NATURAL LANGUAGE PROCESSING

UNIT I

Introduction to Natural Language Processing

References

Primary Textbook

- Daniel Jurafsky & James H. Martin, *Speech and Language Processing*
- Tanveer Siddiqui & U. S. Tiwary, *Natural Language Processing and Information Retrieval*

Topics Covered

- Introduction to Natural Language Processing
- Origins of NLP
- Challenges in NLP
- Language and Grammar
- Regular Expressions
- Finite-State Automata
- Tokenization
- Morphological Parsing
- Spelling Error Detection
- Minimum Edit Distance
- Statistical Language Models
- Processing Indian Languages

Unit Introduction

Natural Language Processing (NLP) is one of the most important fields of Artificial Intelligence (AI). It enables computers to understand, interpret, process, and generate human language. Humans naturally communicate through speech and writing, whereas computers understand binary data and programming instructions. NLP bridges this communication gap by combining knowledge from Computer Science, Artificial Intelligence, Linguistics, Mathematics, and Machine Learning.

Today, NLP is used in almost every digital platform. Search engines, virtual assistants, chatbots, machine translation systems, grammar checkers, and recommendation systems all

depend on NLP techniques. Understanding the concepts covered in this unit provides the foundation for advanced topics such as parsing, information retrieval, machine translation, and speech processing that appear later in the syllabus.

◆Introduction to Natural Language Processing (NLP)

Definition

Definition

Natural Language Processing (NLP) is the branch of Artificial Intelligence that enables computers to understand, interpret, analyze, and generate human language.

Simple Definition

Natural Language Processing is the technology that helps computers communicate with humans using natural languages such as English, Telugu, Hindi, Tamil, and many others.

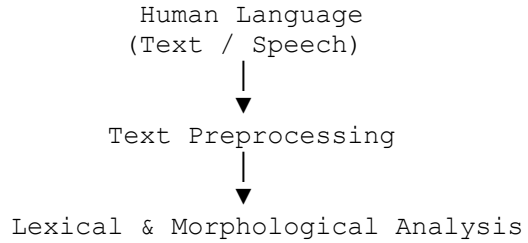
Why is NLP Needed?

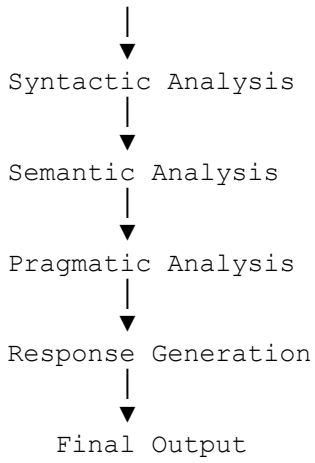
Computers process information in machine language, while humans communicate through natural languages. Human language contains grammar, context, emotions, ambiguity, and variations that cannot be understood directly by computers.

NLP converts human language into a form that computers can process. It allows machines to identify words, analyze sentence structure, determine meaning, and generate appropriate responses.

Without NLP, interacting with computers would require programming languages or predefined commands, making communication difficult for ordinary users.

Conceptual NLP Pipeline





Applications of NLP

Application	Purpose	Example
Machine Translation	Converts one language into another	Google Translate
Chatbots	Provides automated conversation	ChatGPT
Speech Recognition	Converts speech into text	Siri, Google Voice Typing
Search Engines	Retrieves relevant information	Google Search
Grammar Checking	Detects grammar and spelling mistakes	Grammarly
Sentiment Analysis	Identifies opinions in text	Product Review Analysis
Text Summarization	Produces concise summaries	AI Summarizers
Question Answering	Answers user queries	Virtual Assistants

Example

Input:

"Translate 'Good Morning' into Telugu."

The NLP system identifies that the user wants **language translation**, recognizes the source language (English), determines the target language (Telugu), and generates the translated sentence.

Important

NLP is not limited to translation. It includes understanding, analyzing, generating, and processing human language.

Advantages

- Enables natural communication between humans and computers.
- Automates text and speech processing.
- Supports multilingual communication.
- Reduces manual effort.
- Improves customer support through chatbots.
- Processes large volumes of textual data efficiently.

Limitations

- Human language is ambiguous.
- Context is difficult for computers to understand.
- Idioms and sarcasm remain challenging.
- Performance depends on the quality of training data.
- Many regional languages have limited resources.

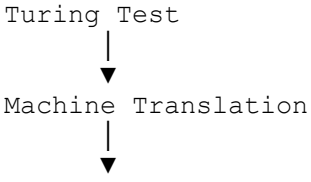
◆Origins of Natural Language Processing

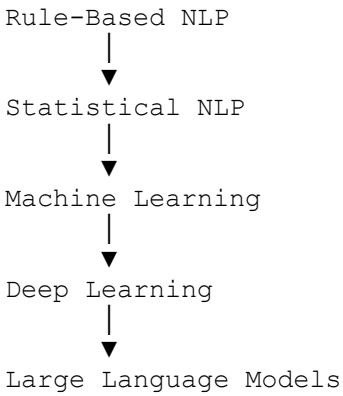
Natural Language Processing developed gradually through the combined efforts of researchers in Artificial Intelligence, Computer Science, and Linguistics. The objective was to enable computers to understand and communicate using human language. Over several decades, NLP evolved from manually written grammar rules to data-driven statistical models and, more recently, deep learning techniques.

Evolution of NLP

Period	Major Development
1950	Turing Test proposed
1950s	Machine Translation research
1960s–1980s	Rule-Based NLP
1990s	Statistical NLP
2000s	Machine Learning approaches
2010–Present	Deep Learning and Large Language Models

Conceptual Timeline





Rule-Based NLP

The earliest NLP systems depended on manually created grammar rules and dictionaries. Linguists defined grammatical rules, and computers followed these rules while processing language. These systems performed well in restricted domains but struggled with ambiguous or unseen language.

Characteristics

- Grammar-based
- Dictionary-driven
- Predictable output
- High human effort
- Limited scalability

Statistical NLP

During the 1990s, researchers began using large text collections (corpora) to learn language patterns automatically. Statistical NLP predicts the most likely interpretation of language using probability rather than fixed grammar rules.

Common techniques include:

- N-Gram Models
- Hidden Markov Models (HMM)
- Probabilistic Parsing

Statistical methods significantly improved the performance of machine translation, speech recognition, and part-of-speech tagging.

Deep Learning-Based NLP

Modern NLP uses neural networks that automatically learn language patterns from large datasets. These models can understand context better than earlier approaches and have enabled major advances in translation, question answering, summarization, and conversational AI.

Large Language Models (LLMs), such as ChatGPT, are examples of deep learning-based NLP systems.

Comparison of NLP Approaches

Feature	Rule-Based	Statistical	Deep Learning
Knowledge Source	Grammar Rules	Data	Neural Networks
Human Effort	High	Moderate	Low
Scalability	Limited	Good	Excellent
Accuracy	Moderate	High	Very High

◆Challenges in Natural Language Processing

Although humans can understand language naturally, it is a difficult task for computers. Human language is rich, flexible, and constantly evolving. The same word may have different meanings in different contexts, and different words may express the same idea. These characteristics make Natural Language Processing one of the most challenging areas of Artificial Intelligence.

The major challenges faced by NLP systems are discussed below.

---

1. Lexical Ambiguity

A single word can have more than one meaning. The computer must determine the correct meaning based on the context.

Example

The bat is flying.

Here, **bat** refers to an animal.

He bought a new bat.

Here, **bat** refers to sports equipment.

This problem is called **Lexical Ambiguity**.

---

2. Syntactic Ambiguity

Sometimes a sentence can have multiple grammatical interpretations.

Example

I saw the man with a telescope.

Possible interpretations:

- I used a telescope to see the man.
- The man was carrying a telescope.

The computer must identify the correct sentence structure.

---

3. Semantic Ambiguity

Even if the grammar is correct, the meaning may still be unclear.

Example

The chicken is ready to eat.

Possible meanings:

- The chicken is ready to eat food.
- The chicken is ready to be eaten.

Understanding the intended meaning is difficult.

---

4. Pragmatic Ambiguity

Pragmatics deals with the intended meaning rather than the literal meaning.

Example

Can you open the window?

Literal meaning:

The speaker asks whether the listener is capable of opening the window.

Actual meaning:

The speaker is requesting the listener to open the window.

---

5. Context Understanding

Words change meaning depending on context.

Example

The bank is closed.

Possible meanings:

- Financial institution
- River bank

The surrounding words help determine the intended meaning.

---

6. Morphological Complexity

Words can have many forms.

Example

Root Word	Variations
Play	Plays, Playing, Played, Player
Write	Writes, Writing, Written, Writer

An NLP system must recognize that these words share the same root.

---

7. Language Diversity

Different languages have different:

- Grammar
- Sentence structure
- Scripts
- Vocabulary
- Writing systems

Developing one NLP system that works well for all languages is extremely difficult.

---

8. Speech Variability

In speech processing, pronunciation changes because of:

- Accent
- Speaking speed
- Background noise
- Regional dialects

Speech recognition systems must handle these variations.

Major Challenges at a Glance

Challenge	Description
Lexical Ambiguity	One word, multiple meanings
Syntactic Ambiguity	Multiple sentence structures
Semantic Ambiguity	Multiple interpretations
Pragmatic Ambiguity	Intended meaning differs from literal meaning
Context Understanding	Meaning depends on surrounding words
Morphological Complexity	Words appear in many forms
Language Diversity	Different grammar and scripts
Speech Variability	Accent and pronunciation differences

Remember

Ambiguity is one of the biggest challenges in NLP because computers cannot naturally infer meaning from context the way humans do.

◆Language and Grammar

Language is the primary medium through which humans communicate ideas, emotions, and information. For a computer to understand language, it must first understand the rules governing that language. These rules are collectively known as **grammar**.

In NLP, language is analyzed at different levels, beginning with words and extending to complete documents. Each level contributes to a better understanding of the meaning of a sentence.

Language

A **language** is a structured system of communication consisting of words, sounds, symbols, and grammatical rules that allow people to express ideas and exchange information.

Natural languages include:

- English
- Telugu
- Hindi
- Tamil
- French
- Japanese

Unlike programming languages, natural languages contain ambiguity, emotions, and variations in expression.

Grammar

Grammar is the set of rules that defines how words are arranged to form meaningful sentences.

Consider the following examples.

✓The student completed the assignment.

✗Assignment completed student the.

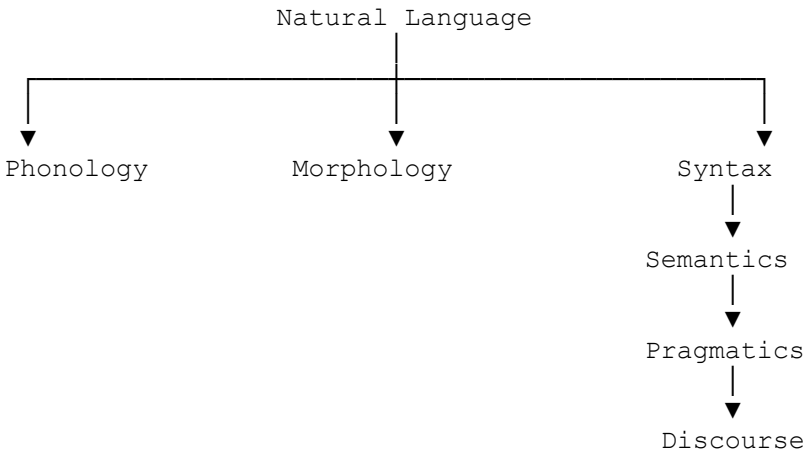
Although both sentences contain the same words, only the first follows grammatical rules.

Grammar helps NLP systems identify:

- Subject
- Verb
- Object
- Sentence structure

Levels of Language Analysis

Language is analyzed at multiple levels.



Phonology

Phonology studies speech sounds.

Example:

The pronunciation of

Cat

is different from

Cut

Although only one letter changes, the pronunciation and meaning change.

Morphology

Morphology studies the internal structure of words.

Example

Teacher

↓

Teach + er

Teaching

↓

Teach + ing

Morphology helps identify the root word and its grammatical form.

Syntax

Syntax studies sentence structure.

Example

✓ The boy plays cricket.

✗ Cricket plays boy the.

Syntax ensures the sentence follows grammatical rules.

Semantics

Semantics deals with meaning.

Sentence:

The dog chased the cat.

Semantics determines who performed the action and who received the action.

Pragmatics

Pragmatics considers context.

Sentence:

It's cold here.

Depending on the situation, the speaker may actually mean:

Please close the window.

Discourse

Discourse studies relationships between multiple sentences.

Example

Riya bought a laptop.

She uses it every day.

The system must understand:

- **She** → Riya
- **It** → Laptop

Relationship Between Language Levels

Level	Studies
Phonology	Speech sounds
Morphology	Word formation
Syntax	Sentence structure
Semantics	Meaning
Pragmatics	Context
Discourse	Relationship between sentences

Important

These six levels work together to help NLP systems understand human language accurately.

◆Regular Expressions

Natural language contains a large amount of textual data. Before a computer understands the meaning of a sentence, it often needs to search, identify, or extract specific patterns such as words, numbers, dates, email addresses, or phone numbers. **Regular Expressions (Regex)** provide a simple and efficient way to perform these tasks.

A Regular Expression is not an NLP algorithm by itself, but it is an important preprocessing tool widely used in text processing and lexical analysis.

Definition

Definition

A **Regular Expression (Regex)** is a sequence of characters that specifies a search pattern used for matching, searching, extracting, and replacing text.

Why are Regular Expressions Important?

Regular Expressions help NLP systems:

- Search specific words or phrases.
- Extract useful information.
- Validate user input.
- Clean textual data.
- Perform tokenization.
- Remove unwanted symbols.

For example, before analyzing a sentence, punctuation marks and extra spaces are often removed using Regular Expressions.

Common Regular Expression Symbols

Symbol	Meaning	Example
.	Matches any single character	c.t → cat, cut
*	Zero or more occurrences	go* → g, go, goo
+	One or more occurrences	go+ → go, goo

Symbol	Meaning	Example
?	Zero or one occurrence	colou?r → color, colour
[]	Character class	[aeiou]
[a-z]	Lowercase letters	a–z
[A-Z]	Uppercase letters	A–Z
[0-9]	Digits	0–9
^	Beginning of line	^Hello
$	End of line	world$

Conceptual Pattern Matching

```
Input Text

My email is student123@gmail.com

            |
            ▼

Regular Expression

[a-zA-Z0-9._%+-]+@[a-zA-Z.-]+\.[A-Za-z]{2,}

            |
            ▼

Matched Result

student123@gmail.com
```

Applications of Regular Expressions

- Email extraction
- Phone number validation
- Password validation
- URL detection
- Text preprocessing
- Tokenization
- Information extraction

Example

Sentence:

My phone number is **9876543210**.

Regex:

```
[0-9]{10}
```

Output:

```
9876543210
```

The expression matches exactly ten digits.

Remember

Regular Expressions identify **patterns**, not meanings. Understanding meaning requires additional NLP techniques such as parsing and semantic analysis.

◆Finite-State Automata (FSA)

Finite-State Automata (FSA) provide a mathematical model for recognizing patterns in text. They are closely related to Regular Expressions because every Regular Expression can be represented by an automaton.

FSAs are widely used in lexical analysis, spell checking, token recognition, and compiler design.

Definition

Definition

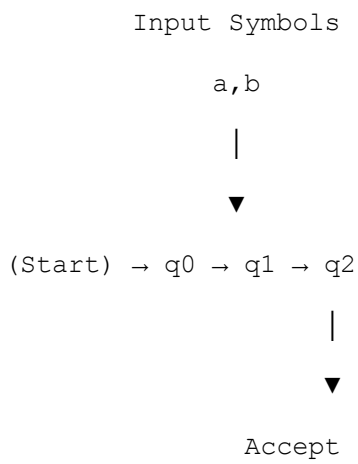
A **Finite-State Automaton (FSA)** is a mathematical model consisting of states and transitions that recognizes whether an input string belongs to a particular language.

Components of an FSA

An FSA consists of five basic components.

Component	Description
States	Different stages of computation
Alphabet	Set of input symbols
Transition Function	Rules for moving between states
Initial State	Starting point
Final State	Accepting state

Conceptual Structure



Types of Finite-State Automata

Deterministic Finite Automaton (DFA)

In a DFA, each state has only one transition for each input symbol.

Characteristics

- One transition per symbol.
- Easy to implement.
- Faster execution.

Non-Deterministic Finite Automaton (NFA)

In an NFA, a state may have multiple transitions for the same input symbol.

Characteristics

- Multiple possible paths.
- Easier to design.
- Equivalent in power to DFA.

DFA vs NFA

DFA	NFA
One transition per input	Multiple transitions allowed
Deterministic	Non-deterministic
Faster execution	Easier construction
More states	Fewer states

Applications of FSA

- Lexical Analysis
- Spell Checking
- Pattern Recognition
- Regular Expression Matching
- Token Recognition

Example

Suppose we want to recognize the word:

```
cat
```

The automaton moves through different states after reading each character.

```
(Start)

q0 --c--> q1 --a--> q2 --t--> q3

q3 = Accept
```

If the input exactly matches **cat**, the machine reaches the accepting state.

Important

Every Regular Expression can be converted into a Finite-State Automaton. This relationship makes FSAs an important tool in NLP and compiler design.

◆Tokenization

Tokenization is one of the first preprocessing steps in almost every NLP application. Instead of processing an entire document as one large block of text, NLP systems divide it into smaller meaningful units called **tokens**.

A token may be a word, sentence, punctuation mark, or even a character, depending on the application.

Definition

Definition

Tokenization is the process of dividing text into smaller units called **tokens**.

Why is Tokenization Needed?

Computers cannot directly understand a complete paragraph. Tokenization breaks the text into manageable pieces that can be processed individually.

For example,

Sentence:

Natural Language Processing is interesting.

After tokenization:

```
Natural | Language | Processing | is | interesting
```

Each word becomes a separate token.

Types of Tokenization

1. Word Tokenization

Splits a sentence into individual words.

Example

```
Machine Learning is powerful.
```

↓

```
Machine | Learning | is | powerful
```

---

2. Sentence Tokenization

Splits a paragraph into individual sentences.

Example

```
NLP is interesting.
It is a branch of AI.
Students enjoy learning it.
```

↓

Sentence 1

Sentence 2

Sentence 3

Tokenization Pipeline

```
Paragraph
      |
      ▼
Sentence Tokenization
      |
      ▼
Word Tokenization
      |
      ▼
Tokens Ready for NLP Processing
```

Applications

- Text preprocessing
- Machine Translation
- Search Engines
- Question Answering
- Chatbots
- Sentiment Analysis

Remember

Tokenization is usually the **first step** in most NLP pipelines because later stages such as stemming, parsing, and language modeling work on tokens rather than raw text.

◆Morphological Parsing

After tokenization, the next step in many NLP systems is **Morphological Parsing**. Human languages contain words in many different forms. For example, the words *play*, *plays*, *played*, and *playing* all originate from the same root word. Morphological Parsing helps an NLP system understand the internal structure of such words and identify their base form.

Definition

Definition

Morphological Parsing is the process of analyzing the internal structure of words by identifying the root word (stem) and its prefixes or suffixes.

Morpheme

The smallest meaningful unit of a language is called a **morpheme**.

Examples:

Word	Morphemes
Unhappy	Un + Happy
Teacher	Teach + er

Word	Morphemes
Rewriting	Re + Write + ing

Types of Morphemes

Root Morpheme

Carries the main meaning of the word.

Example:

```
Teaching
↓
Teach + ing
```

Here, **Teach** is the root morpheme.

Affixes

Affixes modify the meaning of a root word.

They are classified into:

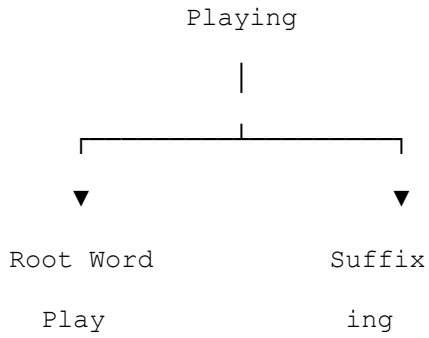
- Prefix
- Suffix

Example:

Prefix	Word
Un	Unhappy
Re	Rewrite

Suffix	Word
er	Teacher
ing	Playing

Morphological Analysis



Stemming

Stemming removes prefixes or suffixes to obtain the **stem** of a word.

Example

Word	Stem
Playing	Play
Connected	Connect
Studies	Studi

Notice that **Studi** is not a valid English word.

Lemmatization

Lemmatization converts a word into its **dictionary form (lemma)**.

Example

Word	Lemma
Playing	Play
Better	Good
Studies	Study

Unlike stemming, the output is always a meaningful dictionary word.

Stemming vs Lemmatization

Feature	Stemming	Lemmatization
Method	Rule-based	Dictionary + Grammar
Output	Stem	Dictionary Word
Accuracy	Moderate	High
Speed	Fast	Slower

Applications

- Search Engines
- Spell Checking
- Machine Translation
- Information Retrieval
- Text Classification

Remember

Every lemma is meaningful, but a stem may not always be a valid dictionary word.

◆Spelling Error Detection

Users often make spelling mistakes while typing. NLP systems detect these mistakes and suggest the correct spelling.

Definition

Definition

Spelling Error Detection is the process of identifying incorrectly spelled words and suggesting appropriate corrections.

Common Types of Errors

1. Insertion

An extra character is added.

Example

```
Bookk
```

Correct

```
Book
```

---

2. Deletion

A required character is missing.

Example

```
Recive
```

Correct

```
Receive
```

---

3. Substitution

One letter is replaced by another.

Example

```
Cot
```

instead of

```
Cat
```

---

4. Transposition

Two adjacent letters exchange places.

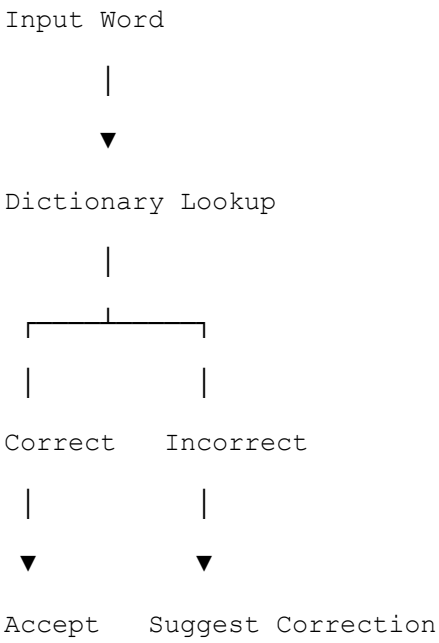
Example

```
Recieve
```

Correct

```
Receive
```

Error Detection Process



Applications

- Grammarly
- Microsoft Word
- Google Search
- Mobile Keyboards
- OCR Systems

◆Minimum Edit Distance (MED)

When a spelling mistake is detected, the computer must determine the closest correct word. **Minimum Edit Distance** provides a mathematical way to measure the similarity between two words.

Definition

Definition

Minimum Edit Distance (MED) is the minimum number of editing operations required to transform one word into another.

Allowed Operations

- Insertion
- Deletion
- Substitution

Example

Convert

```
CAT
```

↓

```
CUT
```

Only one substitution is required.

Therefore,

Minimum Edit Distance = 1

Another Example

Convert

```
BOOK
```

↓

```
BOOKS
```

Operation

Insertion of

```
S
```

MED = **1**

Conceptual Matrix

```
    C   A   T
```

```
      0   1   2   3
C     1
U     2
T     3
```

The complete matrix is filled using Dynamic Programming to calculate the minimum number of edits.

Applications

- Spell Checking
- DNA Sequence Matching
- OCR Error Correction
- Search Engines
- Information Retrieval

Important

Smaller edit distance indicates greater similarity between two words.

◆Statistical Language Models

Language Models estimate the probability of word sequences. They help computers predict which word is most likely to appear next.

Definition

Definition

A **Statistical Language Model** assigns probabilities to sequences of words.

Why are Language Models Needed?

They help NLP systems

- Predict the next word
- Correct spelling
- Translate languages
- Recognize speech
- Generate text

Types of Language Models

Unigram

Each word is treated independently.

Example

```
I

Love

NLP
```

Bigram

Probability depends on one previous word.

```
I Love

Love NLP
```

Trigram

Probability depends on two previous words.

```
I Love NLP
```

N-Gram Concept

```
Sentence

I Love Natural Language Processing

↓

Unigrams

I | Love | Natural | Language | Processing

↓
```

```
Bigrams

I Love

Love Natural

Natural Language

Language Processing

↓

Trigrams

I Love Natural

Love Natural Language

Natural Language Processing
```

Applications

- Auto-complete
- Speech Recognition
- Machine Translation
- Text Prediction
- Chatbots

◆Processing Indian Languages

Indian languages present unique challenges because they differ significantly from English in grammar, morphology, vocabulary, and writing systems.

Examples include:

- Telugu
- Hindi
- Tamil
- Kannada
- Malayalam
- Bengali

Challenges

- Multiple scripts

- Rich morphology
- Free word order
- Compound words
- Code-mixing
- Limited annotated datasets

Applications

- Machine Translation
- Voice Assistants
- OCR
- Search Engines
- Digital Libraries
- Educational Software

Indian Language NLP Pipeline

```
Indian Language Text
          |
          ▼
Unicode Processing
          |
          ▼
Tokenization
          |
          ▼
Morphological Analysis
          |
          ▼
Language Processing
          |
          ▼
Application
```

Unit Summary

In this unit, we studied the foundations of Natural Language Processing. We began with the introduction, need, and applications of NLP, followed by its historical evolution from rule-based systems to deep learning models. We examined the major challenges in processing human language, explored the role of language and grammar, and learned how Regular Expressions and Finite-State Automata are used in text processing. We also studied tokenization, morphological parsing, spelling error detection, Minimum Edit Distance, statistical language models, and the unique characteristics of processing Indian languages.

These concepts form the foundation for advanced topics such as parsing, information retrieval, machine translation, and speech processing covered in later units.

Important Definitions

Term	Definition
NLP	Processing and understanding human language using computers
Morpheme	Smallest meaningful unit of language
Tokenization	Splitting text into smaller units called tokens
Stemming	Reducing words to their stem
Lemmatization	Reducing words to their dictionary form
Regular Expression	Pattern used for text matching
Finite-State Automaton	Mathematical model used for pattern recognition
Minimum Edit Distance	Minimum edits required to transform one word into another
Language Model	Model that predicts the probability of word sequences

---

UNIT II

WORD-LEVEL AND SYNTACTIC ANALYSIS

Prescribed Textbook

Daniel Jurafsky & James H. Martin

Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Pearson Education, 2023.

Topics Covered

- Part-of-Speech (POS) Tagging
- Rule-Based POS Tagging
- Stochastic POS Tagging
- Transformation-Based POS Tagging
- Hidden Markov Models (HMM)
- Maximum Entropy Models
- Context-Free Grammar (CFG)
- Constituency Parsing
- Treebanks
- Normal Forms for Grammar
- Top-Down Parsing
- Bottom-Up Parsing
- CYK Parsing Algorithm
- Probabilistic Context-Free Grammar (PCFG)
- Feature Structures
- Unification

Introduction

In **Unit I**, an NLP system learned how to process **individual words**. It identified patterns using Regular Expressions, recognized valid sequences using Finite-State Automata, divided text into

tokens, analyzed word forms through Morphological Processing, corrected spelling mistakes using Minimum Edit Distance, and predicted word sequences using Statistical Language Models.

These techniques help the system recognize and process words correctly.

However, recognizing words is only the beginning of language understanding.

Consider the following user input.

David opened the door yesterday.

After tokenization, the NLP system identifies the individual words.

```
David | opened | the | door | yesterday
```

At this stage, the system knows **what the words are**, but it does not yet know **what role each word plays** in the sentence.

Several important questions immediately arise.

- Which word represents the person?
- Which word represents the action?
- Which word receives the action?
- Which word describes time?
- Does the sentence follow the grammatical rules of English?

A human reader answers these questions almost instantly because years of language experience have made grammar seem natural.

An NLP system cannot make these assumptions.

It must analyze the grammatical role of every word before it can understand the sentence.

This is the purpose of **Word-Level Analysis** and **Syntactic Analysis**.

Why Is Another Unit Needed?

Imagine reading these two sentences.

David wrote a letter.

Letter wrote David a.

Both sentences contain exactly the same words.

Yet, one sentence is grammatically correct and the other is not.

Why?

The difference is **not the words**.

The difference is **how the words are arranged**.

Understanding this arrangement is called **Syntax**.

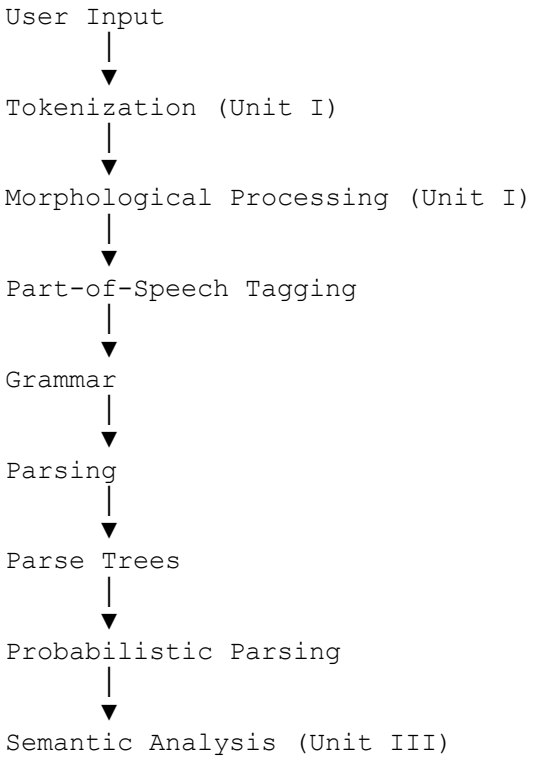
To analyze syntax, the NLP system must first determine the grammatical role of every word.

This leads to the first topic of this unit:

Part-of-Speech (POS) Tagging.

Learning Flow

The topics in this unit are closely connected. Each topic prepares the foundation for the next one.



Rather than studying these topics separately, it is helpful to think of them as successive steps performed by an NLP system while understanding a sentence.

Part-of-Speech (POS) Tagging

Imagine reading the following sentence.

Mary visited Elizabeth yesterday.

Most readers immediately understand that:

- **Mary** is a person's name.
- **visited** describes an action.
- **Elizabeth** is another person's name.
- **yesterday** indicates time.

Humans identify these grammatical roles almost automatically.

An NLP system, however, cannot.

After tokenization, it simply sees:

```
Mary | visited | Elizabeth | yesterday
```

At this stage, another important question naturally arises.

How does the NLP system determine that "Mary" is a noun, "visited" is a verb, and "yesterday" is an adverb?

The answer lies in **Part-of-Speech Tagging**.

What is a Part of Speech?

A **Part of Speech** is the grammatical category of a word based on the role it performs in a sentence.

Some common Parts of Speech are:

Part of Speech	Purpose	Example

Part of Speech	Purpose	Example
Noun	Names a person, place, object, or idea	David, Jerusalem, book
Pronoun	Replaces a noun	he, she, they
Verb	Expresses an action or state	reads, prayed, visited
Adjective	Describes a noun	faithful, beautiful
Adverb	Describes a verb, adjective, or another adverb	quickly, yesterday
Determiner	Introduces or specifies a noun	a, an, the
Preposition	Shows the relationship between words	in, on, under
Conjunction	Connects words or sentences	and, but, because

An Important Observation

Many learners initially assume that **every English word belongs to only one Part of Speech**.

This is not always true.

Consider the word **book**.

Sentence 1

Joseph bought a book.

Here, **book** refers to an object.

Therefore,

`book → Noun`

Sentence 2

They will book the tickets tomorrow.

Here, **book** describes an action.

Therefore,

```
book → Verb
```

The same word performs different grammatical roles in different sentences.

This situation is known as **lexical ambiguity**.

Because of lexical ambiguity, identifying the correct Part of Speech becomes a challenging task for an NLP system.

What is Part-of-Speech Tagging?

Part-of-Speech (POS) Tagging is the process of assigning the correct grammatical category to every word in a sentence according to its meaning and context.

In simple terms,

POS Tagging tells the NLP system whether a word is functioning as a noun, verb, adjective, adverb, or another Part of Speech in a particular sentence.

Example

Consider the sentence:

Daniel reads the Scriptures every morning.

After tokenization,

```
Daniel | reads | the | Scriptures | every | morning
```

After POS Tagging,

Word	POS Tag
Daniel	Proper Noun (NNP)
reads	Verb (VBZ)

Word	POS Tag
the	Determiner (DT)
Scriptures	Proper Noun (NNP) *(or Common Noun depending on context)*
every	Determiner (DT)
morning	Noun (NN)

The NLP system can now identify the grammatical role of each word.

Why is POS Tagging Important?

POS Tagging forms the foundation for many NLP tasks.

Without identifying the grammatical role of each word, an NLP system cannot accurately perform:

- Parsing
- Machine Translation
- Information Extraction
- Grammar Checking
- Question Answering
- Chatbots
- Speech Recognition
- Text Summarization

In other words, POS Tagging acts as the bridge between recognizing words and understanding sentences.

Connecting the Concepts

At this stage, the NLP system knows **what POS Tagging is**, but another important question still remains.

How does the NLP system actually assign these tags?

Does it use grammar rules?

Does it learn from examples?

Does it calculate probabilities?

Researchers developed several techniques to answer these questions. The earliest approach was **Rule-Based POS Tagging**, where linguists manually created grammar rules to guide the tagging process.

This is the next concept to be studied.

Rule-Based POS Tagging

In the previous topic, Part-of-Speech Tagging was introduced as the process of assigning the correct grammatical category to every word in a sentence.

The next question naturally is:

How does an NLP system actually assign these grammatical tags?

One possible solution is quite simple.

Suppose an NLP engineer wants the computer to recognize nouns, verbs and adjectives.

Instead of asking the computer to learn the language on its own, the engineer manually writes grammar rules for it.

For example,

```
If a word follows "the",
it is usually a noun.
```

Another rule may be

```
If a word ends with "ly",
it is usually an adverb.
```

The NLP system simply follows these rules one after another while analysing a sentence.

This approach is called **Rule-Based POS Tagging**.

Definition

Rule-Based POS Tagging is a technique in which an NLP system assigns Parts of Speech using manually written grammatical rules and a lexicon (dictionary).

In simple words,

The NLP system looks up a word in a dictionary and then applies grammar rules to choose the most appropriate Part of Speech.

Two Things the NLP System Needs

For this approach to work, the NLP system requires two resources.

1. A Lexicon

A **lexicon** is a dictionary specially prepared for NLP applications.

Unlike an ordinary English dictionary, a lexicon stores the possible grammatical categories of a word.

For example,

Word	Possible POS Tags
book	Noun, Verb
can	Modal Verb, Noun
light	Noun, Verb, Adjective
open	Verb, Adjective

Notice that some words have more than one possible tag.

The lexicon does **not** decide which tag is correct.

It simply lists all possible tags.

The final decision is made using grammar rules.

---

2. Grammar Rules

Grammar rules tell the NLP system **when** a particular POS tag should be chosen.

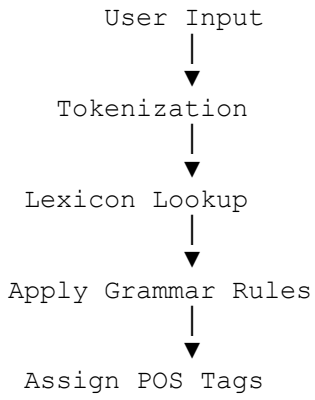
For example,

Rule	Assigned Tag
If a word follows **the**	Usually Noun
If a word follows **will**	Usually Verb
If a word ends with **-ly**	Usually Adverb
If a word begins with a capital letter and is a known name	Proper Noun

These rules are prepared manually by linguists and NLP experts.

How Does Rule-Based POS Tagging Work?

The process can be understood in four simple steps.



Step 1 – Tokenization

Suppose a user enters the sentence

Joseph opened the door.

After tokenization,

```
Joseph | opened | the | door
```

The words are now ready for grammatical analysis.

Step 2 – Lexicon Lookup

The NLP system searches its lexicon.

Word	Possible POS Tags
Joseph	Proper Noun
opened	Verb
the	Determiner
door	Noun, Verb

At this stage, notice that **door** has two possible tags.

The NLP system still does not know which one is correct.

Step 3 – Apply Grammar Rules

The system now applies its grammar rules.

One rule states:

```
If a word follows "the",
it is usually a noun.
```

The sentence contains

```
the door
```

Therefore,

```
door → Noun
```

Another example is

They will book tickets.

The lexicon stores

```
book → Noun, Verb
```

A grammar rule states

```
If a word follows "will",
it is usually a verb.
```

Therefore,

```
book → Verb
```

The rule helps the NLP system choose the correct grammatical category.

Worked Example

Consider the sentence

Mary quickly opened the window.

Step 1

Tokenization

```
Mary | quickly | opened | the | window
```

Step 2

Lexicon Lookup

Word	Possible POS Tags
Mary	Proper Noun
quickly	Adverb
opened	Verb

Word	Possible POS Tags
the	Determiner
window	Noun, Verb

Step 3

Apply Grammar Rules

The rule

```
If a word follows "the",
it is usually a noun.
```

assigns

```
window → Noun
```

The rule

```
Words ending with "ly"
are usually adverbs.
```

assigns

```
quickly → Adverb
```

Final POS Tags

Word	POS Tag
Mary	Proper Noun
quickly	Adverb
opened	Verb
the	Determiner
window	Noun

The sentence has now been successfully tagged.

Why Was Rule-Based POS Tagging Popular?

Rule-Based POS Tagging was one of the earliest successful POS tagging techniques because it was easy to understand and easy to explain.

Every tagging decision could be traced back to a specific grammar rule.

For example,

```
the window
```

The system chooses **window → Noun** because of the rule associated with the word **the**.

This makes the entire decision-making process transparent.

Why Was Another Approach Needed?

Although Rule-Based POS Tagging works well for simple sentences, it has several practical limitations.

First, English contains thousands of grammatical patterns and many exceptions.

Writing rules for every possible sentence becomes extremely difficult.

Second, language constantly evolves.

New words, expressions and sentence patterns appear regularly.

Every new situation requires additional grammar rules.

Finally, many words are highly ambiguous.

A single word may have several possible grammatical categories, and a fixed set of rules may not always choose the correct one.

As the number of rules increases, maintaining and updating the system becomes increasingly difficult.

Applications

Rule-Based POS Tagging is still useful in:

- Educational grammar tools
- Controlled language systems
- Grammar checking applications
- Early NLP systems
- Linguistic research

Although modern NLP systems mainly use statistical and neural approaches, Rule-Based POS Tagging remains important because it introduced the basic idea of automatically assigning grammatical categories to words.

Connecting the Concepts

Rule-Based POS Tagging showed that grammar rules could successfully identify the Parts of Speech of many words.

However, another challenge soon became apparent.

Suppose the word **book** appears in two different sentences.

Joseph bought a book.

They will book the tickets.

Instead of writing separate grammar rules for every possible situation, another idea emerged.

What if the NLP system could observe how words are used in real language and choose the most likely Part of Speech?

This idea led to the development of **Stochastic POS Tagging**, where decisions are made using probabilities learned from language data rather than relying entirely on manually written grammar rules.

Stochastic Part-of-Speech (POS) Tagging

I'm also going to make one important improvement compared to the previous topics.

In Rule-Based POS Tagging, we already learned:

- what POS tagging is,
- what a lexicon is,
- why ambiguity exists,
- why grammar rules are needed.

I **will not explain them again.**

Instead, I'll build directly on what students already know. This makes the handbook feel like one continuous book.

Stochastic Part-of-Speech (POS) Tagging

In the previous topic, Rule-Based POS Tagging assigned grammatical tags by applying manually written grammar rules.

Although this approach worked well for simple sentences, another challenge soon appeared.

Consider the following two sentences.

Joseph bought a book.

They will book the tickets.

In both sentences, the word **book** appears.

However, its grammatical role is different.

In the first sentence, **book** is a **noun**.

In the second sentence, **book** is a **verb**.

A rule-based system can identify the correct tag if an appropriate grammar rule already exists. But English contains thousands of such situations.

This naturally raises another question.

Can an NLP system learn from how words are actually used in language instead of depending entirely on manually written grammar rules?

The answer is **Yes**.

Instead of asking,

"Which grammar rule should I apply?"

the NLP system asks,

"Which Part of Speech is most likely to be correct in this context?"

This approach is known as **Stochastic POS Tagging**.

What Does "Stochastic" Mean?

The word **stochastic** simply means **based on probability**.

Instead of making a decision using only grammar rules, the NLP system uses **probability** to choose the most likely Part of Speech.

In simple words,

Stochastic POS Tagging assigns the POS tag that has the highest probability of being correct.

Where Do These Probabilities Come From?

Another important question naturally arises.

How does the NLP system know these probabilities?

The probabilities are **not guessed**.

They are learned from a large collection of correctly tagged sentences called an **annotated corpus**.

(The concept of a corpus was introduced in Unit I. Here, it is simply used as the source from which the NLP system learns language patterns.)

As the NLP system processes thousands or millions of tagged sentences, it begins to notice patterns.

For example,

```
Proper Noun + Verb
```

appears very frequently.

Similarly,

```
Determiner + Noun
```

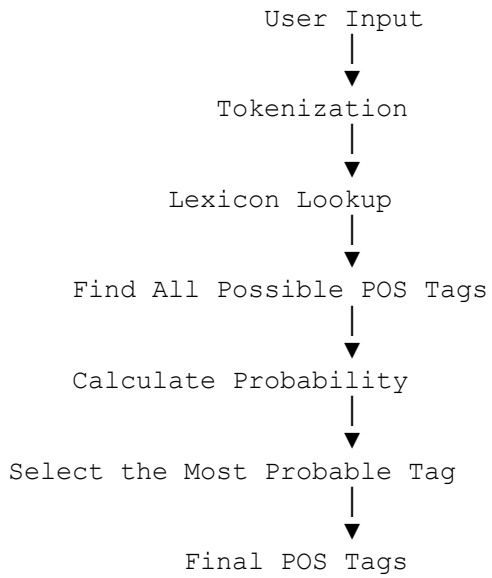
appears much more often than

```
Determiner + Verb
```

These observations are converted into probability values.

How Does Stochastic POS Tagging Work?

The overall process is shown below.



Notice that the first few steps are the same as Rule-Based POS Tagging.

The difference lies only in the **decision-making stage**.

Instead of applying grammar rules, the NLP system compares probabilities.

Worked Example

Consider the sentence

Mary can sing.

From the lexicon,

Word	Possible POS Tags
Mary	Proper Noun
can	Modal Verb, Noun
sing	Verb, Noun

The word **can** has two possible tags.

The NLP system now checks its language data.

Suppose it finds the following observations.

Pattern	Frequency
can + Verb	9,800
can + Noun	150

Since **can + Verb** occurs much more frequently,

the NLP system assigns

```
can → Modal Verb
```

Now consider another sentence.

Mary bought a can.

Again,

```
can
```

has two possible tags.

This time, the previous word is

`a`

The language data shows

Pattern	Frequency
a + Noun	Very High
a + Verb	Very Low

Therefore,

`can → Noun`

The same word receives two different POS tags because the surrounding words change the probability.

Why Is This Better Than Rule-Based Tagging?

Rule-Based POS Tagging depends on manually written grammar rules.

Stochastic POS Tagging depends on **patterns learned from real language**.

Instead of writing thousands of rules,

the NLP system learns from examples.

As more language data becomes available,

the system generally becomes more accurate.

Advantages

- Learns from real language usage.
- Handles ambiguous words more effectively.
- Requires fewer manually written grammar rules.
- Usually provides higher accuracy than Rule-Based POS Tagging.

Limitations

Although Stochastic POS Tagging performs better than Rule-Based Tagging, it also has some limitations.

It requires a **large annotated corpus** for training.

If the training data is limited or does not contain enough examples of a particular word, the calculated probabilities may not be reliable.

Newly introduced words may also be difficult to tag because the system has little or no information about them.

Comparison with Rule-Based POS Tagging

Feature	Rule-Based POS Tagging	Stochastic POS Tagging
Decision Method	Grammar Rules	Probability
Knowledge Source	Linguistic Rules	Annotated Corpus
Manual Rules	Required	Minimal
Learns from Data	No	Yes
Handles Ambiguity	Limited	Better
Accuracy	Good for simple sentences	Generally Higher

Connecting the Concepts

Stochastic POS Tagging significantly reduced the need for manually written grammar rules by learning from language data.

However, another important question still remained.

Is choosing the most probable tag for each word enough to correctly tag an entire sentence?

Consider the sentence:

David can fish.

Should **can** be tagged first?

Or should the NLP system consider the tags of the surrounding words before making a decision?

In other words,

Can the sequence of words in a sentence help the NLP system make better tagging decisions?

Answering this question required a mathematical model that could analyse an **entire sequence of words and tags**, rather than individual words alone.

This led to the development of the **Hidden Markov Model (HMM)**, one of the most important statistical models used in Natural Language Processing.

Hidden Markov Model (HMM)

Why is Another POS Tagging Model Needed?

In the previous topic, Stochastic POS Tagging was introduced as a technique that assigns the Part of Speech with the **highest probability**.

For many sentences, this approach works well.

However, another important question naturally arises.

Is looking at one word at a time enough to correctly understand an entire sentence?

To answer this question, consider the following user input.

David can fish.

The word **can** may be a **modal verb** or a **noun**.

Similarly, **fish** may be a **noun** or a **verb**.

If the NLP system considers each word independently, several tag combinations become possible.

Sentence	Possible POS Tags
David can fish	Noun – Modal Verb – Verb
David can fish	Noun – Verb – Noun
David can fish	Noun – Noun – Verb
David can fish	Noun – Noun – Noun

All of these combinations appear possible when each word is examined separately.

This creates a new challenge.

How does the NLP system decide which entire sequence of tags is most likely to be correct?

The answer cannot be found by looking at individual words alone.

The **relationship between neighbouring words and neighbouring tags** must also be considered.

This need led to the development of the **Hidden Markov Model (HMM).**

Understanding the Main Idea

Before introducing Hidden Markov Models, it is useful to understand the problem they were designed to solve.

Imagine reading the sentence

Mary will sing tomorrow.

Most people immediately understand that

- **Mary** is a noun.
- **will** is a modal verb.
- **sing** is a verb.

Notice something interesting.

The word

```
sing
```

is recognised as a verb partly because it follows

```
will
```

Similarly,

```
will
```

is recognised as a modal verb because it appears between

```
Mary
```

and

```
sing
```

In other words,

Humans do not identify each word independently. They understand each word by considering the surrounding words.

An NLP system should do the same.

This is exactly the idea behind Hidden Markov Models.

Why Isn't Stochastic POS Tagging Enough?

Stochastic POS Tagging chooses the **most probable tag** for an individual word.

Hidden Markov Models go one step further.

Instead of asking

"Which tag is most probable for this word?"

HMM asks

"Which sequence of tags is most probable for the entire sentence?"

This small difference makes a significant improvement in tagging accuracy.

A Simple Analogy

Imagine travelling from **Hyderabad** to **Chennai**.

Suppose there are many possible roads.

Instead of choosing the **best individual road**, most people try to choose the **best complete route**.

The same idea applies to HMM.

Instead of selecting the best tag for each individual word, HMM selects the **best sequence of tags** for the entire sentence.

What is a Hidden Markov Model?

A **Hidden Markov Model (HMM)** is a probabilistic model that predicts the most likely sequence of hidden states based on a sequence of observed data.

In Natural Language Processing,

- the **observed data** are the words in the sentence,
- the **hidden states** are the Parts of Speech.

In simple words,

An HMM identifies the most probable sequence of Parts of Speech for a given sentence by considering both the words and their grammatical relationships.

Why is it Called "Hidden"?

Another important question naturally arises.

If the NLP system can see the words, what exactly is hidden?

Consider the sentence

Joseph can pray.

The NLP system can directly see the words.

```
Joseph
can
pray
```

These words are called the **observations** because they are visible.

However, the correct Parts of Speech are **not directly visible**.

Initially, the system does not know whether

```
can
```

is

- a modal verb,

or

- a noun.

Similarly,

```
pray
```

may be

- a noun,

or

- a verb.

These grammatical categories must be **predicted**.

Since they are **not directly observed**, they are called **hidden states**.

This explains the name

Hidden Markov Model.

Understanding "Hidden" and "Observed"

The following table summarises the idea.

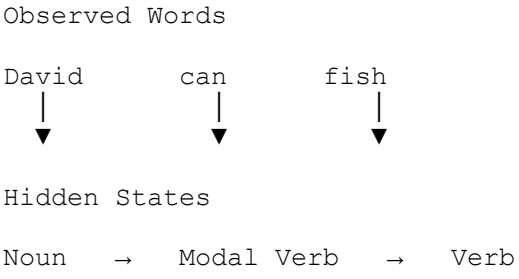
What the NLP System Receives	What the NLP System Must Predict
David	Proper Noun
can	Modal Verb
fish	Verb

The left column contains the **observations**.

The right column contains the **hidden states**.

The purpose of an HMM is to discover these hidden states.

Visualising an HMM



The words are visible.

The Parts of Speech must be predicted.

An Important Observation

Students often wonder:

If the words are already visible, why can't the NLP system simply assign the correct Parts of Speech immediately?

The reason is that many English words have more than one possible grammatical category.

For example,

`can`

may be

- Modal Verb
- Noun

Similarly,

`fish`

may be

- Noun
- Verb

Therefore, the NLP system must determine **which sequence of tags is most likely**, rather than making isolated decisions for individual words.

This is precisely the problem that Hidden Markov Models solve.

Connecting the Concepts

The main idea behind Hidden Markov Models has now been established.

An HMM predicts the most probable sequence of hidden grammatical states from a sequence of observed words.

This naturally raises another question.

What information does an HMM use to make these predictions?

To answer this, the model introduces four important components:

- States
- Observations
- Transition Probability
- Emission Probability

Understanding these components is the next step before studying how an HMM actually performs POS tagging.

Components of a Hidden Markov Model

The previous section explained that an HMM predicts the **most probable sequence of Parts of Speech** for a sentence.

To make this prediction, the model uses four important components.

1. States
2. Observations
3. Transition Probability
4. Emission Probability

These four components work together to determine the most likely sequence of POS tags.

Before studying probabilities, it is important to understand **States** and **Observations**, because they form the foundation of the entire model.

States

Consider the following sentence.

David can fish.

The NLP system receives the words

```
David    can    fish
```

However, its objective is **not simply to read the words**.

Its objective is to determine the grammatical role of each word.

For example,

Word	POS Tag
David	Proper Noun
can	Modal Verb
fish	Verb

These POS tags are called **States** in an HMM.

Definition

A **State** represents the grammatical category that the NLP system is trying to predict for each word.

In POS Tagging, the states are the Parts of Speech.

Some common states are

```
Noun

Verb

Adjective

Adverb

Pronoun

Determiner

Preposition
```

The NLP system moves from one state to another while analysing the sentence.

Why are they called States?

Another important question naturally arises.

Why are Parts of Speech called "States"?

Think of a traveller moving from one city to another.

For example,

```
Hyderabad
```

```
↓
Kurnool
↓
Kadapa
↓
Tirupati
```

Each city represents a **state** in the journey.

Similarly, while analysing a sentence, the NLP system moves through a sequence of grammatical categories.

For example,

```
Proper Noun
↓
Modal Verb
↓
Verb
```

Each grammatical category represents a **state**.

Observations

If the POS tags are called **States**, then what does the NLP system actually observe?

The answer is simple.

It observes the **words**.

Consider the sentence

Mary will sing.

The NLP system can directly see

```
Mary
```

```
will

sing
```

These visible words are called **Observations**.

Definition

An **Observation** is the actual word received by the NLP system.

In POS Tagging,

- Words = Observations
- POS Tags = States

States vs Observations

This is one of the most important ideas in HMM.

Observations	States
Visible words	Hidden POS Tags
Received as input	Predicted by the model
Known	Unknown initially

Consider the sentence

Joseph opened the door.

The NLP system immediately receives

```
Joseph

opened

the

door
```

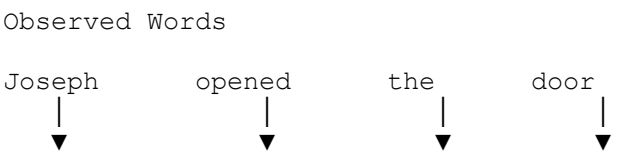
These are the **observations**.

It must then predict

```
Proper Noun

Verb

Determiner

Noun
```

These are the **states**.

Visualising States and Observations



Notice that

- the **words** are visible,
- the **states** are predicted.

An Important Observation

Students often wonder:

If the words are already available, why can't the NLP system directly assign the correct POS tags?

The answer is that **one word may belong to more than one grammatical category.**

Consider the word

```
light
```

Sentence 1

The light is bright.

Here,

```
light → Noun
```

Sentence 2

The room is light.

Here,

```
light → Adjective
```

Sentence 3

Please light the lamp.

Here,

```
light → Verb
```

The observation is the same word.

The state changes depending on the sentence.

This is why HMM predicts **states**, not words.

Relationship Between States and Observations

The following diagram summarises the relationship.

```
Observed Words

David      can      fish
  |         |         |
  ▼         ▼         ▼

Hidden States

Proper Noun → Modal Verb → Verb
```

The NLP system always begins with the observations.

Its goal is to discover the hidden sequence of states.

Connecting the Concepts

The first two components of an HMM are now clear.

- **Observations** are the words received by the NLP system.
- **States** are the Parts of Speech that the system must predict.

However, another important question now arises.

Consider the sentence

David can fish.

How does the NLP system know that a **Proper Noun** is likely to be followed by a **Modal Verb**?

Similarly, how does it know that the word **fish** is likely to represent a **Verb** in this sentence rather than a **Noun**?

To answer these questions, the HMM introduces two probabilities:

- **Transition Probability**, which describes how one state follows another.
- **Emission Probability**, which describes how likely a state is to produce a particular word.

These two probabilities form the mathematical foundation of the Hidden Markov Model and will be studied next.

Transition Probability

The previous section introduced two important ideas in a Hidden Markov Model.

- **Observations** are the words received by the NLP system.
- **States** are the Parts of Speech that the NLP system predicts.

At this stage, another important question naturally arises.

How does the NLP system decide which state should come next?

Consider the sentence:

David will sing.

The predicted POS tags are:

```
Proper Noun → Modal Verb → Verb
```

Notice that these tags appear in a particular order.

A **Proper Noun** is followed by a **Modal Verb**, and the **Modal Verb** is followed by a **Verb**.

The NLP system must determine whether this sequence is likely to occur in English.

This is where **Transition Probability** becomes important.

Understanding the Idea

Imagine walking through a building.

Suppose the rooms are arranged like this.

```
Entrance

↓

Reception

↓

Office

↓

Conference Room
```

From the **Reception**, it is quite common to move to the **Office**.

However, moving directly from the **Reception** to the **Conference Room** may be less common.

After observing many people moving through the building, it becomes possible to estimate which movement is more likely.

The same idea is used in an HMM.

Instead of moving between rooms, the NLP system moves between **Parts of Speech**.

For example,

```
Proper Noun
```

↓

Verb

↓

Noun

or

Pronoun

↓

Modal Verb

↓

Verb

Some sequences occur frequently in English, while others occur very rarely.

Transition Probability measures **how likely one state is to follow another**.

Definition

Transition Probability is the probability of moving from one state to the next state.

In POS Tagging, it represents the probability that one Part of Speech follows another.

In simple words,

Transition Probability tells the NLP system how likely one grammatical category is to follow another.

Example 1

Consider the sentence

Mary reads books.

The POS tags are

```
Proper Noun → Verb → Noun
```

Suppose the NLP system has analysed millions of English sentences.

It may discover that

```
Proper Noun → Verb
```

appears very frequently.

Therefore,

the transition from

```
Proper Noun
↓
Verb
```

has a **high transition probability**.

Example 2

Now consider this sequence.

```
Verb
↓
Verb
```

Although this sequence can occur in English, it is much less common than

```
Proper Noun
↓
Verb
```

Therefore,

its transition probability is lower.

A Simple Comparison

Imagine the NLP system has learned the following patterns.

State Sequence	Frequency
Proper Noun → Verb	Very High
Determiner → Noun	Very High
Modal Verb → Verb	Very High
Verb → Determiner	Low
Determiner → Verb	Very Low

From these observations, the NLP system concludes that some state transitions are much more likely than others.

Worked Example

Consider the sentence

Joseph will pray.

Possible POS tags:

Word	Possible POS Tags
Joseph	Proper Noun
will	Modal Verb, Noun
pray	Verb, Noun

The NLP system now examines the sequence of states.

Option 1

```
Proper Noun

↓

Modal Verb

↓

Verb
```

This sequence occurs frequently in English.

Option 2

```
Proper Noun
↓
Noun
↓
Noun
```

This sequence is much less common.

Since the first sequence has a higher transition probability, the NLP system chooses it.

Why Doesn't the NLP System Look Only at Individual Words?

Another important question naturally arises.

Suppose the word

```
can
```

appears.

Looking only at the word,

the NLP system cannot decide whether it is

- a Modal Verb, or
- a Noun.

However, if the previous state is

```
Proper Noun
```

and the next word is

```
swim
```

the sequence

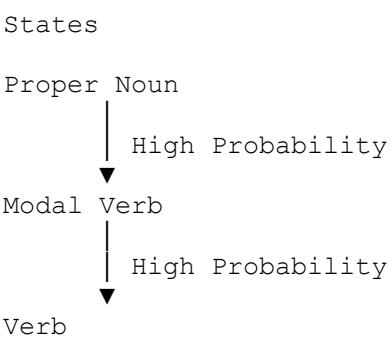
```
Proper Noun

↓

Modal Verb

↓

Verb
```

is much more likely than

```
Proper Noun

↓

Noun

↓

Verb
```

The surrounding states provide valuable information that helps the NLP system choose the correct POS tag.

Visualising Transition Probability



The arrows represent the movement from one state to another.

Each arrow has a probability associated with it.

The higher the probability, the more likely that transition is.

An Important Observation

Transition Probability **does not look at the actual words**.

It considers **only the sequence of states**.

For example,

```
Proper Noun

↓

Verb
```

The model does not care whether the noun is

- David
- Mary
- Joseph
- Daniel

or whether the verb is

- reads
- sings
- prays
- writes

It only considers that a **Proper Noun is followed by a Verb**.

This is an important distinction because many learners mistakenly think Transition Probability works with words.

It actually works with **states**.

Key Difference

Transition Probability Looks At	
Proper Noun → Verb	States
Verb → Noun	States

Transition Probability Looks At
Determiner → Noun States

Words are not considered here.

Connecting the Concepts

Transition Probability helps the NLP system determine **how likely one grammatical category is to follow another**.

However, another important question still remains.

Knowing that a **Verb** is likely to follow a **Modal Verb** does not tell the system **which actual word** should appear.

For example,

if the predicted state is

`Verb`

how does the NLP system decide whether the observed word

`pray`

or

`fish`

or

`sing`

belongs to that state?

To answer this question, the HMM introduces another probability called **Emission Probability**.

Unlike Transition Probability, which connects **one state to another**, Emission Probability connects a **state to an observed word**.

This is the next component of the Hidden Markov Model.

Emission Probability

In the previous topic, Transition Probability explained **how one state is likely to follow another state**.

However, another important question still remains.

Suppose the NLP system predicts that the current state is

`Verb`

How does it determine **which word** belongs to that state?

For example,

can the state **Verb** produce the word

`sing`

or

`book`

or

`pray`

The NLP system must answer this question before assigning the final POS tags.

This is the purpose of **Emission Probability**.

Understanding the Idea

Imagine a classroom.

The teacher asks,

Which student answered the question?

Suppose the answer is

David

Here,

- the **student** is already known,
- the teacher now wants to know **who produced the answer**.

Similarly,

an HMM already predicts a grammatical state.

Now it asks,

Which word is most likely to be produced by this state?

Definition

Emission Probability is the probability that a particular state produces a particular observed word.

In POS Tagging,

it represents the probability that a **Part of Speech generates a particular word**.

In simple words,

Emission Probability tells the NLP system how likely a particular word belongs to a particular Part of Speech.

Example 1

Consider the word

`pray`

Suppose the NLP system has learned from its training data that

State	Word	Frequency
Verb	pray	Very High

State	Word	Frequency
Noun	pray	Very Low

The NLP system therefore concludes

```
Verb
↓
pray
```

has a **high emission probability**.

Example 2

Now consider

```
beautiful
```

Suppose the corpus shows

State	Word	Frequency
Adjective	beautiful	Very High
Verb	beautiful	Almost Never

The NLP system therefore predicts

```
Adjective
↓
beautiful
```

How Is This Different from Transition Probability?

This is the most important distinction in HMM.

Transition Probability answers

Which state comes next?

Example

```
Noun

↓

Verb
```

Emission Probability answers

Which word belongs to this state?

Example

```
Verb

↓

sing
```

Notice the difference.

Transition connects

```
State

↓

State
```

Emission connects

```
State

↓

Word
```

Visualising the Difference

Transition Probability

```
Proper Noun

↓

Verb

↓

Noun
```

The arrows connect **states**.

Emission Probability

```
Proper Noun

↓

David
Verb

↓

opened
Noun

↓

door
```

Here,

each state produces an observed word.

Worked Example

Consider the sentence

Daniel sings hymns.

The observations are

```
Daniel

sings
```

```
hymns
```

The HMM predicts

```
Proper Noun

Verb

Noun
```

Now consider each state separately.

Proper Noun

Possible words

```
Daniel

Mary

Joseph

David
```

Since

```
Proper Noun

↓

Daniel
```

occurs frequently,

its emission probability is high.

Verb

Possible words

```
sings

reads

writes

prays
```

Since

```
Verb
↓
sings
```

is common,

its emission probability is also high.

Noun

Possible words

```
hymns
books
letters
songs
```

Again,

```
Noun
↓
hymns
```

has a high emission probability.

The HMM therefore considers this sequence very likely.

Transition Probability vs Emission Probability

Transition Probability	Emission Probability
Connects one state to another	Connects a state to a word

Transition Probability	Emission Probability
State → State	State → Word
Helps decide the sequence of tags	Helps decide which word fits each tag

---

An Important Observation

Many learners think that both probabilities do the same job.

They do not.

Think of HMM as answering **two different questions**.

Question 1

Which grammatical category should come next?

Transition Probability answers this.

Question 2

Does this word belong to that grammatical category?

Emission Probability answers this.

Both probabilities work together to produce the final POS tags.

Bringing Everything Together

An HMM now has all the information it needs.

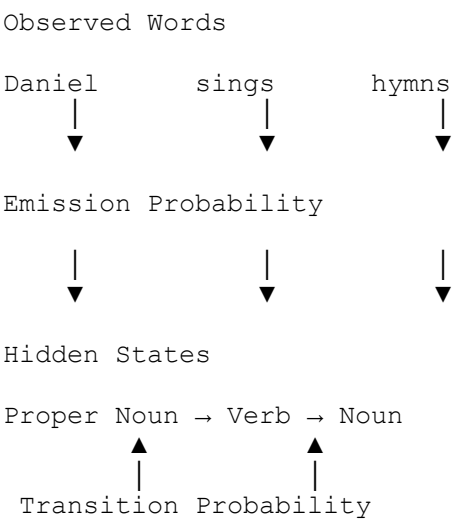
It knows

- the observed words,
- the possible states,
- how states are connected,

- which words belong to which states.

The remaining task is to combine all this information and choose the **best sequence of POS tags**.

HMM Summary



Notice how the two probabilities work together.

- **Transition Probability** moves **between states**.
- **Emission Probability** links **states to words**.

Neither one alone is sufficient.

Connecting the Concepts

The four basic components of a Hidden Markov Model have now been studied.

- Observations
- States
- Transition Probability
- Emission Probability

Another important question now arises.

How does the NLP system combine all these components to select the single best sequence of Parts of Speech for an entire sentence?

The system cannot simply choose the best transition or the best emission independently. It must consider **all possible tag sequences** and identify the one with the highest overall probability.

This problem is solved by the **Viterbi Algorithm**, which efficiently finds the most probable sequence of hidden states in a Hidden Markov Model.

Building a Hidden Markov Model Step by Step

So far, the four main components of a Hidden Markov Model have been studied.

- Observations
- States
- Transition Probability
- Emission Probability

An important question now naturally arises.

How do these four components work together to analyze a sentence?

To answer this question, let us build a complete Hidden Markov Model for a simple sentence.

Step 1: The User Enters a Sentence

Suppose a user enters the sentence:

David will pray.

This is the input received by the NLP system.

Step 2: Identify the Observations

The NLP system first performs tokenization (already studied in Unit I).

The sentence is divided into individual words.

```
David     will     pray
```

These words are the **observations** because they are directly visible to the NLP system.

Word	Observation
David	Observation 1
will	Observation 2
pray	Observation 3

At this stage, the NLP system knows only the words.

It still does not know their grammatical categories.

Step 3: Identify the Possible States

Next, the NLP system looks up each word in its lexicon.

Word	Possible States (POS Tags)
David	Proper Noun
will	Modal Verb, Noun
pray	Verb, Noun

Notice that **will** and **pray** have more than one possible state.

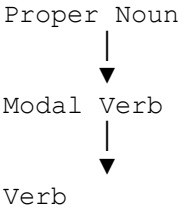
The NLP system must determine the correct one.

Step 4: Consider the Transition Probabilities

The NLP system now asks:

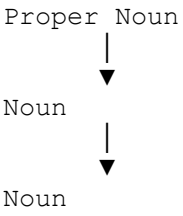
Which sequence of states is most likely?

From its training data, it has learned that the following transitions occur frequently.



These transitions have **high transition probabilities**.

Another possible sequence is



However, this sequence occurs much less frequently in English.

Therefore, it has a **lower transition probability**.

At this point, the NLP system begins to favour the first sequence.

Step 5: Consider the Emission Probabilities

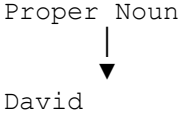
The NLP system now checks whether each state is likely to produce the observed word.

State: Proper Noun

Possible words include

```
David
Mary
Joseph
Daniel
```

Since **David** frequently appears as a Proper Noun,



has a **high emission probability**.

State: Modal Verb

Possible words include

```
will
can
may
must
```

Since **will** is commonly used as a Modal Verb,

```
Modal Verb
      |
      ▼
will
```

also has a **high emission probability**.

State: Verb

Possible words include

```
pray
read
sing
write
```

Since **pray** is frequently used as a Verb,

```
Verb
      |
      ▼
pray
```

has a **high emission probability**.

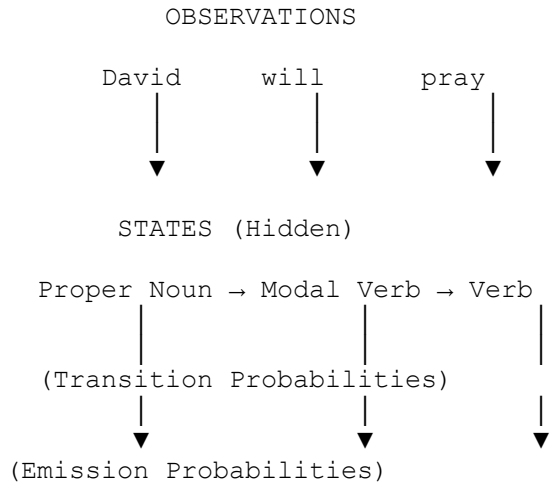
Step 6: Combine Everything

The NLP system now combines

- the observations,
- the possible states,

- the transition probabilities,
- and the emission probabilities.

The complete Hidden Markov Model can be visualized as follows.



Notice that two different types of probabilities are working together.

- **Transition Probability** connects one **state** to another.
- **Emission Probability** connects a **state** to an observed **word**.

Only when both probabilities are considered can the NLP system identify the most likely sequence of POS tags.

Step 7: Final POS Tags

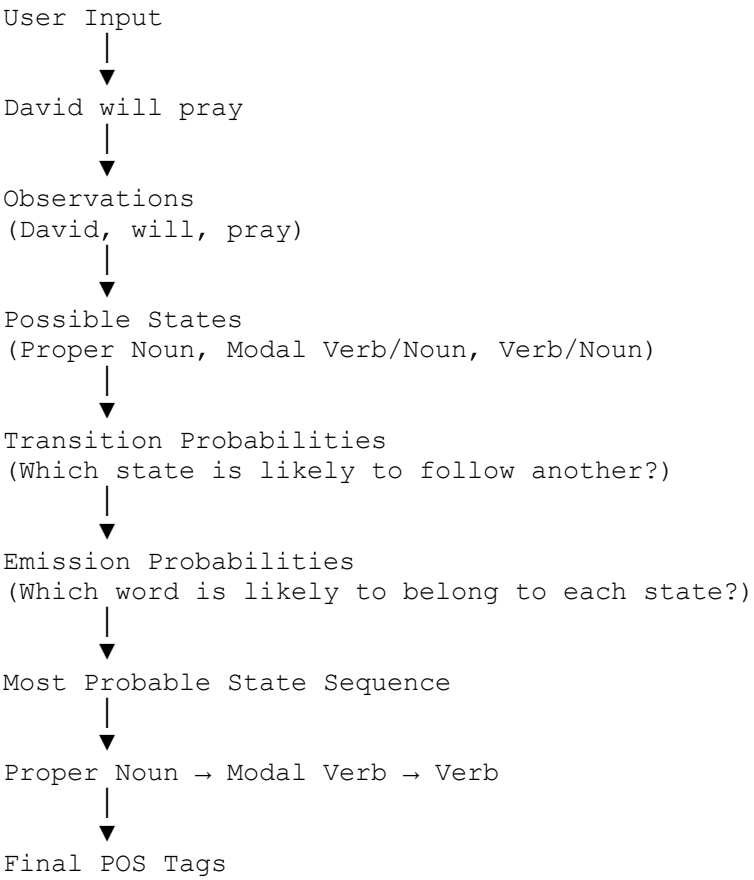
After evaluating all possible combinations, the NLP system predicts

Word	Final POS Tag
David	Proper Noun
will	Modal Verb
pray	Verb

This becomes the final output of the Hidden Markov Model.

Summary

The entire process can now be summarized as follows.



An Important Observation

It is important to remember that **the Hidden Markov Model does not immediately know the correct POS tags**.

Instead, it:

1. Receives the words (**observations**).
2. Identifies all possible POS tags (**states**).
3. Uses **transition probabilities** to evaluate likely sequences of states.
4. Uses **emission probabilities** to determine how well each state matches the observed words.
5. Chooses the **most probable sequence of states** as the final answer.

Connecting the Concepts

The Hidden Markov Model has now been built step by step.

The next question naturally arises:

If several possible state sequences exist, how does the NLP system efficiently identify the single best sequence without checking every combination one by one?

This is the role of the **Viterbi Algorithm**. It is the search algorithm used with Hidden Markov Models to efficiently find the **most probable sequence of hidden states** for a given sentence.

Viterbi Algorithm

The Hidden Markov Model has now been built step by step.

The NLP system can now:

- identify the observations (words),
- identify the possible states (POS tags),
- calculate transition probabilities,
- calculate emission probabilities.

However, another important question naturally arises.

How does the NLP system choose the single best sequence of POS tags?

Why is Another Algorithm Needed?

Consider the sentence

David can fish.

The NLP system identifies the following possible POS tags.

Word	Possible POS Tags
David	Proper Noun
can	Modal Verb, Noun

Word	Possible POS Tags
fish	Verb, Noun

Now the system has several possible tag sequences.

```
Proper Noun → Modal Verb → Verb
Proper Noun → Modal Verb → Noun
Proper Noun → Noun → Verb
Proper Noun → Noun → Noun
```

Which one should it choose?

One possible solution is to calculate the probability of **every possible sequence** and then choose the best one.

For a short sentence, this is possible.

But imagine a sentence containing **15 or 20 words**.

Each word may have two or three possible POS tags.

The number of possible tag sequences becomes extremely large.

Checking every possible sequence would take a considerable amount of time.

Therefore, the NLP system needs a faster and more efficient method.

This method is called the **Viterbi Algorithm**.

What is the Viterbi Algorithm?

The **Viterbi Algorithm** is a dynamic programming algorithm used with Hidden Markov Models to find the **most probable sequence of hidden states** for a given sequence of observations.

In simple words,

Instead of checking every possible POS tag sequence, the Viterbi Algorithm efficiently finds the most likely one.

Notice the phrase **dynamic programming**.

This concept was already introduced in **Unit I** while studying **Minimum Edit Distance**. Here, the same idea is used again to avoid repeating unnecessary calculations.

Understanding the Main Idea

Imagine travelling from **Hyderabad** to **Chennai**.

There may be hundreds of different routes.

Instead of travelling through every route to discover the best one, a navigation system quickly eliminates poor choices and continues with only the most promising routes.

The Viterbi Algorithm works in a similar way.

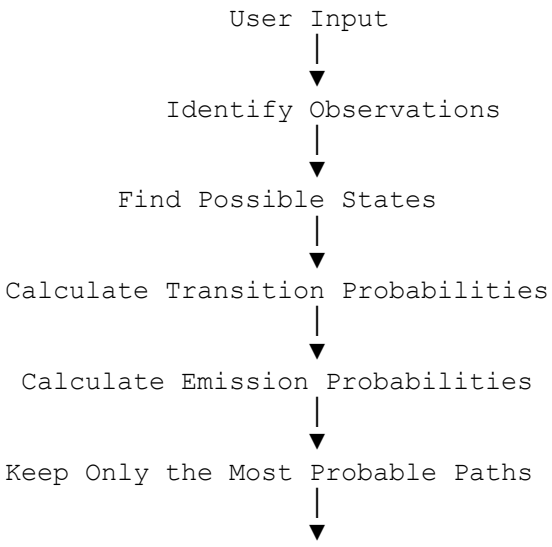
It does **not** examine every possible POS tag sequence.

Instead, it keeps track of the **best sequence found so far** at each step and discards sequences that are less likely.

This makes the tagging process much faster.

How Does the Viterbi Algorithm Work?

The overall process can be understood in the following steps.



```
   Select the Best State Sequence
```

Notice that the Viterbi Algorithm does **not** change the HMM.

It simply helps the HMM **search efficiently**.

Worked Example

Consider the sentence

Mary will sing.

The observations are

```
Mary     will     sing
```

The NLP system identifies the possible states.

Word	Possible States
Mary	Proper Noun
will	Modal Verb, Noun
sing	Verb, Noun

Initially, two possible paths exist.

```
Proper Noun
      |
Modal Verb
```

and

```
Proper Noun
      |
Noun
```

The algorithm calculates the transition and emission probabilities for both paths.

Suppose the first path has a much higher probability.

Instead of continuing with both paths,

the Viterbi Algorithm keeps only

```
Proper Noun
      |
Modal Verb
```

and discards the less likely path.

Next, it repeats the same process for the word **sing**.

Finally, it selects

```
Proper Noun

↓

Modal Verb

↓

Verb
```

as the most probable sequence.

Why is the Viterbi Algorithm Efficient?

The Viterbi Algorithm avoids repeating unnecessary calculations.

Once it determines that one path is less likely than another, it does not continue exploring that path.

As a result,

- fewer calculations are performed,
- less memory is required,
- the best solution is obtained much more quickly.

This makes the algorithm suitable for processing long sentences.

Role of the Viterbi Algorithm in HMM

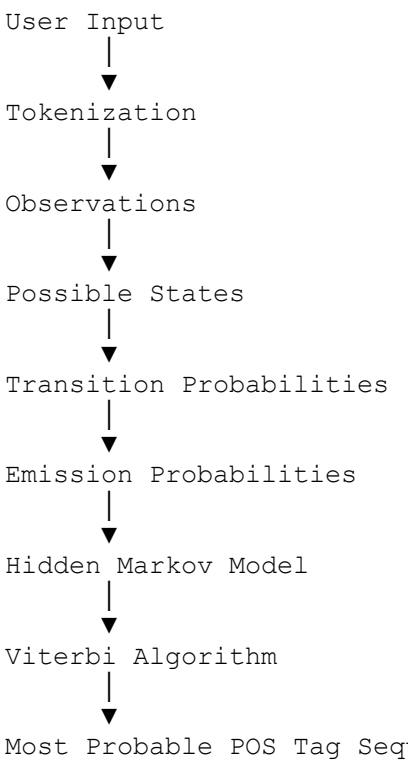
It is important to understand the relationship between HMM and the Viterbi Algorithm.

- **Hidden Markov Model (HMM)** provides the probabilistic model.
- **Viterbi Algorithm** searches that model to find the most probable sequence of states.

An HMM can be compared to a **map**, while the Viterbi Algorithm is the **navigation method** used to find the best route on that map.

Summary

The complete HMM process can now be summarized.



Key Points to Remember

- The Viterbi Algorithm is used **with** a Hidden Markov Model.
- It is based on **Dynamic Programming**, a concept already studied in Unit I.
- It finds the **most probable sequence of hidden states**.
- It avoids checking every possible tag sequence, making the process efficient.
- HMM provides the probabilities, while the Viterbi Algorithm uses those probabilities to find the best sequence.

Connecting the Concepts

Hidden Markov Models and the Viterbi Algorithm greatly improved statistical POS tagging by using probabilities and efficient search techniques.

However, another challenge still remained.

HMM assumes that each decision depends mainly on the previous state. While this assumption works well for many situations, natural language is often influenced by several features at the same time, such as neighbouring words, prefixes, suffixes, capitalization, and punctuation.

This naturally raises another question.

Can an NLP system consider many different features simultaneously instead of relying mainly on state transitions?

To answer this question, researchers developed another statistical approach called the **Maximum Entropy Model**, which will be studied next.

Maximum Entropy Models

Unlike HMM, this topic is **much shorter**. The objective is not to teach the mathematics behind Maximum Entropy but to help students understand **why it was developed and how it differs from HMM**.

Maximum Entropy Models

The Hidden Markov Model (HMM) was a major milestone in POS Tagging. It used **transition probabilities** and **emission probabilities** to predict the most likely sequence of Parts of Speech.

Although HMM performs well for many sentences, researchers observed that it also has certain limitations.

This naturally raises another important question.

Can an NLP system consider more information than just the previous POS tag while making a decision?

The answer is **Yes**.

Instead of depending mainly on transition and emission probabilities, the NLP system can examine **many different features** of a word before assigning its Part of Speech.

This idea led to the development of the **Maximum Entropy Model**.

Why Was Another Model Needed?

Consider the sentence

David can fish.

An HMM mainly considers:

- the previous POS tag,
- the current word.

However, there are many other clues available.

For example,

- Is the word capitalized?
- Does the word end with **-ing**?
- Is the previous word **will**?
- Is the next word a noun or a verb?
- Does the word usually appear at the beginning of a sentence?

All these clues can help identify the correct Part of Speech.

Instead of ignoring this information, the Maximum Entropy Model uses it while making its decision.

Understanding the Idea

Imagine trying to identify a person.

Instead of looking only at the person's height,

most people also consider

- facial features,

- hairstyle,
- voice,
- clothing,
- and many other characteristics.

The more information available, the easier it becomes to identify the correct person.

The Maximum Entropy Model follows the same idea.

Instead of using only one or two clues, it considers **many useful features** before assigning a POS tag.

Definition

A **Maximum Entropy Model** is a statistical model that predicts the most appropriate Part of Speech by considering multiple features of a word and its surrounding context.

In simple words,

The Maximum Entropy Model uses as many useful clues as possible to assign the correct POS tag.

What Are Features?

Another important question naturally arises.

What is meant by a feature?

A **feature** is any piece of information that helps the NLP system make a better decision.

For example, consider the sentence

Mary is singing beautifully.

The NLP system may use the following features.

Feature	Example
Current word	singing
Previous word	is

Feature	Example
Next word	beautifully
Word ending	-ing
Capitalization	No
Position in sentence	Third word

Each of these features provides additional information.

The NLP system combines them to predict the correct POS tag.

Worked Example

Consider the sentence

Joseph is reading a letter.

The word

```
reading
```

could theoretically be

- a Verb,
- a Noun,
- or an Adjective.

Instead of looking only at the previous POS tag, the Maximum Entropy Model examines several features.

Feature	Observation
Previous word	is
Current word	reading
Word ending	-ing
Next word	a
Common language pattern	is + Verb (-ing)

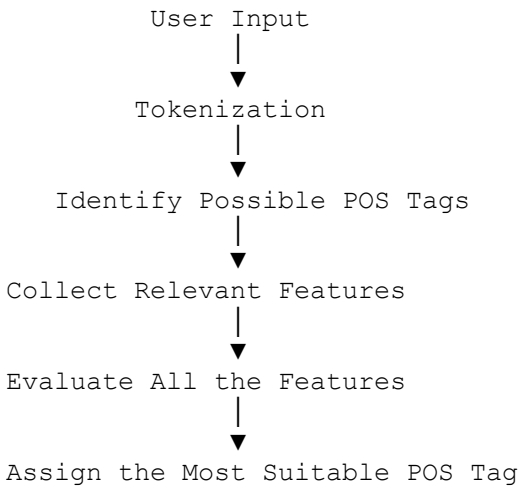
After considering all these features, the NLP system predicts

```
reading → Verb
```

The decision is based on the combined evidence rather than a single probability.

How Does the Maximum Entropy Model Work?

The overall process can be summarized as follows.



Unlike HMM, there are **no transition or emission probabilities** in this process. The model relies on the available features to make its decision.

Advantages

The Maximum Entropy Model offers several advantages.

It can consider many different features at the same time, making it more flexible than HMM.

It is particularly useful when different types of information influence the correct POS tag.

As a result, it often achieves higher tagging accuracy than traditional HMM-based systems.

Limitations

Although the Maximum Entropy Model is powerful, it also has some limitations.

It requires a large amount of annotated training data.

Training the model is computationally more expensive than simpler statistical models.

In addition, selecting useful features requires careful design, since irrelevant features may reduce the model's performance.

Comparison of HMM and Maximum Entropy Model

Hidden Markov Model	Maximum Entropy Model
Uses transition and emission probabilities	Uses multiple features
Depends mainly on the previous state	Uses many contextual clues
Simpler mathematical model	More flexible statistical model
Efficient for sequence modelling	Better at handling rich contextual information

---

Connecting the Concepts

So far, Word-Level Analysis has focused on assigning the correct Part of Speech to each word.

Once every word has been tagged, another important question naturally arises.

How do these tagged words combine to form a grammatically correct sentence?

Knowing that **David** is a noun and **reads** is a verb is useful, but it still does not explain **how these words are related** within the sentence.

To represent sentence structure formally, NLP uses a set of grammatical rules known as a **Context-Free Grammar (CFG)**.

The next topic marks the transition from **Word-Level Analysis** to **Syntactic Analysis**, where the focus shifts from **individual words** to **complete sentence structure**.

Context-Free Grammar (CFG)

So far, the NLP system has successfully identified the Part of Speech of every word in a sentence.

Consider the following user input.

David reads a book.

After POS Tagging, the NLP system identifies:

Word	POS Tag
David	Proper Noun
reads	Verb
a	Determiner
book	Noun

At this stage, the NLP system knows the grammatical role of each individual word.

However, another important question naturally arises.

Is knowing the Part of Speech of each word enough to understand the structure of the sentence?

The answer is **No**.

Knowing that **David** is a noun and **reads** is a verb does not explain **how these words are connected**.

The NLP system still needs to determine:

- Which words form a phrase?
- Which word is the subject?
- Which word is the object?
- Does the sentence follow the grammatical rules of English?

To answer these questions, the NLP system needs a set of grammar rules.

These rules are provided by **Context-Free Grammar (CFG).**

Why Do We Need Grammar?

Consider these two sentences.

David reads a book.

Book reads David a.

After POS Tagging, both sentences produce almost the same POS tags.

Sentence	POS Tags
David reads a book	Noun Verb Determiner Noun
Book reads David a	Noun Verb Proper Noun Determiner

Although the words have been tagged correctly, the second sentence is still grammatically incorrect.

Why?

Because **POS tags alone cannot describe how words should be arranged.**

The NLP system needs rules that describe **how words combine to form phrases and sentences**.

These rules are called **grammar rules**.

Understanding the Idea

Imagine building a house.

Having bricks, cement, steel and wood is not enough.

These materials must be arranged according to a building plan.

If they are arranged randomly, a proper house cannot be built.

Language works in the same way.

Words are like building materials.

Grammar is the building plan that tells the NLP system how those words should be combined to form meaningful sentences.

What is a Context-Free Grammar?

A **Context-Free Grammar (CFG)** is a collection of grammar rules that describes how words and phrases can be combined to form valid sentences.

In simple words,

A Context-Free Grammar tells the NLP system how words should be arranged to create grammatically correct sentences.

What Does "Context-Free" Mean?

Another important question naturally arises.

Why is it called "Context-Free"?

The term **Context-Free** means that a grammar rule can be applied **without considering the surrounding words**.

In other words,

a rule depends only on the grammatical symbol being expanded, not on its neighbouring symbols.

For example,

```
NP → Determiner Noun
```

This rule simply states that a **Noun Phrase (NP)** can consist of a **Determiner** followed by a **Noun**.

It can be applied wherever an **NP** appears, regardless of the surrounding sentence.

This is why it is called **Context-Free Grammar**.

Components of a Context-Free Grammar

A Context-Free Grammar consists of four basic components.

1. Non-Terminal Symbols

These represent larger grammatical units.

Examples include:

```
S

NP

VP

PP
```

where

- **S** → Sentence
- **NP** → Noun Phrase
- **VP** → Verb Phrase
- **PP** → Prepositional Phrase

These symbols help describe the structure of a sentence.

---

2. Terminal Symbols

These are the actual words that appear in the sentence.

For example,

```
David

reads

a

book
```

Unlike non-terminal symbols, terminal symbols cannot be expanded further.

---

3. Production Rules

Production rules describe how grammatical symbols are expanded.

Examples:

```
S → NP VP

NP → Determiner Noun

VP → Verb NP
```

These rules tell the NLP system how a sentence is constructed.

---

4. Start Symbol

Every CFG begins with a **Start Symbol**.

It is usually represented by

```
S
```

because every grammatical analysis begins with a complete sentence.

Building a Simple Grammar

Consider the sentence

David reads a book.

A simple CFG could be written as:

```
S  → NP VP

NP → Proper Noun

NP → Determiner Noun

VP → Verb NP

Proper Noun → David

Determiner → a
```

```
Noun → book

Verb → reads
```

These rules describe how the sentence is formed step by step.

How Does an NLP System Use CFG?

The NLP system begins with the **Start Symbol**.

```
S
```

Using the production rules,

it expands the sentence step by step.

```
S

↓

NP VP

↓

Proper Noun VP

↓

David VP

↓

David Verb NP

↓

David reads NP

↓

David reads Determiner Noun

↓

David reads a book
```

Notice that the sentence is built gradually by applying one grammar rule at a time.

Why is CFG Important?

Context-Free Grammar enables the NLP system to:

- represent sentence structure,
- identify phrases,
- check grammatical correctness,
- prepare the sentence for parsing.

Without CFG, an NLP system would know the Parts of Speech of individual words but would not know how those words combine to form a complete sentence.

Connecting the Concepts

Context-Free Grammar provides the rules needed to build a sentence.

However, another important question still remains.

Once the grammar rules are available, how does the NLP system actually use them to analyse a sentence?

Simply having grammar rules is not enough.

The NLP system must apply those rules to determine the structure of a specific sentence.

This process is known as **Constituency Parsing**.

In the next topic, we will see how an NLP system uses a Context-Free Grammar to construct a **parse tree**, showing how every word belongs to a phrase and how those phrases together form a complete sentence.

ADDITIONAL TOPIC FOR CLARITY:

(If user can't **read** CFG rules, they'll struggle with:

- Parse Trees
- Top-Down Parsing
- Bottom-Up Parsing
- CYK Parsing
- PCFG)

Reading Context-Free Grammar (CFG) Rules

In the previous section, Context-Free Grammar (CFG) was introduced as a collection of grammar rules that describe how words and phrases combine to form a sentence.

The grammar rules may initially look unfamiliar.

For example,

```
S → NP VP
```

Another important question naturally arises.

How should these grammar rules be read?

Fortunately, reading CFG rules is much simpler than it appears.

Understanding the Arrow (→)

The symbol

→

is read as

consists of

or

can be expanded as

It does **not** mean "equals."

For example,

```
S → NP VP
```

is read as

A sentence consists of a noun phrase followed by a verb phrase.

Another example,

```
VP → Verb NP
```

is read as

A verb phrase consists of a verb followed by a noun phrase.

The arrow simply tells the NLP system **how a grammatical symbol can be expanded**.

Reading Common CFG Rules

The following table shows some common grammar rules and how they should be read.

CFG Rule	How to Read It
S → NP VP	A sentence consists of a noun phrase followed by a verb phrase.
NP → Determiner Noun	A noun phrase consists of a determiner followed by a noun.
NP → Proper Noun	A noun phrase can consist of a proper noun.
VP → Verb NP	A verb phrase consists of a verb followed by a noun phrase.
PP → Preposition NP	A prepositional phrase consists of a preposition followed by a noun phrase.

Notice that every rule can be read as a simple English sentence.

Understanding a Rule Step by Step

Consider the rule

```
NP → Determiner Noun
```

The rule has two parts.

Left Side

```
NP
```

This is the grammatical symbol that will be expanded.

Right Side

```
Determiner Noun
```

This tells the NLP system how the noun phrase is formed.

Therefore,

```
NP → Determiner Noun
```

means

A noun phrase consists of a determiner followed by a noun.

Example

Consider the phrase

the book

The words are

```
the     book
```

Their Parts of Speech are

```
Determiner     Noun
```

Applying the grammar rule

```
NP → Determiner Noun
```

gives

```
NP
↓
Determiner Noun
```

```
↓
the book
```

Therefore,

the book is recognised as a **Noun Phrase (NP).**

Another Example

Consider the sentence

David reads.

The grammar rule is

```
S → NP VP
```

The noun phrase is

```
NP
↓
David
```

The verb phrase is

```
VP
↓
reads
```

Combining them,

```
S
↓
NP VP
↓
David reads
```

The sentence has now been built using CFG rules.

Understanding Rule Expansion

Another important question naturally arises.

Why are these rules called production rules?

They are called **production rules** because each rule **produces** the next part of the sentence.

For example,

```
S
```

produces

```
NP VP
```

Then

```
NP
```

produces

```
Proper Noun
```

Finally,

```
Proper Noun
```

produces

```
David
```

The sentence is therefore built gradually, one rule at a time.

Visualising Rule Expansion

Consider the sentence

David reads a book.

The grammar expands in the following order.

```
S
↓
NP VP
↓
Proper Noun VP
↓
David VP
↓
David Verb NP
↓
David reads NP
↓
David reads Determiner Noun
↓
David reads a book
```

Notice that every step applies **one production rule**.

The NLP system continues expanding symbols until only words remain.

An Important Observation

Students often think that a CFG rule describes **one particular sentence**.

It does not.

A grammar rule describes a **pattern** that can generate many different sentences.

For example,

```
NP → Determiner Noun
```

can produce

- the book

- a letter
- the city
- a shepherd
- the king

The same rule is reused again and again.

This is one of the reasons CFG is such a powerful representation of grammar.

Key Points to Remember

- The symbol → is read as **consists of** or **can be expanded as**.
- A CFG rule explains **how one grammatical symbol is expanded**.
- The left side contains **one non-terminal symbol**.
- The right side contains the symbols that replace it.
- A single grammar rule can generate many different phrases and sentences.

Connecting the Concepts

The NLP system can now read and apply Context-Free Grammar rules.

However, another important question still remains.

Once the grammar rules are available, how does the NLP system use them to analyse the structure of an actual sentence?

For example, how does it identify that **"the book"** forms a noun phrase and **"reads a book"** forms a verb phrase?

To answer this question, the NLP system constructs a **parse tree**, showing how words combine into phrases and how those phrases combine to form a complete sentence.

This process is known as **Constituency Parsing**, which is the next topic.

Once the grammar rules are available, how does the NLP system actually analyse the structure of a sentence?

The answer is **Constituency Parsing**.

Constituency Parsing

In the previous topic, Context-Free Grammar (CFG) introduced the grammar rules used to describe the structure of a sentence.

For example,

```
S → NP VP

NP → Determiner Noun

VP → Verb NP
```

These rules explain **how a sentence can be formed**.

However, another important question naturally arises.

How does the NLP system use these rules to analyse an actual sentence?

Consider the sentence

David reads a book.

The NLP system already knows the Parts of Speech.

Word	POS Tag
David	Proper Noun
reads	Verb
a	Determiner
book	Noun

Using the CFG rules, the NLP system must now determine

- Which words form a phrase?
- Which phrase is the subject?
- Which phrase is the predicate?
- How are these phrases connected?

This process is called **Constituency Parsing**.

Understanding the Idea

When people read a sentence, they usually do not understand it **one word at a time**.

Instead, they naturally group words into meaningful units.

Consider the sentence

David reads a book.

Most people automatically recognise

```
David
```

as one unit,

and

```
reads a book
```

as another unit.

Within the second unit,

```
a book
```

is recognised as another smaller unit.

These meaningful groups are called **constituents**.

Constituency Parsing identifies these groups.

What is a Constituent?

A **constituent** is a group of words that functions as a single unit within a sentence.

For example,

in the sentence

David reads a book.

the following constituents can be identified.

```
David
```

↓

Noun Phrase (NP)

```
a book
```

↓

Noun Phrase (NP)

```
reads a book
```

↓

Verb Phrase (VP)

Each of these groups behaves as one grammatical unit.

Definition

Constituency Parsing is the process of dividing a sentence into its grammatical constituents using the rules of a Context-Free Grammar.

In simple words,

Constituency Parsing groups words into meaningful phrases and shows how those phrases together form a complete sentence.

How Does Constituency Parsing Work?

The NLP system follows three simple steps.

Step 1

Identify the Parts of Speech.

(This has already been completed.)

Step 2

Apply the CFG production rules.

Step 3

Group words into phrases.

The final result is called a **Parse Tree**.

Worked Example

Consider the sentence

Mary opened the door.

The POS tags are

Word	POS Tag
Mary	Proper Noun
opened	Verb
the	Determiner
door	Noun

The grammar rules are

```
S → NP VP

NP → Proper Noun
```

```
NP → Determiner Noun

VP → Verb NP
```

Now apply the rules.

First,

```
Mary
```

becomes

```
NP
```

Next,

```
the door
```

becomes

```
NP
```

Then,

```
opened the door
```

becomes

```
VP
```

Finally,

```
NP + VP
```

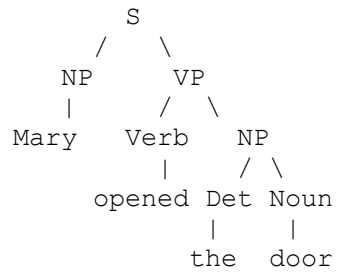
forms

```
S
```

The sentence has now been analysed.

Parse Tree

The result of Constituency Parsing is usually represented using a **Parse Tree**.



Notice that

- the sentence is divided into phrases,
- each phrase is divided into smaller grammatical units,
- the analysis continues until individual words are reached.

Why Is It Called a Parse Tree?

Another important question naturally arises.

Why is the result drawn as a tree?

It is called a **Parse Tree** because the sentence grows from one starting point and branches into smaller grammatical units, just like the branches of a tree.

The top of the tree represents the complete sentence.

The bottom of the tree contains the actual words.

Why Is Constituency Parsing Important?

Constituency Parsing helps the NLP system understand **how words are grouped together**.

This is important because meaning often depends on phrases rather than individual words.

For example,

consider the sentence

Joseph found the old book.

The words

`the old book`

form one **Noun Phrase**.

The adjective

`old`

describes **book**, not **Joseph**.

By identifying this phrase correctly, the NLP system avoids incorrect interpretations.

Applications

Constituency Parsing is widely used in:

- Machine Translation
- Question Answering Systems
- Grammar Checking
- Speech Recognition
- Information Extraction
- Text Summarization

In all these applications, understanding the structure of a sentence is essential before interpreting its meaning.

Connecting the Concepts

Constituency Parsing explains **how a sentence is analysed** using grammar rules.

However, another important question still remains.

Where do these grammar rules come from, and how are they tested on thousands of real sentences?

Researchers solve this problem by storing large collections of correctly parsed sentences.

These collections are called **Treebanks**.

A Treebank acts as a reference that helps linguists and NLP systems develop, evaluate, and improve parsing algorithms.

This is the next topic.

```
POS Tagging
        ↓
How are words tagged?

↓

HMM / Maximum Entropy
        ↓
How are the best tags chosen?

↓

Context-Free Grammar
        ↓
What grammar rules describe a sentence?

↓

Reading CFG Rules
        ↓
How are grammar rules interpreted?

↓

Constituency Parsing
        ↓
How are grammar rules applied?

↓

Treebanks (Next)
```

Now the next logical question is:

If parsing is so important, where do researchers get thousands of correctly parsed sentences to build and evaluate NLP systems?

This brings us to the next topic.

Treebanks

Constituency Parsing enables an NLP system to analyse the grammatical structure of a sentence and represent it as a parse tree.

For example, consider the sentence

David reads a book.

Using Context-Free Grammar, the NLP system constructs a parse tree that shows how the sentence is divided into phrases.

At this stage, another important question naturally arises.

How can researchers verify that the generated parse tree is correct?

Writing a parser is only one part of the task.

The parser must also be tested on thousands of sentences to measure its accuracy and improve its performance.

For this purpose, NLP uses **Treebanks**.

Understanding the Idea

Imagine a teacher preparing a grammar examination.

Before evaluating students' answers, the teacher needs an **answer key** that contains the correct solutions.

Similarly, an NLP researcher needs a collection of sentences with **correct parse trees**.

These correctly analysed sentences serve as a reference for training and evaluating parsers.

This collection is called a **Treebank**.

Definition

A **Treebank** is a collection of sentences that have already been analysed and represented as parse trees.

In simple words,

A Treebank is a database of correctly parsed sentences used to train, test, and improve NLP systems.

Why Are Treebanks Needed?

Suppose an NLP researcher develops a new parser.

How can the researcher determine whether the parser is working correctly?

One approach is to compare the parser's output with a sentence that has already been analysed by language experts.

If both parse trees are the same, the parser has performed correctly.

If they are different, the parser may require improvement.

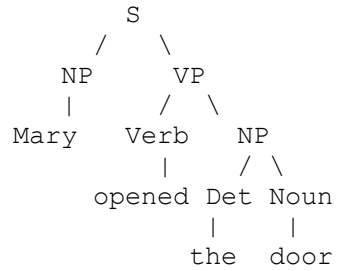
Treebanks make this comparison possible.

Example

Consider the sentence

Mary opened the door.

A Treebank may already contain its correct parse tree.



When another parser analyses the same sentence, its output can be compared with this stored tree.

If the structures match, the parser is considered accurate for that sentence.

How Are Treebanks Created?

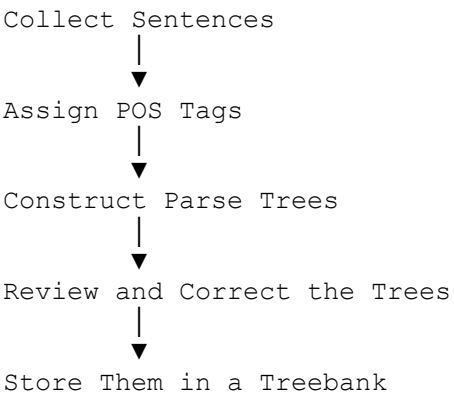
Another important question naturally arises.

Who creates Treebanks?

Treebanks are **not generated automatically**.

They are usually created by **linguists and NLP experts**.

The process generally involves the following steps.



Because every sentence is carefully checked, creating a Treebank is a time-consuming process.

However, the result is a highly reliable linguistic resource.

What Information Does a Treebank Store?

A Treebank usually contains:

- The original sentence.
- The Part-of-Speech tags.
- The parse tree.

- The grammatical relationships between words.

For example,

Sentence	POS Tags	Parse Tree
David reads a book.	NNP VBZ DT NN	Stored
Mary opened the door.	NNP VBD DT NN	Stored
Joseph will pray.	NNP MD VB	Stored

Thus, a Treebank stores much more than plain text.

It stores the complete grammatical analysis of each sentence.

Popular Treebanks

Several Treebanks are widely used in NLP research.

Treebank	Purpose
Penn Treebank	One of the most widely used English Treebanks for POS tagging and parsing research.
Universal Dependencies (UD) Treebanks	Multilingual treebanks used to study grammatical structures across many languages.

These resources have played an important role in the development of modern NLP systems.

Applications of Treebanks

Treebanks are used to:

- Train parsers.
- Evaluate parser accuracy.
- Develop new parsing algorithms.

- Improve POS tagging models.
- Support linguistic research.

Without Treebanks, it would be difficult to measure how well an NLP parser performs.

An Important Observation

A common misconception is that a Treebank is simply a collection of sentences.

This is not correct.

A normal text document contains only sentences.

A Treebank contains **sentences together with their grammatical analysis**, making it much more valuable for NLP research and system development.

Connecting the Concepts

Treebanks provide large collections of correctly parsed sentences.

However, another practical challenge still remains.

Different grammar rules can often produce the same sentence in different ways.

To simplify parsing and make algorithms more efficient, grammar rules are often converted into a standard form.

These standardized grammar representations are known as **Normal Forms for Grammar**, which is the next topic.

Excellent. We're now entering the parsing algorithms section of Unit II.

So far, we've learned:

```
CFG
   ↓
How grammar rules are written

↓
```

```
Constituency Parsing
   ↓
How grammar rules analyse a sentence

↓

Treebanks
   ↓
Where correctly parsed sentences are stored

↓

Normal Forms (Next)
```

"If the grammar rules already work, why should we change them?"

Normal Forms for Grammar

In the previous topic, Treebanks showed how grammar rules are used to analyse thousands of sentences.

However, another practical problem soon appeared.

Different linguists may write the same grammar in different ways.

For example, consider the following grammar rule.

```
S → NP VP PP
```

Another grammar may represent the same idea as

```
S → NP X

X → VP PP
```

Both grammars describe the same sentence structure.

However, they are **not written in the same format**.

This naturally raises another important question.

If every grammar is written differently, how can an NLP algorithm process them efficiently?

To solve this problem, grammars are converted into a **standardized form**.

This standardized representation is called a **Normal Form**.

Understanding the Idea

Imagine receiving documents from different people.

One person writes the date as

```
12/08/2026
```

Another writes

```
08-12-2026
```

Another writes

```
12 August 2026
```

All three represent the same date.

However, processing three different formats is inconvenient.

A computer system usually converts all dates into one standard format before processing them.

Grammar works in exactly the same way.

Although different grammar rules may represent the same language, NLP algorithms work more efficiently when all rules follow a common structure.

Definition

A **Normal Form** is a standardized representation of a Context-Free Grammar in which grammar rules follow specific patterns.

In simple words,

A Normal Form rewrites grammar rules into a standard structure so that parsing algorithms can process them more easily and efficiently.

Why Are Normal Forms Needed?

Most parsing algorithms expect grammar rules to follow a particular structure.

If the grammar contains many different rule formats,

the parser becomes more complicated.

By converting all grammar rules into a standard form,

- parsing becomes easier,
- algorithms become simpler,
- implementation becomes more efficient.

Example

Consider the grammar rule

```
S → NP VP PP
```

This rule contains three symbols on the right-hand side.

Some parsing algorithms cannot process such rules directly.

Therefore, the rule is rewritten.

```
S → NP X
```

```
X → VP PP
```

Notice something important.

The meaning has **not changed**.

Only the representation has changed.

This is the purpose of converting a grammar into a Normal Form.

Types of Normal Forms

Two Normal Forms are commonly used in Natural Language Processing.

Chomsky Normal Form (CNF)

Every production rule follows one of the following patterns.

```
A → BC
```

or

```
A → a
```

where

- **A, B and C** are non-terminal symbols.
- **a** is a terminal symbol.

In other words,

a rule may produce

- **two non-terminals**, or
- **one terminal**.

Greibach Normal Form (GNF)

Every production rule begins with a terminal symbol.

For example,

```
A → aBC
```

where

- **a** is a terminal,
- **B** and **C** are non-terminals.

GNF is mainly used in theoretical computer science and formal language studies.

In NLP, **Chomsky Normal Form** is much more commonly used because several parsing algorithms, including the **CYK Algorithm**, require grammars in this form.

Worked Example

Suppose the grammar contains

```
S → NP VP PP
```

This rule cannot be used directly in Chomsky Normal Form because it contains **three symbols** on the right-hand side.

Introduce a new non-terminal symbol.

```
X → VP PP
```

Now rewrite the original rule.

```
S → NP X
```

The grammar now satisfies the required structure while preserving the same meaning.

An Important Observation

Students often think that converting a grammar into a Normal Form changes the language.

This is **not true**.

The sentences generated by the grammar remain exactly the same.

Only the **representation of the grammar** changes.

The purpose is to make the grammar easier for parsing algorithms to process.

Key Points to Remember

- A Normal Form is a standardized representation of a grammar.
- It simplifies grammar rules without changing their meaning.
- Chomsky Normal Form (CNF) is the most commonly used Normal Form in NLP.
- Many parsing algorithms require the grammar to be converted into CNF before parsing begins.

Connecting the Concepts

The NLP system now has:

- grammar rules,
- parse trees,
- Treebanks,
- and a standardized grammar.

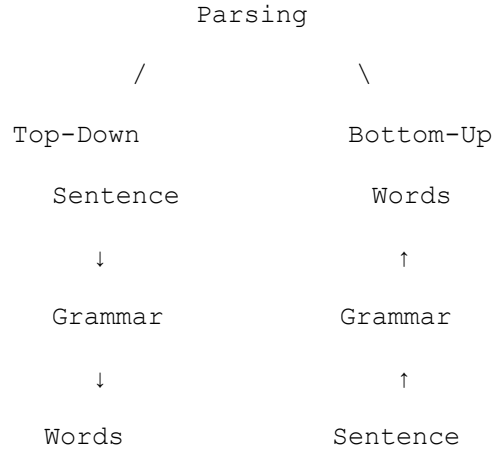
Another important question naturally arises.

Once the grammar has been converted into a suitable form, how does the NLP system actually begin parsing a sentence?

There are two fundamental parsing strategies.

- **Top-Down Parsing**, which starts from the sentence and works toward the words.
- **Bottom-Up Parsing**, which starts from the words and works toward the sentence.

These strategies form the foundation of several parsing algorithms and will be studied next.



Top-Down Parsing

In the previous topic, the grammar was converted into a standard form so that parsing algorithms could process it efficiently.

Another important question naturally arises.

Once the grammar is ready, where should the parsing process begin?

Should the NLP system begin with the complete sentence and gradually identify the words?

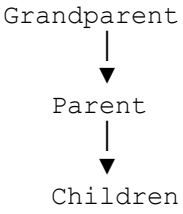
Or should it begin with the individual words and gradually build the sentence?

The first approach is called **Top-Down Parsing**.

Understanding the Idea

Imagine constructing a family tree.

One approach is to begin with the oldest ancestor and gradually move down to children and grandchildren.



This approach starts from the **top** and moves downward.

Top-Down Parsing follows the same idea.

It starts with the **Start Symbol (S)** and repeatedly applies grammar rules until the individual words of the sentence are produced.

Definition

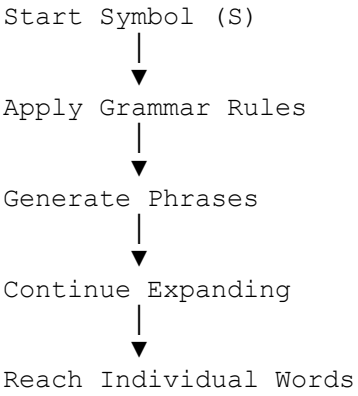
Top-Down Parsing is a parsing technique that begins with the **Start Symbol (S)** and repeatedly expands grammar rules until the input sentence is generated.

In simple words,

Top-Down Parsing starts with the complete sentence and gradually breaks it down into smaller grammatical units until the individual words are reached.

How Does Top-Down Parsing Work?

The NLP system follows these steps.



Worked Example

Consider the sentence

David reads a book.

Suppose the grammar is

```
S → NP VP

NP → Proper Noun

NP → Determiner Noun

VP → Verb NP

Proper Noun → David

Verb → reads

Determiner → a

Noun → book
```

The parser begins with the Start Symbol.

Step 1

```
S
```

Step 2

Apply

```
S → NP VP
```

Result

```
       S
      / \
    NP   VP
```

Step 3

Expand the Noun Phrase.

```
NP → Proper Noun
       S
      / \
    NP   VP
     |
Proper Noun
```

Step 4

Replace the Proper Noun.

```
Proper Noun → David
       S
      / \
   David  VP
```

Step 5

Expand the Verb Phrase.

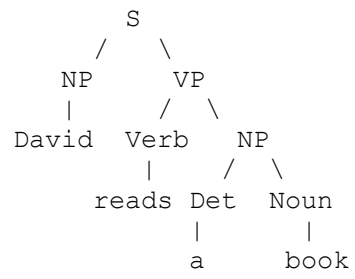
```
VP → Verb NP
          S
        /   \
    David     VP
             /  \
          Verb   NP
```

Step 6

Complete the remaining rules.

```
Verb → reads

NP → Determiner Noun

Determiner → a

Noun → book
```

Final parse tree



The parser has successfully generated the sentence by expanding grammar rules from the top.

Why Is It Called "Top-Down"?

The parser begins with the **highest grammatical symbol**, which is the **Start Symbol (S)**.

It then moves downward through the grammar until it reaches the words.

For this reason, the parsing process proceeds from **top to down**, giving the technique its name.

Advantages

Top-Down Parsing has several advantages.

- It follows the grammar in a logical order.
- It is easy to understand and implement.
- It clearly shows how a sentence is generated from grammar rules.
- It is useful for learning and explaining parse trees.

Limitations

Although Top-Down Parsing is simple, it also has some limitations.

Sometimes the parser expands grammar rules that do not match the actual input sentence.

As a result, it may spend time exploring incorrect paths before finding the correct one.

This makes the parsing process less efficient for complex grammars.

An Important Observation

A common misunderstanding is that Top-Down Parsing starts with the first word of the sentence.

It does **not**.

It always starts with the **Start Symbol (S)**.

The words appear only after the grammar rules have been expanded.

Summary

```
Start with

S

↓

Apply grammar rules

↓

Expand phrases

↓

Generate words

↓

Sentence analysed
```

Connecting the Concepts

Top-Down Parsing begins with the **Start Symbol** and works towards the words.

Another approach follows exactly the opposite strategy.

Instead of starting with the complete sentence, the parser begins with the **actual words** and gradually combines them into phrases until the **Start Symbol (S)** is reached.

This approach is called **Bottom-Up Parsing**, which will be studied next.

What if the parser starts with the words instead of the Start Symbol?

This is exactly what **Bottom-Up Parsing** does.

Bottom-Up Parsing

In the previous topic, Top-Down Parsing began with the **Start Symbol (S)** and gradually expanded grammar rules until the words of the sentence were generated.

Another important question naturally arises.

Can the parsing process begin with the actual words of the sentence instead?

The answer is **Yes**.

Instead of starting with the Start Symbol, the NLP system begins with the words, combines them into phrases, and gradually builds the complete sentence.

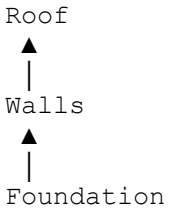
This approach is known as **Bottom-Up Parsing**.

Understanding the Idea

Imagine building a house.

Construction does not begin with the roof.

Instead, it starts with the foundation, followed by the walls, and finally the roof.



The construction moves **from the bottom to the top**.

Bottom-Up Parsing follows the same idea.

It starts with the **words** and gradually combines them into phrases until the complete sentence is formed.

Definition

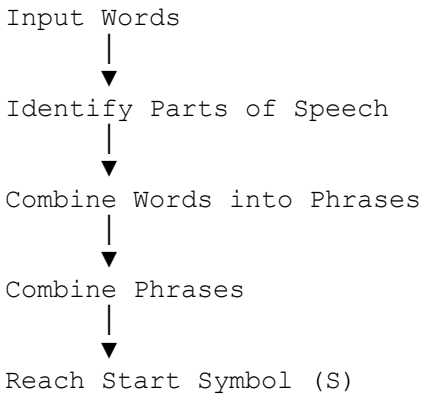
Bottom-Up Parsing is a parsing technique that begins with the words of a sentence and repeatedly combines them using grammar rules until the **Start Symbol (S)** is reached.

In simple words,

Bottom-Up Parsing starts with the words and gradually builds the complete sentence.

How Does Bottom-Up Parsing Work?

The NLP system follows these steps.



Unlike Top-Down Parsing, the process moves **upwards**, beginning with the input words.

Worked Example

Consider the sentence

David reads a book.

Suppose the grammar is

```
S → NP VP

NP → Proper Noun

NP → Determiner Noun

VP → Verb NP
```

Step 1

The parser starts with the words.

```
David    reads    a    book
```

Step 2

Identify the phrases.

```
David

↓

NP
```

```
a book

↓

NP
```

Step 3

Combine the verb and noun phrase.

```
reads + NP

↓

VP
```

Now the sentence becomes

```
NP    VP
```

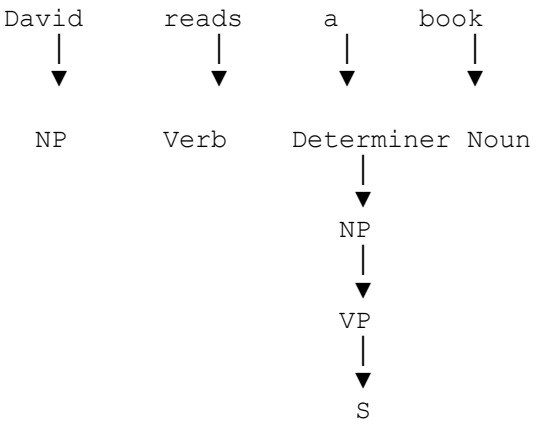
Step 4

Combine the remaining phrases.

```
NP + VP

↓

S
```

The parser has now constructed the complete sentence.

Visualising Bottom-Up Parsing



Notice that the parser starts with the words and gradually moves upward until the complete sentence is formed.

Why Is It Called "Bottom-Up"?

The parser begins with the **lowest level**, which contains the words of the sentence.

It then combines these words into phrases and finally reaches the **Start Symbol (S)**.

Since the process moves from the bottom towards the top, it is called **Bottom-Up Parsing**.

Advantages

Bottom-Up Parsing offers several advantages.

- It begins with the actual input sentence.
- Every step is based on words that already exist in the sentence.
- It avoids expanding grammar rules that are unrelated to the input.
- It is suitable for many efficient parsing algorithms used in NLP.

Limitations

Although Bottom-Up Parsing is effective, it may still encounter difficulties.

If several grammar rules can be applied at the same time, the parser must decide which one to choose first.

This may increase the complexity of the parsing process for ambiguous sentences.

Top-Down Parsing vs Bottom-Up Parsing

Feature	Top-Down Parsing	Bottom-Up Parsing
Starting Point	Start Symbol (S)	Input Words
Direction	Top → Bottom	Bottom → Top
First Step	Expand grammar rules	Combine words into phrases

Feature	Top-Down Parsing	Bottom-Up Parsing
Final Result	Words are generated	Start Symbol (S) is obtained

---

An Important Observation

Students sometimes think that one parsing strategy is always better than the other.

This is **not true**.

Both approaches have the same objective:

To analyse the grammatical structure of a sentence.

The difference lies only in **where the parsing process begins**.

- **Top-Down Parsing** begins with the grammar and works towards the words.
- **Bottom-Up Parsing** begins with the words and works towards the grammar.

Summary

```
Input Words

↓

Identify Phrases

↓

Build Larger Phrases

↓

Reach Start Symbol (S)

↓

Sentence Analysed
```

Connecting the Concepts

Top-Down Parsing and Bottom-Up Parsing describe **two different strategies** for analysing sentence structure.

However, another practical question still remains.

Can a parser analyse a sentence efficiently when the grammar contains many production rules and the sentence is long?

To solve this problem, NLP uses a dynamic programming algorithm called the **CYK (Cocke–Younger–Kasami) Parsing Algorithm**.

The CYK Algorithm combines the ideas of Context-Free Grammar, Chomsky Normal Form, and dynamic programming to analyse sentences efficiently.

This is the next topic.

CYK (Cocke–Younger–Kasami) Parsing Algorithm

Why is Another Parsing Algorithm Needed?

In the previous topics, two parsing strategies were studied.

- **Top-Down Parsing**
- **Bottom-Up Parsing**

Both approaches can successfully analyse the grammatical structure of a sentence.

However, another important question naturally arises.

Can these parsing methods analyse every sentence quickly and efficiently?

The answer is **not always**.

As the sentence becomes longer and the grammar contains more production rules, the number of possible parsing paths increases rapidly.

The parser may spend a significant amount of time exploring incorrect paths before finding the correct one.

Researchers therefore looked for a faster and more systematic parsing method.

This led to the development of the **CYK (Cocke–Younger–Kasami) Parsing Algorithm**.

Understanding the Idea

Imagine solving a large jigsaw puzzle.

One approach is to randomly try different pieces until the puzzle is complete.

This may eventually work, but it takes time.

A better approach is to begin with **small groups of pieces** and gradually combine them into larger sections until the entire picture is completed.

The CYK Algorithm follows the same idea.

Instead of analysing the whole sentence at once, it first analyses **small parts** of the sentence and then gradually combines them into larger phrases until the complete sentence is formed.

Definition

The **CYK (Cocke–Younger–Kasami) Algorithm** is a bottom-up parsing algorithm that uses **dynamic programming** to determine whether a sentence can be generated by a Context-Free Grammar written in **Chomsky Normal Form (CNF).**

In simple words,

The CYK Algorithm builds the sentence step by step by combining smaller parts until the complete sentence is formed.

Notice that the term **dynamic programming** was already introduced in **Unit I** while studying Minimum Edit Distance.

Here, the same idea is used again to avoid repeating unnecessary calculations.

Why Must the Grammar Be in Chomsky Normal Form?

Another important question naturally arises.

Why can't the CYK Algorithm use any Context-Free Grammar?

The CYK Algorithm follows a fixed procedure.

For this procedure to work correctly, every grammar rule must have a standard structure.

This is why the grammar is first converted into **Chomsky Normal Form (CNF)**.

In CNF, every production rule has one of the following forms.

```
A → BC
```

or

```
A → a
```

Because every rule follows the same structure, the CYK Algorithm can systematically combine smaller grammatical units into larger ones.

How Does the CYK Algorithm Work?

The CYK Algorithm follows these steps.

```
User Input
      |
      ▼
Tokenization
      |
      ▼
Convert Grammar to CNF
      |
      ▼
Create CYK Table
      |
      ▼
Fill the Table
      |
      ▼
Check the Top Cell
      |
```

```
Sentence Accepted or Rejected
```

Notice that the parser does **not** begin with a parse tree.

Instead, it builds a **table**.

Why Does CYK Use a Table?

Students often ask:

Why does the CYK Algorithm use a table instead of directly drawing a parse tree?

The answer is efficiency.

The table allows the parser to store intermediate results.

If the same word combination appears again, the parser simply reuses the stored result instead of calculating it again.

This is exactly the idea behind **dynamic programming**.

Example Sentence

To understand the algorithm, consider the sentence

David reads books

The words are

```
David
```

```
reads
```

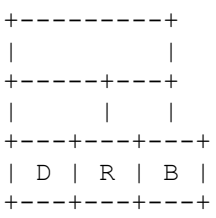
```
books
```

The grammar has already been converted into Chomsky Normal Form.

The parser now creates a CYK table.

CYK Table Structure

For a sentence containing three words, the table looks like this.



The **bottom row** contains the words of the sentence.

The remaining cells will gradually store larger grammatical structures.

At first glance, this table may look unfamiliar.

There is no need to worry.

Each row is filled **one step at a time**, and every cell has a specific purpose.

What Does Each Row Represent?

The table is filled **from the bottom upwards**.

Bottom Row

Each cell represents **one word**.

```
David

reads

books
```

Second Row

Each cell represents **two consecutive words**.

For example,

```
David reads
```

and

```
reads books
```

Top Row

The final cell represents

```
David reads books
```

If the **Start Symbol (S)** appears in this top cell, the sentence is accepted by the grammar.

Otherwise, it is rejected.

An Important Observation

Many students assume that the CYK table stores **words**.

This is **not** correct.

The table stores **grammar symbols** (such as **NP**, **VP**, and **S**) that can generate different parts of the sentence.

The words appear only in the bottom row.

Every other cell stores grammatical information.

Connecting the Concepts

The basic idea behind the CYK Algorithm is now clear.

The parser:

- creates a table,
- fills it from the bottom upwards,
- combines smaller grammatical units into larger ones,
- and finally checks whether the **Start Symbol (S)** appears at the top.

Another important question now arises.

How is each cell of the CYK table actually filled?

The next section will construct a complete CYK table for a sentence **step by step**, showing exactly how each value is calculated.

Constructing the CYK Table Step by Step

In the previous section, the CYK Algorithm was introduced as a bottom-up parsing algorithm that uses a table to analyse a sentence.

Another important question naturally arises.

How is the CYK table actually filled?

The table is **not filled randomly**.

Each cell is completed by applying the grammar rules in a systematic order.

To understand this process, let us analyse a simple sentence.

Example Sentence

Consider the sentence

John sees Mary

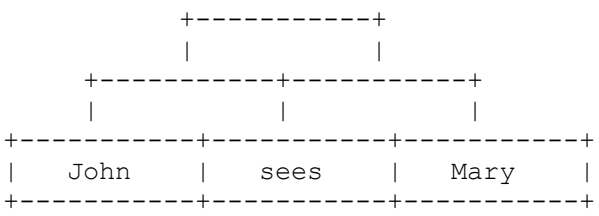
Suppose the grammar is already in **Chomsky Normal Form (CNF).**

```
S  → NP VP

VP → V NP

NP → John

NP → Mary

V  → sees
```

Step 1 – Draw the CYK Table

The sentence contains **three words**.

Therefore, we create a triangular table with three rows.



The **bottom row** always contains the words of the sentence.

Step 2 – Fill the Bottom Row

Look at each word separately.

Word 1

```
John
```

The grammar contains

```
NP → John
```

Therefore,

the first cell becomes

```
NP
```

Word 2

```
sees
```

The grammar contains

```
V → sees
```

Therefore,

the second cell becomes

```
V
```

Word 3

```
Mary
```

The grammar contains

```
NP → Mary
```

Therefore,

the third cell becomes

```
NP
```

The table now becomes

```
            +-----------+
            |           |
      +-----------+-----------+
      |           |           |
+-----------+-----------+-----------+
|    NP     |     V     |    NP     |
+-----------+-----------+-----------+
|   John    |   sees    |   Mary    |
+-----------+-----------+-----------+
```

Notice that **the words remain at the bottom**, while the row above them stores the corresponding grammar symbols.

Step 3 – Fill the Second Row

Now we combine **two neighbouring cells**.

First Combination

```
NP     V
```

Ask the question:

Does any grammar rule produce NP V?

Our grammar is

```
S → NP VP
```

```
VP → V NP
```

There is **no rule**

```
X → NP V
```

Therefore,

this cell remains **empty**.

Second Combination

Now combine

```
V      NP
```

Look through the grammar.

We find

```
VP → V NP
```

Therefore,

this cell becomes

```
VP
```

The table now becomes

```
            +-----------+
            |           |
     +------------+-----------+
     |            |    VP     |
+------------+-----------+-----------+
|    NP      |     V     |    NP     |
+------------+-----------+-----------+
|   John     |   sees    |   Mary    |
+------------+-----------+-----------+
```

Step 4 – Fill the Top Cell

Now consider the entire sentence.

We combine

```
NP      VP
```

Look through the grammar.

We find

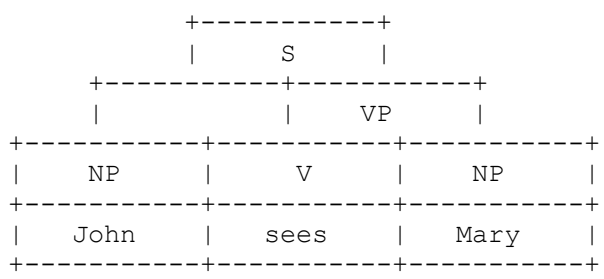
```
S → NP VP
```

Therefore,

the top cell becomes

```
S
```

The completed table is



How Do We Know the Sentence is Accepted?

The CYK Algorithm always checks **one place**.

It looks at the **top cell**.

If the top cell contains

```
S
```

the sentence **can be generated** by the grammar.

Therefore,

John sees Mary is **accepted**.

If the top cell does **not** contain **S**, the sentence is **rejected** because the grammar cannot generate it.

Rules to Remember

When filling a CYK table, always follow the same order.

Rule 1

Fill the **bottom row** first using rules of the form

```
A → word
```

Rule 2

Move upward one row at a time.

Never skip a row.

Rule 3

Each upper cell is obtained by combining the neighbouring cells below it.

Rule 4

Look for grammar rules that match the combination.

Example

```
V NP
```

matches

```
VP → V NP
```

Therefore,

write **VP** in the cell.

Rule 5

Finally, check the **top cell**.

If it contains **S**, the sentence is accepted.

Summary

```
Sentence

↓

Fill Bottom Row
(using terminal rules)

↓

Combine Adjacent Cells

↓

Apply Grammar Rules

↓

Continue Upwards

↓

Check Top Cell

↓

Top Cell = S ?

↓

Yes → Sentence Accepted

No → Sentence Rejected
```

An Important Observation

Students often think that **every cell must contain a symbol**.

This is **not true**.

Some cells may remain **empty** because no grammar rule matches the combination of symbols below them.

An empty cell simply means that **the grammar cannot form a valid phrase from that particular combination**.

Connecting the Concepts

The CYK table has now been constructed step by step.

Another important question naturally arises.

Can these steps be summarized into a systematic procedure that a computer can follow automatically?

The answer is **Yes**.

The complete sequence of steps is formally described by the **CYK Algorithm**, which will be studied next.

CYK Parsing Algorithm

In the previous section, a CYK table was constructed step by step to analyse a sentence.

The process followed while filling the table can now be summarized into a systematic algorithm.

This algorithm enables the NLP system to determine whether a sentence can be generated by a given Context-Free Grammar.

The CYK Algorithm

The CYK Algorithm follows these steps.

Step 1

Receive the input sentence.

Example:

John sees Mary

Step 2

Convert the grammar into **Chomsky Normal Form (CNF)** if it is not already in CNF.

This ensures that every grammar rule follows a standard format.

Step 3

Create a triangular CYK table.

The size of the table depends on the number of words in the sentence.

For a sentence containing **n** words, the table will contain **n rows**.

Step 4

Fill the bottom row.

For each word, find the grammar rule that directly produces that word.

For example,

```
NP → John
```

```
V → sees
```

```
NP → Mary
```

Write the corresponding grammar symbols above the words.

Step 5

Move upward through the table.

Combine neighbouring grammar symbols and search for grammar rules that match those combinations.

Whenever a matching rule is found, write the corresponding non-terminal symbol in the appropriate cell.

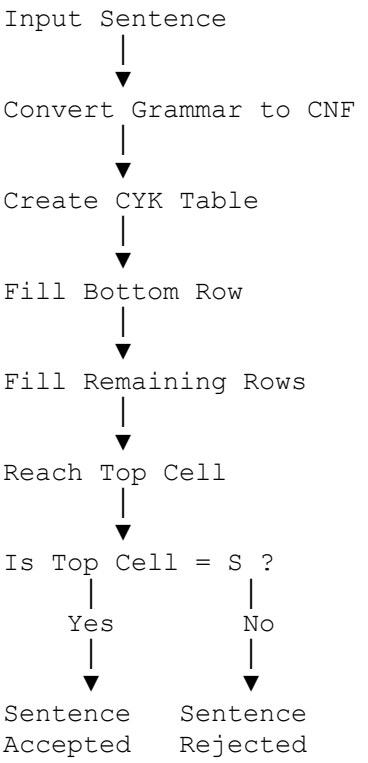
Continue this process until the top row is reached.

Step 6

Check the top cell.

- If the top cell contains the **Start Symbol (S)**, the sentence is accepted.
- If the top cell does not contain **S**, the sentence is rejected.

Algorithm Flow

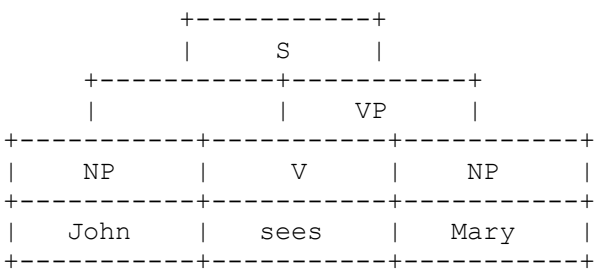


Worked Summary

Consider the sentence

John sees Mary

After applying the grammar rules, the completed CYK table becomes



The top cell contains

```
S
```

Therefore,

the sentence is **accepted** by the grammar.

Advantages of the CYK Algorithm

The CYK Algorithm offers several advantages.

- It provides a systematic method for parsing sentences.
- It uses **dynamic programming**, avoiding unnecessary repeated calculations.
- It is efficient for parsing grammars written in **Chomsky Normal Form**.
- It can determine whether a sentence belongs to a given grammar.
- It forms the foundation for several advanced parsing techniques.

Limitations of the CYK Algorithm

Although the CYK Algorithm is effective, it also has some limitations.

- It requires the grammar to be converted into **Chomsky Normal Form** before parsing.
- Constructing the parsing table requires additional memory.
- For very long sentences, the table becomes larger and the computation increases.
- It determines whether a sentence can be generated by the grammar, but the table alone does not directly explain the meaning of the sentence.

Key Points to Remember

- CYK is a **Bottom-Up Parsing Algorithm**.
- It uses **Dynamic Programming**.
- It works only with **Context-Free Grammars in Chomsky Normal Form (CNF)**.
- It fills a **triangular table** from the bottom upwards.
- The **top cell** determines whether the sentence is accepted or rejected.

CYK at a Glance

Feature	CYK Parsing Algorithm
Parsing Strategy	Bottom-Up
Grammar Required	Chomsky Normal Form (CNF)
Technique Used	Dynamic Programming
Output	Accepts or Rejects the sentence
Main Data Structure	Triangular Parsing Table

Connecting the Concepts

The CYK Algorithm determines **whether a sentence can be generated by a grammar**.

However, another important question still remains.

If more than one parse tree is possible for the same sentence, how does the NLP system decide which parse tree is the most likely?

Until now, every grammar rule has been treated as equally important.

In real language, however, some sentence structures occur much more frequently than others.

To represent this difference, probabilities are assigned to grammar rules.

This leads to the concept of **Probabilistic Context-Free Grammar (PCFG)**, which combines **Context-Free Grammar** with **probability** to identify the most likely parse tree.

Notice how naturally PCFG follows CYK.

- **CFG** answered: *What grammar rules can generate a sentence?*
- **CYK** answered: *Can this grammar generate the sentence?*
- **PCFG** answers: *If several parse trees are possible, which one is the most likely?*

This is exactly the question we'll answer.

Probabilistic Context-Free Grammar (PCFG)

In the previous topic, the CYK Algorithm determined whether a sentence could be generated by a Context-Free Grammar.

However, another important question naturally arises.

What happens if a sentence can be parsed in more than one way?

Consider the sentence

I saw the man with a telescope.

This sentence has two possible meanings.

Interpretation 1

The speaker used **a telescope** to see the man.

Interpretation 2

The man being seen **was carrying a telescope**.

Both interpretations are grammatically correct.

Both can be generated using Context-Free Grammar.

This creates a new challenge.

Which parse tree should the NLP system choose?

A normal Context-Free Grammar cannot answer this question because it treats every grammar rule as equally important.

To solve this problem, probabilities are assigned to grammar rules.

This approach is called **Probabilistic Context-Free Grammar (PCFG).**

Understanding the Idea

Imagine a road junction.

There are two roads leading to the same destination.

One road is used by thousands of people every day.

The other road is used only occasionally.

If someone asks which road is **more likely** to be chosen,

most people will select the road that is used more frequently.

The same idea applies to grammar.

If two different parse trees are possible,

the NLP system chooses the one that is **more likely** according to language usage.

Definition

A **Probabilistic Context-Free Grammar (PCFG)** is a Context-Free Grammar in which every production rule is assigned a probability.

In simple words,

A PCFG helps the NLP system choose the most likely parse tree by assigning probabilities to grammar rules.

Why Are Probabilities Needed?

Consider the following grammar.

```
S → NP VP

VP → Verb NP

VP → VP PP
```

Suppose both rules can be applied.

A normal CFG simply says

Both are possible.

It cannot tell which one is **more likely**.

A PCFG solves this by assigning probabilities.

Example

```
VP → Verb NP      (0.80)

VP → VP PP        (0.20)
```

Now the NLP system knows that

```
VP → Verb NP
```

is much more likely.

Understanding Rule Probabilities

Another important question naturally arises.

What does the probability of a grammar rule represent?

The probability shows **how often that rule is used** in real language.

For example,

```
NP → Determiner Noun      (0.75)

NP → Proper Noun          (0.25)
```

This indicates that, according to the training data,

a noun phrase consisting of a **Determiner followed by a Noun** occurs more frequently than a noun phrase consisting of a **Proper Noun**.

The NLP system therefore gives greater importance to the first rule.

Worked Example

Consider the sentence

The boy saw the girl with a telescope.

Suppose two parse trees are possible.

Parse Tree 1

The telescope belongs to the **action of seeing**.

Grammar probability

```
0.72
```

Parse Tree 2

The telescope belongs to **the girl**.

Grammar probability

```
0.28
```

Since

```
0.72 > 0.28
```

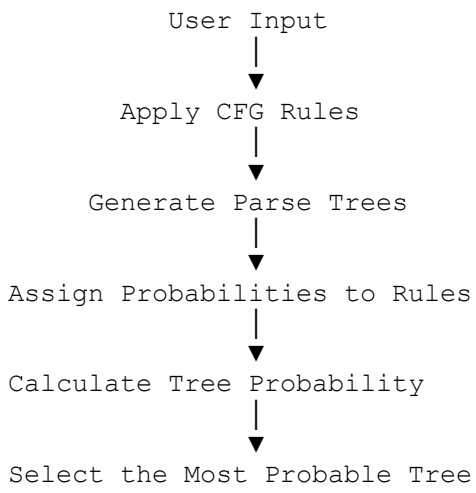
the NLP system chooses **Parse Tree 1**.

The parser does not simply choose a valid tree.

It chooses the **most probable** valid tree.

How Does a PCFG Work?

The overall process is shown below.



CFG vs PCFG

Context-Free Grammar (CFG)	Probabilistic Context-Free Grammar (PCFG)
Uses grammar rules	Uses grammar rules with probabilities
Checks whether a sentence is valid	Chooses the most likely valid parse tree
All rules are treated equally	Rules have different probabilities
May produce multiple parse trees	Selects the most probable parse tree

---

Advantages

PCFG provides several advantages over a normal Context-Free Grammar.

- It resolves ambiguity more effectively.
- It selects the most likely parse tree instead of choosing arbitrarily.
- It reflects real language usage by considering probabilities.
- It improves the accuracy of syntactic parsing.

Limitations

Although PCFG is more powerful than a normal CFG, it also has some limitations.

It requires a large collection of parsed sentences to estimate reliable rule probabilities.

In addition, it mainly relies on grammar probabilities and may not capture every aspect of meaning or context.

An Important Observation

Students often think that PCFG is a completely different grammar.

It is **not**.

A PCFG is simply a **Context-Free Grammar with probabilities attached to its production rules**.

The grammar remains the same.

Only additional probability information is added.

Summary

```
Context-Free Grammar

↓

Generate Parse Trees

↓

Assign Probabilities

↓

Calculate Tree Probability

↓

Choose the Most Probable Parse Tree
```

Connecting the Concepts

So far, the NLP system has learned how to identify words, assign Parts of Speech, construct parse trees, and choose the most probable sentence structure.

However, another important challenge still remains.

Grammar rules alone cannot represent every detail of a sentence.

For example, consider the sentence

These books are interesting.

The words **These** and **books** agree in **number**.

Similarly, **books** and **are** also agree in **number**.

How can the NLP system represent grammatical properties such as **number, gender, person, tense, or case**?

To solve this problem, NLP introduces **Feature Structures**, which allow additional grammatical information to be stored along with grammar rules.

Until now, the NLP system has focused on:

- words,
- Parts of Speech,
- grammar rules,
- parse trees.

Now another important question naturally arises.

Is knowing the Part of Speech of a word enough to fully describe it?

The answer is **No**.

Knowing that **books** is a noun is useful.

But it does not tell us that it is **plural**.

Similarly, knowing that **went** is a verb does not tell us that it is in the **past tense**.

To represent these additional grammatical details, NLP uses **Feature Structures**.

Feature Structures

So far, grammatical analysis has mainly focused on identifying the Part of Speech of each word and analysing sentence structure.

However, another important question naturally arises.

Can the Part of Speech alone describe every grammatical property of a word?

Consider the following two sentences.

This book is interesting.

These books are interesting.

In both sentences,

- **book** and **books** are nouns.

However, there is an important difference.

The first noun is **singular**.

The second noun is **plural**.

Similarly,

- **is** is a singular verb.
- **are** is a plural verb.

A normal Context-Free Grammar simply identifies these words as **Nouns** and **Verbs**.

It does **not** store additional information such as:

- Number
- Person
- Gender
- Tense
- Case

To represent these grammatical properties, NLP uses **Feature Structures**.

Understanding the Idea

Imagine a school identity card.

Knowing only a student's name is usually not enough.

The identity card also contains additional information such as:

- Roll Number
- Class
- Section
- Date of Birth

All these details together provide a more complete description of the student.

A Feature Structure works in exactly the same way.

Instead of storing only the Part of Speech,

it also stores additional grammatical information about the word.

Definition

A **Feature Structure** is a collection of grammatical features that describe the properties of a word or phrase.

In simple words,

A Feature Structure stores additional grammatical information about a word beyond its Part of Speech.

What is a Feature?

Another important question naturally arises.

What is meant by a feature?

A **feature** is simply a grammatical property of a word.

Some common features are shown below.

Feature	Example Values
Number	Singular, Plural
Person	First, Second, Third
Gender	Masculine, Feminine, Neuter
Tense	Present, Past, Future
Case	Nominative, Accusative

Each feature provides extra information that helps the NLP system understand the sentence more accurately.

Representing a Feature Structure

A Feature Structure is usually written as a set of **feature–value pairs**.

Consider the word

books

Its Feature Structure can be written as

```
Noun
Number = Plural
```

Similarly,

for the word

book

```
Noun
Number = Singular
```

Notice that both words are nouns.

The Feature Structure tells the NLP system **how they differ**.

Another Example

Consider the verb

went

The POS tag is

```
Verb
```

The Feature Structure provides additional information.

```
Verb

Tense = Past
```

Similarly,

goes

can be represented as

```
Verb

Tense = Present

Person = Third
```

The Feature Structure therefore gives a much richer description of the word.

Worked Example

Consider the sentence

These books are useful.

The NLP system first performs POS Tagging.

Word	POS Tag
These	Determiner
books	Noun
are	Verb
useful	Adjective

Next, it stores the Feature Structures.

Word	Feature Structure
These	Number = Plural
books	Number = Plural
are	Number = Plural, Tense = Present
useful	Adjective

The NLP system can now recognise that the determiner, noun and verb all agree in **number**.

Why Are Feature Structures Important?

Feature Structures provide information that ordinary grammar rules cannot represent.

They help the NLP system:

- distinguish singular and plural forms,
- identify verb tense,
- represent grammatical agreement,
- store detailed grammatical information,
- improve sentence analysis.

Without Feature Structures, many grammatical relationships would be difficult to represent accurately.

An Important Observation

Students often think that a Feature Structure **replaces** the Part of Speech.

This is **not true**.

The Part of Speech is still identified first.

The Feature Structure simply adds more grammatical information.

For example,

```
Word

↓

Part of Speech

↓

Feature Structure
```

The Feature Structure builds upon the POS tag; it does not replace it.

Summary

```
Word

↓

Identify POS

↓

Store Features

(Number, Gender,
Person, Tense, Case)

↓

Complete Grammatical Information
```

Key Points to Remember

- A Feature Structure stores **additional grammatical information**.
- It uses **feature–value pairs**.
- It supplements the Part of Speech; it does not replace it.
- Common features include **Number, Person, Gender, Tense, and Case**.
- Feature Structures help the NLP system represent grammatical agreement more accurately.

Connecting the Concepts

Feature Structures allow the NLP system to store detailed grammatical information.

However, another important question still remains.

How does the NLP system check whether the grammatical features of different words are compatible with one another?

For example,

consider the sentence

These books are useful.

Both **These** and **books** are **plural**.

Similarly, **books** and **are** also agree in **number**.

The NLP system must verify that these features are compatible before accepting the sentence as grammatically correct.

This process is called **Unification**, the final topic of Unit II.

Unification

In the previous topic, Feature Structures were introduced to store additional grammatical information about words, such as:

- Number
- Person
- Gender
- Tense

- Case

However, another important question naturally arises.

Is storing grammatical features alone enough?

The answer is **No**.

The NLP system must also verify whether the grammatical features of different words are **compatible** with each other.

Consider the following two sentences.

This book is interesting.

These book are interesting.

Both sentences contain valid English words.

However, the second sentence is grammatically incorrect.

Why?

Because:

- **These** is plural.
- **book** is singular.

Their grammatical features do not match.

The NLP system must detect this mismatch before accepting the sentence.

This process is called **Unification**.

Understanding the Idea

Imagine connecting two pieces of a jigsaw puzzle.

The pieces fit together only if their shapes match.

If the shapes are different,

they cannot be joined.

Grammar works in the same way.

Two words can be combined only if their grammatical features are compatible.

Unification performs this compatibility check.

Definition

Unification is the process of comparing Feature Structures to determine whether their grammatical features are compatible.

In simple words,

Unification checks whether the grammatical features of different words agree with one another.

How Does Unification Work?

The NLP system compares the Feature Structures of related words.

If the features match,

the words can be combined.

If the features conflict,

the sentence is considered grammatically incorrect.

Example 1 – Successful Unification

Consider the sentence

These books are useful.

The Feature Structures are

Word Feature Structure

Word Feature Structure
These Number = Plural
books Number = Plural
are Number = Plural

Now compare the features.

```
Plural
=
Plural
=
Plural
```

All the grammatical features agree.

Therefore,

the Feature Structures **unify successfully**.

The sentence is grammatically correct.

Example 2 – Unsuccessful Unification

Now consider

These book are useful.

The Feature Structures become

Word Feature Structure
These Number = Plural
book Number = Singular
are Number = Plural

Comparing the features,

```
Plural
≠
```

```
Singular
```

The Feature Structures do **not** match.

Therefore,

unification fails.

The sentence is grammatically incorrect.

Another Example

Consider the sentence

She writes well.

Feature Structures

Word Feature Structure
She Person = Third
writes Person = Third

Since both words have compatible grammatical features,

the Feature Structures unify successfully.

Now consider

She write well.

Here,

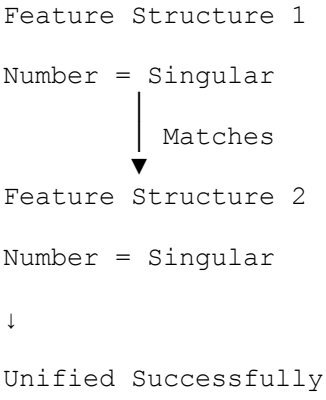
the grammatical features no longer agree.

Therefore,

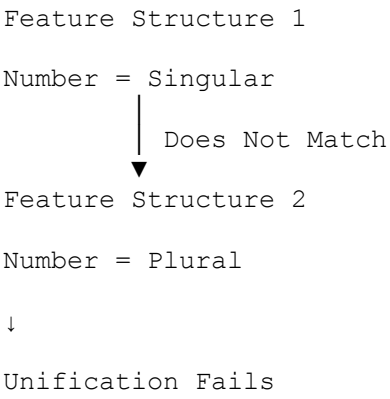
unification fails.

Visualising Unification

Successful Unification



Failed Unification



Why Is Unification Important?

Unification helps the NLP system ensure that grammatical relationships are correct.

It is commonly used to verify:

- Subject–Verb Agreement
- Singular–Plural Agreement
- Person Agreement
- Gender Agreement
- Tense Consistency

Without Unification, the NLP system might accept grammatically incorrect sentences simply because every individual word is valid.

Applications

Unification is used in:

- Grammar Checking Systems
- Machine Translation
- Dialogue Systems
- Information Extraction
- Syntactic and Semantic Analysis

It helps improve the grammatical accuracy of NLP applications.

Feature Structures vs Unification

Feature Structures	Unification
Store grammatical information	Compare grammatical information
Represent features such as number, tense, and person	Check whether the features are compatible
Describe words	Validate relationships between words

An Important Observation

Students often think that Feature Structures and Unification are the same concept.

They are **closely related**, but they perform different tasks.

Think of them this way:

- **Feature Structures** answer:

  "What grammatical information does this word have?"

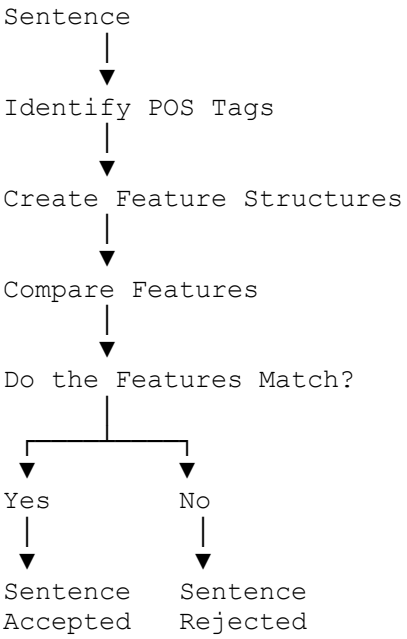
- **Unification** answers:

  "Do the grammatical features of these words agree?"

Feature Structures **store** the information.

Unification **checks** the information.

Summary



Key Points to Remember

- Unification compares **Feature Structures**.
- It checks grammatical agreement between related words.
- Matching features result in successful unification.
- Conflicting features result in failed unification.
- It is widely used in grammar checking and sentence analysis.

Connecting the Concepts

With Unification, the study of **Word-Level and Syntactic Analysis** is complete.

In this unit, the NLP system learned how to:

- assign Parts of Speech to words,
- resolve ambiguity using statistical models,
- represent sentence structure through grammar,
- analyse sentences using parsing techniques,

- select the most probable parse tree,
- store grammatical features,
- and verify grammatical agreement.

These techniques help the NLP system understand **how a sentence is structured**.

However, another important question still remains.

Even after identifying the correct sentence structure, does the NLP system truly understand what the sentence means?

Understanding the **meaning** of words, phrases, and sentences is the focus of the next unit:

Unit III – Semantic Analysis

In Unit III, the focus shifts from **"How is the sentence constructed?"** to **"What does the sentence mean?"**

```
POS Tagging
      ↓
Rule-Based POS Tagging
      ↓
Stochastic POS Tagging
      ↓
Transformation-Based POS Tagging
      ↓
Hidden Markov Model (HMM)
      ↓
Maximum Entropy Model
      ↓
Context-Free Grammar (CFG)
      ↓
Reading CFG Rules
      ↓
Constituency Parsing
      ↓
Treebanks
      ↓
Normal Forms
      ↓
Top-Down Parsing
      ↓
Bottom-Up Parsing
      ↓
CYK Parsing
      ↓
Probabilistic CFG (PCFG)
      ↓
Feature Structures
      ↓
```

Unification

↓

Semantic Analysis (Unit III)