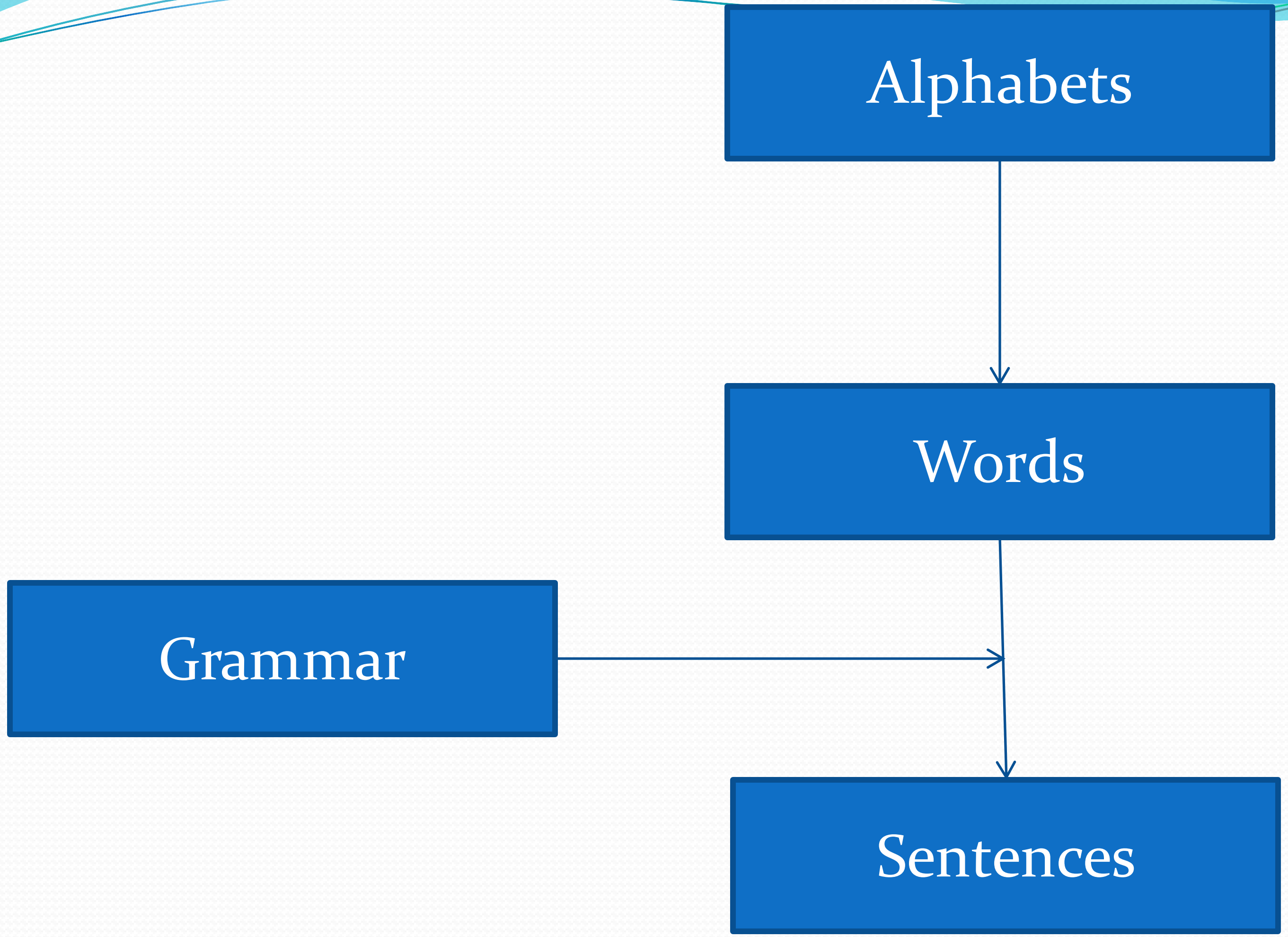
Parse Trees

--Sakshi Surve

Basics of Grammar :

- A language is set of strings over a set of symbols
  - Language --- Finite set of sentences
  - Sentence --- Finite set of words
  - Words --- Finite set of Alphabets / Symbols
- Grammar is essential to give syntactical structure to the language
- Grammar is the set of rules used to describe string of Language
- The language may be Programming language or natural language …Any type will require grammar


Alphabets
Words
Grammar
Sentences

Example :

- If we want English statement "Dog Runs" ....We may use following rules :
    - <Sentence> → <Noun> <Verb>
    - <Noun> → Dog
    - < Verb > → Runs

- <Sentence> → <Noun> <Verb>
  → Dog Runs

- These rules indicate how the sentence of the form 'Noun' followed by 'Verb' can be generated.

- There are many such rules of the language and they are collectively called the **Grammar** for the language

Constituents of Grammar :

- **Two Symbols :**
  - Terminals
  - Non Terminals

- **Terminals** are part of the generated sentence
  - E.g. In the above example, ‘**Dog**’ and ‘**Runs**’ are terminal symbols as they collectively formulate the statement and are part of the statement

- **Non Terminals** take part in the formation of the statement , but are not part of the generated sentence.

- No statement that is generated using grammar will contain Non Terminals in it.
  - E.g. In above example , '**Sentence**' , '**Noun**' , '**Verb**' are Non-terminals…which are not in the generated statement but took part on the formation of the statement
- <Sentence> → <Noun> <Verb>

  → Dog Runs

- Thus, Non-terminals are essential while declaring the rules
- These rules are called as '**Productions**' or '**Production Rules**'

Formal Definition :

- Like a natural language has Constituents like Nouns, Verbs, Adjectives etc….

- **Two Constituents :**
  - Terminals
  - Non terminals

- This grammar that is based on Constituent structure is called **Constituent Structure Grammar** Or **Phrase Structure Grammar**

- **The idea is …Basing a grammar on Constituent structure blocks**

Summary :

- If we want English statement “Dog Runs” ....We may use following rules :
  - <Sentence> → <Noun> <Verb>
  - <Noun> → Dog
  - < Verb > → Runs

- We have to begin with rule ....<Sentence> → <Noun> <Verb>
- **Start Symbol** ....Sentence
- **Non Terminals**.....Sentence, Noun, Verb
- **Terminals**.......Dog ,Runs
- **Rules**......indicating how the sentence can be generated.

Grammar :

A Grammar G is a four tuple collection **G = (V, T, P, S)** ....Where

- **V** is the (finite) set of variables or **Non terminals** ....They take part in the derivation , but are not part of the derived sentence
- **T** is a finite set of **Terminals**, i.e., the symbols that form the strings of the language being defined
- **P** is a finite set of **Production Rules**
- **S** is the **Start Symbol** (One of the Non terminals)

- **In the example before :**

- S = Sentence
- V = { Noun , Verb}
- T = {Dog , Runs }
- P

  <Sentence> → <Noun> <Verb>

  <Noun> → Dog

  < Verb > → Runs

Example :

- G = { S, V, P, T }

- T = { Man, Book, Reads, The }
- V = { N, V ,A}
- S = S

- P

  S → ANVN

  A → A | An | The

  N → Man | Book

  V → Reads

Derive string " The Man Reads Book"

Derivation ....

- S→ ANVN

  →The NVN

  → The Man VN

  → The Man Reads N

  → The Man Reads Book

Leftmost Derivation

S → ANVN
A → A | An | The
N → Man | Book
V → Reads

The Man Reads Book

Derivation ....

- S→ ANVN

  → ANV Book

  → AN Reads Book

  → A Man Reads Book

  → The Man Reads Book

S → ANVN
A → A | An | The
N → Man | Book
V → Reads

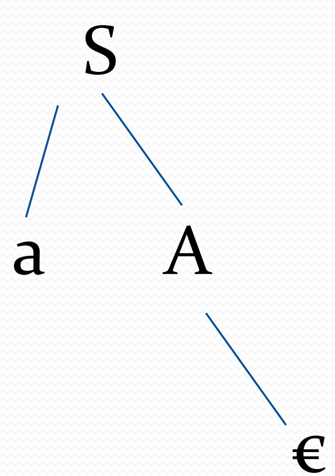
The Man Reads Book

Rightmost Derivation

- $\sum = \{ a , b \}$
- L = { w € L | w begins with a }
- L = { a , aa, ab, aab, aba, aaa, ………}

- S -> aA

  A -> aA | bA | €
- For ‘a’



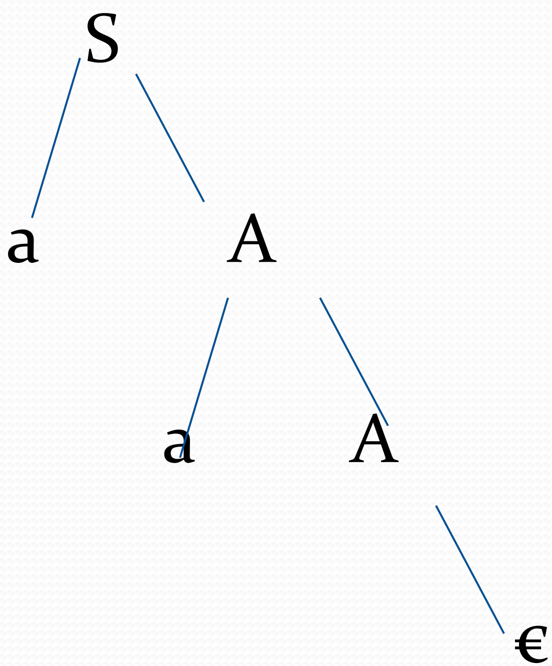
Derivation of ‘a’

S →aA

→a

S -> aA

A -> aA | bA | €

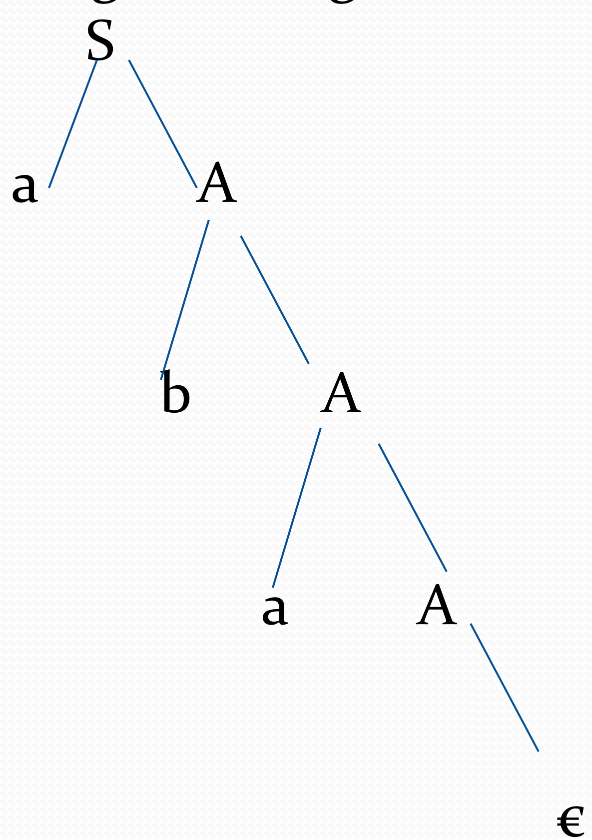
- For generating 'aa'
- 

Derivation of 'aa'

S → aA
→ aaA
→ aa

S -> aA

A -> aA | bA | €

- For generating 'aba'
- 

Derivation of 'aba'

S →aA
→abA
→abaA
→aba

In this grammar, the tuples are :

V = {S, A}
T = { a,b }
P
S = S

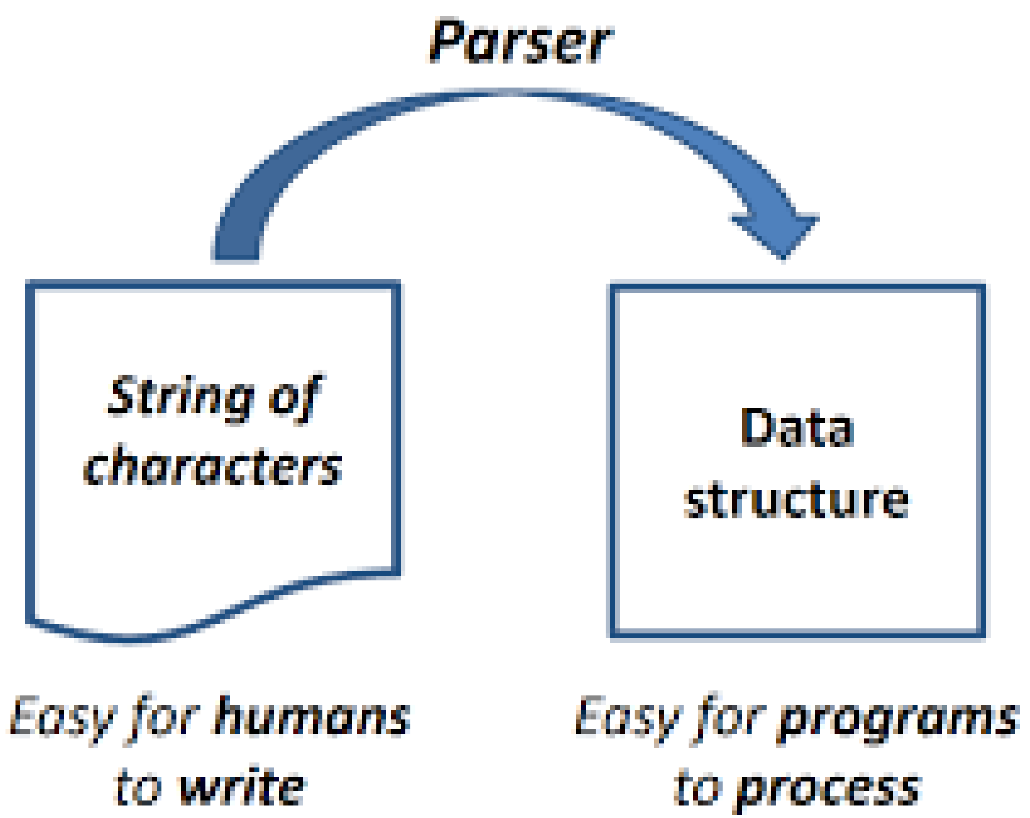
Parser :

- Parser is a component of a Compiler or Interpreter that breaks data into smaller elements

- Parser takes the input in the form of a sequence of tokens and builds a data structure in the form of a tree called **Parse tree**

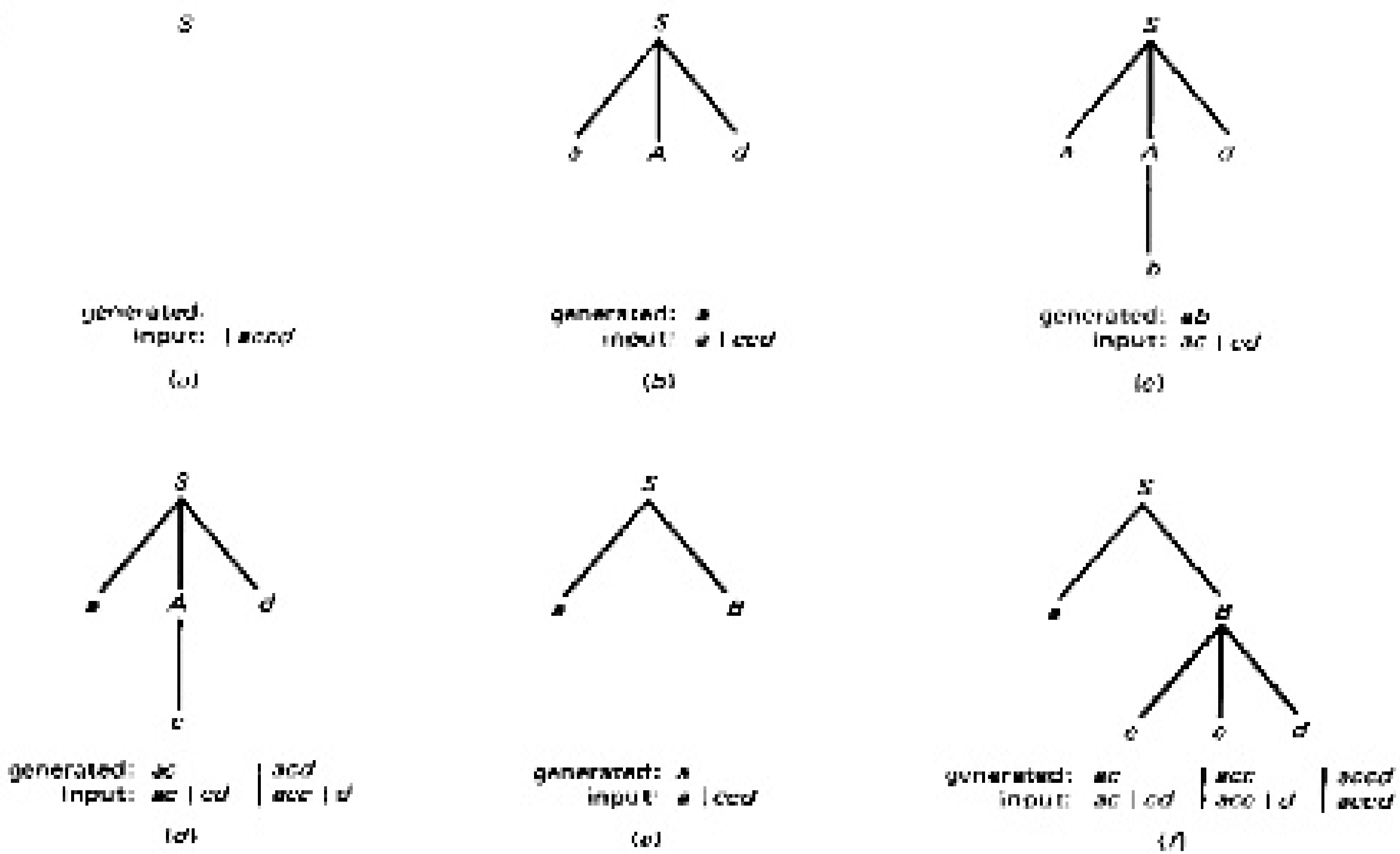
- Deriving a Syntactic tree like structure from the stream of tokens is called Parsing

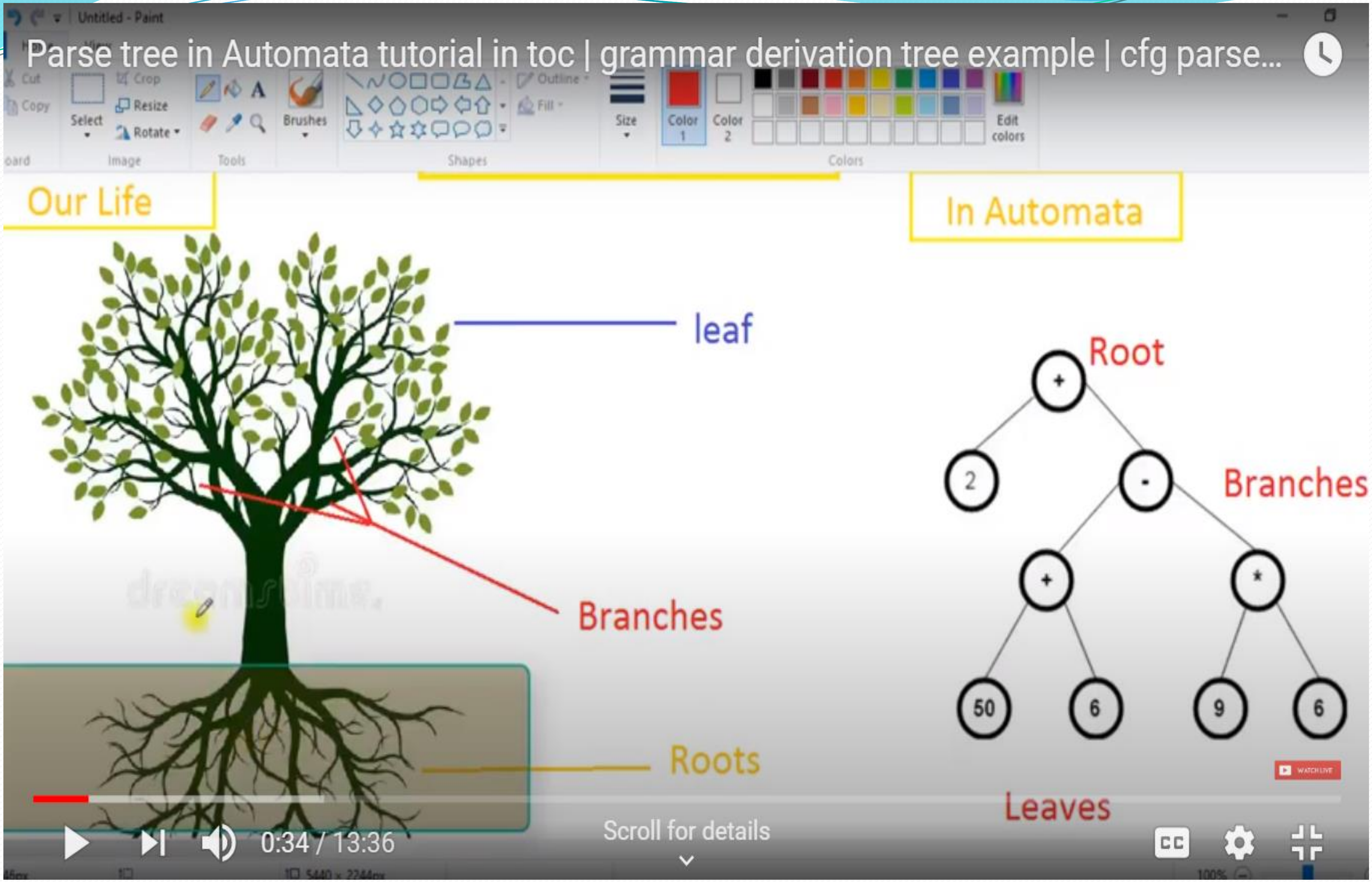
- Parsing is a process of determining if a string of tokens can be generated by a grammar

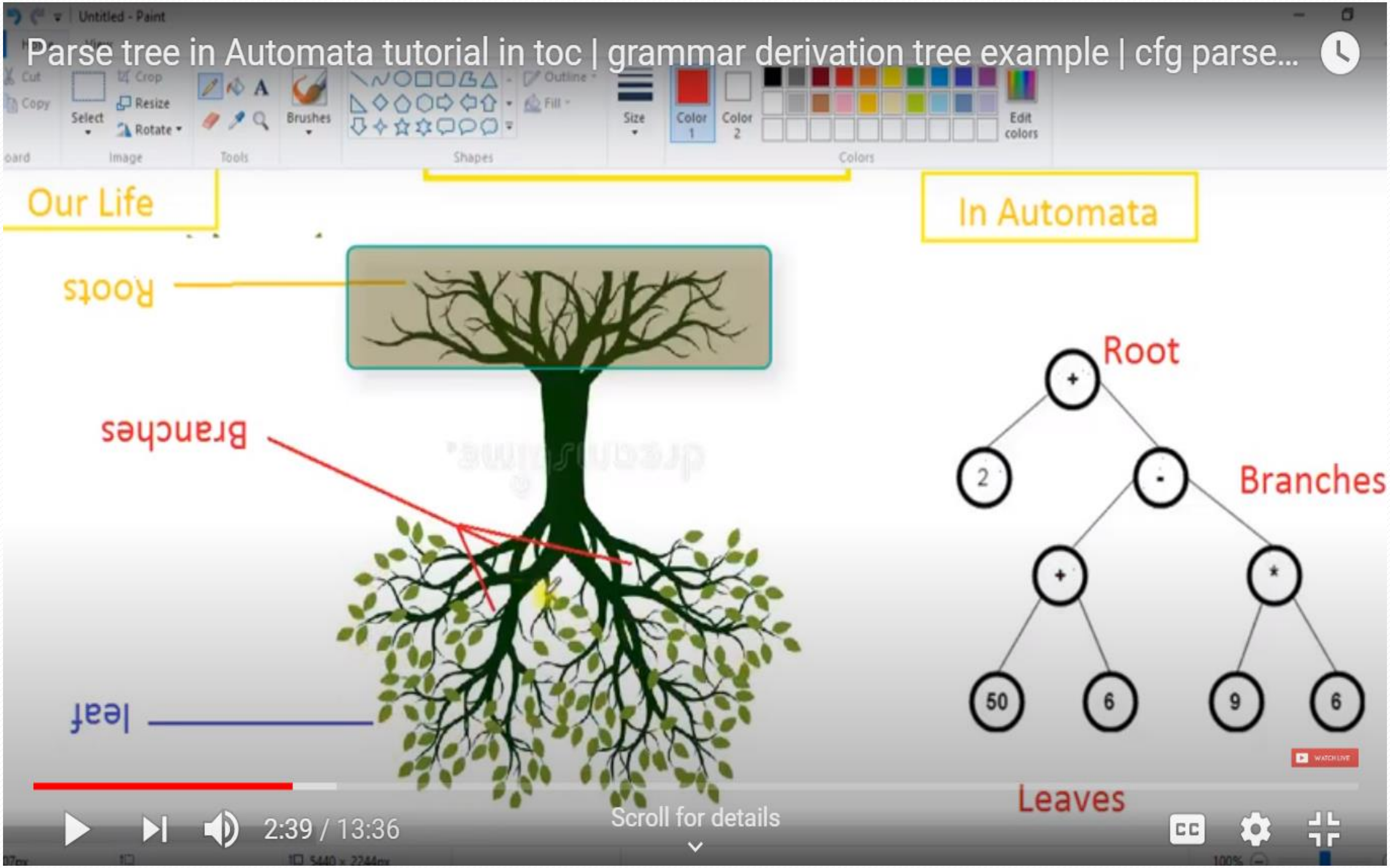

Parser
String of characters
Data structure
Easy for humans to write
Easy for programs to process

EXAMPLE FOR TOP DOWN PARSING

- Supppose the given production rules are as follows:
- S-> aAd|aB
- A-> b|c
- B->ccd



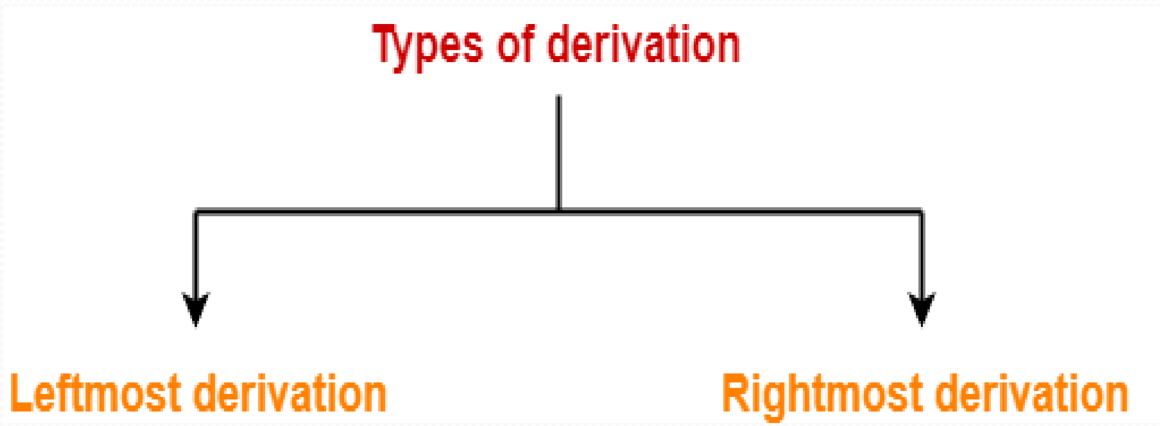

Untitled - Paint
Parse tree in Automata tutorial in toc | grammar derivation tree example | cfg parse...
Cut
Copy
Select
Crop
Resize
Rotate
Image
Tools
Brushes
Shapes
Outline
Fill
Size
Color 1
Color 2
Edit colors
Colors
Our Life
In Automata
leaf
Root
+
2
-
Branches
+
*
50
6
9
6
Branches
Roots
Leaves
0:34 / 13:36
Scroll for details
CC


Parse tree in Automata tutorial in toc | grammar derivation tree example | cfg parse...
Our Life
In Automata
Roots
Branches
leaf
Root
Branches
Leaves
+
2
-
+
*
50
6
9
6
2:39 / 13:36
Scroll for details

Parse Tree :

- The process of deriving a string using grammar rules is called as **derivation**.

- The geometrical representation of a derivation is called as a **parse tree** or **derivation tree.**



Leftmost Derivation

- The process of deriving a string by expanding the leftmost non-terminal at each step is called as **leftmost derivation**.

- The geometrical representation of leftmost derivation is called as a **leftmost derivation tree**.

Example 1 :

- Consider the following example :

$$S \rightarrow aB \;/\; bA$$
$$A \rightarrow aS \;/\; bAA \;/\; a$$
$$B \rightarrow bS \;/\; aBB \;/\; b$$

- Let us consider a string

$$w = aaabbabbba$$

- Now, let us derive the string w using leftmost derivation.

Leftmost Derivation -

$S \rightarrow aB / bA$
$A \rightarrow aS / bAA / a$
$B \rightarrow bS / aBB / b$

aaabbabbba

$S \rightarrow a\mathbf{B}$

$\rightarrow aa\mathbf{B}B$ (Using $B \rightarrow aBB$)

$\rightarrow aaa\mathbf{B}BB$ (Using $B \rightarrow aBB$)

$\rightarrow aaab\mathbf{B}B$ (Using $B \rightarrow b$)

$\rightarrow aaabb\mathbf{B}$ (Using $B \rightarrow b$)

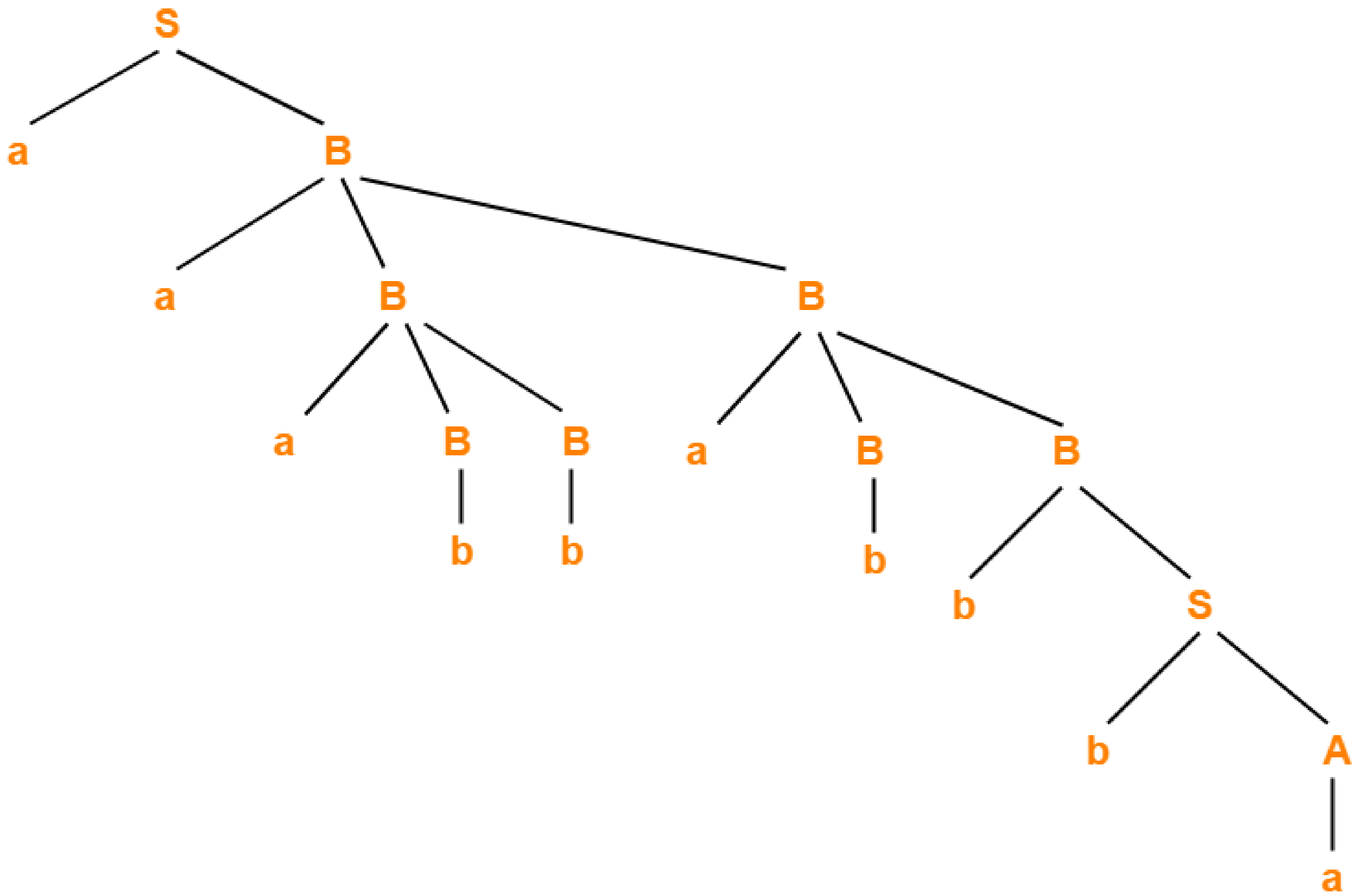
$\rightarrow aaabba\mathbf{B}B$ (Using $B \rightarrow aBB$)

$\rightarrow aaabbab\mathbf{B}$ (Using $B \rightarrow b$)

$\rightarrow aaabbabb\mathbf{S}$ (Using $B \rightarrow bS$)

$\rightarrow aaabbabbb\mathbf{A}$ (Using $S \rightarrow bA$)

$\rightarrow aaabbabbba$ (Using $A \rightarrow a$)



Leftmost Derivation Tree

Rightmost Derivation.

- The process of deriving a string by expanding the rightmost non-terminal at each step is called as **rightmost derivation**.

- The geometrical representation of rightmost derivation is called as a **rightmost derivation tree**.

Example 1 :

- Consider the following grammar-

  $S \rightarrow aB \,/\, bA$

  $S \rightarrow aS \,/\, bAA \,/\, a$

  $B \rightarrow bS \,/\, aBB \,/\, b$

- Let us consider a string w = aaabbabbba

- Now, let us derive the string w using rightmost derivation.

Rightmost Derivation-

$S \to aB \,/\, bA$
$A \to aS \,/\, bAA \,/\, a$
$B \to bS \,/\, aBB \,/\, b$

aaabbabbba

$S \to a\mathbf{B}$

$\to ab$ (Using $B \to b$)

This is NOT what we want ....

Rightmost Derivation-

$S \to aB \,/\, bA$
$A \to aS \,/\, bAA \,/\, a$
$B \to bS \,/\, aBB \,/\, b$

aaabbabbba

$S \to a\mathbf{B}$
$\to a\,bS$ (Using $B \to bS$)
$\to abbA$ (Using $S \to bA$)
$\to abba$ (Using $A \to a$)

This is NOT what we want ....

Rightmost Derivation-

$S \rightarrow aB \,/\, bA$
$A \rightarrow aS \,/\, bAA \,/\, a$
$B \rightarrow bS \,/\, aBB \,/\, b$

aaabbabbba

$S \rightarrow a\mathbf{B}$
$\rightarrow aaB\mathbf{B}$ (Using $B \rightarrow aBB$)
$\rightarrow aaBb\mathbf{S}$ (Using $B \rightarrow bS$)
$\rightarrow aaBbb\mathbf{A}$ (Using $S \rightarrow bA$)
$\rightarrow aaBbba$ (Using $A \rightarrow a$)
$\rightarrow aaaB\mathbf{B}bba$ (Using $B \rightarrow aBB$)
$\rightarrow aaaBbbba$ (Using $B \rightarrow b$)
$\rightarrow aaabSbbba$ (Using $B \rightarrow bS$)
$\rightarrow aaabbAbbba$ (Using $S \rightarrow bA$)
$\rightarrow aaabbaSbbba$ (Using $A \rightarrow aS$)

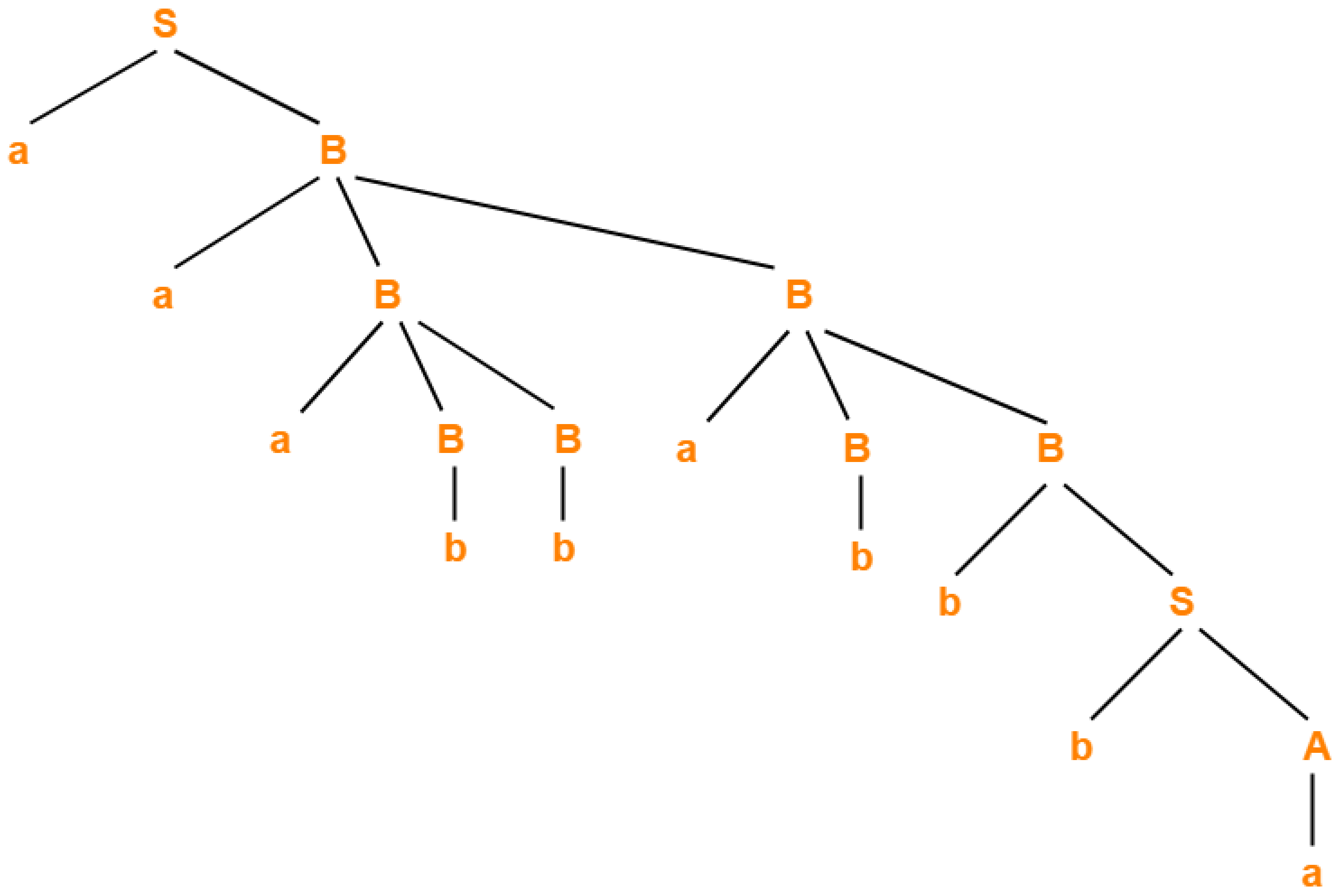
This is NOT what we want ....

Rightmost Derivation-

$S \rightarrow aB \,/\, bA$
$A \rightarrow aS \,/\, bAA \,/\, a$
$B \rightarrow bS \,/\, aBB \,/\, b$

aaabbabbba

$S \rightarrow a\mathbf{B}$
$\rightarrow aaB\mathbf{B}$ (Using $B \rightarrow aBB$)
$\rightarrow aaBaB\mathbf{B}$ (Using $B \rightarrow aBB$)
$\rightarrow aaBaBb\mathbf{S}$ (Using $B \rightarrow bS$)
$\rightarrow aaBaBbb\mathbf{A}$ (Using $S \rightarrow bA$)
$\rightarrow aaBa\mathbf{B}bba$ (Using $A \rightarrow a$)
$\rightarrow aa\mathbf{B}abbba$ (Using $B \rightarrow b$)
$\rightarrow aaaB\mathbf{B}abbba$ (Using $B \rightarrow aBB$)
$\rightarrow aaa\mathbf{B}babbba$ (Using $B \rightarrow b$)
$\rightarrow aaabbabbba$ (Using $B \rightarrow b$)



Rightmost Derivation Tree

Parse Tree :

- **<u>Properties Of Parse Tree-</u>**
  - Root node of a parse tree is the **start symbol** of the grammar.
  - Each leaf node of a parse tree represents a **terminal symbol**.
  - Each interior node of a parse tree represents a **non-terminal symbol**.
  - Parse tree is independent of the order in which the productions are used during derivations.

- **<u>Yield Of Parse Tree-</u>**
  - Concatenating the leaves of a parse tree from the left produces a string of terminals.
  - This string of terminals is called as **yield of a parse tree**.

Example 2:

Consider the grammar-

$S \rightarrow bB \;/\; aA$

$A \rightarrow b \;/\; bS \;/\; aAA$

$B \rightarrow a \;/\; aS \;/\; bBB$

For the string **w = bbaababa**, find-

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

Solution :

1. Leftmost Derivation-

$S \rightarrow b\mathbf{B}$

$\rightarrow bb\mathbf{B}B$ (Using $B \rightarrow bBB$)

$\rightarrow bba\mathbf{B}$ (Using $B \rightarrow a$)

$\rightarrow bbaa\mathbf{S}$ (Using $B \rightarrow aS$)

$\rightarrow bbaab\mathbf{B}$ (Using $S \rightarrow bB$)

$\rightarrow bbaaba\mathbf{S}$ (Using $B \rightarrow aS$)

$\rightarrow bbaabab\mathbf{B}$ (Using $S \rightarrow bB$)

$\rightarrow bbaababa$ (Using $B \rightarrow a$)

2. Rightmost Derivation-

$S \rightarrow b\mathbf{B}$

$\rightarrow bbB\mathbf{B}$ (Using $B \rightarrow bBB$)

$\rightarrow bbBa\mathbf{S}$ (Using $B \rightarrow aS$)

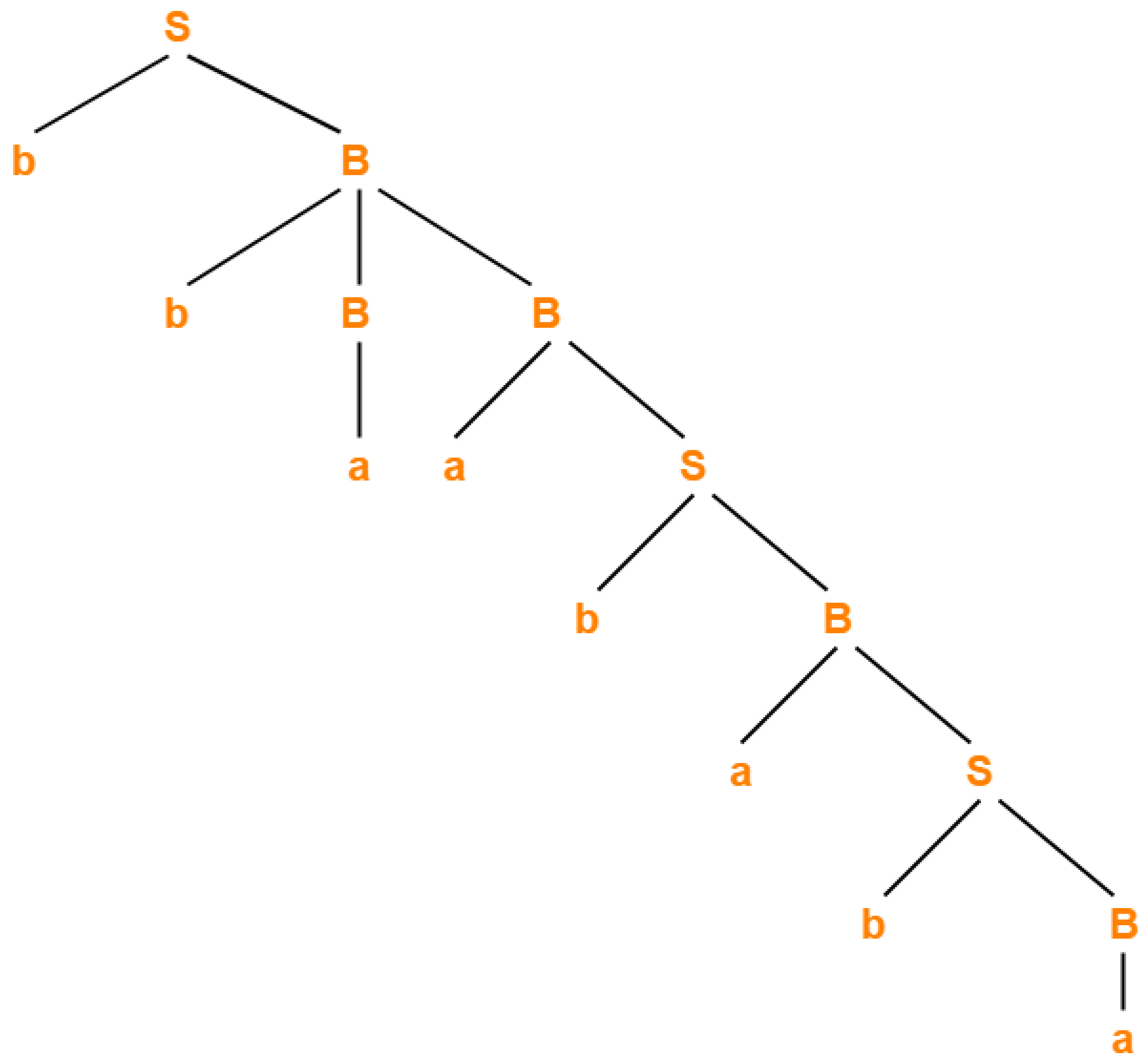
$\rightarrow bbBab\mathbf{B}$ (Using $S \rightarrow bB$)

$\rightarrow bbBaba\mathbf{S}$ (Using $B \rightarrow aS$)

$\rightarrow bbBabab\mathbf{B}$ (Using $S \rightarrow bB$)

$\rightarrow bb\mathbf{B}ababa$ (Using $B \rightarrow a$)

$\rightarrow bbaababa$ (Using $B \rightarrow a$)



Parse Tree

Example 3 :

Consider the grammar-

$S \rightarrow A1B$

$A \rightarrow 0A \ / \ \in$

$B \rightarrow 0B \ / \ 1B \ / \ \in$

For the string **w = 00101**, find-

Leftmost derivation

Rightmost derivation

Parse Tree

Solution-

1. Leftmost Derivation-

$S \rightarrow A1B$

$\rightarrow 0\mathbf{A}1B$ (Using $A \rightarrow 0A$)

$\rightarrow 00\mathbf{A}1B$ (Using $A \rightarrow 0A$)

$\rightarrow 001\mathbf{B}$ (Using $A \rightarrow \in$)

$\rightarrow 0010\mathbf{B}$ (Using $B \rightarrow 0B$)

$\rightarrow 00101\mathbf{B}$ (Using $B \rightarrow 1B$)

$\rightarrow 00101$ (Using $B \rightarrow \in$)

2. Rightmost Derivation-

$S \rightarrow A1\mathbf{B}$

$\rightarrow A10\mathbf{B}$ (Using $B \rightarrow 0B$)

$\rightarrow A101\mathbf{B}$ (Using $B \rightarrow 1B$)

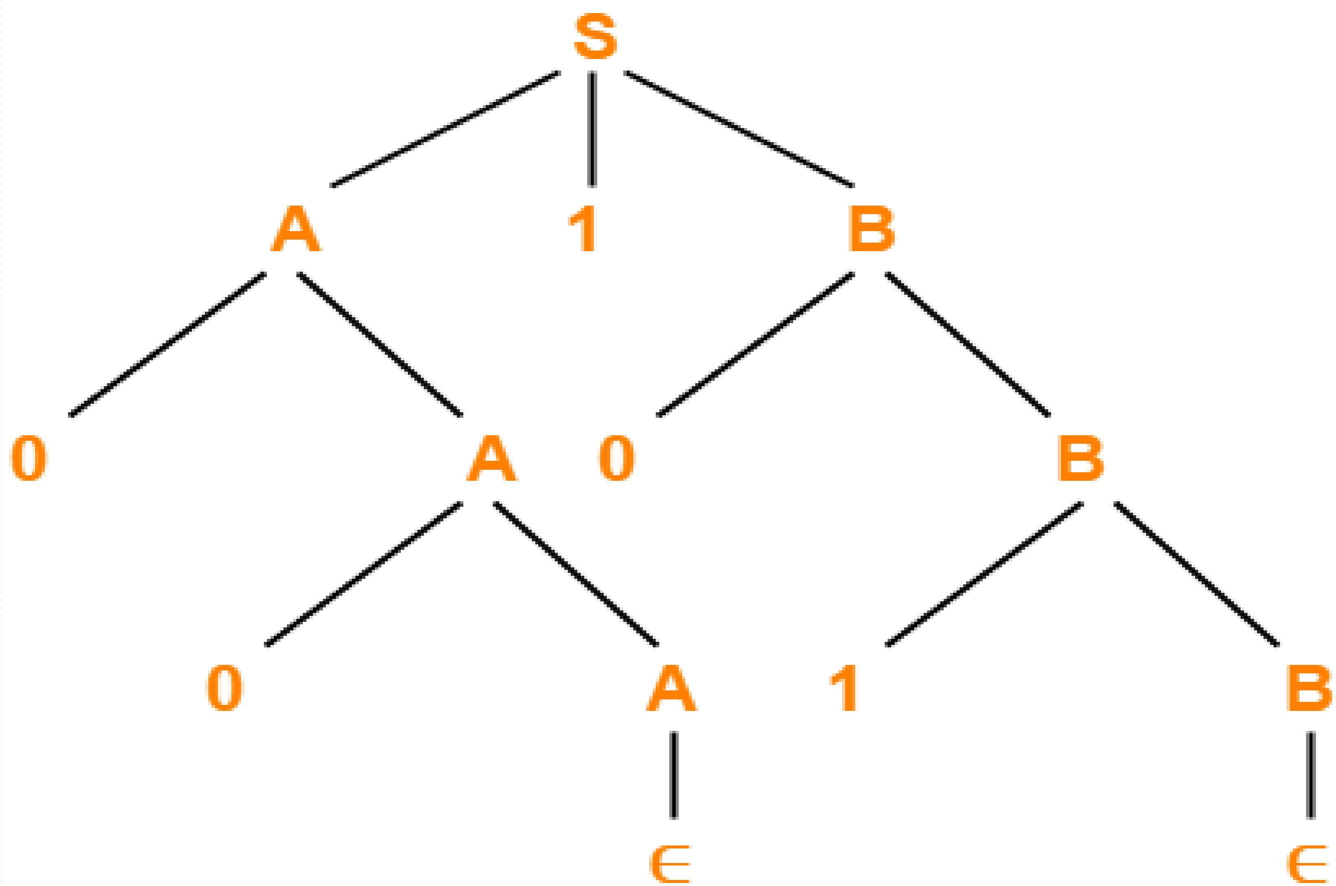
$\rightarrow \mathbf{A}101$ (Using $B \rightarrow \in$)

$\rightarrow 0\mathbf{A}101$ (Using $A \rightarrow 0A$)

$\rightarrow 00\mathbf{A}101$ (Using $A \rightarrow 0A$)

$\rightarrow 00101$ (Using $A \rightarrow \in$)

Parse Tree :



Parse Tree

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}.

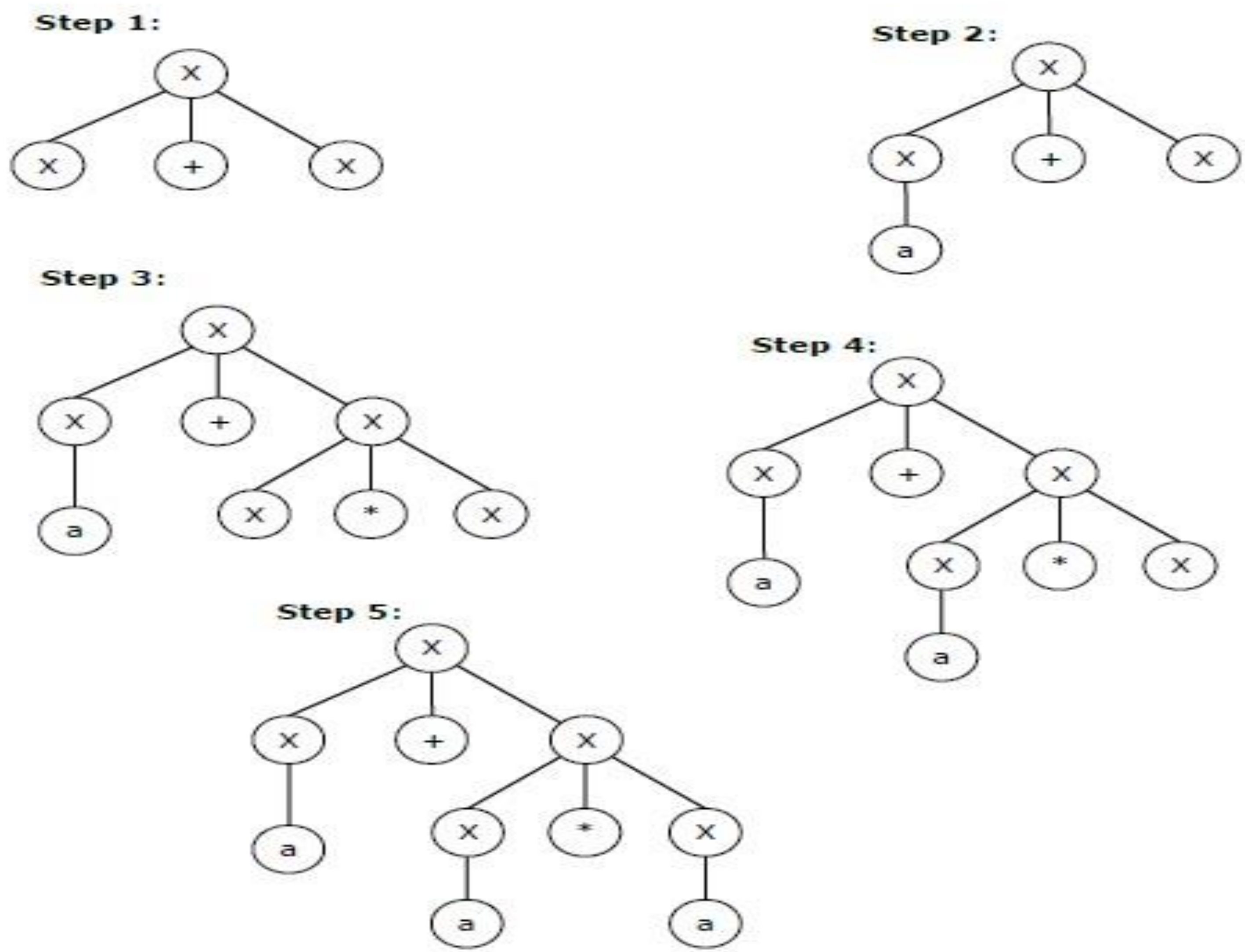
Show derivation for the string "a+a*a"

X → X+X
X→ X*X
X→X
X→ a

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

$$
\begin{aligned}
X &\rightarrow X+X \\
&\rightarrow a+X \\
&\rightarrow a + X*X \\
&\rightarrow a+a*X \\
&\rightarrow a+a*a
\end{aligned}
$$

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

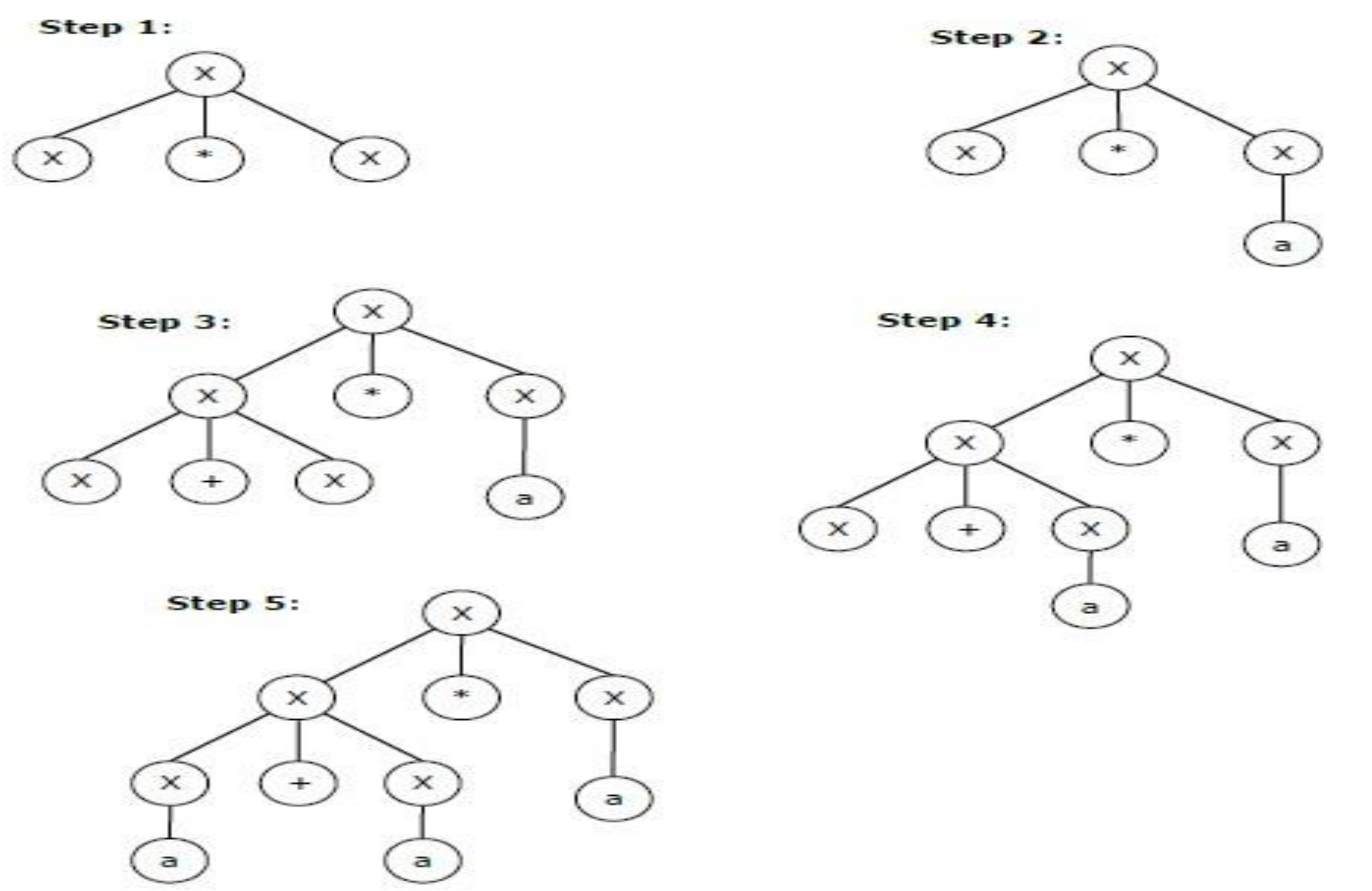


$X \rightarrow X+X$
$X \rightarrow X*X$
$X \rightarrow X$
$X \rightarrow a$

The rightmost derivation for the above string "**a+a*a**" may be –

$$
\begin{aligned}
X &\rightarrow X*X \\
&\rightarrow X*a \\
&\rightarrow X+X*a \\
&\rightarrow X+a*a \\
&\rightarrow a+a*a
\end{aligned}
$$

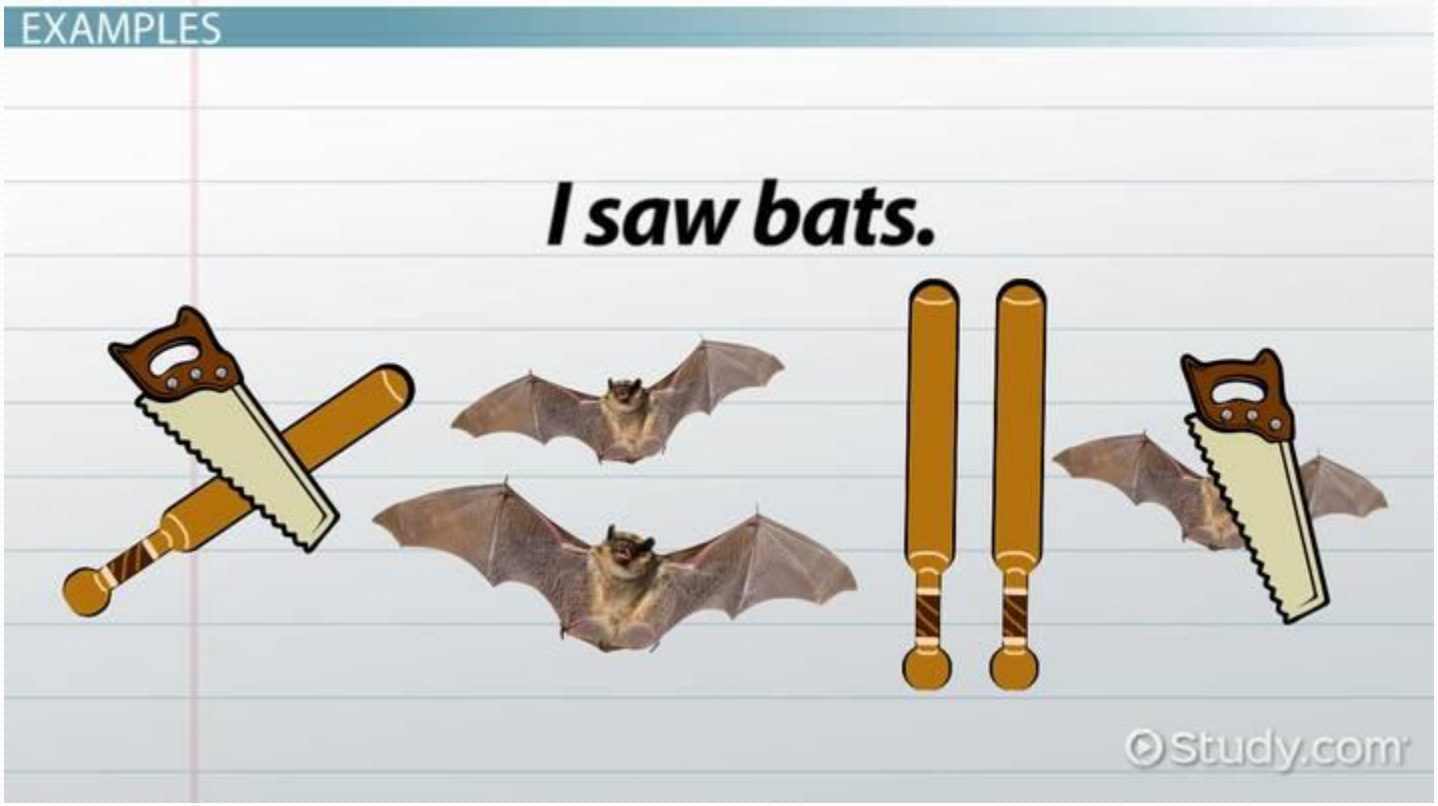
The stepwise derivation of the above string is shown as below –



Ambiguous and Unambiguous Grammar :

--**Sakshi Surve**

Ambiguity :

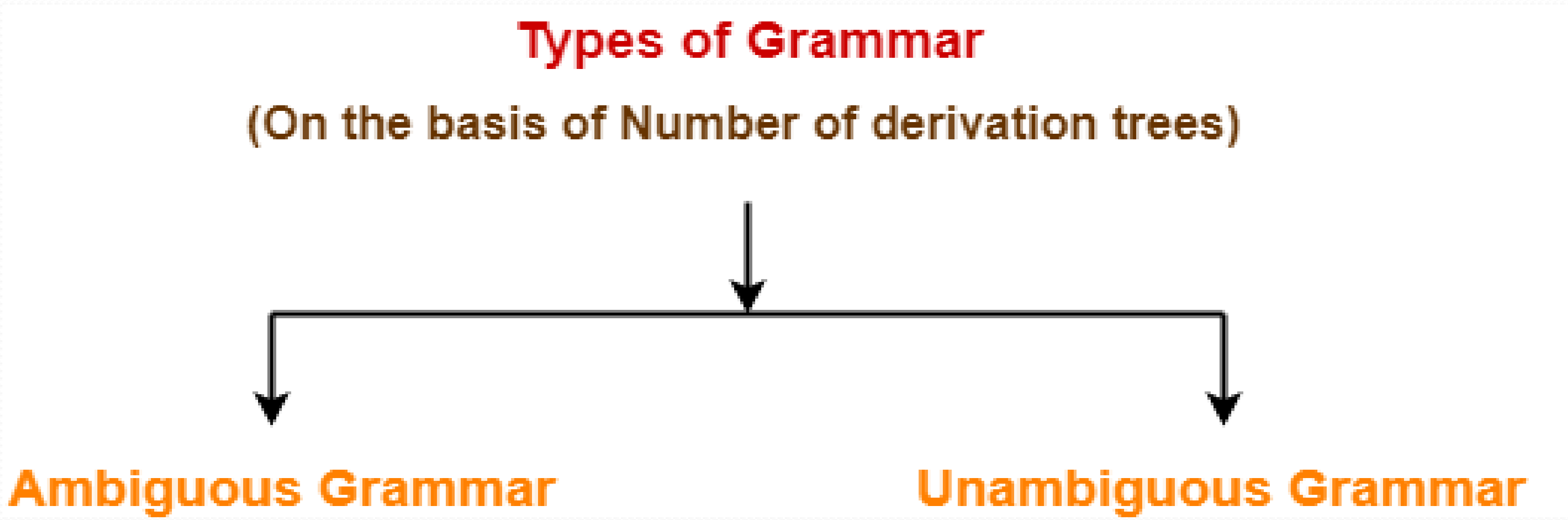


- **Ambiguous** means Open to more than one interpretation , Not having one obvious meaning

- A grammar is said to ambiguous if for any string generated by it, it produces more than one-
  - Parse tree **Or**
  - Leftmost Derivation **Or**
  - Rightmost Derivation

If there exists at least one such string, then the grammar is ambiguous otherwise unambiguous.

Ambiguous and Unambiguous Grammar :



Grammar Ambiguity-

1. There exists no general algorithm to remove the **ambiguity** from **grammar**.

2. To check **grammar ambiguity**, we try finding a string that has more than one parse tree.

3. If any such string exists, then the **grammar** is **ambiguous** otherwise not.

Example 01-

- Consider the following grammar-

$$E \rightarrow E + E \mid E \times E \mid id$$

- **Ambiguous Grammar**

- This grammar is an example of ambiguous grammar.

- Any of the following reasons can be stated to prove the grammar ambiguous-

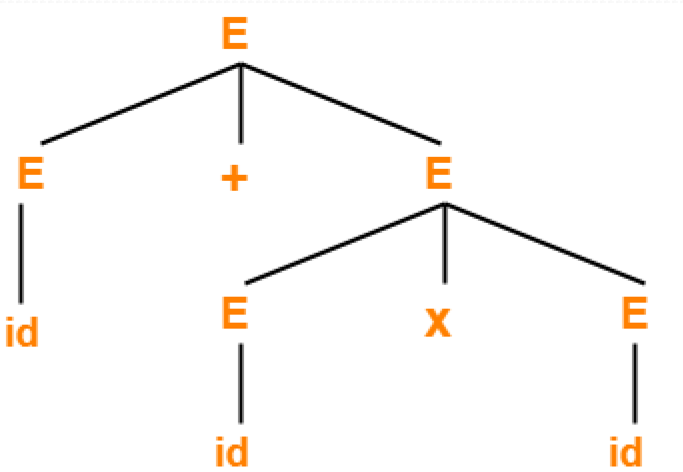
Reason-01: Parse Tree

$E \rightarrow E + E \mid E \times E \mid id$

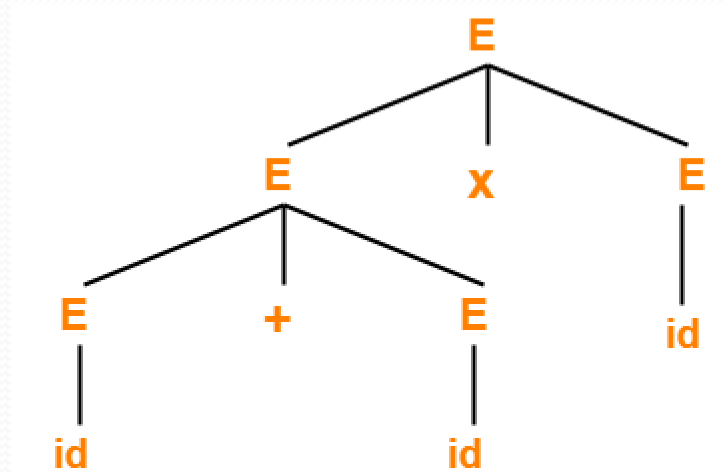
- Let us consider a string w generated by the grammar-

  w = id + id x id

- Now, let us draw the parse trees for this string w.



Parse Tree-01



Parse Tree-02

- **Since two parse trees exist for string w, the grammar is ambiguous.**

Reason-02: Leftmost Derivati

$E \to E + E \mid E \times E \mid id$

- Let us consider a string w generated by the

$$w = id + id \times id$$

- Now, let us write the leftmost derivations for this string w.

$$\begin{aligned} E &\to \mathbf{E} + E \\ &\to id + \mathbf{E} \\ &\to id + \mathbf{E} \times E \\ &\to id + id \times \mathbf{E} \\ &\to id + id \times id \end{aligned}$$

Leftmost Derivation-01

$$\begin{aligned} E &\to \mathbf{E} \times E \\ &\to \mathbf{E} + E \times E \\ &\to id + \mathbf{E} \times E \\ &\to id + id \times \mathbf{E} \\ &\to id + id \times id \end{aligned}$$

Leftmost Derivation-02

Since two leftmost derivations exist for string w, the grammar is ambiguous.

Reason-03 : Rightmost Derivation

- Let us consider a string w generated by the gr

$E \to E + E \mid E \times E \mid id$

$$w = id + id \times id$$

- Now, let us write the rightmost derivations for this string w.

$E \to E + E$
$\to E + E \times E$
$\to E + E \times id$
$\to E + id \times id$
$\to id + id \times id$

Rightmost Derivation-01

$E \to E \times E$
$\to E \times id$
$\to E + E \times id$
$\to E + id \times id$
$\to id + id \times id$

Rightmost Derivation-02

Since two rightmost derivations exist for string w, the grammar is ambiguous.

Example -02:

- Check whether the given grammar is ambiguous or not-

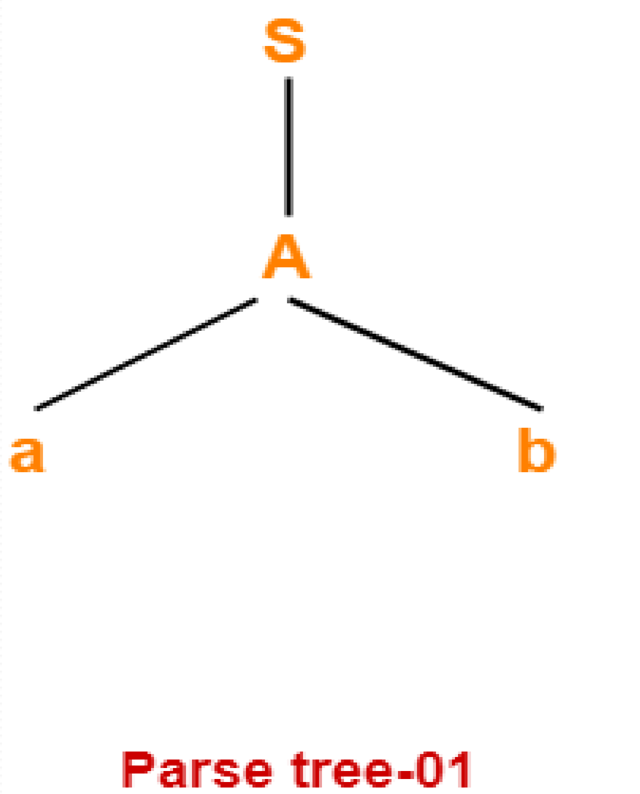
$$S \rightarrow A \,/\, B$$
$$A \rightarrow aAb \,/\, ab$$
$$B \rightarrow abB \,/\, \in$$

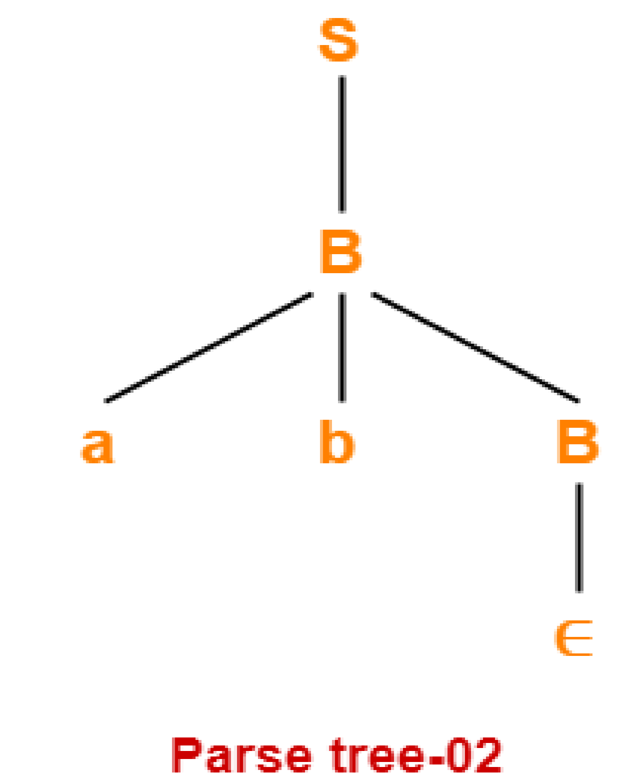
- **Solution-**

Let us consider a string w generated by the given grammar-

$$w = ab$$

Now, let us draw parse trees for this string w.





Since two different parse trees exist for string w, the given grammar is ambiguous.

Example - 03:

Check whether the given grammar is ambiguous or not-

$S \rightarrow AB \ / \ C$

$A \rightarrow aAb \ / \ ab$

$B \rightarrow cBd \ / \ cd$

$C \rightarrow aCd \ / \ aDd$

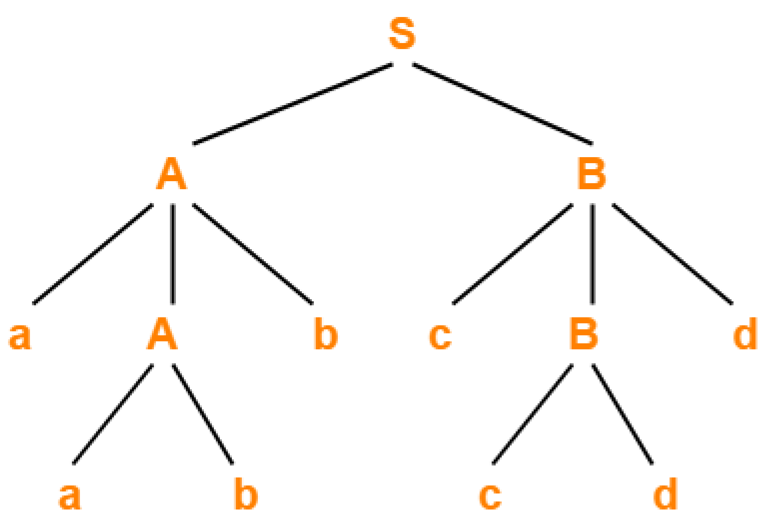
$D \rightarrow bDc \ / \ bc$

Solution-

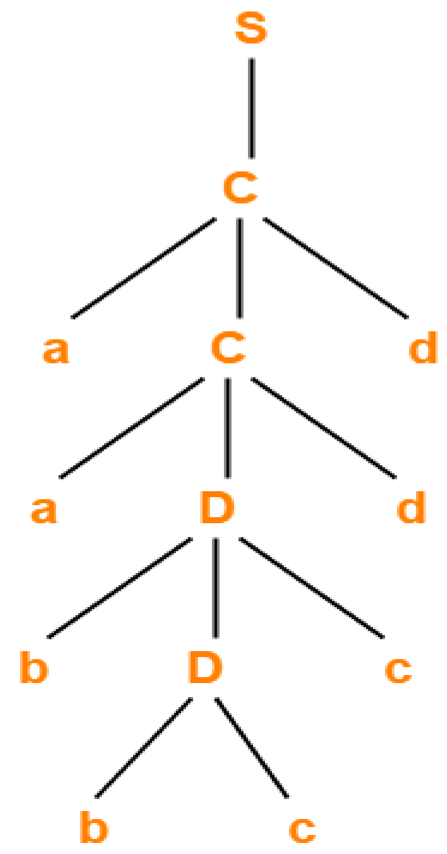
Let us consider a string w generated by the given grammar-

w = aabbccdd

Now, let us draw parse trees for this string w.



Parse tree-01



Parse tree-02

Since two different parse trees exist for string w, the given grammar is ambiguous.

Example - 04:

Check whether the given grammar is ambiguous or not-

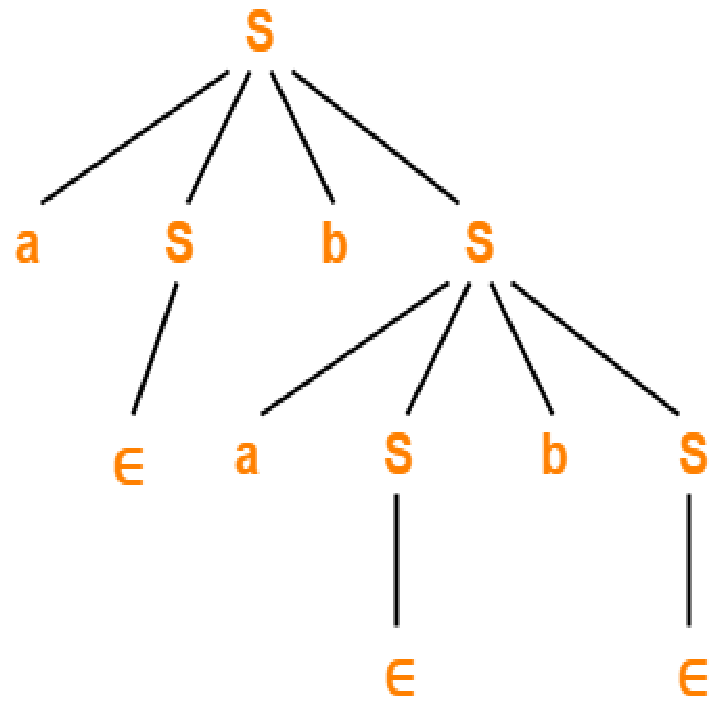
$$S \rightarrow aSbS \ / \ bSaS \ / \ \in$$

Solution-

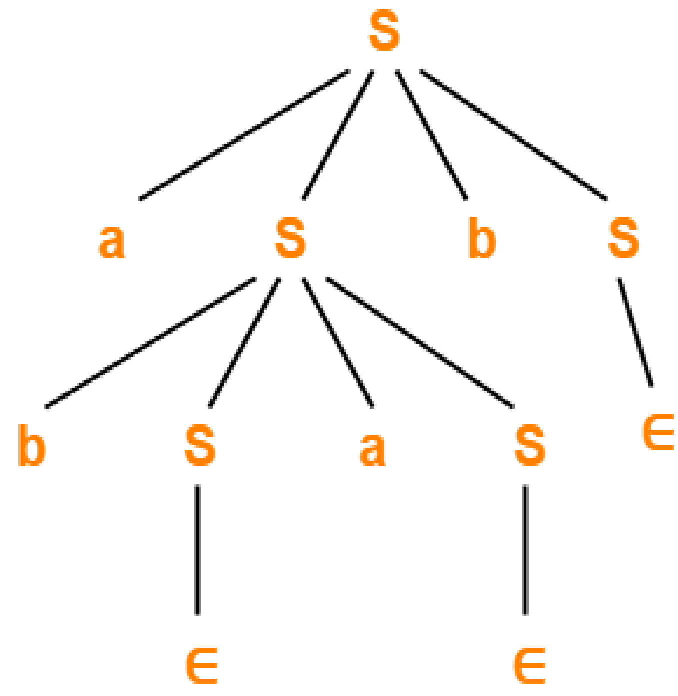
Let us consider a string w generated by the given grammar-

$$w = abab$$

Now, let us draw parse trees for this string w.



Parse tree-01



Parse tree-02

Since two different parse trees exist for string w, the given grammar is ambiguous.

Example -05:

Check whether the given grammar is ambiguous or not-

$S \rightarrow SS$

$S \rightarrow a$

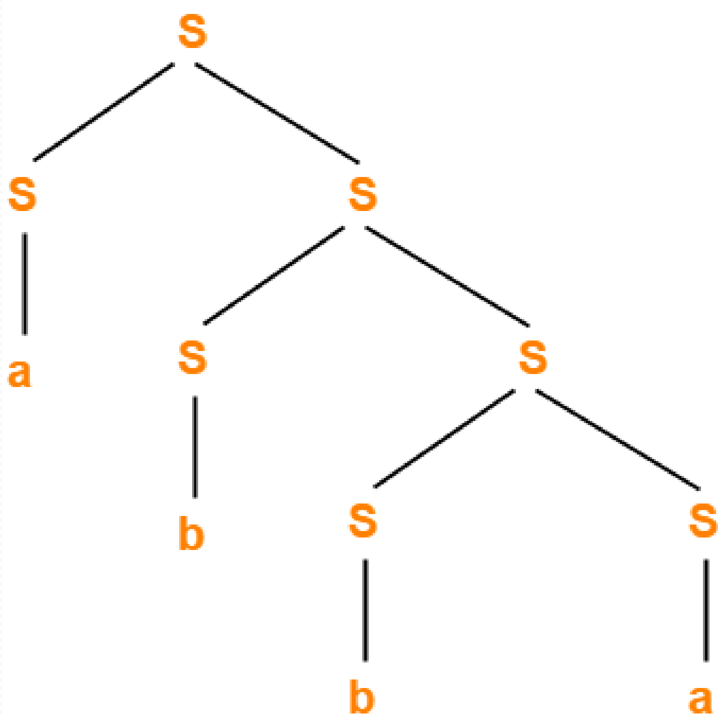
$S \rightarrow b$

Solution-

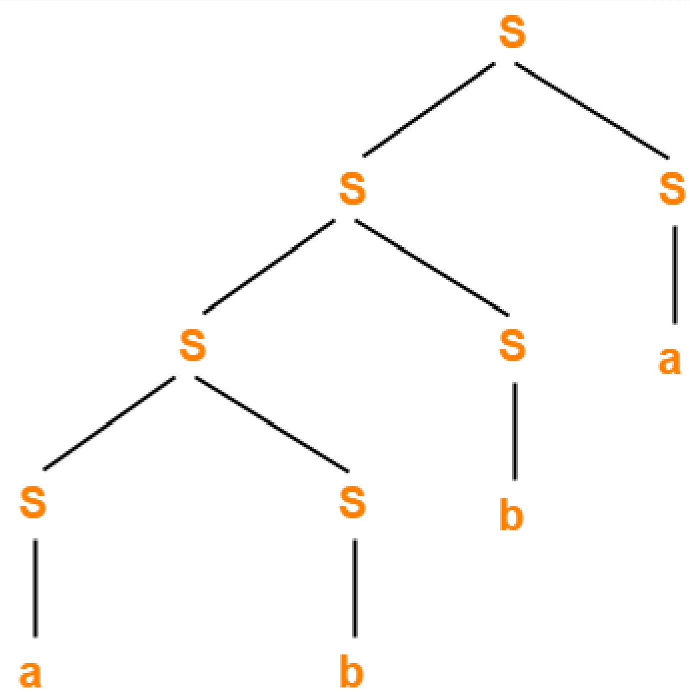
Let us consider a string w generated by the given grammar-

$w = abba$

Now, let us draw parse trees for this string w.



Parse tree-01



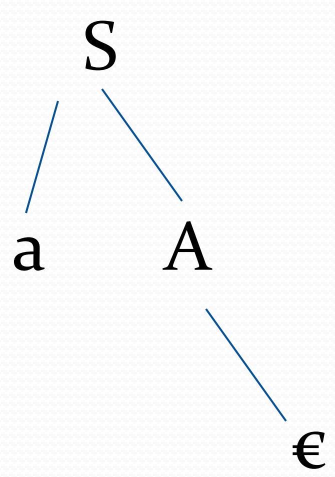
Parse tree-02

Since two different parse trees exist for string w, therefore the given grammar is ambiguous.

- $\sum = \{ a , b \}$
- L = { w € L | w begins with a }
- L = { a , aa, ab, aab, aba, aaa, ...}

- S -> aA

  A -> aA | bA | €
- For 'a'



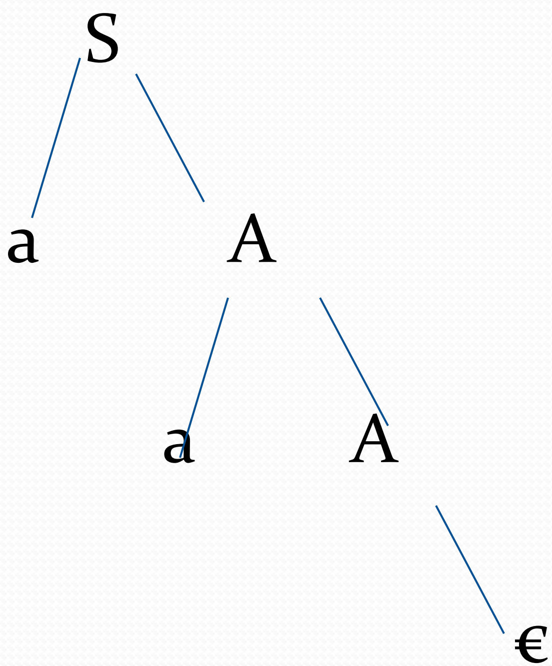
Derivation of 'a'

S →aA

  →a

S -> aA

A -> aA | bA | €

- For generating 'aa'
- 

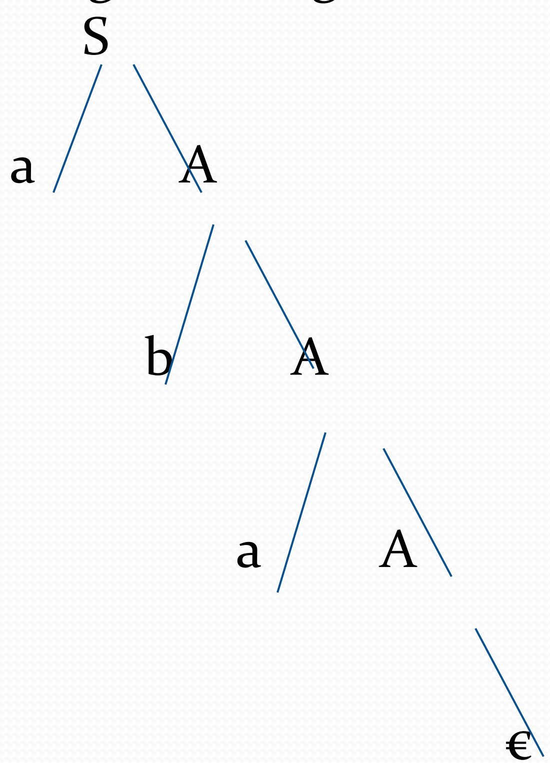
Derivation of 'aa'

S →aA
→aaA
→aa

S -> aA

A -> aA | bA | €

- For generating 'aba'
- S



Derivation of 'aba'

S →aA
→abA
→abaA
→aba

In this grammar, the tuples are :

V = {S, A}
T = { a,b }
P
S = S

This is an example of Unambiguous Grammar

Example 6 :

- Consider the following example :

$$S \rightarrow aB \,/\, bA$$
$$A \rightarrow aS \,/\, bAA \,/\, a$$
$$B \rightarrow bS \,/\, aBB \,/\, b$$

- Let us consider a string

$$w = aaabbabbba$$

- Now, let us derive the string w using leftmost derivation.

Leftmost Derivation -

$S \rightarrow aB \mid bA$
$A \rightarrow aS \mid bAA \mid a$
$B \rightarrow bS \mid aBB \mid b$

aaabbabbba

$S \rightarrow a\mathbf{B}$

$\rightarrow aa\mathbf{B}B$ (Using $B \rightarrow aBB$)

$\rightarrow aaa\mathbf{B}BB$ (Using $B \rightarrow aBB$)

$\rightarrow aaab\mathbf{B}B$ (Using $B \rightarrow b$)

$\rightarrow aaabb\mathbf{B}$ (Using $B \rightarrow b$)

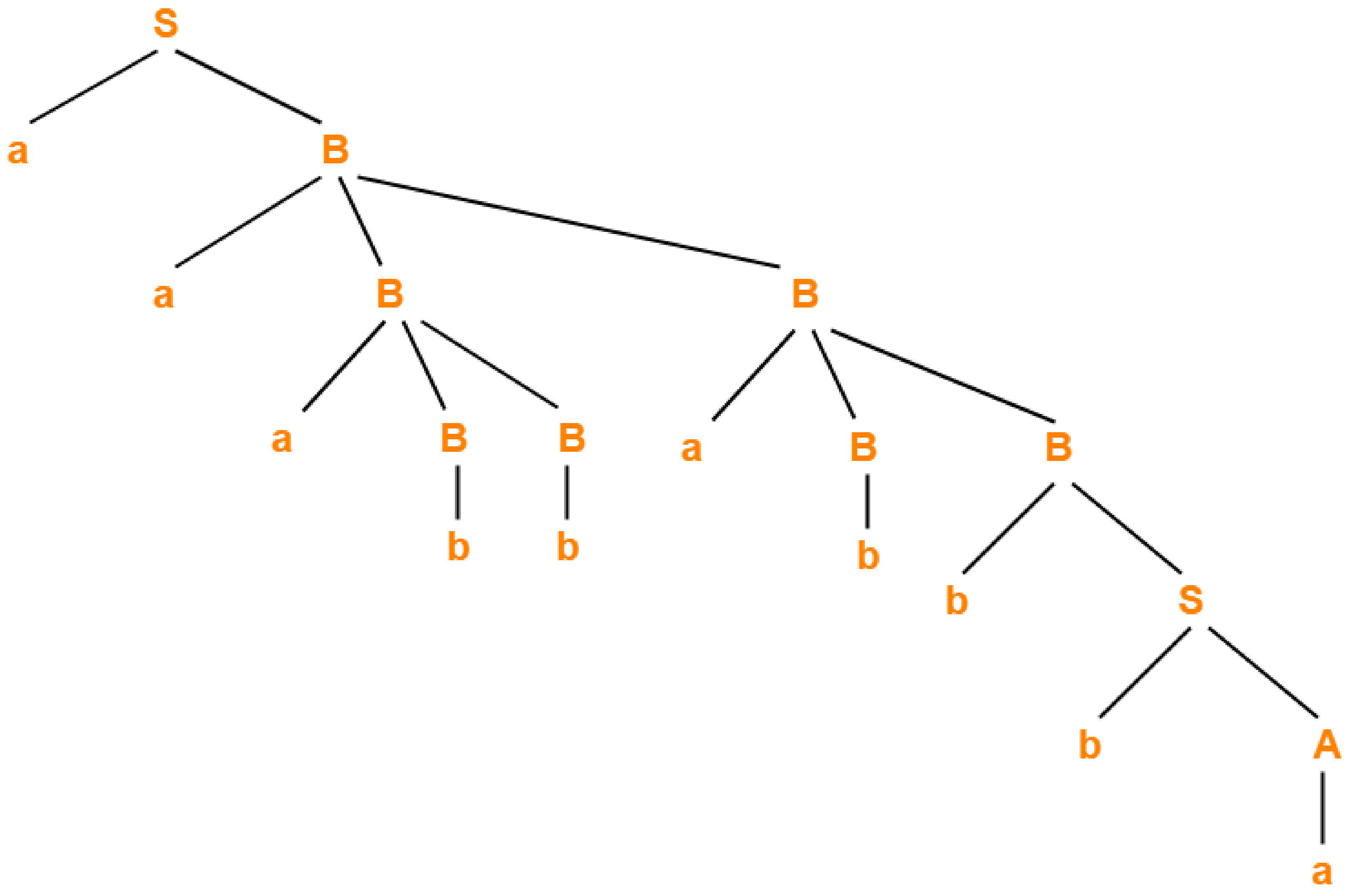
$\rightarrow aaabba\mathbf{B}B$ (Using $B \rightarrow aBB$)

$\rightarrow aaabbab\mathbf{B}$ (Using $B \rightarrow b$)

$\rightarrow aaabbabb\mathbf{S}$ (Using $B \rightarrow bS$)

$\rightarrow aaabbabbb\mathbf{A}$ (Using $S \rightarrow bA$)

$\rightarrow aaabbabbba$ (Using $A \rightarrow a$)



Leftmost Derivation Tree

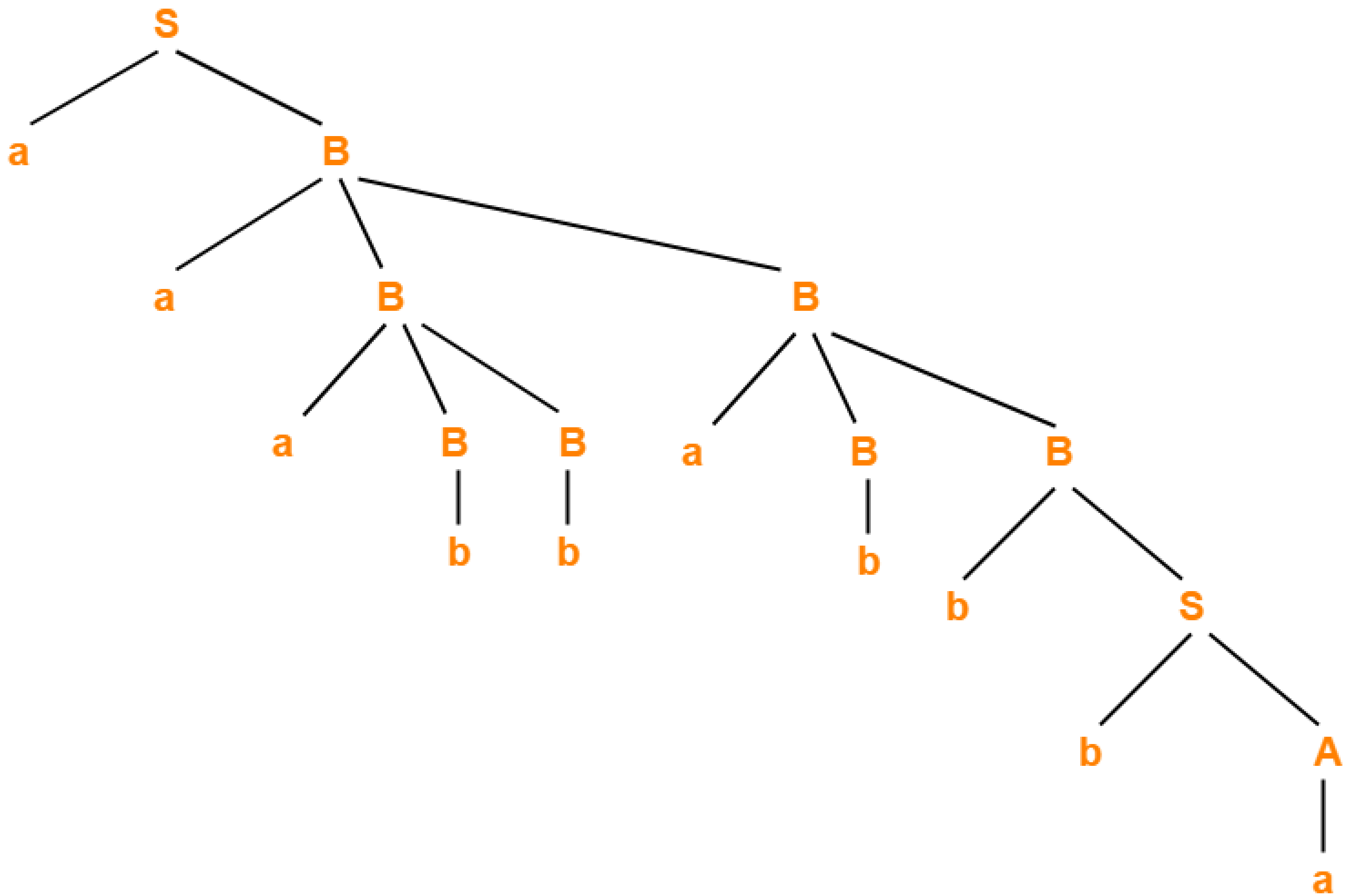
Rightmost Derivation-

S → aB / bA
A → aS / bAA / a
B → bS / aBB / b

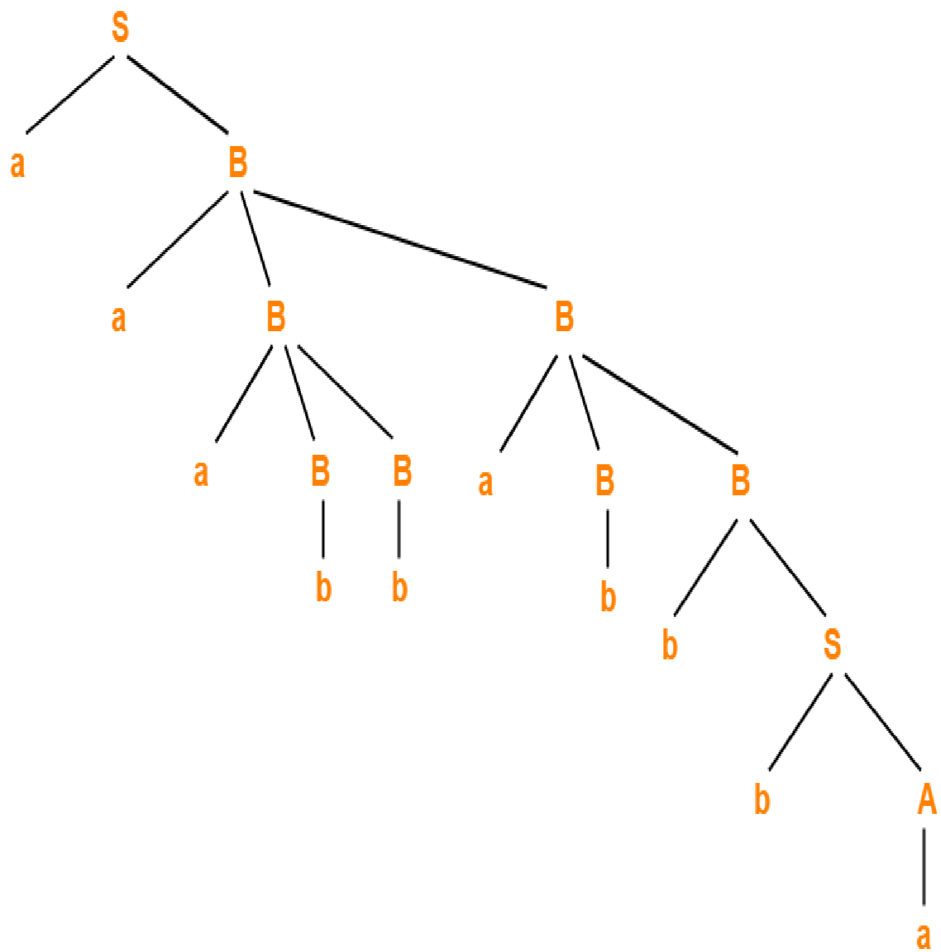
aaabbabbba

S → a**B**

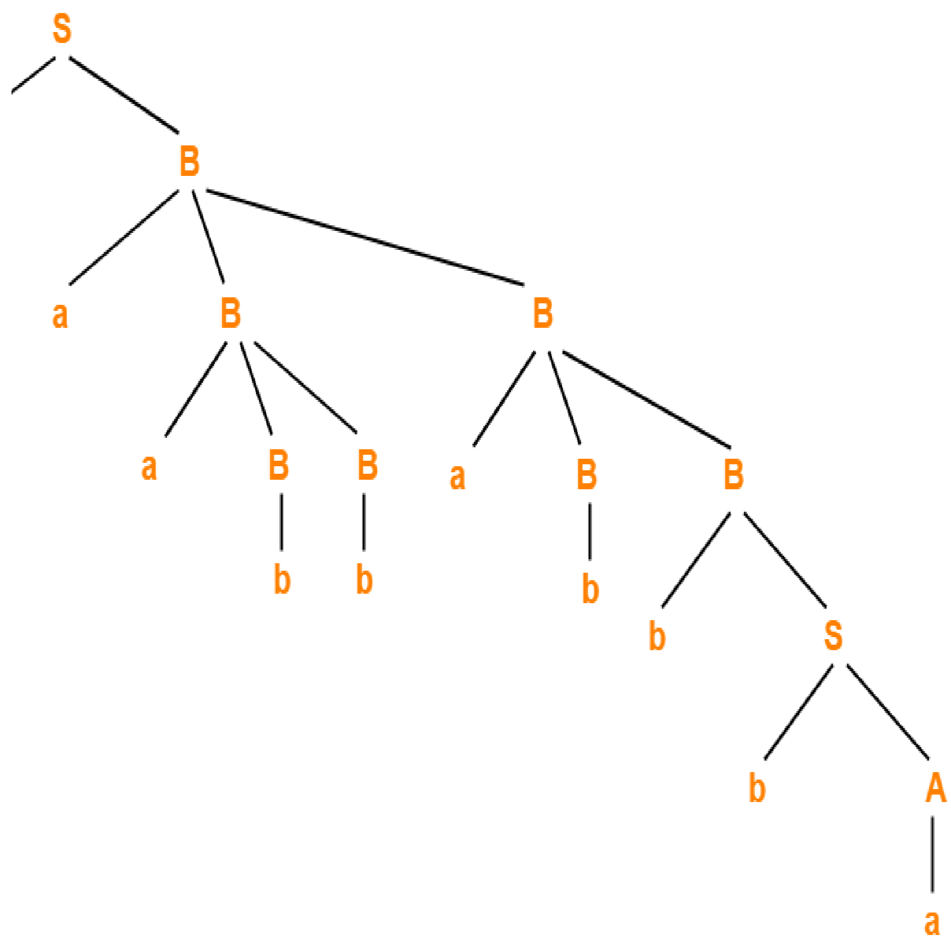
→ aaB**B**	(Using B → aBB)
→ aaBaB**B**	(Using B → aBB)
→ aaBaBb**S**	(Using B → bS)
→ aaBaBbb**A**	(Using S → bA)
→ aaBa**B**bba	(Using A → a)
→ aa**B**abbba	(Using B → b)
→ aaaB**B**abbba	(Using B → aBB)
→ aaa**B**babbba	(Using B → b)
→ aaabbabbba	(Using B → b)



Rightmost Derivation Tree



Leftmost Derivation Tree



Rightmost Derivation Tree

Since one parse tree exists for string w, the given grammar is unambiguous.