UNIT 3
Context Free Grammar

By:Padmavati Sarode

Content:

Chomsky hierarchy of languages. Definition of Context-Free Grammars, Derivations Using a Grammar, Leftmost and Rightmost Derivations, the Language of a Grammar, Sentential Forms, Parse Tress, Applications of Context-Free Grammars, Ambiguity in Grammars and Languages.

In formal language theory, a grammar (when the context is not given, often called a formal grammar for clarity) **describes how to form strings from a language's alphabet that are valid according to the language's syntax**. ... A formal grammar is defined as a set of production rules for strings in a formal language.

What are the types of formal grammar?
Formal grammars fall into two main categories: **generative and analytic**. A generative grammar, the most well-known kind, is a set of rules by which all possible strings in the language to be described can be generated by successively rewriting strings starting from a designated start symbol.

The theory of formal languages finds its applicability extensively in the fields of Computer Science.

Noam Chomsky gave a mathematical model of grammar in 1956 which is effective for writing computer languages.

Grammar

A grammar **G** can be formally written as a 4-tuple (N, T, S, P) where −

- **N** or $\mathbf{V}_N$ is a set of variables or non-terminal symbols.
- **T** or $\sum$ is a set of Terminal symbols.
- **S** is a special variable called the Start symbol, $S \in N$
- **P** is Production rules for Terminals and Non-terminals.
- A production rule has the form $\alpha \rightarrow \beta$,
- where α and β are strings on $V_N \cup \sum$ and least one symbol of α belongs to V_N.

Example

Grammar G1 −

$(\{S, A, B\}, \{a, b\}, S, \{S \rightarrow AB, A \rightarrow a, B \rightarrow b\})$

Here,

S, A, and **B** are Non-terminal symbols;

a and **b** are Terminal symbols

S is the Start symbol, $S \in N$

Productions, $\mathbf{P : S \rightarrow AB, A \rightarrow a, B \rightarrow b}$

Example

Grammar G2 −

$((\{S, A\}, \{a, b\}, S, \{S \rightarrow aAb, aA \rightarrow aaAb, A \rightarrow \varepsilon\})$

Here,

S and **A** are Non-terminal symbols.

a and **b** are Terminal symbols.

ε is an empty string.

S is the Start symbol, $S \in N$

Production $\mathbf{P : S \rightarrow aAb, aA \rightarrow aaAb, A \rightarrow \varepsilon}$

Derivations from a Grammar

Strings may be derived from other strings using the productions in a grammar. If a grammar **G** has a production **α → β**, we can say that **x α y** derives **x β y** in **G**. This derivation is written as −

x α y ⇒G x β y

Example

Let us consider the grammar −

G2 = ({S, A}, {a, b}, S, {S → aAb, aA → aaAb, A → ε })

Some of the strings that can be derived are −

S ⇒ aAb using production S → aAb

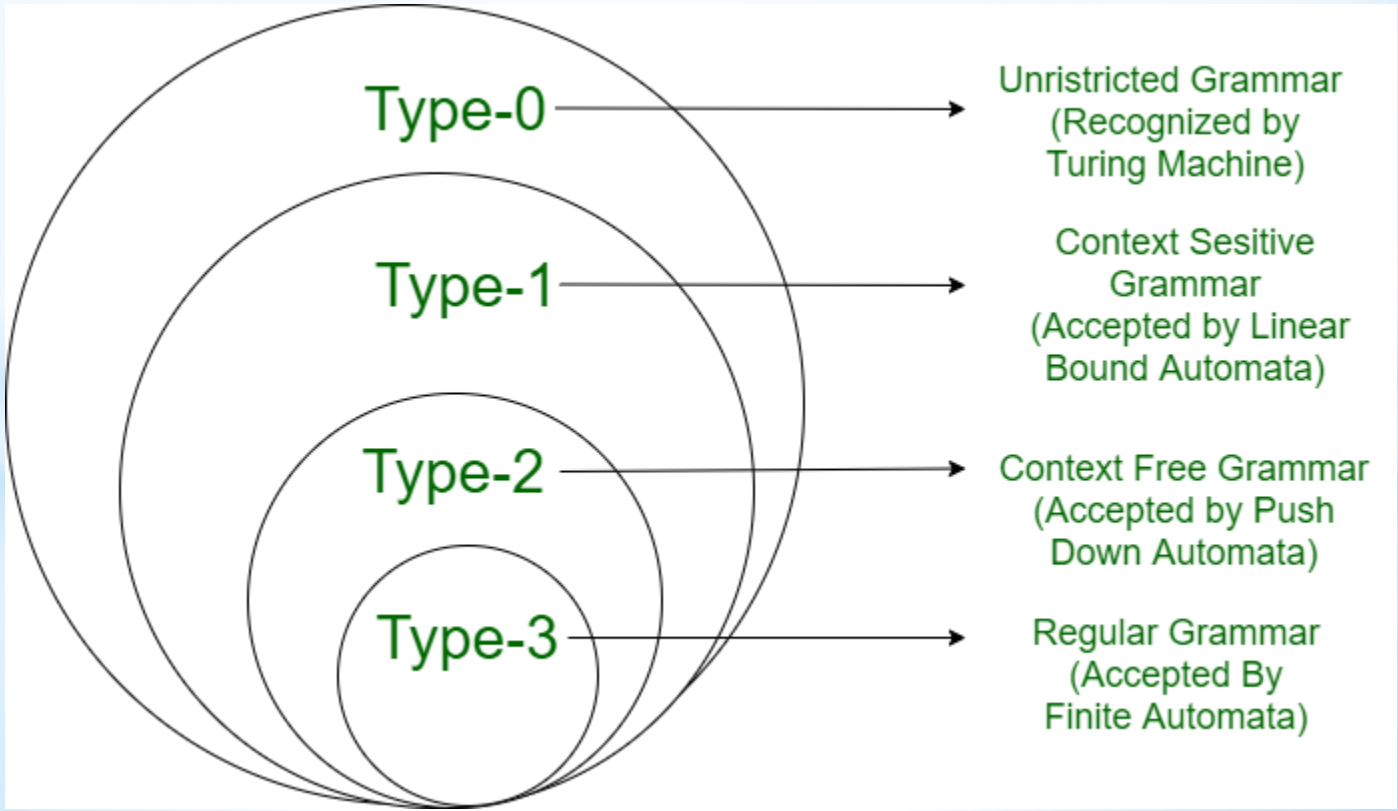
⇒ aaAbb using production aA → aAb

⇒ aaaAbbb using production aA → aaAb

⇒ aaabbb using production A → ε

According to Chomsky hierarchy, grammars is divided into 4 types:

Type 0 known as unrestricted grammar.
Type 1 known as context sensitive grammar.
Type 2 known as context free grammar.
Type 3 Regular Grammar.



Type 0: Unrestricted Grammar:

In Type 0

Type-0 grammars include all formal grammars. Type 0 grammar language are recognized by turing machine. These languages are also known as the Recursively Enumerable languages.

Grammar Production in the form of

$\alpha \rightarrow \beta$

where

$\alpha$is (V + T)* V (V + T)*

V : Variables

T : Terminals.

$\beta$is (V + T)*.

In type 0 there must be at least one variable on Left side of production.

For example,

Sab –> ba

A –> S.

Type 1: Context Sensitive Grammar)

Type-1 grammars generate the context-sensitive languages. The language generated by the grammar are recognized by the Linear Bound Automata

In Type 1

I. First of all Type 1 grammar should be Type 0.

II. Grammar Production in the form of

$\alpha \rightarrow \beta$

$|\alpha| <= |\beta|$

i.e count of symbol in α is less than or equal to β

For Example,

S -> AB

AB -> abc

B -> b

Type 2: Context Free Grammar:

Type-2 grammars generate the context-free languages. The language generated by the grammar is recognized by a Pushdown automata.

In Type 2,

1. First of all it should be Type 1.
2. Left hand side of production can have only one variable.

$|\alpha| = 1$.

Their is no restriction on β.

For example,

S -> AB

A -> a

B -> b

Type 3: Regular Grammar:

Type-3 grammars generate regular languages. These languages are exactly all languages that can be accepted by a finite state automaton.

Type 3 is most restricted form of grammar.

Type 3 should be in the given form only :

V –> VT / T (left-regular grammar)

(or)

V –> TV /T (right-regular grammar)

for example:

S –> a

The above form is called as strictly regular grammar.

There is another form of regular grammar called extended regular grammar. In this form :

V –> VT* / T*. (extended left-regular grammar)

(or)

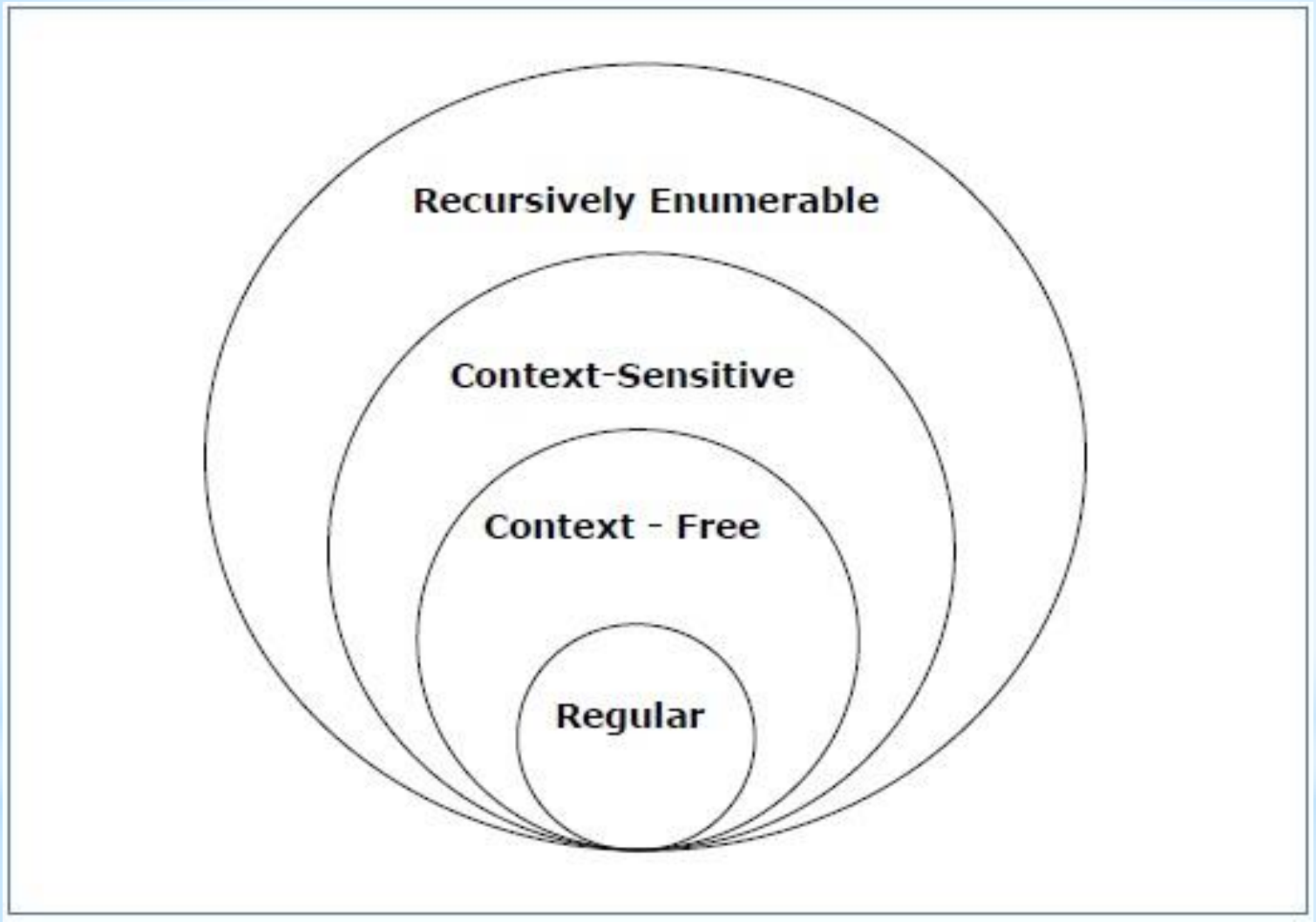
V –> T*V /T* (extended right-regular grammar)

for example :

S –> ab.

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted grammar	Recursively enumerable language	Turing Machine
Type 1	Context-sensitive grammar	Context-sensitive language	Linear-bounded automaton
Type 2	Context-free grammar	Context-free language	Pushdown automaton
Type 3	Regular grammar	Regular language	Finite state automaton

Take a look at the following illustration. It shows the scope of each type of grammar −



Languages associated to Chomsky-hierarchy grammars

Grammar	Languages	Automaton
Type-0	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine
Type-2	Context-free	Non-deterministic pushdown automaton
Type-3	Regular	Finite state automaton

A linear bounded automaton is a multi-track non-deterministic Turing machine with a tape of some bounded finite length.
Length = function (Length of the initial input string, constant c)

context-sensitive language :
In formal language theory, a context-sensitive language is **a language that can be defined by a context-sensitive grammar** (and equivalently by a non contracting grammar). Context-sensitive is one of the four types of grammars in the Chomsky hierarchy.

A context-sensitive grammar whose productions are of the form
$\alpha A \beta \rightarrow \alpha \gamma \beta$

The language generated by such a grammar is called a context-sensitive language.
Every context-free grammar is also context-sensitive =⇒ the context-free languages are a subset of the context-sensitive languages (see Chomsky Normal Form).
But, not every context-sensitive language is context-free.

The Chomsky Hierarchy

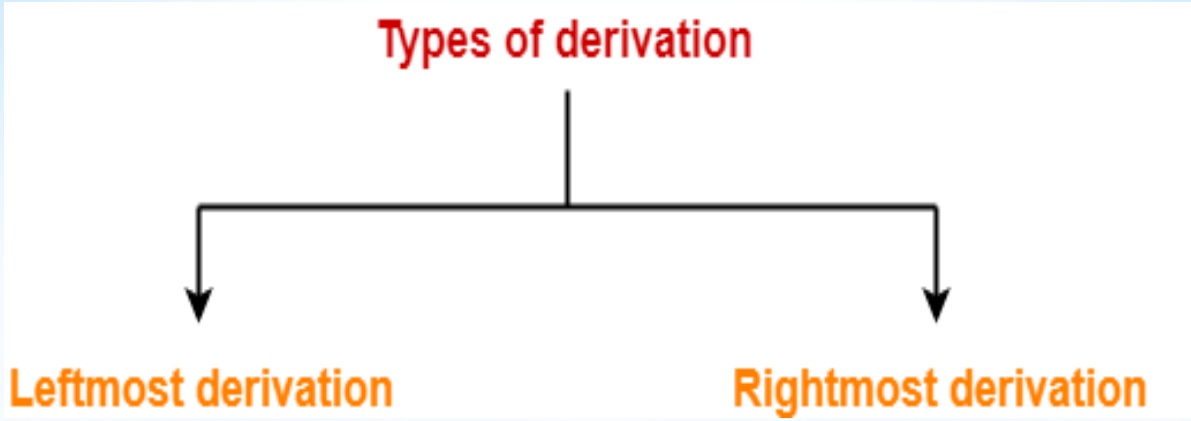
Gram mar	Languages	Automaton	Production rules
Type-0	Recursively enumerable	Turing machine	No restrictions
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow aB$ $A \rightarrow a$

Derivations Using a Grammar

A derivation of a string for a grammar is **a sequence of grammar rule applications that transform the start symbol into the string**. A derivation proves that the string belongs to the grammar's language.

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.**



1. 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-

Consider the following grammar-

$S \rightarrow aB / bA$
$A \rightarrow aS / bAA / a$
$B \rightarrow bS / aBB / b$
(**Unambiguous Grammar**)

Let us consider a string w = aaabbabbba
Now, let us derive the string w using leftmost derivation.

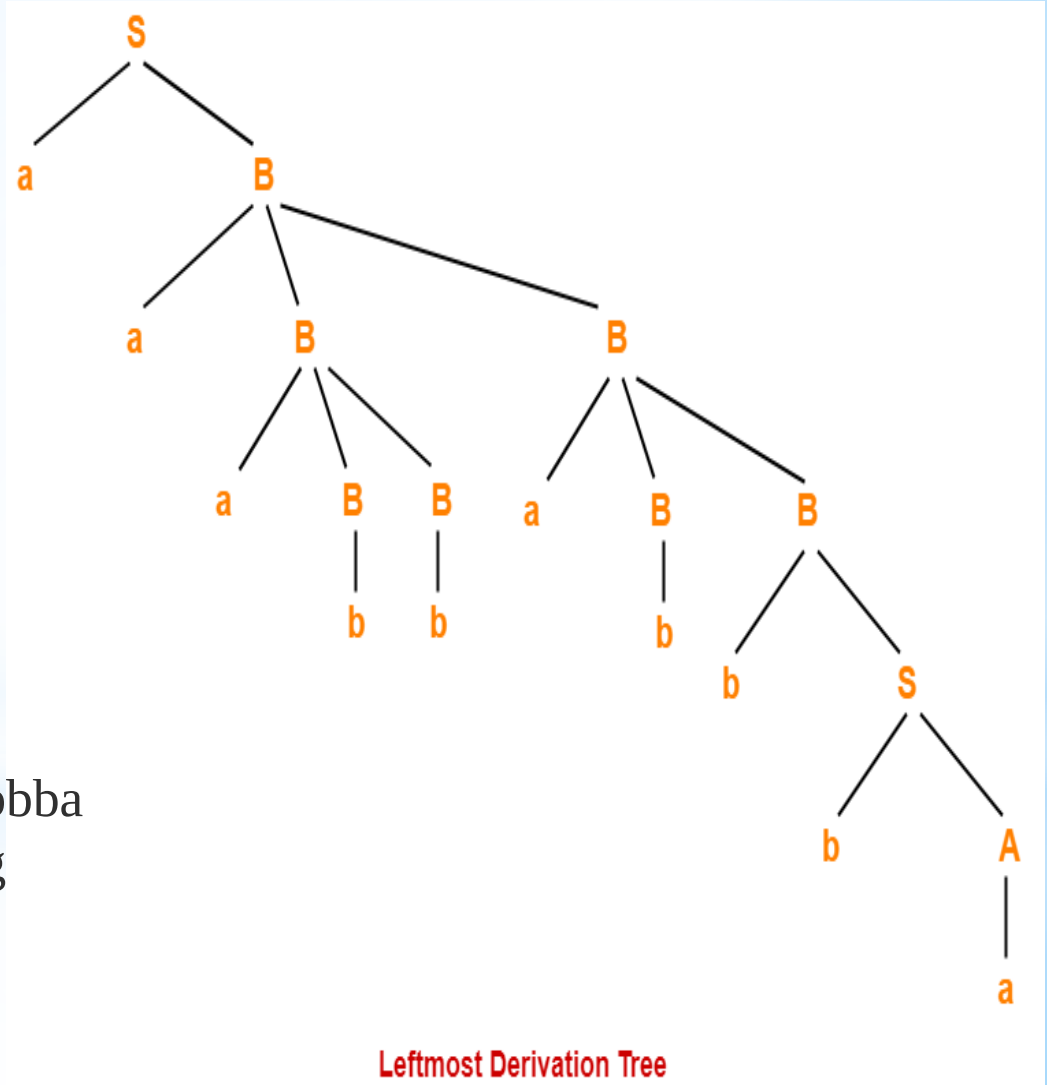
Leftmost Derivation-

$S \rightarrow aB$
$\rightarrow aaBB$ (Using $B \rightarrow aBB$)

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)

S → aB / bA
A → aS / bAA / a
B → bS / aBB / b
(**Unambiguous Grammar**)

Let us consider a string w = aaabbabbba
Now, let us derive the string w using leftmost derivation.



Leftmost Derivation Tree

2. 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**.

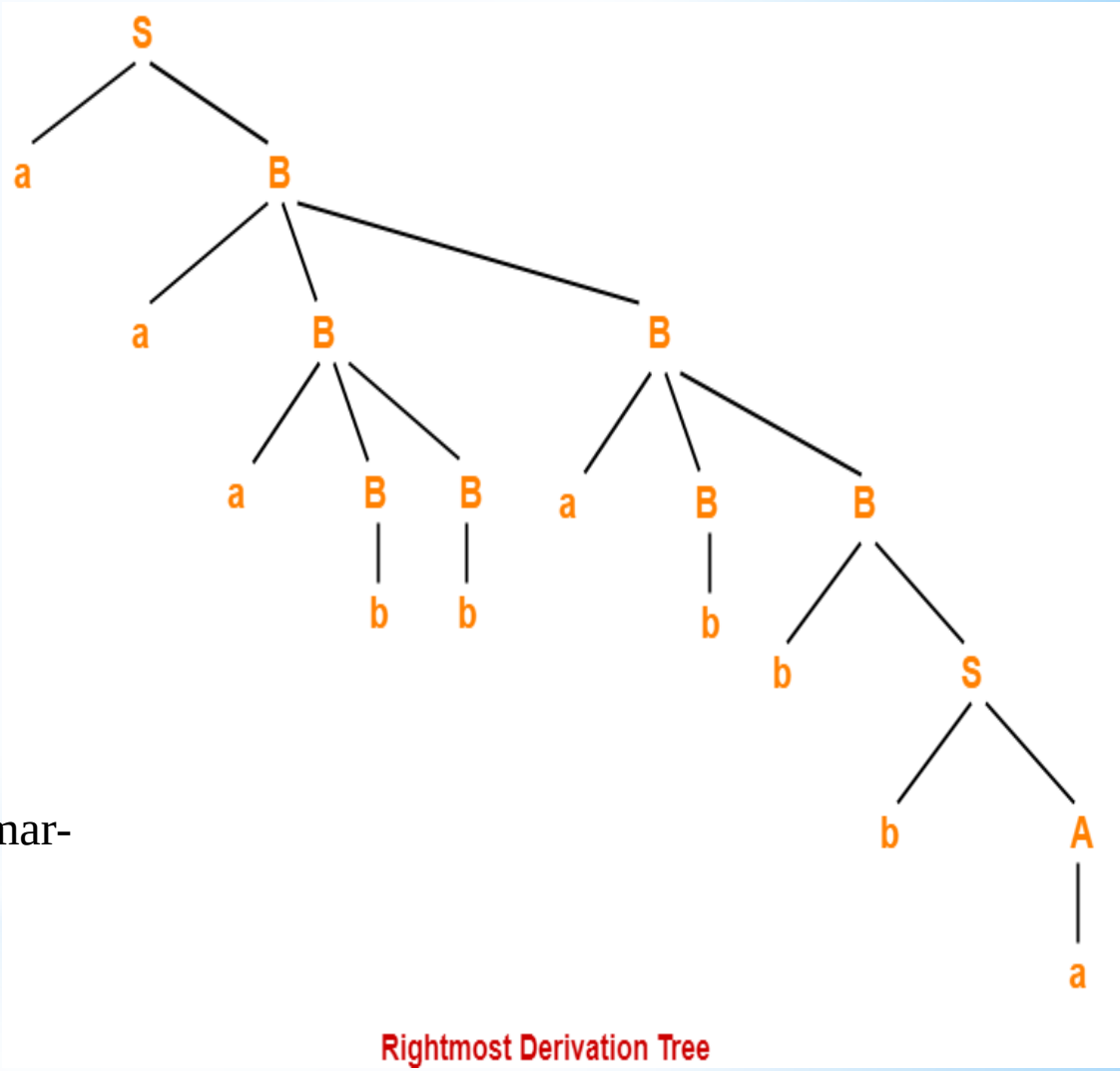
Example-

Consider the following grammar-
S → aB / bA
A → aS / bAA / a
B → bS / aBB / b
(**Unambiguous Grammar**)

Let us consider a string w = aaabbabbba
Now, let us derive the string w using rightmost derivation.

Rightmost Derivation-

S → a**B**
→ aa**B**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

Example-

Consider the following grammar-
S → aB / bA
A → aS / bAA / a
B → bS / aBB / b
(**Unambiguous Grammar**)

Let us consider a string w = aaabbabbba

NOTES

For unambiguous grammars, Leftmost derivation and Rightmost derivation represents the same parse tree.

For ambiguous grammars, Leftmost derivation and Rightmost derivation represents different parse trees.

Here,

The given grammar was unambiguous.

That is why, leftmost derivation and rightmost derivation represents the same parse tree.

Leftmost Derivation Tree = Rightmost Derivation Tree

Properties Of Parse Tree-

1.Root node of a parse tree is the start symbol of the grammar.
2.Each leaf node of a parse tree represents a terminal symbol.
3.Each interior node of a parse tree represents a non-terminal symbol.
4.Parse tree is independent of the order in which the productions are used during derivations.

Yield Of Parse Tree-

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**.

PRACTICE PROBLEMS BASED ON DERIVATIONS AND PARSE TREE-

Problem-01:

Consider the grammar-
$S \rightarrow bB / aA$
$A \rightarrow b / bS / aAA$
$B \rightarrow a / aS / bBB$

For the string w = bbaababa, find-
Leftmost derivation
Rightmost derivation
Parse Tree

Solution-

Consider the grammar-

$S \rightarrow bB / aA$

$A \rightarrow b / bS / aAA$

$B \rightarrow a / aS / bBB$

For the string w = bbaababa

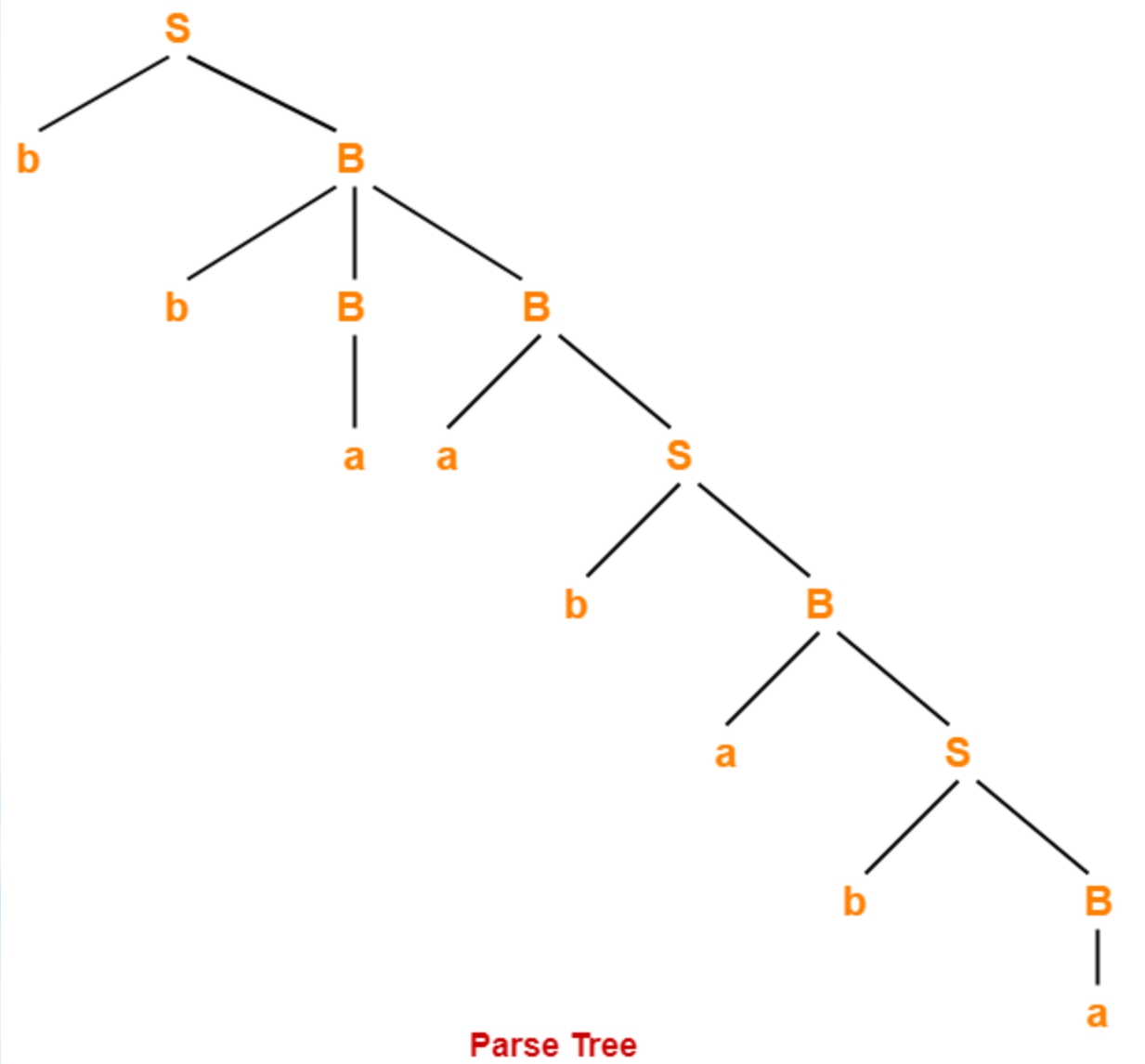
1. Leftmost Derivation-

$S \rightarrow b\mathbf{B}$
$\rightarrow bb\mathbf{B}B$ (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 $B \rightarrow a$)

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-
$S \rightarrow A1B$
$A \rightarrow 0A / \in$
$B \rightarrow 0B / 1B / \in$

For the string w = 00101, find-
Leftmost derivation
Rightmost derivation
Parse Tree

Solution-

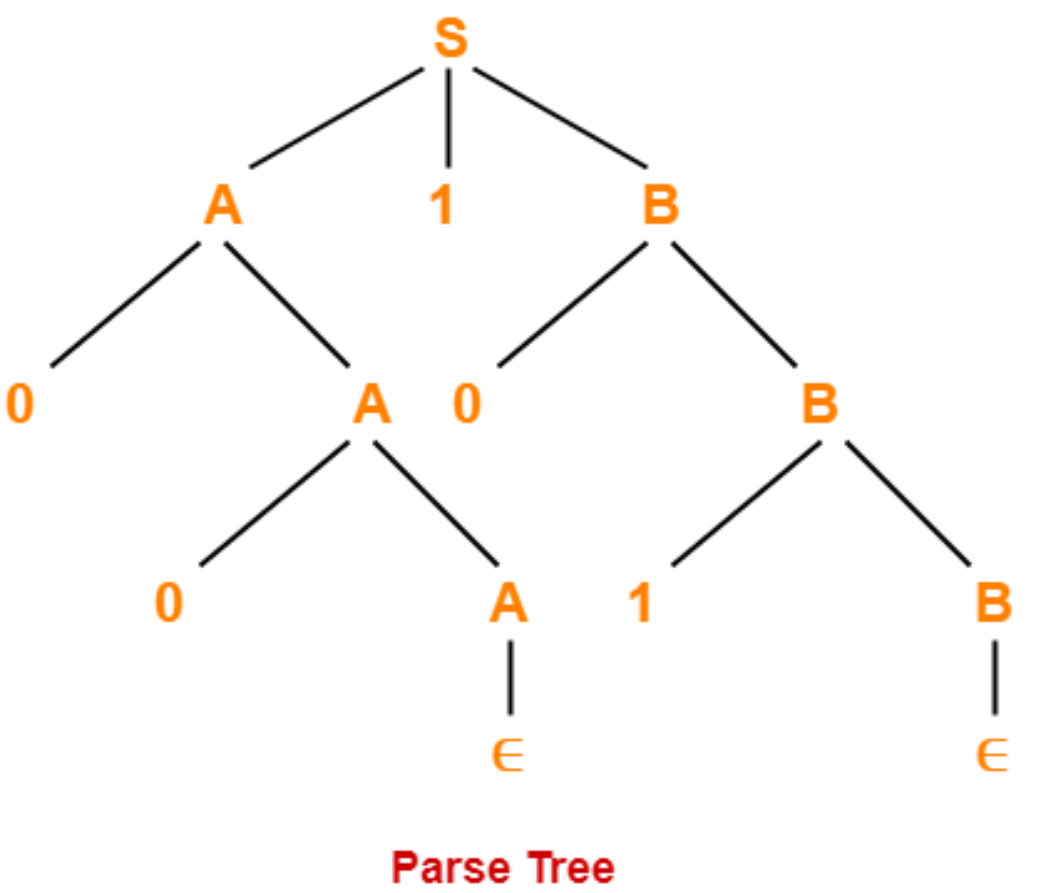
1. Leftmost Derivation-

$S \rightarrow \mathbf{A}1B$
$\rightarrow 0\mathbf{A}1B$ (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-

$S \rightarrow A1B$
$\rightarrow A10B$ (Using $B \rightarrow 0B$)
$\rightarrow A101B$ (Using $B \rightarrow 1B$)
$\rightarrow A101$ (Using $B \rightarrow \in$)
$\rightarrow 0A101$ (Using $A \rightarrow 0A$)
$\rightarrow 00A101$ (Using $A \rightarrow 0A$)
$\rightarrow 00101$ (Using $A \rightarrow \in$)

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.

To gain better understanding about Derivations and Parse Tree,

1. Ambiguous Grammar-

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

Example-

Consider the following grammar-
$E \rightarrow E + E \mid E \times E \mid id$
Ambiguous Grammar

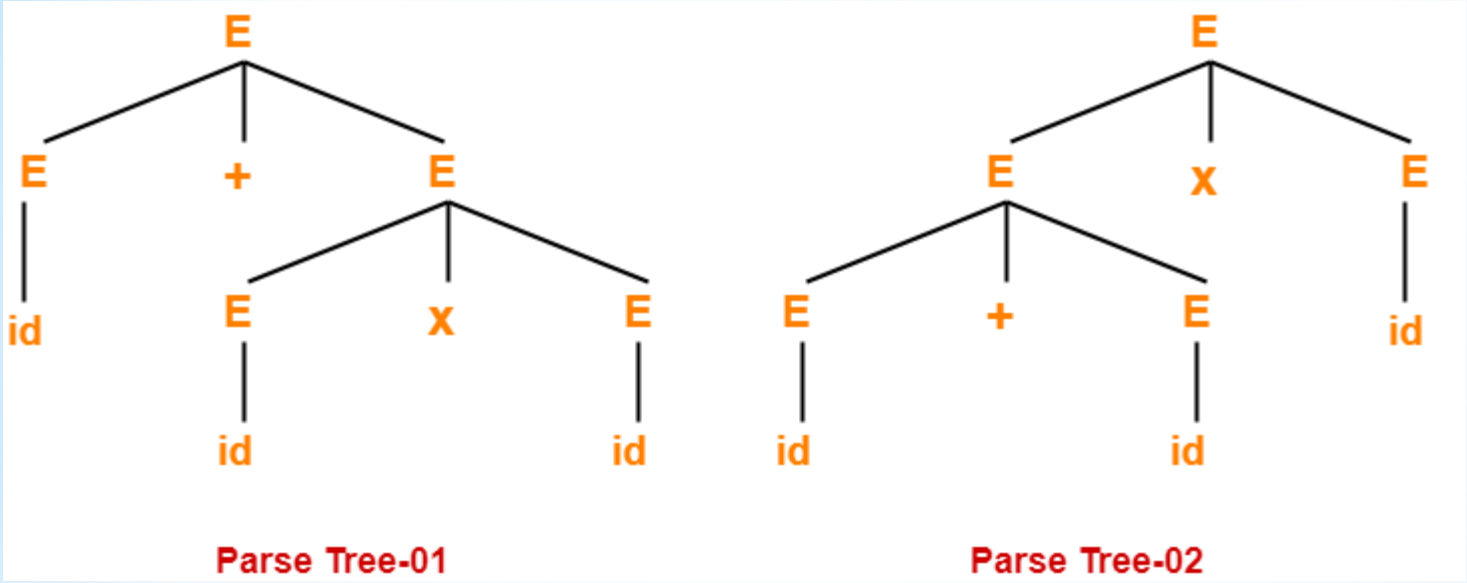
This grammar is an example of ambiguous grammar.
Any of the following reasons can be stated to prove the grammar ambiguous-

Reason-01:

Let us consider a string w generated by the grammar-

w = id + id x id

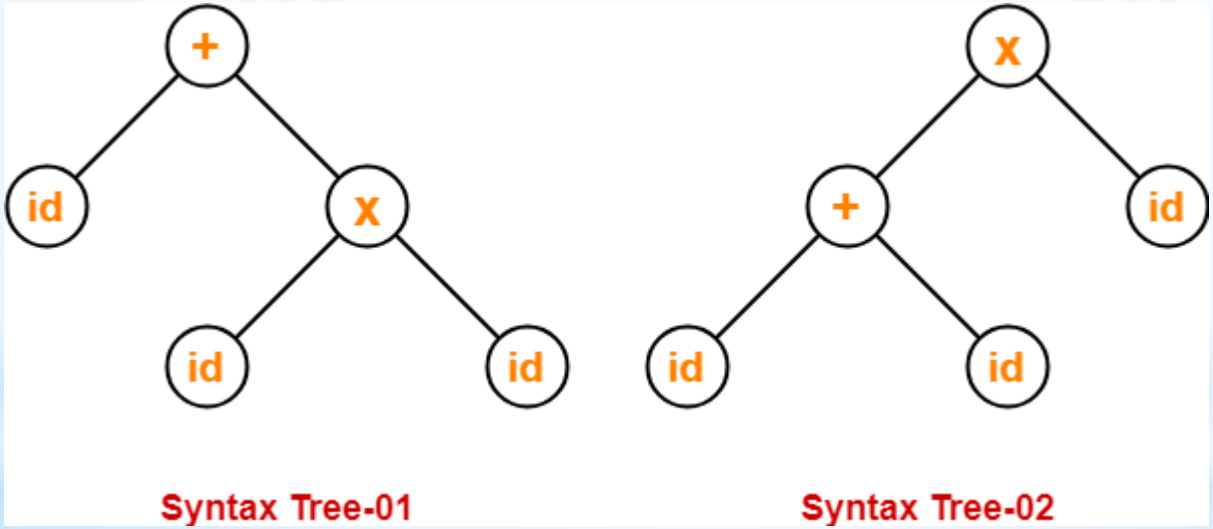
Now, let us draw the parse trees for this string w. $E \rightarrow E + E \mid E \times E \mid id$



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

Reason-02:

Let us consider a string w generated by the grammar-
w = id + id x id
Now, let us draw the syntax trees for this string w.



Since two syntax trees exist for string w, therefore the grammar is ambiguous.

Reason-03:

Let us consider a string w generated by the grammar-
w = id + id x id
Now, let us write the leftmost derivations for this string w.

Leftmost Derivation-01	Leftmost Derivation-02
E → **E** + E	E → **E** x E
→ id + **E**	→ **E** + E x E
→ id + **E** x E	→ id + **E** x E
→ id + id x **E**	→ id + id x **E**
→ id + id x id	→ id + id x id

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

Reason-04:

Let us consider a string w generated by the grammar-
w = id + id x id
Now, let us write the rightmost derivations for this string

$E \rightarrow E + \mathbf{E}$
$\rightarrow E + E \times \mathbf{E}$
$\rightarrow E + \mathbf{E} \times id$
$\rightarrow \mathbf{E} + id \times id$
$\rightarrow id + id \times id$

Rightmost Derivation-01

$E \rightarrow E \times \mathbf{E}$
$\rightarrow \mathbf{E} \times id$
$\rightarrow E + \mathbf{E} \times id$
$\rightarrow \mathbf{E} + id \times id$
$\rightarrow id + id \times id$

Rightmost Derivation-02

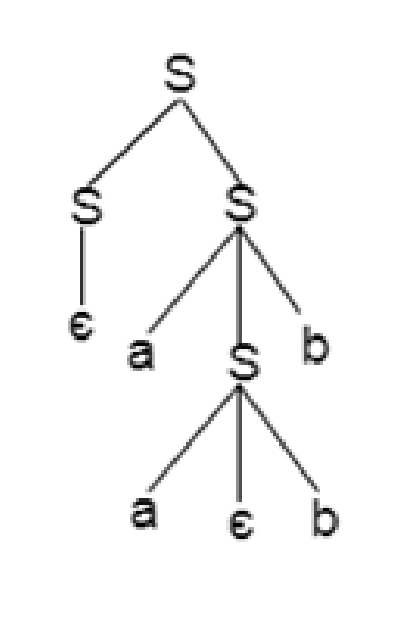
Since two rightmost derivations exist for string w, therefore the grammar is ambiguous.

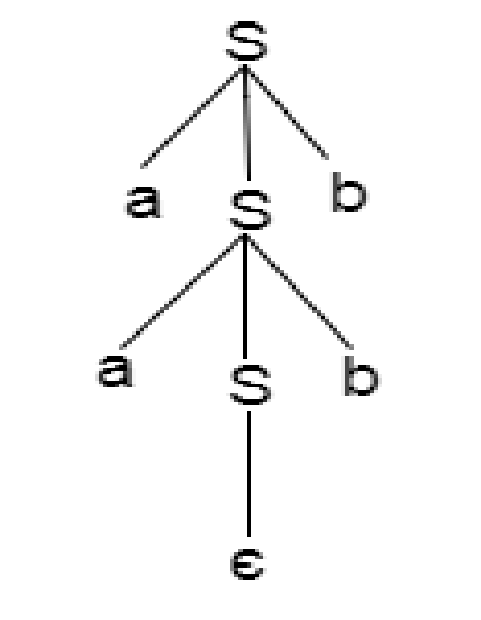
Example:

S = aSb | SS

S = ∈

For the string aabb, the above grammar generates two parse trees:





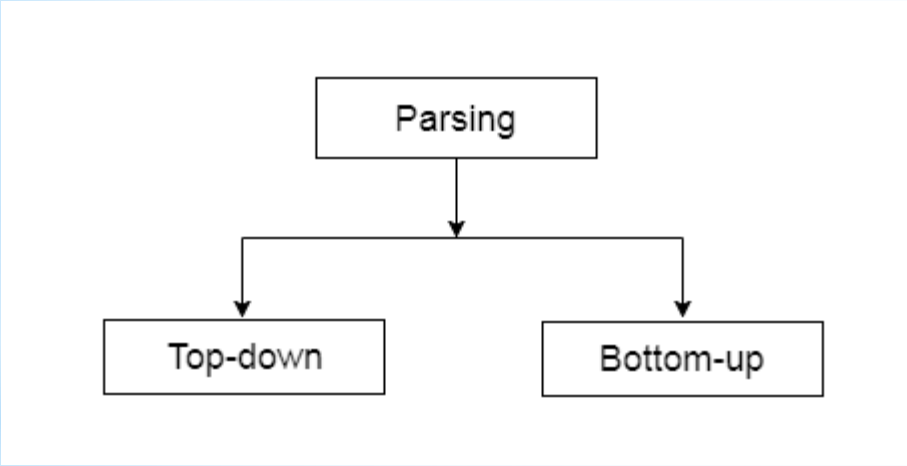
If the grammar has ambiguity then it is not good for a compiler construction. No method can automatically detect and remove the ambiguity but you can remove ambiguity by re-writing the whole grammar without ambiguity.

Parser

Parser is a compiler that is used to break the data into smaller elements coming from lexical analysis phase.

A parser takes input in the form of sequence of tokens and produces output in the form of parse tree.

Parsing is of two types: top down parsing and bottom up parsing.



Lexical analysis is the **first phase of a compiler**. ... The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer.

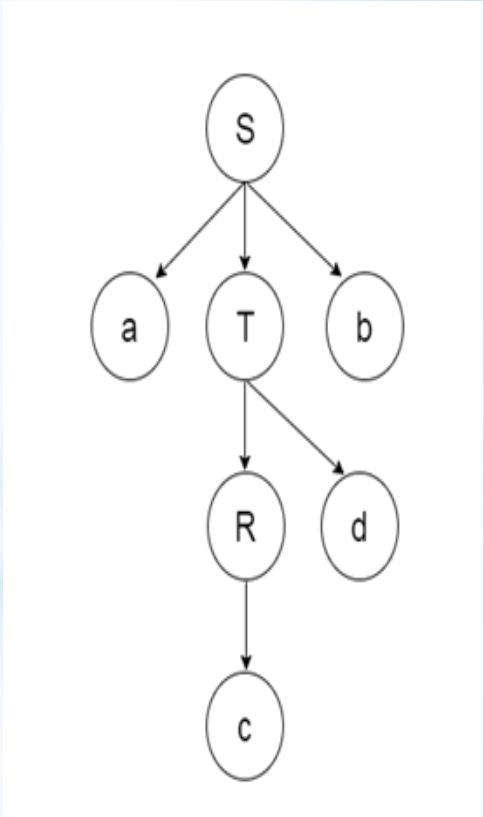
Top down paring

The top down parsing is known as recursive parsing or predictive parsing.

Bottom up parsing is used to construct a parse tree for an input string.

In the top down parsing, the parsing starts from the start symbol and transform it into the input symbol.

Parse Tree representation of input string "acdb" is as follows



Bottom up parsing

Bottom up parsing is also known as shift-reduce parsing.

Bottom up parsing is used to construct a parse tree for an input string.

In the bottom up parsing, the parsing starts with the input symbol and construct the parse tree up to the start symbol by tracing out the rightmost derivations of string in reverse.

Example

Production

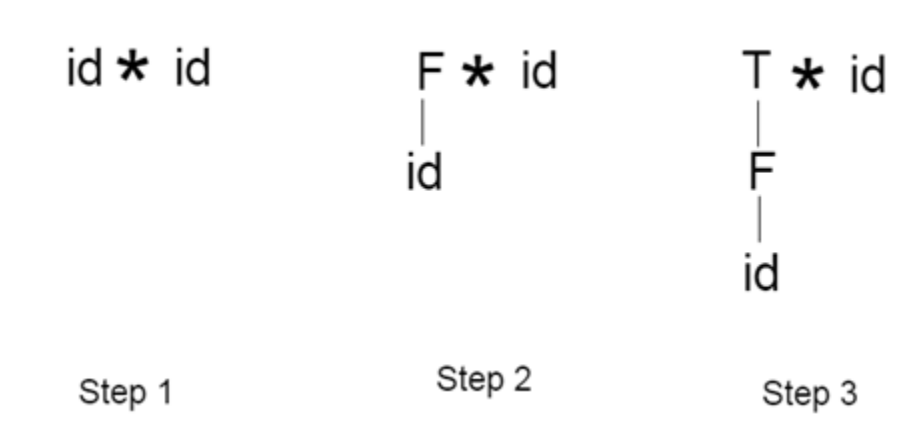
E → T

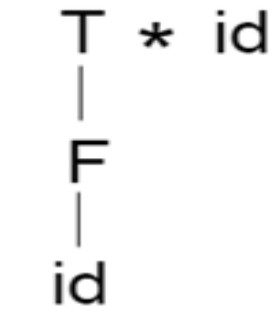
T → T * F

T → id

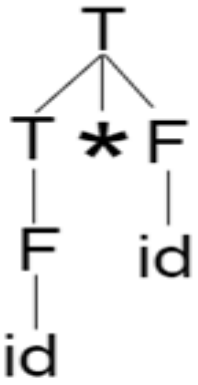
F → T

F → id

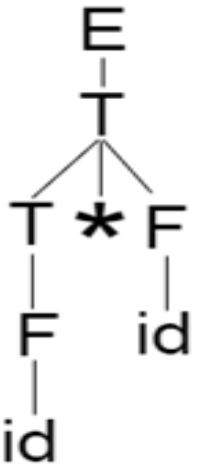




Step 4



Step 5



Step 6

Bottom up parsing is classified in to various parsing. These are as follows:

Shift-Reduce Parsing

Operator Precedence Parsing

Table Driven LR Parsing

LR(1)

SLR(1)

CLR (1)

LALR(1)

What is a grammar in formal languages and automata theory?
In automata theory, a formal language is **a set of strings of symbols drawn from a finite alphabet**. A formal language can be specified either by a set of rules (such as regular expressions or a context-free grammar) that generates the language, or by a formal machine that accepts (recognizes) the language.

Sentential Form and Partial Derivation Tree

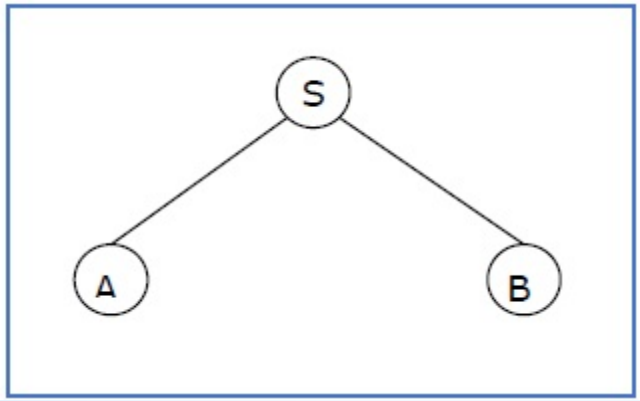
A partial derivation tree is a sub-tree of a derivation tree/parse tree such that either all of its children are in the sub-tree or none of them are in the sub-tree.

Example

If in any CFG the productions are −

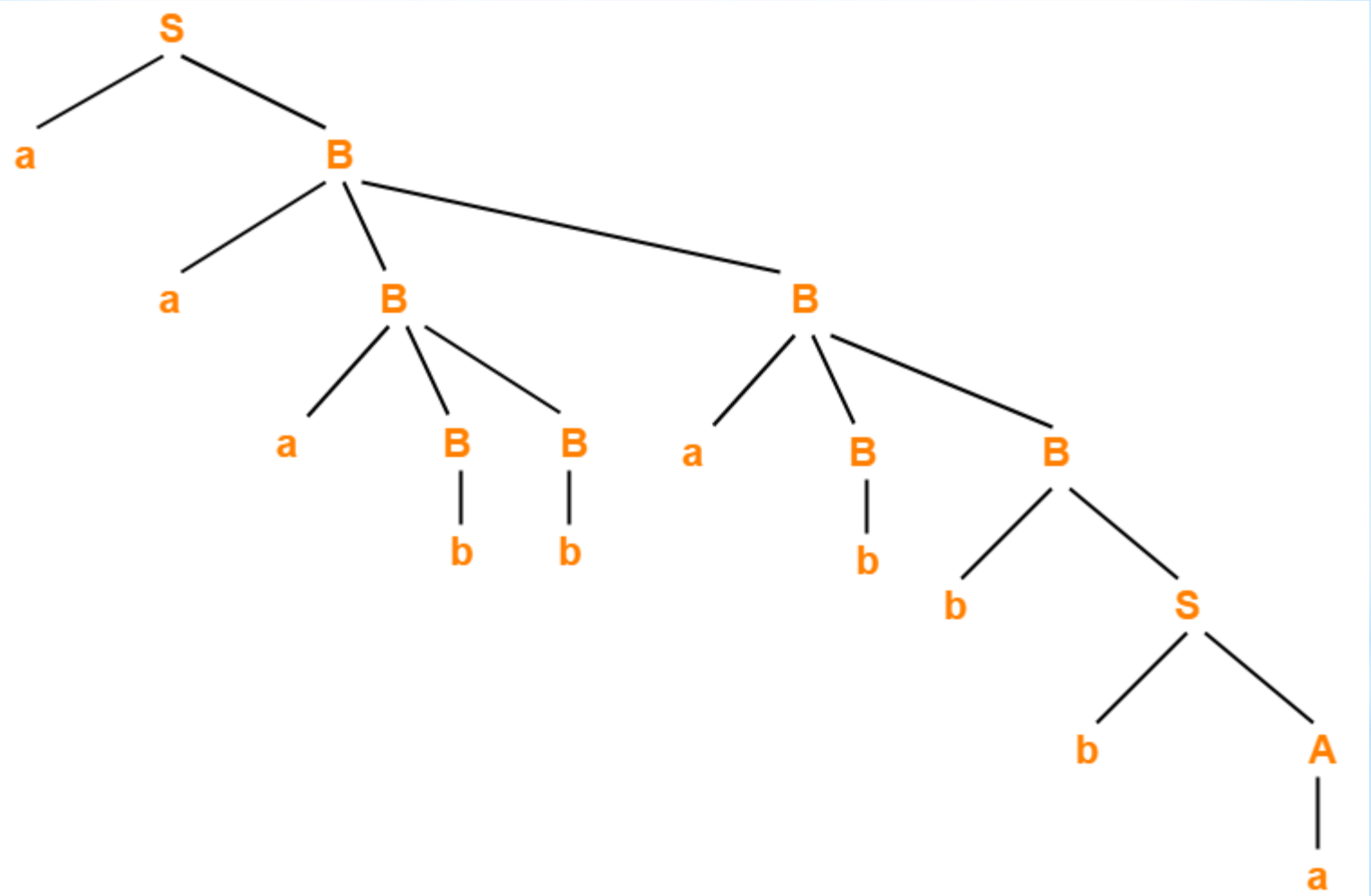
$S \rightarrow AB$, $A \rightarrow aaA \mid \varepsilon$, $B \rightarrow Bb \mid \varepsilon$

the partial derivation tree can be the following −



If a partial derivation tree contains the root S, it is called a **sentential form**. The above sub-tree is also in sentential form.

A parse tree or parsing tree or derivation tree or concrete syntax tree is **an ordered, rooted tree that represents the syntactic structure of a string according to some context**-free grammar.



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:

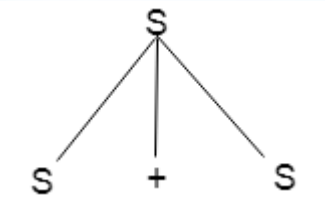
S= S + S | S * S

S = a|b|c

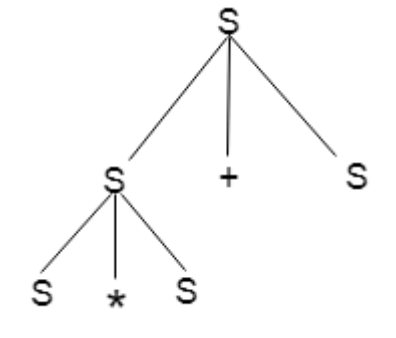
Input:

a * b + c

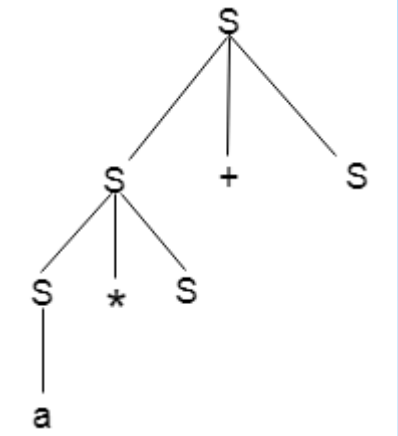
Step 1:



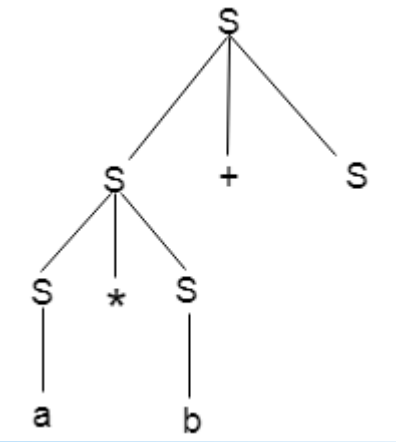
Step 2:



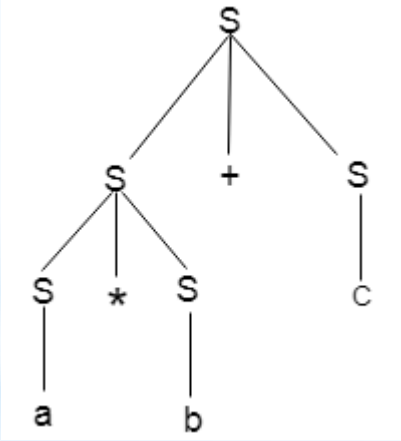
Step 3:



Step 4:



Step 5:



Applications-

For defining programming languages.
For parsing the program by constructing syntax tree.
For translation of programming languages.
For describing arithmetic expressions.
For construction of compilers.