Derivation

Derivation is a sequence of production rules. It is used to get the input string through these production rules. During parsing we have to take two decisions. These are as follows:

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

We have two options to decide which non-terminal to be replaced with production rule.

Left-most Derivation

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

Example:

Production rules:

1. S = S + S
2. S = S - S
3. S = a | b |c

Input:

a - b + c

The Left-most derivation is

S **=>** S - S + S **=>** a - S + S **=>** a - b + S **=>** a - b + c

Example:

Consider the following grammar-

S → aB / bA

S → aS / bAA / a

B → bS / aBB / b

Let us consider a string w = aaabbabbba

Leftmost Derivation-

S → a**B**

→ aa**B**B (Using B → aBB)

→ aaa**B**BB (Using B → aBB)

→ aaab**B**B (Using B → b)

→ aaabb**B** (Using B → b)

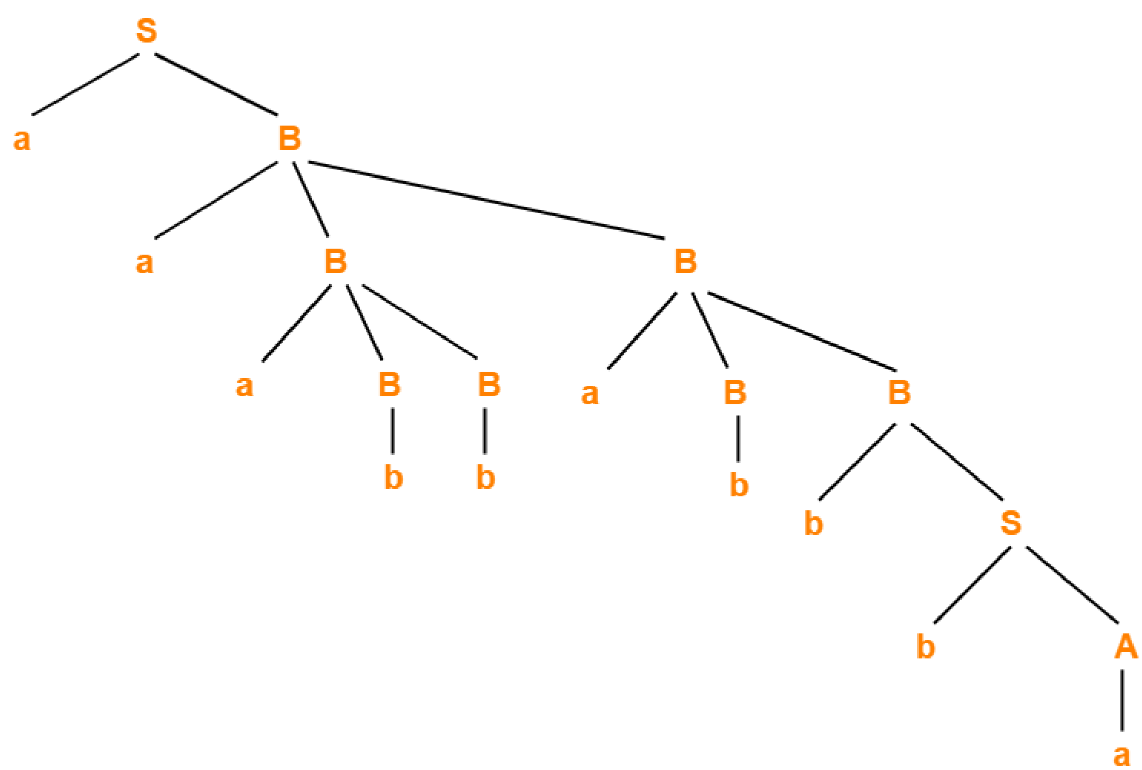
→ aaabba**B**B (Using B → aBB)

→ aaabbab**B** (Using B → b)

→ aaabbabb**S** (Using B → bS)

→ aaabbabbb**A** (Using S → bA)

→ aaabbabbba (Using A → a)



Leftmost Derivation Tree

Right-most Derivation

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

Example:

1. S = S + S
2. S = S - S
3. S = a | b |c

Input:

a - b + c

The right-most derivation is:

S = S - S => S - S + S => S - S + c => S - b + c => a - b + c

Example-

Consider the following grammar-

S → aB / bA

S → aS / bAA / a

B → bS / aBB / b

Let us consider a string w = aaabbabbba

Rightmost Derivation-

S → a**B**

→ aaB**B** (Using B → aBB)

→ aaBaB**B** (Using B → aBB)

→ aaBaBb**S** (Using B → bS)

→ aaBaBbb**A** (Using S → bA)

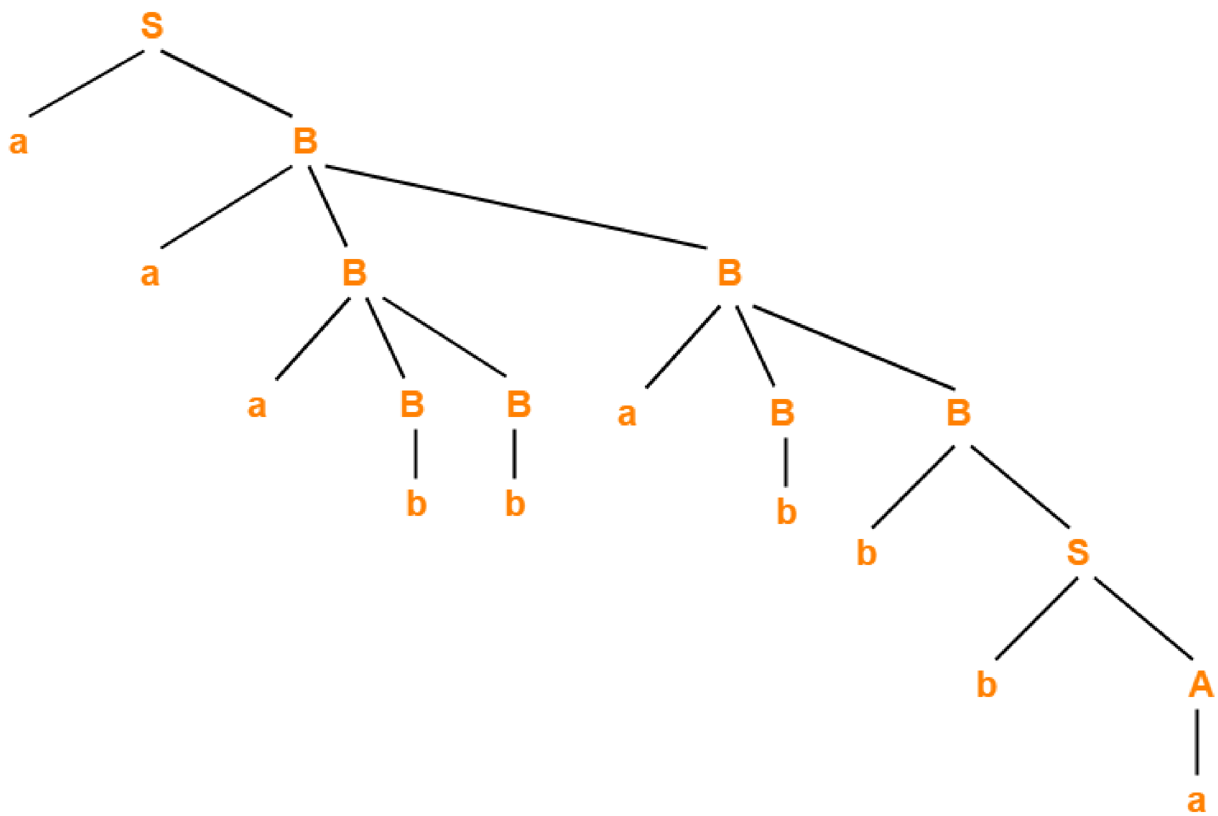
→ aaBa**B**bba (Using A → a)

→ aa**B**abbba (Using B → b)

→ aaaB**B**abbba (Using B → aBB)

→ aaa**B**babbba (Using B → b)

→ aaabbabbba (Using B → b)



Rightmost Derivation Tree

Parse tree

- Parse tree is the graphical representation of symbol. The symbol can be terminal or non-terminal.
- In parsing, the string is derived using the start symbol. The root of the parse tree is that start symbol.
- It is the graphical representation of symbol that can be terminals or non-terminals.
- 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.

The parse tree follows these points:

- All leaf nodes have to be terminals.
- All interior nodes have to be non-terminals.
- In-order traversal gives original input string.

Example:

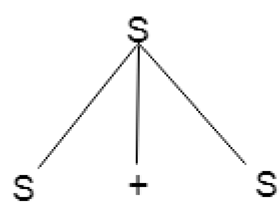
Production rules:

1. T= T + T | T * T
2. T = a|b|c

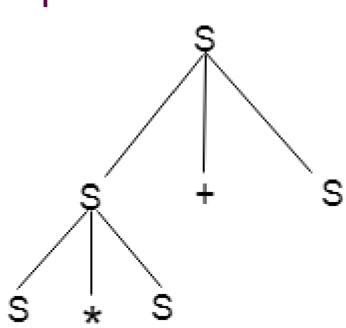
Input:

a * b + c

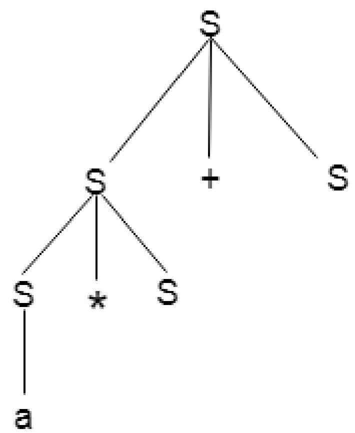
Step 1:



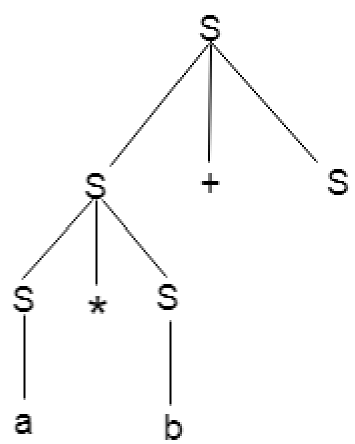
Step 2:



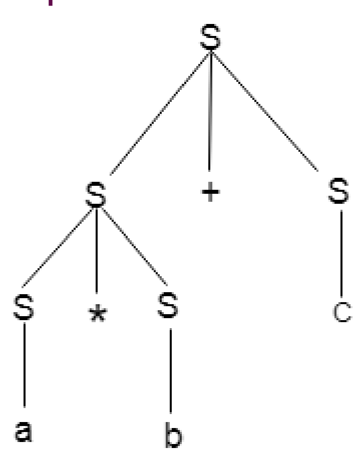
Step 3:



Step 4:



Step 5:



PRACTICE PROBLEMS BASED ON DERIVATIONS AND PARSE TREE-

Problem-01:

Consider the grammar-

S → bB / aA

A → b / bS / aAA

B → a / aS / bBB

For the string w = bbaababa, find-

1. Leftmost derivation
2. Rightmost derivation
3. Parse Tree

Solution-

1. Leftmost Derivation-

S → b**B**

→ bb**B**B (Using B → bBB)

→ bba**B** (Using B → a)

→ bbaa**S** (Using B → aS)

→ bbaab**B** (Using S → bB)

→ bbaaba**S** (Using B → aS)

→ bbaabab**B** (Using S → bB)

→ bbaababa (Using B → a)

2. Rightmost Derivation-

S → b**B**

→ bbB**B** (Using B → bBB)

→ bbBa**S** (Using B → aS)

→ bbBab**B** (Using S → bB)

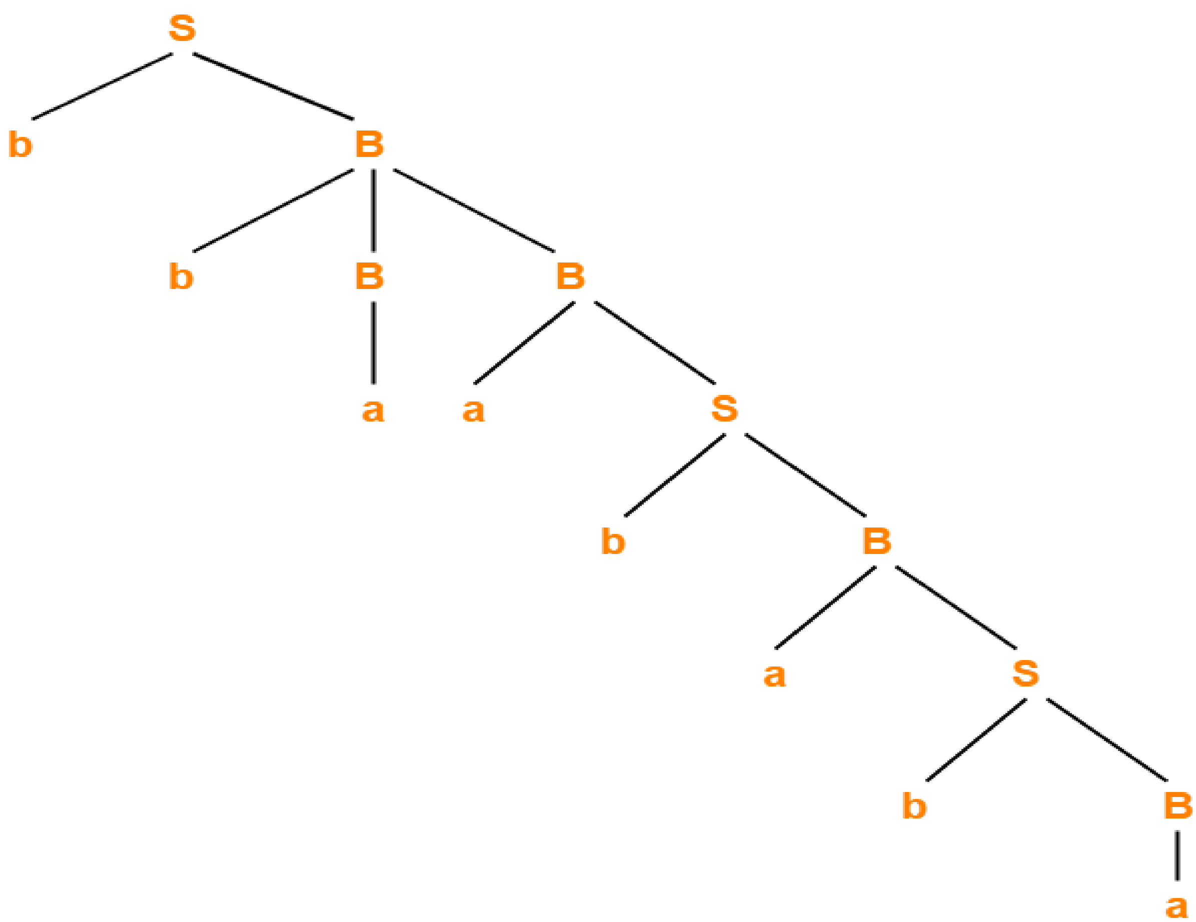
→ bbBaba**S** (Using B → aS)

→ bbBabab**B** (Using S → bB)

→ bb**B**ababa (Using B → a)

→ bbaababa (Using B → a)

3. Parse Tree-



Parse Tree

- Whether we consider the leftmost derivation or rightmost derivation, we get the above parse tree.
- The reason is given grammar is unambiguous.

Problem-02:

Consider the grammar-

$$S \rightarrow A1B$$

$$A \rightarrow 0A \mid \in$$

$$B \rightarrow 0B \mid 1B \mid \in$$

For the string w = 00101, find-

1. Leftmost derivation
2. Rightmost derivation
3. Parse Tree

Solution-

1. Leftmost Derivation-

S → **A**1B

→ 0**A**1B (Using A → 0A)

→ 00**A**1B (Using A → 0A)

→ 001**B** (Using A → ∈)

→ 0010**B** (Using B → 0B)

→ 00101**B** (Using B → 1B)

→ 00101 (Using B → ∈)

2. Rightmost Derivation-

S → A1**B**

→ A10**B** (Using B → 0B)

→ A101**B** (Using B → 1B)

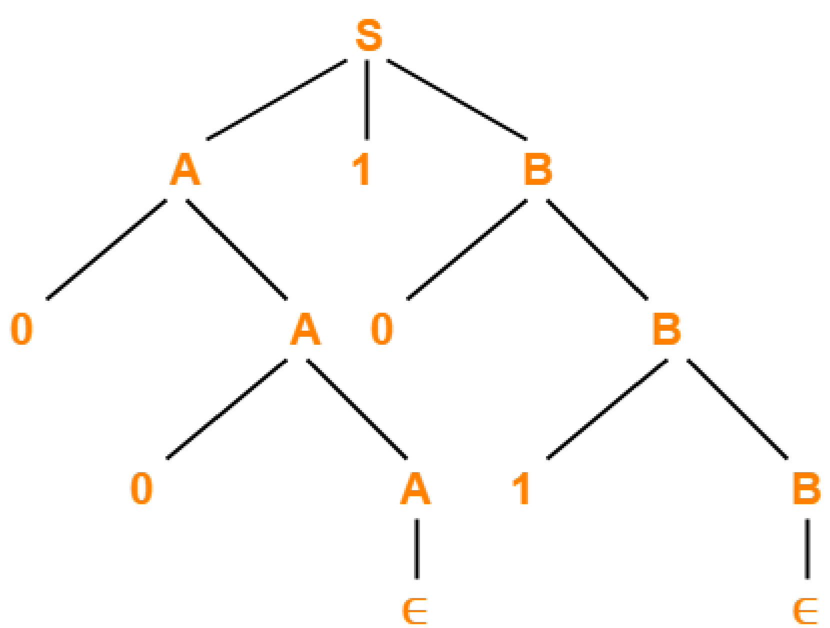
→ **A**101 (Using B → ∈)

→ 0**A**101 (Using A → 0A)

→ 00**A**101 (Using A → 0A)

→ 00101 (Using A → ∈)

3. Parse Tree-



Parse Tree

- Whether we consider the leftmost derivation or rightmost derivation, we get the above parse tree.
- The reason is given grammar is unambiguous.