Page 1

UNIT III

SYNTACTIC ANALYSIS Context-Free Grammars, Grammar rules for English, Treebanks, Normal Forms for grammar – Dependency Grammar – Syntactic Parsing, Ambiguity, Dynamic Programming parsing – Shallow parsing – Probabilistic CFG, Probabilistic CYK, Probabilistic Lexicalized CFGs - Feature structures, Unification of feature structures.

Context-Free Grammars
Natural Language Processing (NLP) is the capacity of a computer to "understand" natural language text at a level that allows meaningful interaction between the computer and a person working in a particular application domain.

Application Domains of NLP:

- text processing - word processing, e-mail, spelling and grammar checkers
- interfaces to data bases - query languages, information retrieval, data mining, text summarization
- expert systems - explanations, disease diagnosis
- linguistics - machine translation, content analysis, writers' assistants, language generation

Tools for NLP:

- Programming languages and software - Prolog , ALE , Lisp/Scheme, C/C++
- Statistical Methods - Markov models, probabilistic grammars, text-based analysis
- Abstract Models - Context-free grammars (BNF), Attribute grammars, Predicate calculus and other semantic models, Knowledge-based and ontological methods

Linguistic Organization of NLP

- Grammar and lexicon - the rules for forming well-structured sentences, and the words that make up those sentences
- Morphology - the formation of words from stems, prefixes, and suffixes

E.g., eat + s = eats

- Syntax - the set of all well-formed sentences in a language and the rules for forming them

- Semantics - the meanings of all well-formed sentences in a language

- Pragmatics (world knowledge and context) - the influence of what we know about the real world upon the meaning of a sentence. E.g., "The balloon rose." allows an inference to be made that it must be filled with a lighter-than-air substance.

- The influence of discourse context (E.g., speaker-hearer roles in a conversation) on the meaning of a sentence

- Ambiguity
  - lexical - word meaning choices (E.g., *flies*)
  - syntactic - sentence structure choices (E.g., *She saw the man on the hill with the telescope*.)
  - semantic - sentence meaning choices (E.g., *They are flying planes.*)

Grammars and parsing

Syntactic categories (common denotations) in NLP

- np - noun phrase
- vp - verb phrase

- s - sentence
- det - determiner (article)
- n - noun
- tv - transitive verb (takes an object)
- iv - intransitive verb
- prep - preposition
- pp - prepositional phrase
- adj - adjective

A *context-free grammar (CFG)* is a list of rules that define the set of all well-formed sentences in a language. Each rule has a left-hand side, which identifies a syntactic category, and a right-hand side, which defines its alternative component parts, reading from left to right.

> E.g., the rule s --> np vp means that "a sentence is defined as a noun phrase followed by a verb phrase." Figure 1 shows a simple CFG that describes the sentences from a small subset of English.

Figure 1. A grammar and a parse tree for "the giraffe dreams".

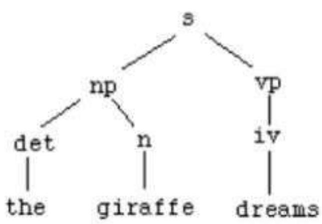
```
s    --> np vp

np   --> det n
vp   --> tv np
     --> iv

det  --> the
     --> a
     --> an

n    --> giraffe
     --> apple

iv   --> dreams
tv   --> eats
     --> dreams
```



A *sentence* in the language defined by a CFG is a series of words that can be derived by systematically applying the rules, beginning with a rule that has s on its left-hand side. A *parse* of the sentence is a series of rule applications in which a syntactic category is replaced by the right-hand side of a rule that has that category on its left-hand side, and the final rule application yields the sentence itself. E.g., a parse of the sentence "the giraffe dreams" is:

s => np vp => det n vp => the n vp => the giraffe vp => the giraffe iv => the giraffe dreams

A convenient way to describe a parse is to show its *parse tree*, which is simply a graphical display of the parse. Figure 1 shows a parse tree for the sentence "the giraffe dreams". Note that the root of every subtree has a grammatical category that appears on the left-hand side of a rule, and the children of that root are identical to the elements on the right-hand side of that rule.

> If this looks like familiar territory from your study of programming languages, that's a good observation. CFGs are, in fact, the orignin of the device called BNF (Backus-Naur Form) for describing the syntax of programming languages. CFGs were invented by the linguist Noam Chomsky in 1957. BNF originated with the design of the Algol programming language in 1960.

Grammar rules for English

Treebank

Corpus

A corpus is a large and structured set of machine-readable texts that have been produced in a natural communicative setting. Its plural is corpora. They can be derived in different ways like text that was originally electronic, transcripts of spoken language and optical character recognition, etc.

Elements of Corpus Design

Language is infinite but a corpus has to be finite in size. For the corpus to be finite in size, we need to sample and proportionally include a wide range of text types to ensure a good corpus design.

Let us now learn about some important elements for corpus design −

Corpus Representativeness

Representativeness is a defining feature of corpus design. The following definitions from two great researchers − Leech and Biber, will help us understand corpus representativeness −

- **According to Leech (1991),** "A corpus is thought to be representative of the language variety it is supposed to represent if the findings based on its contents can be generalized to the said language variety".
- **According to Biber (1993),** "Representativeness refers to the extent to which a sample includes the full range of variability in a population".

In this way, we can conclude that representativeness of a corpus are determined by the following two factors −

- **Balance** − The range of genre include in a corpus
- **Sampling** − How the chunks for each genre are selected.

Corpus Balance

Another very important element of corpus design is corpus balance – the range of genre included in a corpus. We have already studied that representativeness of a general corpus depends upon how balanced the corpus is. A balanced corpus covers a wide range of text categories, which are supposed to be representatives of the language. We do not have any reliable scientific measure for balance but the best estimation and intuition works in this concern. In other words, we can say that the accepted balance is determined by its intended uses only.

Sampling

Another important element of corpus design is sampling. Corpus representativeness and balance is very closely associated with sampling. That is why we can say that sampling is inescapable in corpus building.

- According to **Biber(1993)**, "Some of the first considerations in constructing a corpus concern the overall design: for example, the kinds of texts included, the number of texts, the selection of particular texts, the selection of text samples from within texts, and the length of text samples. Each of these involves a sampling decision, either conscious or not."

While obtaining a representative sample, we need to consider the following −

- **Sampling unit** − It refers to the unit which requires a sample. For example, for written text, a sampling unit may be a newspaper, journal or a book.
- **Sampling frame** − The list of al sampling units is called a sampling frame.
- **Population** − It may be referred as the assembly of all sampling units. It is defined in terms of language production, language reception or language as a product.

Corpus Size

Another important element of corpus design is its size. How large the corpus should be? There is no specific answer to this question. The size of the corpus depends upon the purpose for which it is intended as well as on some practical considerations as follows −

- Kind of query anticipated from the user.
- The methodology used by the users to study the data.
- Availability of the source of data.

With the advancement in technology, the corpus size also increases. The following table of comparison will help you understand how the corpus size works −

Year	Name of the Corpus	Size (in words)
1960s - 70s	Brown and LOB	1 Million words

1980s	The Birmingham corpora	20 Million words
1990s	The British National corpus	100 Million words
Early 21st century	The Bank of English corpus	650 Million words

In our subsequent sections, we will look at a few examples of corpus.

TreeBank Corpus

It may be defined as linguistically parsed text corpus that annotates syntactic or semantic sentence structure. Geoffrey Leech coined the term 'treebank', which represents that the most common way of representing the grammatical analysis is by means of a tree structure. Generally, Treebanks are created on the top of a corpus, which has already been annotated with part-of-speech tags.

Types of TreeBank Corpus

Semantic and Syntactic Treebanks are the two most common types of Treebanks in linguistics. Let us now learn more about these types −

Semantic Treebanks

These Treebanks use a formal representation of sentence's semantic structure. They vary in the depth of their semantic representation. Robot Commands Treebank, Geoquery, Groningen Meaning Bank, RoboCup Corpus are some of the examples of Semantic Treebanks.

Syntactic Treebanks

Opposite to the semantic Treebanks, inputs to the Syntactic Treebank systems are expressions of the formal language obtained from the conversion of parsed Treebank data. The outputs of such systems are predicate logic based meaning representation. Various syntactic Treebanks in different languages have been created so far. For example, **Penn Arabic Treebank, Columbia Arabic Treebank** are syntactic Treebanks created in Arabia language. **Sininca** syntactic Treebank created in Chinese language. **Lucy, Susane** and **BLLIP WSJ** syntactic corpus created in English language.

Applications of TreeBank Corpus

Followings are some of the applications of TreeBanks −

In Computational Linguistics

If we talk about Computational Linguistic then the best use of TreeBanks is to engineer state-of-the-art natural language processing systems such as part-of-speech taggers, parsers, semantic analyzers and machine translation systems.

In Corpus Linguistics

In case of Corpus linguistics, the best use of Treebanks is to study syntactic phenomena.

In Theoretical Linguistics and Psycholinguistics

The best use of Treebanks in theoretical and psycholinguistics is interaction evidence.

PropBank Corpus

PropBank more specifically called "Proposition Bank" is a corpus, which is annotated with verbal propositions and their arguments. The corpus is a verb-oriented resource; the annotations here are more closely related to the syntactic level. Martha Palmer et al., Department of Linguistic, University of Colorado Boulder developed it. We can use the term PropBank as a common noun referring to any corpus that has been annotated with propositions and their arguments.

In Natural Language Processing (NLP), the PropBank project has played a very significant role. It helps in semantic role labeling.

Page 5

VerbNet(VN)

VerbNet(VN) is the hierarchical domain-independent and largest lexical resource present in English that incorporates both semantic as well as syntactic information about its contents. VN is a broad-coverage verb lexicon having mappings to other lexical resources such as WordNet, Xtag and FrameNet. It is organized into verb classes extending Levin classes by refinement and addition of subclasses for achieving syntactic and semantic coherence among class members.

Each VerbNet (VN) class contains −

A set of syntactic descriptions or syntactic frames

For depicting the possible surface realizations of the argument structure for constructions such as transitive, intransitive, prepositional phrases, resultatives, and a large set of diathesis alternations.

A set of semantic descriptions such as animate, human, organization

For constraining, the types of thematic roles allowed by the arguments, and further restrictions may be imposed. This will help in indicating the syntactic nature of the constituent likely to be associated with the thematic role.

WordNet

WordNet, created by Princeton is a lexical database for English language. It is the part of the NLTK corpus. In WordNet, nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms called **Synsets**. All the synsets are linked with the help of conceptual-semantic and lexical relations. Its structure makes it very useful for natural language processing (NLP).

In information systems, WordNet is used for various purposes like word-sense disambiguation, information retrieval, automatic text classification and machine translation. One of the most important uses of WordNet is to find out the similarity among words. For this task, various algorithms have been implemented in various packages like Similarity in Perl, NLTK in Python and ADW in Java.

Normal Forms for grammar

CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

Start symbol generating ε. For example, $A \rightarrow \varepsilon$.

A non-terminal generating two non-terminals. For example, $S \rightarrow AB$.

A non-terminal generating a terminal. For example, $S \rightarrow a$.

For example:

$G1 = \{S \rightarrow AB, S \rightarrow c, A \rightarrow a, B \rightarrow b\}$

$G2 = \{S \rightarrow aA, A \rightarrow a, B \rightarrow c\}$

The production rules of Grammar G1 satisfy the rules specified for CNF, so the grammar G1 is in CNF. However, the production rule of Grammar G2 does not satisfy the rules specified for CNF as $S \rightarrow aZ$ contains terminal followed by non-terminal. So the grammar G2 is not in CNF.

Steps for converting CFG into CNF

Step 1: Eliminate start symbol from the RHS. If the start symbol T is at the right-hand side of any production, create a new production as:

$S1 \rightarrow S$

Where S1 is the new start symbol.

Step 2: In the grammar, remove the null, unit and useless productions. You can refer to the **Simplification of CFG**.

Step 3: Eliminate terminals from the RHS of the production if they exist with other non-terminals or terminals. For example, production $S \rightarrow aA$ can be decomposed as:

S → RA

R → a

Step 4: Eliminate RHS with more than two non-terminals. For example, S → ASB can be decomposed as:

S → RS

R → AS

Example:

Convert the given CFG to CNF. Consider the given grammar G1:

S → a | aA | B

A → aBB | ε

B → Aa | b

Solution:

Step 1: We will create a new production S1 → S, as the start symbol S appears on the RHS. The grammar will be:

S1 → S

S → a | aA | B

A → aBB | ε

B → Aa | b

Step 2: As grammar G1 contains A → ε null production, its removal from the grammar yields:

S1 → S

S → a | aA | B

A → aBB

B → Aa | b | a

Now, as grammar G1 contains Unit production S → B, its removal yield:

S1 → S

S → a | aA | Aa | b

A → aBB

B → Aa | b | a

Also remove the unit production S1 → S, its removal from the grammar yields:

S0 → a | aA | Aa | b

S → a | aA | Aa | b

A → aBB

B → Aa | b | a

Step 3: In the production rule S0 → aA | Aa, S → aA | Aa, A → aBB and B → Aa, terminal a exists on RHS with non-terminals. So we will replace terminal a with X:

S0 → a | XA | AX | b

S → a | XA | AX | b

A → XBB

B → AX | b | a

X → a

Step 4: In the production rule A → XBB, RHS has more than two symbols, removing it from grammar yield:

S0 → a | XA | AX | b

S → a | XA | AX | b

A → RB

B → AX | b | a

X → a

R → XB

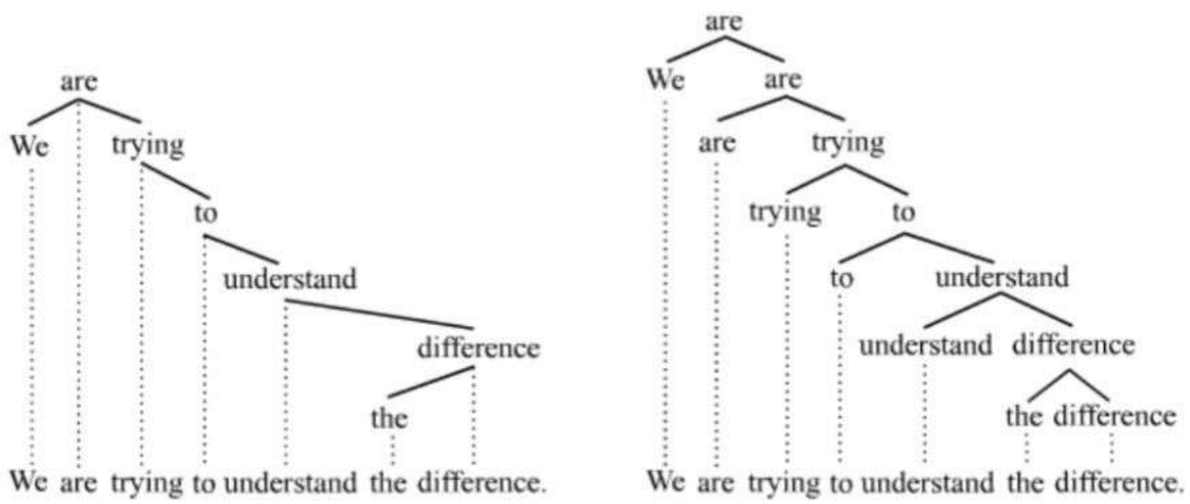
Hence, for the given grammar, this is the required CNF

Dependency grammar (DG)

is a class of modern grammatical theories that are all based on the dependency relation.

Dependency is the notion that linguistic units, e.g. words, are connected to each other by directed links. The (finite) verb is taken to be the structural center of clause structure. All other syntactic units (words) are either directly or indirectly connected to the verb in terms of the directed links, which are called *dependencies*. Dependency grammar differs from phrase structure grammar. A dependency structure is determined by the relation between a word (a head) and its dependents. Dependency structures are flatter than phrase structures in part because they lack a finite verb phrase constituent

Dependency is a one-to-one correspondence: for every element (e.g. word or morph) in the sentence, there is exactly one node in the structure of that sentence that corresponds to that element. The result of this one-to-one correspondence is that dependency grammars are word (or morph) grammars. All that exist are the elements and the dependencies that connect the elements into a structure. This situation should be compared with phrase structure. Phrase structure is a one-to-one-or-more correspondence, which means that, for every element in a sentence, there is one or more nodes in the structure that correspond to that element. The result of this difference is that dependency structures are minimal[7] compared to their phrase structure counterparts, since they tend to contain many fewer nodes.



Dependency

Constituency (BPS)

Page 8

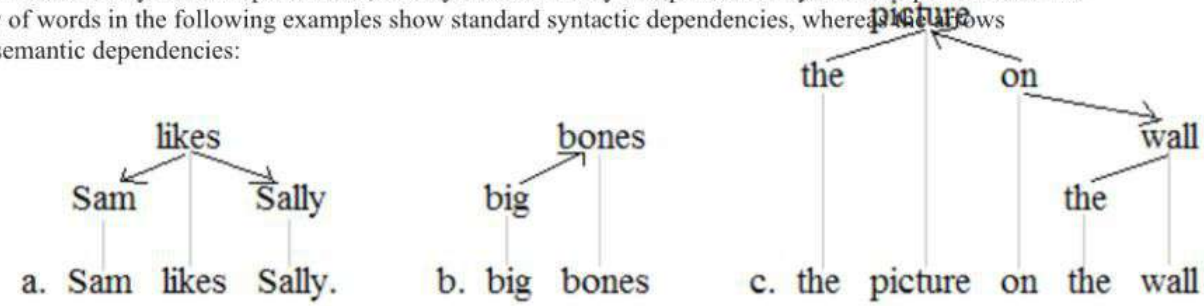
These trees illustrate two possible ways to render the dependency and phrase structure relations (see below). This dependency tree is an "ordered" tree, i.e. it reflects actual word order

Types of dependencies[edit]

The dependency representations above (and further below) show syntactic dependencies. Indeed, most work in dependency grammar focuses on syntactic dependencies. Syntactic dependencies are, however, just one of three or four types of dependencies. Meaning–text theory, for instance, emphasizes the role of semantic and morphological dependencies in addition to syntactic dependencies.[13] A fourth type, prosodic dependencies, can also be acknowledged. Distinguishing between these types of dependencies can be important, in part because if one fails to do so, the likelihood that semantic, morphological, and/or prosodic dependencies will be mistaken for syntactic dependencies is great. The following four subsections briefly sketch each of these dependency types. During the discussion, the existence of syntactic dependencies is taken for granted and used as an orientation point for establishing the nature of the other three dependency types.

Semantic dependencies[edit]

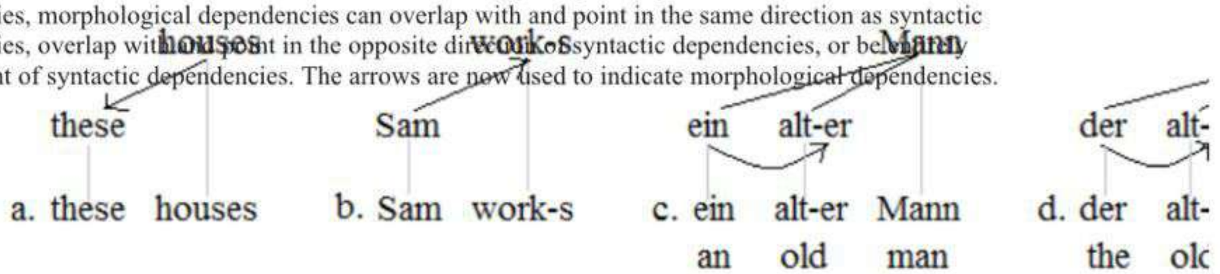
Semantic dependencies are understood in terms of predicates and their arguments.[14] The arguments of a predicate are semantically dependent on that predicate. Often, semantic dependencies overlap with and point in the same direction as syntactic dependencies. At times, however, semantic dependencies can point in the opposite direction of syntactic dependencies, or they can be entirely independent of syntactic dependencies. The hierarchy of words in the following examples show standard syntactic dependencies, whereas the arrows indicate semantic dependencies:



The two arguments *Sam* and *Sally* in tree (a) are dependent on the predicate *likes*, whereby these arguments are also syntactically dependent on *likes*. What this means is that the semantic and syntactic dependencies overlap and point in the same direction (down the tree). Attributive adjectives, however, are predicates that take their head noun as their argument, hence *big* is a predicate in tree (b) that takes *bones* as its one argument; the semantic dependency points up the tree and therefore runs counter to the syntactic dependency. A similar situation obtains in (c), where the preposition predicate *on* takes the two arguments *the picture* and *the wall*; one of these semantic dependencies points up the syntactic hierarchy, whereas the other points down it. Finally, the predicate *to help* in (d) takes the one argument *Jim* but is not directly connected to *Jim* in the syntactic hierarchy, which means that semantic dependency is entirely independent of the syntactic dependencies.

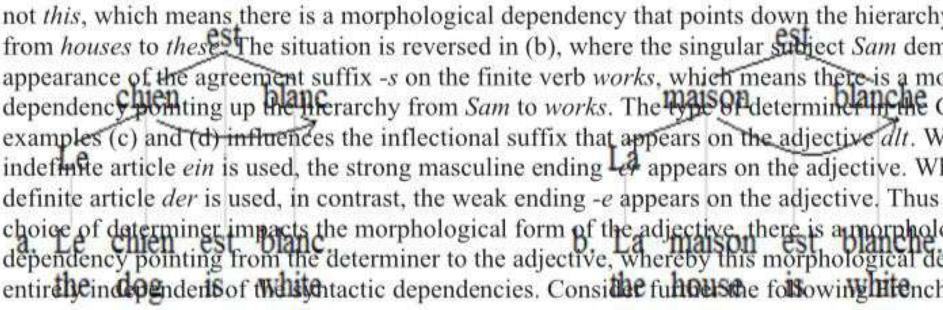
Morphological dependencies[edit]

Morphological dependencies obtain between words or parts of words.[15] When a given word or part of a word influences the form of another word, then the latter is morphologically dependent on the former. Agreement and concord are therefore manifestations of morphological dependencies. Like semantic dependencies, morphological dependencies can overlap with and point in the same direction as syntactic dependencies, overlap with and point in the opposite direction of syntactic dependencies, or be entirely independent of syntactic dependencies. The arrows are now used to indicate morphological dependencies.



Page 9

The plural *houses* in (a) demands the plural of the demonstrative determiner, hence *these* appears, not *this*, which means there is a morphological dependency that points down the hierarchy from *houses* to *these*. The situation is reversed in (b), where the singular subject *Sam* demands the appearance of the agreement suffix *-s* on the finite verb *works*, which means there is a morphological dependency pointing up the hierarchy from *Sam* to *works*. The type of determiner in the German examples (c) and (d) influences the inflectional suffix that appears on the adjective *alt*. When the indefinite article *ein* is used, the strong masculine ending *-er* appears on the adjective. When the definite article *der* is used, in contrast, the weak ending *-e* appears on the adjective. Thus since the choice of determiner impacts the morphological form of the adjective, there is a morphological dependency pointing from the determiner to the adjective, whereby this morphological dependency is entirely independent of the syntactic dependencies. Consider further the following French sentences:

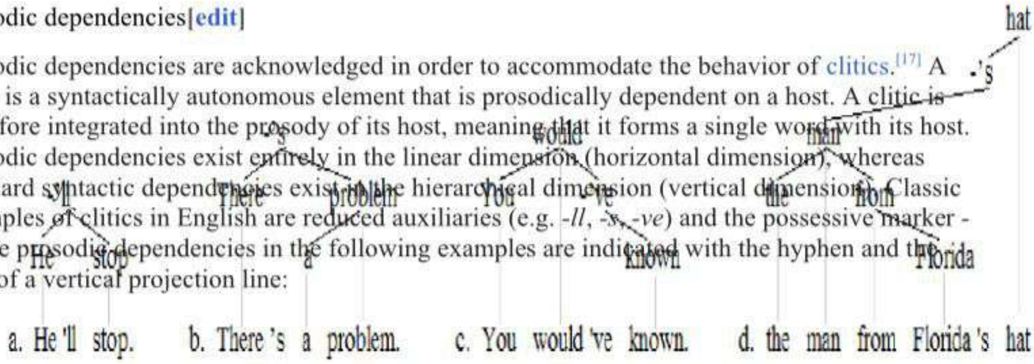


The masculine subject *le chien* in (a) demands the masculine form of the predicative adjective *blanc*, whereas the feminine subject *la maison* demands the feminine form of this adjective. A morphological dependency that is entirely independent of the syntactic dependencies therefore points again across the syntactic hierarchy.

Morphological dependencies play an important role in typological studies. Languages are classified as mostly head-marking (*Sam work-s*) or mostly dependent-marking (*these houses*), whereby most if not all languages contain at least some minor measure of both head and dependent marking.[16]

Prosodic dependencies[edit]

Prosodic dependencies are acknowledged in order to accommodate the behavior of clitics.[17] A clitic is a syntactically autonomous element that is prosodically dependent on a host. A clitic is therefore integrated into the prosody of its host, meaning that it forms a single word with its host. Prosodic dependencies exist entirely in the linear dimension (horizontal dimension), whereas standard syntactic dependencies exist in the hierarchical dimension (vertical dimension). Classic examples of clitics in English are reduced auxiliaries (e.g. *-ll*, *-s*, *-ve*) and the possessive marker *-s*. The prosodic dependencies in the following examples are indicated with the hyphen and the lack of a vertical projection line:

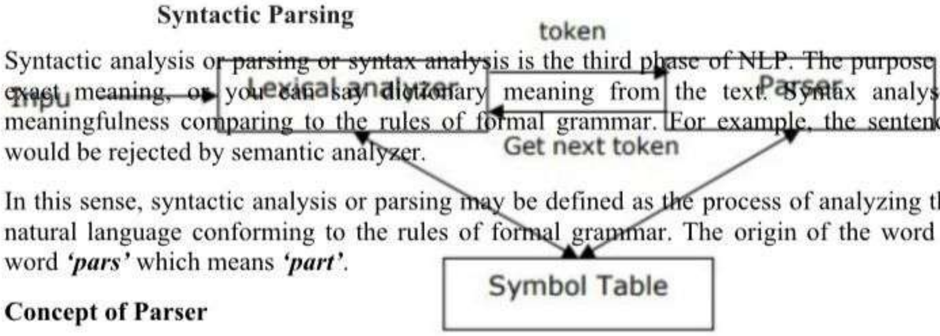


The hyphens and lack of projection lines indicate prosodic dependencies. A hyphen that appears on the left of the clitic indicates that the clitic is prosodically dependent on the word immediately to its left (*He'll*, *There's*), whereas a hyphen that appears on the right side of the clitic (not shown here) indicates that the clitic is prosodically dependent on the word that appears immediately to its right. A given clitic is often prosodically dependent on its syntactic dependent (*He'll*, *There's*) or on its head (*would've*). At other times, it can depend prosodically on a word that is neither its head nor its immediate dependent (*Florida's*).

Syntactic dependencies[edit]

Syntactic dependencies are the focus of most work in DG, as stated above. How the presence and the direction of syntactic dependencies are determined is of course often open to debate. In this regard, it must be acknowledged that the validity of syntactic dependencies in the trees throughout this article is being taken for granted. However, these hierarchies are such that many DGs can largely support them, although there will certainly be points of disagreement. The basic question about how syntactic dependencies are discerned has proven difficult to answer definitively. One should acknowledge in this area, however, that the basic task of identifying and discerning the presence and direction of the syntactic dependencies of DGs is no easier or harder than determining the constituent groupings of phrase structure grammars. A variety of heuristics are employed to this end, basic tests for constituents being useful tools; the syntactic dependencies assumed in the trees in this article are grouping words together in a manner that most closely matches the results of standard permutation, substitution, and ellipsis tests for constituents. Etymological considerations also provide helpful clues about the direction of dependencies. A promising principle upon which to base the existence of syntactic dependencies is distribution.[18] When one is striving to identify the root of a given phrase, the word that is most responsible for determining the distribution of that phrase as a whole is its root.

Syntactic Parsing



Syntactic analysis or parsing or syntax analysis is the third phase of NLP. The purpose of this phase is to draw exact meaning, or you can say dictionary meaning from the text. Syntax analysis checks the text for meaningfulness comparing to the rules of formal grammar. For example, the sentence like "hot ice-cream" would be rejected by semantic analyzer.

In this sense, syntactic analysis or parsing may be defined as the process of analyzing the strings of symbols in natural language conforming to the rules of formal grammar. The origin of the word ***'parsing'*** is from Latin word ***'pars'*** which means ***'part'***.

Concept of Parser

It is used to implement the task of parsing. It may be defined as the software component designed for taking input data (text) and giving structural representation of the input after checking for correct syntax as per formal grammar. It also builds a data structure generally in the form of parse tree or abstract syntax tree or other hierarchical structure.

Page 11

The main roles of the parse include −

- To report any syntax error.
- To recover from commonly occurring error so that the processing of the remainder of program can be continued.
- To create parse tree.
- To create symbol table.
- To produce intermediate representations (IR).

Types of Parsing

Derivation divides parsing into the followings two types −

- Top-down Parsing
- Bottom-up Parsing

Top-down Parsing

In this kind of parsing, the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input. The most common form of topdown parsing uses recursive procedure to process the input. The main disadvantage of recursive descent parsing is backtracking.

Bottom-up Parsing

In this kind of parsing, the parser starts with the input symbol and tries to construct the parser tree up to the start symbol.

Concept of Derivation

In order to get the input string, we need a sequence of production rules. Derivation is a set of production rules. During parsing, we need to decide the non-terminal, which is to be replaced along with deciding the production rule with the help of which the non-terminal will be replaced.

Types of Derivation

In this section, we will learn about the two types of derivations, which can be used to decide which non-terminal to be replaced with production rule −

Left-most Derivation

In the left-most derivation, the sentential form of an input is scanned and replaced from the left to the right. The sentential form in this case is called the left-sentential form.

Right-most Derivation

In the left-most derivation, the sentential form of an input is scanned and replaced from right to left. The sentential form in this case is called the right-sentential form.

Concept of Parse Tree

It may be defined as the graphical depiction of a derivation. The start symbol of derivation serves as the root of the parse tree. In every parse tree, the leaf nodes are terminals and interior nodes are non-terminals. A property of parse tree is that in-order traversal will produce the original input string.

Page 12

Concept of Grammar

Grammar is very essential and important to describe the syntactic structure of well-formed programs. In the literary sense, they denote syntactical rules for conversation in natural languages. Linguistics have attempted to define grammars since the inception of natural languages like English, Hindi, etc.

The theory of formal languages is also applicable in the fields of Computer Science mainly in programming languages and data structure. For example, in 'C' language, the precise grammar rules state how functions are made from lists and statements.

A mathematical model of grammar was given by **Noam Chomsky** in 1956, which is effective for writing computer languages.

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

- **N** or $\mathbf{V_N}$ = set of non-terminal symbols, i.e., variables.
- **T** or $\sum$ = set of terminal symbols.
- **S** = Start symbol where $S \in N$
- **P** denotes the Production rules for Terminals as well as Non-terminals. It 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

Phrase Structure or Constituency Grammar

Phrase structure grammar, introduced by Noam Chomsky, is based on the constituency relation. That is why it is also called constituency grammar. It is opposite to dependency grammar.

Example

Before giving an example of constituency grammar, we need to know the fundamental points about constituency grammar and constituency relation.

- All the related frameworks view the sentence structure in terms of constituency relation.
- The constituency relation is derived from the subject-predicate division of Latin as well as Greek grammar.
- The basic clause structure is understood in terms of **noun phrase NP** and **verb phrase VP**.

We can write the sentence **"This tree is illustrating the constituency relation"** as follows −

✅ **What Makes a "Better" Answer in Exams?**

Basic Answer (Average Marks)	Better Answer (High Marks)
Covers just the points	Explains + gives examples
Less structured	Well organized (headings, flow)
No diagram or weak diagram	Includes neat diagram (parse tree)
Generic info	Specific to topic, with technical terms explained simply

---

Syntactic Analysis in NLP

(10-Mark Answer)

1. Introduction

Natural Language Processing (NLP) is a technology that allows machines to understand human languages like English. One of the key steps in NLP is **Syntactic Analysis**, which deals with the **structure** of sentences.

---

2. What is Syntactic Analysis?

Syntactic analysis (also called **Parsing**) is the process of analyzing a sentence to check if it follows the correct grammatical structure. It helps in understanding how words are arranged and related.

---

3. Context-Free Grammar (CFG)

CFG is a type of grammar used to define the structure of sentences. It uses rules to break a sentence into smaller parts (phrases and words).

Example CFG Rules:

- S → NP VP
- NP → Det N
- VP → IV
- Det → the
- N → giraffe
- IV → dreams

These rules can be used to form sentences like: **"The giraffe dreams"**

---

4. Syntactic Categories (Symbols Used)

Symbol	Meaning
S	Sentence
NP	Noun Phrase
VP	Verb Phrase
Det	Determiner (e.g., the)
N	Noun (e.g., giraffe)
IV	Intransitive Verb

---

5. Parse Tree Diagram

A parse tree shows how a sentence is formed using CFG rules.

📌 **Sentence Example:** *The giraffe dreams*

CFG Rules Applied:

- S → NP VP
- NP → Det N
- VP → IV

Parse Tree:

```
      S
    /  \
   NP    VP
  / \     |
Det   N    IV
|    |     |
the giraffe dreams
```

This shows the sentence structure clearly and helps machines understand grammar.

---

6. Types of Ambiguity in NLP

1. **Lexical Ambiguity** – Same word, different meanings (e.g., *flies*)
2. **Syntactic Ambiguity** – Sentence has multiple structures (e.g., *She saw the man with the telescope*)
3. **Semantic Ambiguity** – Sentence has multiple meanings (e.g., *They are flying planes*)

7. Applications of Syntactic Analysis

- Grammar and spell checking
- Machine translation (e.g., Google Translate)
- Chatbots and virtual assistants
- Information retrieval and summarization
- Sentence generation (e.g., AI writing tools)

---

8. Treebank and Corpus

- **Treebank**: A collection of sentences with their parse trees.
- **Corpus**: A large database of real-world text used to train NLP models.

---

9. Conclusion

Syntactic analysis is a key step in NLP that checks sentence structure using rules like CFG. Parse trees help visualize how a sentence is built, making it easier for machines to understand language. It plays a major role in applications like translation, chatbots, grammar tools, and more.

☆ **Tips for Better Marks:**

- **Write clearly and to the point**
- **Draw the parse tree neatly**
- **Use real-world examples**
- **Underline key terms (if allowed)**

☑ **How to Score *Even Better*?**

To **maximize marks**, you can:

1. **Add a neat diagram** (like a tree structure if the question asks about Treebanks)
2. **Underline keywords** like “Corpus”, “Treebank”, “Semantic Treebank”, etc. (if allowed)
3. **Use bullet points** instead of long paragraphs

4. **Write a short introduction and a conclusion**

✨ **Updated High-Scoring Exam-Ready Answer (Final Version)**

Here's a **refined final version** — perfectly tuned for a 10-mark question titled:

Explain Corpus Design and Treebank Corpus in NLP.

(10 Marks)

---

1. What is a Corpus?

A **corpus** is a large and structured set of texts used for linguistic analysis in NLP. It helps machines understand and process human language.

---

2. Elements of Good Corpus Design

📌 **a. Representativeness**

- It means the corpus should represent the full range of a language.
- **Leech (1991):** Corpus should reflect the language variety it stands for.
- **Biber (1993):** Sample must include full range of variation.

✅ Determined by:

- **Balance:** Variety of genres (news, fiction, academic, etc.)
- **Sampling:** How samples are chosen.

📌 **b. Balance**

- Includes a wide range of text types (genres).
- A balanced corpus gives more accurate and useful results.

📌 **c. Sampling**

Important terms:

- **Sampling Unit:** The source of text (e.g., book, newspaper).
- **Sampling Frame:** List of all sampling units.
- **Population:** The total language data being studied.

📌 **d. Corpus Size**

Size depends on:

- Type of query or research goal
- Data availability
- Technology used

🗂 **Corpus Size Comparison Table:**

Year	Corpus Name	Size
1960s–70s	Brown and LOB	1 Million words
1980s	Birmingham Corpora	20 Million words
1990s	British National Corpus	100 Million words
2000s+	Bank of English Corpus	650 Million words

---

3. Treebank Corpus

A **Treebank** is a special type of corpus where each sentence is annotated with **syntactic** or **semantic structure** using a **tree format**.

🔷 Coined by Geoffrey Leech
🔷 Used for analyzing grammar structure (syntax trees)

---

4. Types of Treebank

✅ **a. Semantic Treebank**

- Focuses on **meaning** of sentences
- Examples: Geoquery, Robot Commands Treebank

✅ **b. Syntactic Treebank**

- Focuses on **sentence structure**
- Examples:
  - **Penn Arabic Treebank** (Arabic)
  - **Columbia Arabic Treebank**
  - **Sinica** (Chinese)
  - **BLLIP WSJ** (English)

---

5. Applications of Treebanks

💡 **In Computational Linguistics** – Used in POS taggers, parsers, machine translation
💡 **In Corpus Linguistics** – To study grammar patterns
💡 **In Theoretical Linguistics** – For language research and psycholinguistics

---

6. PropBank (Proposition Bank)

- A verb-oriented corpus with **proposition structure**
- Annotated with **verb arguments** (who did what)
- Helps with **semantic role labeling** in NLP

Conclusion

Corpus and Treebank design are key in NLP. A well-structured corpus helps in language analysis, while Treebanks make sentence structure clear to machines. Together, they improve applications like translation, chatbots, and grammar tools.

🎯 **Final Verdict:**

✅ This answer includes everything a **top scorer** would write:

- Definitions
- Research references
- Tables
- Applications
- Examples
- Conclusion

You can write this confidently in the exam and **expect full or near-full marks**.

✍ **Explain VerbNet, WordNet, and Chomsky Normal Form (CNF) in NLP.**

(10 Marks)

---

1. VerbNet (VN)

VerbNet is the **largest verb lexicon** in English that provides both **semantic** and **syntactic information** about verbs. It is domain-independent and widely used in NLP applications.

◈ **Organized by**: Extending Levin verb classes
◈ **Mapped to**: WordNet, FrameNet, XTAG (helps in cross-resource usage)

☑ **Each VerbNet class includes:**

- **Syntactic Frames**: Show different sentence patterns like transitive, intransitive, resultatives, etc.
- **Semantic Roles**: Indicate roles like *animate, human, organization*, etc.

◈ These features help machines **understand the meaning** and **structure** of sentences.

---

2. WordNet

WordNet is a **lexical database for English**, developed at **Princeton University**. It's part of the **NLTK** corpus and is widely used in NLP.

☑ **Key Features:**

- Words are grouped into **Synsets** (sets of synonyms).
- Synsets are linked by **semantic** and **lexical relations**.
- Very useful for tasks like:
  - **Word sense disambiguation**
  - **Information retrieval**
  - **Machine translation**
  - **Text classification**

◈ Supported in tools/languages like **NLTK (Python)**, **Perl**, and **Java (ADW)**

---

3. Chomsky Normal Form (CNF)

Chomsky Normal Form is a standard form of writing **Context-Free Grammar (CFG)**. It simplifies grammar so that **parsing algorithms** like CYK can work efficiently.

☑ **A CFG is in CNF if:**

1. **Start symbol → ε** (nullable)

2. **A → BC** (two non-terminals)
3. **A → a** (terminal)

📌 **Examples:**

Let's take two grammars:

- **G1 = {S → AB, S → c, A → a, B → b}** → ✅ CNF
- **G2 = {S → aA, A → a, B → c}** → ❌ Not CNF (S → aA has terminal + non-terminal)

Steps to Convert CFG to CNF:

Step 1: Eliminate start symbol from the right-hand side (RHS)

→ Create new symbol: S1 → S

Step 2: Remove null (ε), unit, and useless productions

→ Use simplification techniques

Step 3: Replace terminals on RHS if mixed with non-terminals

→ S → aA becomes S → XA, where X → a

Step 4: Ensure RHS has only 2 non-terminals or 1 terminal

→ S → ASB becomes S → AR, R → SB

✅ **Example: Convert CFG to CNF**

Given CFG:
S → a | aA | B
A → aBB | ε
B → Aa | b

Step 1: Add new start symbol:
S1 → S

Step 2: Remove ε:
A → aBB
B → Aa | b | a
S → a | aA | Aa | b

Step 3: Replace terminals with non-terminals (let X → a, Y → b):
S → X | XA | AX | Y
A → XBB
B → AX | Y | X

Conclusion

- **VerbNet** and **WordNet** are important for providing structured lexical and semantic information for NLP systems.
- **Chomsky Normal Form (CNF)** helps simplify grammar rules for parsing.
  These concepts are widely used in machine translation, syntactic parsing, and other NLP applications.

10-Marks Answer: Treebanks, VerbNet, WordNet, CNF, and Dependency Grammar

1. VerbNet (VN)

- **VerbNet** is a large, structured resource of English verbs.
- It gives both **semantic (meaning)** and **syntactic (sentence structure)** information about verbs.
- It groups verbs into classes that share similar meanings and sentence patterns.
- Each class includes:
  - **Syntactic frames** (like transitive/intransitive constructions),
  - **Semantic roles** (like animate, human, organization) to define how words behave in a sentence.

---

2. WordNet

- **WordNet** is a **lexical database** developed by **Princeton University**.
- It groups **nouns, verbs, adjectives, and adverbs** into sets of synonyms called **Synsets**.
- These Synsets are linked through **semantic and lexical relations**.
- Used in **Natural Language Processing (NLP)** for:
  - **Word sense disambiguation**,
  - **Text classification**, and
  - **Machine translation**.

---

3. CNF - Chomsky Normal Form

- **CNF** is a special format for writing **Context-Free Grammar (CFG)**.

- A grammar is in CNF if each production rule is of the form:
    - A → BC (non-terminals)
    - A → a (terminal)
    - A → ε (empty string)
- The purpose is to simplify grammar for parsing algorithms.

Steps to Convert CFG to CNF:

1. **Eliminate the start symbol** from RHS if it appears.
2. **Remove null, unit, and useless productions.**
3. **Replace terminals** in long RHS with non-terminals.
4. **Reduce RHS** to two symbols using new variables.

Example Given:

CFG is converted step-by-step to CNF:

- Terminal symbols (like "a") replaced with non-terminal (like X → a).
- Longer rules like A → XBB are broken down into two-symbol rules using new non-terminals.

---

4. Dependency Grammar (DG)

- DG focuses on the **relationships between words** in a sentence.
- Every word is **connected to a verb** through **directed links** called **dependencies**.
- These links create a **tree structure** where:
    - The **verb** is the center (head).
    - Other words are **dependents**.

Differences from Phrase Structure Grammar:

- **DG** uses **one-to-one** relation (each word has one link).
- **Phrase Structure Grammar** allows **one-to-many** (a phrase can have many nodes).

---

5. Types of Dependencies:

A. Semantic Dependencies

- Based on the **meaning** of words.
- Example: In “Sam likes Sally”, both “Sam” and “Sally” depend on the verb “likes”.

B. Morphological Dependencies

- Based on the **form of words**.

- For example, "works" depends on "Sam" (subject-verb agreement).
- Examples in **German** like "der alte Mann" (the old man) also show how endings and word forms depend on each other.

---

6. Visual Comparison:

- **Two trees** show how sentence structure can be represented:
  - One using **Dependency Grammar**
  - One using **Constituency Grammar**
- **Dependency** tree is simpler and shows how each word is connected to the verb directly.
- **Constituency** tree breaks the sentence into **phrases** and **sub-phrases**.

Conclusion

This content explains **important linguistic tools and theories** used in grammar and NLP:

- **VerbNet** and **WordNet** help organize words.
- **CNF** simplifies grammar rules.
- **Dependency Grammar** shows how words relate to the verb.
- **Understanding dependencies** (semantic, morphological) helps in computational and theoretical linguistics.

✅ **Tips for Writing in Exam:**

- Use **clear headings**.
- Write **short definitions** and **simple explanations**.
- Support answers with **examples**.
- If there's a diagram (like dependency trees), **draw it neatly** for extra marks.

📓 **Chomsky Normal Form (CNF) Conversion and Dependency Grammar**

✅ **Part 1: Chomsky Normal Form (CNF)**

Chomsky Normal Form (CNF) is a special way to write a **Context-Free Grammar (CFG)** where each production rule must follow one of the following forms:

1. **A → BC** (where A, B, C are non-terminals)
2. **A → a** (A is a non-terminal and a is a terminal)
3. **A → ε** (only allowed for start symbol if needed)

💡 Steps to Convert CFG to CNF:

Step 1: Add a new start symbol
If the start symbol is used on the right-hand side, add a new symbol like S_0 → S.

Step 2: Remove null (ε) productions
Any production like A → ε should be removed carefully and replaced.

Step 3: Remove unit productions
For example, if you have S → A and A → a, replace S → A with S → a.

Step 4: Convert terminals in mixed rules
If a rule has terminals with non-terminals (e.g., A → aB), replace a with a new symbol like X → a and change A → aB to A → XB.

Step 5: Break RHS with more than 2 symbols
If a production has more than two non-terminals like A → XBB, break it into A → XR and R → BB.

✔ Final CNF Example:

Given Grammar:
S → a | aA | Aa | b
A → aBB
B → Aa | b | a
X → a (new symbol)
R → XB

So the final CNF looks like:

S_0 → a | XA | AX | b

A → RB

B → AX | b | a

X → a

R → XB

This is now in CNF ☑

☑ Part 2: Dependency Grammar (DG)

Dependency Grammar is a modern grammar theory where **words depend on each other**. The main idea is that **a verb (action word)** is the center of the sentence, and all other words are **connected to it**.

- The **verb** is called the **head**, and the connected words are **dependents**.
- Every word is connected directly or indirectly to the verb.

📊 **Comparison: Dependency vs Phrase Structure**

Feature	Dependency Grammar	Phrase Structure Grammar
Structure	One-to-one	One-to-one or more
Center	Verb (Head)	Phrases (like NP, VP)
Simplicity	More flat structure	More complex structure
Example	Minimal nodes	More nodes

Example Tree Comparison:

Sentence: *We are trying to understand the difference.*

- In **Dependency Grammar**, each word connects directly to a head (e.g., "understand" connects to "trying", "trying" to "are", etc.).
- In **Phrase Structure Grammar**, phrases like NP (noun phrase), VP (verb phrase) are built.

🔍 **Types of Dependencies:**

1. Syntactic Dependencies

Show how words are grammatically connected.
E.g., In "Sam likes Sally", both "Sam" and "Sally" are linked to "likes".

2. Semantic Dependencies

Focus on **meaning** and what words depend on each other based on **understanding**.
E.g., "big bones" – "big" depends on "bones" semantically.

3. Morphological Dependencies

Show how **word forms** depend on each other.
E.g., "Sam works" → "work" changes to "works" because of "Sam".

🖼 **Diagram Examples:**

- **Semantic Dependency Tree**:
  - Sam → likes ← Sally
- **Morphological Dependency**:
  - these → houses
  - Sam → works

Also includes German examples like:

- ein → alter → Mann (an old man)

✅ **Conclusion**

- CNF helps to convert any CFG into a standard format for parsing.
- Dependency Grammar is useful to understand sentence structure based on word relationships.
- Diagrams help in better understanding of grammar structures.

✅ **Tip for Exams**:

- Draw **dependency trees** clearly.
- Mention **steps** in CNF conversion with **examples**.
- Add both **theory and diagrams** for better marks.

Explanation of Morphological and Syntactic Dependencies, Prosodic Dependencies, and Syntactic Parsing

1. Morphological Dependencies:

- Morphological dependencies happen when the form of one word depends on another word in the sentence.
- For example, in English, the plural form of a noun affects the form of the demonstrative word:
    - "houses" (plural) requires "these" (not "this").
- In French and German, the endings of adjectives change depending on the noun and the article used (like "le chien est blanc" - the dog is white).
- These dependencies show that some word forms depend on other words, even when they are not right next to each other.

2. Prosodic Dependencies:

- Prosodic dependencies are related to how words sound and are spoken, especially with clitics (small words like "'s" or "n't").

- For example, in English, words like "He'll" or "There's" show prosodic dependency where the clitic ('ll or 's) depends on the word before it.
- Hyphens in sentences help show where clitics attach to words and how they are connected prosodically.
- Examples:
    - a) He'll stop.
    - b) There's a problem.
    - c) You would've known.
    - d) The man from Florida's hat.

3. Syntactic Dependencies:

- Syntactic dependencies explain the grammatical relationship between words in a sentence.
- These dependencies show how words connect and form phrases or sentences based on grammar rules.
- They help us understand the sentence structure and which word depends on which.

4. Syntactic Parsing:

- Parsing is the process of analyzing a sentence to understand its structure.
- The goal of parsing is to find the correct meaning and grammatical structure of the sentence.
- For example, the parser checks if the sentence follows the grammar rules, like rejecting sentences that don't make sense (e.g., "hot ice-cream").
- The word "parsing" comes from Latin meaning "part" because it breaks down the sentence into parts.

5. Concept of Parser:

- A parser is a software tool that processes sentences to check if they follow grammar rules.
- It builds a structure (called a parse tree or syntax tree) to show how words in a sentence are connected.
- This helps computers understand human language better.

Parsing and Grammar in Compiler Design

1. Roles of Parsing:

- Parsing checks the program for any syntax errors.
- It helps fix common errors so the program can continue running.

- It builds a parse tree, which shows the structure of the code.
- It creates a symbol table to store information about variables.
- It generates intermediate representations (IR) used later in the compilation process.

2. Types of Parsing:
Parsing is of two types:

- **Top-down Parsing:**
  Starts from the start symbol and tries to transform it into the input string by breaking it down. It uses recursion but can face problems like backtracking.
- **Bottom-up Parsing:**
  Starts from the input string and tries to build up to the start symbol by combining smaller parts into bigger parts.

3. Concept of Derivation:
Derivation means producing the input string using a set of production rules. It tells us how to replace non-terminal symbols (like variables) step-by-step to reach the final string.

4. Types of Derivation:

- **Left-most Derivation:**
  The leftmost non-terminal is always replaced first in the input string.
- **Right-most Derivation:**
  The rightmost non-terminal is replaced first.

5. Concept of Parse Tree:

- A parse tree visually represents the derivation of a string.
- The root of the tree is the start symbol.
- Leaf nodes are terminal symbols (actual code parts).
- Internal nodes are non-terminals (variables).
- If you read the tree in order (left to right), it gives the original input string.

Grammar in Compiler Design

1. Concept of Grammar:

- Grammar defines rules to describe the structure of programs.
- It is important for defining syntax in programming languages like C.
- Linguists use grammar to describe natural languages too.

2. Formal Grammar Model (4-tuple):

- Grammar is defined as (N, T, S, P), where:
  - **N:** Non-terminal symbols (variables).

- **T:** Terminal symbols (actual characters or tokens).
- **S:** Start symbol.
- **P:** Production rules that show how non-terminals can be replaced.

3. Phrase Structure (Constituency) Grammar:

- Introduced by Noam Chomsky.
- It shows how sentences (or code statements) are made of smaller parts.
- It focuses on the relationship between parts of a sentence (constituency).
- It's the opposite of dependency grammar.

4. Example (Constituency Relation):

- Constituency explains how sentence structure is divided into parts like noun phrase (NP) and verb phrase (VP).
- For example, in the sentence "This tree is illustrating the constituency relation,"
  - "This tree" is the noun phrase (NP),
  - "is illustrating the constituency relation" is the verb phrase (VP).
