

(ii) If $A \rightarrow B$.
then add leading of B to A.



Trailing ( )

(i) $A \rightarrow \beta x Y$ (x → terminal
Y → single variable or ϵ)

(ii) $A \rightarrow \alpha B$ (End with a variable) Add trail(B) to A.

Ex:

$E \rightarrow E + T$

$E \rightarrow T$

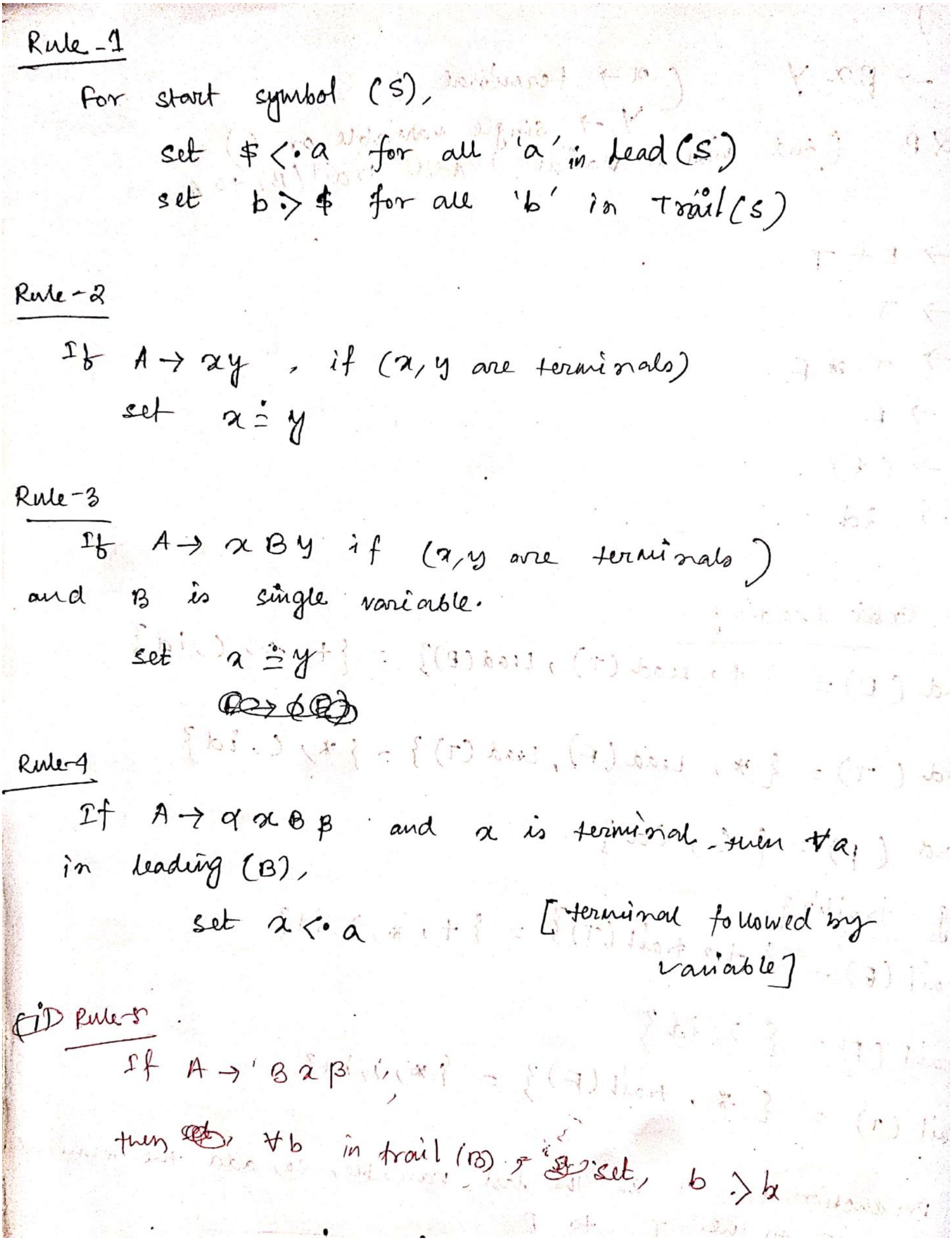
$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow id$.

Creating an operator precedence table:



Rule-1

For start symbol (S),

set $\$ \lessdot a$ for all 'a' in lead(S)

set $b \gtrdot \$$ for all 'b' in trail(S)

Rule-2

If $A \to xy$, if (x, y are terminals)

set $x \doteq y$

Rule-3

If $A \to xBy$ if (x, y are terminals)

and B is single variable.

set $x \doteq y$

Rule-4

If $A \to \alpha x B \beta$ and x is terminal then $\forall a$, in leading (B),

set $x \lessdot a$ [terminal followed by variable]

(ii) Rule-5

If $A \to$ ' $B x \beta$

then $\forall b$ in trail(B), set, $b \gtrdot x$

	+	*	(	)	Id	$
+						
*						
(						
)						
Id						
$						