What is Syntactic Analysis?

- Syntactic Analysis, also known as **parsing**, is the process of analyzing the grammatical structure of a sentence based on a given set of syntax rules (grammar). It ensures that a sentence follows the correct structure according to a language's rules.

Example of Syntactic Analysis

- Let's take an example sentence:

  👉 **"The cat sleeps on the mat."**

Step 1: Tokenization

Breaking the sentence into individual components (words).

Tokens: [The, cat, sleeps, on, the, mat]

Step 2: Parts of Speech (POS)

- **Step 2: Parts of Speech (POS) Tagging** Each word is labeled with its grammatical category.

Word	POS Tag
The	Determiner (DET)
cat	Noun (N)
sleeps	Verb (V)
on	Preposition (P)
the	Determiner (DET)
mat	Noun (N)

Step 3: Syntax Tree Construction

Now, we analyze the sentence's structure based on grammar rules.

Grammar Rules Applied:

- **Sentence (S) → NP (Noun Phrase) + VP (Verb Phrase)**
- **NP → DET + N** (Noun Phrase consists of a determiner and a noun)
- **VP → V + PP** (Verb Phrase consists of a verb and a prepositional phrase)
- **PP → P + NP** (Prepositional Phrase consists of a preposition and a noun phrase)

Syntax Tree Construction

The syntax tree for the sentence:

```
       S
      /    \
    NP      VP
    /   \     /  \
  DET  N   V   PP
  |       |    /      \
The  cat sleeps P  NP
                |    /      \
               on DET  N
                     |     |
                   the  mat
```

Key Takeaways

- **Syntactic Analysis** checks whether a sentence is grammatically correct based on a language’s syntax rules.
- It involves **Tokenization, POS Tagging, and Parsing**.
- The sentence structure is represented using **Parse Trees** or **Dependency Graphs**.
- Used in **Natural Language Processing (NLP)** for machine translation, chatbots, and more.

Parsing in Natural Language Processing (NLP)

- Parsing in NLP refers to analyzing the grammatical structure of a sentence to understand relationships between words. It helps machines process human language by structuring it according to grammar rules.

- **Types of Natural Language Parsing**

  Natural Language Parsing is mainly divided into:

1. **Dependency Parsing**
2. **Constituency Parsing**
3. **Shallow Parsing (Chunking)**
4. **Semantic Parsing**

Dependency Parsing

Definition: Analyzes grammatical relationships between words in a sentence by forming a dependency tree.

🔹 **Focus:** Determines how words depend on each other.

🔹 **Example:**

👉 **Sentence:** "The cat sits on the mat."

✅ **Dependency Tree Representation:**

```
   sits
   / | \
 cat  on  mat
  |     |
The  the
```

Explanation:

•"sits" is the **root (main verb)** of the sentence.

•"cat" is the **subject** of "sits."

•"on" is a **prepositional relation** to "sits."

•"mat" is the **object** of "on."

📌 **Use Cases:**

✅ Chatbots (Understanding sentence meaning)

✅ Machine Translation (Google Translate)

Constituency Parsing

🔹 **Definition:** Breaks a sentence into nested sub-phrases (constituents) based on a formal grammar.

🔹 **Focus:** Identifies noun phrases (NP), verb phrases (VP), etc.

🔹 **Example:**

👉 **Sentence:** "The cat sleeps."

✅ **Constituency Tree Representation:**

```
•          S
•         / \
•       NP   VP
•       / \   |
      DET  N   V
•       |   |   |
      The  cat sleeps
```

Explanation:

•**S (Sentence)** consists of **NP (Noun Phrase) + VP (Verb Phrase)**.

•**NP → DET + N →** "The cat"

•**VP → V →** "sleeps"

📌 **Use Cases:**

✅ Text-to-Speech Systems

✅ Grammar Checking (Grammarly)

Shallow Parsing (Chunking)

🔹**Definition:** Identifies phrases (chunks) in a sentence without analyzing the full tree structure.

Focus: Extracts noun phrases (NP), verb phrases (VP), etc.

Example:

👉 **Sentence:** "The black cat sits on the mat."

✅ **Chunking Output:**

[NP The black cat] [VP sits] [PP on] [NP the mat]

Use Cases:

✅ Named Entity Recognition (NER)

✅ Information Extraction

Semantic Parsing

🔹 **Definition:** Converts natural language into a machine-readable format.

🔹 **Focus:** Understanding the **meaning** of a sentence rather than just structure.

🔹 **Example:**

👉 **Sentence:** "Book a flight from New York to London."

✅ **Semantic Representation (JSON format):**

```
{ "intent“        : "book_flight",
 "origin“         : "New York",
 "destination“   : "London" }
```

📌 **Use Cases:**

✅ Voice Assistants (Alexa, Siri)

✅ Question-Answering Systems (Google Search)

Comparison of Parsing Techniques

Parsing Type	Focus	Example Output	Use Cases
Dependency Parsing	Word relationships	sits → (subject: cat)	Chatbots, AI Assistants
Constituency Parsing	Sentence structure	S → NP VP	Grammar Checking
Shallow Parsing (Chunking)	Phrase extraction	[NP The black cat]	Information Extraction
Semantic Parsing	Meaning extraction	{ "intent": "book_flight" }	Voice Assistants

Final Thoughts

📌 **Dependency Parsing** → Best for understanding word relationships.

📌 **Constituency Parsing** → Useful for grammar and structure analysis.

📌 **Shallow Parsing** → Quick and efficient for phrase extraction.

📌 **Semantic Parsing** → Most advanced, used for AI-based understanding.

Part-of-Speech (POS) Tag Parsing

- **Definition:** Identifies the grammatical category (e.g., noun, verb, adjective) of each word in a sentence.
- **Example Sentence:**

👉 "The quick brown fox jumps over the lazy dog."

- **POS Tagging Output:**

Word	POS Tag
The	Determiner (DET)
quick	Adjective (ADJ)
brown	Adjective (ADJ)
fox	Noun (N)
jumps	Verb (V)
over	Preposition (P)
the	Determiner (DET)
lazy	Adjective (ADJ)
dog	Noun (N)

POS Parsing Structure:

POS Parsing Structure:

[DET The] [ADJ quick] [ADJ brown] [N fox] [V jumps] [P over] [DET the] [ADJ lazy] [N dog]

Use Cases:

✅ Named Entity Recognition (NER)

✅ Sentiment Analysis

✅ Grammar Checking

Syntactic Parsing (Grammar-Based Parsing)

- **Definition:** Analyzes the **sentence structure** based on grammar rules.
- **Focus:** Identifies relationships between words (subject, verb, object).
- **Example Sentence:**

👉 "The cat sits on the mat."

✅ **Syntax Tree Representation:**

Syntax Tree Representation:

```
        S
       / \
     NP   VP
    / \    / \
DET  N   V    PP
  |      |   |     / \
The  cat sits  P  NP
                |   /  \
            on  DET  N
                    |   |
                 the  mat
```

Syntactic Parsing

Breakdown:

•S (Sentence) → NP (Noun Phrase) + VP (Verb Phrase)

•NP → DET (Determiner) + N (Noun) ⟶ "The cat"

•VP → V (Verb) + PP (Prepositional Phrase) ⟶ "sits on the mat"

Use Cases:

✅ Machine Translation

✅ Chatbots

✅ AI-based Sentence Correction

Semantic Parsing (Meaning Extraction)

🔹 **Definition:** Converts human language into machine-readable meaning representation.

🔹 **Focus:** Understanding the **meaning** rather than just structure.

Example Sentence:

👉 "Book a flight from New York to London."

✅ **Semantic Representation (JSON Format):**

```
{ "intent“     : "book_flight",
"origin“        : "New York",
"destination": "London" }
```

Use Cases:

✅ Virtual Assistants (Alexa, Siri)

✅ Question-Answering Systems (Google Search)

✅ AI-Powered Chatbots

Comparison of Parsing Types

Parsing Type	Focus	Example Output	Use Cases
POS Tag Parsing	Word Categories	[N cat] [V jumps] [DET the]	Text Analysis, NER
Syntactic Parsing	Sentence Structure	S → NP VP	Grammar Checking, Translation
Semantic Parsing	Meaning Extraction	{ "intent": "book_flight" }	AI Assistants, Chatbots

Final Thoughts

📌 **POS Tag Parsing** → Helps in identifying parts of speech for each word.

📌 **Syntactic Parsing** → Analyzes the structure of sentences using grammar rules.

📌 **Semantic Parsing** → Extracts meaning from the sentence for AI applications.

Treebank

What is a Treebank?

- A **Treebank** is a collection of sentences annotated with their syntactic structures, used for **syntax analysis** in Natural Language Processing (NLP).
- Treebanks provide training data for **syntactic parsers**, helping machines learn grammar rules and understand sentence structures.

Types of Treebanks in Syntax Analysis

Treebanks used in **syntax analysis** are primarily of two types:

1. **Constituency Treebanks** (Phrase-Structure Treebanks)
2. **Dependency Treebanks** (Relation-Based Treebanks)

Constituency Treebanks (Phrase-Structure Treebanks)

🔹 **Definition:**

Represents sentences using **phrase structure trees** based on **Context-Free Grammar (CFG)**.

Breaks sentences into **hierarchical phrases** like **Noun Phrases (NP)**, **Verb Phrases (VP)**, and **Prepositional Phrases (PP)**.

🔹 **Example:**

👉 **Sentence:** "The small dog barks loudly."

Dependency Tree Representation:

```
          S
        /   \
      NP     VP
     /  \    / \
   DET  N   V   ADV
    |   |   |    |
   The small dog barks loudly
```

Breakdown:

- **S → NP VP** *(Sentence consists of a Noun Phrase and a Verb Phrase)*
- **NP → DET N** *(Noun Phrase = Determiner + Noun → "The small dog")*
- **VP → V ADV** *(Verb Phrase = Verb + Adverb → "barks loudly")*

Famous Constituency Treebanks:

- **Penn Treebank (PTB)** – Annotates English text with phrase structure trees.
- **British National Corpus (BNC)** – Annotates large-scale English text.
- **NEGRA & TIGER Treebanks** – Used for **German language** parsing.

Use Cases:

✅ **Grammar Checking** (e.g., Grammarly)

✅ **Machine Translation** (Google Translate)

✅ **Text-to-Speech Systems**

Dependency Treebanks (Relation-Based Treebanks)

Definition:

Focuses on the **dependency relations** between words.

Each word is linked to another word, forming a **dependency tree** instead of a phrase structure.

Used in **Dependency Parsing**, where words have a **head** and **dependent**.

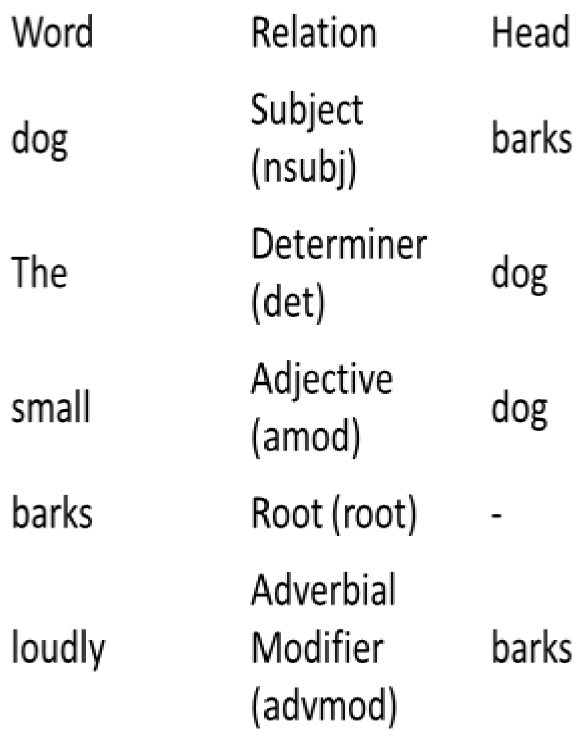
Example (Universal Dependencies Format):

👉 **Sentence:** "The small dog barks loudly."

Dependency Tree Representation:

```
        barks
      /   |   \
     dog  loudly
    /   \
  The  small
```

Dependency Relations:



Word	Relation	Head
dog	Subject (nsubj)	barks
The	Determiner (det)	dog
small	Adjective (amod)	dog
barks	Root (root)	-
loudly	Adverbial Modifier (advmod)	barks

Famous Dependency Treebanks:

•**Universal Dependencies (UD)** – Multi-language dependency annotations.

•**Prague Dependency Treebank (PDT)** – Focuses on **Czech language** parsing.

•**Turku Dependency Treebank** – Annotates Finnish sentences.

📌 **Use Cases:**

✅ **Chatbots & AI Assistants** (e.g., Siri, Alexa)

✅ **Sentiment Analysis**

✅ **Information Extraction**

Data-Driven Approach to Syntax

What is a Data-Driven Approach to Syntax?

- A data-driven approach to syntax in NLP refers to the use of machine learning models trained on large datasets (Treebanks) rather than relying on manually crafted grammar rules. These models learn syntactic structures from annotated corpora and generalize them to new text.

Key Components of a Data-Driven Syntax Approach

1. Annotated Corpora (Treebanks)
2. Machine Learning Algorithms
3. Feature Engineering
4. Statistical Parsing Models
5. Deep Learning for Syntax Parsing

1. Annotated Corpora (Treebanks)

🔷 A data-driven syntax approach relies on **Treebanks**, which contain sentences annotated with syntactic structures.

🔷 **Example Sentence:**

👉 `"The cat sits on the mat."`

✅ **Dependency Tree Representation (from Universal Dependencies Treebank):**

```nginx
        sits
       /  |  \
   cat  on   mat
    |     |
  The   the
```

Word	Relation	Head
cat	Subject (nsubj)	sits
sits	Root (root)	-
on	Preposition (prep)	sits
mat	Object (pobj)	on
The	Determiner (det)	cat
the	Determiner (det)	mat

🔷 **Common Treebanks Used in Data-Driven Syntax:**

- **Penn Treebank (PTB)** – English constituency parsing.
- **Universal Dependencies (UD)** – Multilingual dependency parsing.
- **Prague Dependency Treebank (PDT)** – Czech dependency parsing.

2. Machine Learning Algorithms for Syntax Parsing

🔷 Instead of manually defining grammar rules, machine learning models are trained on **annotated data** to learn patterns.

Supervised Learning for Parsing

- **Training Data:** Treebanks with labeled syntactic structures.
- **Algorithms Used:**
  - Support Vector Machines (SVM)
  - Random Forests
  - Neural Networks (LSTMs, Transformers)

✅ Example: Training a Syntax Parser

- Input: `"The dog runs."`
- Output: **POS Tags and Tree Structure** learned from data.

3. Feature Engineering in Data-Driven Syntax

🔷 Features extracted from training data help models understand syntax.

🔷 Examples of features:

- **POS Tags:** `"dog" = Noun, "runs" = Verb`
- **Word Dependencies:** `"dog" → subject of → "runs"`
- **Word Embeddings:** Vector representation of words (e.g., Word2Vec, BERT)

✅ **Example:**

Using **word dependencies** as features:

```scss
nsubj(dog, runs)
root(runs, -)
```

📌 **Machine Learning models learn that "dog" is usually the subject of "runs" from many examples.**

4. Statistical Parsing Models

Statistical parsing **assigns probabilities** to different possible parse trees.

Probabilistic Context-Free Grammar (PCFG)

Each **grammar rule** is assigned a **probability** based on its frequency in the dataset.

🔷 **Example PCFG Rules with Probabilities:**

```java
S → NP VP      (0.9)
NP → DET N     (0.6)
VP → V NP      (0.7)
```

📌 The parser selects **the most probable parse tree** based on training data.

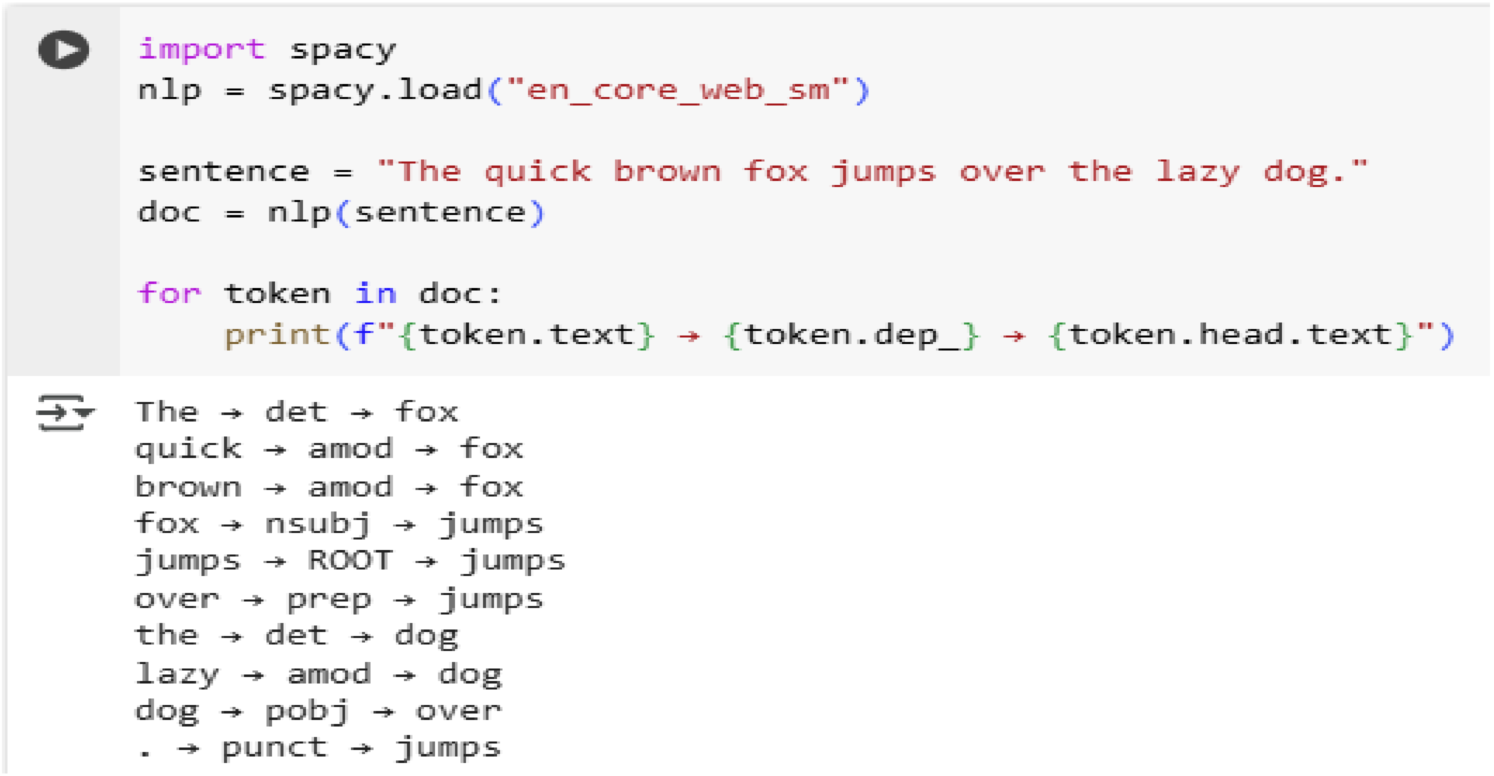
5. Deep Learning for Syntax Parsing

🔷 **Neural Network models** improve syntax analysis by **learning hierarchical structures** from data.

🔷 **Common architectures:**

- **Recurrent Neural Networks (RNNs)**
- **Long Short-Term Memory (LSTM) Networks**
- **Transformers (BERT, GPT)**

✅ **Example: Dependency Parsing with Neural Networks** Using **spaCy** in Python:



```python
import spacy
nlp = spacy.load("en_core_web_sm")

sentence = "The quick brown fox jumps over the lazy dog."
doc = nlp(sentence)

for token in doc:
    print(f"{token.text} → {token.dep_} → {token.head.text}")
```

```
The → det → fox
quick → amod → fox
brown → amod → fox
fox → nsubj → jumps
jumps → ROOT → jumps
over → prep → jumps
the → det → dog
lazy → amod → dog
dog → pobj → over
. → punct → jumps
```

Approach	Description	Example Output	Strengths
Rule-Based Parsing	Uses manually defined grammar rules	S → NP VP	High precision, but inflexible
Statistical Parsing	Uses probabilities from training data	P(S → NP VP) = 0.9	More flexible, but needs large data
Deep Learning Parsing	Uses neural networks to learn syntax	BERT predicts subject-object relations	Most accurate, but requires more computing power

Final Thoughts

📌Rule-based parsing is limited, while data-driven approaches generalize better.

📌 Deep learning-based parsers (BERT, spaCy) achieve state-of-the-art performance.

📌 Treebanks (e.g., Penn Treebank, UD) provide essential training data for syntax models.

Why Use a Data-Driven Approach for Syntax?

✅**More Flexible** – Adapts to different languages and dialects.

✅ **Higher Accuracy** – Learns from real-world examples.

✅ **Scalability** – Works well with large-scale NLP applications.

✅ **Less Manual Effort** – No need to manually define grammar rules.

Representation of Syntactic Structure in NLP

What is Syntactic Structure?

- Syntactic structure represents the **grammatical arrangement of words** in a sentence. It helps in understanding **how words and phrases relate to each other** in a sentence.
- In **Natural Language Processing (NLP)**, there are two primary ways to represent syntactic structures:

1. **Constituency Structure (Phrase Structure)**
2. **Dependency Structure (Relation-Based)**

1. Constituency Structure (Phrase-Structure Representation)

🔷 Definition:

- Represents syntax using **phrase structure trees** (Constituents).
- Based on Context-Free Grammar (CFG).
- Breaks sentences into phrases like Noun Phrases (NP), Verb Phrases (VP), and Prepositional Phrases (PP).

🔷 Example Sentence:

👉 `"The small dog barks loudly."`

✅ Constituency Tree Representation:

```markdown
        S
       / \
     NP    VP
    /  \   /  \
  DET  N  V    ADV
   |   |   |    |
  The small dog barks loudly
```

🔷 Breakdown:

- S → NP VP *(Sentence consists of a Noun Phrase and a Verb Phrase)*
- NP → DET N *(Noun Phrase = Determiner + Noun → "The small dog")*
- VP → V ADV *(Verb Phrase = Verb + Adverb → "barks loudly")*

📌 Use Cases:

✅ Grammar Checking (e.g., Grammarly)

✅ Machine Translation (Google Translate)

✅ Text-to-Speech Systems

2. Dependency Structure (Relation-Based Representation)

🔷 Definition:

- Focuses on the **dependency relations** between words.
- Each word is linked to another word, forming a **dependency tree.**
- Used in **Dependency Parsing**, where words have a **head** and **dependent.**

🔷 Example Sentence:

👉 `"The small dog barks loudly."`

✅ Dependency Tree Representation:

markdown

```
      barks
     /  |  \
   dog  loudly
  /  \
The  small
```

🔷 Dependency Relations:

Word	Relation	Head
dog	Subject (nsubj)	barks
The	Determiner (det)	dog
small	Adjective (amod)	dog
barks	Root (root)	-
loudly	Adverbial Modifier (advmod)	barks

📌 Use Cases:

✅ **Chatbots & AI Assistants** (e.g., Siri, Alexa)

✅ **Sentiment Analysis**

✅ **Information Extraction**

Comparison of Syntactic Structure Representations

Representation Type	Focus	Example Output	Use Cases
Constituency Structure	Sentence structure	`S → NP VP`	Grammar Checking, Machine Translation
Dependency Structure	Word relationships	`nsubj(dog, barks)`	Chatbots, Sentiment Analysis

Implementation of Syntactic Structure in Python (Using spaCy)

Dependency Parsing Example with spaCy

python

```python
import spacy

# Load English NLP model
nlp = spacy.load("en_core_web_sm")

# Input sentence
sentence = "The small dog barks loudly."
doc = nlp(sentence)

# Print dependencies
for token in doc:
    print(f"{token.text} → {token.dep_} → {token.head.text}")
```

🔷 Output:

nginx

```
The → det → dog
small → amod → dog
dog → nsubj → barks
barks → ROOT → barks
loudly → advmod → barks
```

📌 This shows how words relate to each other in the sentence.

Final Thoughts

📌 Constituency Structure → Good for **phrase-level parsing** (used in Penn Treebank).

📌 Dependency Structure → Good for **word-to-word relationships** (used in Universal Dependencies).

📌 **spaCy & NLP models use dependency parsing for AI-powered applications.**

Parsing Algorithms

- **What is Parsing in Syntax Analysis?**
- Parsing in **Natural Language Processing (NLP)** is the process of **analyzing the syntactic structure** of a sentence to understand relationships between words. It is crucial for grammar checking, machine translation, question answering, and chatbots.
- There are **two major types of parsing algorithms** used in **syntax analysis**:

1. **Top-Down Parsing**
2. **Bottom-Up Parsing**

Each type has different algorithms, including:

- **Top-Down Parsing:**
  - Recursive Descent Parsing
  - Predictive Parsing
- **Bottom-Up Parsing:**
  - Shift-Reduce Parsing
  - CYK (Cocke-Younger-Kasami) Parsing
- **Statistical & Neural Parsing:**
  - Probabilistic Context-Free Grammar (PCFG)
  - Dependency Parsing with Machine Learning

1. Top-Down Parsing

🔷 **Definition:**

- Starts from the **root (Sentence S)** and expands into **subcomponents** (NP, VP, etc.).
- Works based on **grammar rules (Context-Free Grammar - CFG).**

1.1 Recursive Descent Parsing

🔷 **How it works:**

- Uses a **set of recursive functions** to expand grammar rules.
- Backtracks when a rule fails.

🔷 **Example Grammar:**

```mathematica
S → NP VP
NP → DET N
VP → V NP
DET → "The"
N → "dog" | "cat"
V → "chased"
```

🔷 **Example Sentence:** `"The dog chased the cat."`

✅ **Recursive Parsing Process:**

```sql
1. Start with S → NP VP
2. NP → DET N  → "The" "dog"
3. VP → V NP   → "chased" "the cat"
4. Sentence successfully parsed!
```

1.2 Predictive Parsing (LL(1) Parsing)

🔷 Improvement over Recursive Descent:

- Uses **lookahead tokens** to decide the next step, avoiding backtracking.

🔷 Example Grammar:

```mathematica
S → NP VP
NP → DET N
VP → V NP
```

✅ Predictive Parsing Table:

Lookahead	Production Rule
"The"	`NP → DET N`
"dog"	`N → "dog"`
"chased"	`V → "chased"`

📌 Benefit:

- **More efficient** than recursive descent.

2. Bottom-Up Parsing

🔷 Definition:

- Starts from **words (tokens)** and builds up the parse tree **towards the root (Sentence S)**.

2.1 Shift-Reduce Parsing

🔷 How it works:

- **SHIFT** moves a word onto a stack.
- **REDUCE** replaces words with **grammar rules** when possible.

🔷 **Example Sentence:** `"The dog chased the cat."`

✅ **Shift-Reduce Steps:**

Stack	Action
The	SHIFT
The dog	SHIFT
NP (The dog)	REDUCE
NP chased	SHIFT
NP chased the	SHIFT
NP chased the cat	SHIFT
NP chased NP	REDUCE (VP → V NP)
S (NP VP)	REDUCE (S → NP VP)
✔ Sentence Parsed!	✅

2.2 CYK (Cocke-Younger-Kasami) Parsing

🔷 How it works:

- Uses **dynamic programming** to parse sentences based on a **Chomsky Normal Form (CNF) grammar.**
- Builds a **table from words to sentence structure.**

🔷 **Example:**

For sentence `"The dog sleeps"` with CNF Grammar:

```mathematica
S → NP VP
NP → DET N
VP → V
DET → "The"
N → "dog"
V → "sleeps"
```

✅ CYK Table:

"The"	"dog"	"sleeps"
DET	N	V
NP		VP
S		

3. Statistical & Neural Parsing

🔷 Why Statistical Parsing?

- **Traditional parsers struggle with real-world language**, where grammar rules are not always followed.
- **Statistical parsers use probabilities** to determine the most likely parse tree.

3.1 Probabilistic Context-Free Grammar (PCFG)

🔷 How it works:

- Assigns **probabilities** to different grammar rules based on training data.

🔷 Example Probabilities:

```java
S → NP VP  (0.9)
NP → DET N  (0.6)
VP → V NP   (0.7)
```

✅ Sentence Parsing Example:

For `"The dog sleeps"`:

```mathematica
P(S → NP VP) = 0.9
P(NP → DET N) = 0.6
P(VP → V) = 0.7
```

📌 **Higher probability trees are preferred.**

3.2 Dependency Parsing with Machine Learning

🔷 Modern NLP uses deep learning models like:

- Neural Networks (LSTMs, Transformers)
- Pre-trained models (BERT, spaCy, Stanza)

✅ Example: Dependency Parsing using spaCy

```python
import spacy
nlp = spacy.load("en_core_web_sm")

sentence = "The cat chased a mouse."
doc = nlp(sentence)

for token in doc:
    print(f"{token.text} → {token.dep_} → {token.head.text}")
```

🔷 **Output:**

```bash
The → det → cat
cat → nsubj → chased
chased → ROOT → chased
a → det → mouse
mouse → dobj → chased
```

📌 Machine learning parsers generalize better than rule-based methods!

Comparison of Parsing Algorithms

Algorithm	Type	Strengths	Weaknesses
Recursive Descent Parsing	Top-Down	Simple	Inefficient due to backtracking
Predictive Parsing (LL(1))	Top-Down	Fast, No Backtracking	Needs left-factored grammar
Shift-Reduce Parsing	Bottom-Up	Efficient for small grammars	Hard for ambiguous sentences
CYK Parsing	Bottom-Up	Handles complex grammars	Requires Chomsky Normal Form
Probabilistic Parsing (PCFG)	Statistical	Handles real-world text	Needs large training data
Neural Dependency Parsing	ML-Based	Most accurate	Requires high computational power

---

Final Thoughts

📌 **Top-Down Parsing (Recursive & Predictive)** is useful for small rule-based grammars.

📌 **Bottom-Up Parsing (Shift-Reduce, CYK)** is efficient for programming languages.

📌 **Statistical Parsing (PCFG, ML Models)** is used in modern NLP (Google Translate, Chatbots).