RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Theory of Computation**

Subject Code: **IT-503**

Semester: **5th**

Department of Information Technology
Subject Notes
IT503 (A) - Theory of Computation
B.Tech, IT-5th Semester

Unit I

Syllabus : Introduction of the theory of computation, Finite state automata – description of finite automata, properties of transition functions, Transition graph, designing finite automata, FSM, DFA, NFA, 2-way finite automata, equivalence of NFA and DFA, Mealy and Moore machines.

Unit Objective: Student learns some fundamental concepts in automata theory and designing of Finite Automata, conversion NFA to DFA. Application of Finite Automata in computer science and real world.

Introduction of Theory of Computation:

Automata theory (also known as Theory of Computation) is a theoretical branch of Computer Science and Mathematics, which mainly deals with the logic of computation with respect to simple machines, referred to as automata.

Characteristics of TOC:

- 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)

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

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".

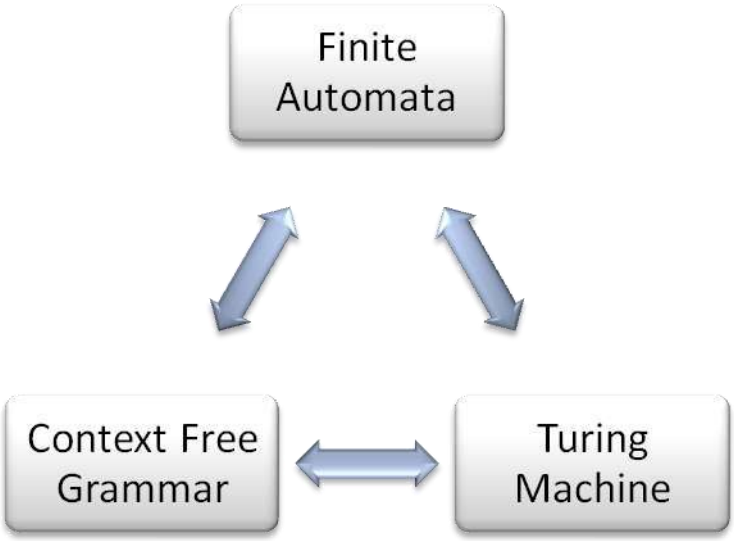


Figure 1.1: Computational Model of Automata Theory

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.

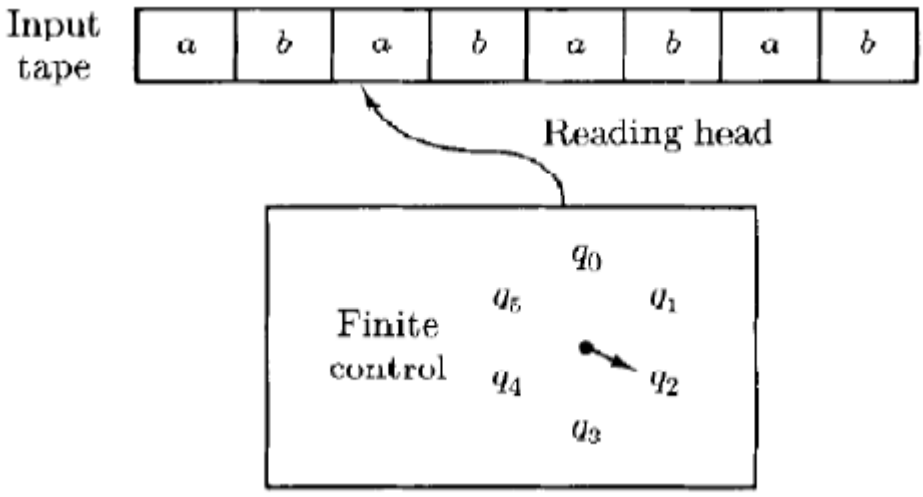


Figure 1.2: Finite Automata

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:

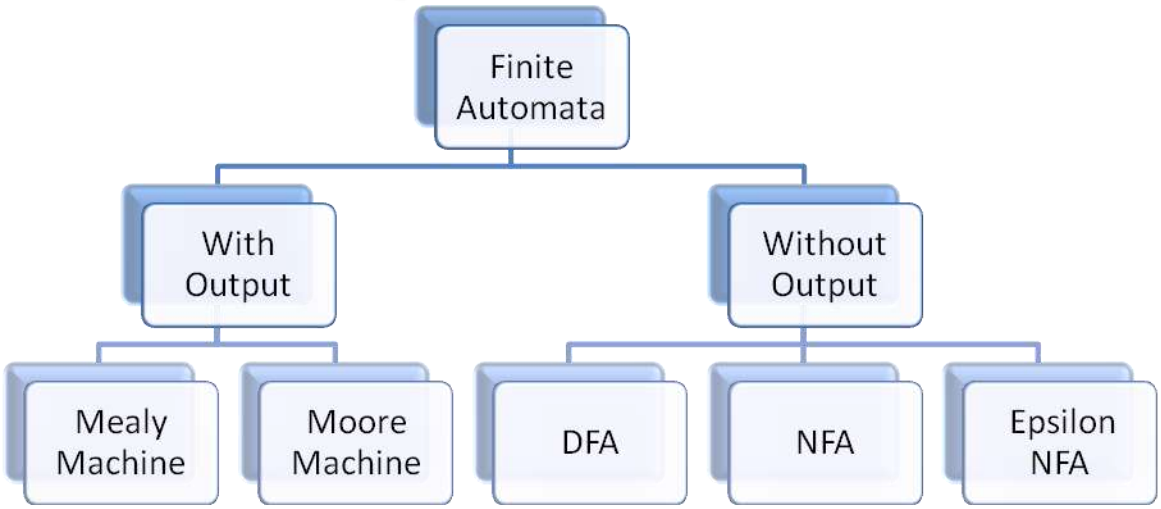


Figure 1.3: Classification of Finite Automata

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 count the number of 1's to check whether it is odd or even
- If the number of 1's are 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

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

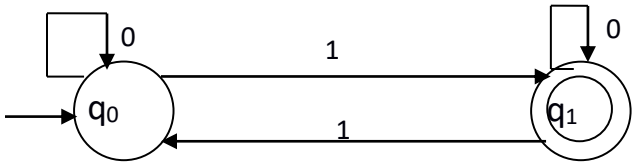


Figure 1.4: Transition Graph

Transition Graph: it is a finite directed labeled graph in which each vertexes (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.

Deterministic Finite Automata (DFA):

Deterministic automaton is one in which each move (transition from one state to another) is uniquely determined by the current configuration. If the internal states, input and contents of the storage are known it is possible to predict the next (future) behavior of the automaton.

Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, Σ, δ, q0, F) where:

1. Q is a finite set of states.
2. Σ is a finite set of symbols called the alphabet.
3. δ is the transition function where δ: Q × Σ → Q
4. q0 is the initial state from where any input is processed (q0 ∈ Q).
5. F is a set of final state/states of Q (F ⊆ Q).

DFA can be represented by Transition Graph and Transition Diagram as shown in Table 1.1 and Figure 1.4

Example:

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

Solution:

Step-01: All strings of the language starts 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-

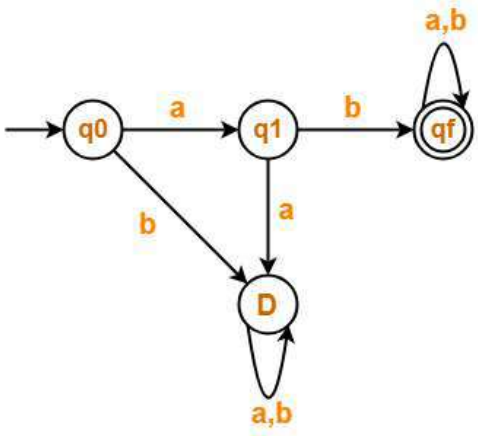


Figure 1.5: Example of DFA

Non-Deterministic Finite Automata (NDFA):

In NDFA, for a particular input symbol, the machine can move to any combination of the states. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called Non-deterministic Automaton.

Formal Definition of NDFA

An NDFA can be represented by a 5-tuple (Q, Σ, δ, q0, F) where:

1. Q is a finite set of states.
2. Σ is a finite set of symbols called the alphabets.
3. δ is the transition function where δ: Q × Σ → 2Q
   (Here the power set of Q (2Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
4. q0 is the initial state from where any input is processed (q0 ∈ Q).
5. F is a set of final state/states of Q (F ⊆ Q).

Example:

Design an NFA with ∑ = {0, 1} accepts all string ending with 01.

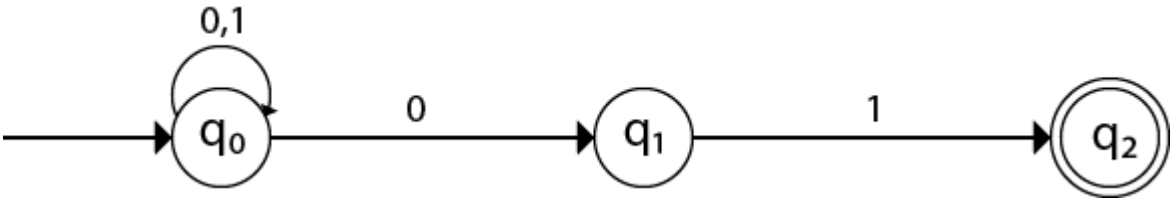


Figure 1.6: NFA

Difference between DFA & NDFA:

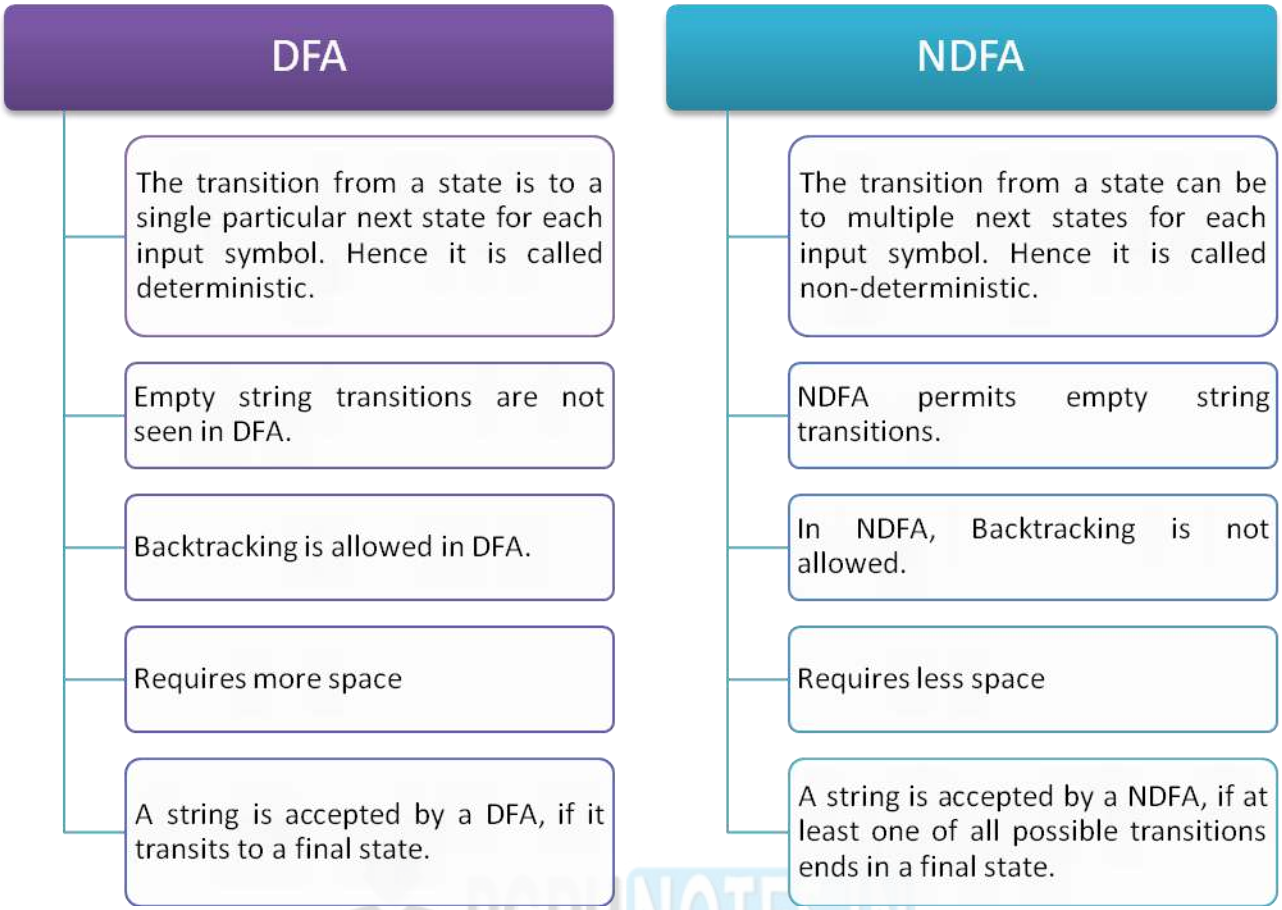


Figure 1.7: Difference between DFA & NDFA

Equivalence of DFA and NDFA:

Although the DFA and NFA have distinct definitions, a NFA can be translated to equivalent DFA using the subset construction algorithm. i.e., the constructed DFA and the NFA recognize the same formal language. Both types of automata recognize only regular languages.

Therefore every language that can be described by some NDFA can also be described by some DFA.

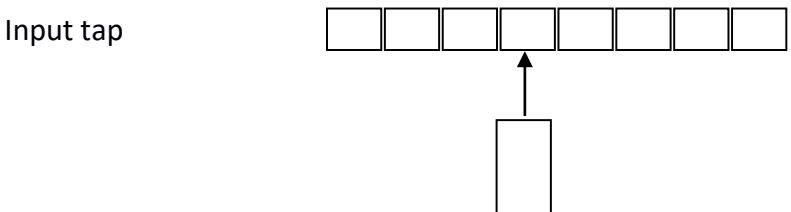
Theorem

Let M = (Q, Σ, δ, q_0, F) be a non-deterministic finite automaton. There exists a deterministic finite automaton M', such that L(M') =L(M).

i.e. for every NDFA there exists a DFA which simulates the behavior of NDFA . Hence if language L is accepted by NDFA, then there exist a DFA M' which also accept L. Where M' = (Q', Σ, δ', q'_0, F')

Two way finite automata

The finite automata discussed so far has a δ transition which indicate where to next from current state on receiving particular input. But 2-way FA is a model in which linear direction is mentioned on receiving particular and being in some current state. There are two direction that are allowed in 2-FA and these are left and right direction as shown in fig. 1.4



Tape head which can move L-R Finite Control

Figure 1.8: Two way finite automata

Formal Definition of Two way finite automata

It is a collection of 5-tuple M = (Q, Σ, δ, q_0, F), where

Q: a finite set of states

Σ: a finite set called the input alphabet

δ: a transition function which maps Q x {L,R}

q_0: a start state (also called initial state) which is an element of Q ($q_0 \in Q$)

F: Set of final states.

NDFA to DFA:

Let X = (Qx, Σ, δx, q0, Fx) be an NDFA which accepts the language L(X). We have to design an equivalent DFA Y = (Qy, Σ, δy, q0, Fy) such that L(Y) = L(X). The following procedure converts the NDFA to its equivalent DFA:

Procedure:

Input: An NDFA

Output: An equivalent DFA

Step 1: Create state table from the given NDFA.

Step 2: Create a blank state table under possible input alphabets for the equivalent DFA.

Step 3: Mark the start state of the DFA by q0 (Same as the NDFA).

Step 4: Find out the combination of States {Q0, Q1,..., Qn} for each possible input alphabet.

Step 5: Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.

Step 6: The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example: Let us consider the NDFA shown in the figure 1.9 below.

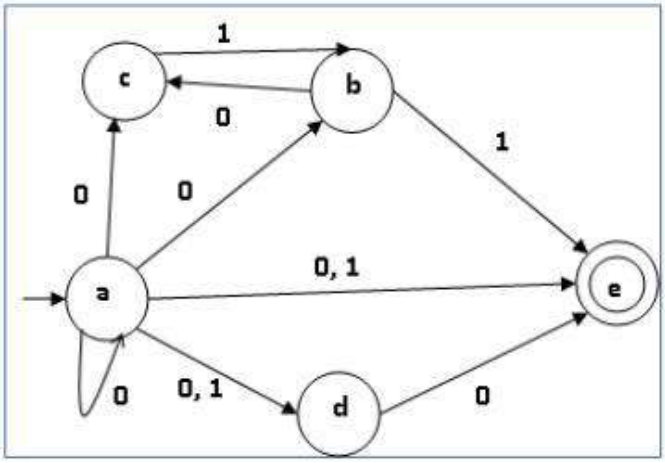


Figure 1.9: NFA

Q	δ(q,0)	δ(q,1)
A	{a,b,c,d,e}	{d,e}
B	{c}	{e}
C	Ø	{b}
D	{e}	Ø
E	Ø	Ø

Using the above algorithm, we find its equivalent DFA. The state table of the DFA is shown in below.

q	δ(q,0)	δ(q,1)
[a]	[a,b,c,d,e]	[d,e]
[a,b,c,d,e]	[a,b,c,d,e]	[b,d,e]
[d,e]	[e]	Ø
[b,d,e]	[c,e]	[e]
[e]	Ø	Ø
[c, e]	Ø	[b]
[b]	[c]	[e]
[c]	Ø	[b]

State diagram for DFA is as shown below in figure 1.10:

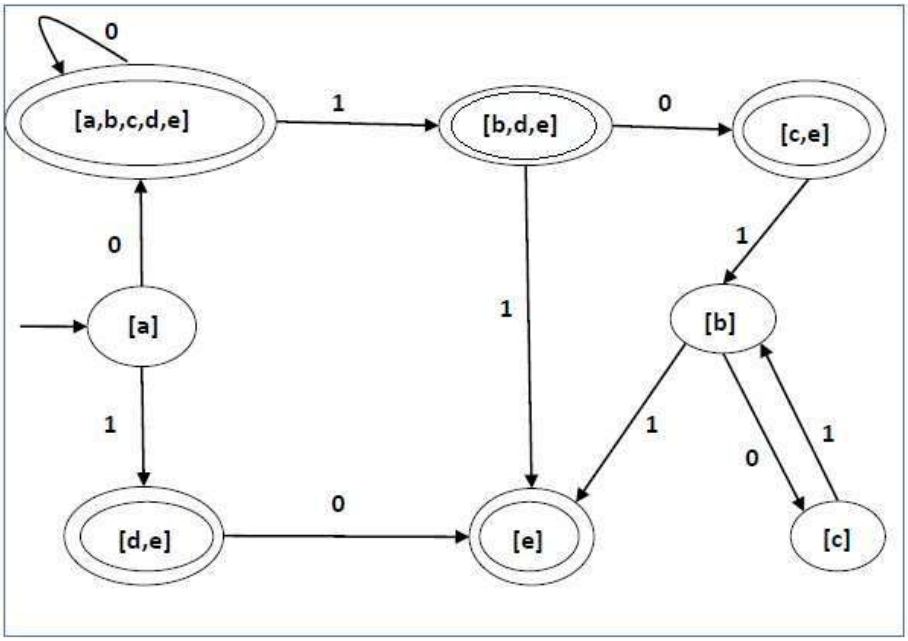


Figure 1.10: DFA

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.

Mealy Machine : A Mealy Machine is an FSM whose output depends on the present state as well as the present input.

Formal definition of Mealy Machine

It can be described by a 6 tuple (Q, Σ, O, δ, X, q0) where −

1. Q is a finite set of states.
2. Σ is a finite set of symbols called the input alphabet.
3. is a finite set of symbols called the output alphabet.
4. δ is the input transition function where δ: Q × Σ → Q
5. X is the output transition function where X: Q × Σ → O
6. 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 −

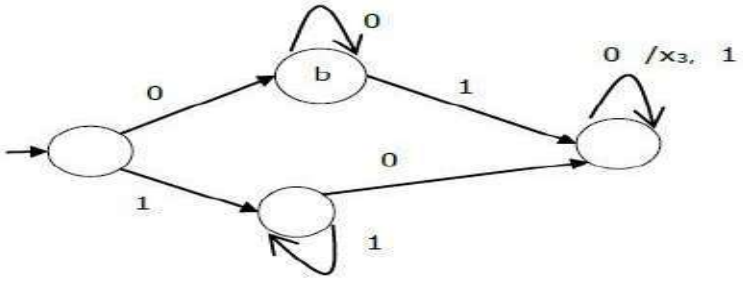


Figure 1.11:Mealy Machine Transition Graph

Moore Machine: Moore machine is an FSM whose outputs depend on only the present state.

Formal Definition of Moore Machine

A Moore machine can be described by a 6 tuple (Q, Σ, O, δ, X, q0) where:

1. Q is a finite set of states.
2. Σ is a finite set of symbols called the input alphabet.
3. is a finite set of symbols called the output alphabet.
4. δ is the input transition function where δ: Q × Σ → Q
5. X is the output transition function where X: Q → O
6. 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 −

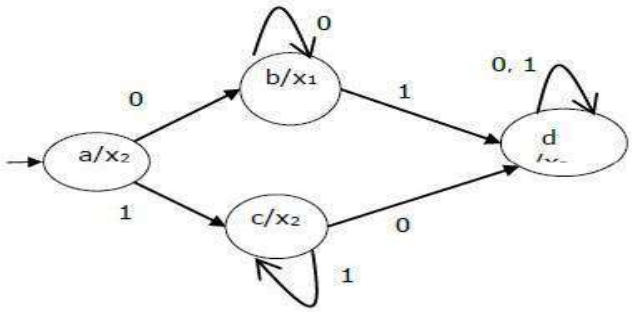


Figure 1.12: Moore Machine Transition Graph

Difference between Mealy and Moore Machine:

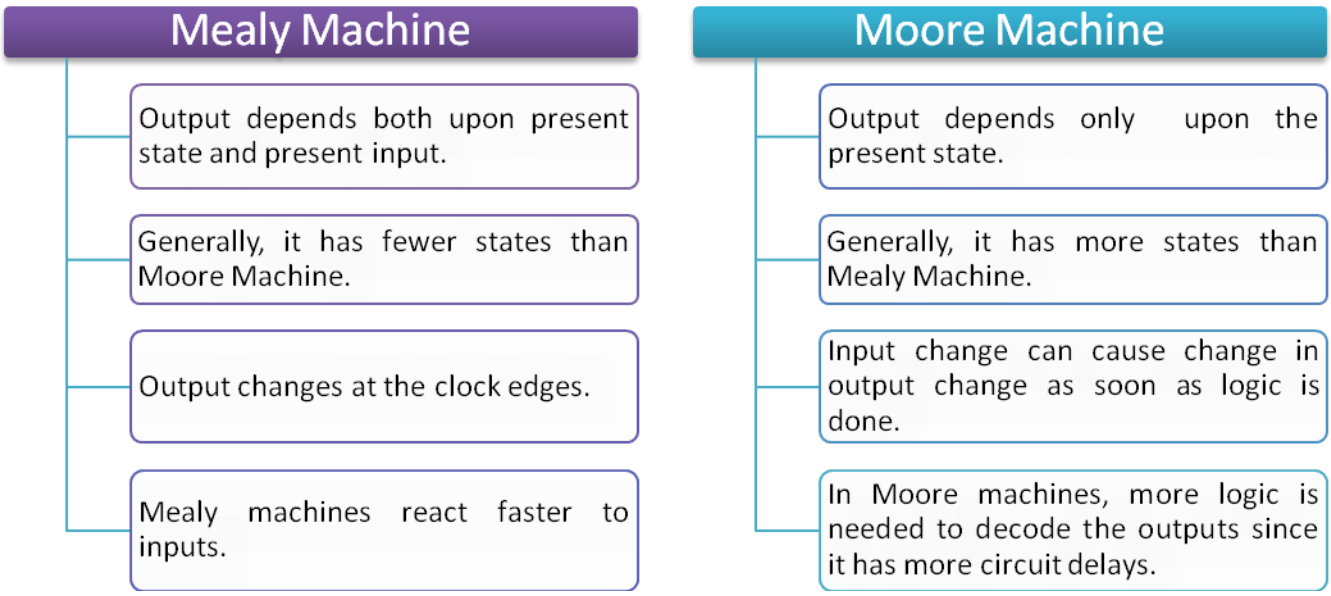


Figure 1.13: Difference between Mealy & Moore Machine

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

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in