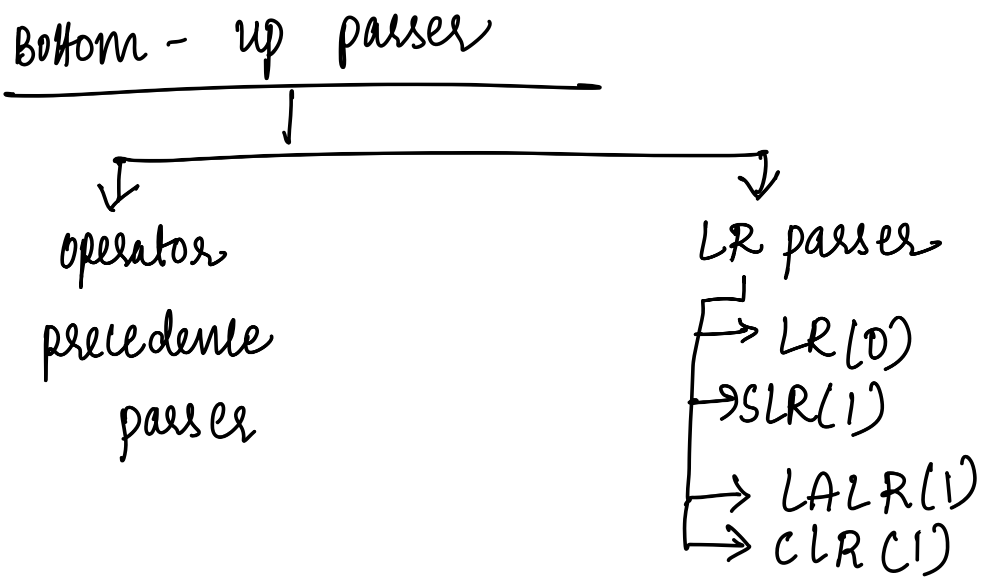
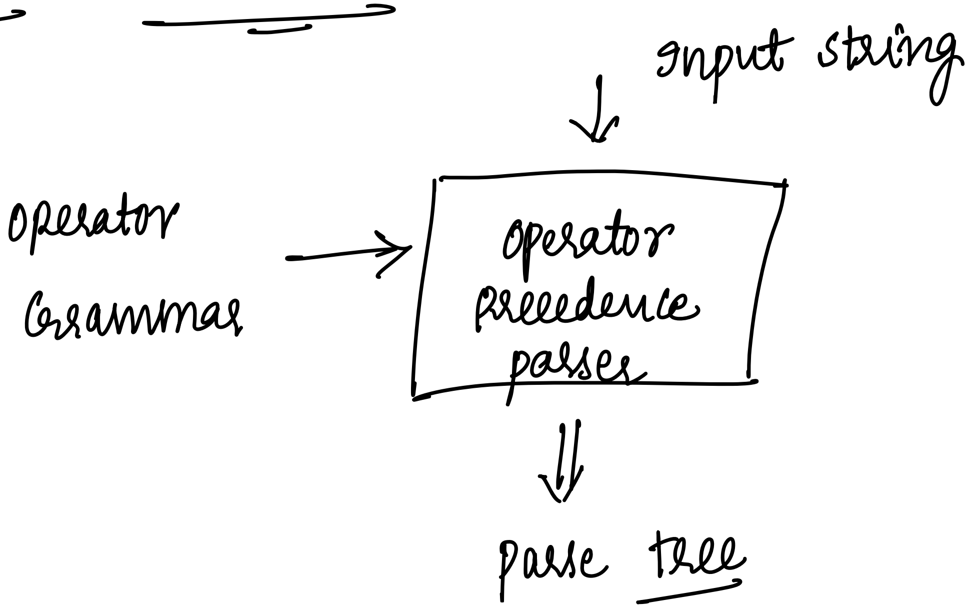
Bottom- up parser :

It constructs the tree from Bottom-up and it considers the Terminals (input) from right - left



* OPERATOR PRECEDENCE



Rules:

operator Grammar:

How to Identify? checking the below conditions:

① NO epsilon (ε) on the right side

② NO adjacent variable. (Having 2 or more N.T together)

Eg:

Adjacent variable yes/NO.

1. AB — YES ✓
2. A+B — NO ✗
3. A-B — NO ✗
4. A/B — ✗ NO

If you find 2, 3, 4 → grammar is Op Grammar.
(operator precedence)

1. $E \rightarrow E+E / E*E / id$?

Is this OP Grammar YES

2. $E \rightarrow EAE / id$ NO

3. $A \rightarrow + / *$ YES

If there are adjacent variables, then you have to convert it into OP Grammar.

Example: $S \rightarrow S\underset{\sim}{A}S \mid id$

$A \rightarrow aSa \mid a$

Conversion:

1. $S \rightarrow S\underset{\sim}{A}S \mid id$
2. $A \rightarrow \underline{aSa} \mid \underline{a}$

In ① substitute 2.

$S \rightarrow SaSaS \mid S \rightarrow SaS \mid id$

$A \rightarrow aSa \mid a$

Basic knowledge of operators:

id, a, b, c ...., terminals → High ↑

$ → Low

$+ \;>\; +$ (left most has more precedence).

$* \;>\; *$ "

id ≠ id not equal

$ A $ Accept

(1) $T \rightarrow T+T \mid T*T \mid id$

with the help of following grammar parse the input string

" id + id * id "

Solution STEPS :-

1. check if the grammar is OP or not
2. OP Relation table
3. Parse the given string
4. Generate the parse tree.

1. ✓ Grammar is OP.

2. $T \rightarrow T+T \mid T*T \mid id$

High precedence
1. *
2. +
3. —
4. %

	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>	-	>
$	<	<	<	A

Operator precedence Relation Table

A : Accept

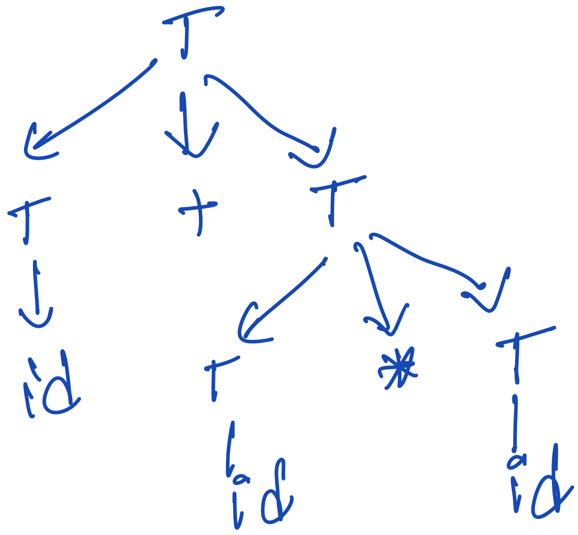
$T \rightarrow T+T$ ✓
$T \rightarrow T*T$ ✓
$T \rightarrow id$

3. Parse the given String · id + id * id $

Stack	Relation	input	Action (Shift / Reduce
$ compare $ & id·	<	id + id * id $	shift id·
$ id if this is greater then reduce·	>	+ id * id $	Reduce T→id
$ T	<	+ id * id $	shift +
$ T+	<	id * id $	shift id
$ T+ id ↓	>	* id $	Reduce T→id
$ T+T	<	* id $	shift *
$ T+T*	<	id $	shift id
$ T+T* id	>	$	Reduce T→id

$ T+T*T	>	$	Reduce T → T*T
$ T+T	>	$	Reduce T → T+T
$ T	Accept	$	

4. Considering the stack values from Bottom to up we build the parse tree.



T
T
+
T
id
T
*
T
id
id