## Generation of Derivation Tree

Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG. It is the simple way to show how the derivation can be done to obtain some string from a given set of production rules. The derivation tree is also called a **Parse tree**.

The derivation or the yield of a parse tree is the final string obtained by concatenating the labels of the leaves of the tree from left to right, ignoring the Nulls. However, if all the leaves are Null, derivation is Null.

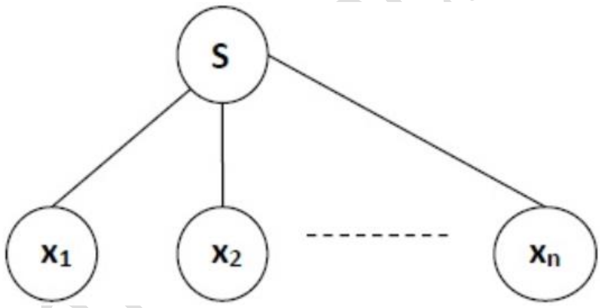
Parse tree follows the precedence of operators. The deepest sub-tree traversed first. So, the operator in the parent node has less precedence over the operator in the sub-tree.

## Representation Technique

A parse tree contains the following properties:

1. The root node is always a node indicating start symbols.
2. The derivation is read from left to right.
3. The leaf node is always terminal nodes.
4. The interior nodes are always the non-terminal nodes.

If $S \rightarrow x_1x_2\ldots\ldots x_n$ is a production rule in a CFG, then the parse tree/derivation tree will be as follows:



There are two different approaches to draw a derivation tree:

- **Top-down Approach:**
  - Starts with the starting symbol S.
  - Goes down to tree leaves using productions.
- **Bottom-up Approach:**
  - Starts from tree leaves.
  - Proceeds upward to the root which is the starting symbol **S.**

## Example

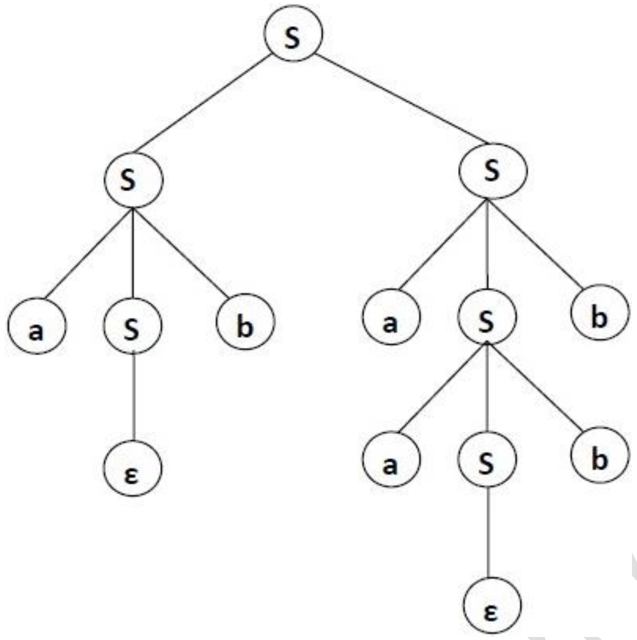
Let a CFG {N, T, P, S} be

N = {S},
T = {a, b},
Starting symbol = S,
P = $S \rightarrow SS \mid aSb \mid \varepsilon$

One derivation from the above CFG is "**abaabb**"

$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSb \Rightarrow abaaSbb \Rightarrow abaabb$

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.



**Leftmost and Rightmost Derivation of a String**

- **Leftmost derivation:** A leftmost derivation is obtained by applying production to the leftmost variable in each step.
- **Rightmost derivation:** A rightmost derivation is obtained by applying production to the rightmost variable in each step.

**Example**

Let any set of production rules in a CFG be,

$X \rightarrow X+X \mid X*X \mid X \mid a$

over an alphabet {a}.

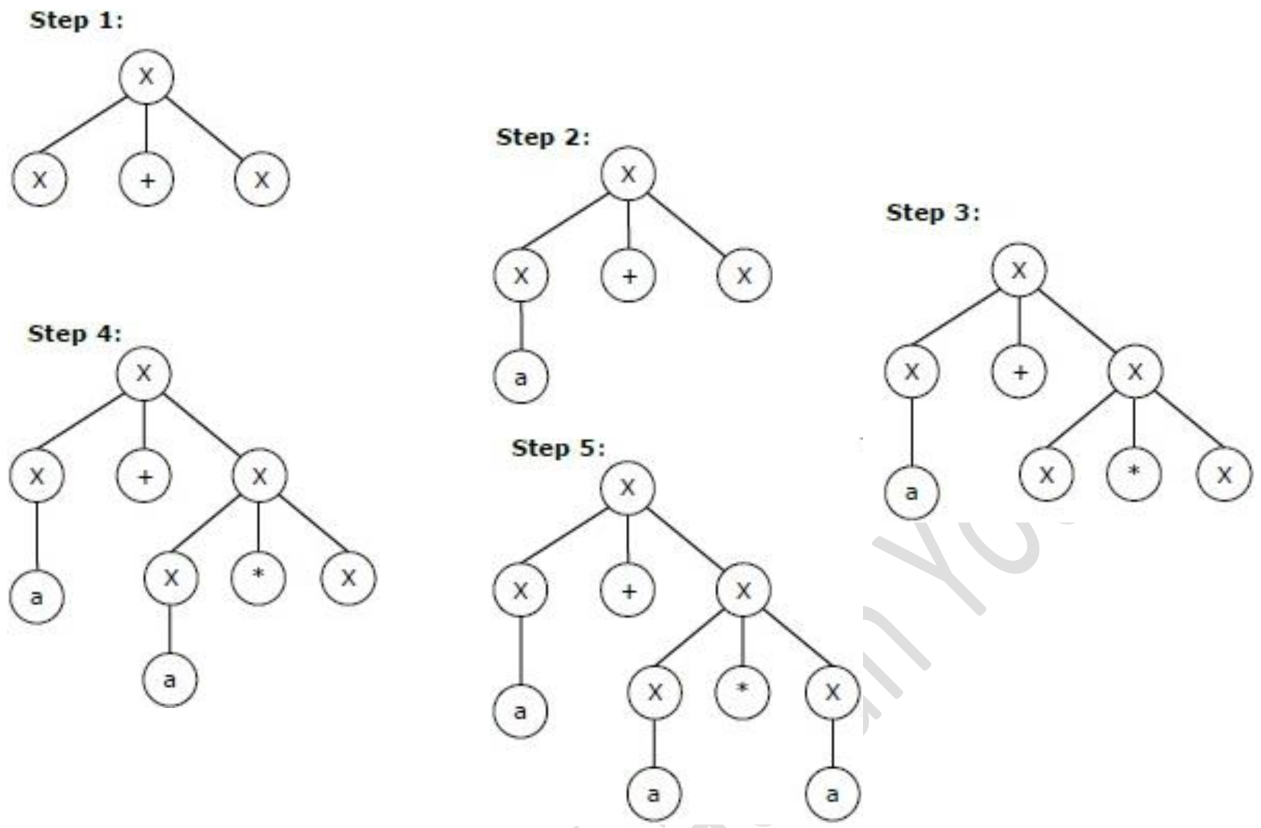
The **leftmost** derivation for the string **"a+a*a"** may be:

$X \Rightarrow X+X \Rightarrow a+X \Rightarrow a+X*X \Rightarrow a+a*X \Rightarrow a+a*a$

The stepwise derivation of the above string is shown as below:

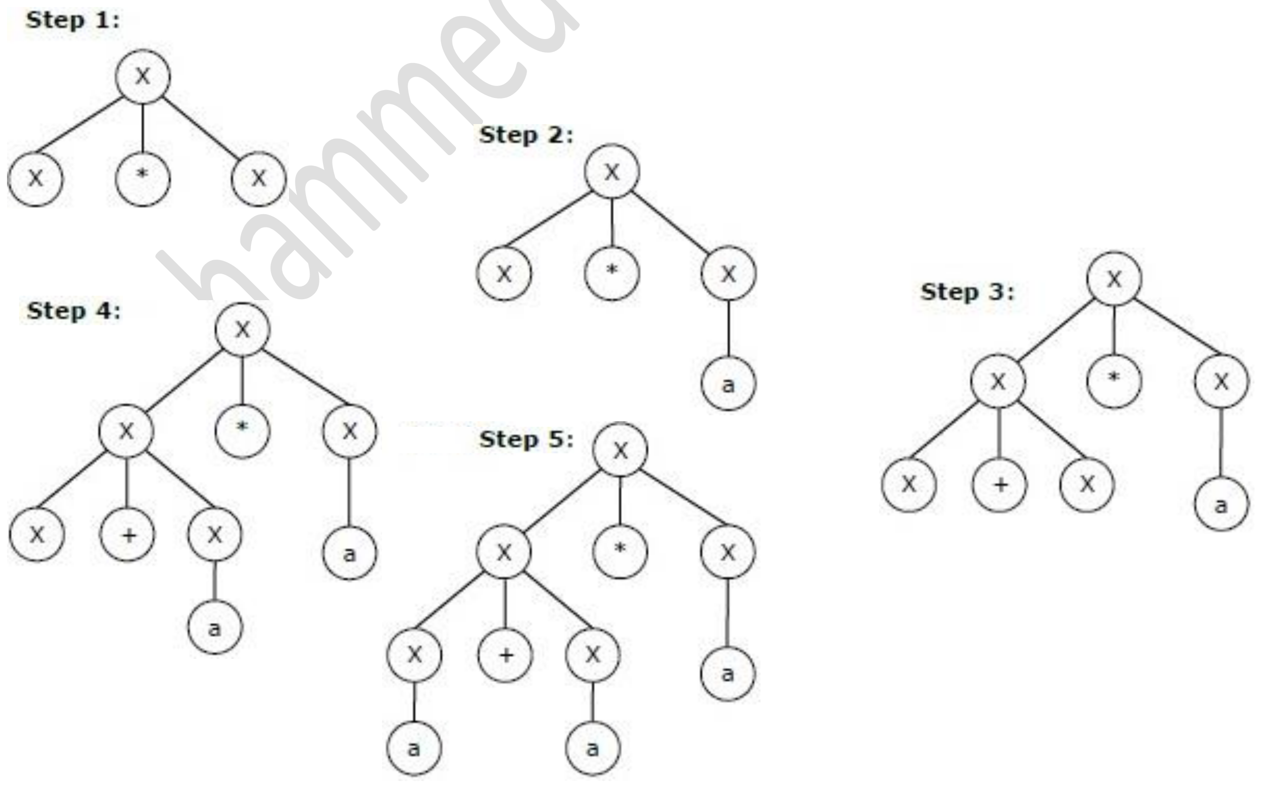
**Note:** We can draw a derivation tree step by step or directly in one step.

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.



The **rightmost** derivation for the above string **"a+a*a"** may be:
X ⇒ X*X ⇒ X*a ⇒ X+X*a ⇒ X+a*a ⇒ a+a*a

The stepwise derivation of the above string is shown as below:



(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.

**Example.** Write a CFG for the language:
L = {$a^i b^j c^{i+j}$ | i, j >=1},
and then find the derivation tree for the word in which i, j=2.

**Solution:**
L={$a^i b^j c^j c^i$ | i, j >=1}
G=({S, A}, {a, b, c}, S, P), where P:
S → aSc | aAc
A → bAc | bc
S ⇒ aSc ⇒ aaAcc ⇒ aabAccc ⇒ **aabbcccc**
***HW.*** *Find the derivation tree*

***HW 1.*** *Draw a derivation tree for the string "**bab**" from the CFG given by:*
S → bSb | a | b

***HW 2.*** *Construct a derivation tree for the string "**aabbabba**" for the CFG given by,*
S → aB | bA
A → a | aS | bAA
B → b | bS | aBB

***HW 3.*** *Show the derivation tree for string "**aabbbb**" with the following grammar:*
S → AB | ε
A → aB
B → Sb

***HW 4.*** *Show the derivation tree for string "**a*b+c**" with the following grammar:*
E → E + E | E * E | a | b | c

***HW 5.*** *Show the derivation tree for strings "**bccaabccc**" and "**aabcccbcc**". Consider the following grammar:* G = ({S, A, B}, {a, b, c}, S, P), where P:
S → ABc
A → aB | Bc
B → aAc | bc

***HW 6.*** *Show the derivation tree for string "**aaabbabbabbba**". Consider the following grammar:*
G = ({S, A, B}, {a, b, c}, S, P), where P:
S → aB | bA
A → aS | bAA | a
B → bS | b | aBB

## Left and Right Recursive Grammars

In a context-free grammar **G**, if there is a production in the form **X → Xa** where **X** is a non-terminal and **'a'** is a string of terminals, it is called a **left recursive production**. The grammar having a left recursive production is called a **left recursive grammar**.

And if in a context-free grammar **G**, if there is a production is in the form **X → aX** where **X** is a non-terminal and **'a'** is a string of terminals, it is called a **right recursive production**. The grammar having a right recursive production is called a **right recursive grammar**.

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.

## Ambiguity in Context-Free Grammars

If a context free grammar **G** has more than one derivation tree for some string **w ∈ L(G)**, it is called an **ambiguous grammar**. There exist multiple right-most or left-most derivations for some string generated from that grammar. In other words, a 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.

A common example of ambiguity in computer programming languages is the **dangling else** problem. In many languages, the **else** in an **If–then(–else)** statement is optional, which results in **nested conditionals** having multiple ways of being recognized in terms of the context-free grammar.
Concretely, in many languages one may write conditionals in two valid forms: the if-then form, and the if-then-else form – in effect, making the else clause optional:

In a grammar containing the rules:
Statement → **if** Condition **then** Statement |
**if** Condition **then** Statement **else** Statement |
...
Condition → ...

some ambiguous phrase structures can appear. The expression:
**if** a **then if** b **then** s **else** $s_2$
can be parsed as either:
**if** a **then begin if** b **then** s **end else** $s_2$
or as:
**if** a **then begin if** b **then** s **else** $s_2$ **end**
depending on whether the **else** is associated with the first **if** or second **if**.

This is resolved in various ways in different languages. Sometimes the grammar is modified so that it is unambiguous, such as by requiring an **endif** statement or making **else** mandatory. In other cases, the grammar is left ambiguous, but the ambiguity is resolved by making the overall phrase grammar context-sensitive, such as by associating an **else** with the nearest **if**. In this latter case the grammar is unambiguous, but the context-free grammar is ambiguous.

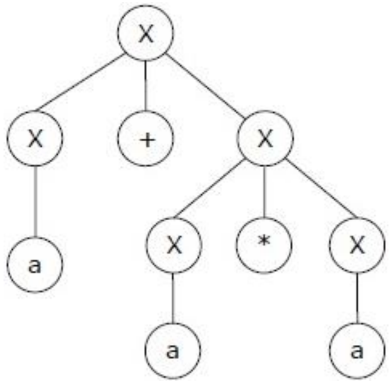
**Example 1.**
Check whether the grammar G with production rules:
X → X+X | X*X |X| a
is ambiguous or not.

**Solution**
Let's find out the derivation tree for the string "**a+a*a**". It has two leftmost derivations:

**Derivation 1**: X ⇒ X+X ⇒ a +X ⇒ a+X*X ⇒ a+a*X ⇒ **a+a*a**

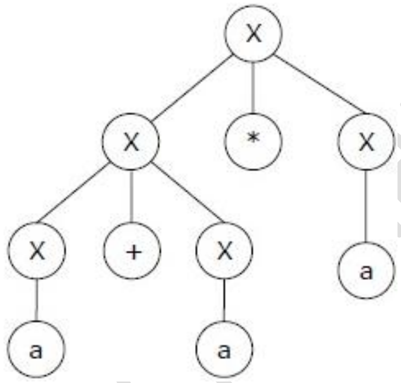
**Parse tree 1**:

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.



**Derivation 2**: X ⇒ X*X ⇒ X+X*X ⇒ a+X*X ⇒ a+a*X ⇒ **a+a*a**

**Parse tree 2**:



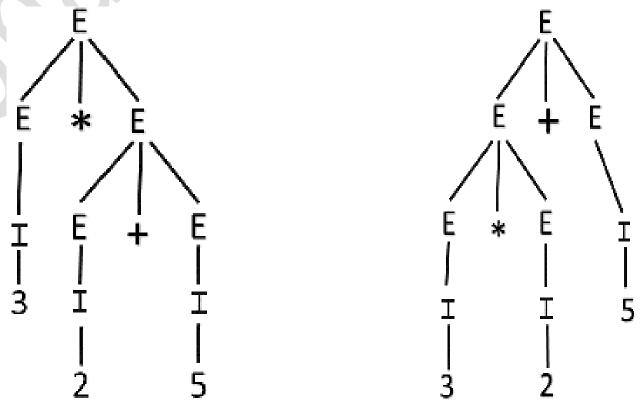
Since there are two parse trees for a single string "**a+a*a**", the grammar **G** is ambiguous.

**Example 2.**
Let us consider a grammar G with the production rule:

$E \rightarrow I \mid E+E \mid E*E \mid (E)$

$I \rightarrow \varepsilon \mid 0 \mid 1 \mid 2 \mid \dots \mid 9$

**Solution**
For the string "**3*2+5**", the above grammar can generate two parse trees by leftmost derivation:



Since there are two parse trees for a single string "**3*2+5**", the grammar G is ambiguous.

***HW.** Find the derivations for the above string.*

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.

**Example 3.**
Check whether the given grammar G is ambiguous or not:
E → E+E | E-E | id

**Solution**
From the above grammar String "**id+id-id**" can be derived in 2 ways:

**First Leftmost derivation**
E ⇒ E+E ⇒ id+E ⇒ id+E-E ⇒ id+id-E ⇒ **id+id-id**

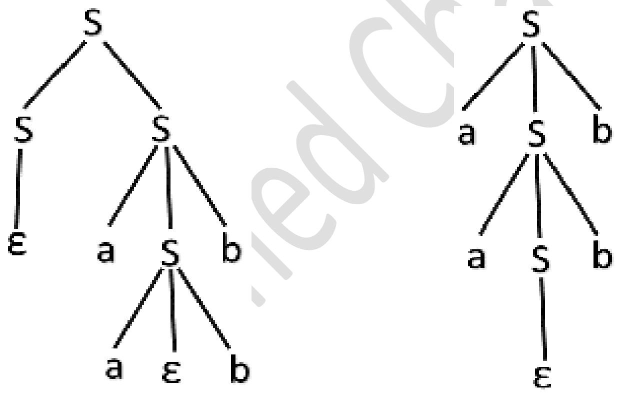
**Second Leftmost derivation**
E ⇒ E-E ⇒ E+E-E ⇒ id+E-E ⇒ id+id-E ⇒ **id+id-id**

Since there are two leftmost derivations for a single string "**id+id-id**", the grammar G is ambiguous.

***HW.*** *Draw the parse trees for the above string.*

**Example 4.**
Check whether the given grammar G is ambiguous or not:
S → aSb | SS | ε

**Solution**
For the string "**aabb**" the above grammar can generate two parse trees:



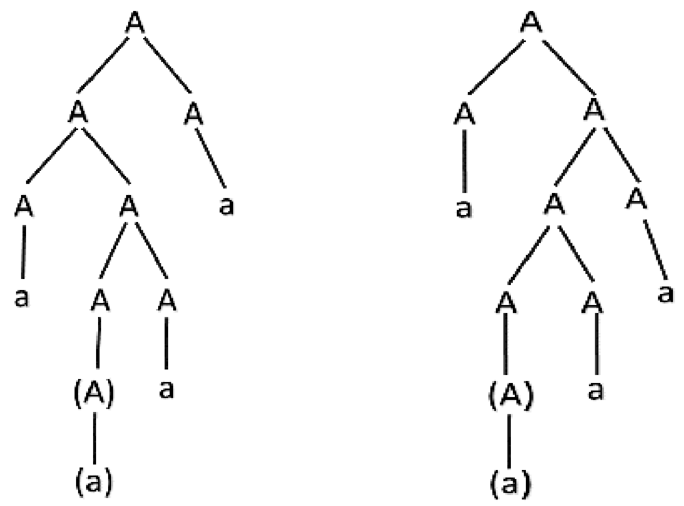
Since there are two parse trees for a single string "**aabb**", the grammar G is ambiguous.

***HW.*** *Find the derivations for the above string.*

**Example 5.**
Check whether the given grammar G is ambiguous or not:
A → AA | (A) | a

**Solution**
For the string "**a(a)aa**" the above grammar can generate two parse trees:

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.



Since there are two parse trees for a single string "**a(a)aa**", the grammar G is ambiguous.

***HW.*** *Find the derivations for the above string.*

***HW 1.*** *Check whether the given grammar G is ambiguous or not:*
*G=({S, A}, {a}, S, P), where P:*
$S \rightarrow AA$
$A \rightarrow aSa \mid a$

***HW 2.*** *For the string "**aabbab**", check whether the given grammar G is ambiguous or not:*
*G=({S, A, B}, {a, b}, S, P), where P:*
$S \rightarrow bA \mid aB$
$A \rightarrow a \mid aS \mid bAA$
$B \rightarrow b \mid bS \mid aBB$

***HW 3.*** *For the string "**a+a*a**", check whether the given grammar G is ambiguous or not:*
*G=({S}, {+, *, a}, S, P), where P:*
$S \rightarrow S+S \mid S*S \mid a$

***HW 4.*** *Consider the grammar G:*
*G=({S}, {a, b}, S, P), where P:*
$S \rightarrow aSbS \mid bSaS \mid \varepsilon$
*Show that this grammar is ambiguous by constructing different derivations for the sentence "**abab**".*

(*) أستاذ مساعد، قسم علوم الحاسوب، كلية علوم الحاسوب والرياضيات، جامعة الموصل.