CHAPTER 1

Automata Theory

University Prescribed Syllabus

Automata Theory : Defining Automaton, Finite Automaton, Transitions and Its properties, Acceptability by Finite Automaton, Nondeterministic Finite State Machines, DFA and NDFA equivalence, Mealy and Moore Machines, Minimizing Automata

1.1 Some Mathematical Terms

Before we start theory of Computation, review some maths terms.

1.1.1 Sets

Informally, a set is a (finite or infinite) collection of distinct elements; the order of elements does not matter.

Finite examples

{0,1,2} = Set of three numbers.

{1,0,2} = The **same** set of three numbers.

{0,0,2} = Not a set (since it has a duplicate).

ϕ = The empty set.

Infinite examples

{0, 1, 2, 3,...} = Set of natural numbers

{i : i > 0 and i is even} = Set of positive even numbers

$x \in S$ = x is an element of S, e.g. $1 \in \{0,1,2\}$

$S_1 \subseteq S_2$ = S_1 is a subset of S_2 ; that is, every element of S_1 is in S_2.

e.g. $\{0, 1\} \subseteq \{0, 1, 2\}$

$\{i : i > 0$ and i is even$\} \subseteq \{0, 1, 2, 3, ...\}$

WRONG : $1 \subseteq \{0,1,2\}$

RIGHT : $1 \subseteq \{0,1,2\}$ or $\{1\} \subseteq \{0,1,2\}$

(Since 1 is not a set).

A. Set operations

1. Union
2. Intersection
3. Difference
4. Complementation

1. Union

$S_1 \cup S_2$ = All elements from S_1 and S_2.

e.g. $\{0\} \cup \{1, 2\} = \{0, 1, 2\}$

$\{0,1\} \cup \{1, 2\} = \{0, 1, 2\}$

$\{i : i > 0$ and i is even$\} \cup \{i : i > 0$ and i is odd$\} = \{1,2,3,.....\}$

2. Intersection

$S_1 \cap S_2$ = Elements that are in both sets.

e.g. $\{0,1,2\} \cap \{1,2,3\} = \{1,2\}$

$\{i : i > 0\} \cap \{i : i < 3\} = \{1,2\}$

$\{i : i$ is even$\} \cap \{i : i$ is odd$\} = \phi$

3. Difference

$S_1 - S_2$ = Elements that are in S_1 and not in S_2.

e.g. $\{0, 1, 2\} - \{0,2\} = \{1\}$

$\{0, 1, 2\} - \{2, 3\} = \{0,1\}$

$\{i : i$ is even$\} - \{i : i$ is odd$\} = \{i : i$ is even$\}$

4. Complementation

- We often consider a fixed **universal set** (also known as universe) of all possible values, denoted U.

  e.g. Set of all integer numbers

  Set of all computer programs

  Set of all possible grades for the course

- For a set $S \subseteq U$, the complement of S includes all elements of U that are not in S.

For examples

1. U is a set of all final grades, from 0 to 100.

   $S = \{x : x \geq 50\}$ – passing grades.

   $\bar{S} = \{x : x < 50\}$ – failing grades.

2. U is a set of integers

   $S = \{i : i$ is even$\}$

   $\bar{S} = \{i : i$ is odd$\}$

(i) Size of a set

$|S|$ = Number of elements

e.g. $|\{0,1,2\}| = 3$

$|\phi| = 0$

$|\{i : i$ is even$\}| = \infty$

(ii) Power set

2^s = Set of all subsets of S

e.g. $2^{\{1,2\}} = \{\phi, \{1\}, \{2\}, \{1, 2\}\}$

$2^{\{1,2,3\}} = \{\phi, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

$2^{\phi} = \{\phi\}$

If $|S| = n$, then $|2^S| = 2^n$; that is, $|2^S| = 2^{|S|}$

B. Cartesian product

$S_1 \times S_2$ = List of all possible pairs with the first element from S_1 and the second element from S_2.

e.g. $\{1,2\} \times \{a, b\} = \{(1,a), (1,b), (2,a), (2,b)\}$

$\{1,2\} \times \{1,2,3\} = \{(1,1), (1,2), (1,3), (2,1), (2,2), (2,3)\}$

If S_1 is a list of boys, and S_2 is a list of girls, then $S_1 \times S_2$ is the list of possible marriages.

$S_1 \times S_2 \times \ldots \times S_n = \{(x_1, x_2, \ldots x_n) : x_1 \in S_1, x_2 \in S_2, \ldots, x_n \in S_n\}$

e.g. $\{1,2\} \times \{a, b\} \times \{X, Y\}$

$= \{(1,a,X), (1,a,Y), (1,b,X), (1,b,Y))$

$(2,a,X), (2,a,Y), (2,b,X), (2,b,Y)\}$

$|S_1 \times S_2 \times \ldots \times S_n| = |S_1| \cdot |S_2| \ldots |S_n|$

C. Simple laws

$S \cup \phi = S$ $\quad \overline{\phi} = U$

$S - \phi = S$ $\quad$ If $S_1 = \overline{S}_2$, then $S_2 = \overline{S}_1$; $\quad$ that is, $\overline{S} = S$. $S \cap \phi = \phi$

Fig. 1.1.1

D. De Morgan's laws

$\overline{S_1 \cup S_2} = \overline{S}_1 \cap \overline{S}_2$

$\overline{S_1 \cap S_2} = \overline{S}_1 \cup \overline{S}_2$

Fig. 1.1.2

E. Functions and Relations

1. Function : $f : S_1 \rightarrow S_2$

- For each element of S_1, a function determines exactly one element of S_2 (in the textbook, it is called a total function).
- S_1 is the **domain** of f. Set of all elements in S_2 "pointed" by f is the **range** of f as shown in Fig. 1.1.3.

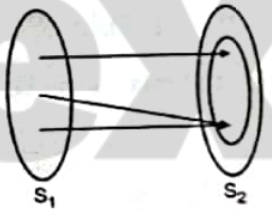


Fig. 1.1.3

Domain : {1,2,3}

Range : {a,b}

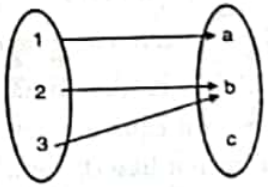


Fig. 1.1.4

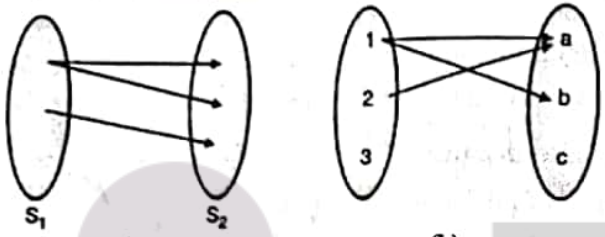
$f(X) = X^2$

Domain : All real numbers

Range : All non-negative real numbers

2. Relation

- For each element of S_1, a relation determines several (possibly none) elements of S_2.



(a) (b)

Fig. 1.1.5

- e.g. Mapping from students to the classes they take–a student can take zero, one, or more classes.

Equivalence

- **A special case of a relation**, usually denoted by "≡", which satisfies three **properties** :

1. $X \equiv X$ (**reflexivity**)
2. if $X \equiv Y$, then $Y \equiv X$ (**symmetry**)
3. If $X \equiv Y$ and $Y \equiv Z$, then $X \equiv Z$ (**transitivity**)

- For example (i) "Living in the same city" is an equivalence. (ii) Liking someone is not an equivalence : If John like Mary and Mary like Donald, John may not like Donald.

More examples

(i) X = Y = Equivalence

X and Y have the same sign = Equivalence,

(ii) X < Y = Not equivalence

1.1.2 Graphs and Trees

1. Graph

- **A graph** is a collection of vertices connected by directed edges (e.g intersections connected by one-way streets).

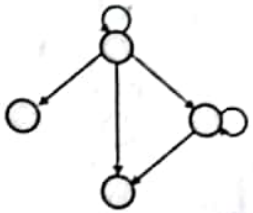


Fig. 1.1.6

- **A path** is a "walk" from one vertex to another along directed edges, where each edge is visited at most once.

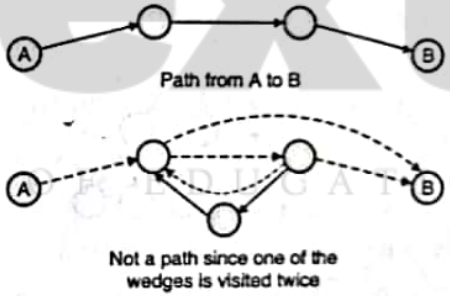


Fig. 1.1.7

- If a path visits each vertex at most once, it is a simple path.

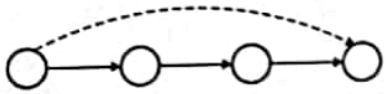


Fig 1.1.8

- A cycle is a path from a vertex to itself.

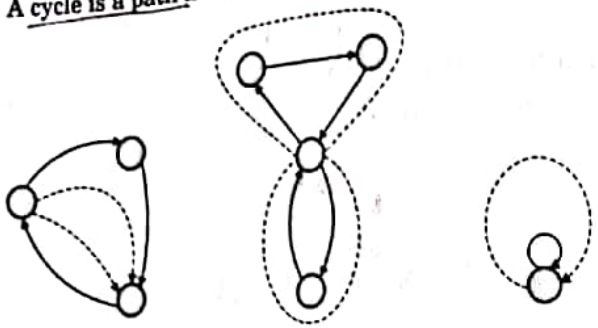


Fig. 1.1.9

2. Trees

- A tree is a special case of a graph. It has a **root** vertex and exactly one path from the root to any other vertex.

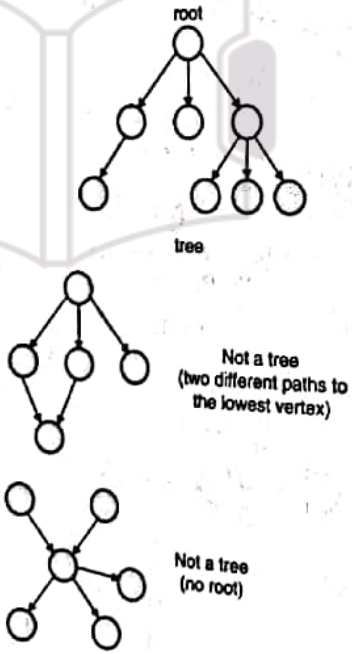


Fig. 1.1.10

- If there is an edge from A to B, then A is a **parent** of B, and B is a **child** of A.

Fig. 1.1.11

- A node without children is a **leaf.**

1.1.3 Proof Techniques

A proof is a sequence of steps that lead from some known facts to the desired conclusion; each step must be **obviously** connect.

A. Proof by contradiction

- To prove some fact P, we show that "not P" is false. That is, we suppose "not P" and demonstrate that it leads to an obviously wrong result.

For example : Prove that $\sqrt{2}$ is not rotational.

- Suppose that it is rotational, that is $\sqrt{2} = \frac{n}{m}$, where n and m do not have a common actor.
- Then, $2 \cdot m^2 = n^2$, which implies that n is even, $n = 2k$. Thus $2 \cdot m^2 = 4 \cdot k^2$, which means that $m^2 = 2 \cdot k^2$; hence, m is even. We conclude that m and n are both even, contradicting the assumption that they do not have a common factor.

B. Proof by induction

- We show that some fact is true for every natural number n using two arguments :

1. **Base :** It is true for n = 1 (or for some other small numbers).
2. **Step :** If it is true for n, then it is also true for (n + 1)

   True for 1 = True for 2 = True for 3 = ...

   Informally, it is like an infinite loop.

For example : Prove that $1 + 2 + ... + n = \frac{n(n+1)}{2}$

Base : $n = 1, i = \frac{1+(1+1)}{2}$ = True

Step : Suppose that it is true for n, that is, $\underbrace{1 + 2 + ...+ n}_{\text{Inductive hypothesis, also known as inductive assumption}} = \frac{n(n+1)}{2}$

$$1 + 2 + 3 + ...+ n + (n+1) = \frac{n(n+1)}{2} + (n+1)$$

$$\frac{n(n+1)}{2} = \frac{n(n+1) + 2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

$$= \frac{(n+1)((n+1)+1)}{2}$$

1.2 Basic Concepts

1.2.1 Strings

- A **string** is a finite sequence of symbols; strings are usually denoted by u, v and w.

  e.g. u = abcab is a string on Σ = {a, b, c}
- The **empty string** (no symbols at all) is denoted λ.
- A part of a string is a **substring**.

  e.g. bca is a substring of abcab.
- A beginning of a string (upto any symbol) is a prefix and an ending is a suffix.

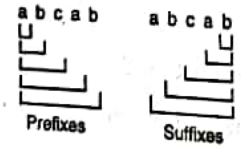


Fig. 1.2.1

Note : A string is a prefix and suffix of itself.

λ is a prefix and suffix of any string.

String operations

1. **Concatenation :** wv = Appearing v to the end of w.

   e.g. w = abc, v = ab

   wv = abc ab

   w v

Note : λw = wλ = w.

2. **Reverse :** w^R = w in reverse order.

   e.g. w = abc, w^R = cba
3. **Length of a string :** |w| = Number of symbols

   e.g. |abc| = 3

Note : |λ| = 0

$|w^R| = |w|$

|wv| = |w| + |v|

w^n = w repeated n times; that is, $w^n = \underbrace{www.....w}_{\text{n times}}$

e.g. $(abc)^3$ = abc abc abc

Note : $w^0 = \lambda$

$|w^n| = n \cdot |w|$

1.2.2 Languages

- Σ = An alphabet is a non-empty finite set of symbols,

  E.g. Σ = {a, b, c} is an alphabet.

- Σ^* is the set of all strings on Σ.

e.g. $\Sigma = \{a, b, c\}$

$\Sigma^* = \{\lambda, a, b, c, aa, ab\ ac, ba, ...\}$

- Σ^+ is the set of all strings except λ.

Note : Σ^* and Σ^+ are infinite.

- A **language** is a subset of Σ^*, usually denoted L.
- It may be finite or infinite.

e.g. : {a, ba, abc}

{λ, a, aa, aaa, ...}

ϕ

- If a string w is in L, we say that w is a **sentence** of L.

Operations on Languages

- Union, intersection and difference are same as on sets.

1. **Intersection :** e.g. {a, ba, abc} $\cap$ {λ, a, aa, aaa, ...} = {a}
2. **Complementation :** $\bar{L} = \Sigma^* - L$

e.g. L = {λ, a, aa, aaa, ...}

L = {w : w includes b or c}

3. **Reverse :** $L^R = \{w^R : w \in L\}$ = The set of reversed strings.

e.g. L = {a, ba, abc}

$\bar{L}$ = {a, ab, cba}

4. **Concatenation :** $L_1 L_2 = \{wv : w \in L_1 \text{ and } v \in L_2\}$

= Similar to the Cartesian product

e.g. L_1 = {a, ba, abc}

L_2 = {bcb, b}

$L_1 L_2$ = {a bcb, a b, ba bcb, ba b, abc bcb, abc a}

$L^n = \underbrace{L\,L\,....\,L}_{n \text{ times}}$ = L concatenated with itself n times

e.g. L = {a, aa}

L^3 = {a, aa} · {a, aa} · {a, aa} = {aaa, aaaa, aaaaa, aaaaaa}.

Note : $L^* = \{\lambda\}$

$L^* = L$

5. **Star-closure** (also known as Kleene star) : $L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup ....$

e.g. L = {a, aa}

L^* = {λ, a, aa, aaa, aaaa, ...}

6. **Positive closure (also known as Kleene plus)**
$L^+ = L^1 \cup L^2 \cup L^3 \cup ....$

e,g. {a, aa}$^+$ = {a, aa, aaa, ...}

{λ, a, aa}$^+$ = {λ, a, aa, aaa, ...}

Note : L^+ includes λ if and only if L includes λ.

1.3 TOC : Theory of Computation

- The theory of computation is a branch of Computer Science that deals with whether and how efficiency problems can be solved on a model of computation, using an algorithm.
- This field is divided into three major branches :
  1. Automata theory
  2. Computability theory
  3. Computational complexity theory
- Automata theory is the study of abstract mathematical machines called automata and the problems that can be solved using these machines.

- An automata is characterized by a number of states it can be in, a number of trensitions between those states and an alphabets of symbols it accepts.

Syllabus Topic : Defining Automaton

1.3.1 Definition of an Automaton

What is Automata ?

- **Automaton** are abstract models of machines that perform computations on an input by moving through a series of states or configurations.
  - At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration.
  - As a result, once the computation reaches an accepting configuration, it accepts that input.
- The most general and powerful automata is the **Turing machine.**
- The **major objective** of automata theory is to develop methods by which computer scientists can describe and analyze the dynamic behaviour of discrete systems, in which the signals are sampled periodically.
- The behaviour of these discrete systems is determined by the way that the system is constructed from storage and combinational elements.

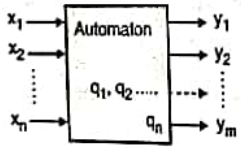


Fig. 1.3.1 : Model of a discrete automaton

1.3.2 Characteristics of an Automata

Write characteristics of an Automata ?

- **Inputs :** It is nothing but sequences of symbols selected from a finite set I of input signals.

For example set I is the set of $(x_1, x_2, x_3, \ldots x_n)$ where n is the number of inputs.

- **Outputs :** It is the sequences of symbols selected from a finite set O.

  For example set O is the set of $(y_1, y_2, y_3, \ldots, y_m)$ where m is the number of outputs.
- **States :** It is finite set Q, whose definition depends on the type of automaton.
- **State relation :** The state relation defines the status of next state which is determined by the current state and the current input at that instance of time.
- **Output relation :** The output depends on either state only or to both input and state.

Control Unit

- The most **important feature** of the automaton is its control unit. It can be in any one of the finite number of interval states at any point.
- It can change state in some defined manner which is determined by a transition function.
- The Fig. 1.3.2 shows a diagrammatic representation of a generic automation.

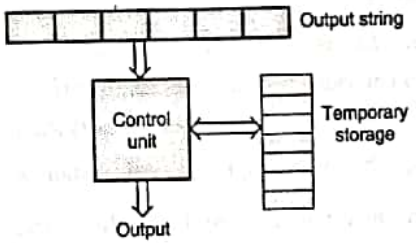


Fig. 1.3.2

- There are **four major families of automaton**
  1. Finite-state machine
  2. Pushdown automata

3. Linear-bounded automata
4. Turing machine

1.3.3 Automaton Types

- **Automaton without a memory :** output depends only on the input
- **automaton with a finite memory :** output depends on the input as well as state.
- **Moore machine:** the output depends only on the states of the machine
- **Mealy machine :** the output depends on the state as well as on the input at any instant of time

1.3.4 Automaton Labels

- The states are represented by circles
- The transitions are represented by arrows.
- Initial state is represented by arrow with Circle

Final state is represented by concentric Circle

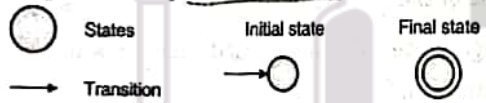


Fig. 1.3.3

- As the automaton sees a symbol of input, it makes a transition (or jump) to another state. This is as per its transition function which takes the current state and the recent symbol as its inputs.

Exam

1.3.5 Uses of Automata : Compiler Design and Parsing

Every automaton fulfils the following basic requirements :

1. Every automaton has a mechanism for reading input.
   - The input is assumed to be a sequence of symbols represented with an 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. Complex versions of automata can both read in any direction and can also change the input.
2. The automaton can produce output of some form
   - If the output response to an input string is binary (accept/reject or yes/no, or true/false etc), then it is called an accepter.
   - If the output response is a sequence to an input sequence, then it is called a transducer (or automaton with output).
3. The automaton may have a temporary storage consisting of an unlimited number of cells. Each cell may be capable of holding a symbol from an alphabet (which may be different from the input alphabet).
   - The automaton can read and change the contents of the storage cells in the temporary storage.
   - The accumulating capability of this storage varies depending on the type of the storage.

1.3.6 Operation of the Automaton

Write operation of an Automata.

- At any point of time the automaton is in some integral state and is reading a particular symbol from the input tape by using the mechanism for reading input.
- In the next time step, the automaton then moves to some other integral state (or remains in the same state) as defined by the transition function.
- The transition function is based on the current state, the input symbol that has been read and the contents of the temporary storage.
- It is also possible that at the same time the contents of the storage may be changed and the input that was read may be modified.
- The automation may also produce some output during transition.
- The internal state, input and the content of storage at any point defines the configuration of the automaton at that point.

- The transition from one configuration to the next (as defined by th transition function) is called a move.
- Any system, that is at any point of time in one of the finite number o interval state and moves among these states in a defined manner i response to some input, can be modelled by a **finite automaton**.
- It doesn't have any temporary storage and hence a restricted model o computation.

Syllabus Topic : Transition and It's Properties

1.4 Transition Systems

Transition graph can be interpreted as a flowchart for an algorithm recognizing a language. A transition graph consists of following things :

1. A transition graph or a transition system is a finite directed labele graph in which each vertex (or node) represents a state and the directed edges indicate the transition of a state and the edges are labelled with input/output.
2. In a finite set of states, at least one state should be the start state and some of which are designated as final states.
   - Initial state is represented by arrow with Circle
   - Final state is represented by concentric Circle
5. An alphabet Σ of possible input symbols from which the input strings are formed. Example : The edges are labeled by input / output (e.g. by 1/0 or 1/1 or a/b).
6. A finite set of transitions that show the change of state from the given state on a given input.

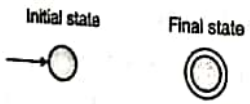


Fig. 1.4.1

- A transition system or transition graph is a finite directed labelled graph in which each vertex represents a state and the directed edges indicate of a state and the edges are labelled with input/output.
- Fig. 1.4.2 shows a transition system and initial and final states.

q_0 is Initial state

q_1 is Final State.

0/1 are inputs and outputs

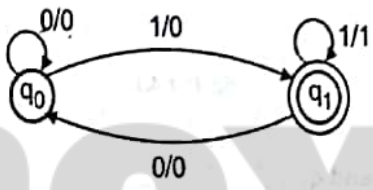


Fig. 1.4.2

1.4.1 Definition of a Transition System

Define transition system in brief.

1. A transition system is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$.
   - Q = the finite nonempty set of states
   - Σ = the input alphabet
   - F = the set of final states,.
   - $q_0 \in Q$ and q_0 is nonempty
   - δ is a finite subset of $Q \times \Sigma^* \times Q$.

   Imp

   In other words, if (q_1, w, q_2) is in δ, it means that the graph starts at the vertex q_1, goes along a set of edges, and reaches the vertex q_2. The concatenation of the label of all the edges thus encountered is w.
2. A transition system accepts a string w in Σ^* if
   - There exists a path which originates from some initial state, goes along the arrows, and terminates at some final state.

- The path value obtained by concatenation of all edge-labels of the path is equal to w.

Example 1.4.1 : Consider the transition system given in Fig. P. 1. 3.1. Determine the initial states, the final states and the acceptability of 101011,111010.

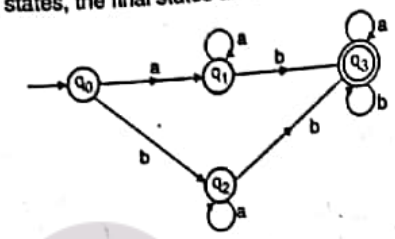


Fig. P. 1.4.1

Solution :

1. Initial states : q_0 and q_1
2. Final state : q_3
3. For string : 101011

   The path-value is $q_0 - q_0 - q_2 - q_3$

   As q_3 is the final state, 101011 is accepted by the transition system.

For string 111010 is not accepted by the transition system as there is no path with path value 111010 because here transition system having two initial states.

1.4.2 Transition Table

- A transition table is a tabular representation of the transition function.
- In table, the column contains the state in which the automaton will be on the input represented by that column. The row corresponds to the state, the finite control unit can be in.
- The entry for one row corresponding to state q and the column corresponds to input a is the state δ (q, a).
- For the transition graph in the Fig. 1.4.3 the transition table would be as shown in Table 1.4.1.

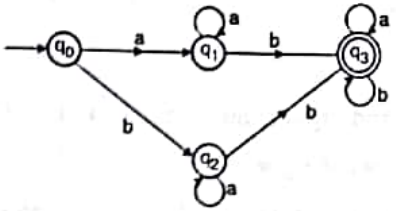


Fig. 1.4.3 : Transition diagram

Table 1.4.1

States	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_2	q_3
*q_3	q_3	q_3

Following are the steps to draw a state diagram or transition graph from the transition table.

1. Draw the circles to represent the states given.
2. For each of the states, scan across the corresponding row and draw an arrow to the given state in transition table..

   There can be multiple arrows for an input character if the automaton is an NFA.
3. Designate a state as the start state. The start state is given in the formal definition of the automaton.
4. Designate one or more states as accept state. This is also given in the formal definition.

1.4.3 Properties of Transition Functions

Write transition properties in brief.

Property 1

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

- It requires valid input.

Property 2

For all strings w and input symbols a, $\delta(q, aw) = \delta(\delta(q, a), w)$.

$$\delta(q, wa) = \delta(\delta(q, w) a)$$

- 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.

Syllabus Topic : Finite Automaton

1.5 Finite Automata

- **Finite state machine** or **finite automation** is the simplest type of **abstract machine.**
- Automata (singular : automation) are simple but useful model of computation.

1.5.1 States, Transitions and Finite – State Transition System

☞ **Write components of finite automata.**

- **A state of a system** is an instantaneous description of that system which gives all relevant information necessary to determine how the system can evolve from that point onwards.
- **Transitions** are changes of states that can occur spontaneously or in response to inputs to the states.
- Some examples of state transition systems are : digital systems, vending machines, etc.
- A system containing only a finite number of states and transitions among them is called a **finite –state transition system.**
- Finite-state transition systems can be modelled abstractly by a mathematical model called **Finite automation.**

1.5.2 Finite Automaton Representation

A finite automaton can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

(i) **Q** is a finite non-empty **set of states.**

(ii) Σ is a finite non-empty set of inputs called the **input alphabet.**

(iii) δ is a function which maps $Q \times \Sigma$ into Q

- δ is called the direct transition function.
- This is the function which describes the change of states during the transition.
- Transition function is usually represented by a transition table or a transition diagram.

(iv) $q_0 \in Q$ **is the initial state.**

(v) $F \subseteq Q$ is the **set of final states.** (there may be more than one final states)

Syllabus Topic : Acceptability by Finite Automaton

1.5.3 Acceptability of a String by a Finite Automaton

- Consider $a_1, a_2, a_3, \ldots\ldots, a_n$ is a sequence of input symbols.
- $q_0, q_1, q_2, \ldots\ldots, q_n$ are set of states where, q_0 is start state and q_n is final state.
- The transition function is processed as,

$$\delta(q_0, a_1) = q_1$$
$$\delta(q_1, a_2) = q_2$$
$$\delta(q_2, a_3) = q_3$$
$$\ldots\ldots\ldots$$
$$\ldots\ldots\ldots$$
$$\delta(q_{n-1}, a_n) = q_n$$

- Input $a_1, a_2, a_3, \ldots\ldots, a_n$ is said to be 'accepted' since q_n is a member of the final state, and if not then it is 'rejected'.

- Language accepted by DFA, 'M' written as,

$$\{ L(M) = \{ w / \delta(q_0, w) = q_n \text{ for some } q_n \text{ in } F \}$$

Here, F means final state.

Final state is also called as **accepting state.**

For example

Consider the finite state machine whose transition function δ is given in Fig. form of a transition table.

Here, $Q = \{ q_0, q_1, q_2, q_3 \}$ and $\Sigma = \{ 0, 1 \}$, $F = \{ q_0 \}$

States \ Inputs	0	1
→ q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Give the entire sequence of states for the input string 110101.

$$\delta(q_0, 110101) = \delta(q_1, 10101)$$
$$\downarrow$$
$$= \delta(q_0, 0101)$$
$$\downarrow$$
$$= \delta(q_2, 101)$$
$$\downarrow$$
$$= \delta(q_3, 01)$$
$$\downarrow$$
$$= \delta(q_1, 1)$$
$$\downarrow$$
$$= \delta(q_0, \epsilon)$$
$$\downarrow$$
$$= q_0$$

Transition diagram

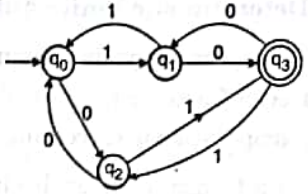


Fig. 1.5.1

The symbol ↓ indicates that the current input symbol is being processed by the machine

$$q_2 \xrightarrow{1} q_1 \xrightarrow{1} q_0\ q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_0$$

Syllabus Topic : Non-Deterministic Finite State Machines, DFA and NDFA Equivalence, Minimizing Automata, Mealy and Moore Machines

1.5.4 Finite Automata Classification

Write a classification of finite automata and explain in brief.

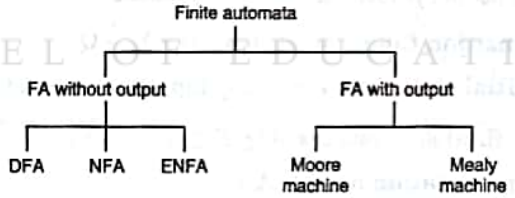


Fig. 1.5.2

Imp Exam

A. Finite Automata without Output

1. Deterministic finite automata (DFA)

DFA is defined as an abstract mathematical concept, but due to the deterministic nature of DFA it is implementable in hardware and software for solving various problems.

- In this concept, each input symbol determines 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.**
- A DFA has a start state (arrow symbol) from where the computations begin. It also has a set of accepted states (denoted by double circles) which define when a computation is successful.
- For each state there is a transition arrow leading out to a next state for each symbol.
- Deterministic means that there is only one outcome. It means move to next state when the symbol matches ($S_0 \rightarrow S_1$) or move back to the same state when the symbol matches ($S_0 \rightarrow S_0$)

Definition of a DFA

Q Define DFA.

A DFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,

- *Q is a finite **set of states.**
- Σ is a finite set of symbols called the **alphabet.**
- δ is the **transition function** where $\delta : Q \times \Sigma \rightarrow Q$
- 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 a DFA

A DFA is represented by digraphs called **state diagram.**

- The **vertices** represent the states.
- The arcs labelled 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 deterministic finite automaton be,

$$Q = \{q_0, q_1, q_2\}$$

$$\Sigma = (0, 1)$$

$$q_0 = \{q_0\}$$

$$F = \{q_2\}, \text{ and}$$

Transition function δ as shown by the Table.

Table 1.5.1

States \ Inputs	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Graphical representation would be as shown in Fig. 1.5.3

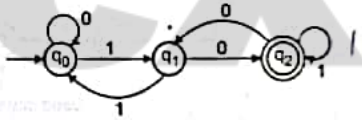


Fig. 1.5.3

Solved examples based on DFA

Example 1.5.1 : Design a DFA for language L_1 ;

L_1 = Set of all strings that state with 0

= {0, 00, 01, 00D , 010, 011, 0000, ...}

Solution :

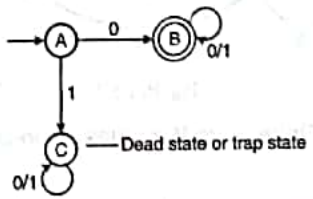


Fig. P. 1.5.1(a)

Dead state or trap state

e.g. 001 is a string

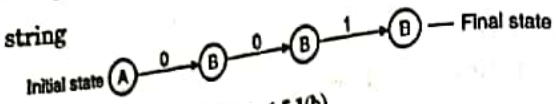


Fig. P. 1.5.1(b)

101 is a string

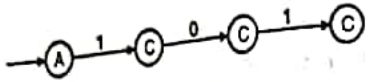


Fig. P. 1.5.1(c)

But C is not a final state, so this string is rejected.

Example 1.5.2 : Construct DFA that accepts sets of all strings over { 0, 1 } of lengths 2.

Solution :

$\Sigma = \{0, 1\}$, $L = \{00, 01, 10, 11\}$

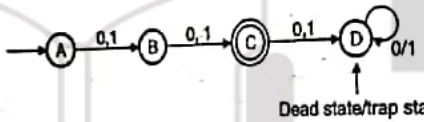


Fig. P. 1.5.2

Example 1.5.3 : Construct a DFA that accepts any string over {a, b} that contain the string aabb in it.

Solution : $\Sigma = \{a, b\}$

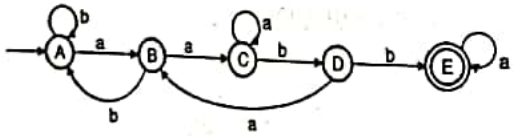


Fig. P. 1.5.3

2. Non-Deterministic Finite State Machines / Non-Deterministic Finite Automaton (NFA)

The Fig. 1.5.4 will explain the concept of non-deterministic finite automaton using a transition diagram.

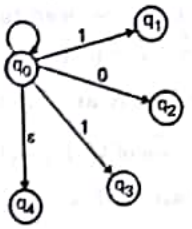


Fig. 1.5.4 : Transition system representing non-deterministic automaton

- In NFA, for the given current state, there could be multiple next states.
- The next state may be chosen at random.
- All the next states may be chosen in parallel.

Definition of Non-deterministic Automata

- For a particular input symbol, the machine can move to any combination of the states in the machine. Final state does not go anywhere.
- 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 Automation NDFA.

Example :

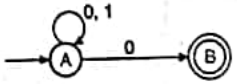


Fig. 1.5.5

Definition of an NDFA

☞ **Define NDFA.**

An NDFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where,

- Q is a finite **set of states .**
- Σ is a finite set of symbols called the **alphabets.**
- δ is the **transition function** where $\delta : Q \times \Sigma \rightarrow 2^Q$

- (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

- The **vertices** represent the **states.**
- The **arcs labelled 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.**

For example

Let a non-deterministic finite automaton be denoted as follows :

$Q = \{q_0, q_1, q_2\}$

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

$q_0 = \{q_0\}$

$F = \{q_2\}$ and

The transition function δ as shown below,

States \ Inputs	0	1
q_0	$\{q_0, q_1\}$	q_1
q_1	q_2	$\{q_0, q_2\}$
q_2	$\{q_1, q_2\}$	q_2

Graphical representation would be as shown in Fig. 1.5.6

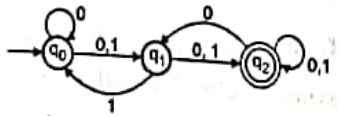


Fig. 1.5.6

NFA Examples :

L_1 = { Set of all strings that end with '1'}

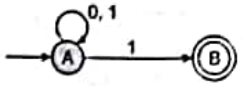


Fig. 1.5.7

L_2 = { Set of all strings that contain '0' }

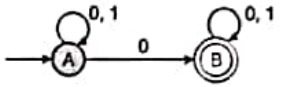


Fig. 1.5.8

L_3 = { Set of all strings that starts with '01' }

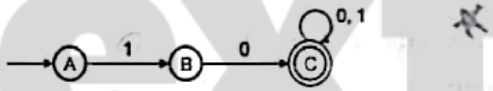


Fig. 1.5.9

L_3 = { Set of all strings that contain '01' }

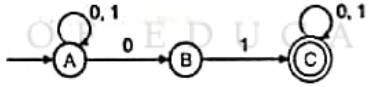


Fig. 1.5.10

L = { Set of all strings that ends with '11' }

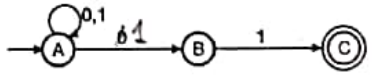


Fig. 1.5.11

Solved examples based on NFA

Example 1.5.4 : Design NFA for language L ;

L = { Set of all strings that start with 0 }

= { 0, 00, 01, 000, 010, 011, ... }

Solution :

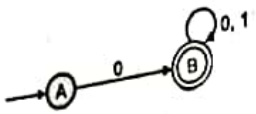


Fig. P. 1.5.4(a)

Initial state = A

Final state = B

Input alphabet = {0, 1}

Check for string 001,

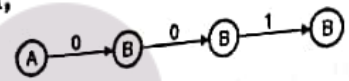


Fig. 1.5.4(b)

Last state = Final state

String is accepted

Check for string 101,

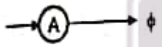


Fig. P. 1.5.4(c) : Dead configuration

String is not accepted

Example 1.5.5 : Construct NFA that accepts set of all strings over {0, 1} of length 2.

$\Sigma = \{0, 1\}$, $L = \{00, 01, 10, 11\}$

Solution :

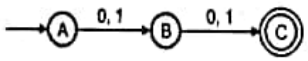


Fig. P. 1.5.5(a)

Check for string 00

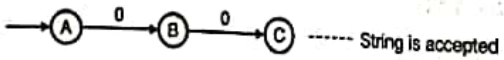


Fig. P. 1.5.5(b)

Check for string 001

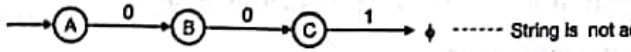


Fig. P. 1.5.5(c)

Imp Exam

Difference between DFA and NDFA

Write the difference between DFA and NDFA.

- The difference between the deterministic and non-deterministic automata is only in δ. $\delta : Q \times \Sigma \to Q$
- For deterministic automaton (DFA), the outcome is a state i.e. an element of Q. $\delta : Q \times \Sigma \to 2^Q$
- For non-deterministic automaton the outcome is a subset of Q.

Table 1.5.1

No.	DFA	NDFA
1.	The transition from a state is to a single particular next state for each input symbol. Hence it is called deterministic.	The transition from a state to multiple next states for each input symbol. Hence it is called non-deterministic.
2.	Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
3.	Backtracking is allowed in DFA.	Backtracking is not always possible.
4.	Requires more space.	Requires less space.
5.	A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state.
6.	Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.

Equivalence of two Finite Automata

Write steps to identify equivalence of two finite automata.

Steps to identify equivalence

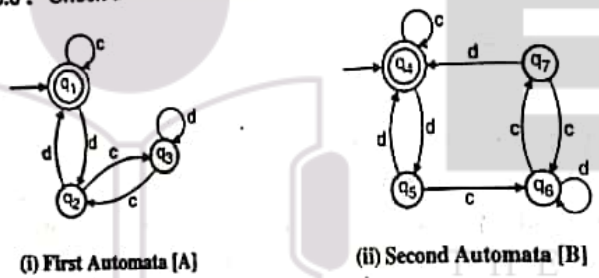
1. For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined $\{q_a, q_b\}$ where ;

   $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$

   The two automates are not equivalent if for pair $\{q_a, q_b\}$ one intermediate state and other is final state

2. If initial state is final state of one automation, then in second automa also, initial state must be final state for them to be equivalent.

Example 1.5.6 : Check these two automata are equivalent or not.



(i) First Automata [A]

(ii) Second Automata [B]

Fig. P. 1.5.6

Solution :

Check condition I

- For every pair of states the pair of states generates particular inp showable either both should be on intermediate state or both should on final state.

States	Input	
	C	D
$\{q_1, q_4\}$	$\{q_1, q_4\}$ FS FS	$\{q_2, q_5\}$ IS IS
$\{q_2, q_5\}$	$\{q_3, q_6\}$ IS IS	$\{q_1, q_4\}$ FS FS
$\{q_3, q_6\}$	$\{q_2, q_7\}$ IS IS	$\{q_3, q_5\}$ IS IS
$\{q_2, q_7\}$	$\{q_3, q_6\}$ IS IS	{q1, q4} FS FS

Note FS = final state

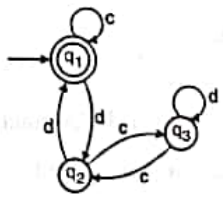
IS = Intermediate state

So first condition is satisfied.

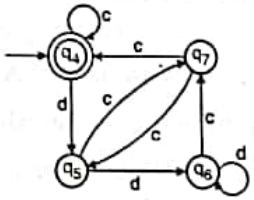
Check condition II

- Initial state and final state are same, in both automata.
- In first automata, q_1 is initial state as well as final state and in second automata q_4 is initial state as well as final state. So these two automata are equivalent.

Example 1.5.7 : Find out whether the following automata are equivalent or not :



[A]



[B]

Fig. P. 1.5.7

Solution

In this example, second condition is satisfied because q_1 is initial and final state in A Automata and q_4 is initial state and final state in B Automata

Check condition I

	Inputs	
State	c	d
(q_1, q_4)	(q_1, q_4) FS FS	(q_2, q_5) IS, IS
(q_2, q_5)	(q_3, q_7) IS, IS	(q_1, q_6) FS, IS

A and B are not equivalent.

DFA and NDFA equivalence

Conversions of NDFA to DFA

Write steps to convert NDFA to DFA.

- Every DFA is an NFA, but not vice versa. But there is an equivalent DF
- for every NFA.
- **Transition function of DFA is,**

$$\delta = Q \times \Sigma \to Q$$

- **Transition function of NFA is,**

$$\delta = Q \times \Sigma \to 2^Q$$

- We can convert every NFA into DFA.

Steps to convert NFA into DFA

1. Construct the transaction table of given NFA machine
2. Scan the next states column in transaction table from initial sta to final state

3. If any of the next state consists more than one state on single input alphabet, then merge them and make a new state .place this new constructed state in DFA transaction table as a present state.
4. The next state of this new constructed state on input alphabet will be the summation of each next state which parts in the NFA transition table.
5. Repeat steps 2 to4 until all states in NFA transition table will be scanned completely.
6. The final transition table must have single next state at single input alphabet.

Example 1.5.8 : Find equivalent DFA for the NFA given by L = [{A, B, C}, (a, b), δ, A, {c}] where δ is given by :

Table P. 1.5.8 : Transition table for NFA

	a	b
A	A, B	C
B	A	B
©C	–	A, B

Solution :

States : A, B, C

Inputs : a, b

Transition function δ : Given in transition table

Initial state : A

Final state : C

State diagram for NFA

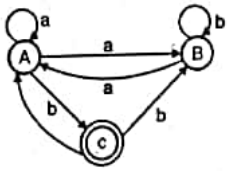


Fig. P. 1.5.8(a)

Convert NFA into DFA

Transition table for DFA

	a	b
→ A	{A, B}	C
AB	AB	BC
(BC)	A	AB
(C)	D	AB
D	0	D

Note

- To find AB states, we must look at transition state diagram of NFA and apply union operation.
- In DFA we cannot leave a state so we have to write new state. In our example we have written 0 is a new state which is called dead state.
- The input that comes in dead state remains there.
- To find **final state of DFA**
  - Check final state of NFA and after that select state from DFA that contains the final state of NFA.
  - In our example, 'C' state D a final state 1 NFA. So select state from DFA which contains 'C' state in DFA. So we get two final state in our example.

- State diagram for DFA

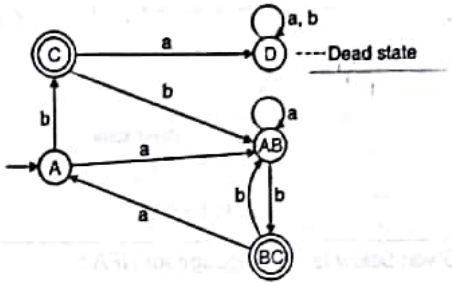


Fig. P. 1.5.8(b)

Example 1.5.9 : L = {Set of all strings over (0, 1) that starts with '0'}

Σ = {0, 1}

Fig. P. 1.5.9(a)

State transition diagram,

States	0	1
A	B	ϕ
B	B	B

Convert the NFA into DFA

Solution :

Transition state

	0	1	
A	B	C	.... C is a dead state or trap state
B	B	B	
C	C	C	

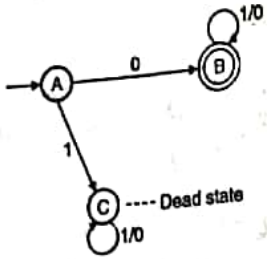


Fig. P. 1.5.9(b)

Example 1.5.10 : Given below is the language for NFA :

L = {Set of all strings over (0, 1) that ends with '01'} construct equivalent DFA.

Solution :

First draw NFA

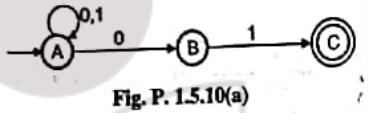


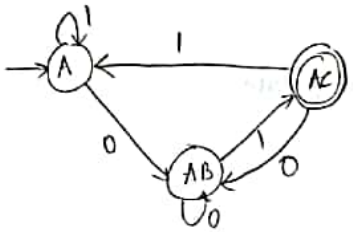
Fig. P. 1.5.10(a)

Transition table

	0	1
→ A	A, B	A
B	ϕ	C
(C)	ϕ	ϕ

Convert NFA into DFA

Transition table for DFA



	0	1
→ A	AB	A
AB	AB	AC
(AC)	AB	A

Fig. P. 1.5.10(b)

Example 1.5.11 : Design an NFA for language that accepts all strings over {0, 1} in which second last symbol is always '1'. Then convert it into its equivalent DFA.
e.g. string 1010 or 110 or 1101010.

Solution :

Design NFA

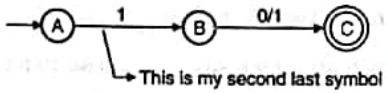


Fig. P. 1.5.11(a)

State transition table for NFA

	0	1
→ A	A	A, B
B	C	C
(C)	ϕ	ϕ

Convert NFA into DFA

Transition table

	0	1
→ A	A	AB
AB	AC	ABC
(AC)	A	AB
(ABC)	AC	ABC

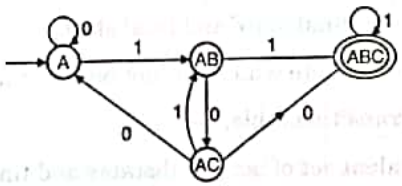


Fig. P. 1.5.11(b)

Minimization of DFA

- How to do minimization of DFA ?
- How to reduce the number of states from DFA ?

– Minimization of DFA means reduce the number of states.

States and equivalence between states

– Consider two states p and q we say that these two states are equivalent.

Condition

$$\delta(p, w) \in f$$

$$\Rightarrow \quad \delta(q, w) \in f$$

Non equivalent :

$\therefore$ If $\delta(p, w) \notin f$

$\Rightarrow \delta(q, w) \notin f$.

If any two states follow this property, then it is called as two states are equivalent.

If length of w = 0, then it is 0 equivalent

$\therefore |w| = 0, \ldots\ldots\ldots 0$ equivalent

$|w| = 1, \ldots\ldots\ldots 1$ equivalent

.

.

.

$|w| = n, \ldots\ldots\ldots n$ equivalent

Steps to find minimization of DFA

1. Identify the non-final state and final state.
2. Try to delete the state which can not be reachable from Initial state.
3. Draw state transition table.
4. Find 0 equivalent set of non-final states and final states.

5. Find 1 equivalence sets :

   Take previous states and check whether they are in same state or not. If they are in same state they are equivalent to each other.
6. Repeat the procedure till we get two same equivalence.
7. Draw a transition diagram or state diagram.

Example 1.5.12 : Construct a minimum DFA equivalent to the DFA

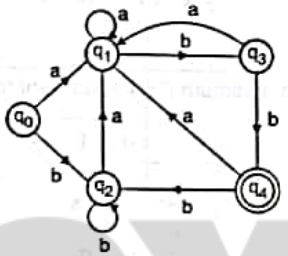


Fig. P. 1.5.12(a)

Solution :

	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
(q_4)	q_1	q_2

		Non final states	Final states
0	Equivalence	$[q_0, q_1, q_2, q_3]$	$[q_4]$
1	Equivalence	$[q_0, q_1, q_2]$ $[q_3]$ $[q_4]$	(q_0, q_1)
2	Equivalence	$[q_0, q_2]$ $[q_1]$ $[q_3]$	$[q_4]$
3	Equivalence	$[q_0, q_2]$ $[q_1]$ $[q_3]$	$[q_4]$

Draw a DFA

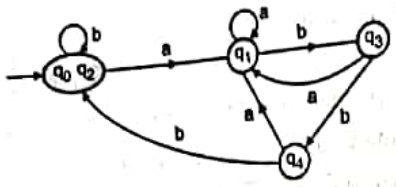


Fig. P. 1.5.12(b)

Minimization of DFA

Example 1.5.13 : Construct a minimum DFA equivalent to the DFA :

	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Solution :

Initial state = q_0

Final state = q_2

Input = {0, 1}

0th Equivalence

$\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\ \{q_2\}$

1st Equivalence

$\{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\ \{q_2\}$

2nd Equivalence

$\{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_2\}$

3rd Equivalence

$\{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}, \{q_2\}$

New transition table

	0	1
→ $\{q_0, q_4\}$	$\{q_1, q_7\}$	$\{q_3, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_0, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
$(\{q_2\})$	$\{q_0, q_4\}$	$\{q_2\}$

Example 1.5.14 : Construct a minimum DFA equivalent to the DFA,

	0	1
→ q_0	q_1	q_5
q_1	q_6	* q_2
* q_2	q_0	* q_2
q_4	q_7	q_5
q_5	* q_2	q_6
q_6	q_6	q_4
q_7	q_6	* q_2

(* Mark indicates final state)

Solution :

0 equivalence : $[q_0\ q_1\ q_4\ q_5\ q_6\ q_7]\ [q_2]$

1 equivalence : $[q_0\ q_4\ q_6]\ [q_1\ q_7]\ [q_5]\ [q_2]$

2 equivalence : $[q_0\ q_4][q_6][q_1q_7][q_5][q_2]$

3 equivalence : $[q_0\ q_4]\ [q_6][q_1\ q_7]\ [q_5][q_2]$

4 equivalence : $[q_0\ q_4][q_6][q_1\ q_7][q_5][q_2]$

Minimization of DFA : When there are more than one final states involved.

Example 1.5.15 : Construct a minimum DFA equivalent to the DFA,

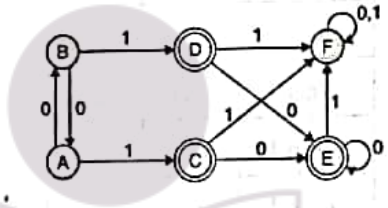


Fig. P. 1.5.15(a)

	0	1
→ A	B	C
B	A	D
(C)	E	F
(D)	E	F
(E)	E	F
F	F	F

Solution :

0 - Equivalence - {A, B, F} {C, D, E}

1 - Equivalence - {A, B} {F} {C, D, E}

2 - Equivalence - {A, B} {F} {C, D, E}

New transition table

	0	1
→ {A, B}	{A, B}	{C, D, E}
{F}	{F}	{F}
{C, D, E}	{C, D, E}	{F}

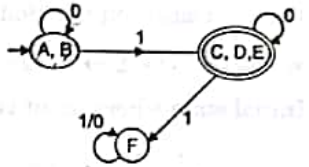


Fig. 1.5.15(b)

B. Finite state Automata with output

There are two types of finite state machine :

(i) Mealy machine
(ii) Moore machine

(i) Mealy machine

☞ **Explain Mealy machine with example.**

The value of the output function in most general case is function of present state and present input.

Lets consider,

y(t) = Output function

q(t) = Present state

x(t) = Input

Then we can say that,

$$y(t) = \delta\ (q(t), x(t))$$

Mealy machine can be described by six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where,

Q = Finite set of states

Σ = Input alphabet

Δ = Output alphabet

δ = Input transition function

where $\delta : Q \times \Sigma \to Q$

λ = Output transition function

where $\lambda : Q \times \Sigma \to \Delta / Q \to \Delta$

q_0 = Initial state where input is ($q_0 \in Q$)

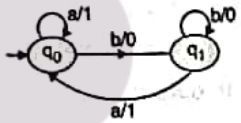


Fig. 1.5.12

$\lambda : Q \times \Sigma \to \Delta$

$(q_0, a) \to 1$

$(q_0, b) \to 0$

$(q_1, b) \to 0$

$(q_1, a) \to 1$

$q_0 \xrightarrow[1]{a} q_0 \xrightarrow[0]{b}$

If we give 'n' bit input then output will be n bit.

Example 1.5.16 : Construct a Mealy machine that takes binary number as input and produces 2's complement of that number as output. Assume the string is read LSB to MSB and end carry is discarded.

Solution :

Σ = {0, 1}

Δ = {0, 1} ... 2's complement of number which is binary

1. First find 1's complement of input

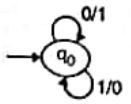


Fig. P. 1.5.16(a)

Check for input 1011 → 1's complement is 0100

$q_0 \xrightarrow[0]{1} q_0 \xrightarrow[1]{0} q_0 \xrightarrow[0]{1} q_0 \xrightarrow[0]{1} q_0$

2. Find 2's complement

∴ MSB LSB

Input → 1 [1] [00]

1's complement ⟶ 0 0 1 1

2's complement (adding 1) ⟶ + 1

0 [1] [0 0]

← LSB

Input → 1 1 1 0 1 [1 0 0]

←

1's complement ⟶ 0 0 0 1 0 0 1 1

\+ 1

2's complement ⟶ 0 0 0 1 0 [1] 0 0

←

Note : Whenever we see first '1' it remains same and after that take 1's complement of each bit.

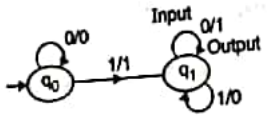


Fig. P. 1.5.16(b)

Check for 1100 → 2's complement for 0100

read from LSB

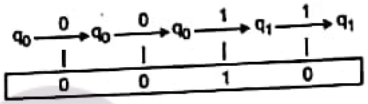


Fig. P. 1.5.16(c)

(ii) Moore machine

Explain Moore machine with example.

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

A moore machine can be described by a 6 tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

Where -

Q - Finite set of states

Σ - Input alphabet

Δ - Output alphabet

δ - Input transition function where $\delta : Q \times \Sigma \rightarrow \theta$

λ - Output transition function where $\lambda : Q \rightarrow \Delta$

q_0 – Initial state where input is $(q_0 \in \theta)$

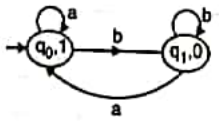


Fig. 1.5.13

In Fig. 1.5.13, a and b are inputs and 1and 0 are outputs, q_0 and q_1 are states.

Output transition function $\lambda : Q \rightarrow \Delta$

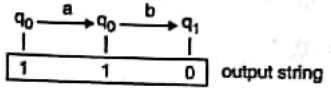


Fig. 1.5.14

If we give 'n' bit input then output will be 'n' bit.

Example 1.5.17 : Construct a Moore machine that takes set of all strings over {a, b} as input and prints '1' as output for every occurrence of 'ab' as a substring.

Solution :

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

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

1. First construct DFA -

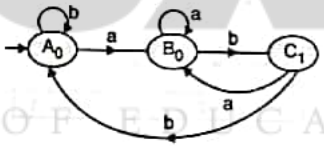


Fig. P. 1.5.17(a)

2. If the input string is 'ab'

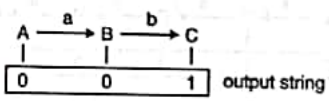


Fig. P. 1.5.17(b)

3. If the input string is 'abab'

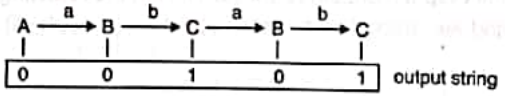


Fig. P. 1.5.17(c)

Example 1.5.18 : Construct a moore machine that takes set of all strings over {a, b} and count number of occurrence of substring 'baa'.

Solution :

$$\sum = \{a, b\}$$

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

(1) First construct DFA

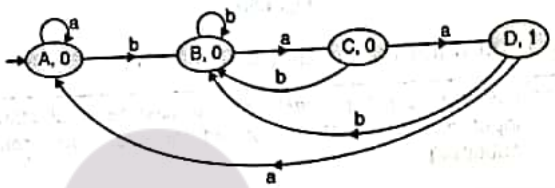


Fig. P. 1.5.18(a)

Note : (1) Search the state where "baa" sub-string is present.

(2) In that state put 1 as a output and remaining outputs are zero.

1. If input is "baa"

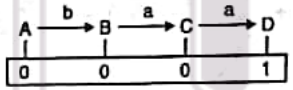


Fig. P. 1.5.18(b)

If input is "baabaa"

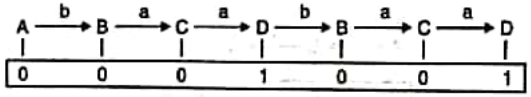


Fig. P. 1.5.18(c)

Example 1.5.19 : For the following moore machine the input alphabet is $\sum$ = {a, b} and the output alphabet is Δ = {0, 1}. Run the following input sequence and find the respective outputs : (i) aabab (ii) abbb (iii) ababb

States	a	b	Outputs
$\rightarrow q_0$	q_1	q_2	0
q_1	q_2	q_3	0
q_2	q_3	q_4	1
q_3	q_4	q_4	0
q_4	q_0	q_0	0

Solution :

(i) If input is "aabab"

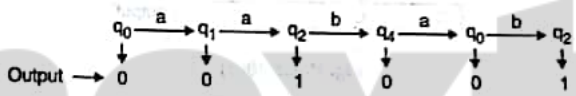


Fig. P. 1.5.19(a)

(ii) If input is "abbb"

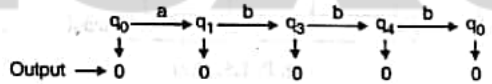


Fig. P. 1.5.19(b)

(iii) If input is "ababb"

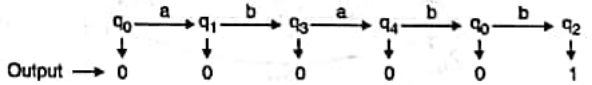


Fig. P. 1.5.19(c)

Example 1.5.20 : Construct a moore machine that takes set of all strings over {0, 1} and produces 'A' as output. If input ends with '01' or produces B as output if input ends with '11' otherwise produce 'C'.

Solution :

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

$$\Delta = \{A, B, C\}$$

If input ends with 01 output $\rightarrow$ A

If input ends with 11 output → B

Otherwise output → C

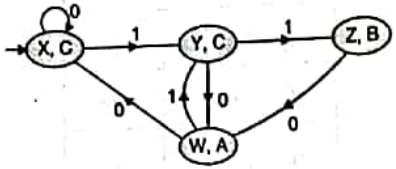


Fig. P. 1.5.20(a)

Check for 111

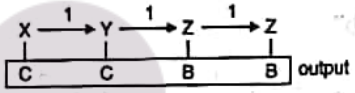


Fig. P. 1.5.20(b)

Check for 110

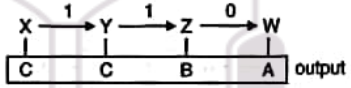


Fig. P. 1.5.20(c)

Example 1.5.21 : What is the output produced by machine shown in Fig. P. 1.5.20(a) ?

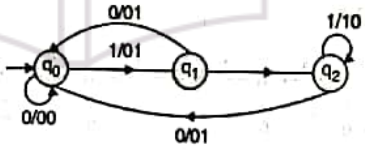


Fig. P. 1.5.21(a)

Solution :

Steps

- Start searching string from LSB.
- Whenever we find '1' in string let it remain as it is.
- From next bit onwards take 1's complement of each bit.

1. State table of Moore machine

Present state	Next State		Output
	a = 0	a = 1	
→ 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

Draw state diagram

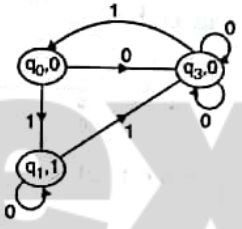


Fig. P. 1.5.21(b)

For input string 0 1 1 1 the transition of states is given by,

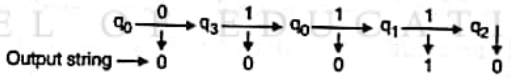


Fig. P. 1.5.21(c)

2. State table of Mealy machine

Present state	Next state			
	a = 0		a = 1	
	State	Output	State	Output
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

Draw a state diagram

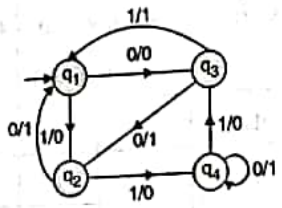


Fig. P. 1.5.21(d)

For input string 0 0 1 1

The transition of states is given by,

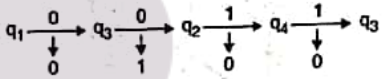


Fig. P. 1.5.21(e)

Output string is 0 1 0 0

Conversion of Mealy machine into Moore machine

☞ **Convert Mealy machine into Moore machine.**

Input - Mealy machine

Output - Moore machine

Step 1 : Calculate number of different outputs for each state that are in state table.

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 where n = 0, 1, 2, ...

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

For example State table of mealy machine

Present	Next state			
	a = 0		a = 1	
	State	Output	State	Output
→ q_1	q_3	0	q_2	0
q_2	q_1	1	q_4	0
q_3	q_2	1	q_1	1
q_4	q_4	1	q_3	0

In this example,

q_1 is associated with output 1, q_0 is associated with two different outputs 0 and 1

Similarly q_3 is associated with 0 and q_4 is associated with two different output 0 and 1.

Split q_2 into q_{20} and q_{21}

Similarly q_4 into q_{40} and q_{41}

Reconstruct state table for new states

Present	Next state			
	State	Output	State	Output
	input a = 0		input a = 1	
→ q_1	q_3	0	q_{20}	0
q_{20}	q_1	1	q_{40}	0
q_{21}	q_1	1	q_{40}	0
q_3	q_{21}	1	q_1	1
q_{40}	q_{41}	1	q_3	0
q_{41}	q_{41}	1	q_3	0

Step 4. Pair of states and outputs in next column can be rearranged

Present	Next state		Output
	a = 0	a = 1	
→ q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

- In this state table, we observe that initial state q_1 is associated with output 1, it means that if machine starts at state q_1 we get output 1.
- Thus this Moore machine accepts a zero length sequence which is not accepted by mealy machine.
- To overcome this problem, we have to add new starting state q_0, whose state transition are similar to q_1 but output is 0.

Step 5 : State table for Moore machine

Present State	Next state		Output
	a = 0	a = 1	
→ q_0	q_3	q_{20}	0
q_1	q_3	q_{20}	1
q_{20}	q_1	q_{40}	0
q_{21}	q_1	q_{40}	1
q_3	q_{21}	q_1	0
q_{40}	q_{41}	q_3	0
q_{41}	q_{41}	q_3	1

Conversion of Moore machine into Mealy machine.

How to Convert Moore machine into Mealy machine ? Explain it with example.

Input = Moore machine

Output = Mealy machine

Step 1 : Take black mealy machine transition table format.

Step 2 : Copy all moore machine transition state into this table format.

Step 3 : Check present states and their corresponding outputs in moore machine state table, if for a state Q_i output is m, copy it into output columns of mealy machines state table wherever Q_i appears in next state.

State table for moore machine

Present State	Next state		Output
	a = 0	a = 1	
→ 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

Step 4 : Draw a blank mealy machine transition table format and copy all moore machine transition state into table.

Present state	Next state			
	a = 0		a = 1	
	State	Output	State	Output
→ q_0	q_3		q_1	
q_1	q_1		q_2	
q_2	q_2		q_3	
q_3	q_3		q_0	

Step 5 : Transition

Draw table for mealy machine

Present state	Next state			
	a = 0		a = 1	
	State	Output	State	Output
→ q_0	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Difference between Mealy machine and Moore machine

☞ **Write the difference between Mealy Machine and Moore Machine.**

Sr. No.	Mealy Machine	Moore Machine
1.	Output depends upon present state as well as present input.	Output depends only on present state.
2.	It has less states than moore machine.	However, it has more states than mealy machine.
3.	These machines react faster to inputs.	In these machines more logic is needed to decode the output. Since it has more circuit delays.

□□□

CHAPTER 2

Formal Languages

University Prescribed Syllabus

Formal Languages : Defining Grammar, Derivations, Languges generated by Grammar, Chomsky Classification of Grammar and Languages, Recursive Enumerable Sets, Operations on Languages, Languages and Automata.

2.1 Introduction

- The formal languages in automata theory started in 1959, with formal definition given by Roam Chomsky. He tried to find out what is a grammar ? What is a mathematical model of grammar ?
- Let us for take some sentences in English and see how the sentences can be parsed.

For Example,

"The man ate the fruit".

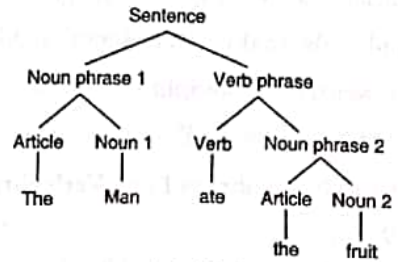


Fig. 2.1.1