Unit 3

Moore machine

Machine with finite number of states and for which the output symbol at any given time depends only on the current state of the machine

[Q , sigma, del, Q0, delta, lambda]

Q: finite set of states

Sigma: finite set of input alphabets

Del: transition function

Del: Q x sigma -> Q

Q0: single initial state

Delta: finite set of output alphabets

Lambda: machine function

Lambda: Q -> delta

Mealy machine

Machine with finite number of states and for which output symbol at any given time depends on current input as well as the current state

[Q, sigma, del, Q0, delta, lambda]

Q: finite set of states

Sigma: finite set of input alphabets

Del: transition function

Del: Q x sigma -> Q

Q0: single initial state

Delta: finite set of output alphabets

Lambda: machine function

Lambda: Q x sigma -> delta

Moore machine	Mealy machine
Output depends only upon the present state.	Output depends on the present state as well as present input.
Moore machine also places its output in the state.	Mealy Machine places its output on the transition.
More states are required.	Less number of states are required.
If input string length is n, output length will always be n+1	if input string length is n, output length will always be n
Output is placed on states.	Output is placed on transitions.
Easy to design.	It is difficult to design.
If input changes, output does not change, as moore machine is a type of finite state machine where the output is determined solely by the current state, not by the current input.	If input changes, output also changes, because in mealy machine output depends on the present state as well as present input.

Unit 4

Definition

Grammar for a particular language is defined as finite set of formal rules for generating syntactically correct sentences

Grammar consists of 2 types of symbols, terminals and non-terminals. Terminal symbols are those that are part of a generated statement/sentence, non-terminal symbols are those that take part in the formation of a statement

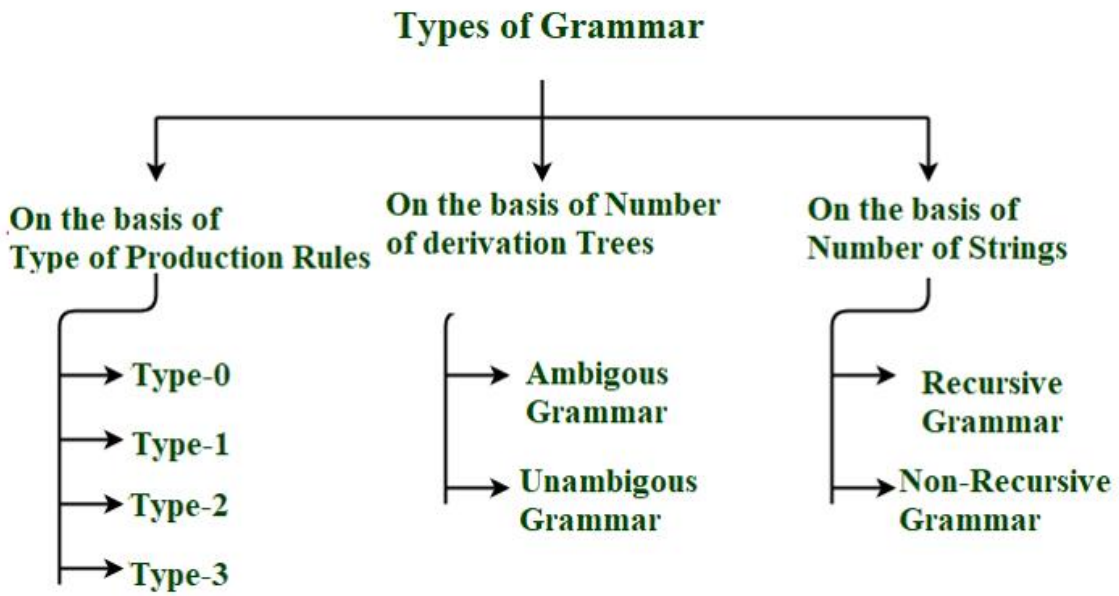
[V T P S]

V – finite set of non-terminals (variables)

T – finite set of terminals

P – production rules

S – start symbols



Example:

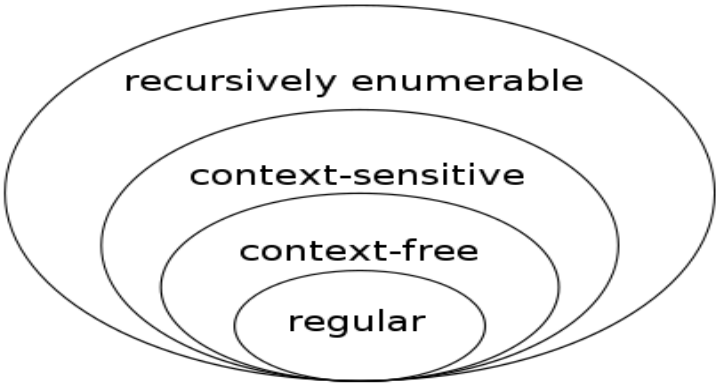
Consider a grammar G = (V , T , P , S) where

- V = { S } // **Set of Non-Terminal** symbols
- T = { a , b } // **Set of Terminal** symbols
- P = { S → aSbS , S → bSaS , S → ∈ } // **Set of production rules**
- S = { S } // **Start symbol**

Language = {any number of a's followed by any number of b's or any number of b's followed by any number of a's (a^nb^n or b^na^n)} (it's not this idk what it is)

Types of grammar by Chomsky hierarchy

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 0 – unrestricted grammar

This grammar is called phase structure grammar (PSG) or unrestricted grammar.

It is called unrestricted because it is the highest degree of grammar with no restriction

General definition

$(V \cup T)^{+} \rightarrow (V \cup T)^{*}$

Left hand side of the production can contain variables and terminals but not empty symbols (epsilon) and right-hand side of the production can contain variables and terminals including empty symbols (epsilon)

Language of this grammar is called **recursively enumerable language**

Examples

S -> Aba/E aS -> aAb AB->CB

Type 1 – non contracting grammar or context sensitive grammar

G = (V, T, S, P)

V is finite set of variables (non-terminals)

T is finite set of terminal symbols

S ∈ V is a designated symbol called the start symbol

For type 1,2,3 write the above 4 lines first and then write P onwards

P: αAβ -> αγβ with A ∈ V , γ ∈ (V U T)+ , α, β ∈ (V U T)* or

S->ε and if S-> ε ∈ P, then s does not appear on the right hand side of any production

Examples:

S -> cD

aA -> aBb

SA -> CB

Language of this grammar is **context sensitive language**

Machine of this grammar **linear bounded automata**

Type 2 – context free grammar

P: V -> (V U T)*

Right hand side of the production has no left or right context

Let αAβ is string here α is left context and β is right context.

εAε here no left and right context.

$\varepsilon A = A\varepsilon = A$

This grammar is called context free grammar because right hand side of the production V has left or right context free

Examples:

S -> CD/ε

A -> aBb

B -> BC

Language is **context free language**

Machine is **push down automata**

Type 3 – regular grammar or finite state grammar

P: V -> T* / T*V (right linear)

V -> T*/VT* (left linear)

A regular grammar is one that is either left linear or right linear but not both

What is linear: only one variable will be there. ax + b is linear but ax^2 + bx + c is non-linear because it has the power of x(variable). xy + 6 is not linear because 2 variables

Examples:

S -> a/

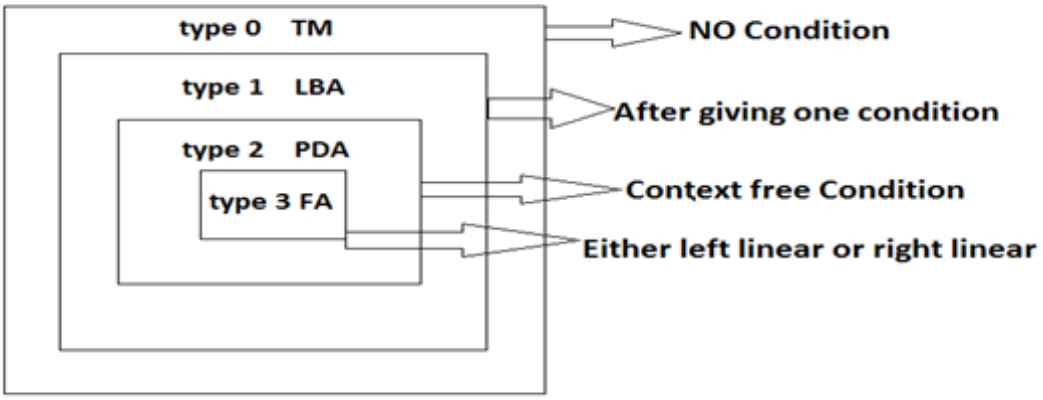
A -> aA (right linear)

B -> Bb (left linear)

C -> abba

Language is **regular language or finite state langauge**

Machine is **finite automata**



Why is Context free grammar called so

Context Free Grammar provides no mechanism to restrict the usage of the production rule $A \rightarrow \alpha$ within some specific context unlike other types of grammars.

Chatgpt explanation

Context-Free Grammar (CFG) is called "context-free" because the production rules in the grammar can be applied regardless of the surrounding symbols (i.e., the **context**) in which a non-terminal appears.

In other words, each rule in a CFG has the form:

$A \rightarrow \alpha A$

where:

- AA is a **single** non-terminal symbol.
- α is a string of terminals and/or non-terminals.

The key point is that the left-hand side of a production consists of only **one** non-terminal, and its replacement does not depend on what symbols appear before or after it in a string.

Example:

A CFG for balanced parentheses:

1. $S \rightarrow (S)S$
2. $S \rightarrow SS$
3. $S \rightarrow \epsilon S$

Here, the rule $S \rightarrow (S)$ can be applied **anywhere** in a string where SS appears, without considering its surrounding symbols.

In contrast:

In **context-sensitive grammar**, rules depend on the surrounding symbols: Example: $ABC \rightarrow ADC$ where B can only be replaced if it is between A and C

Thus, CFGs are called "context-free" because each non-terminal can be replaced **independently** of its context.

Applications of context free grammar

- For defining programming languages
- Parsing the program by constructing syntax tree
- Translation of programming languages
- Describing arithmetic expressions
- Compiler construction

Ambiguous grammar

A context free grammar for a language is said to be ambiguous if there exist at least one string which can be derived in more than one way

This means that there exists more than one leftmost derivation or more than one rightmost derivation or more than 1 derivation tree

Normal form for context free grammars

Context free grammar can be written in certain standard form known as normal form

These normal forms impose certain restrictions on the production in the CFG

Chomsky normal form

Any context free language without ε which is generated by a grammar in which all the productions are of the form

A -> BC

A -> a

Where A, B, C are non-terminals and a is terminal

In special case where language consists of ε, then S -> ε is allowed

Greibach normal form

Every context free language without ε can be generated by a grammar whos every production is of the form

A -> aα

where A is non-terminal, a is terminal and α can be ε or set of non-terminals

Extra

The **Chomsky hierarchy**, introduced by Noam Chomsky in 1956, is a classification of formal grammars used in theoretical computer science and linguistics. It consists of four levels, each defining a class of languages with increasing expressive power. At the lowest level, **Type 3 (Regular languages)** are generated by regular grammars and can be recognized by finite state automata, making them efficient for lexical analysis. **Type 2 (Context-Free Languages)** are produced by context-free grammars (CFGs) and can be recognized by pushdown automata, widely used in programming language parsing.

Moving up, **Type 1 (Context-Sensitive Languages)** are described by context-sensitive grammars, requiring linear-bounded automata for recognition and allowing rules that depend on surrounding symbols. At the highest level, **Type 0 (Recursively Enumerable Languages)** encompass all possible formal languages, generated by unrestricted grammars and recognized by Turing machines. While Type 0 languages are the most powerful, they are not necessarily decidable. This hierarchy is fundamental in computational theory, helping distinguish between languages based on their computational complexity and the machines required to process them.