Program : **B.Tech**

Subject Name: **Theory of Computation**

Subject Code: **CS-501**

Semester: **5th**

Subject Notes
CS501- Theory of Computation
B.Tech, CSE-5th Semester

Syllabus: Introduction of Automata Theory: Examples of automata machines, Finite Automata as a language acceptor and translator, Moore machines and mealy machines, composite machine, Conversion from Mealy to Moore and vice versa.

Unit-I: Introduction of Automata Theory

Automata

The term "Automata" is derived from the Greek word "αὐτόματα" which means "self-acting". An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

Automata are computational devices to solve language recognition problems. Language recognition problem is to determine whether a word belongs to a language.

Automaton = abstract computing device, or "machine".

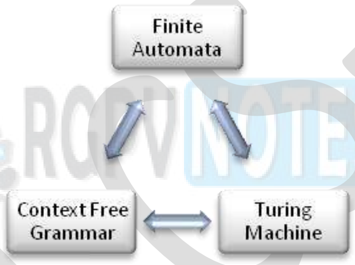


Figure 1.1: Computational Model of Automata Theory

Examples of automata machines:

- **Sequential machine:** A sequential machine is a mathematical model of a certain type of simple computational structure. Its behavior represents the working process of finite Automata.
- **Vending Machines:** A vending machine is an automated machine that dispenses numerous items such as cold drinks, snacks, beverages, alcohol etc. Vending machine is works on finite state automate to control the functions process.
- **Traffic Lights:** The optimization of traffic light controllers in a city is a systematic representation of handling the instructions of traffic rules. Its process depends on a set of instruction works in a loop with switching among instruction to control traffic.
- **Video Games:** Video games levels represent the states of automata. In which a sequence of instructions is followed by the players to accomplish the task.
- **Text Parsing:** Text parsing is a technique which is used to derive a text string using the production rules of a grammar to check the acceptability of a string.
- **Regular Expression Matching:** It is a technique to checking the two or more regular expression are similar to each other or not. The finite state machine is useful to checking out that the expressions are acceptable or not by a machine or not.
- **Speech Recognition:** Speech recognition via machine is the technology enhancement that is capable to identify words and phrases in spoken language and convert them to a machine-readable format.

Receiving words and phrases from real world and then converted it into machine readable language automatically is effectively solved by using finite state machine.

-

Characteristics:

- FA is having only a finite number of states.
- Finite Automata can only "count" (that is, maintain a counter, where different states correspond to different values of the counter) a finite number of input scenarios.
- Able to solve limited set of problem
- Outcome in the form of "yes "or "No" (Accept / Reject)

Limitation:

There is no finite automaton that recognizes these strings:

- The set of binary strings consisting of an equal number of 1's and 0's
- The set of strings over '(' and ')' that have "balanced" parentheses.

Applications of TOC:

- Finite State Programming
- Event Driven Finite State Machine (FSM)
- Virtual FSM
- DFA based text filter in Java
- Acceptors and Recognizers
- Transducers
- UML state diagrams
- Hardware Application

Finite Automata:

An automaton with a finite number of states is called a Finite Automaton (FA) or Finite State Machine (FSM).

An automaton has a mechanism to read input, which is string over a given alphabet. This input is actually written on an input tape /file, which can be read by automaton but cannot change it. Input file is divided into cells each of which can hold one symbol. Automaton has a control unit which is said to be in one of finite number of internal states.

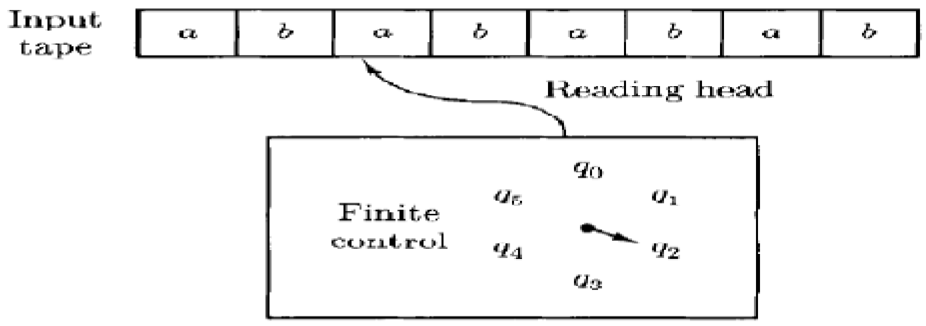


Figure 1.2: Finite Automata

Formal definition of Finite Automata:

A finite automaton is a 5-tuple M = (Q, Σ, δ, q, F), where

1. Q is a finite set, whose elements are called states,
2. Σ is a finite set, called the alphabet; the elements of Σ are called symbols,
3. δ: Q × Σ → Q is a function, called the transition function,
4. q is an element of Q; it is called the start state or initial state,
5. F is a subset of Q; the elements of F are called accept states or final state.

Related Terminologies:

Alphabet: An alphabet is any finite set of symbols.

Example: Σ = {a, b, c, d} is an alphabet set where 'a', 'b', 'c', and d' are symbols.

String: A string is a finite sequence of symbols taken from Σ.

Example: 'cabcad' is a valid string on the alphabet set Σ = {a, b, c, d}

Length of a String: It is the number of symbols present in a string. (Denoted by |S|).

Examples: If S='cabcad', |S|= 6

If |S|= 0, it is called an empty string (Denoted by λ or ε)

Language: A language is a subset of Σ* for some alphabet Σ. It can be finite or infinite.

Example: If the language takes all possible strings of length 2 over Σ = {a, b}, then L = {ab, bb, ba, bb}

Classification of Finite Automata:

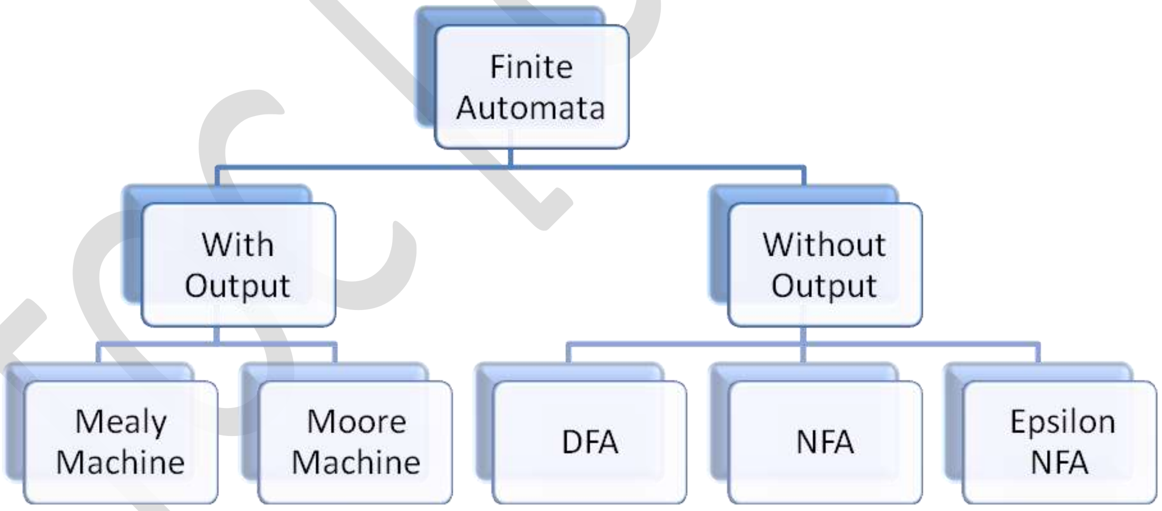


Figure 1.3: Classification of Finite Automata

An example of finite automata

Let A = {w: w is a binary string containing an odd number of 1s}.

We claim that this language A is regular. In order to prove this, we have to construct a finite automaton M such that A =L(M).

Steps to construct finite automata:

- The FA reads the input string w from left to right and keep scanning on the number of 1's
- After scanning the entire string, it counts the number of 1's to check whether it is odd or even
- If the number of 1's is odd then the string is acceptable otherwise it is rejected

Using this approach, the finite automaton needs a state for every integer i ≥ 0; indicating that the number of 1s read so far is equal to i. Hence, to design a finite automaton that follows this approach, we need an infinite number of states. But, the definition of finite automaton requires the number of states to be finite. A better, and correct approach, is to keep track of whether the number of 1s read so far is even or odd. This leads to the following finite automaton:

• The set of states is Q = {q_0, q_1}. If the finite automaton is in state q1, then it has an even number of 1's; if it is in state q0, then it has an odd number of 1's.

• The alphabet is Σ = {0, 1}.

• The start state is q0, because at the start, the number of 1's read by the automaton is equal to 0, and 0 is even.

• The set F of accept states is F = {q_1}.

• The **transition function δ** is given by the following table:

State	Input 0	Input 1
q_0	**q_0**	**q_1**
q_1	**q_1**	**q_0**

Table 1.1: Transition Table/Matrix

This finite automaton M = (Q, Σ, δ, q_0, F) can also be represented by its state diagram.

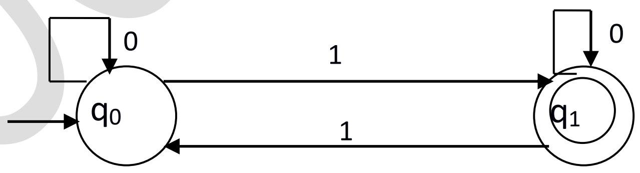


Figure 1.4: Transition Graph

Transition Graph: it is a finite directed labeled graph in which each vertex (or node) represent a state and the directed edges indicate the transition of state. Edges are labeled with input symbol.

Transition Matrix: It is two-dimension matrixes between states of automata and Input symbol. Elements of matrix are state form mapping (Σ X Q) into Q.

Finite state acceptor is a finite state machine with no outputs. The user of a finite state acceptor cares only about the final state: if the machine ends in an **accepting** state after processing a series of inputs, the machine is said to have **accepted** the input; otherwise, it is said to have **rejected** the input.

Description of Finite-State Machines using Graphs

Any finite-state machine can be shown as a graph with a finite set of nodes. The nodes correspond to the states. There is no other memory implied other than the state shown. The start state is designated with an arrow directed into the corresponding node, but otherwise unconnected.

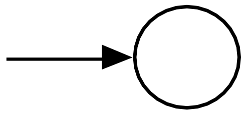


Figure 1.5: An unconnected in-going arc indicates that the node is the start state

The arcs and nodes are labeled differently, depending on whether we are representing a transducer, a classifier, or an acceptor. In the case of a **transducer**, the arcs are labeled as shown below, where q1 is the input symbol and q2 is the output symbol. The state- transition is designated by virtue of the arrow going from one node to another.

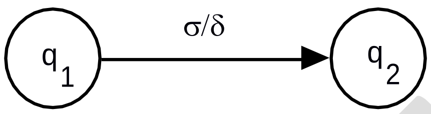


Figure 1.6: Transducer transition from q to q , based on input □, giving output □

In the case of an **acceptor**, instead of labeling the states with categories 0 and 1, we sometimes use a double-lined node for an accepting state and a single-lined node for a rejecting state.

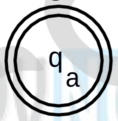


Figure 1.7: Acceptor, an accepting state

Acceptor Example

Let us give an acceptor that accepts those strings with exactly one edge. We can use the state transitions from the previous classifier. We need only designate those states that categorize there being one edge as accepting states and the others as rejecting states.

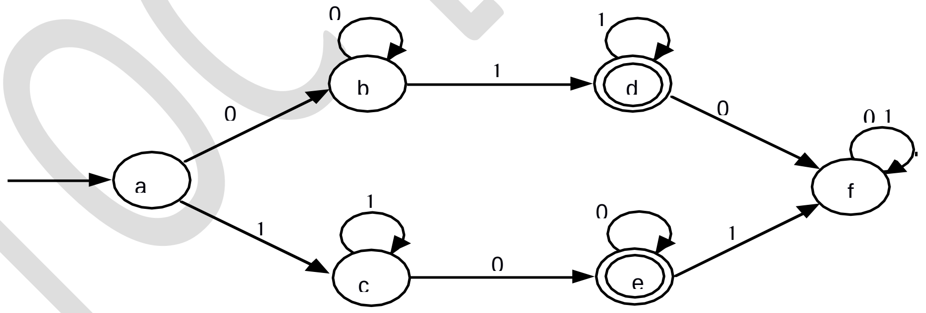


Figure 1.8: Acceptor for strings with exactly one edge. Accepting states are d and e.

Examples for Practice

Problem-01: Draw a DFA for the language accepting strings starting with 'ab' over input alphabets ∑ = {a, b}

Solution-Regular expression for the given language = ab (a + b)*

Step-01: All strings of the language start with substring "ab".

So, length of substring = 2.

Thus, Minimum number of states required in the DFA = 2 + 2 = 4.

It suggests that minimized DFA will have 4 states
Step-02: We will construct DFA for the following strings-
Ab, aba, abab
Step-03: The required DFA is-

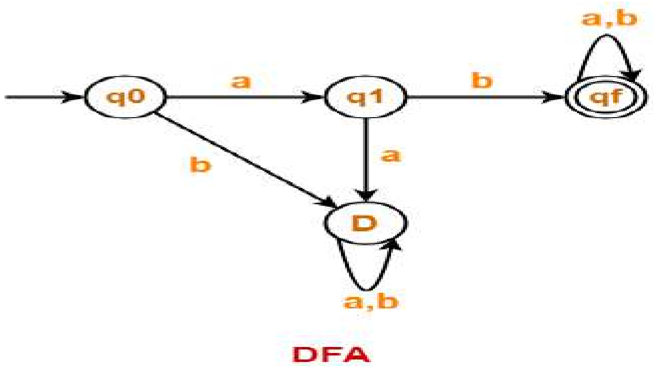


Figure 1.9: Transition Graph

Problem-02: Draw a DFA for the language accepting strings starting with 'a' over input alphabets ∑ = {a, b}
Solution-Regular expression for the given language = a(a + b)*
Step-01: All strings of the language starts with substring "a".
So, length of substring = 1.
Thus, Minimum number of states required in the DFA = 1 + 2 = 3.
It suggests that minimized DFA will have 3 states.
Step-02: We will construct DFA for the following strings-
A,aa
Step-03: The required DFA is-

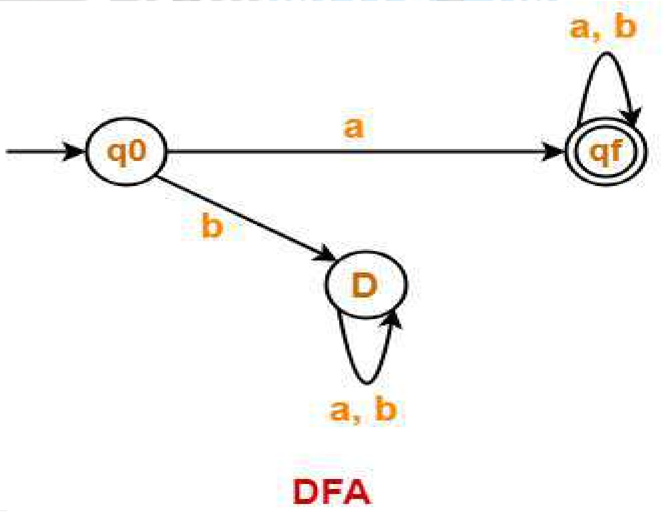


Figure 1.10: Transition Graph

Problem 3: Construct a minimal DFA, which accepts set of all strings over {0, 1}, which when interpreted as binary number is divisible by '3'. Means 110 in binary is equivalent to 6 in decimal and 6 is divisible by 3. **Answer** So if you think in the way of considering remainders if you divide by 3 that is {0, 1, 2*}*

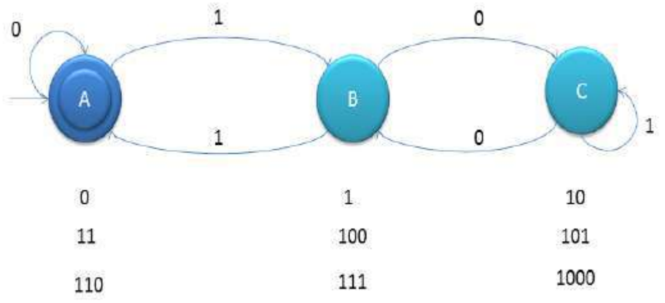


Figure 1.11: Transition Graph

As you can see that binary number which is divisible by 2 is appearing on left state. Short Trick to create such DFAs

Create Transition Table as below

Table creation rules

State	0	1
q0	q0	q1
q1	q2	q0
q2	q1	q2

- ➢ First write the input alphabets, example 0, 1
- ➢ There will n states for given number which is divisor.
- ➢ Start writing states, as for n=3: q0 under 0, q1 under 1
- ➢ Q2 under 0 and q0 under 1
- ➢ Q1under 0 and q2 under 1

If the input alphabets are 0, 1, 2 then table will expand accordingly with same rules above. Example For ternary pattern and n=4, table will be as follows;

State	*0*	*1*	*2*
q0	*q0*	*q1*	*q2*
q1	*q3*	*q0*	*q1*
q2	*q2*	*q3*	*q0*
q3	*q1*	*q2*	*q3*

Mealy and Moore Machine:

These machines are modeled to show transition and output symbol. These machines do not define a language by accepting or rejecting input string, so there is no existence of final state. Finite automata generate outputs related to each act. While two types of finite state machines create output –

- **Mealy Machine**
- **Moore machine**

Mealy Machine:

A Mealy Machine is considered as an FSM, the output will be based on the present state and the present input. **Mealy machine is explained as a 6 tuple (Q, ∑, O, δ, X, q0) where –**

- Q is a finite set of states.
- ∑ is a finite set of symbols called the input alphabet.
- is a finite set of symbols called the output alphabet.
- δ is the input transition function where δ: Q × ∑ → Q
- X is the output transition function where X: Q × ∑ → O
- q0 is the initial state from where any input is processed (q0 ∈ Q).

The state table of a Mealy Machine is mentioned below –

Present state	**Next state**			
	input = 0		**input = 1**	
	State	**Output**	**State**	**Output**
→ a	b	x_1	c	x_1
b	b	x_2	d	x_3
c	d	x_3	c	x_1
d	d	x_3	d	x_2

The state diagram of the above Mealy Machine is –

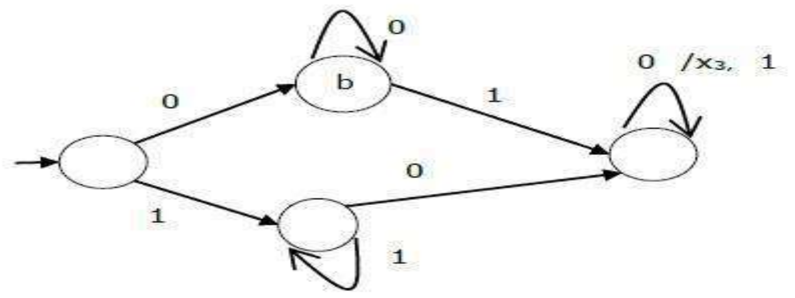


Figure 1.12: Transition Graph

Moore Machine

Moore machine is also considered as an FSM and the outputs depend on the present state.

A Moore machine is also explained by a 6 tuple (Q, ∑, O, δ, X, q0) where –

- Q is a finite set of states.
- ∑ is a finite set of symbols called the input alphabet.
- is a finite set of symbols called the output alphabet.
- δ is the input transition function where δ: Q × ∑ → Q
- X is the output transition function where X: Q → O
- q0 is the initial state from where any input is processed (q0 ∈ Q).

The state table of a Moore Machine is shown below –

Present state	**Next State**	**Output**

	Input = 0	Input = 1	
→ a	b	c	x_2
b	b	d	x_1
c	c	d	x_2
d	d	d	x_3

The state diagram of the above Moore Machine is –

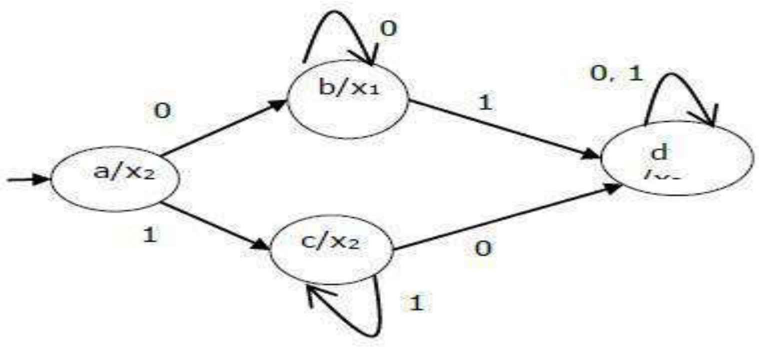


Figure 1.13: Transition Graph

Composite finite-state machines:

A finite state machine, FSM, is a box with C input/output channels, and S states, and a fixed map f:S×C→S×C∪0f:S×C→S×C∪0. If a state (ci,sj) (ci,sj) is mapped to the 0 element it means it enters a loop and can't exit. if we join the channels of several FSM's pair wise, we obtain a new FSM, i.e. we get a system with some number of un-joined channels, C, and internal states given by the product of the number of internal states of each component FSM, S.A composite FSM is allowed to have internal loops, i.e. we may never exit through an un-joined channel.

Conversion from Moore Machine to Mealy Machine

Algorithm
Input – Moore Machine
Output – Mealy Machine
Step 1 – Take a blank Mealy Machine transition table format.
Step 2 – Copy all the Moore Machine transition states into this table format.
Step 3 – Now check the present states and their corresponding outputs in the Moore Machine state table; if for a state Qi output is m, copy it into the output columns of the Mealy Machine state table wherever Qi appears in the next state.

Example

Let's see the following Moore machine –

Present State	Next State		Output
	a = 0	a = 1	
→ a	d	b	1

b	a	d	0
c	c	c	0
d	b	a	1

Now we apply Algorithm to convert it to Mealy Machine.

Step 1 & 2 –

Present State	**Next State**			
	a = 0		**a = 1**	
	State	**Output**	**State**	**Output**
→ a	d		b	
b	a		d	
c	c		c	
d	b		a	

Step 3 –

Present State	**Next State**			
	a = 0		**a = 1**	
	State	**Output**	**State**	**Output**
=> a	d	1	b	0
b	a	1	d	1
c	c	0	c	0
d	b	0	a	1

Conversion from Mealy Machine to Moore Machine

Algorithm
Input – Mealy Machine
Output – Moore Machine
Step 1 – Here measure the number of different outputs for each state (Qi) that are available in the state table of the Mealy machine.
Step 2 – Incase if all the outputs of Qi are same, copy state Qi. If it has n distinct outputs, break Qi into n states as Qin where n = 0, 1, 2.......

Step 3 – If the output of the initial state is 1, insert a new initial state at the beginning which gives 0 output.

Example

Let's see the following Mealy Machine –

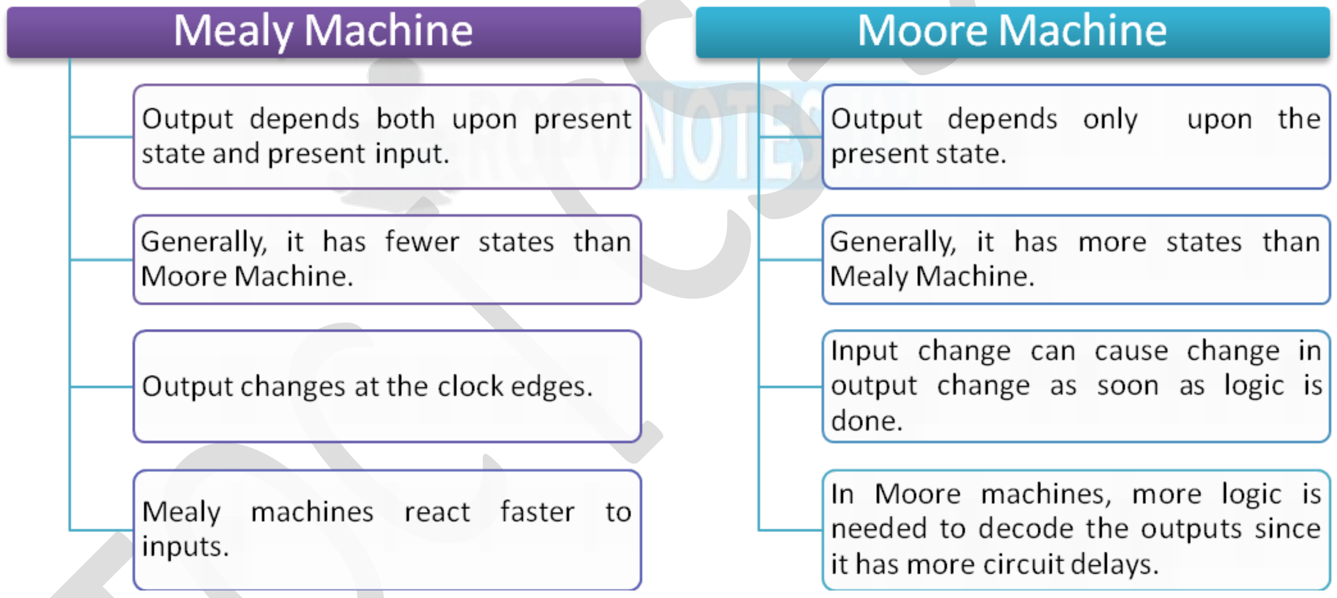
Present State	**Next State**

	a = 0		a = 1	
	Next State	Output	Next State	Output
→ a	d	0	b	1
b	a	1	d	0
c	c	1	c	0
d	b	0	a	1

While the states ‘a’ and ‘d’ provide only 1 and 0 outputs respectively, so we create states ‘a’ and ‘d’. But states ‘b’ and ‘c’ delivers different outputs (1 and 0). So, we divide b into b0, b1 and c into c0, c1.

Present State	Next State		Output
	a = 0	a = 1	
→ a	d	b_1	1
b_0	a	d	0
b_1	a	d	1
c_0	c_1	C_0	0
c_1	c_1	C_0	1
d	b_0	a	0

Difference between Mealy and Moore Machine:



Mealy Machine

Output depends both upon present state and present input.

Generally, it has fewer states than Moore Machine.

Output changes at the clock edges.

Mealy machines react faster to inputs.

Moore Machine

Output depends only upon the present state.

Generally, it has more states than Mealy Machine.

Input change can cause change in output change as soon as logic is done.

In Moore machines, more logic is needed to decode the outputs since it has more circuit delays.

Figure 1.12: Difference between Mealy & Moore Machine