Unit 2

Contents

Syntactic Analysis: Syntactic Representations of Natural Language

- Syntactic analysis, or parsing, focuses on determining the grammatical structure of sentences in natural language.
- It identifies **how words combine to form phrases, clauses**, and complete sentences, adhering to the syntactic rules of a given language.
- Key syntactic representations include:

Constituency-Based Representations

- **Parse Trees**: Hierarchical structures where **nodes** represent **syntactic categories** (e.g., NP, VP), and **leaves** correspond to **words.**
- **Phrase Structure Grammar**: Frameworks like Context-Free Grammars (CFG) underlie these representations.

Dependency-Based Representations

- **Dependency Trees**: Represent relationships between words using directed edges, emphasizing head-dependent structures.
- **Dependency Parsing**: Efficient for languages with flexible word order.

Hybrid Representations

- Combine constituency and dependency representations for richer syntactic insights.

Parse Trees

- A **parse tree** (or **syntax tree**) is a tree structure that represents the syntactic structure of a sentence according to a given grammar, typically a **context-free grammar (CFG)**.

- It visualizes **how words** in a sentence are grouped into phrases and how these **phrases relate to one another hierarchically**.

- Parse trees are fundamental tools in computational linguistics and NLP, enabling deeper linguistic insights and effective language processing.

Components of a Parse Tree

1. **Root Node**: Represents the start symbol of the grammar, often denoted as S (Sentence).
2. **Non-Terminal Nodes**: Represent intermediate syntactic categories like noun phrases (NP), verb phrases (VP), or prepositional phrases (PP).
3. **Terminal Nodes**: Represent the actual words (or tokens) in the sentence.

Example

Consider the sentence:

"The cat sleeps.“

Using a simplified CFG:

1. S → NP VP
2. NP → Det N
3. VP → V
4. Det → "The"
5. N → "cat"
6. V → "sleeps"

The parse tree looks like this:

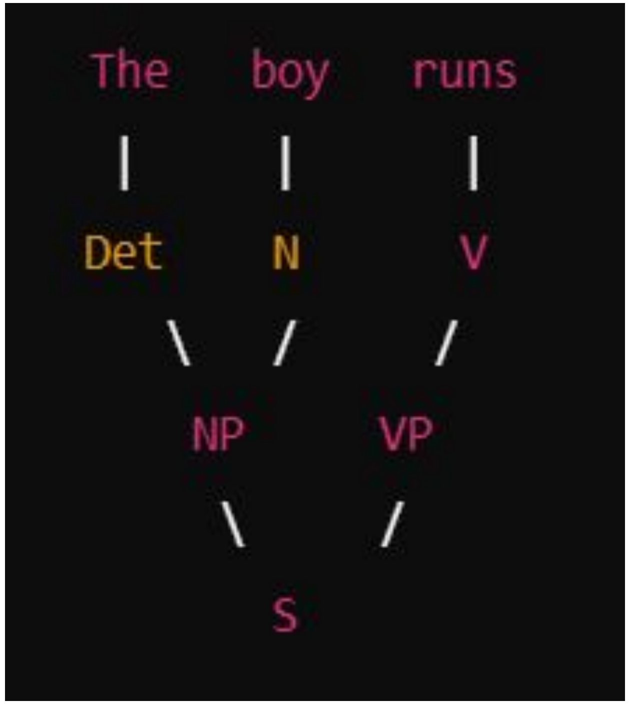
How Parse Trees Are Generated

- **Top-Down Parsing**: Start from the root (S) and expand nodes based on grammar rules until terminals match the input sentence.

- **Bottom-Up Parsing**: Start with the words (terminals) and combine them using grammar rules until the root (S) is derived.

Sentence: "The boy runs."

- Start with S → Expand into NP (Noun Phrase) and VP (Verb Phrase).
- Expand NP into Det (Determiner) and N (Noun).
- VP expands into V (Verb).

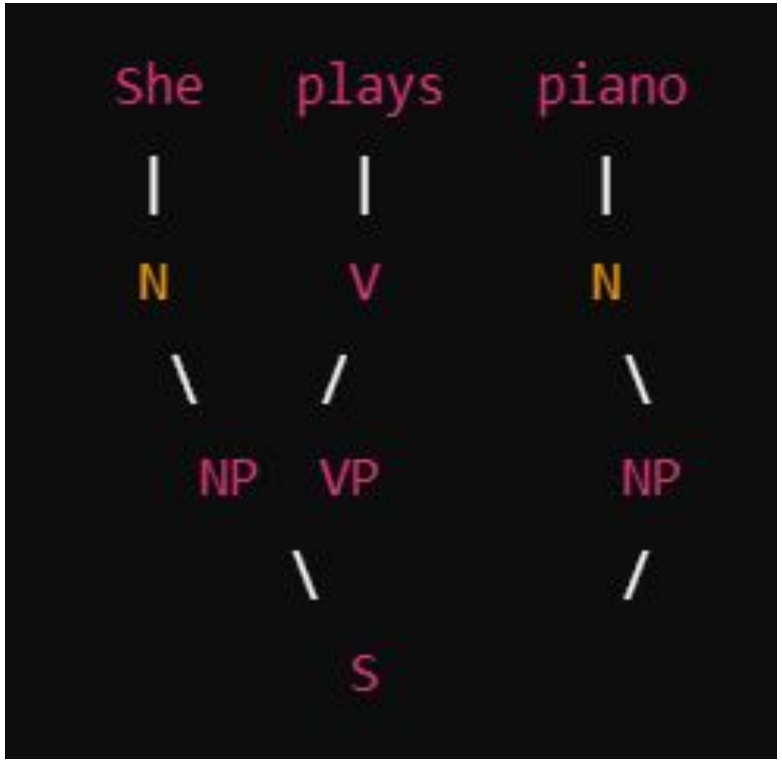


Begin with Det, N, and V → Combine Det and N into NP.

Combine NP and V into S.

Sentence: "She plays piano."


S
NP
VP
N
V
NP
plays piano

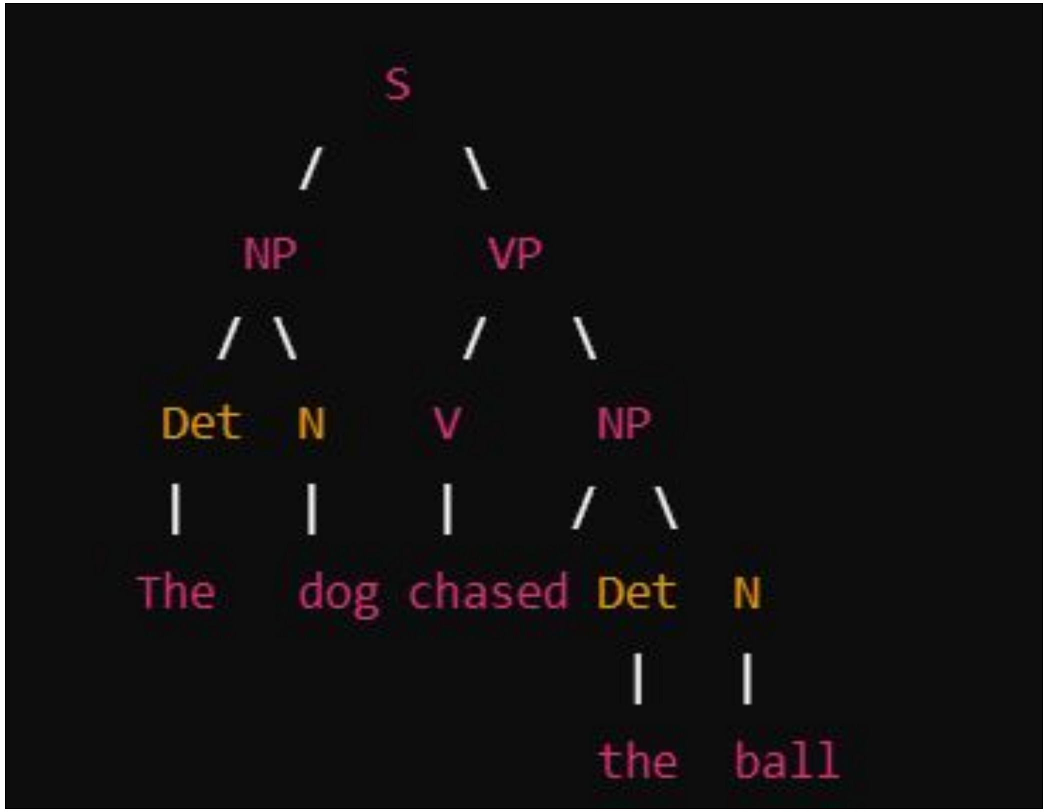

She
plays
piano
N
V
N
NP
VP
NP
S

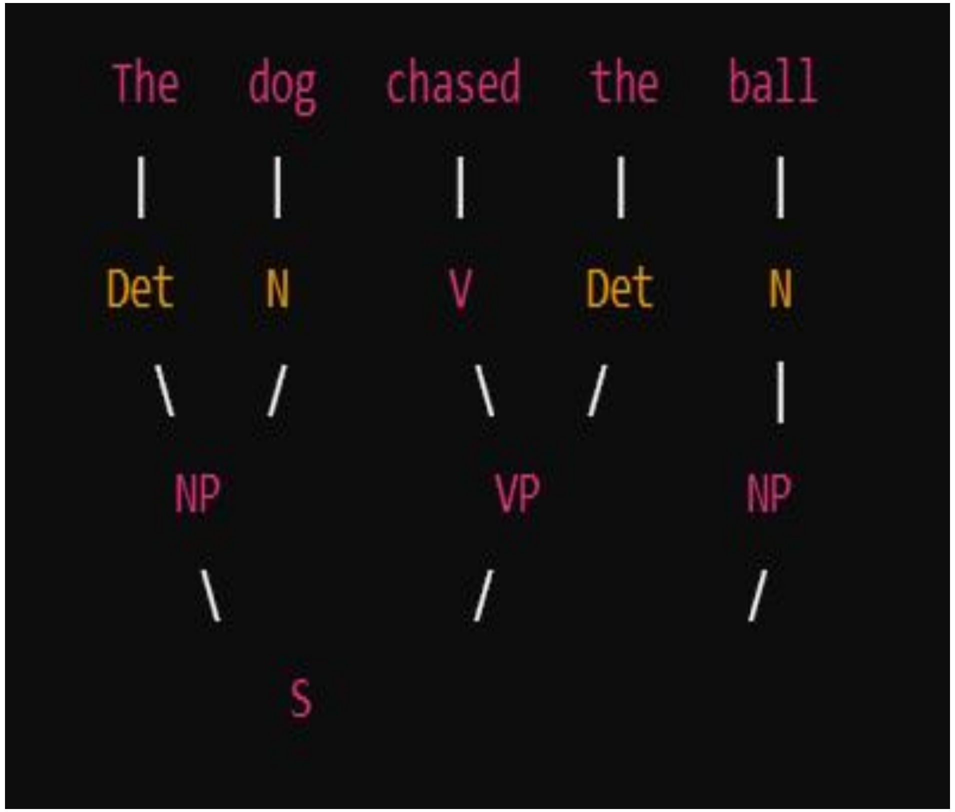
Sentence: "Birds fly."

```
     S
   /   \
  NP    VP
   |     |
   N     V
   |     |
 Birds  fly
```

```
Birds    fly
  |       |
  N       V
   \     /
    NP  VP
      \ /
       S
```

Sentence: "The dog chased the ball."


S
NP
VP
Det
N
V
NP
The
dog
chased
Det
N
the
ball


The
dog
chased
the
ball
Det
N
V
Det
N
NP
VP
NP
S

Phrase Structure Grammar

- Phrase Structure Grammar is a **type of formal grammar** used to represent the syntactic structure of sentences. Frameworks like **Context-Free Grammars (CFG) play a foundational role** in this representation.
- A set of rules that describe how words and phrases combine to form sentences in a language.
- It defines the hierarchical structure of a sentence by breaking it into smaller constituents or phrases (e.g., noun phrases, verb phrases).

Role of Context-Free Grammars (CFG) in NLP

1. **Definition of CFG**:

CFG is a type of grammar where **production rules** are used **to generate sentences.**

Each rule has a single non-terminal symbol (like S, NP, VP) on the left-hand side and a combination of terminal and non-terminal symbols on the right-hand side.

$S \rightarrow NP\ VP$

$NP \rightarrow Det\ N$

$VP \rightarrow V\ NP$

2. **Tree Representation**:

- CFG helps in constructing **parse trees**, which visually depict the syntactic structure of a sentence.

Applications in NLP:

- **Parsing**: CFGs are widely used **in parsers to analyze sentence structure**. Tools like **Stanford Parser** and **spaCy** implement such approaches.
- **Syntax Checking**: CFG **ensures that a sentence conforms** to the **grammatical rules** of a language.
- **Machine Translation**: By understanding sentence structure, CFG aids in **generating syntactically correct translations.**
- **Text Generation**: CFG helps in generating text that adheres to grammatical rules.

Limitations of CFG in NLP:

- CFGs assume that the structure of a language can be fully captured by context-free rules, but **natural languages often exhibit dependencies** (e.g., subject-verb agreement, long-distance dependencies) that are not easily handled by CFG alone.

- Extensions like **Lexicalized CFGs** or **Tree-Adjoining Grammars (TAGs)** are used to address these complexities.

Dependency-Based Representations

- Dependency-based representations **focus on the syntactic relationships** between words in a sentence. Instead of relying on **phrase structure (like in constituency-based** parsing), dependency representations emphasize the grammatical dependencies between individual words.

- **Key Features:**

1. **Head-Dependent Structure**:
    1. Each word in a sentence (except the root) is dependent on another word, which acts as its **head**.
    2. For example, in the sentence *"She eats an apple"*, the word ***"eats"* is the head**, and *"She"*, *"an"*, and *"apple"* are its dependents.

Directed Graph Representation:
Sentences are represented as directed graphs or trees, where:

- Nodes correspond to words.
- Edges represent dependencies (e.g., subject, object, modifier).

Grammatical Relations:
Dependencies are labeled with grammatical relations such as subject (*nsubj*), object (*obj*), and modifier (*amod*).

Non-Projectivity:
Some languages (e.g., German) exhibit non-projective dependencies, where dependencies can cross, making them more challenging to parse.

Dependency Trees

- A dependency tree is a tree structure that represents the syntactic structure of a sentence based on dependency relations.

Characteristics:

1. **Single Root**:
   - The tree has a single root node, typically the main verb or predicate of the sentence.
   - Example: In *"She quickly eats an apple"*, *"eats"* is the root.
2. **Child Nodes**:
   - Words directly dependent on the root or other nodes are represented as child nodes.
   - Each word (except the root) has exactly one parent.
3. **Labeled Arcs**:
   - Arcs (edges) between words are labeled with dependency types (e.g., subject, object).

For the sentence *"She eats an apple"*, the dependency tree might look like this:

```
      eats (root)
      /    \
   She    apple
             |
            an
```

- *She* → *eats*: Subject relation (*nsubj*).
- *eats* → *apple*: Object relation (*obj*).
- *apple* → *an*: Determiner relation (*det*).

Dependency Parsing

- Dependency parsing is the process of analyzing the grammatical structure of a sentence by **identifying dependency relationships between words** and constructing a dependency tree.

- **Types of Dependency Parsing:**

1. **Projective Parsing**:
    - Assumes **no crossing dependencies**.
    - Works well for languages with **simple structures** (e.g., English).
2. **Non-Projective Parsing**:
    - Allows crossing dependencies.
    - Necessary for languages like **Czech or German** with complex word order.

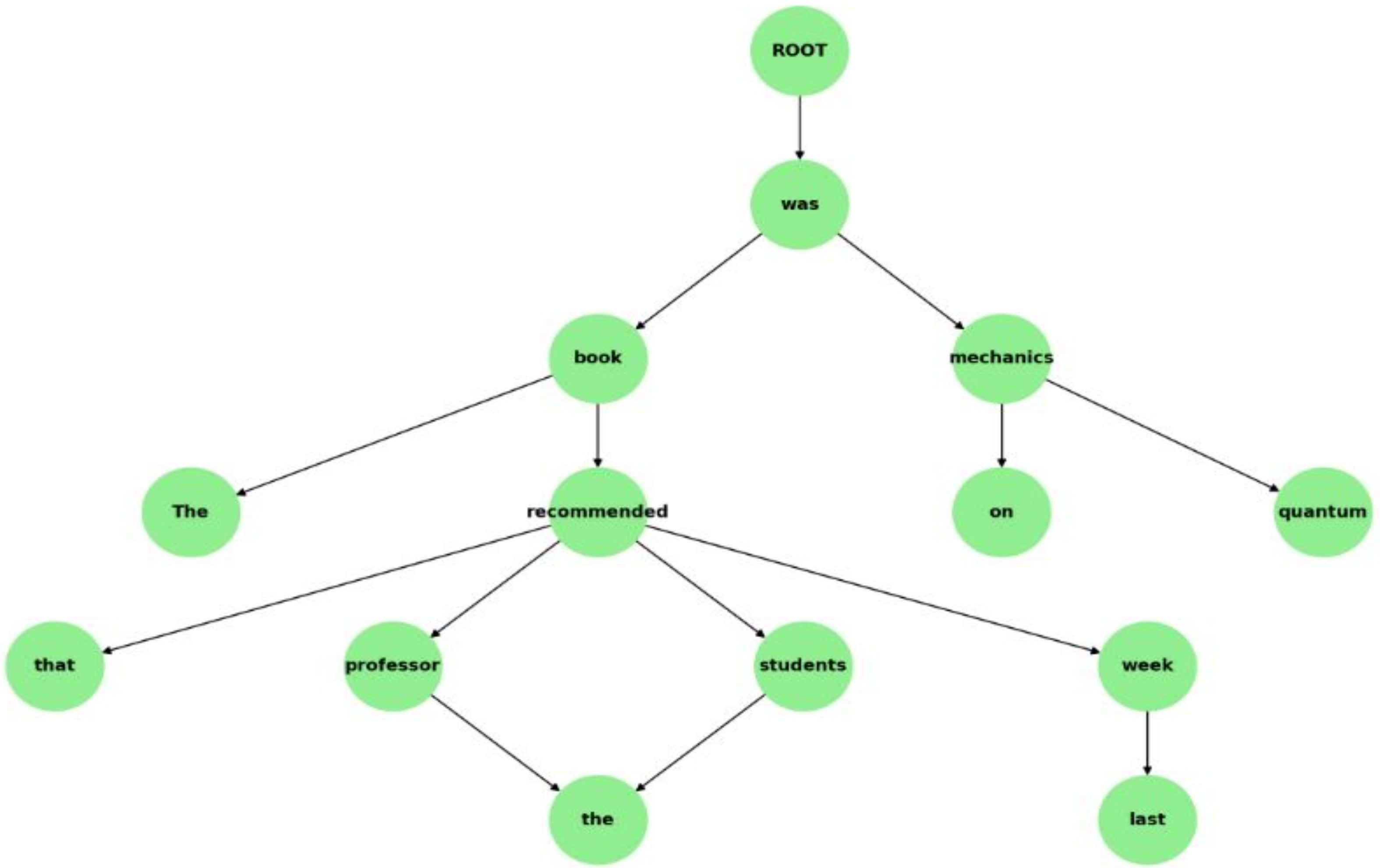
Parsing Methods:

1. **Rule-Based Parsers**:
    - Use hand-crafted rules to determine dependencies.
    - Example: Grammar-based parsers.
2. **Statistical Parsers**:
    - Use machine learning to predict dependencies based on training data.
    - Example: Transition-based parsers and graph-based parsers.
3. **Neural Parsers**:
    - Leverage deep learning models for dependency parsing.
    - Example: BERT-based parsers.

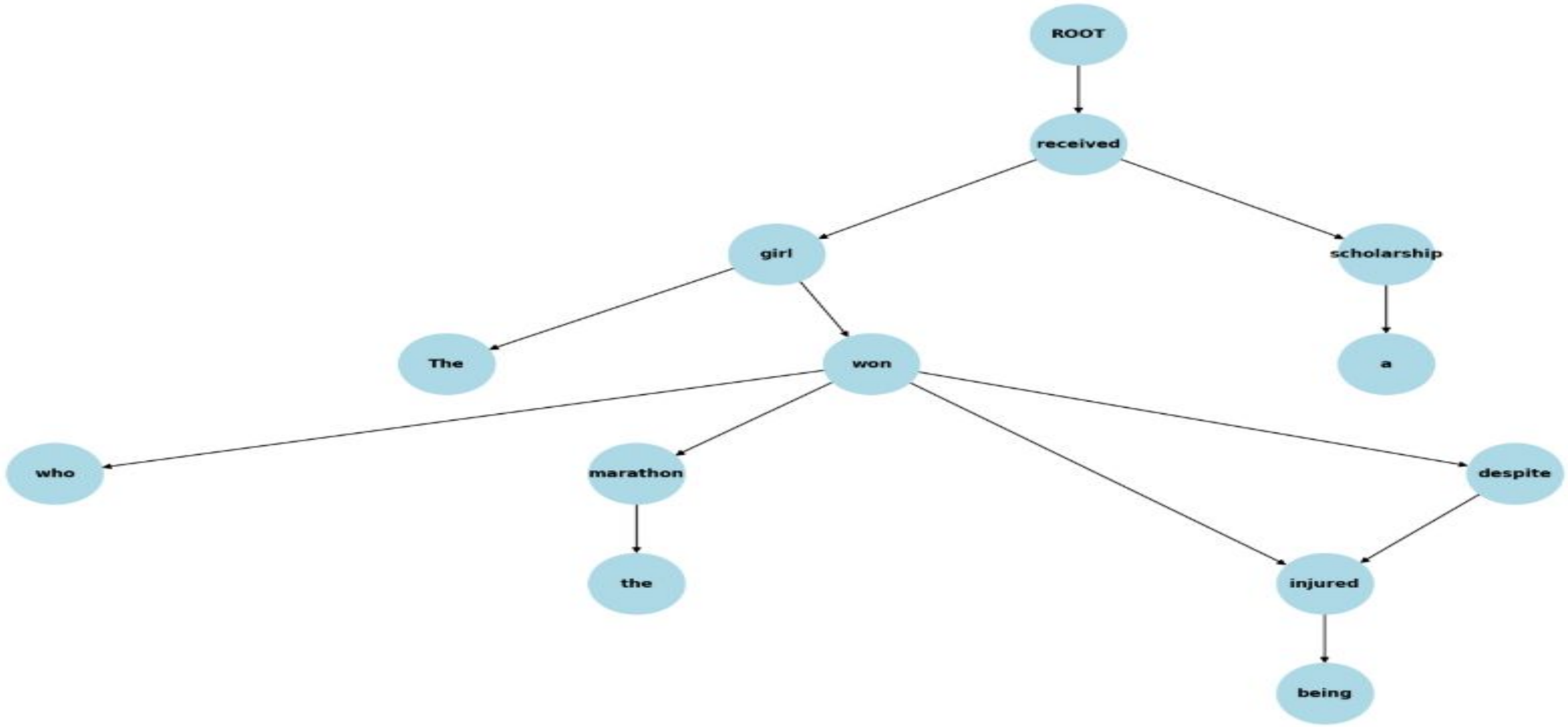
For the sentence: *"The quick brown fox jumps over the lazy dog"*, a dependency tree would look like:

```
        jumps (root)
      /      |       \
    fox    over      dog
   / |       |         |
The quick   the      lazy
      brown
```

"The book that the professor recommended to the students last week was on quantum mechanics."



"The girl who won the marathon despite being injured received a scholarship."



- Dependency tree generation involves algorithms that analyze the syntactic structure of sentences based on the relationships between words. These algorithms fall under the domain of **dependency parsing** in Natural Language Processing (NLP).

Transition-Based Parsing

- Transition-based parsers incrementally build dependency trees by performing a sequence of actions (e.g., SHIFT, REDUCE).
- **Key Algorithms:**
  - **Arc-Standard Parser:** Builds dependency arcs bottom-up.
  - **Arc-Eager Parser:** Builds arcs as soon as possible (top-down approach).
  - **Arc-Hybrid Parser:** Combines aspects of both Arc-Standard and Arc-Eager.
- **Popular Libraries:** SpaCy, Stanford CoreNLP.

- **Common Libraries and Tools**

1. **SpaCy:** Lightweight, fast, and uses transition-based parsing.
2. **Stanza (StanfordNLP):** Neural pipeline supporting dependency parsing.
3. **UDPipe:** Pre-trained models for Universal Dependencies datasets.
4. **MaltParser:** Implements transition-based dependency parsing.
5. **ZPar:** A statistical dependency parsing library.

Projective vs. Non-Projective Dependency Trees

- Dependency trees can be classified as **projective** or **non-projective** based on whether the edges cross when drawn over the linear sequence of words in a sentence.

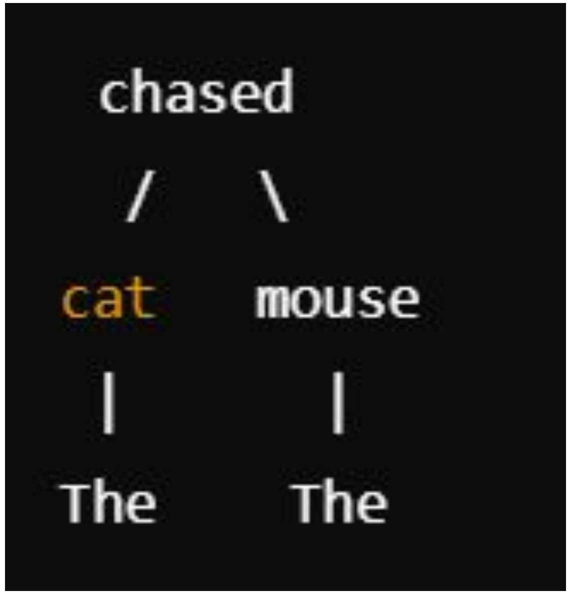
1. Projective Dependency Tree

In a projective tree:

- All edges can be drawn without crossing each other.
- This structure is typical for strict word-order languages like English.
- **Example Sentence (Projective):**
  "The cat chased the mouse."

Dependency Tree:

- chased → cat (subject)
- chased → mouse (object)
- cat → The (determiner)
- mouse → The (determiner)



- The cat chased the mouse
- The mouse that the cat chased ran away. (Non Projective)

2. Non-Projective Dependency Tree

In a non-projective tree:

- At least one edge crosses another.
- This structure occurs in free-word-order languages like Hindi, Russian, or Czech.

Example Sentence (Non-Projective):

"The report, I told you yesterday, is on the table."

Dependency Tree:

- is → report (subject)
- is → on (prepositional phrase)
- on → table (object of preposition)
- report → told (relative clause)
- told → I (subject of told)
- told → yesterday (temporal modifier)
- told → you (object)