THEORY OF AUTOMATA AND COMPUTING
UNIT - I

In theoretical computer science and mathematics, the theory of computation is the branch that deals with how problems can be efficiently solved on a model of computation, using an algorithm. This field is divided into three major branches:

- Automata theory.
- Computability theory.
- Computational complexity theory**.**

Automata theory:

- Automata theory is the study of abstract computational devices.
- Abstract devices are (simplified) models of real computations
- Computations happen everywhere: On your laptop, on your cell phone etc.

Computability theory:
Computability theory deals primarily with the question of the extent to which a problem is solvable on a computer. Turing machine is one of the example of computability theory. Much of computability theory builds on the halting problem result.

Computational complexity theory:
Complexity theory considers not only whether a problem can be solved at all on a computer, but also how efficiently the problem can be solved. Two major aspects are considered:

- Time complexity: and how many steps does it take to perform a computation.
- Space complexity: and how much memory is required to perform that computation

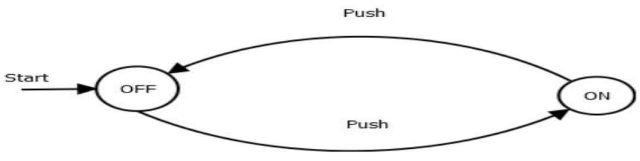
What is Automata?
The term "Automata" is derived from the Greek word "αὐτόματα" which means "self-acting". An automaton (Automata in plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

Applications of Automata Theory:

1. In Compilers
   Lexical Analsis
   Parser Generators.

2. Modeling the circuits: Example On-Off Switch.

Example : A Finite Automata modeling an ON/OFF switch



Some important applications of Automata Theory in General:

1. Word Search and translation of Natural Languages.
2. Parity Checkers, Vending Machines, Communication Protocols
3. Video Games

4. DNA.
5. Securiy.
6. Artificial intelligence.

Terms & Terminologies.

Symbol: It is a basic building block of TOC, It is an abstract entity such as letters and digits.

Example: a, b, 0, 1..

Alphabet:

Definition − An **alphabet** is any finite non empty set of symbols. It is represented by ∑

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

String

Definition − A **string** is a finite sequence of symbols taken from ∑.

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

Length of a String

Definition − It is the number of symbols present in a string. (Denoted by **|w|**).

Examples −

If w='cabcad', |S|= 6

Empty string: It is the string consisting of zero symbols that is

|w|= 0, it is called an **empty string** (Denoted by **λ** or)

Suffix and prefix of the string.

A prefix of the string is any no. of leading symbols of that string.

Example: abc string has prefixes, a, ab,abc.

A suffix of a string is any no. of trailing symbols of that string.

Example: abc string has suffixes c, bc, abc

OPERATIONS ON STRINGS

The basic operation for strings is the binary concatenation operation. We define this operation as follows:

Let x and y be two strings in $\sum^*$.

Property 1 : Concatenation on a set $\sum^*$ is associative since for each x, y, z in $\sum^*$
$x\,(yz)=(xy)z$

Property 2 : Identity element*:* The set $\sum^*$ has an identity element w.r.t the binary operation of concatenation as
$x. = \text{x} = x$ for every x in $\sum^*$

Property 3 : $\sum^*$ has left and right cancellations. For x, y, z in $\sum^*$,

$Zx = zy$ *implies* $x = y$ (left cancellation)
$Xz = yz$ implies $x = y$ (right cancellation)

Property 4: For x, y in $\sum^*$, we have
$|xy| = |\text{x}| + |\text{y}|$

Where $|\text{x}|$, $|y|$, $|xy|$ denote the lengths of the strings $x, y, xy,$ respectively.

We introduce some more operations on strings.

Transpose Operation

The concatenation operation to define the transpose operation as follows:

For any x in $\sum^*$ and a in $\sum$,
$(xa)^T = a(x)^T$

For example, $(aaabab)^T$ is *babaaa.*

Palindrome: A palindrome is a string which is the same whether written forward or backward, e.g. Malayalam.

A palindrome of even length can be obtained by concatenation of a string and its transpose.

Language

If Σ is an alphabet, and L ⊆ Σ*, then L is a (formal) language over Σ.

It is the set of strings of symbols from some alphabet. It can be finite or infinite.

Example: If the language take all possible strings of length 2 over $\sum$={a,b} then L={ab,bb,ba,bb}.

Operations Performed on Languages

The **union** of two languages L and M, denoted L ∪ M, is the set of strings that are in either L, or M, or both. Example If L = {001, 10, 111 } and M = { **ε**, 001 } then L ∪ M = { **ε**, 001, 10, 111 }

The **concatenation** of languages L and M, denoted L.M or just LM , is the set of strings that can be formed by taking any string in L and concatenating it with any string in M.

Example If L = {001, 10, 111 } and M = { **ε**, 001 } then L.M = {001, 10, 111, 001001, 10001, 111001 }

The **closure** of a language L is denoted L and represents the set of those strings that can be formed by taking any number of strings from L, possibly with repetitions (i.e., the same string may be selected more than once) and concatenating all of them.

Examples:

If L = {0, 1} then L^* is all strings of 0 and 1

If L = {0, 11) then L^* consists of strings of 0 and 1 such that the 1 come in pairs, e.g., 011, 11110 and **ε**. But not 01011 or 101.

Power of an Alphabet:

If Σ is an alphabet, we can express the set of all strings of a certain length from that alphabet by using the exponential notation:

Σ^k: the set of strings of length k, each of whose is in Σ

Examples:

Σ^0 : { **ε** }, regardless of what alphabet Σ is. That is **ε** is the only string of length 0

If Σ = { 0, 1 }, then:

1. Σ^1 = { 0, 1 }

2. Σ^2 = {00, 01, 10, 11 }

3. Σ^3 = {000, 001, 010, 011, 100, 101, 110, 111}

Note: confusion between Σ and Σ^1 :

1. Σ is an alphabet; its members 0 and 1 are symbols

2. Σ^1 is a set of strings; its members are strings (each one of length 1)

Kleene Star

Definition − The set $\sum^*$ is the infinite set of all possible strings of all possible lengths over $\sum$ including

The symbol * is called Kleene star and is named after the mathematician and logician Stephen Cole Kleene.

It is represented as

$$\sum^* = \sum_0 \cup \sum_1 \cup \sum_2 \cup \ldots\ldots$$

Example − If $\sum$ = {a, b}, $\sum^*$= { **ε**, a, b, aa, ab, ba, bb,……..}

Kleene Closure/Plus

Definition −The set $\sum^+$ is the infinite set of all possible strings of all possible lengths over $\sum$ excluding **ε**.

It is represented as $\sum^+ = \sum_0 \cup \sum_1 \cup \sum_2 \cup \ldots\ldots$

$$\sum^+ = \sum^* - \{ \varepsilon \}$$

Example − If $\sum$ = { a, b } , $\sum^+$ ={ a, b, aa, ab, ba, bb,……..}

GRAPHS AND TREES

A Graph (or undirected graph) consists of
(i) a nonempty set *V* called the set of vertices.
(ii) a set *E* called the set of edges, and
(iii) a map Φ which assigns to every edge a unique unordered pair of vertices.

Representation of a Graph
A un-directed graph represented by a diagram where the vertices are represented by points or small circles, and the edges by arcs joining the vertices of the associated pair (given by the map Φ

Figure 2.3, for example, gives an undirected graph. Thus, the unordered pair {V_1 V_2} is associated with the edge e_1: the pair *(V2,V2)* is associated with e_6 (e_6 is a self-loop. In general an edge is called a self-loop if the vertices in its associated pair coincide.)

Fig.

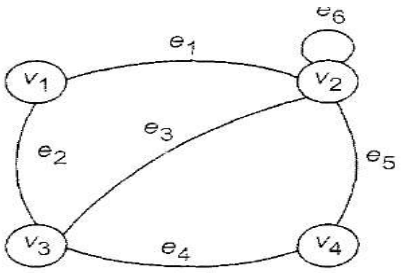


Fig. 2.3 An undirected graph.

Directed graph: A directed graph (or digraph) consists of
(i) a nonempty set '*V' is* called the set of vertices,
(ii) a set *E* called the set of edges, and
(iii) (iii) a map Φ which assigns to every edge a unique ordered pair of vertices.

Representation of a Digraph:

The representation is as in the case of undirected graphs except that the edges are represented by directed arcs.

Figure 2.4, for example gives a directed graph. The ordered pairs $(V_2, V_3),(V_3,V_4),(V_1,V_3)$*is* associated with the edges e_3, e_4, e_2, respectively.

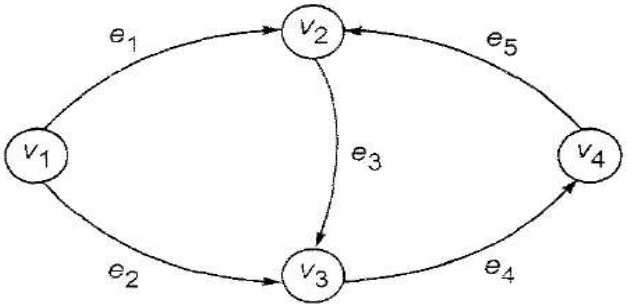


Fig. 2.4 A directed graph.

Definitions

If *(Vi, Vj)*is associated with an edge *e,* then V_i and V_j are called the end vertices of *e;* V_i is called a predecessor of V_j which is a successor of V_i

In Fig. 2.3. V_2 and V_3 are the end vertices of e_3. *I*n Fig. 2.4, v2 is a predecessor of *v3* which is a successor of V_2. Also, v_4 is a predecessor of v2 and successor of *v3*

Degree of a Vertex:

The degree of a vertex in a graph (directed or undirected) is the number of edges with *V* as an end vertex. (A self-loop is counted twice while calculating the degree.)

In Fig. 2.3, deg(v_1) =2, deg(v_3) =3, deg(V_2) =5. In Fig. 2.4, deg(v_2) = 3, deg(v_4) = 2.

TREES

A graph (directed or undirected) is called a tree if it is connected and has no circuits.

The graphs given in Figs. 2.6 and 2.7, for example, are trees. The graphs given in Figs. 2.3 and 2.4 are not trees

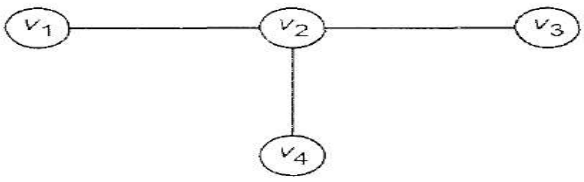


Fig. 2.6 A tree with four vertices.

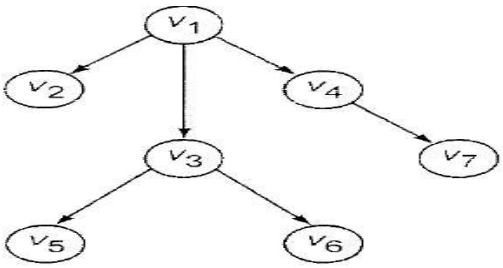


Fig. 2.7 A tree with seven vertices.

Properties of a Tree:

Property 1: A Tree is a connected graph with no circuits or loops

Property 2: In a tree there is one and only one path between every pair of vertices.

Property 3 If in a graph there is a unique (i.e. one and only one) path between every pair of vertices, then the graph is a tree.

Property 4: A tree with *n* vertices has *n* - 1 edges.

Property5: If a connected graph with *n* vertices has *n* - 1 edge, then it is a tree

Property6: If a graph with no circuits has *n* vertices and *n* - 1 edges, then it is a tree

Leaf of a Tree

A leaf in a tree can be defined as a vertex of degree one. Vertices other than leaves are called internal vertices.

In Fig. 2.6, for example, v_1, v_3, v_4 are leaves and v_2 is an internal vertex. In Fig. 2.7 v_2, v_5, v_6, v_7 *are* leaves and v_1, v_3 , v_4 are internal vertices.

Ordered Directed tree:

An ordered directed tree is a digraph satisfying the following conditions:

1. There is one vertex called the root of the tree which is distinguished from all the other vertices and the root has no predecessors.
2. There is a directed path from the root to every other vertex.
3. Every vertex except the root has exactly one predecessor.
4. The successors of each vertex are ordered "from the left".

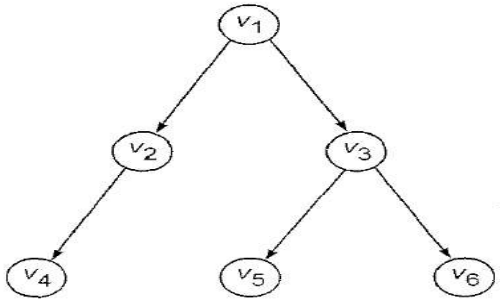


Fig. 2.8 An ordered directed tree.

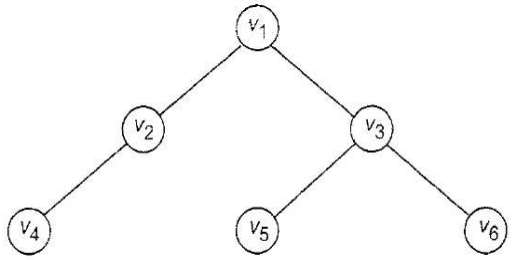


Fig. 2.9 Representation of an ordered directed tree.

Binary Tree:

A binary tree is a tree in which the degree of the root is 2 and the remaining vertices are of degree 1 or 3

Note: In a binary tree any vertex has at most two successors. For example, the trees given by Figs. 2.11 are binary trees. The tree given by Fig. 2.9 is not a binary tree.

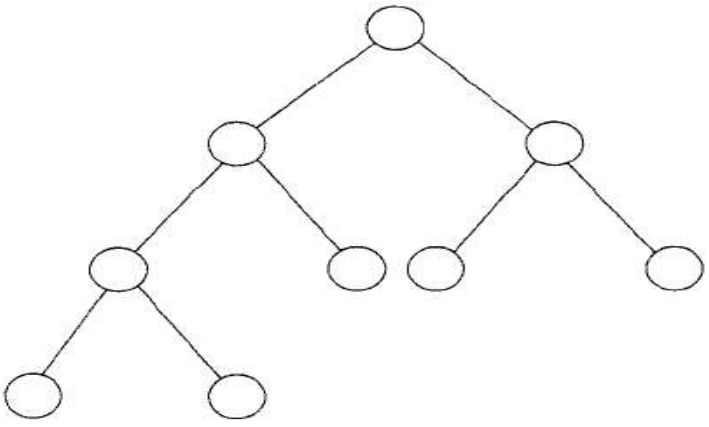


Fig. 2.11 Binary tree of minimum height with 9 vertices.

The following are the terminology regarding trees:

i. A son of a vertex *v* is a successor of V
ii. The father of *v* is the predecessor of V.
iii. If there is a directed path from V1 to V_2, V_1 is called an ancestor of V_2, and V_2 is called a descendant of V_1 .
iv. The number of edges in a path is called the length of the path.
v. The height of a tree is the length of a longest path from the root. For example, for the tree given by Fig. 2.9, the height is 2. (Actually there are three longest paths, V1→ V2→ V4, V1→ V3 →V5, V1→ V3 →V6, Each is of length 2.)
vi. A vertex *V* in a tree is at level *k* if there is a path of length *k* from the root to the vertex *V* (the maximum possible level in a tree is the height of the tree).

Figure 2.10, for example gives a tree where the levels of vertices are indicated.

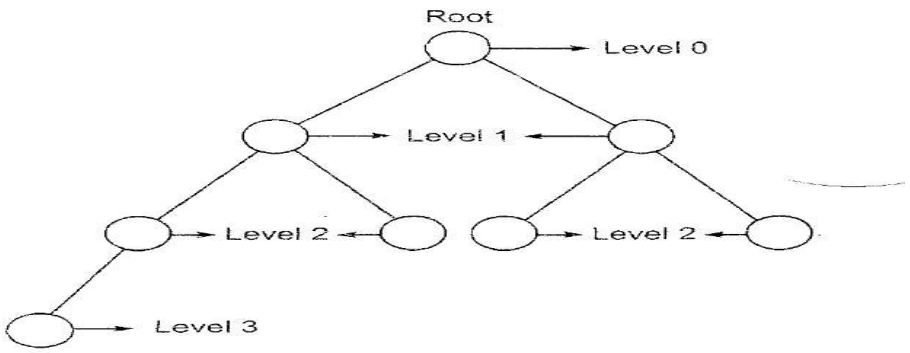
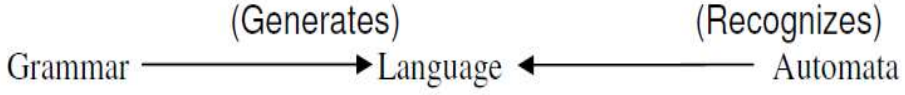


Fig. 2.10 Illustration of levels of vertices.

- For a binary tree '*T*' with *n* vertices the minimum possible height is $[\log_2=(n + 1) – 1]$, and the maximum possible height is *(n - 1)/2*.
- The no. of vertices in a binary tree is odd.
- The No. of leaves in binary tree 'T' is n+1/2, where 'n' is the no. of vertices.

Finite State systems



In theoretical computer science, **automata theory** is the study of abstract machines (or more appropriately, abstract 'mathematical' machines or systems) and the computational problems that can be solved using these machines. These abstract machines are called automata.

Automata: A algorithm or program that automatically recognizes if a particular string belongs to the language or not, by checking the grammar of the string.

An automata is an abstract computing device (or machine). There are different varities of such abstract machines (also called models of computation) which can be defined mathematically.

Automaton consists of

- **states** (represented in the figure by circles),
- And **transitions** (represented by arrows).

As the automaton sees a symbol of input, it makes a *transition* (or *jump*) to another state,

Every Automaton fulfills the three basic requirements

- Every automaton consists of some essential features as in real computers. It has a mechanism for reading input. The input is assumed to be a sequence of symbols over a given alphabet and is placed on an input tape(or written on an input file). The simpler automata can only read the input one symbol at a time from left to right but not change. Powerful versions can both read (from left to right or right to left) and change the input

- The state machines can be subdivided into Transducers, Acceptors, **Acceptors** (also **recognizers** and **sequence detectors**) produce a binary output, saying either *yes* or *no* to answer whether the input is accepted by the machine or not. Transducers generate output based on a given input . (or automaton with output).
  .
- The automaton may have a temporary storage, consisting of an unlimited number of cells, each capable of holding a symbol from an alphabet (which may be different from the input alphabet). The automaton can both read and change the contents of the storage cells in the temporary storage.

- The most important feature of the automaton is its control unit, which can be in any one of a finite number of interval states at any point. It can change state in some defined manner determined by a transition function

Finite Automata

An automaton with a finite number of states is called a **Finite Automaton**.
A finite-state machine (FSM) or finite-state automaton (plural: automata), or simply a state machine, is a mathematical model of computation used to design both computer programs and sequential logic circuits. It is conceived as an abstract machine that can be in one of a finite number of states. Finite Automata are used in text processing, and hardware design.

The machine is in only one state at a time; the state it is in at any given time is called the current state. It can change from one state to another when initiated by a triggering event or condition, this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

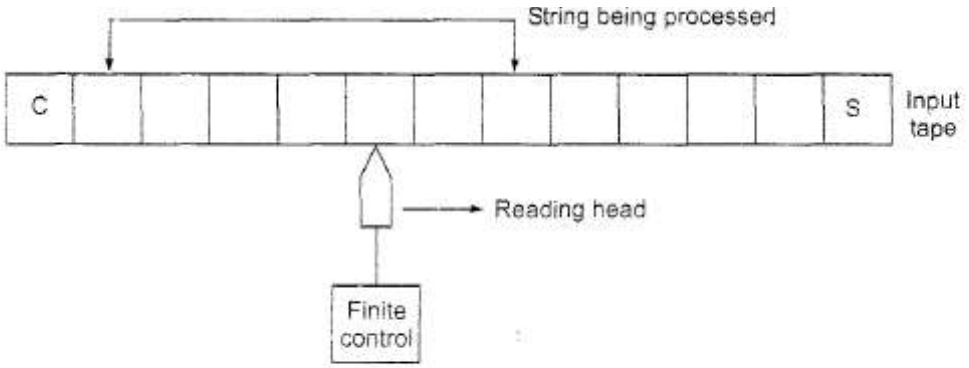


Fig. 3.4 Block diagram of a finite automaton.

Figure 3.4 is the block diagram for a finite automaton. The various components are explained as follows:

(i) **Input tape:** The input tape is divided into squares, each square containing a single symbol from the input alphabet $\sum$. The end squares of the tape contain the end-marker ¢ at the left end and the endmarker $ at the right end. Absence of end-markers indicates that the tape is of infinite length. The left-to-right sequence of symbols between the two end-markers is the input string to be processed.

(ii) **Reading head*:*** The head examines only one square at a time and can move one square either to the left or to the right. For further analysis, we restrict the movement of the R-head only to the right side.

(iii) ***Finite control.*** The input to the finite control is symbol under the R-head, say *a,* and the present state of the machine, say *q,* to give the following outputs:
(a) A motion of R-head along the tape to the next square
(b) the next state of the finite state machine given by *(q, a).*

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 is divide 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.

Once the entire string has been processed, the state in which the automation enters is examined. If it is an accept state , the input string is accepted ; otherwise, the string is rejected . Summarizing all the above we can formulate the following formal definition:

TRANSITION SYSTEMS

A transition graph or a transition system is a finite directed labeled graph in which each vertex (or node) represents a state and the directed edges indicate the transition of a state and the edges are labeled with input or output.

A typical transition system is shown in Fig. 3.5. In the figure, the initial state is represented by a circle with an arrow pointing towards it, the final state by two concentric circles, and the other states are represented by just a circle.

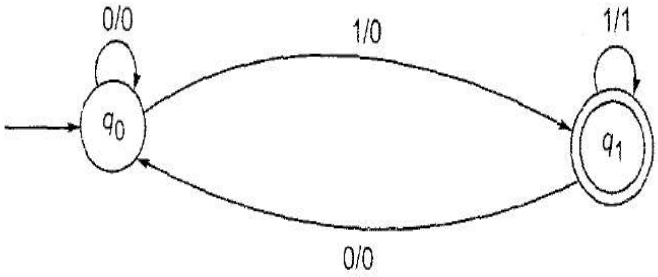


Fig. 3.5 A transition system.

The transition function of finite Automata is represented in two ways

1. Diagram or Graphical Representation
2. Transition table

Diagram or Graphical Representation: In this the states are represented as nodes, the initial state is preceded with an arrow, final state is represented with concentric circles and the remaining states are represented with normal circles.

Transition table:
It is basically a tabular representation of the transition function that takes two arguments (a state and a symbol) and returns a value (the "next state").

- Rows correspond to states,
- Columns correspond to input symbols,
- Entries correspond to next states
- The start state is marked with an arrow
- Final state is marked with two concentric circles

Write from notes

Finite Automaton can be classified into two types −

- Deterministic Finite Automaton (DFA)
- Non-deterministic Finite Automaton (NDFA / NFA)

Deterministic Finite Automaton (DFA)

In DFA, for each input symbol, one can determine the state to which the machine will move. Hence, it is called **Deterministic Automaton.** As it has a finite number of states, the machine is called **Deterministic Finite Machine** or **Deterministic Finite Automaton.**

Formal Definition of a DFA

A DFA can be represented by a 5-tuple (Q, ∑, δ, q_0, F) where −

- **Q** is a finite set of states.
- ∑ is a finite set of symbols called the alphabet.
- **δ** is the transition function where δ: Q × ∑ → Q
- $\mathbf{q_0}$ is the initial state from where any input is processed (q0 ∈ Q).
- **F** is a set of final state/states of Q (F ⊆ Q).

Acceptability of a string by Finite Automata:

Definition: A string *x* is accepted by a finite automaton *M* = (Q, ∑, δ, q_0, F) if δ $(q_o, x) = q$ for some $q \in F$, is the acceptance condition of a string .

Note*:* A final state is also called an accepting state

PROPERTIES OF TRANSITION FUNCTIONS:

Property 1: δ *(q,* ε) =*q* is a finite automaton. This means that the state of the system can be changed only by an input symbol.

Property 2 For all strings *w* and input symbols *a,*

δ *(q, aw)* = δ *(*δ *(q,* ***a),*** *w)*

δ *(q, wa)* = δ *(*δ *(q, w), a)*

This property gives the state after the automaton consumes or reads the first symbol of a string *aw* and the state after the automaton consumes a prefix of the string *wa.*

Non –deterministic FA (NDFA) or (NFA)

In NDFA, for a particular input symbol, the machine can move to any combination of the states in the machine. In other words, the exact state to which the machine moves cannot be determined. Hence, it is called **Non-deterministic Automaton.** As it has finite number of states, the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**.

Formal Definition of an NDFA:

An NDFA can be represented by a 5-tuple (Q, ∑, δ, q_0, F) where −

- **Q** is a finite set of states.
- ∑ is a finite set of symbols called the alphabets.

- **δ** is the transition function where δ: Q × ∑ → **2^Q** (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of an NDFA − (same as DFA)

An NDFA is represented by digraphs called state diagram.

- The vertices represent the states.
- The arcs labeled with an input alphabet show the transitions.
- The initial state is denoted by an empty single incoming arc.
- The final state is indicated by double circles.

Example

Let a non-deterministic finite automaton be →

- Q = {a, b, c}
- ∑ = {0, 1}
- q_0 = {a}
- F={c}
- The transition function δ as shown below −

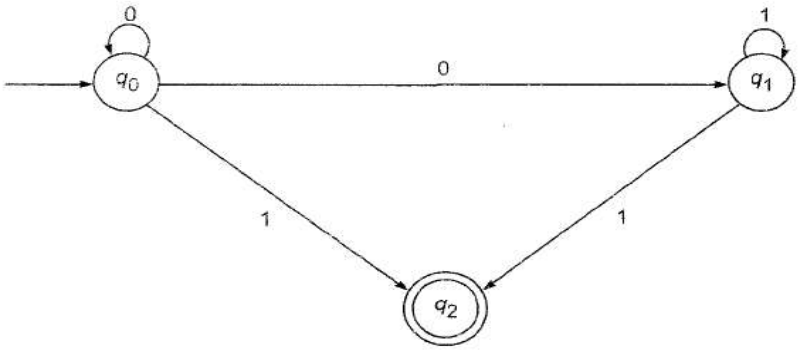


Fig. 3.7 Transition system representing nondeterministic automaton.

Equivalence of DFA's and NFA's

Since every DFA is an NFA, the class of languages accepted by NFA's includes the regular sets (the languages accepted by DFA's).

A DFA's can simulate NFA's; that is, for every NFA we can construct an equivalent DFA (one which accepts the same language).

The way a DFA simulates an NFA is to allow the states of the DFA to correspond to sets of states of the NFA.

Statement: Let L be a set accepted by a nondeterministic finite automata then there exists a deterministic finite automaton that accepts L

Proof: Let M = (Q, ∑, , q_0, F) be an NFA accepting L.

Define a DFA, $M^1 = (Q^1, \sum, {}^1 q_0{}^1, \mathbf{F}^1)$, as follows.

The states of M^1 are all the subsets of the set of states of M

That is, $Q^1 = 2^Q$.

M^1 will keep track in its state of all the states M could be in at any given time.

$\mathbf{F}^1$ is the set of all states in Q^1 containing a final state of M.

An element of Q^1 will be denoted by $[q_1, q_2, \ldots q_i]$, where $q_1. q_2. \ldots, q_i$ are in Q.

$q_0{}^1 = [q_0]$

We define

$$\delta'([q_1, q_2, \ldots, q_i], a) = [p_1, p_2, \ldots, p_j]$$

if and only if

$$\delta(\{q_1, q_2, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}.$$

That is, ^{1} applied to an element $[q_1, q_2, \ldots, q_i]$ of Q^1 is computed by applying to each state of Q represented by $[q_1, q_2, \ldots, q_i]$.

On applying to each of $q_1, q_2, \ldots, q_i$ and taking the union, we get some new set of states,

$p_1, p_2, \ldots p_j$

This new set of states has represented as $[p_1, p_2, \ldots p_j]$ in Q^1 and that element is the value

$^1([q_1, q_2, \ldots, q_i], a)$.

It is easy to show by induction on the length of the input string *x* that

$$\delta'(q_0', x) = [q_1, q_2, \ldots, q_i]$$

if and only if

$$\delta(q_0, x) = \{q_1, q_2, \ldots, q_i\}.$$

Basis : The result is trivial l for = 0,

since $q_0^1 = [q_0]$ and must be 1.

Induction: Suppose that the hypothesis is true for inputs of length m or less.

Let '*x*a' be a string of length m + 1 with 'a' in ∑.

Then

$$\delta'(q_0', xa) = \delta'(\delta'(q_0', x), a).$$

By the inductive hypothesis,

$$\delta'(q_0', x) = [p_1, p_2, \ldots, p_j]$$

If and only if

$$\delta(q_0, x) = \{p_1, p_2, \ldots, p_j\}.$$

But from definition of

$$\delta'([p_1, p_2, \ldots, p_j], a) = [r_1, r_2, \ldots, r_k]$$

if and only if

$$\delta(\{p_1, p_2, \ldots, p_j\}, a) = \{r_1, r_2, \ldots, r_k\}.$$

Thus

$$\delta'(q_0', xa) = [r_1, r_2, \ldots, r_k]$$

if and only if

$$\delta(q_0, xa) = \{r_1, r_2, \ldots, r_k\}.$$

Hence, the inductive hypotheses is True

$^1(q_0^1, x)$ is in F^1 exactly when (q_0, x)

Contains a state of Q that is in F

Thus L (M)=L(M^1)

FINITE AUTOMATA WITH -MOVES

The model of the nondeterministic finite automaton can be extended to include transitions on the empty input. Because of moves we are able to change the state without reading any symbol

The transition diagram of such an NFA accepting the language consisting of any number (including zero) of 0's followed by number of l's followed by any number of 2's is given in Fig. 2.8

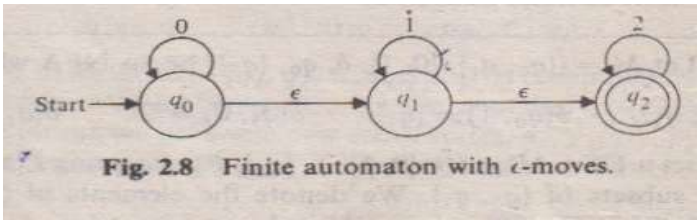


Fig. 2.8 Finite automaton with ϵ-moves.

The NFA accepts a string *w i*f there is some path labeled *w* from the initial state to a final state. The edges labeled may be included in the path, even though do not appear explicitly in *w*.

For example, the word 002 is accepted by the NFA given in the above Fig:

by the path q0, q0, q0, q1, q2, q2 with arcs labeled 0, 0, , , 2.

Definition: NFA with

An NDFA with is a quintuple M= (Q, ∑, , q_0, F)

Where Q→Set of non-empty set of Finite states

∑ →set of alphabets including

→transition function mapping from: Q X (∑ U{→2^Q

q_0 →initial state, q_0Q

F→ Set of Final states & (F ⊆ Q).

Epsilon-closure: CLOSURE of q (-closure(q)) denotes set of states which can be reached by Finite Automata without reading any input symbol from q.

transition define for -CLOSURE (p) where p is the set of states be union of q in p

$_{q \text{ in } p}$ **-CLOSURE(q)**

Properties:

- (q,) = -CLOSURE (q)
- For w in ∑* and a in ∑, (q,wa)= -CLOSURE (p), where P={ p |for some r in (q,w), p is in (r,a)}

  It is convenient to extend and to set of states by
- $=_{q \text{ in } R}$ (q,a) and
- (R,w)=$_{q \text{ in } r}$ (q,w)

Note:

q,a) is not always equal to q,a)

Therefore q,a) includes all states reachable from q by paths labeled 'a' (including path with arcs labeled),while q,a) includes only those states reachable from q by arcs labeled 'a'

L(M) the language accepted by M=(Q, ∑, , q_0, F) is defined as {w| (q,w) contains a state in F}

CONVERSION OF NONDETERMINISTIC SYSTEMS TO DETERMINISTIC SYSTEMS

The following are the steps to convert NDFA to DFA

Step 1 Convert the given transition system into state transition table where each state corresponds to a row and each input symbol corresponds to a column.

Step 2 Construct the successor table which lists the subsets of states reachable from the set of initial states

Step 3 The transition graph given by the successor table is the required deterministic system. The final states contain some final state of NDFA. If possible, reduce the number of states.

Problem:
Obtain the deterministic graph (system) equivalent to the transition system given in Fig. 5.11.

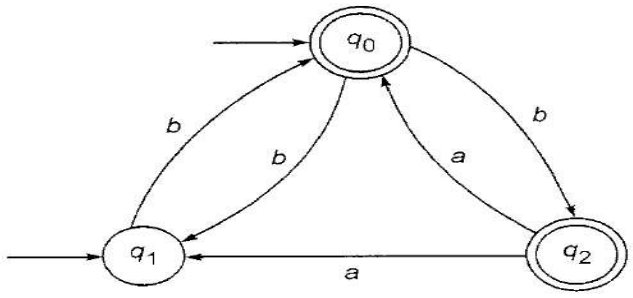


Fig. 5.11 Nondeterministic transition system of Example 5.7.

Solution

We construct the transition table corresponding to the given nondeterministic system. It is given in Table 5.1.

TABLE 5.1 Transition Table for Example 5.7

State/Σ	a	b
→q_0		q_1, q_2
q_1		q_0
q_2	q_0, q_1	

We construct the successor table by starting with $[q_0, q_1]$. From Table 5.1 we see that $[q_0, q_1, q_2]$ is reachable from $[q_0, q_1]$ by a b-path. There are no a-paths from $[q_0, q_1]$.

Similarly, $[q_0, q_1]$ is reachable from $[q_0, q_1, q_2]$ by an a-path and $[q_0, q_1, q_2]$ is reachable from itself. We proceed with the construction for all the elements in Q^1 in $\sum$

We terminate the construction when all the elements of Q^1 appear in the successor table. Table 5.2 gives the successor table. From the successor table it is easy to construct the deterministic transition system described by Fig. 5.12

TABLE 5.2 Deterministic Transition Table for Example 5.7

Q	a	b
$[q_0, q_1]$	∅	$[q_0, q_1, q_2]$
$[q_0, q_1, q_2]$	$[q_0, q_1]$	$[q_0, q_1, q_2]$
∅	∅	∅

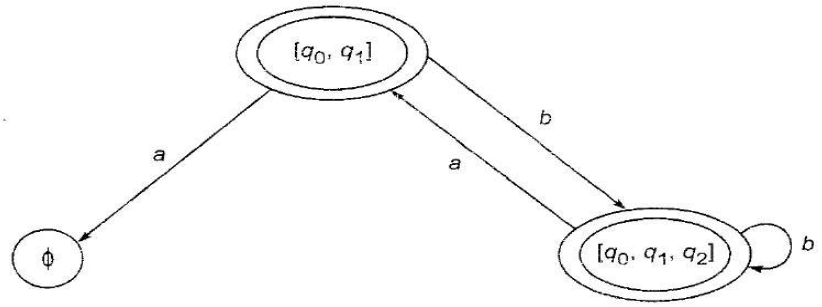


Fig. 5.12 Deterministic transition system for Example 5.7.

As q_0 and q_2 are the final states of the nondeterministic system $[q_0, q_1]$ and $[q_0, q_1, q_2]$ are the final states of the deterministic system.

Equivalence of NFA's with and without -moves

Statement: If L is accepted by an NFA with transitions, then L is accepted an NFA without transitions

Proof:

Let M =(Q, ∑, , q_0, F) be an NFA with transitions

Construct M^1= (Q, ∑, ^{1}, q_0, F^1) where

$$F' = \begin{cases} F \cup \{q_0\} & \text{if } \epsilon\text{-CLOSURE}(q_0) \text{ contains a state of } F, \\ F & \text{otherwise,} \end{cases}$$

^{1}(q,a) is **q,a) for q in Q and a in** ∑

M^1 has no transitions thus we use ^{1} for but we must continue to distinguish between and

We have to show by induction on that ^{1}(q_0, x) =**q_0,x)**

The statement may not be true for x=
Since q,) ={q_0 }

Therefore we begin our induction at 1

Basis: =1 then x is a symbol a and ^{1}(q,a)= q,a) by definition of ^{1}

Induction: 1. Let x=wa for symbol a in ∑ then

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a).$$

By the inductive hypothesis ^{1}(q_0, w) =**q_0,w).**

Let q_0, w)=P.

^{1}**(q0, wa)=(q0,wa)**

=^{1}(^{1}**(q0,w),a)**

=^{1} **(P, a)**

we must show that

^{1}(P, a) = q_0,wa). **But**

$$\delta'(P, a) = \bigcup_{q \text{ in } P} \delta'(q, a) = \bigcup_{q \text{ in } P} \hat{\delta}(q, a).$$

Then as P= q_0, w), we have

$$\bigcup_{q \text{ in } P} \hat{\delta}(q, a) = \hat{\delta}(q_0, wa)$$

by rule (2) in the definition of **, Thus**

$$\delta'(q_0, wa) = \hat{\delta}(q_0, wa).$$

To complete the proof we have to show that $^{1}(q_0, x)$ contains a state in F1 if and only if **q_0,x) contains a state of F.**

If x= i,e. $^{1}(q_0,)=\{q_0\}$

And q_0 is placed in F^1

Whenever $^{1}(q_0,)$ **which is an -CLOSURE(q0) contains a state in F**

If x≠ and x=wa if q_0,x)

Contains a same state in F^1

Conversely, also true

If $^{1}(q_0,)$ contains q_0 and q_0 is not in F then **q_0,x)= -CLOSURE((q_0,w),a))**

Hence the theorem is proved.

Minimiziation Finite Automaton Algorithm.

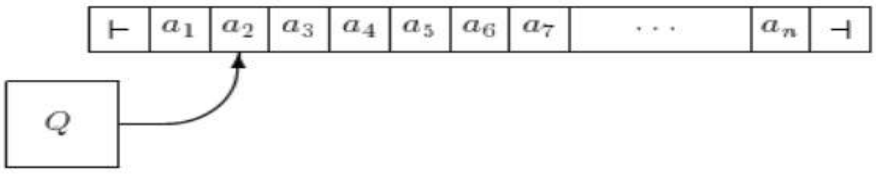
begin

1) *for p in F and q in Q-F do mark(p, q);*
2) *for each pair of distinct states (p, q) in F x F or (Q - F) x (Q - F) do*
3) *if for some input symbol a, ((p, a), b(q, a)) is marked then*
   begin
4) *mark (p. q);*
5) *recursively mark all unmarked pairs on the list for (p. q) and on lists of other pairs that are marked at this step.*
   end
   else / no pair ((p, a), (q, a)) is marked */*
6) *for all input symbols a do*
7) *put (p. q) on the list for ((p. a), (q, a)) unless (p, a) = (q, a)*
   end

Two-way Finite Automata

By extending the finite automata by align the tape R/W head to move the ability to move left as well as right. Such a finite automaton is called a two-way finite automaton.

Structure of 2DFA



Definitions:

A Two-way deterministic finite automaton (2DFA) is a quintuple M= (Q, ∑, , q_0, F), where

- Q is non empty set of finite states.
- ∑ is a non empty set of I/P symbols or alphabets
- q_0 Q is an initial state
- F ⊆Q is a set of final state.
- is a transition function i,e..: Q X ∑→Q X{L,R}
  - If (q, a) = (p. L), then in state 'q', scanning input symbol 'a', the 2DFA enters state 'p' and moves its head left one square.
  - If (q, a) = (p. R), then 2DFA enters state p and moves its head right one square.

Instantaneous Description:

The instantaneous description (ID) of a 2DFA, describes the i/p string, current state, and current position of the input R/W head.

The relation $\vdash_M$ on ID's such as I_1 $\vdash_M$ I_2 if and only if M can go from the instantaneous description I_1 to I_2 in one move.

An ID of M is a string in ∑*Q∑*. The ID wqx, where w and x are in ∑*, q is in Q, is intended to represent the facts that

1) wx is the input string,
2) q is the current state, and
3) the input head is scanning the first symbol of x.

If *x* =, then the input head has moved off the right end of the input. We define the relation $\vdash_M$ or just $\vdash_M$ if M is understood, by

1) $a_1,a_2\ldots a_{i-1}\ qa_i\ldots\ldots a_n \vdash a_1\ a_2\ a_{i-1}\ a_i\ pa_{i+1}\ldots\ldots a_n$ whenever $(q, a_i) = (p. R)$, and

2) $a_1,a_2\ldots a_{i-2}\ a_{i-1}\ qa_i\ldots\ldots a_n \vdash a_1\ a_2\ldots\ a_{i-2}\ pa_{i-1}a_i\ldots\ldots a_n$ whenever $(q, a_i)=(p. L)$ and $i> 1$.

We Define $L(M)=\{w|\ q_0w \vdash wp$ for some p in F$\}$

i.e., w is accepted by M if starting state q_0 with w on the input tape and R/w head at the end of w. M enters the final state at the same time it falls off the right end of the input tape.

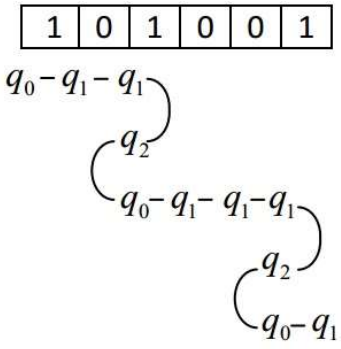
	0	1
q_0	(q_0, R)	(q_1, R)
q_1	(q_1, R)	(q_2, L)
q_2	(q_0, R)	(q_2, L)

$$
\begin{aligned}
q_0101001 &\vdash 1q_101001\\
&\vdash 10q_11001\\
&\vdash 1q_201001\\
&\vdash 10q_01001\\
&\vdash 101q_1001\\
&\vdash 1010q_101\\
&\vdash 10100q_11\\
&\vdash 1010q_201\\
&\vdash 10100q_01\\
&\vdash 101001q_1
\end{aligned}
$$

Crossing sequences:

The behavior picture of a 2DFA consists of the input, the path followed by the head, and the state each time the boundary between two tape squares is crossed, with the assumption that the control enters its new state prior to moving the head.

For example, the behavior of the 2DFA 'M' of Example 2.14 on string 1010011 shown in Fig. 2.19.



The list of states below each boundary between squares is term as crossing sequence.

Observations:

- The first time the boundary is crossed, the head must be moving right,
- Thus odd-numbered elements of a crossing sequence represent right moves and
- Even-numbered elements represent left moves.
- If the input is accepted, it follows that all crossing sequences are of odd length.
- A crossing sequence $q_1, q_2, ..., q_k$ is said to be valid if it is of odd length, and no two odd- and no two even-numbered elements are identical.
- The number of valid crossing sequences is finite.

We construct a NFA (non-deterministic finite automata) that will simulate the 2DFA M and whose states are the valid crossing sequences of M.

In order to construct the transition function we first examine the relationship between adjacent crossing sequences.

Suppose we are given an isolated tape square holding the symbol **a** and are also given valid crossing sequences $q_1, q_2,\ldots q_k$ and $p_1,p_2\ldots p_l$ at the left and the right boundaries of the square, respectively

We can test two sequences for local compatibility as follows:

- If the tape head moves left from the square holding a in state q_i, restart the automaton on the square holding a in state q_{i+1}
- If the tape head moves right from the square holding a in state p_i, restart the automaton on the square holding **a** in state p_{i+1}

We take $q_1, q_2,\ldots q_k$ to appear at the left boundary of **a** and $p_1,p_2\ldots p_l$ at the right boundary of **a** then,

We define right matching and left matching phase of crossing sequences recursively in one to 5 below.

Case 1:

The null sequence left- and right-matches the null sequence. That is, it will never reach the square holding **a**, then it is consistent that the boundaries on neither side should be crossed.

Case 2:

If $q_3\ldots q_k$ right matches $p1,\ldots p_l$ and $(q_1, a) = (q_2, L)$, then $q_1\ldots q_k$ right-matches $P_1\ldots p_l$, That is, if the first crossing of the left boundary is in state q1 and the head immediately moves left in state q2, then if we follow these two crossings by any consistent behavior starting from another crossing of the left boundary, we obtain a consistent pair of sequences with first crossing moving right, i.e., a right-matched pair.

Case 3:

If q_1,…. q_k left-matches p_2 ..., p_l, and $(q_1, a) = (p_1, R)$, then q_1, ..., q_k right matches p_1… p_l. That is, if the first crossing of the left boundary is in state q1 and the head immediately moves right in state p1, then if we follow these two crossings by any consistent behavior starting from a crossing of the right boundary, we obtain a consistent pair of sequences with the first crossing from the left, i.e., a right-matched pair.

Case 4:

If q_1… q_k left-matches p_3,... .p_l and $(p_1, a) = (p_2, R)$, then q_1, ... qk left matches P1….p_l. The justification is similar to that for rule (ii).

Case 5:

If q2….. ,qk right-matches p2 ..., p_l and (p_1, a) (q_1, L), then q1, ..., qk ,left matches p_1,……p_l. The justification is similar to rule (iii).

FINITE AUTOMATA WITH OUTPUT

One limitation of the finite automaton as we have defined it is that its output is limited to a binary signal: **"accept or don't accept".**

Finite automata may have outputs corresponding to each transition. There are two types of finite state machines that generate output −

- Mealy Machine
- Moore Machine

Moore Machine: Moore machine is an FSM whose outputs depend on only the present state. A Moore machine is a six- tuple M= (Q, ∑, , , λ, q_0), where

- Q is a non-empty set of finite sates.
- ∑ is a set of inputs symbols or alphabets
- , is a set of output alphabet.
- is a transition function mapping from : Q X ∑→Q and:
- **λ: Q**→and
- q_0 is the initial state

Note that any Moore machine gives output λ (q_0) in response to input.

For example:

For example Table 3.8 describes a Moore machine. The initial state q_0 is marked with an arrow. The table defines and λ

TABLE 3.8 A Moore Machine

Present state	Next state δ		Output
	$a = 0$	$a = 1$	λ
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

For the input string 0111, the transition of states is given by $q_0 \rightarrow q_3 \rightarrow q_0 \rightarrow q_1 \rightarrow q_2$.
The output string is 00010. For the input string, the output is $\lambda(q_0) = 0$

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

A mealy machine is also a six-tuple M = (Q, ∑, , , λ,q_0), where

- Q is a non-empty set of finite sates.
- ∑ is a set of inputs symbols or alphabets
- , is a set of output alphabet.
- is a transition function mapping from : Q X ∑→Q and:
- λ is an output function mapping from **λ: Q X** ∑→

For the input string 0011, the transition states is given by q1→ q3→ q2→ q4→ q3,and the output string is 0100

In the case of Mealy machine, we get an output only on the application of an input symbol. So for input string the output is only

For example:

Transition Table .3.9 describes a Mealy machine.

TABLE 3.9 A Mealy Machine

Present state	Next state			
	$a = 0$		$a = 1$	
	state	output	state	output
$\rightarrow q_1$	q_3	0	q_2	0
q_2	q_4	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Note: For the input string 0011, the transition of states is given by $q_1 \rightarrow q_3 \rightarrow q_2 \rightarrow q_4 \rightarrow q_3$, and the output string is 0100.

In the case of a Mealy machine we get an output only on the application of an input symbol. So for the input string the output is only .It may be observed that in the case of a Moore machine, we get $\lambda (q_o)$ for the input string

Procedure for transforming a Mealy machine into a Moore machine

Input: Mealy Machine

Output: Moore Machine

Step 1 Calculate the number of different outputs for each state (Q_i) that are available in the state table of the Mealy machine.

Step 2 If all the outputs of Q_i are same, copy state Q_i. If it has **n** distinct outputs, break Q_i into **n** states as Q_{in} 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.

Procedure for transforming a Moore machine to Corresponding Mealy machine

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 Check the present states and their corresponding outputs in the Moore Machine state table; if for a state Q_i output is m, copy it into the output columns of the Mealy Machine state table wherever Q_i appears in the next state..