* **Table of Contents:**

1. Introduction to CFG
2. Derivations of CFG [leftmost and rightmost]
3. Parse tree
4. Ambiguity
5. Elimination of left recursion
6. Left factoring
7. **Introduction to CFG**

A context Free Grammar (CFG) is a 4-tuple such that-

G = (V, T, P, S)

where-

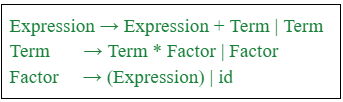
V = Finite non-empty set of variables / non-terminal symbols

T = Finite set of terminal symbols

P = Finite non-empty set of production rules

S = Start symbol

**Example-01: Find out the characteristics of CFG from the following expression: -**



Variables V = {Expression, Term, Factor}

Terminals T = {+, \*, (, ), id}

Production rules P = {

Expression 🡪 Expression + Term | Term

Term 🡪 Term \* Factor | Factor

Factor 🡪 (Expression) | id

}

Start Symbol S = {Expression}

**Example-02: Write down the formal definition of CFG for the following expression: -**

|  |
| --- |
| S → aSb / ∈ |

Variables V = {S}

Terminals T = {a, b, ∈}

Production rules P = S → aSb / ∈

Start Symbol S = {S}

**Example-03: Write down the formal definition of CFG for the following expression: -**

|  |
| --- |
| S → aAb / ∈  A → aAb / ∈ |

Consider a grammar G = (V, T, P, S) where-

Variables V = {S}

Terminals T = { a , b }

Production rules P = { S → aSbS , S → bSaS , S → ∈ }

Start Symbol S = { S }

**Example-04: Find out the characteristics of CFG from the following expression: -**

|  |
| --- |
| S → SS  S → (S)  S → ∈ |

Consider a grammar G = (V , T , P , S) where-

V = { S }

T = { ( , ) }

P = { S → SS , S → (S) , S → ∈ }

S = { S }

**Example 5 : Find out the characteristics of CFG from the following expression: -**

|  |
| --- |
| S → 0S | 1S  S → ε |

Consider a grammar G = (V , T , P , S) where-

V = { S }

T = { 0, 1}

P = { S → 0S, S 🡪 1S , S → ε  }

S = { S }

**Example 6 : Find out the characteristics of CFG from the following expression: -**

|  |
| --- |
| S → ABa ,  A → BB ,  B → ab ,  AA → b |

Consider a grammar G = (V , T , P , S) where-

V = { S , A , B }                                                  // Set of Non-Terminal symbols

T = { a , b }                                                        // Set of Terminal symbols

P = { S → ABa , A → BB , B → ab , AA → b }  // Set of production rules

S = { S }    // Start symbol

1. **Derivations of CFG [leftmost and rightmost]**

**Ambiguity and Limitations of CFG**

**One big challenge** with using CFGs in compilers is ambiguity. Ambiguity happens when the same piece of code can be interpreted in more than one way, which makes it unclear what the actual structure should be.



**Figure 2:** A common example is the “if-then-else” problem.

If you have two if statements followed by an else, the compiler might get confused about which if the else is supposed to match with. This makes it hard to figure out the intended structure of the code.

To deal with this, compiler designers use **disambiguation rules**. A typical rule is that an else clause always pairs with the closest if statement. Another way to resolve ambiguity is using advanced parsing techniques, like operator precedence, to handle tricky cases and make sure things are interpreted correctly.

**Another limitation** of CFGs is that they can’t handle everything in modern programming languages. Some languages have context-sensitive rules, meaning that how certain parts of the code should be understood depends on the surrounding context. Since CFGs are context-*free*, they can’t directly handle those cases. To get around this, compilers use additional methods like attribute grammars or extra checks during the semantic analysis phase.

**Another limitation** of CFG is that it cannot handle all the features of modern programming languages. Some languages have **context-sensitive** rules, which require the meaning of a part of the code to depend on the context in which it appears. Since CFG is **context-free**, it cannot handle such cases directly. To address this, compilers often use **attribute grammars** or perform additional checks during semantic analysis (*6.0 Semantic Analysis Translation and Attribute Grammars*, 2024).