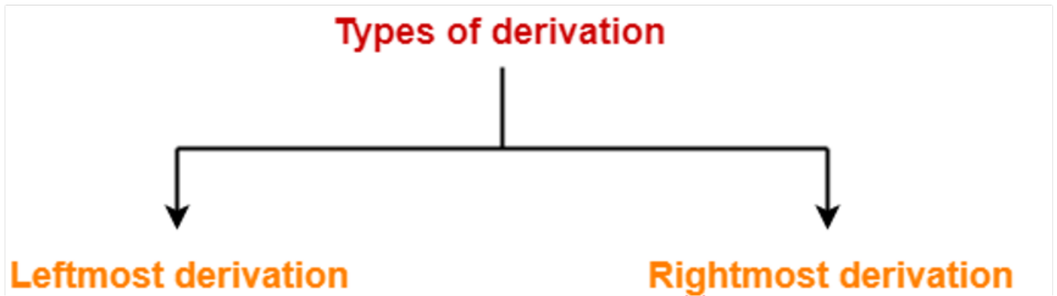
Parse Tree-

The process of deriving a string is called as derivation.
The geometrical representation of a derivation is called as a parse tree or derivation tree.



The parse tree follows these points:
1.All leaf nodes have to be terminals.
2.All interior nodes have to be non-terminals.
3.In-order traversal gives original input string.

ng left most derivation.

Consider the following grammar-

S → aB / bA

S → aS / bAA / a

B → bS / aBB / b

A ---> a

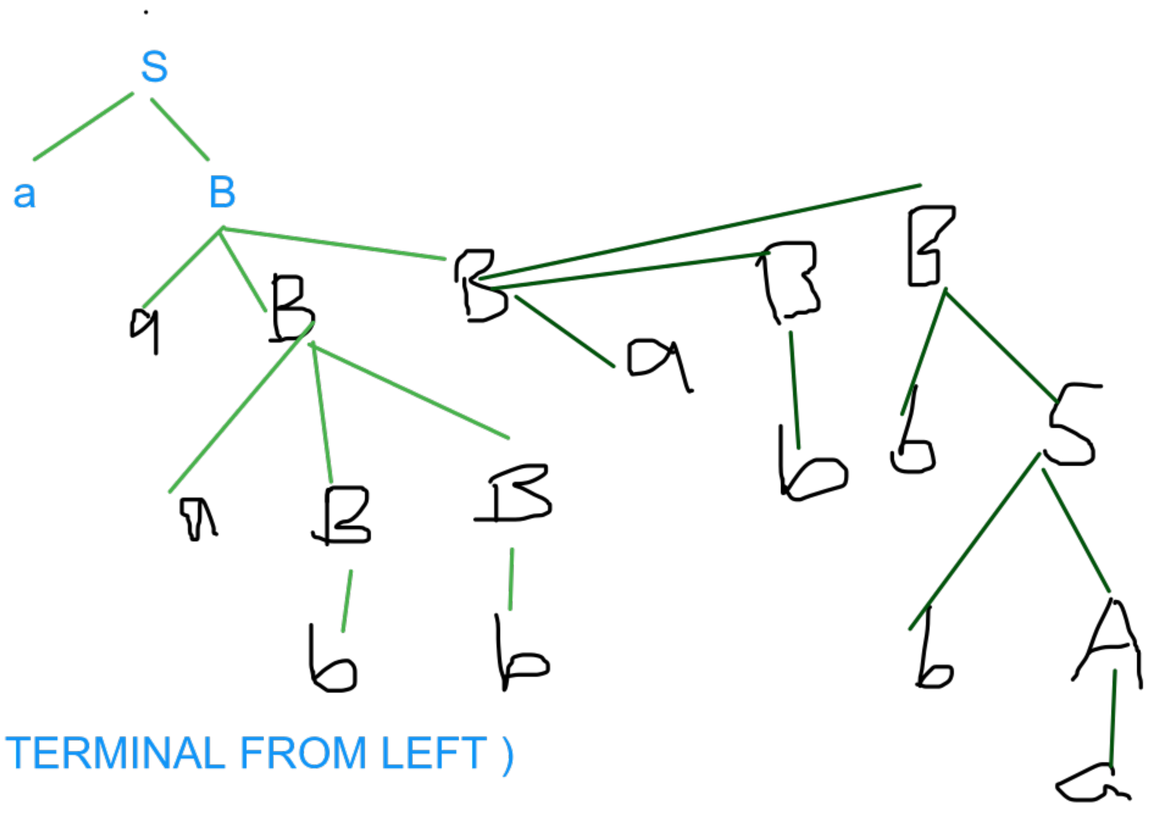
Let us consider a string w = aaabbabbba

s-->aB
---->aaBB (B-> aBB)
---->aaaBBB (B-> aBB)
----->aaabBB (B->b)
----> aaabbB (B->b)
---->aaabbaBB (B->aBB)
---->aaabbabB (B->b)

-----> aaabbabbS (B->bS)

-----> aaabbabbbA (s-> bA)

-------> aaabbabbba (A->a)



CHANGE THE NON TERMINAL VALUE TO TERMINAL FROM LEFT)

Problem-01:

Consider the grammar-

S → bB / aA

A → b / bS / aAA

B → a / aS / bBB

For the string w = bbaababa, find-

Rightmost derivation
Parse Tree

S ----------> bB

-------->bbBB (B-> bBB)

-------->bbBaS (last B -> aS)

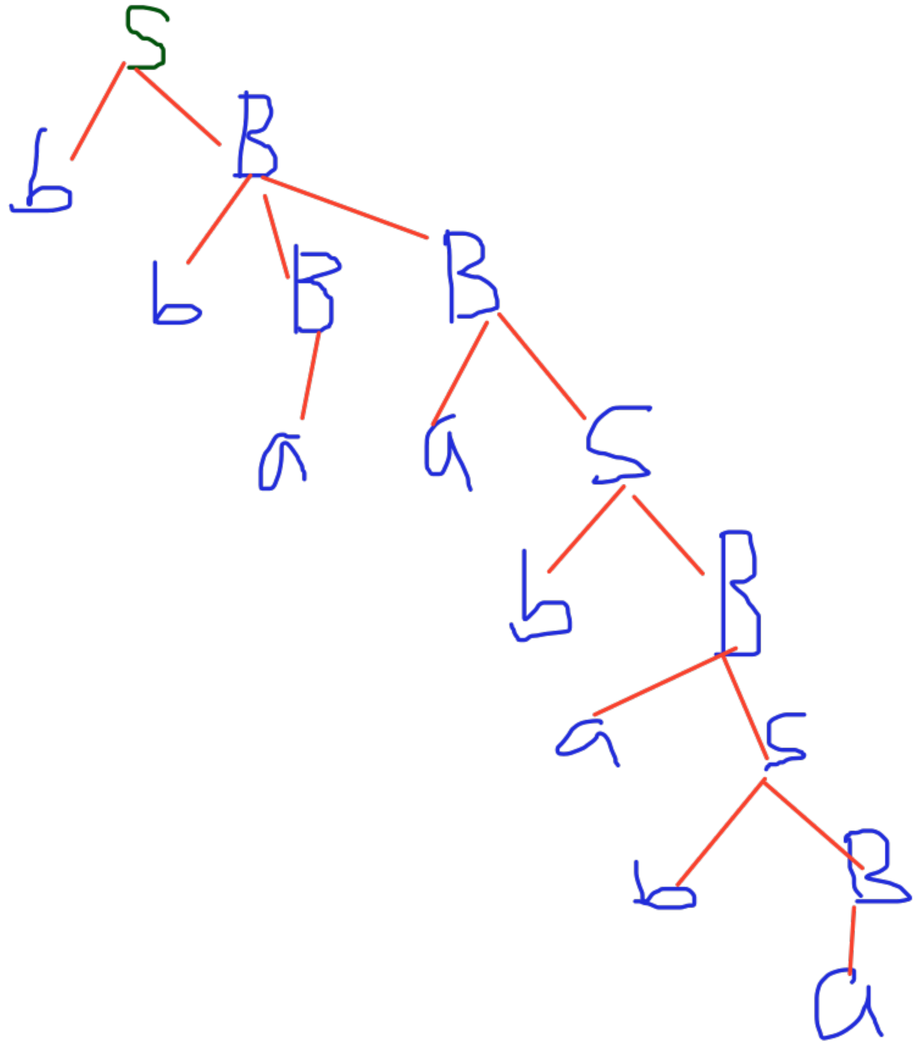
------>bbBabB (S-->bB)

------> bbBabaS (B-->aS)

-----> bbBababB (S--->bB)

-----> bbBababa (B--->a)

------> bbaababa (B--->a)



Problem-02:

Consider the grammar-

S → A1B

A → 0A / □

B → 0B / 1B / □

For the string w = 00101, find-

Leftmost derivation
Rightmost derivation
Parse Tree

S----> A1B

---->0A1B

---->00A1B

---> 001B (A--> ∈)

---> 0010B

---->00101B

----.>00101

LEFT

S ----->A1B

S----> A10B

S---->A101B

----->A101

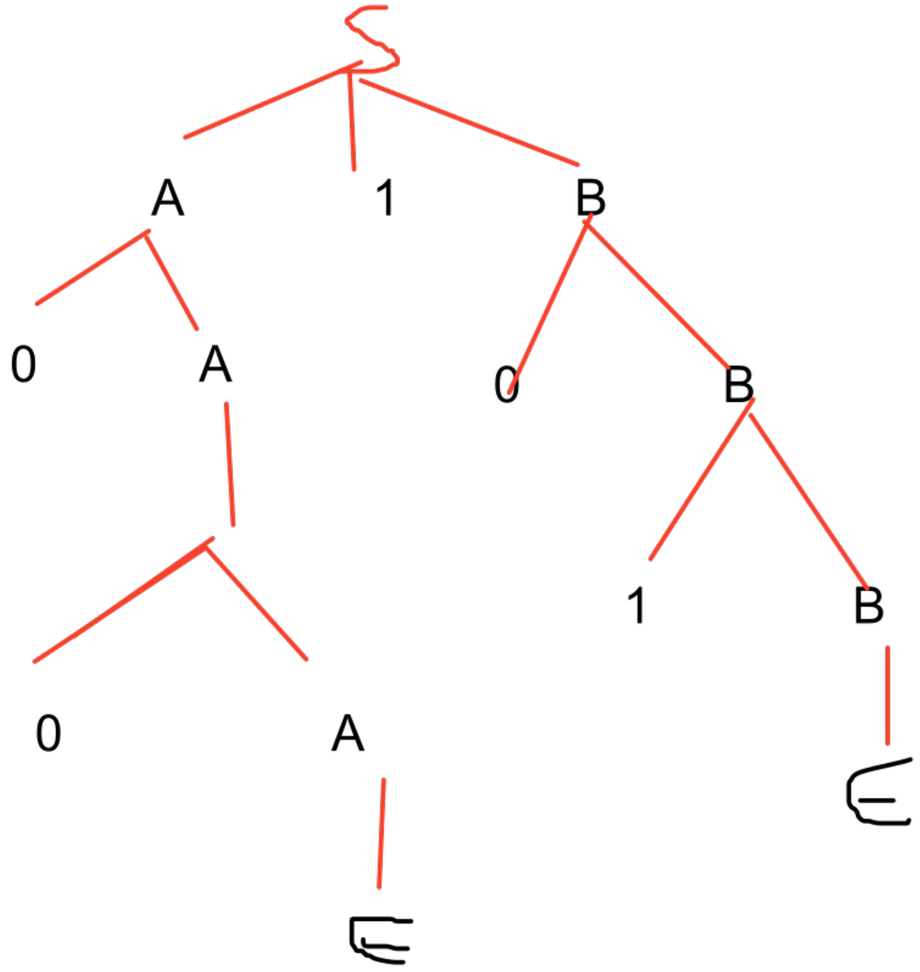
---->0A101

--->00A101

--->00101

RIGHT

PARSE TREE



Construct a CFG for a language L = {wcwR | where w € (a, b)*}.

.

LANGUAGE L ={aacbb,abcbb,abbcbba,aaacbbb....}

S ----->aSa

S-----> c

S----> bSb

S----->aSb

aacbb

S--->aSb

S----->aaSbb

S---->aacbb

Construct a CFG for the language L = a^nb^2n where n>=1.

.

l={abb,aabbbb,aaabbbbbb.....}

S----aSbb

S---aabbbb

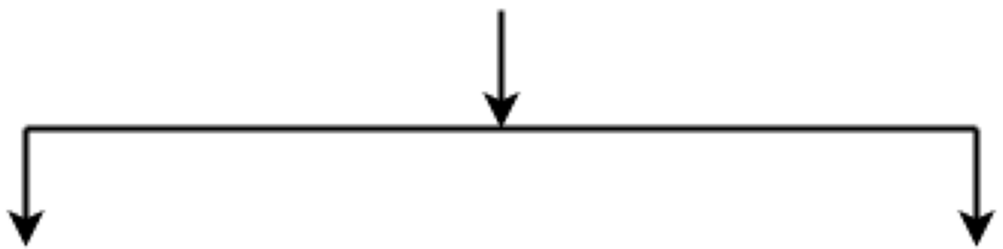
Ambiguity in Grammar

grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string. If the grammar is not ambiguous, then it is called unambiguous.

If the grammar has ambiguity, then it is not good for compiler construction.
No method can automatically detect and remove the ambiguity, but we can remove ambiguity by re-writing the whole grammar without ambiguity.

Types of Grammar

(On the basis of Number of derivation trees)



Ambiguous Grammar

A grammar is said to ambiguous
if for any string generated by it,
it produces more than one-

Parse tree
Or derivation tree
Or syntax tree
Or leftmost derivation
Or rightmost derivation

Unambiguous Grammar

A grammar is said to unambiguous
if for every string generated by it,
it produces exactly one-

Parse tree
Or derivation tree
Or syntax tree
Or leftmost derivation
Or rightmost derivation

more than one parse tree,syntax tree, left and right

Consider the following grammar-

E → E + E / E x E / id

Ambiguous Grammar

E--->E + E

----.id+ E

---- id + E * E

---> id + id * id

E ---> E * E

---id * E---id * E+E

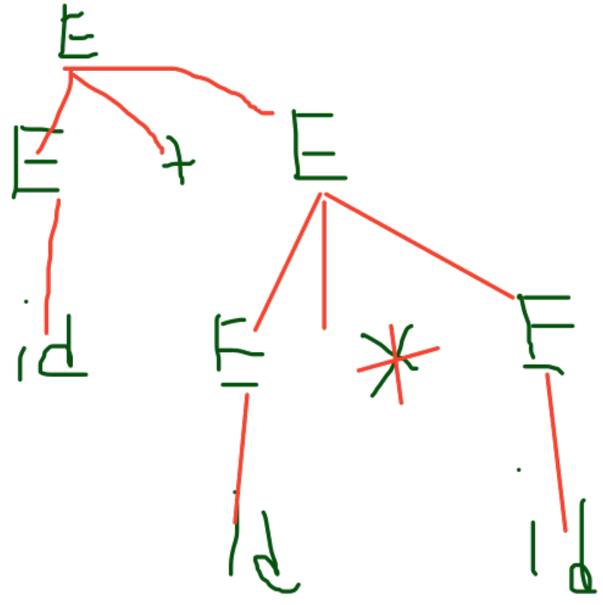
---- id * id + id

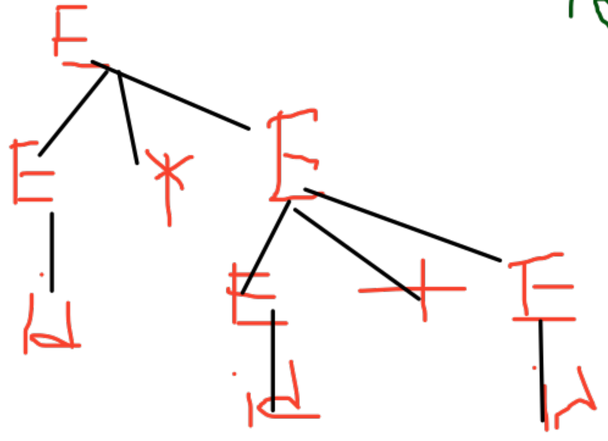
E ---> E + E

----.> E +id

--- E * E + id

---- id * id +id





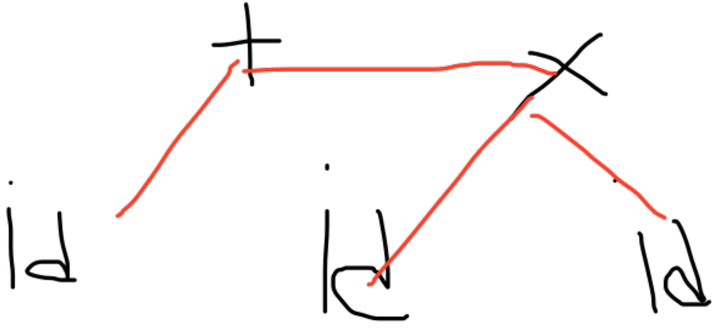
Consider the following grammar-

E → E + T / T

T → T x F / F

F → id

Unambiguous Grammar



Check whether the given grammar is ambiguous or not-

$S \rightarrow A / B$

$A \rightarrow aAb / ab$

$B \rightarrow abB / \square$

Check whether the given grammar is ambiguous or not-

$S \rightarrow AB / C$

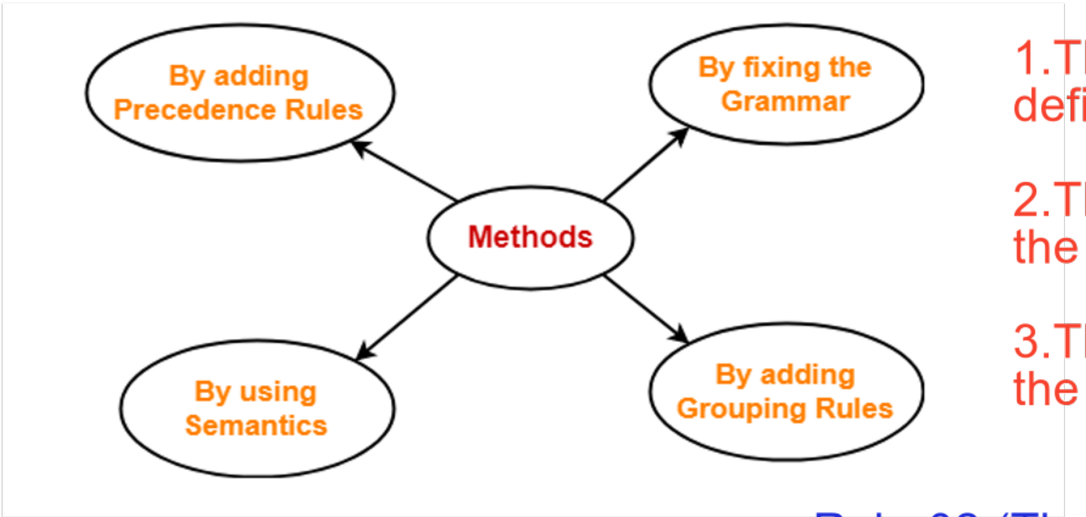
$A \rightarrow aAb / ab$

$B \rightarrow cBd / cd$

$C \rightarrow aCd / aDd$

$D \rightarrow bDc / bc$

Methods To Remove Ambiguity-



Removing Ambiguity By Precedence & Associativity Rules-

Rule-01:(The precedence constraint)

1.The level at which the production is present defines the priority of the operator contained in it.

2.The higher the level of the production, the lower the priority of operator.

3.The lower the level of the production, the higher the priority of operator.

Rule-02:(The associativity constraint)

If the operator is left associative, induce left recursion in its production.
If the operator is right associative, induce right recursion in its production.