Syntax Analysis (Parsing)

- **Definition:** Syntax analysis is the process of analyzing a sequence of words according to the grammar rules of a language.
- It generates a **parse tree** showing how a sentence fits into a grammar.
- Two major approaches are: **Top-Down Parsing** and **Bottom-Up Parsing**.

---

1. Top-Down Parsing

Meaning

- Parsing starts from the **start symbol (S)** of the grammar.
- Expands non-terminals **step by step** until the input string is derived.
- Works like: **From root (S) → to leaves (terminals).**

Working

1. Begin with the **start symbol (S)**.
2. Expand using **production rules**.
3. Keep expanding until the derived string matches the input.

Example

Grammar:

S → NP VP

NP → Det N

VP → V NP

Det → 'the'

N → 'dog'

V → 'chased'

Input string: **“the dog chased the dog”**

Steps:

1. Start: S
2. S → NP VP
3. NP → Det N → the dog
4. VP → V NP → chased NP

5. NP → Det N → the dog

Derived string matches input. Parse successful.

Types

- **Recursive Descent Parsing** (can use backtracking).
- **LL(1) Parsing** (uses lookahead).

Advantages

- Conceptually simple.
- Builds tree directly from grammar.
- Good for **predictive parsing** when grammar is suitable.

Disadvantages

- May enter **infinite recursion** if grammar has **left recursion**.
- Requires grammar to be **factored** (LL form).

---

2. Bottom-Up Parsing

Meaning

- Parsing starts from the **input string** and works **backwards** toward the start symbol.
- Builds parse tree from **leaves (terminals) → to root (start symbol)**.
- Works like: **From input → to start symbol (S).**

Working

1. Begin with the **input string**.
2. Apply grammar rules in **reverse** (reductions).
3. Continue until you reach the start symbol.

Example

Input string: **"the dog chased the dog"**

Steps (reductions):

1. the → Det

2. dog → N
3. Det N → NP
4. chased → V
5. V NP → VP
6. NP VP → S

Reduced to start symbol (S). Parse successful.

Types

- **Shift-Reduce Parsing**
- **LR Parsing (SLR, LALR, Canonical LR)**

Advantages

- Works for a **wider class of grammars** (LR grammars).
- More powerful and efficient for programming languages.

Disadvantages

- More complex to implement.
- Harder to understand intuitively.