

## "Introduction of TOC"

\* Automata  $\Rightarrow$

→ Study of abstract computing devices or machines.

\* Symbol  $\Rightarrow$

→ A symbol is an abstract entity.

Ex = a, b, 0, 1.....

\* Alphabet  $\Rightarrow$

→ An alphabet is a finite set / collection of symbols.

Ex =  $S = \{a, b, c\}$

$\Sigma = \{0, 1\}$

## \* String $\Rightarrow$

→ Sequence of symbols.

Ex  $\Rightarrow \Sigma = \{a, b\}$

strings = { a, b, aa, ab, ba, bb }    hence we have 6 strings.  
Length = 2

→ If the length of string is zero then it is known as Empty / NULL string.

→ It is denoted by  $\infty$  or  $\lambda$ .

## \* Language $\Rightarrow$

→ Collection of strings.

→ where strings are restricted over given alphabet.

Ex  $\Rightarrow \Sigma = \{a, b\}$

$L_1$  = set of all strings length of 2

$$\{aa, ab, ba, bb\}$$

where language  $L_1$  is FINITE.

$L_2$  = set of all strings which starts with 'a'

$$\{a, aa, ab, aab, aba, aaa, abb, \dots\}$$

where language  $L_2$  is INFINITE.

\* if length is given  
for a string then  
it can be finite.

### \* Length of a string $\Rightarrow$

→ No. of symbols in a string.

$$\text{Ex} = |\{a, b\}| = 2$$

$$|\{\in\}| / |\in| = 0$$

### \* Prefix of a string $\Rightarrow$

→ If  $w = \{x, y\}$ , for some string  $y$  then  $x$  is a prefix of  $w$ .

$$\text{Ex} = w = \{0011\}$$

$$\begin{matrix} \{0, 01\} & \{00, 11\} & \{001, 1\} \\ x & y & x \\ y & y & y \end{matrix}$$

\* Suffix of a string  $\Rightarrow$

$\rightarrow$  if  $w = xy$  for a string  $x$ , then  $y$  is a suffix of  $w$ .

Ex=  $w = \{ \underline{0} \underline{0} 1 \underline{1} \underline{y} \}$

$$\left. \begin{array}{ll} \frac{001}{x} & \frac{1}{y} \\ \frac{00}{x} & \frac{11}{y} \\ \frac{0}{x} & \frac{011}{y} \end{array} \right\} \underline{\{ \underline{1}, \underline{11}, \underline{011} \}}$$

\* if asking about no. of prefixes or suffixes then answer would be  $\boxed{n}$ .

- \*  $n = \text{no. of alphabet.}$
- \*  $n$  is included  $\epsilon$ .

\*  $\{ \epsilon, 1, 11, 011 \}$

## \* Kleene Closure $\Rightarrow$

- It is denoted by \* (Astexisk) after the name of alphabet i.e.  $E^*$ .
- This notation is also known as Kleene star.

Ex  $\Rightarrow \Sigma = \{a, b\}$

$\Sigma^0$  = set of all strings of length '0' =  $\{\epsilon\}$

$\Sigma^1$  = set of all strings of length '1' =  $\{a, b\}$

$\Sigma^2 = \Sigma^1 \cdot \Sigma^1 = \{a, b\} \{b, a\} = \overbrace{\{aa, ab, ba, bb\}}$   
Set of All strings of Length '2'

$\Sigma^3 = \Sigma^1 \cdot \Sigma^1 \cdot \Sigma^1 = |\Sigma|^3 = 8$  = set of all strings of length '3'

$\Sigma^n$  = set of all strings of length 'n' =  $2^n$

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \dots$

\* no. of strings possible  
over  $\Sigma^*$  = infinite  
\* no. of languages possible  
over  $\Sigma^*$  = infinite

$\Sigma^* = \boxed{\text{set of all strings of all length over } \{a, b\}}$   
 $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$       Universal Set

## \* Grammar $\Rightarrow$

- It enumerates string of language.
- It is a finite set of rules for defining a language.
- A grammar is defined as 4 Tuples.  
 $(V, T, P, S)$

where  $V$  = Set of Non-Terminals.

$T(\epsilon)$  = Set of Input Terminals.

$P$  = finite set of production rule.

$S$  = Start of Symbol.

## \* Getting a string from a grammar = Derivation

Ex =

Production  $\Rightarrow \begin{cases} S = a\underline{S}B \\ S = aB \\ B = b \end{cases}$

here =  $V = \{S, B\}$

$T = \{a, b\}$

Q⇒ find out aabb from the given symbol -

$$S = aSB$$

where =  $S = aB$   
 $B = b$

$$S = a\underline{S}B$$

$$\begin{array}{c} S = a\underline{a}BB \\ \hline S = aabb \end{array}$$

} Derivation

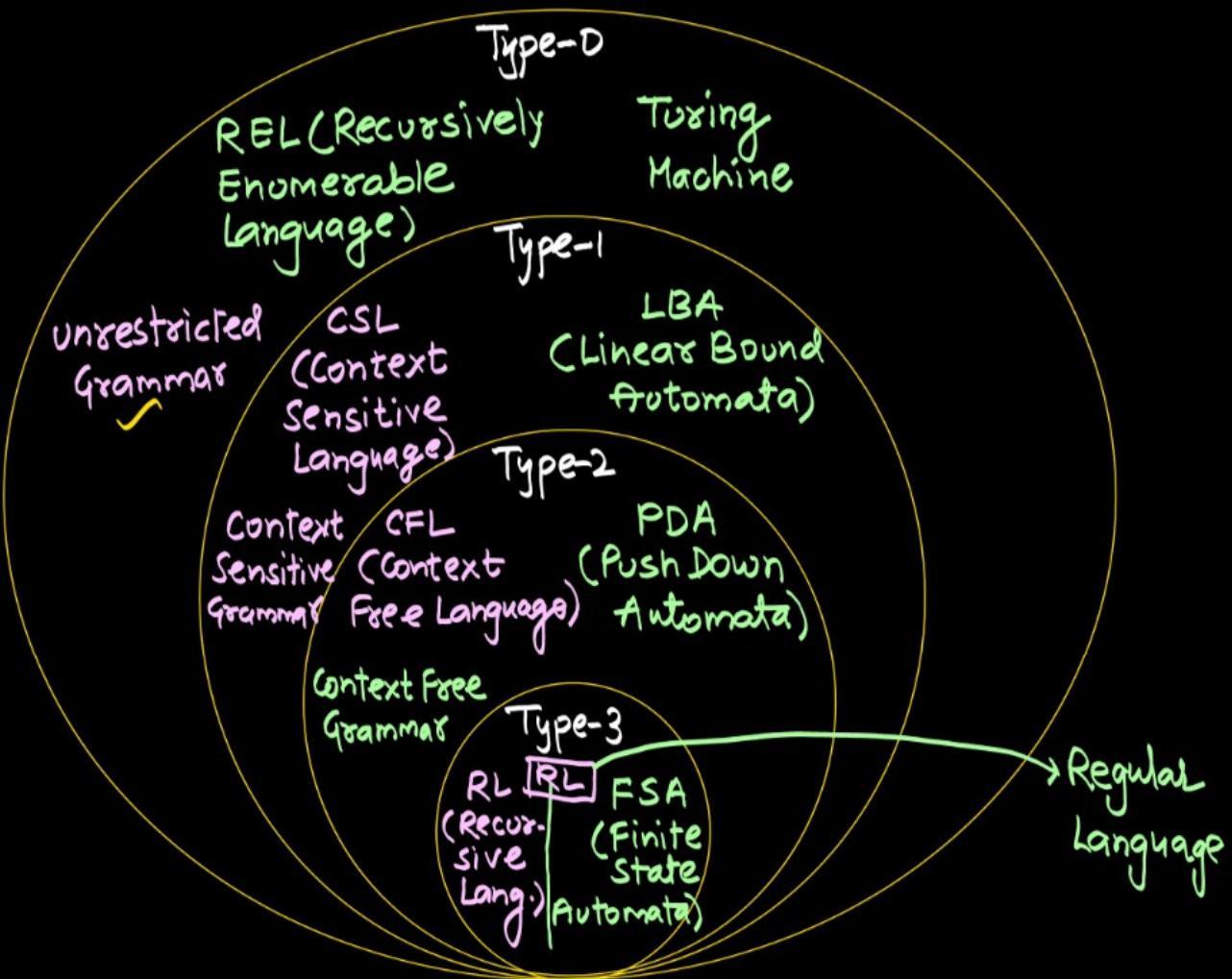
$$\rightarrow S = aabB$$

$$\underline{\underline{S = aabb}}$$

## \* \* Chomsky Hierarchy →

→ It consists of 4 classes -

- A. Type 0
- B. Type 1
- C. Type 2
- D. Type 3



### A. Type 0 Grammar $\Rightarrow$

- Unrestricted Grammar.
- मट Unrestricted Grammar है, जो किसी formal languages को include करती है।
- मट Grammar exactly सारी Languages को generate करती है, जिन्हें Turing Machine Recognize कर सकती है।
- Also known as = Turing Machine, Recursively Enumerable Language.

### B. Type-1 Grammar →

- Context Sensitive Grammar.
- Type-1 Grammar के द्वारा define की गई Languages को Linear Bound Automata के द्वारा accept किया जाता है।
- Also Known As- Context Sensitive Language, Linear Bound Automata.

### C. Type-2 Grammar →

- Context Free Grammar
- Type -2 Grammar के द्वारा define की गई Languages को Push Down Automata के द्वारा Accept किया जाता है।
- Also Known As- Context free Language, Push Down Automata.

#### D. Type-3 grammar $\Rightarrow$

- Regular Grammar
  - Type-3 grammar के द्वारा define की गई Languages को Finite State Automata के द्वारा Accept किया जाता है।
  - Regular Grammar Left या Right Linear को follow करती है।
- Ex  $\Rightarrow$      $A = aB$  - Right Linear  
               $A = Ba$  - Left Linear
- Also known as - Regular language , Finite Automata

## \* Finite Automata (FA) $\Rightarrow$

- Machines with fixed amount of unstructured memory, accepts regular languages.
- Applications  $\Rightarrow$  useful for modeling chips, communication protocols, Adventure games, some control systems etc.

## \* Push Down Automata (PDA) $\Rightarrow$

- Finite automata with unbounded structured memory in the form of a pushdown stack.
- Applications  $\Rightarrow$  useful for modeling parsing, compilers, programming languages design etc.

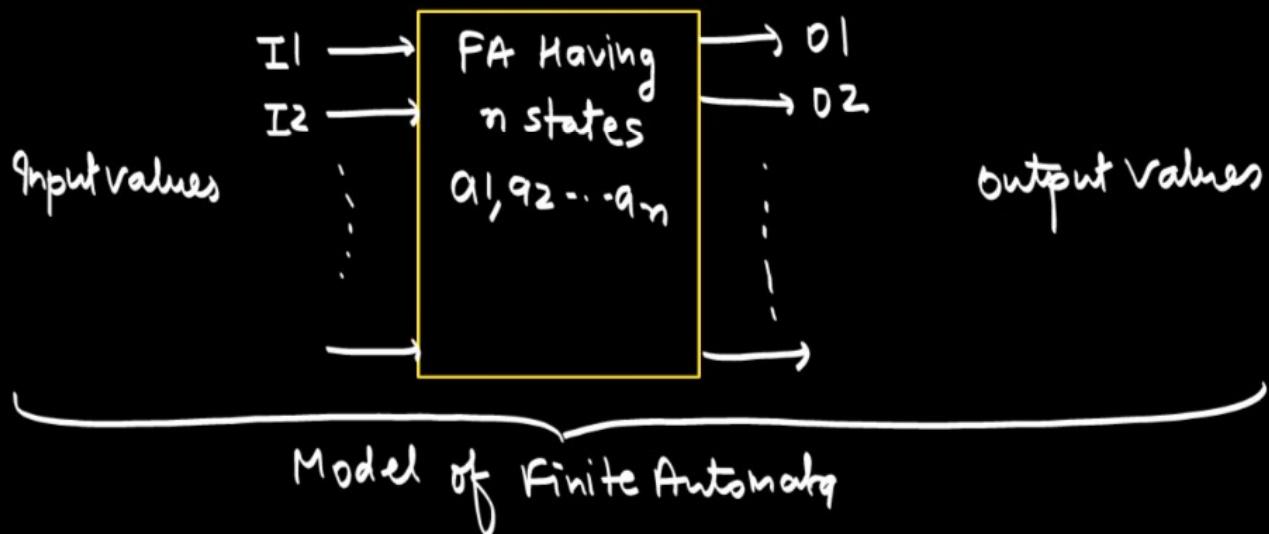
## \* Turing Machine (TM) :-

- Finite Automata with unbounded tape accepts or enumerable recursively language.
- Equivalent to RAM & various programming languages model.
- Applications ⇒ Model for general sequential computing

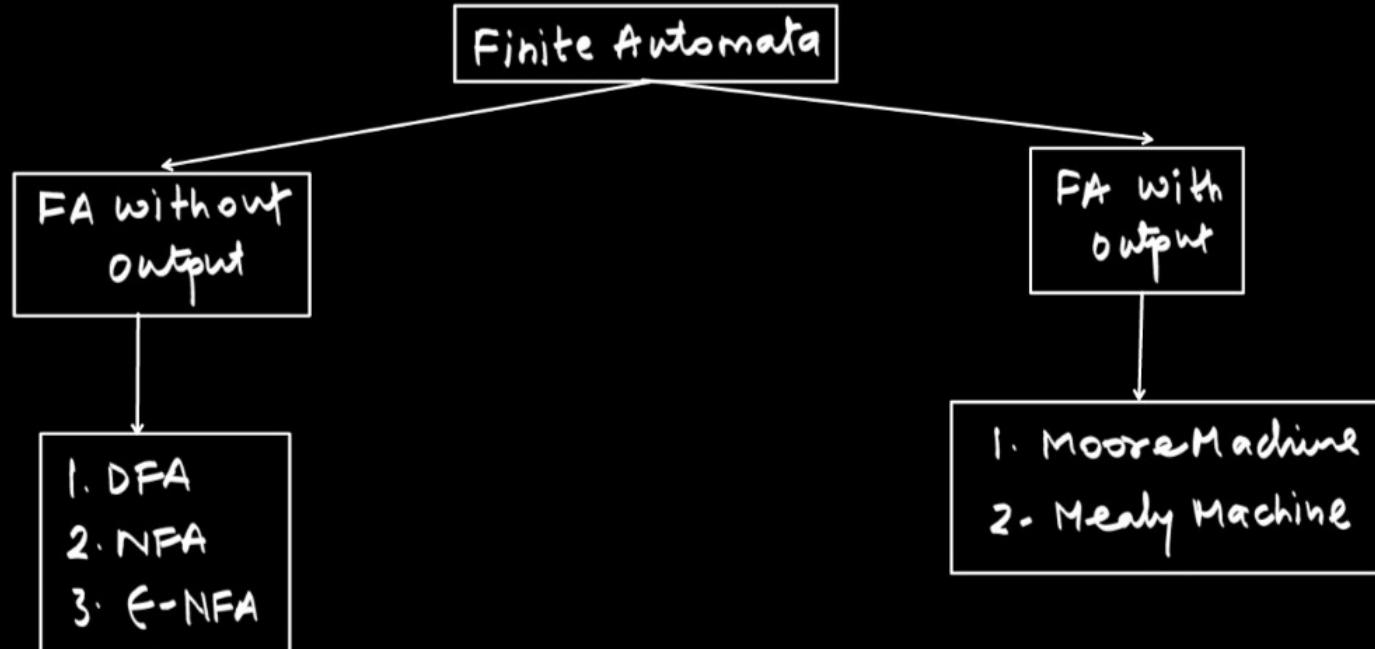
\* Regular Expressions & Finite Automata ⇒

\* Finite Automata ⇒

→ A Finite state machine (FSM) or finite state automata (FSA) is an abstract machine used in the study of computation and language that has only a finite & constant amount of memory.

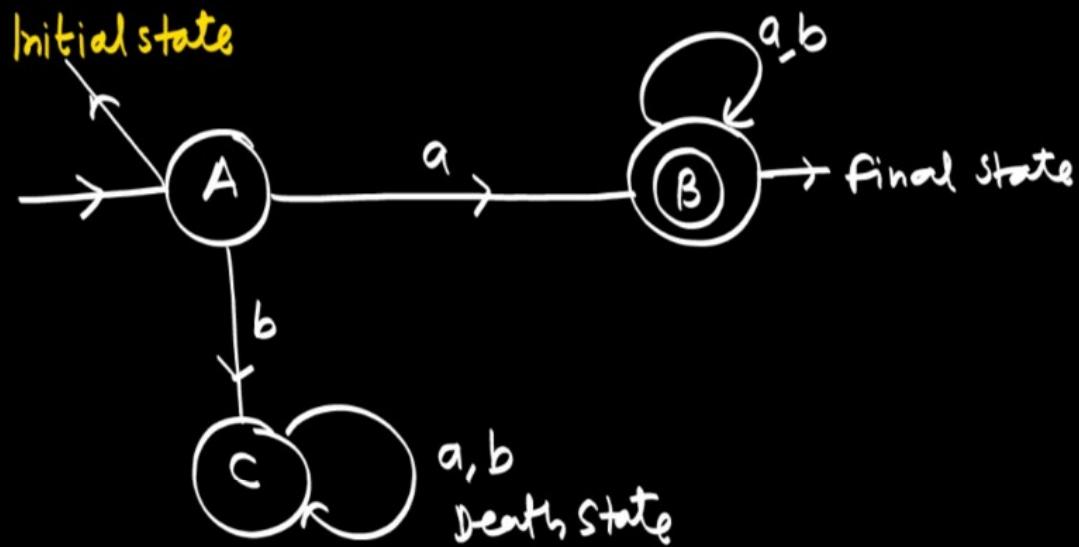


## \* Types of Finite Automata $\Rightarrow$



\* DFA (Deterministic Finite Automata)  $\Rightarrow$

→ A finite automata is defined as five tuples  $(Q, \Sigma, \delta, F, q_0)$ .



where  
 $Q = \text{set of All states } (A, B, C)$

$\Sigma = \text{Inputs } (a, b)$

$q_0 = \text{Initial state } 'A'$

$F = \text{final states } 'B'$

$\delta = \text{Transmission function}$

## ★ NFA ⇒

- Non Deterministic Finite Automata
- प्रेरित DFA के जैसा ही होता है, बल्कि कुछ Additional features होते हैं -
- NFA में NULL ( $\epsilon$ ) की move करना Allowed होता है, अर्थात् पहले किसी भी symbol को Read किये बिना आगे move हो सकता है।
- NFA में किसी विशेष input के लिए Machine किसी भी state में move हो सकती है अर्थात् उस state को identify नहीं कर सकते हैं, जिसमें machine move होती है।
- NFA में additional features add on होने के बाद जी NFA पर कोई power add नहीं होता है, इसलिए NFA व DFA Power Equivalent ही होते हैं।
- इन ही additional features के कारण प्रत्येक DFA, NFA हो सकता है परन्तु प्रत्येक NFA, DFA नहीं हो सकता है।

## \* DFA v/s NFA =>

DFA	NFA
<ol style="list-style-type: none"> <li>Each DFA can be NFA.</li> <li>DFA &amp; NFA में same power होने के कारण DFA को NFA में Translate किया जा सकता है।</li> <li>Both can have multiple final states.</li> <li>Compilers इसका Use Lexical Analysis में करता है।</li> <li>Backtracking Possible.</li> <li>DFA में Space की ज्यादा जरूरत नहीं है।</li> </ol>	<ol style="list-style-type: none"> <li>Each NFA can't be DFA.</li> <li>DFA &amp; NFA की power same होने के बाद भी NFA के Additional features के कारण इसे DFA में Translate नहीं किया जा सकता है।</li> <li>Both can have multiple final states.</li> <li>Compiler doesn't use NFA.</li> <li>Backtracking is not always possible.</li> <li>NFA में Space की कम जरूरत नहीं है।</li> </ol>

## \* $\epsilon$ -NFA $\Rightarrow$

- NFA में  $\text{NULL}(\epsilon)$  की move करना allowed होता है लेकिन यह किसी भी symbol की Read किये बिना भागे बढ़ सकता है, इसे  $\epsilon$ -NFA कहते हैं।
- Use - Regular Language को identify करने में use.
- यह NFA को more powerful बनाता है, क्योंकि यह बिना input के भी Transmit कर सकता है।

## \* Moore v/s Mealy Machine $\Rightarrow$

### 1. Basic definition $\Rightarrow$

- Moore Machine का Output केवल Current State पर depend करता है, जबकि Mealy Machine का Output Current state + Input दोनों पर depend करता है
- Moore Machine का Output Tab, States से Associated होता है, जबकि Mealy Machine का Output Tab, Inputs से Associated होता है

### 2. Structure/Components $\Rightarrow$

- Common = State ( $Q$ )
- Initial State ( $q_0$ )
- Output Alphabet ( $\Lambda$ )
- Input Alphabet ( $\Sigma$ )
- Transition function ( $\delta$ )

Different = Output function ( $\lambda$ ) =  $Q \rightarrow \Lambda$   
(Moore Machine)

O/P function ( $\lambda$ ) =  $Q \times \Sigma \rightarrow \Lambda$   
(Mealy Machine)

### 3. Examples $\Rightarrow$

→ यदि Moore Machine में current state =  $q_1$  हो, तो output = 1 होगा, जबकि input कुछ नहीं हो।

State	Output
$q_0$	0
$q_1$	1
$q_2$	0

→ यदि Mealy Machine के current state  $q_1$  हो तभी input = 0 हो, तो output = 1 होगा और यदि current state =  $q_1$  & input = 1 हो, तो output = 0 होगा।

Transition	Output
$q_1, 0$	1
$q_1, 1$	0

#### 4. Speed of Output $\Rightarrow$

$\rightarrow$  Moore Machine का Output रक्क state change होने के बाद मिलता है, इसलिए Slow होता है, जबकि Mealy Machine में Input के साथ ही output आता है, इसलिए Fast है।

#### 5. Number of states $\Rightarrow$

$\rightarrow$  Moore Machine में states क्षयित होती है, क्योंकि Output, States से linked होता है जबकि Mealy Machine में states कम होती है, क्योंकि Output, Transitions से linked होता है।

#### 6. Conversion $\Rightarrow$

$\rightarrow$  Moore to Mealy Possible but States कम होते हैं तभ्या Mealy to Moore Conversion भी possible है, लेकिन States क्षयित होते हैं।

## \* Regular Expressions (RE) $\Rightarrow$

- इसी Method द्वारा का use करके Regular Languages को define किया जाता है।
- RE, Symbols & Operators का use करके strings के pattern को define करता है।

जैसे -  $a|b = \frac{a \text{ or } b}{RL}$

$a^*$  = Zero or More

- Precedence (Highest to Lowest) =
  - A. Kleene Star (\*)
  - B. Concatenation
  - C. Union (|)

## \* Turing Machine ⇒

- मह एक Mathematical Model of computation होता है, जो किसी भी Algorithm को Simulate कर सकता है।
- By - Alan Turing (1936)
- मह एक Theoretical computation Machine है, जो मह दिखाती है कि Computer किस Problem को Solve कर सकता है और किसे बटी।

## \* Structure of Turing Machine →

### A. Tape →

- infinite strip जिस पर Input दिया जाता है।
- Computation के दौरान Machine जिसका / Erase भी कर सकती है।

### B. Tape Head →

- Pointer जो Current cell पर होता है।
- यह Read/Write तथा Move (left/right) हो सकता है।

### C. Finite Control / States

→ एक finite set of states ( $q_0, q_1, q_2 \dots$ ) जो Machine के current status को विवरते हैं।

#### D. Transition Function $\Rightarrow$

- यह decide करता है कि किस input symbol पर क्या Action होगा |
- $\delta$  (delta)

#### \* ID $\Rightarrow$

- Instantaneous Description
- एक ID describe करता है, कि Machine किस state में है, Tape पर क्या मिर्का हुए है और Head किस Position पर है।

## \* Languages accepted by TM $\Rightarrow$

- एक Turing Machine एक language Accept करती है, अगर Input string को process करने के बाद Turing Machine Accept State में पहुँचती है।
- Turing Machine में Acceptance के 2 Modes होते हैं-
  - A. Acceptance by final state
  - B. Acceptance by Halting state
- TM Recursively Enumerable (RE) या Type 0 language को accept करती है।

## \* Variants of TM $\Rightarrow$

1. Multi Tape TM  $\Rightarrow$  Multiple Tapes but equivalent to Single Tape TM
2. Multi Track TM  $\Rightarrow$  एक ही Tape पर Multiple Tracks. (Parallel)
3. Non Deterministic TM  $\Rightarrow$  एक Input पर Multiple Tracks Possible.
4. Two way Infinite Tape TM  $\Rightarrow$  Infinite Tape on both sides.

$\Rightarrow$  इन सभी का Computational Power, Standard Turing Machine के जैसा ही होता है।

## \* Universal Turing Machine ⇒

- ऐसी TM जो किसी भी अन्य TM के Behaviour को Simulate कर सकती है।
- इसमें Input + Program (Type of TM) दोनों डिप्टे जाते हैं।
- पट General Purpose Computer के Concept का Base होता है।

## \* Church's Thesis ⇒

- Statement = "whatever is computable by an effective method is computable by TM."
- अर्थात् कोई भी Arithmetic problem जो solve हो सकती है, उसके लिए एक Turing Machine बनाई जा सकती है।

\* PDA  $\Rightarrow$

→ Push Down Automata

→ यह एक प्रकार का Finite Automation होता है, जिसके पास एक Stack होता है और इस Stack का use करके यह Machine अधिक Complex languages की Recognize करती है।

जैसे - Balanced Parenthesis, Palindroms etc.

## \* PDA V/S Finite Automata $\Rightarrow$

Features	Finite Automata	PDA
1. Memory	No External Memory	Stack
2. Power	Less Powerful (Regular Languages)	More Powerful (Context-free languages)
3. Example.	String ending with "ab".	Balanced Parenthesis & Palindromes

## \* Working of PDA $\Rightarrow$

$\rightarrow$  PDA जब किसी Input को Read करता है तो साथ में Symbols को stack में Push भी pop भी करता है।

$\rightarrow$  PDA 2 प्रकार से किसी string को Accept कर सकता है -

A. By Final state - जब input finish हो जाए और Machine final state में पहुँच जाए।

B. By Empty stack - जब input finish हो जाए और stack empty हो जाए।

\* मेरे दोनों Methods equivalent ही होती हैं, इनमें से किसी का भी प्रयोग करके Some languages को Recognize किया जा सकता है।

Parsing  
LR-0P

\* DPDA v/s NPDA  $\Rightarrow$

$\rightarrow$  Deterministic PDA & Non-Deterministic PDA

एक ही Possibility  
एक step पर

एक से ज्यादिक Possibilities  
ग्रेटर step पर

$\rightarrow$  NPDA ज्यादा Powerful होता है, जबकि यह क्षमता languages को Accept कर सकता है, जबकि DPDA कुछ ही languages को Accept कर सकता है।