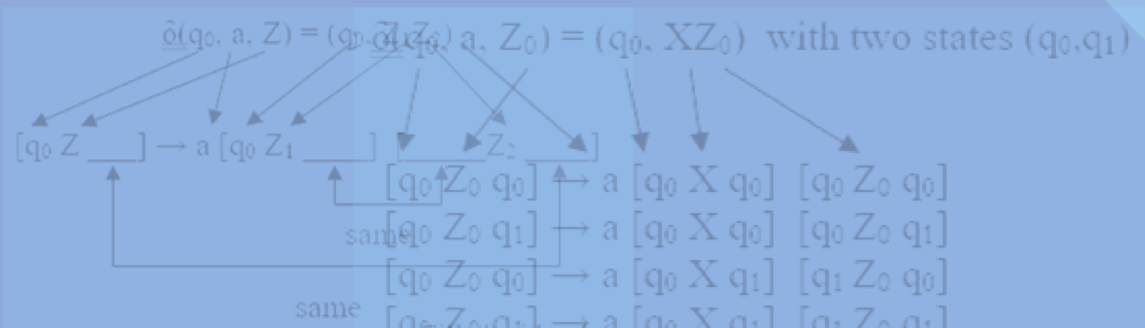
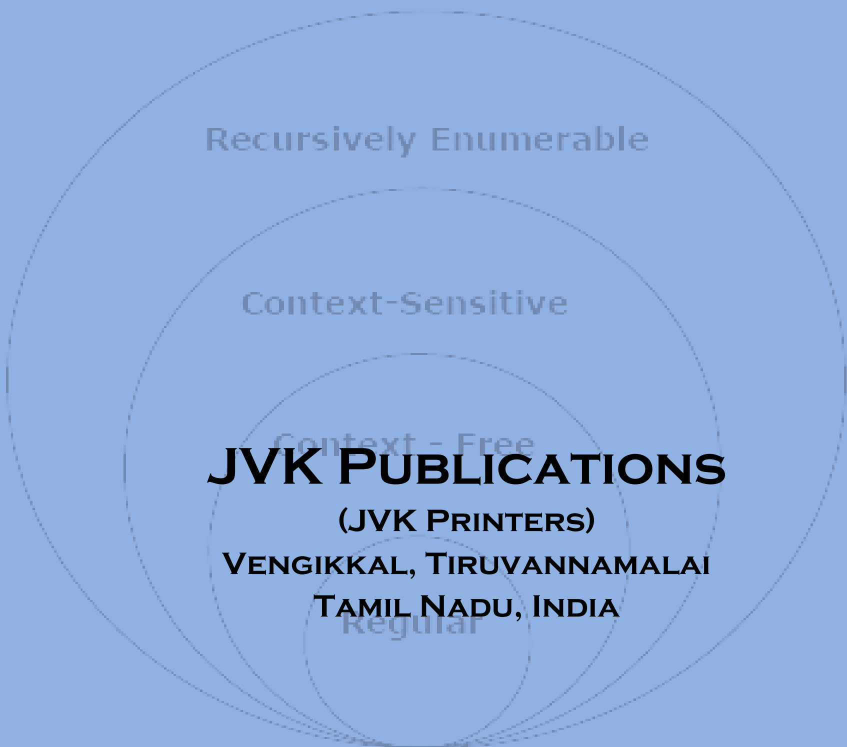
# Theory of Computation



Dr. D. Jagadeesan




JVK Publications
(JVK Printers)
Vengikkal, Tiruvannamalai
Tamil Nadu, India

# Theory of Computation

**Dr. D. Jagadeesan**
The Apollo University, Chittoor, Andhra Pradesh

**JVK Publications**
**(JVK Printers)**
**Vengikkal, Tiruvannamalai**
**Tamil Nadu, India**

JVK Printers
1799, Sri Raghavendra Nagar,
Vengikkal, Tiruvannamalai,
Tamil Nadu, India - 606604



**Price: ₹150.00**

**First Edition, 2024**
ISBN 978-93-5620-409-6

ISBN 978-93-5620-409-6
9 789356 204096

# INDEX

CHAPTER 1

# FINITE AUTOMATA

Formal languages and Automata Theory describes the basic ideas and models underlying computing. It suggests various abstract models of computation, represented mathematically.

## 1.1 History

Mathematicians and logicians made major contributions to the field of finite automata as early as the 20th century. The concept was first introduced by David Hilbert in 1927, but it was the work of Alonzo Church and Alan Turing in the 1930s that laid the theoretical foundation for finite automata as models of computation. In the mid-20th century, the renowned mathematician and computer scientist John von Neumann made substantial contributions to automata theory. The formalization of deterministic and nondeterministic finite automata emerged in the 1950s, with notable contributions from Michael O. Rabin and Dana Scott. Finite automata became integral to theoretical computer science, playing a crucial role in formal language theory and compiler design. The development of finite automata reflects a collaborative effort by pioneers in mathematics and computer science to understand the fundamental principles of computation and lay the groundwork for subsequent advances in the field.

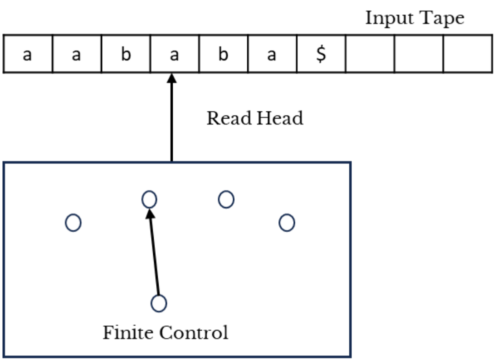
## 1.2 Finite State systems

A finite automaton can also be thought of as the device shown below consisting of a tape and a control circuit which satisfy the following conditions:

- ✓ The tape has the left end and extends to the right without an end.
- ✓ The tape is dividing into squares in each of which a symbol can be written prior to the start of the operation of the automaton.
- ✓ The tape has a read only head.

- ✓ The head is always at the leftmost square at the beginning of the operation.
- ✓ The head moves to the right one square every time it reads a symbol.

  It never moves to the left. When it sees no symbol, it stops and the automaton terminates its operation.
- ✓ There is a finite control which determines the state of the automaton and also controls the movement of the head.

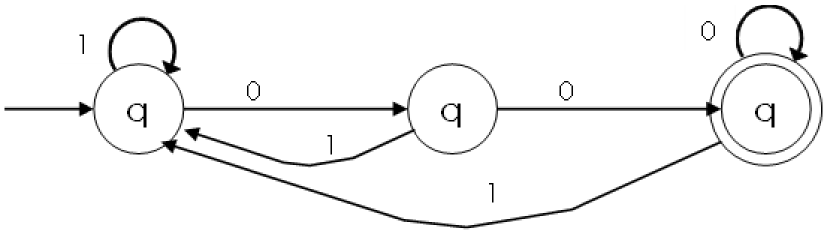



Finite Automaton

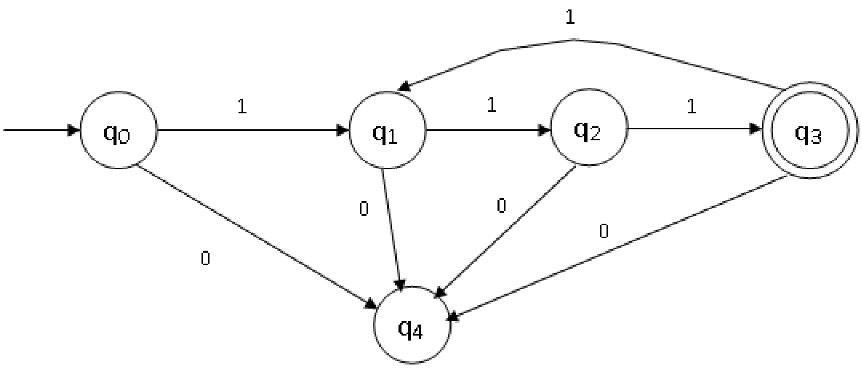
## 1.3 Applications of Finite Automata

- ✓ Finite automata are extensively used in compilers for lexical analysis, the first phase of language processing.
- ✓ It is useful in text processing and searching applications.
- ✓ Finite state machines are utilized to model and analyze network protocols.
- ✓ Finite state machines are used to design and model digital circuits.
- ✓ Finite automata contribute to pattern recognition tasks, where recognizing specific sequences or patterns in data is essential.
- ✓ Finite state machines are employed to model and control the behavior of robots.
- ✓ Finite state machines play a crucial role in designing the control units of digital systems within these circuits.
- ✓ Finite state machines and automata models are used in designing decision-making processes for AI agents.
- ✓ Finite state machines are commonly used in game development for modeling the behavior of non-player characters.

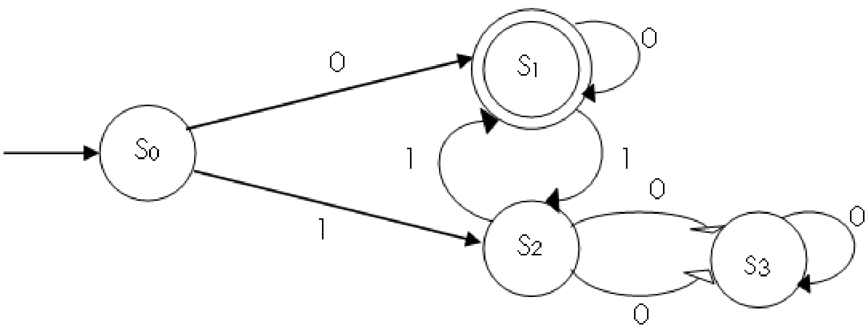
**3. Design FA to accept the string that always ends with 00.**



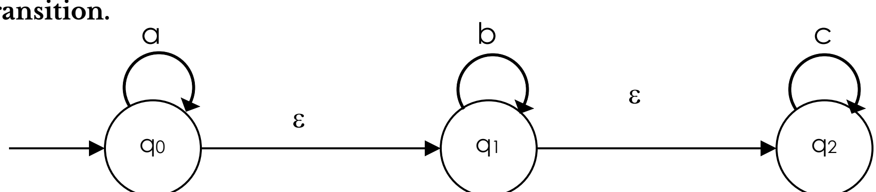

**4. Design FA to check whether a given unary number is divisible by 3.**




**5. Design FA to check whether a given binary number is divisible by 3.**




**6. Obtain the ε closure of states q0 and q1 in the following NFA with ε transition.**




**Solution:**

ε - CLOSURE {q0} = {q0, q1, q2}

ε - CLOSURE {q1} = {q1, q2}

Now we will show that

$\delta'(p,a) = \delta(q0,wa)$

But,

$\delta'(p,a) = \delta'(q,a) = \delta''(q,a)$ as $p = \delta''(q0,w)$

We have

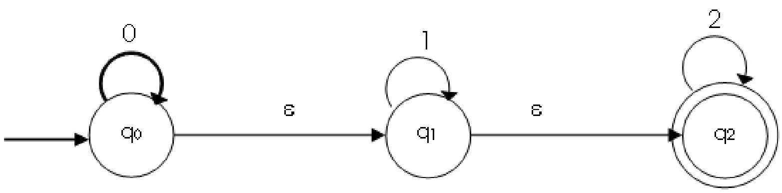
$\delta''(q,a) = \delta''(q0,wa)$

Thus, by definition of $\delta''$

$\delta'(q0, wa) = \delta''(q0,wa)$

## 1.9.1 Problems for Converting NFA with ε into NFA without ε

**1. Construct NFA without ε from NFA with ε.**




Solution:

Find the ε – closure of all states:

**ε – closure (q0) = {q0, q1, q2}**

**ε – closure (q1) = {q1, q2}**

**ε – closure (q2) = {q2}**

ε – closure (q0)
= { q0,q1,q2}

Compute δ' function:

**δ'(q0,0)** = δ'' (q0,0) = ε – closure (δ(δ'(q0,ε),0))
= ε – closure (δ({q0,q1,q2},0))
= ε – closure (q0) = **{q0,q1,q2}**

**δ'(q0,1)** = δ'' (q0,1) = ε – closure (δ(δ'(q0,ε),1))
= ε – closure (δ({q0,q1,q2},1))
= ε – closure (q1) = **{q1,q2}**

**δ'(q0,2)** = δ'' (q0,2) = ε – closure (δ(δ'(q0,ε),2))
= ε – closure (δ({q0,q1,q2},2))
= ε – closure (q2) = **{q2}**

**δ'(q1,0)** = δ'' (q1,0) = ε – closure (δ(δ'(q1,ε),0))
= ε – closure (δ({q1,q2},0))
= ε – closure (ϕ) = **{ϕ}**

CHAPTER 2

# REGULAR EXPRESSION

A regular expression plays a crucial role in describing languages. Let's explore what regular expressions are and how they relate to regular languages.

## 2.1 Regular Language

A language is called regular language if there exists a finite automaton that recognizes it. For example, finite automaton M recognizes the language L if L = {w | M accepts w}.

### 2.1.1 Operations on Regular Languages

Let A and B be languages. We define regular language operations union, concatenation and closure as follows:

| | |
|---|---|
| Union | : $A \cup B = \{x \mid x \in A \lor x \in B\}$ |
| Concatenation | : $A \circ B = \{xy \mid x \in A \land y \in B\}$ |
| Closure | : $A^* = \{x1x2 \ldots xk \mid k \geq 0 \land xi \in A, 1 \leq i \leq k\}$ |

## 2.2 Regular Expression

A regular expression is a string that defines a finite pattern of strings or symbols. Each pattern corresponds to a set of strings, serving as a name for that set.

Regular expressions are used to describe languages, and they allow us to express rules for constructing valid strings within those languages.

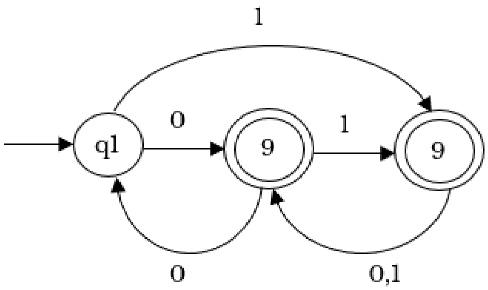
### 2.2.1 Operations on Regular Expressions

Let Σ be an alphabet. The regular expressions over Σ and the sets that they denote are defined recursively as follows:

- ✓ Ø is a regular expression and denotes the empty set {}.
- ✓ ε is a regular expression and denotes the set {ε}.

**2.5.1.2 Problems**

1. **Covert the given finite automata into a regular expression using substitution method.**




Solution:

r = $r_{12}^{3} + r_{13}^{3}$

$$r_{ij}^{0} = \begin{cases} a1 + a2 + a3 + \cdots + an & [\delta(qi,a) = qj]\ \ if\ i \neq j \\ a1 + a2 + a3 + \cdots + an + \varepsilon & [\delta(qi,a) = qj]\ \ if\ i = j \end{cases}$$

<u>K= 0</u>

| | | |
|---|---|---|
| $r_{11}^{0} = \varepsilon$ | $r_{21}^{0} = 0$ | $r_{31}^{0} = \emptyset$ |
| $r_{12}^{0} = 0$ | $r_{22}^{0} = \varepsilon$ | $r_{32}^{0} = 0 + 1$ |
| $r_{13}^{0} = 1$ | $r_{23}^{0} = 1$ | $r_{33}^{0} = \varepsilon$ |

<u>K= 1</u>

$r_{ij}^{k} = r_{ij}^{k-1} + r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1}$

$r_{11}^{1} = r_{11}^{0} + r_{11}^{0} (r_{11}^{0})^* r_{11}^{0}$ = ε + ε (ε)*ε = ε+ε = **ε**

$r_{12}^{1} = r_{12}^{0} + r_{11}^{0} (r_{11}^{0})^* r_{12}^{0}$ = 0+ ε (ε)* 0 = 0+0=**0**

$r_{13}^{1} = r_{13}^{0} + r_{11}^{0} (r_{11}^{0})^* r_{13}^{0}$ = 1 + ε (ε)*ε 1 = 1+1 =**0**

$r_{21}^{1} = r_{21}^{0} + r_{21}^{0} (r_{11}^{0})^* r_{11}^{0}$ = 0+ 0 (ε)*ε = 0+0=**0**

$r_{22}^{1} = r_{22}^{0} + r_{21}^{0} (r_{11}^{0})^* r_{12}^{0}$ = ε + 0 (ε)*0 = **ε+00**

$r_{23}^{1} = r_{23}^{0} + r_{21}^{0} (r_{11}^{0})^* r_{13}^{0}$ = 1 + 0 (ε)*1 = **1+01**

CHAPTER 3

# REGULAR GRAMMAR

**Language**: “A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols.”

**Grammar:** “A grammar can be regarded as a device that enumerates the sentences of a language.”

A formal grammar is a quad-tuple G = (N, T, P, S)

where

N is a finite set of non-terminals

T is a finite set of terminals

P is a finite set of production rules of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$

Where $\alpha, \beta, \gamma \in (N \cup T)*$, $A \in N \neq \varepsilon$

$S \in N$ is the start symbol

## 3.1 Chomsky Hierarchy (Type of Grammars)

| Class | Grammars | Languages | Automaton | Rules |
|---|---|---|---|---|
| Type-0 | Unrestricted Grammar | Recursively Enumerable Language | Turing Machine | $\boldsymbol{\alpha \rightarrow \beta}$<br>$\alpha, \beta \in (N \cup T)*$<br>$\alpha \neq \varepsilon$ |
| Type-1 | Context Sensitive Grammar | Context Sensitive Language | Linear Bounded Automaton | $\boldsymbol{\alpha A \beta \rightarrow \alpha \gamma \beta}$<br>$\alpha, \beta, \gamma \in (N \cup T)*$<br>$A \in N \neq \varepsilon$ |
| Type-2 | Context-free Grammar | Context-free Language | Pushdown automaton | $A \rightarrow \alpha$<br>where $A \in N$<br>$\alpha \in (N \cup T)*$ |
| Type-3 | Regular Grammar | Regular Language | Finite automaton | $A \rightarrow \alpha B$<br>$A \rightarrow B \alpha$<br>$A \rightarrow \alpha$<br>where $A, B \in N$ and<br>$\alpha \in T*$ |

CHAPTER 4

# CONTEXT FREE GRAMMAR

A context-free grammar is a collection of recursive rules employed to produce patterns of strings. A context-free grammar is capable of describing all regular languages and additional languages, but not all possible languages. Context-free grammars are an area of study in the fields of theoretical computer science, compiler design and linguistics.

## 4.1 Motivation and Introduction

Formal language theory and computer science use a context-free grammar (CFG) as a formalism to describe a language's syntax. A set of production rules define the replacement of symbols or non-terminals with sequences of other symbols and terminals.

A Context Free Grammar is consisting of four components. They are finite set of non-terminals, finite set of terminals, set of productions and start symbol.

## 4.2 Formal Definition of Context Free Grammars (CFG)

- A CFG is a mathematical object, G, with four components,
  G = (N, T, P, S)

  where

  N is a nonempty, finite set of non-terminal symbols

  T is a finite set of terminal symbols

  P is a set of grammar rules as per below form

  $A \rightarrow \alpha$ where $A \in N$ and $\alpha \in (N \cup T)^*$

  S is the start symbol $S \in N$

- Example

  Let G = ({S},{0,1,ε},P,S) be a CFG, where productions are

  $S \rightarrow 0S0/1S1/\varepsilon$

CHAPTER 5

# PUSHDOWN AUTOMATA

A pushdown automaton (PDA) is a type of automaton, which is a mathematical model of computation. PDAs are used to recognize context-free languages, which are languages generated by context-free grammars. A PDA consists of Input tape, Finite control, Stack. The transitions of a PDA are determined by the current state, the symbol read from the input tape, and the symbol popped from the stack. Based on these factors, the PDA can change its state, push symbols onto the stack, or pop symbols from the stack. PDAs can be in one of three states: accepting, rejecting, or non-accepting. The PDA accepts the input string if, after processing the entire string, it is in an accepting state. Otherwise, it rejects the input string.

## 5.1 Formal Definition of Pushdown Automaton

A pushdown automaton consists of seven tuples

$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where

- Q - Finite set of states
- Σ - Finite input alphabet
- Γ - Finite alphabet of pushdown symbols
- δ - Transition function $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma$
- $q_0$ - start / initial state $q_0 \in Q$
- $Z_0$ - start symbol on the pushdown $Z_0 \in \Gamma$
- F - set of final states $F \in Q$

## 5.2 Model of PDA

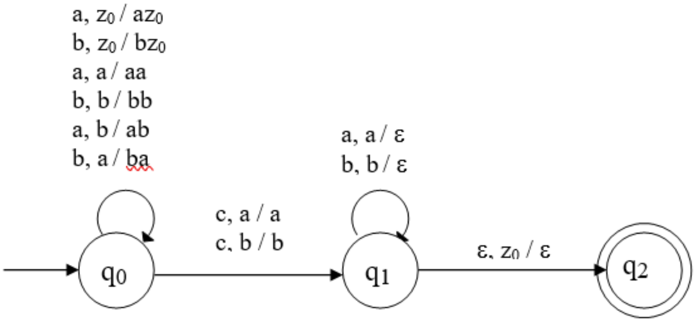
Pushdown Automata is a finite automaton with extra memory called stack which helps Pushdown automata to recognize Context Free Languages. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.

The PDA consists of a finite set of states, a finite set of input symbols and a finite set of push down symbols. The finite control has control of both the input tape and the push down store. The stack head scans the top symbol of the stack.

9. $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$
10. $\delta(q_1, b, b) = \{(q_1, \varepsilon)\}$ } Pop operations

11. $\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, \varepsilon)\}$ Accept the empty stack

Transition Diagram:




7. Design a PDA that accepts $L = \{ww^R ; w \in (0+1)^*\}$ accepted by final state. (or) Design a PDA for even length palindrome.

Solution:

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA

The productions are:

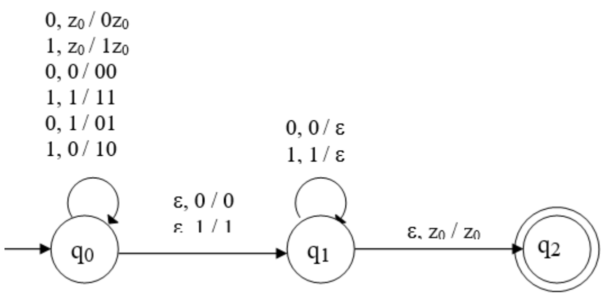
1. $\delta(q_0, 0, z_0) = \{(q_0, 0z_0)$
2. $\delta(q_0, 1, z_0) = \{(q_0, 1z_0)$
3. $\delta(q_0, 0, 0) = \{(q_0, 00)$
4. $\delta(q_0, 1, 1) = \{(q_0, 11)$
5. $\delta(q_0, 0, 1) = \{(q_0, 01)$
6. $\delta(q_0, 1, 0) = \{(q_0, 10)$ } Push operations

7. $\delta(q_0, \varepsilon, 0) = \{(q_1, 0)\}$
8. $\delta(q_0, \varepsilon, 1) = \{(q_1, 1)\}$ } Accept the separator 'ε'

9. $\delta(q_1, 0, 0) = \{(q_1, \varepsilon)\}$
10. $\delta(q_1, 1, 1) = \{(q_1, \varepsilon)\}$ } Pop operations

11. $\delta(q_1, \varepsilon, z_0) = \{(q_2, z_0)\}$ Accept the Final State

Transition Diagram:



CHAPTER 6

# TURNING MACHINE

A Turing machine (TM) is a theoretical model of computation introduced by Alan Turing in 1936. It is an accepting device which accepts the languages (recursively enumerable set) generated by type 0 grammars.

A Turing Machine (TM) is a mathematical model which consists of

- An **infinite length tape** divided into cells; each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol (here written as '0') and one or more other symbols. The tape is assumed to be arbitrarily extendable to the left and to the right.
- A **head** which reads the input tape.
- A **state** register stores the state of the Turing machine.
- After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or **left**. If the TM reaches the final state, the input string is accepted, otherwise rejected.

## 6.1 Formal Definition of Turing Machine

A TM can be formally described as a 7-tuple M = $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where

Q is a finite set of states
$\Sigma$ is the input alphabet
$\Gamma$ is the tape alphabet
$\delta$ is a transition function; $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.
$q_0$ is the initial state, $q_0 \in Q$
B is the blank symbol, $B \in \Gamma$
F is the set of final states, $F \in Q$

## 6.2 Model of Turing Machine (TM)

Turing Machine has three components:

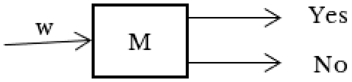
i. **Finite state control**:
   - It is in one of a finite number of states at each instant, and is connected to the tape head.

CHAPTER 7

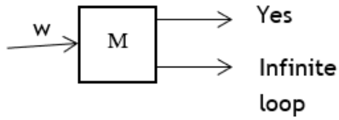
# UNDECIDABILITY

Undecidability is a concept in mathematics and computer science that refers to situations where there is no algorithmic way to determine whether a given statement is true or false within a particular formal system. Undecidability is a concept in mathematics and computer science that refers to situations where there is no algorithmic way to determine whether a given statement is true or false within a particular formal system. This concept often arises in the study of formal languages, logic, and computational theory. Alan Turing famously proved the halting problem, a classic example of undecidability, in 1936. The halting problem asks whether a given program, when provided with a particular input, will eventually halt (i.e., stop running). Turing showed that it is impossible to write a general algorithm that can decide whether any arbitrary program will halt or run forever.

**Recursive Language:** A language is recursive if there exists a Turing Machine that accepts every string of the language and reject every string that is not in the language.




**Recursively Enumerable Language:** A language is recursive enumerable if there exists a Turing Machine that accepts every string of the language and does not accept strings that are not in the language. The strings that are not in the language may be rejected and it may cause the TM to go to an infinite loop.



UPI
UNIFIED PAYMENTS INTERFACE

Accepted here

Scan and Pay using bob world

*send payment proof  to 7904647133 after verification of payment publisher will send the softcopy of book

ISBN 978-93-5620-409-6

9 789356 204096

**JVK PUBLICATIONS**

**(JVK PRINTERS)**

**VENGIKKAL, TIRUVANNAMALAI**

**TAMIL NADU, INDIA**