Table of Contents

Syntax analysis in computer science refers to the process of analyzing the structure of a given sequence of symbols according to a formal grammar.

A context-free grammar (CFG) is commonly used to specify the syntax or structure of programming languages, natural languages, and other formal languages.

Here’s an explanation of how CFGs are used for syntax analysis:

1. Context-Free Grammar (CFG): A CFG consists of a set of production rules that define how non-terminal symbols can be replaced with sequences of terminal and non-terminal symbols. It consists of a set of non-terminal symbols, a set of terminal symbols, a start symbol, and a set of production rules.
2. Non-Terminal Symbols: Non-terminal symbols represent syntactic categories or placeholders that can be further expanded or replaced with other symbols. They are usually represented by uppercase letters. Examples include expressions, statements, variables, or program.
3. Terminal Symbols: Terminal symbols represent the basic units or tokens in the language being parsed. They are usually represented by lowercase letters or specific symbols. Examples include keywords, identifiers, operators, literals, or punctuation marks.
4. Production Rules: Production rules specify the transformations or replacements that can be applied to non-terminal symbols. They define how non-terminal symbols can be

expanded into sequences of terminal and non-terminal symbols. Production rules are typically written in the form: A → β, where A is a non-terminal symbol and β is a sequence of terminal and non-terminal symbols.

5. Derivations: Derivations are a sequence of rule applications starting from the start symbol of the CFG, eventually leading to the desired sequence of terminal symbols. Each step in the derivation applies a production rule to replace a non-terminal symbol with its expansion.
6. Parse Tree: A parse tree represents the hierarchical structure of a parsed input according to the CFG. It is a graphical representation where each node corresponds to a symbol in the grammar, and the edges represent the application of production rules.
7. Ambiguity: CFGs can sometimes be ambiguous, meaning that a given input can have multiple valid parse trees or interpretations. Ambiguity can cause difficulties in parsing and can be resolved by either disambiguating the grammar or using additional parsing techniques.
8. Top-Down Parsing: Top-down parsing is a technique that starts from the start symbol and recursively expands non-terminal symbols, trying to match the input symbols. It usually employs techniques like recursive descent or LL parsing.
9. Bottom-Up Parsing: Bottom-up parsing is a technique that starts from the input symbols and applies production rules in reverse, reducing them to non-terminal symbols until reaching the start symbol. Techniques like LR parsing or shift-reduce parsing are commonly used for bottom-up parsing.
10. Syntax Analysis: Syntax analysis, also known as parsing, is the process of analyzing a sequence of symbols based on the rules defined by the CFG. It checks if the input adheres to the grammar rules and constructs a parse tree or reports syntax errors.

Syntax analysis or parsing is the second phase of a compiler.

What is Context-Free Grammar ?

A context-free grammar (CFG) consisting of a finite set of grammar.

A context-free grammar has four components:

1. Non-terminals (V): Non-terminals are syntactic variables that denote sets of strings.
2. Terminal symbols (Σ): Terminals are the basic symbols from which strings are formed.
3. Productions (P): The productions of a grammar specify the manner in which the terminals and non-terminals can be combined to form strings. Each production consists of a non-terminal called the left side of the production, an arrow, and a sequence of terminals, called the right side of the production.
4. Start symbol (S): From where the production begins.

For example, in CFG,

```
S –> 0 | 1A, A –> 1
```

V = {S, A}
Σ = {0, 1}
P = {S–>0| S –> 1A | A –> 1}
S = {S}