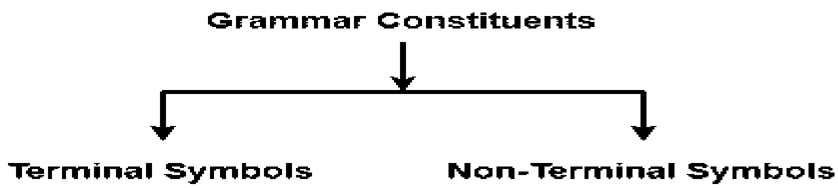
Grammar: (Context free grammar CFG) is a finite set of formal rules that are generating syntacticaly correct sentences. The formal definition of grammar is that it is defined as four tuples − G=(N,T,P,S)

- G is a grammar, which consists of a set of production rules. It is used to generate the strings of a language.
- T is the final set of terminal symbols. It is denoted by lower case letters.
- N is the final set of non-terminal symbols. It is denoted by capital letters.
- P is a set of production rules, which is used for replacing non-terminal symbols (on the left side of production) in a string with other terminals (on the right side of production).
- S is the start symbol used to derive the string.

Grammar is composed of two basic elements



Terminal Symbols - Terminal symbols are the components of the sentences that are generated using grammar and are denoted using small case letters like a, b, c . . . etc.

Non-Terminal Symbols - Non-Terminal Symbols take part in the generation of the sentence but are not the component of the sentence. These types of symbols are also called Auxiliary Symbols and Variables. They are represented using a capital letter like A, B, C, . . . etc.

Example 1

Consider a grammar **G = (N , T , P , S)** Where,

N = {S , A , B } ⇒ Non-Terminal symbols

T = {a, b} ⇒ Terminal symbols

Production rules P = {S → ABa , A → BB , B → ab , AA → b }

S = { S } ⇒ Start symbol

Example 2

Consider a grammar **G=(N,T,P,S)** Where,

N= {S, A, B} ⇒ non terminal symbols

T = {0,1} ⇒ terminal symbols

Production rules P = {S→A1B, A→0A| λ, B→0B| 1B| λ}

S= {S} ⇒ start symbol.

Derivation: is a sequence of production rules. It is used to get input strings. During parsing, we have to take two decisions, which are as follows

We have to decide the non-terminal which is to be replaced and We have to decide the production rule by which the non-terminal will be replaced.

Two options to decide which non-terminal has to be replaced with the production rule are as follows: −

-**Left most derivation**

In the leftmost derivation, the input is scanned and then replaced with the production rule from left side to right. So, we have to read that input string from left to right.

Example

Production rules: E=E+E (rule1), E=E-E (rule2), E=a|b (rule3)

Let the input be a-b+a

when we perform the Left Most Derivation, the result will be as follows: −

E=**E**+E

E=**E-E**+E from rule2

E=**a**-E+E from rule3

E=a-**b**+E from rule3

E=a-b+**a** from rule3

Finally, the given string is parsed

-**Right Most Derivation**

In Right most derivation, the input is scanned and replaced with the production rule right to left. So, we have to read the input string from right to left.

Example

Production rule: E=E+E (rule1), E=E-E (rule2), E=a|b (rule3)

Let the input be a-b+a

when we perform the Right Most Derivation, we get the following result: −

E=E-**E**

E=E-**E+E** from rule1

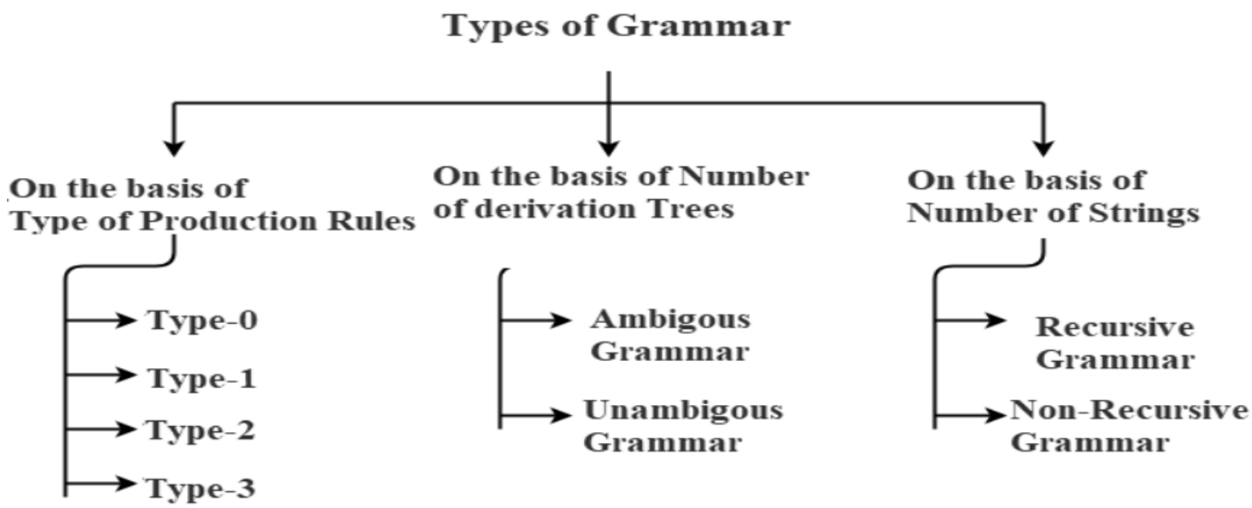
E=E-E+**a** from rule3

E=E-**b**+a from rule3

E=**a**-b+a from rule3

Types Of Grammars: Grammar can be divided onto: -

- Type of Production Rules
- Number of Derivation Trees
- Number of Strings



Chomsky Normal Form A CFG is in Chomsky Normal Form if the Productions are in the following forms :–

$A \to a$, $A \to BC$, $S \to \lambda$ where A, B, and C are non-terminals and **a** is terminal.

Algorithm to Convert into Chomsky Normal Form: –

Step 1 − If the start symbol **S** occurs on some right side, create a new start symbol **S'** and a new production **S'→ S**.

Step 2 − Remove Null productions. (Using the Null production removal algorithm discussed earlier)

Step 3 − Remove unit productions. (Using the Unit production removal algorithm discussed earlier)

Step 4 − Replace each production $\mathbf{A \to B_1 \dots B_n}$ where $\mathbf{n > 2}$ with $\mathbf{A \to B_1C}$ where $\mathbf{C \to B_2 \dots B_n}$. Repeat this step for all productions having two or more symbols in the right side.

Step 5 − If the right side of any production is in the form **A → aB** where a is a terminal and **A, B** are non-terminal, then the production is replaced by **A → XB** and **X → a**. Repeat this step for every production which is in the form **A → aB**.

Problem

Convert the following CFG into CNF

$S \to ASA \mid aB$, $A \to B \mid S$, $B \to b \mid \lambda$

Solution

(1) Since **S** appears in R.H.S, we add a new state S_0 and $S_0 \rightarrow S$ is added to the production set and it becomes −

$S_0 \rightarrow S$, $S \rightarrow ASA \mid aB$, $A \rightarrow B \mid S$, $B \rightarrow b \mid \lambda$

(2) Now we will remove the null productions −

$B \rightarrow \lambda$ and $A \rightarrow \lambda$

After removing $B \rightarrow \varepsilon$, the production set becomes −

$S_0 \rightarrow S$, $S \rightarrow ASA \mid aB \mid a$, $A \rightarrow B \mid S \mid \in$, $B \rightarrow b$

After removing $A \rightarrow \lambda$, the production set becomes −

$S_0 \rightarrow S$, $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA \mid S$, $A \rightarrow B \mid S$, $B \rightarrow b$

(3) Now we will remove the unit productions.

After removing $S \rightarrow S$, the production set becomes −

$S_0 \rightarrow S$, $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$, $A \rightarrow B \mid S$, $B \rightarrow b$

After removing $S_0 \rightarrow S$, the production set becomes −

$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$, $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow B \mid S$, $B \rightarrow b$

After removing $A \rightarrow B$, the production set becomes −

$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$, $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow S \mid b$

$B \rightarrow b$

After removing $A \rightarrow S$, the production set becomes −

$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$, $S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid ASA \mid aB \mid a \mid AS \mid SA$, $B \rightarrow b$

(4) Now we will find out more than two variables in the R.H.S

Here, $S_0 \rightarrow ASA$, $S \rightarrow ASA$, $A \rightarrow ASA$ violates two Non-terminals in R.H.S.

Hence, we will apply step 4 and step 5 to get the following final production set which is in CNF:-

$S_0 \rightarrow AX \mid aB \mid a \mid AS \mid SA$

$S \rightarrow AX \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid AX \mid aB \mid a \mid AS \mid SA$

B → b

X → SA

(5) We have to change the productions S_0→ aB, S→ aB, A→ aB

And the final production set becomes −

S_0→ AX | YB | a | AS | SA

S→ AX | YB | a | AS | SA

A → b A → b |AX | YB | a | AS | SA

B → b

X → SA

Y → a

Derivations and Parse Trees

Derivations mean replacing a given string's non-terminal by the right-hand side of the production rule. The sequence of applications of rules that makes the completed string of terminals from the starting symbol is known as derivation. The parse tree is the pictorial representation of derivations. Therefore, it is also known as derivation trees. The derivation tree is independent of the other in which productions are used.

A parse tree is an ordered tree in which nodes are labeled with the left side of the productions and in which the children of a node define its equivalent right parse tree also known as syntax tree, generation tree, or production tree.

A Parse Tree for a CFG G =(V,∑, P,S) is a tree satisfying the following conditions: −

- Root has the label S, where S is the start symbol.
- Each vertex of the parse tree has a label which can be a variable (V), terminal (Σ), or ε.
- If A → C_1, $C_{2,}$ . . . C_n is a production, then C_1, $C_{2,}$ . . . Cn are children of node labeled A.
- Leaf Nodes are terminal (Σ), and Interior nodes are variable (V).
- The label of an internal vertex is always a variable.
- If a vertex A has k children with labels A_1, A_2.......A_k, then A →A_1,A_2.......A_k will be production in context-free grammar G.

Yield − Yield of Derivation Tree is the concatenation of labels of the leaves in left to right ordering.

Example1 − If CFG has productions.

S → a A S | a

A → Sb A | SS | ba

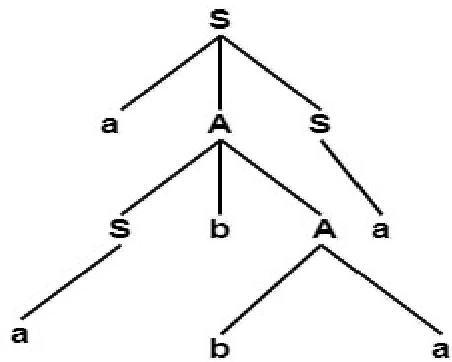
Show that S ⇒ aa bb aa & construct parse tree whose yield is aa bb aa.

Solution

S ⇒a A S
⇒ a SbAS
⇒ aabAS
⇒ aa bbaS

∴ S ⇒ aa bb aa

Derivation Tree



Yield = Left to Right Ordering of Leaves = aa bb aa

Example2: - Consider the CFG

S → bB | aA

A → b | bS | aAA

B → a |aS | bBB

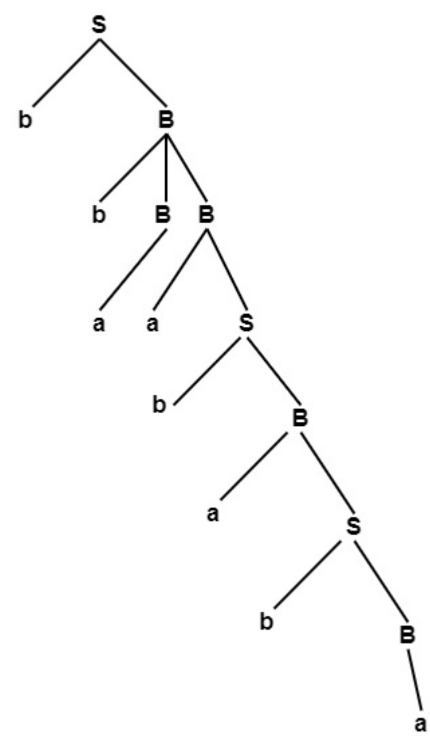
Find (a) Leftmost, (b) Rightmost Derivation for string b aa baba and (c) derivation Trees.
Solution

a) Leftmost Derivation

S⇒b **B**
⇒ bb **B**B
⇒ bba**B**
⇒ bbaa**S**
⇒ bbaa**bB**
⇒ bb aa b **aS**
⇒ bb aa bab **B**

⇒ bb aa ba ba

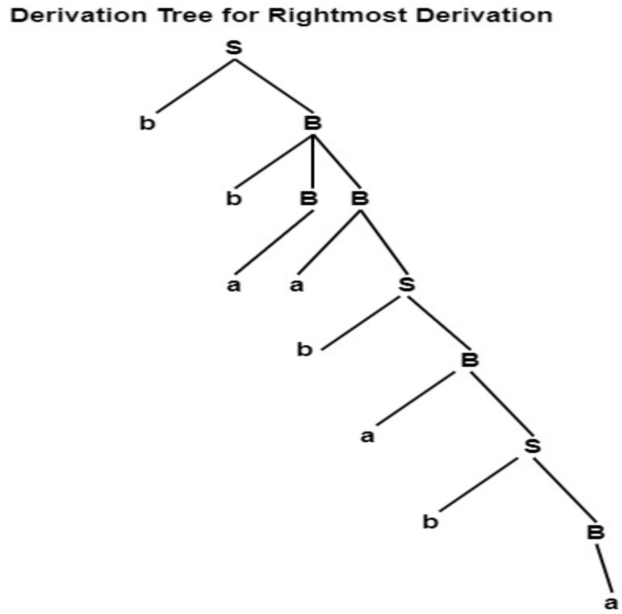
Derivation Tree for Leftmost Derivation



Basrah University
Faculty of Education for Pure Sciences
Computer Department

- **Rightmost Derivation**

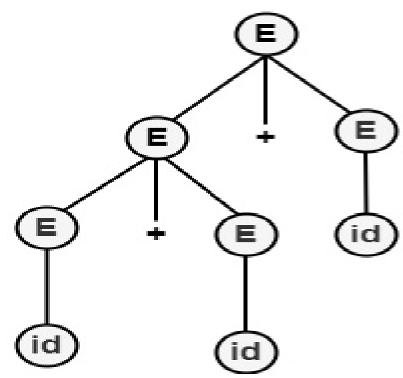
S ⇒ b**<u>B</u>**
⇒ bb B**<u>B</u>**
⇒ bbBa**<u>S</u>**
⇒ bbBab**<u>B</u>**
⇒ bbBaba**<u>S</u>**
⇒ bbBabab**<u>B</u>**
⇒ bb**<u>B</u>**abab a
⇒ bbaababa



Example3 − Consider the Grammar given below: −

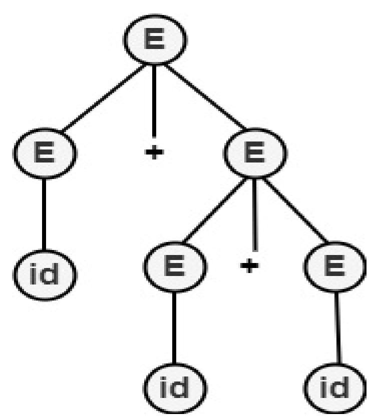
E⇒ E+E|E ∗E|id
Find: - Leftmost, -Rightmost Derivation for the string.

Solution
-Leftmost Derivation
E ⇒ **<u>E</u>**+E
⇒ **<u>E</u>**+E+E
⇒ id+E+E
⇒ id+id+E
⇒ id+id+id



- **Rightmost Derivation**

E ⇒ E+**<u>E</u>**
⇒ E+E+**<u>E</u>**
⇒ E+E+id
⇒ E+id+id
⇒ id+id+id



Basrah University
Faculty of Education for Pure Sciences
Computer Department

Ambiguity in Grammar: A grammar is said to be ambiguous if there exists more than one left most derivation or more than one right most derivation or more than one parse tree for a given input string.

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

Example: Let us consider a grammar with production rules, as shown below : −

E = I

E = E+E

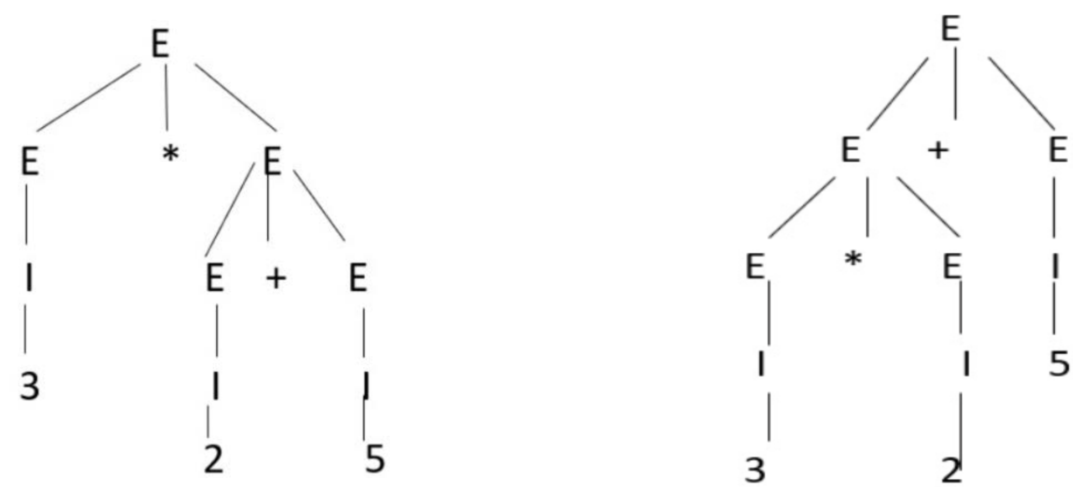
E = E*E

E = (E)

I = ε|0|1|2|3...9

Let's consider a string "3*2+5"

If the above grammar generates two parse trees by using Left most derivation (LMD) then, we can say that the given grammar is ambiguous grammar.



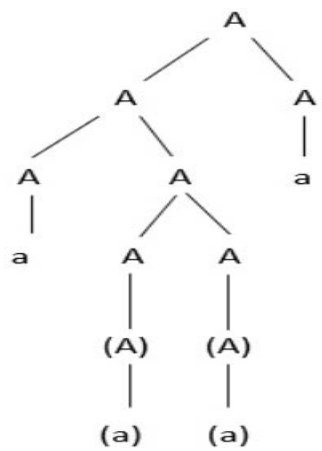
Since there are two parse trees for a single string, then we can say the given grammar is ambiguous grammar.

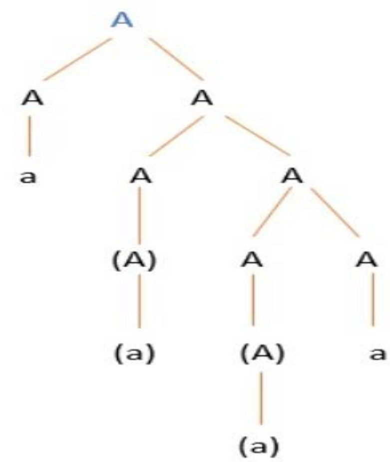
Example : Check whether the grammar is ambiguous or not.

A --> AA
A-->(A)
A-->a

For the string "a(a)(a)a" the above grammar can generate two parse trees, as given below: −

Basrah University
Faculty of Education for Pure Sciences
Computer Department





Rules: To convert the ambiguous grammar to the unambiguous grammar, we apply the following rules :–

Rule 1 – If the left associative operators (+ ,- , * , /) are used in the production rule, then apply left recursion in the production rule. Left recursion is nothing but left most symbol on the right side is the same as the non-terminal on the left side. For Example X--> Xa

Rule 2 – If the right associative operator (^) is used in the production rule then apply right recursion in the production rule.

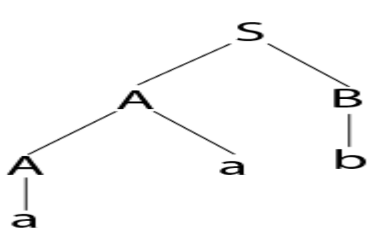
Right recursion is nothing but right most symbols on the left side are the same as the nonterminal on the right side. For example, X--> aX

Example: Consider a grammar G is given as follows:

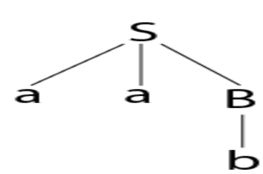
S → AB | aaB
A → a | Aa
B → b

Determine whether the grammar G is ambiguous or not. If G is ambiguous, construct an unambiguous grammar equivalent to G.

Solution: Let us derive the string "aab"



Parse tree 1



Parse tree 2

As there are two different parse trees for deriving the same string, the given grammar is ambiguous.

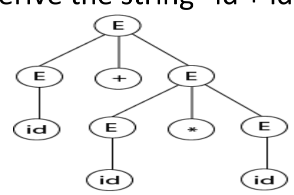
Unambiguous grammar will be:

S → AB

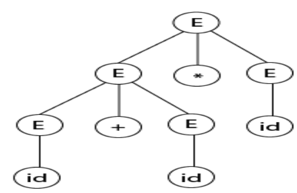
A → Aa | a

B → b

Example: Show that the given grammar is ambiguous. Also, find an equivalent unambiguous grammar.
E → E + E
E → E * E
E → id
Solution: Let us derive the string "id + id * id"



Parse tree 1



Parse tree 2

As there are two different parse trees for deriving the same string, the given grammar is ambiguous.
Unambiguous grammar will be:
E → E + T
E → T
T → T * F
T → F
F → id