**Unit 3**

**Grammar:**

 A formal grammar is a set of production rules for rewriting strings or to generate strings of a language. A Grammar is a 4-tuple such that-$G=(V\_{N},Σ,P,S)$ **where**

$V\_{N}$ **is set of non-terminal symbol which is given by only capital letter.**

$Σ$ **is set of terminal symbol which is given by only small letter.**

$P$ **is the set of productions (rules) for generating string in the form of** $α⟹β$ **where** $α and β$ **is combination of non-terminal and terminal. So**$ α$**,**$ β\in (V\_{N}∪Σ)$**\***

$S$ **is the start symbol.**

**Grammar Constituents:**

A Grammar is mainly composed of two basic elements-



1. Terminal symbols

2. Non-terminal symbols

**1. Terminal Symbols-**

* Terminal symbols are those which are the constituents of the sentence generated using a grammar.
* Terminal symbols are denoted by using small case letters such as a, b, c etc.

**2. Non-Terminal Symbols-**

* Non-Terminal symbols are those which take part in the generation of the sentence but are not part of it.
* Non-Terminal symbols are also called as **auxiliary symbols** or **variables**.
* Non-Terminal symbols are denoted by using capital letters such as A, B, C etc.

### Language generated by a grammar (language of grammar): Given a grammar G, its corresponding language L(G) represents the set of all strings generated from G.

### $$L\left(G\right)=\{w|S\left\{\begin{array}{c}\*\\⟹\\G\end{array}\right.w\}$$

* For a given grammar G, its corresponding language L(G) is unique.
* The language L(G) corresponding to grammar G must contain all strings which can be generated from G.
* The language L(G) corresponding to grammar G must not contain any string which can not be generated from G.
* For a given language L(G), there can be more than one grammar which can produce L(G).
* The grammar G corresponding to language L(G) must generate all possible strings of L(G).
* The grammar G corresponding to language L(G) must not generate any string which is not part of L(G).

## ****Types of Grammars:****

Grammars are classified on different basis as-



### https://www.gatevidyalay.com/wp-content/uploads/2018/08/Language-of-Grammar.png

**Context Free Grammar**: A Context free Grammar is a quadruple (4-tuple) such that-

$ G=(V\_{N},Σ,P,S)$

**where** $V\_{N}$ **is set of non-terminal symbol which is given by only capital letter.**

$Σ$ **is set of terminal symbol which is given by only small letter.**

$P$ **is the set of productions (rules) for generating string in the form of** $α⟹β$ **where** $α and β$ **is combination of non-terminal and terminal. So**$ α$**=**$V\_{N}$**, |**$ α|$**=1 and** $β$**= (**$V\_{N}∪Σ$**)\*.**

**So in the L.H.S. there is only one capital letters but in the R.H.S. combination of capital and small both.**

$S$ **is the start symbol.**

The language generated using Context Free Grammar is called as **Context Free Language**.

**Derivations:** Derivation is a sequence of production rules. So the process of deriving string with the help of CFG is called derivations.

A string is derived by a CFG iff

### $$L\left(G\right)=\{w|S\left\{\begin{array}{c}\*\\⟹\\G\end{array}\right.w\}$$

## 1. Leftmost Derivation:

In the leftmost derivation, the input is scanned and replaced with the production rule from left to right. So in leftmost derivation, we read the input string from left to right.

A leftmost derivation is obtained by applying production to the leftmost variable in each step.

## 2. Rightmost Derivation:

In rightmost derivation, the input is scanned and replaced with the production rule from right to left. So in rightmost derivation, we read the input string from right to left.

A rightmost derivation is obtained by applying production to the rightmost variable in each step.

**Parse Tree:** A derivation tree or parse tree is an ordered rooted tree that graphically represents the semantic information a string derived from a context-free grammar. So Derivation tree is a graphical representation for the derivation of the given production rules for a given CFG.

It is the simple way to show how the derivation can be done to obtain some string from a given set of production rules. The derivation tree is also called a parse tree.

### Representation Technique:

* **Root vertex** − Must be labeled by the start symbol.
* **Vertex** − Labeled by a non-terminal symbol.
* **Leaves** − Labeled by a terminal symbol or ε.

If S → x1x2 …… xn is a production rule in a CFG, then the parse tree / derivation tree will be as follows −



A parse tree contains the following properties:

1. The root node is always a node indicating start symbols.
2. The derivation is read from left to right.
3. The leaf node is always terminal nodes.
4. The interior nodes are always the non-terminal nodes.

**Yield/ Derivation of Parse Tree:**

Concatenating the leaves of a parse tree from the left produces a string of terminals. This string of terminals is called as **yield of a parse tree**.

### Sentential Form and Partial Derivation Tree:

A partial derivation tree is a sub-tree of a derivation tree/parse tree such that either all of its children are in the sub-tree or none of them are in the sub-tree.

### If a partial derivation tree contains the root S, it is called a sentential form. The above sub-tree is also in sentential form.

**Ambiguity:** A grammar is said to be ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string.

 If the grammar is not ambiguous, then it is called unambiguous.

 If a context free grammar **G** has more than one derivation tree for some string **w ∈ L(G)**, it is called an **ambiguous grammar**.

 There exist multiple right-most or left-most derivations for some string generated from that grammar.

Finally A grammar is said to ambiguous if for any string generated by it, it produces more than one Parse tree. Parse tree is also called derivation tree, syntax tree and generation tree.

# No method can automatically detect and remove the ambiguity, but we can remove ambiguity by re-writing the whole grammar without ambiguity.

# Unambiguous Grammar: A grammar can be unambiguous if the grammar does not contain ambiguity that means if it does not contain more than one leftmost derivation or more than one rightmost derivation or more than one parse tree for the given input string.

# Note:

* There exists no general algorithm to remove the ambiguity from grammar.
* To check grammar ambiguity, we try finding a string that has more than one parse tree.
* If any such string exists, then the grammar is ambiguous otherwise not.

**Methods To Remove Ambiguity:**

The ambiguity from the grammar may be removed using the following methods-



* By fixing the grammar
* By adding grouping rules
* By using semantics and choosing the parse that makes the most sense
* By adding the precedence rules or other context sensitive parsing rules

### Regular grammar: The language accepted by finite automata can be described using a set of productions known as regular grammar. The productions of a regular grammar are of the following form:

|  |
| --- |
| $$A\rightarrow Λ$$$$A\rightarrow a$$$$A\rightarrow aB$$$$A\rightarrow Ba$$ |

###  Where $a\in $ $∑$ and $A,B\in V\_{N}.$

### A language generated by a regular grammar is known as regular language.

### Form of regular grammar: There are two types of regular grammar.

### 1. Left linear grammar

### 2. Right linear grammar

### Left linear grammar: A regular grammar is in the form of left linear grammar if production are in form $A\rightarrow Λ$

### $$A\rightarrow a$$

###  $A\rightarrow Ba$.

### Right linear grammar: A regular grammar is in the form of right linear grammar if production are in following form

###   $A\rightarrow Λ$

### $$A\rightarrow a$$

###  $A\rightarrow aB$.

### Conversion of Finite automata into Regular grammar: Every DFA can be described by regular grammar.

### Let the DFA,$M=(Q,Σ,δ,q\_{0,}F)$ and its right regular grammar $G=(V\_{N},Σ,P,S)$**then use following steps,**

### **Rename state qi as Ai.**

### **If qj is not final state.**

###

### $A\_{i}⟶ aA\_{j}$

### **3. If qj is final state then write**

###

### $A\_{i}⟶ aA\_{j}|a$

**Conversion of Finite automata into Regular grammar:** In this following steps are required to write a left linear grammar corresponding to DFA.

1. Interchange starting state and final state.
2. Reverse the direction of all the transitions.
3. Write the grammar from the transition graph in left linear.

Regular grammar into DFA:$ A\_{i}⟶ aA\_{j}$ then draw



If $A\_{i}⟶ a$ then draw



**Simplification of CFG**: All the grammars are not always optimized that means the grammar may consist of some extra symbols (non-terminal). Having extra symbols, unnecessary increase the length of grammar. Simplification of grammar means reduction of grammar by removing useless symbols. A CFG can be simplified by eliminating:

1. Use less symbol
2. $Λ$**-** Production
3. Unit Production

**Elimination of Useless Symbol**: A grammar may contain symbol and production which are not useful for derivation of string.

There are two types of useless symbol:

1. Non generating symbol

2. Non reachable symbol

**Generating symbol**: A symbol $X\in V\_{N}$ is a generating symbol iff $X⇒W.$

So X is called generating symbol if its derives only a set of terminal (small letter). Otherwise it is called non generating variables.

**Reachable Symbol**: A variable symbol X is called reachable if there is a path from initial state to variable X as$ S⟶AXB$.

For this we use dependency graph for variable with productions as



If there is no path from start symbol S to a variable X, then X is non- reachable.

**Null Production**: A production of the form$A\rightarrow Λ$**,** is called a null production.

**Unit Production**: A production of the form$A\rightarrow B$**,** is called a null production.

So $α⟹β$ **is null if**$\left|α\right|=1 and \left|β\right|=0$**.**

**And** $α⟹β$ **is unit if** $\left|α\right|= \left|β\right|=1 where both are non terminal as capital letter.$

**Normal form:** Production in G, satisfying certain restriction are said to be in Normal Form. For converting CFG into Normal Form , CFG must be in simplified/reduced. So CFG must be without null production, unit production and useless symbol**.**

There are two normal forms for CFG:

1. Chomsky Normal Form (CNF)
2. Greibach Normal Form(GNF)

**Chomsky Normal Form:** A simplified context free grammar CFG is said to be in CNF if every production is in of the form:

1. $A\rightarrow BC$
2. $ A\rightarrow a$

Where $A,B,C\in V\_{N}$

According to CNF in the left hand side there is only one capital letter but in the right hand side either exact two non-terminals (capital letter) or exact one terminal (small letter).

**Greibach Normal Form:** A simplified context free grammar CFG is said to be in GNF if every production is in of the form:

$$A\rightarrow aα$$

Where $A\in V\_{N}$ and $α$ is a string of zero or more variables (capital letter).

According to GNF in the left hand side there is only one capital letter but right hand side should start with a terminal followed by a string of non-terminal of length zero or more. So the right hand side always starts with only one small letter and followed by sequence of capital letter.

**Chomsky Hierarchy:**

Chomsky Hierarchy represents the class of languages that are accepted by the different machine. The category of language in Chomsky's Hierarchy is as given below:

1. Type 0 known as Unrestricted Grammar.
2. Type 1 known as Context Sensitive Grammar.
3. Type 2 known as Context Free Grammar.
4. Type 3 Regular Grammar.

 

According to Noam Chomosky, there are four types of grammars − Type 0, Type 1, Type 2, and Type 3. The following table shows how they differ from each other −

|  |  |  |  |
| --- | --- | --- | --- |
| Grammar Type | Grammar Accepted | Language Accepted | Automaton |
| Type 0 | Unrestricted grammar | Recursively enumerable language | Turing Machine |
| Type 1 | Context-sensitive grammar | Context-sensitive language | Linear-bounded automaton |
| Type 2 | Context-free grammar | Context-free language | Pushdown automaton |
| Type 3 | Regular grammar | Regular language | Finite state automaton |

## Type - 3 Grammar:

Type-3 grammars generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form X → a or X → aY

Where X, Y ∈ $V\_{N}$ (Non terminal)

and a ∈ $∑$ (Terminal)

The rule S → $∧$ is allowed if S does not appear on the right side of any rule.

## Type - 2 Grammar:

Type-2 grammars generate context-free languages.

The productions must be in the form A → γ

where A ∈ N (Non terminal)

and γ ∈ (T ∪ N)\* (String of terminals and non-terminals).

These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

## Type - 1 Grammar:

Type-1 grammars generate context-sensitive languages. The productions must be in the form

α A β → α γ β

where A ∈ N (Non-terminal)

and α, β, γ ∈ (T ∪ N)\* (Strings of terminals and non-terminals)

The strings α and β may be empty, but γ must be non-empty.

The rule S → ε is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

## Type - 0 Grammars:

Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.

They generate the languages that are recognized by a Turing machine.

The productions can be in the form of α → β where α is a string of terminals and non-terminals with at least one non-terminal and α cannot be null. β is a string of terminals and non-terminals.