SRM INSTITUTE OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTING TECHNOLOGIES

21CSE356T

NATURAL LANGUAGE PROCESSING

UNIT- 2

S.PRABU
Assistant Professor
C-TECH-SRM-IST-KTR

UNIT-2

Context Free Grammars, Grammar Rules for English, Top-Down Parsing, Bottom-Up Parsing, Ambiguity, CKY Parsing, Dependency Parsing, Earley Parsing - Probabilistic Context-Free Grammars

Topic 1: Context-Free Grammar (CFG)

A Context-Free Grammar (CFG) is a set of rules used to generate sentences in a language. It helps define the structure of valid sentences in programming languages and natural languages.

Why Do We Need CFG?

- Helps in parsing (understanding) programming languages.
- Useful for natural language processing (NLP) to analyze sentence structures.
- Defines rules for syntactic correctness in language processing.

Components of a CFG

A CFG consists of four main components:

1. **Terminals (T)**: The basic symbols (words or characters) that appear in the final output.
    - Example: {a, b, 0, 1, if, else} in a programming language.
2. **Non-terminals (N)**: Symbols that represent groups of words or structures in the language.
    - Example: {Sentence (S), Noun Phrase (NP), Verb Phrase (VP)} in English.
3. **Production Rules (P)**: Rules that define how non-terminals can be replaced with terminals and other non-terminals.
    - Example: S → NP VP (A sentence consists of a noun phrase and a verb phrase).
4. **Start Symbol (S)**: The symbol where the derivation starts.
    - Example: S (Sentence).

Example of CFG

Consider a simple grammar for basic English sentences:

```
S → NP VP
NP → N | Det N
VP → V | V NP
N → "cat" | "dog"
V → "eats" | "chases"
Det → "the" | "a"
```

Sentence Generation Using CFG

Let's generate a sentence step by step:

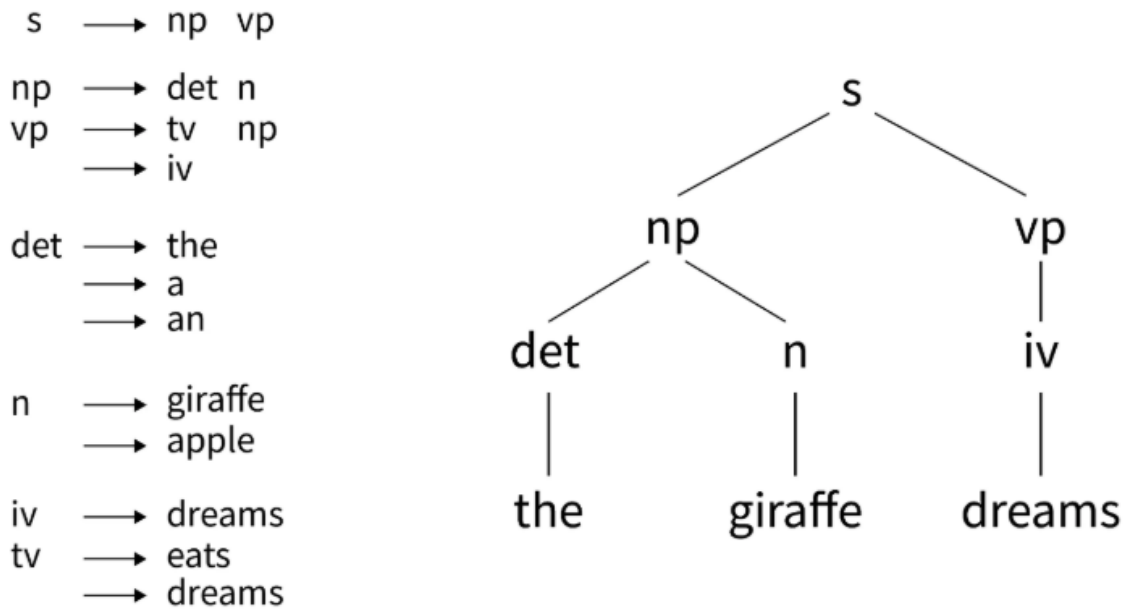
1. Start with S:
    - S → NP VP
2. Replace NP:
    - NP → Det N → "the cat"
3. Replace VP:
    - VP → V NP → "chases the dog"
4. Final sentence:
    - "the cat chases the dog"

Context-free grammar (CFG) can also be seen as the list of rules that define the set of all well-formed sentences in a language. Each rule has a left-hand side that identifies a syntactic category and a right-hand side that defines its alternative parts reading from left to right. - **Example:** The rule s --> np vp means that "a sentence is defined as a noun phrase followed by a verb phrase."

- **Formalism in rules for context-free grammar:** A sentence in the language defined by a CFG is a series of words that can be derived by systematically applying the rules, beginning with a rule that has s on its left-hand side.

- Use of parse tree in context-free grammar: A convenient way to describe a parse is to show its parse tree, simply a graphical display of the parse.
- A parse of the sentence is a series of rule applications in which a syntactic category is replaced by the right-hand side of a rule that has that category on its left-hand side, and the final rule application yields the sentence itself.

- **Example:** A parse of the sentence "the giraffe dreams" is: s => np vp => det n vp => the n vp => the giraffe vp => the giraffe iv => the giraffe dreams



- If we look at the example parse tree for the sample sentence in the illustration the giraffe dreams, the graphical illustration shows the parse tree for the sentence
- We can see that the root of every subtree has a grammatical category that appears on the left-hand side of a rule, and the children of that root are identical to the elements on the right-hand side of that rule.

Topic 2 Grammar Rules for English

Natural Language Processing (NLP) involves analyzing and understanding human language. Grammar rules help NLP models process text efficiently.

1. Parts of Speech (POS) Tagging

In NLP, words are categorized into parts of speech using **POS tagging** to understand their role in a sentence.

In NLP, words are categorized into parts of speech using **POS tagging** to understand their role in a sentence.

Part of Speech	Example
Noun (NN)	cat, book, city
Pronoun (PRP)	he, she, they
Verb (VB)	run, eat, is
Adjective (JJ)	beautiful, fast
Adverb (RB)	quickly, well
Preposition (IN)	in, on, under
Conjunction (CC)	and, but, because
Determiner (DT)	the, a, an

NLP Applications:

POS tagging helps in

- text analysis,
- machine translation, and
- speech recognition.

2. Sentence Structure & Parsing

Syntax Analysis (Parsing)

- NLP parsers analyze sentence structures using Context-Free Grammar (CFG).
- Example CFG rule for a sentence (S):

```
S → NP VP
NP → Det N
VP → V NP
```

Example Sentence: "The cat eats fish."

- S → NP VP
- NP → Det N ("The cat")
- VP → V NP ("eats fish")

Word Relationships in NLP

Dependency Parsing

- Identifies **word relationships** in a sentence.
- Example:
  - "The boy plays football."
  - "boy" → **subject** of "plays"
  - "football" → **object** of "plays"

Grammar-Based NLP Applications

POS Tagging	→	Identifies word categories
Parsing	→	Analyzes sentence structure
Named Entity Recognition (NER)	→	Finds names, places, dates
Machine Translation	→	Uses grammar rules to translate text
Chatbots & Virtual Assistants	→	Understands user input

Topic 3 Top-Down Parsing

Top-down parsing is a **syntax analysis technique** in NLP that starts from the **start symbol (S)** of a grammar and breaks it down into smaller parts (non-terminals and terminals) until it matches the input sentence.

Start from the **root (S)** and expand step by step to derive the input sentence.

Steps in Top-Down Parsing

1. Start with the Start Symbol (S).
2. Apply Production Rules (from grammar).
3. Expand Until Terminals Match Input.
4. If No Match, Backtrack & Try Another Rule.

Example CFG (Context-Free Grammar):

```
S → NP VP
NP → Det N
VP → V NP
Det → "the"
N → "cat" | "dog"
V → "chases" | "eats"
```

Parsing "The cat chases the dog" using Top-Down Approach:

1. Start with **S → NP VP**
2. Expand **NP → Det N** → "the cat"
3. Expand **VP → V NP** → "chases the dog"
4. Matches input!

Types of Top-Down Parsing

1. Recursive Descent Parsing

- Uses recursive functions to process each rule.
- Simple but suffers from backtracking issues.

2. Predictive Parsing (LL Parsing)

- Eliminates backtracking by using lookahead symbols.
- Example: LL(1) Parsing, where "1" means it looks one token ahead.

Problems in Top-Down Parsing

Backtracking Issue : If a rule fails, it must go back and try another.
Left Recursion : If a rule calls itself first, it causes an infinite loop.

- Example: S → S VP (Bad, causes infinite recursion).
- Solution: **Convert left recursion to right recursion**.

Applications of Top-Down Parsing in NLP

Natural Language Understanding (NLU) – Helps machines understand text.
Compiler Design – Used in programming language parsers.
Chatbots & Virtual Assistants – Helps process user queries.

Using Top-Down technique, parser searches for a parse tree by trying to build from the root node S down to the leaves. The algorithm starts by assuming the input can be derived by the designated start symbol S. The next step is to find the tops of all the trees which can start with S, by looking on the grammar rules with S on left hand side, all the possible trees are generated . Top-Down parsing is viewed as generation of parse tree in preorder.

Let's consider the Grammar rules

S → NP V S → NP AuxV VP

S → VP NP → Proper-Noun

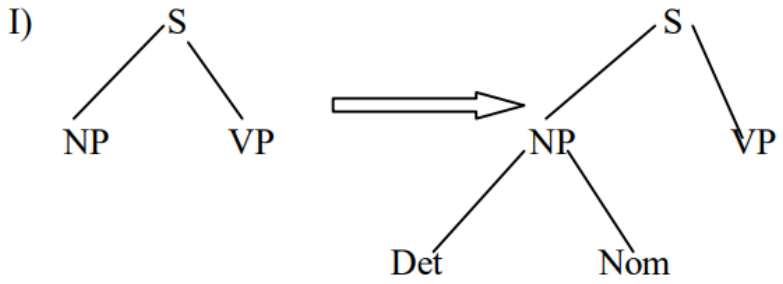
NP → Det Nominal Nominal → Noun

Nominal → Noun Nominal

VP → Verb VP → V NP

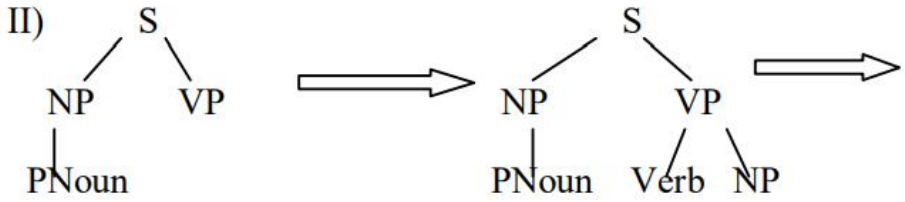
Taking the sentence: "John is playing game", and applying Top-down parsing

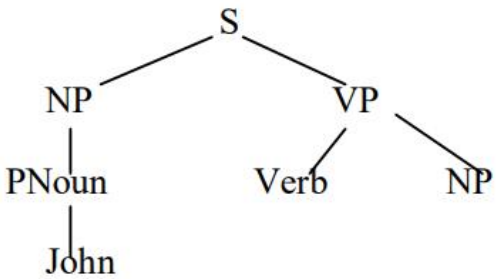
I)



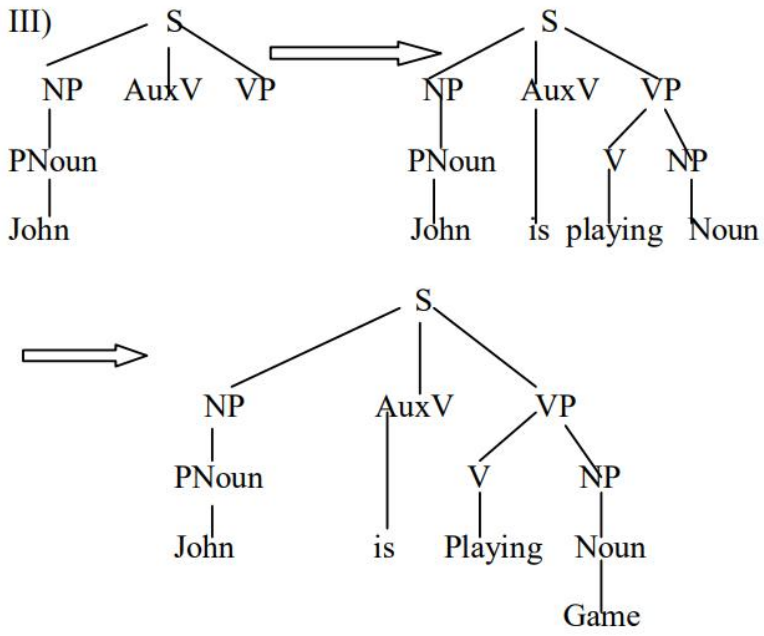
Part-of-speech does not match the input string, backtrack to the node NP

II)





Part-of-speech Verb does not match the input string, backtrack to the node S, since PNoun is matched.



Final parse tree for the sentence

The advantage of Top-Down strategy is that it never wastes time exploring trees that cannot result in S, means it also never explores subtrees that cannot find a place in some S- rooted tree

Topic 4 Bottom-up parsing

Bottom-up parsing is a syntax analysis technique where parsing starts from the input (terminals) and moves upwards to form the start symbol (S) by applying grammar rules.

Start with the words in the sentence and gradually combine them using production rules until we reach S.

Steps in Bottom-Up Parsing

1. Read the input sentence (terminals).
2. Find grammar rules that match parts of the input.
3. Replace matched parts with their corresponding non-terminals.
4. Repeat until we reach the start symbol (S).

Example CFG (Context-Free Grammar):

```
S → NP VP
NP → Det N
VP → V NP
Det → "the"
N → "cat" | "dog"
V → "chases" | "eats"
```

Parsing "The cat chases the dog" using Bottom-Up Approach:

1. Identify **terminals**: "The cat chases the dog"
2. Apply **NP → Det N** → "The cat"
3. Apply **NP → Det N** → "The dog"
4. Apply **VP → V NP** → "chases the dog"
5. Apply **S → NP VP** → "The cat chases the dog"
6. Successfully reached **S**

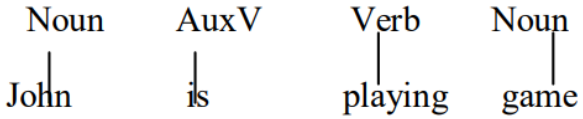
4. Problems in Bottom-Up Parsing

Conflicts: Shift-Reduce and Reduce-Reduce conflicts.
Slow for large grammars: Can be inefficient.

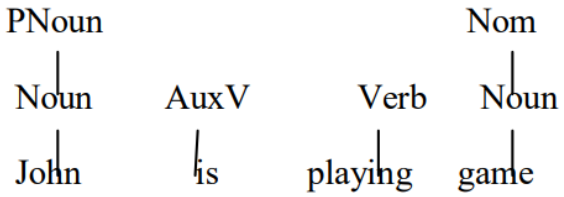
Bottom – Up parsing starts with the words of input and tries to build trees from the words up, again by applying rules from the grammar one at a time. The parse is successful if the parse succeeds in building a tree rooted in the start symbol S that covers all of the input. Bottom-Up parsing can be viewed as generation of parse tree in post order.

Considering the grammar rules in Figure and the input sentence “John is playing game”. The Bottom-Up parsing works as follows:

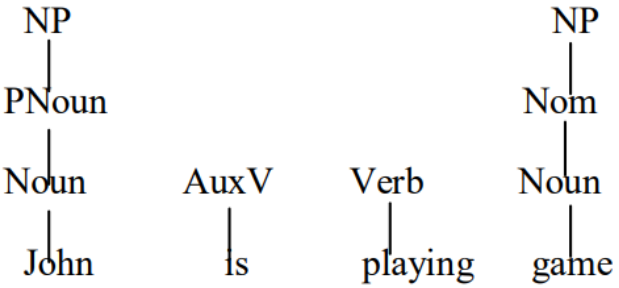
I)


Noun
AuxV
Verb
Noun
John
is
playing
game

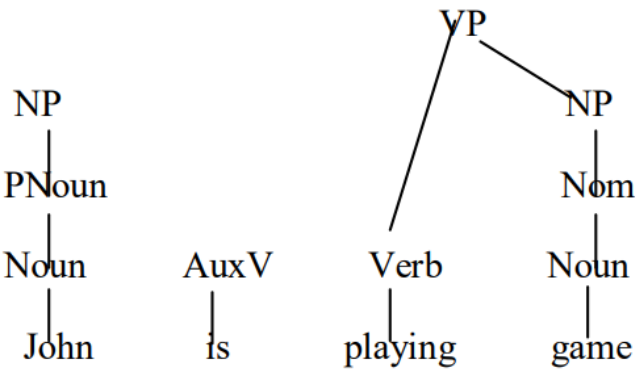
II)


PNoun
Nom
Noun
AuxV
Verb
Noun
John
is
playing
game

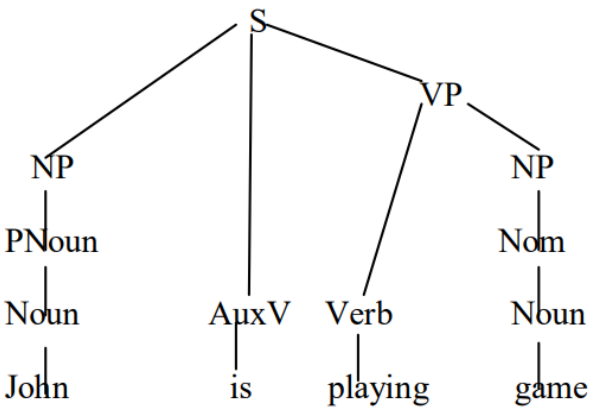
III)


NP
NP
PNoun
Nom
Noun
AuxV
Verb
Noun
John
is
playing
game

IV)


VP
NP
NP
PNoun
Nom
Noun
AuxV
Verb
Noun
John
is
playing
game

V)


S
VP
NP
NP
PNoun
Nom
Noun
AuxV
Verb
Noun
John
is
playing
game

Bottom-Up parsing adopts the shift-reduce paradigm, where a stack holds the grammar symbol and input buffer stores the rest of the input sentence. The Shift-reduce parsing is achieved using four primary actions (1)Shift, pushes the next input symbol on the top of the stack (2)Reduce, reduces Right Hand Side of the production to its Left Hand Side Accept, successful completion of parsing and(4) Error, discover the syntax error and call the error recovery routine.

Topic 5 Ambiguity

Ambiguity in Natural Language Processing (NLP) occurs when a sentence, phrase, or word has **multiple meanings**, making it difficult for machines to interpret correctly.

Example:

- **"I saw the man with a telescope."**
    - Did **I** use the telescope to see the man?
    - Or did the **man** have a telescope?

Types of Ambiguity in NLP

1. Lexical Ambiguity (Word Meaning Confusion)

- A single word has **multiple meanings**.
- Example:
    - **"The bank is closed."**
        - **Bank** (financial institution) or **Bank** (side of a river)?

2. Syntactic Ambiguity (Sentence Structure Confusion)

- A sentence has **multiple valid structures**.
- Example:
    - **"Old men and women were seated."**
        - Are only **old men** seated, or **old men and women**?

3. Semantic Ambiguity (Meaning Interpretation Confusion)

- The **meaning of the sentence is unclear**.

- Example:
    - **"She cannot bear children."**
        - Does she dislike children?
        - Or is she unable to have children?

4. Pragmatic Ambiguity (Context-Dependent Meaning)

- The meaning depends on the **context of conversation**.
- Example:
    - **"Can you pass the salt?"**
        - Is it a **request** (asking to pass salt)?
        - Or a **question** about ability?

5. Anaphoric Ambiguity (Pronoun Reference Confusion)

- **Pronouns** (he, she, it, they) refer to multiple nouns.
- Example:
    - **"John told Mike he won."**
        - Who won? **John** or **Mike**?

How to Resolve Ambiguity in NLP?

Part-of-Speech (POS) Tagging – Helps identify the correct meaning of words.

Syntax Parsing – Identifies sentence structures.

Word Sense Disambiguation (WSD) – Determines the correct meaning of a word.

Machine Learning & Deep Learning – Uses models like BERT, GPT for better understanding.

Context Analysis – Checks previous sentences to resolve references.

Topic 6 CKY Parsing

The Cocke-Kasami-Younger (CKY) algorithm is a bottom-up parsing technique used in Natural Language Processing (NLP) for syntactic analysis of sentences. It checks whether a given sentence can be generated using a Context-Free Grammar (CFG) in Chomsky Normal Form (CNF).

CKY Parsing breaks a sentence into smaller parts and uses dynamic programming to efficiently find valid sentence structures.

Steps in CKY Parsing

Step 1: Convert CFG to CNF

- Chomsky Normal Form (CNF) rules must be in either of these two formats:
    1. A → BC (where A, B, and C are non-terminals)
    2. A → a (where A is a non-terminal and "a" is a terminal)

Example CFG & CNF Conversion

```
Original Rule: S → NP VP
Converted Rule: S → X VP, X → NP
```

Step 2: Construct the CKY Table (Parsing Table)

- The **sentence is broken into substrings** and stored in a **table (matrix)**.
- Each **cell** in the table stores possible **non-terminals** that can generate that substring.

Example Sentence: "The cat chases the dog"

Grammar (CNF format):

```
S → NP VP
NP → Det N
VP → V NP
Det → "the"
N → "cat" | "dog"
V → "chases"
```

CKY Table Filling Process:

1. **Step 1:** Identify terminal rules (Det → the, N → cat, etc.).
2. **Step 2:** Combine smaller parts into larger structures (NP → Det N, VP → V NP).
3. **Step 3:** Continue until S is reached.

Step 3: Check if the Start Symbol (S) Appears

- If **S** appears in the **top-right cell of the table**, the sentence is valid.
- If **S is missing**, the sentence is **not valid** under the given grammar.

3. Advantages of CKY Parsing

- Efficient for CNF grammars using dynamic programming.
- Guarantees finding the correct parse if it exists.
- Used in NLP applications like Machine Translation & Speech Recognition.

4. Limitations of CKY Parsing

- Requires grammar in CNF, which may increase complexity.
- Slow for large sentences, as it has $O(n^3)$ time complexity.
- Cannot handle ambiguity well (requires Probabilistic CFG for disambiguation).