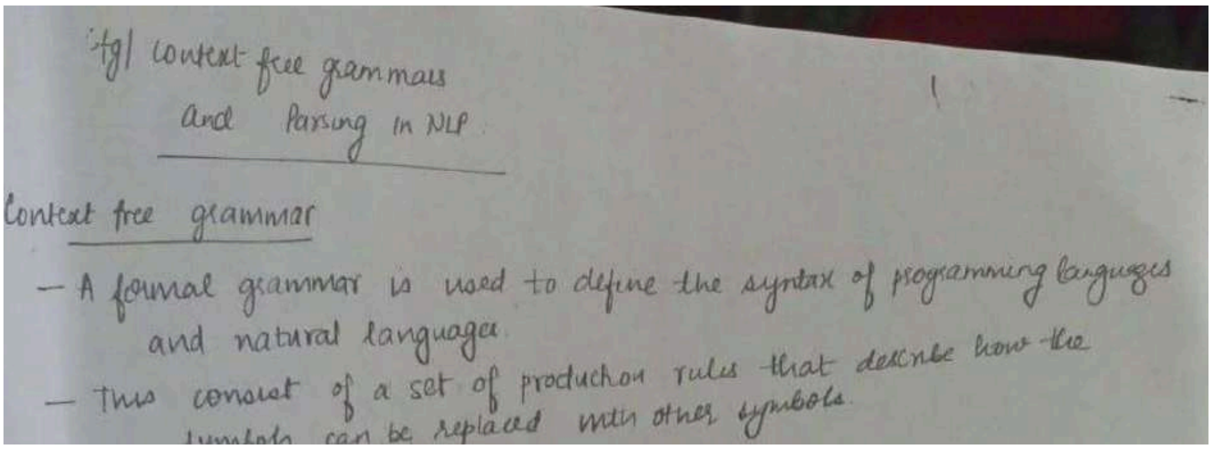
Context free Grammar

Context Free Grammar is formal grammar, the syntax or structure of a formal language can be described using context-free grammar (CFG), a type of formal grammar.



tg| context free grammars
And Parsing in NLP

Context free grammar

- A formal grammar is used to define the syntax of programming languages and natural language.
- This consist of a set of production rules that describe how the [illegible] can be replaced with other symbols.

Components of CFG

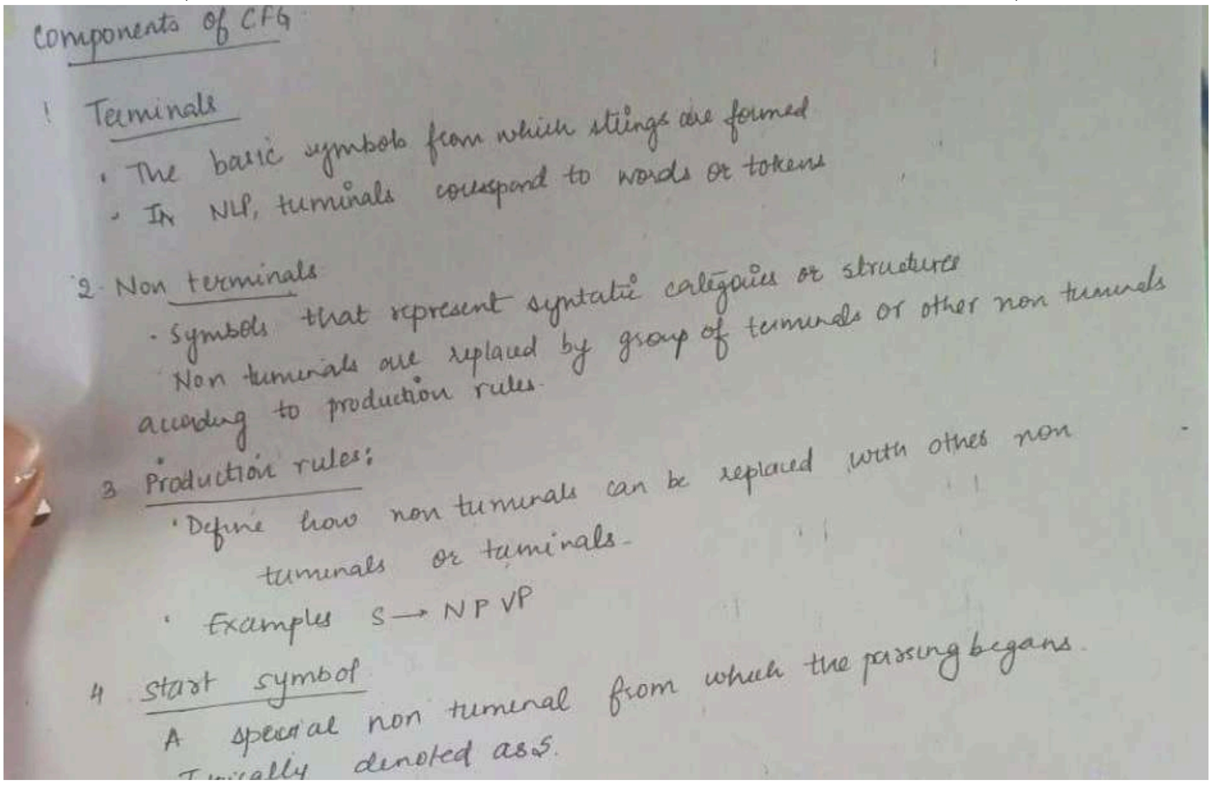
The grammar has four tuples: (V,T,P,S).

`V - It is the collection of variables or non-terminal symbols.`

`T - It is a set of terminals.`

`P - It is the production rules that consist of both terminals and non-terminals.`

`S - It is the starting symbol.`



Components of CFG

1 Terminals
- The basic symbols from which strings are formed
- In NLP, terminals correspond to words or tokens

2 Non terminals
- Symbols that represent syntatic categories or structures
  Non terminals are replaced by group of terminals or other non terminals according to production rules.

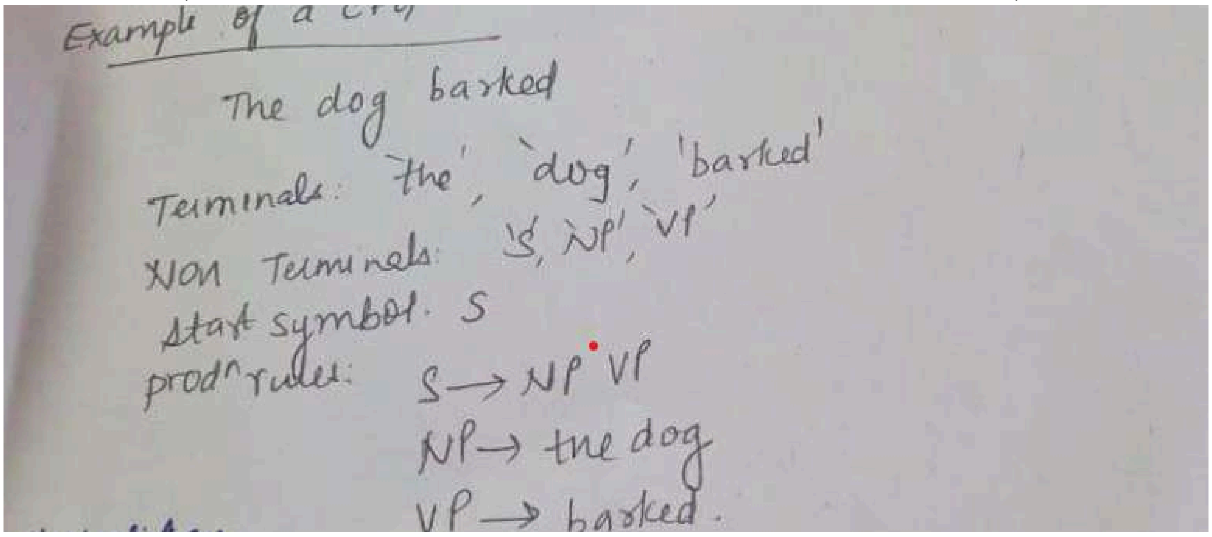
3 Production rules:
- Define how non terminals can be replaced with other non terminals or terminals.
- Examples $S \rightarrow NP\ VP$

4 Start symbol
A special non terminal from which the parsing begans.
[illegible] denoted as S.

Example of CFG

For example, the grammar A = { S, a, b } having productions:

- Here S is the starting symbol.
- {a, b} are the terminals generally represented by small characters.
- S is the variable.



Example of a cfg

The dog barked

Terminals: 'the', 'dog', 'barked'

Non Terminals: 'S', 'NP', 'VP'

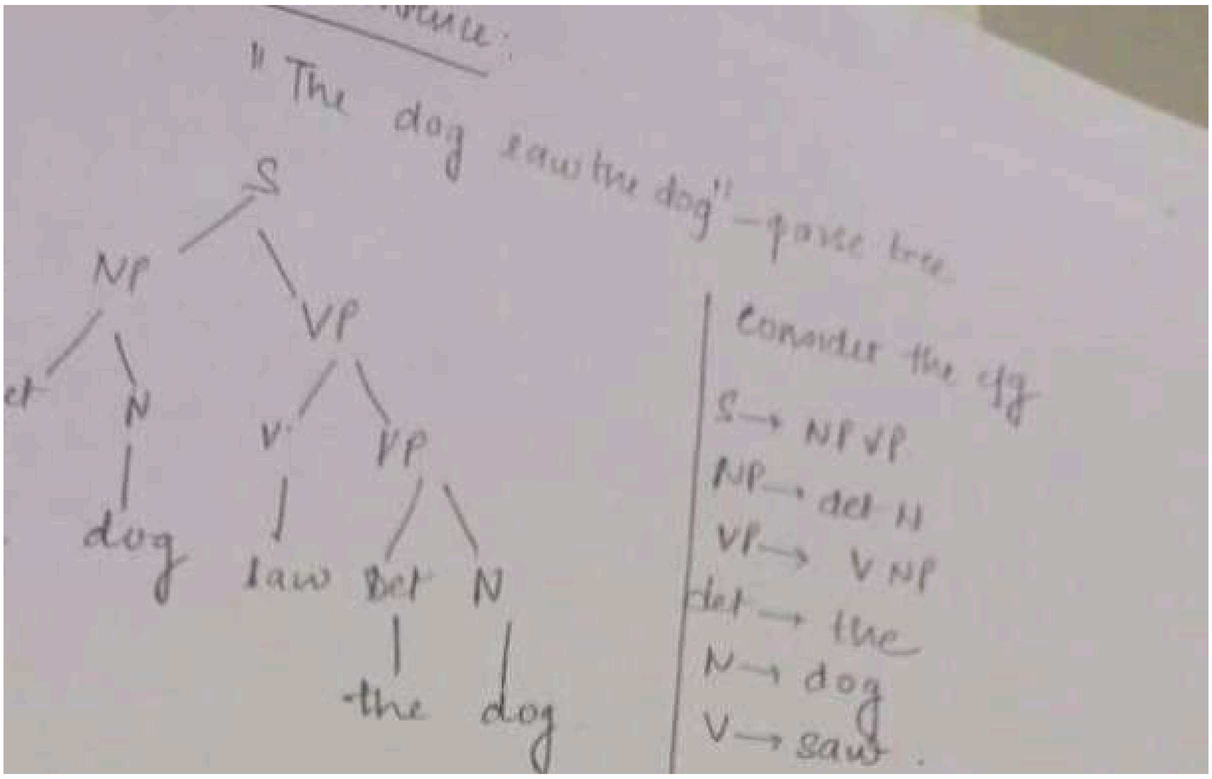
start symbol: S

prod^n rules: $S \rightarrow NP\ VP$

$NP \rightarrow$ the dog

$VP \rightarrow$ barked.

Example of parse and derivation tree



"The dog saw the dog" – parse tree

Consider the cfg

$S \rightarrow NP\ VP$

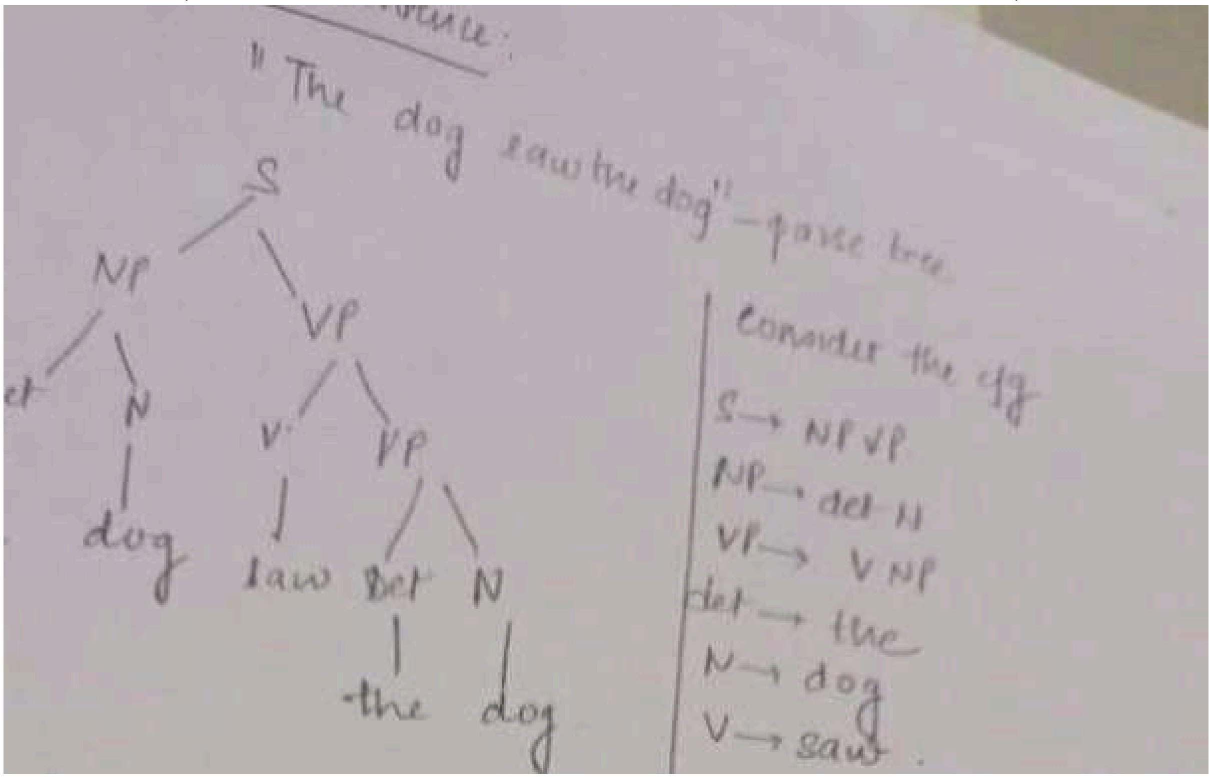
$NP \rightarrow det\ N$

$VP \rightarrow V\ NP$

$det \rightarrow$ the

$N \rightarrow$ dog

$V \rightarrow$ saw.



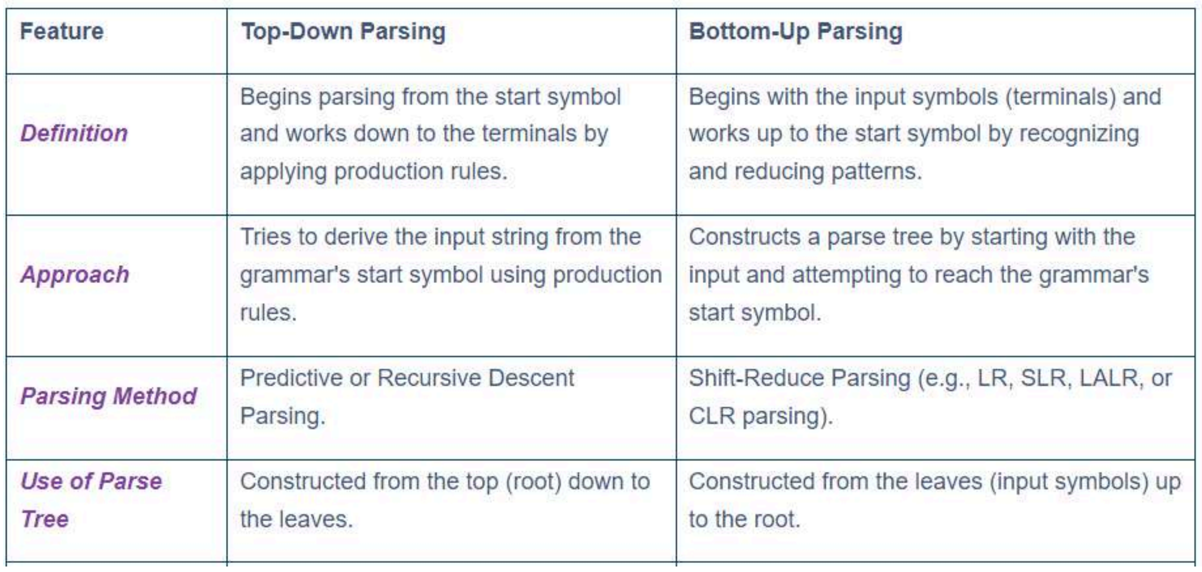
What is parsing

Parsing in NLP refers to the process of analyzing the structure and relationships between words in a text. It involves identifying how words combine to form phrases or constituents and how those phrases recursively combine to form a tree structure for the whole input. An example-based approach to parsing relies on examples of parsed sentences to extract the required information for parsing new sentences
This approach involves annotating examples with a representation tree and the correspondence between substrings in the sentence and subtrees in the representation tree. The parsing process then involves building subtrees for phrases in the input sentence based on successful matches in the example base, and combining these subtrees to form a single rooted representation tree

1

1

Difference between Top Down and Bottom up parsing



Feature	Top-Down Parsing	Bottom-Up Parsing
Definition	Begins parsing from the start symbol and works down to the terminals by applying production rules.	Begins with the input symbols (terminals) and works up to the start symbol by recognizing and reducing patterns.
Approach	Tries to derive the input string from the grammar's start symbol using production rules.	Constructs a parse tree by starting with the input and attempting to reach the grammar's start symbol.
Parsing Method	Predictive or Recursive Descent Parsing.	Shift-Reduce Parsing (e.g., LR, SLR, LALR, or CLR parsing).
Use of Parse Tree	Constructed from the top (root) down to the leaves.	Constructed from the leaves (input symbols) up to the root.

What is Top Down Parsing?

Top-down parsing is a strategy used in syntax analysis and parsing in the field of compiler design, where the parsing process starts from the highest-level construct (the start symbol) and works its way down to the individual tokens (the leaves of the parse tree).

What is Bottom Up Parsing?

Bottom-up parsing is a method used in compiler design for syntax analysis, where the parsing process starts with the leaf nodes of the parse tree (the input tokens) and works its way up to the root (the start symbol of the grammar).

This approach constructs the parse tree from the bottom up by recognizing and reducing sequences of tokens into grammatical constructs, according to the rules of a

given grammar, until the entire sequence is reduced to the start symbol. It effectively builds the parse tree by starting with the details (tokens) and combining them into higher-level constructs until it constructs the whole program.

Applications of CFG:(Context free Grammar)

- 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