UNIT 1

Syllabus: **Introduction to Finite Automata:** Structural Representations, Automata and Complexity, the Central Concepts of Automata Theory – Alphabets, Strings, Languages, and Problems. **Nondeterministic Finite Automata:** Formal Definition, an application, Text Search, Finite Automata with Epsilon-Transitions. **Deterministic Finite Automata:** Definition of DFA, How A DFA Process Strings, The language of DFA, Conversion of NFA with €-transitions to NFA without €-transitions. Conversion of NFA to DFA, Moore and Melay machines

---

Introduction toFinite Automata

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

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

Formal definition of a Finite Automaton

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

- **Q** is a finite set of states.
- **Σ** is a finite set of symbols, called the alphabet of the automaton.
- **δ** is the transition function.
- **q0** is the initial state from where any input is processed (q0 ∈ Q).
- **F** is a set of final state/states of Q (F ⊆ Q).

Structural representation of Finite Automata

- An automaton with a finite number of states is called as Finite Automaton or Finite State Machine.
- The block diagram of finite automaton consists of falling three major components.
  - ➢ Input tape
  - ➢ Reading Head
  - ➢ Finite Control

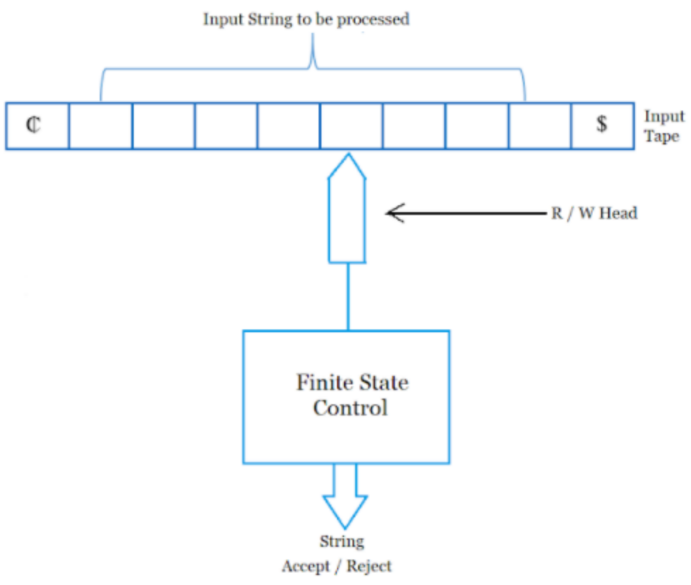


Fig: Block Diagram of Finite Automata

Input Tape:

- ➢ The input tape is divided into squares, each square containd a single symbol from input alphabet ∑.
- ➢ The end squares of each tape contain end markers at left end and right end.
- ➢ Absence of end markers indicates that tape is infinite length.
- ➢ The left to right sequence of symbols between end markers is the input string to be processed.

Reading Head:

- ➢ The Reading head examines only one square at a time and can move one square towards left till end o the string.

Finite State Control:

- The FSC is responsible for controlling total functioning of finite automate machine.
- It will decide which input symbol is read next and where to move either to the left or right.
- The input to the finite control will be usually
    - Input symbol from input tape
    - Present state of machine
- The output may be
    - ➢ Movement of Reading head along the tape to the next square or to null move
    - ➢ The next state/new state of FSM

The Central Concepts of Automata Theory

Alphabets

Definition: An alphabet is any finite set of symbols. It is denoted by Σ

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

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

$$\Sigma = \{a, b, c, \&, z\}$$

$$\Sigma = \{\#, \nabla, \spadesuit, \beta\}$$

Strings

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

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

It is not the case that a string over some alphabet should contain all the symbols from the alphabet. Fo example, the string cc over the alphabet { a, b, c } does not contain the symbols a and b. Hence, it is true that a string over an alphabet is also a string over any superset of that alphabet.

Length of a String

- Definition : It is the number of symbols present in a string. (Denoted by |S|).
- Examples:
    - If S=‘cabcad’, |S|= 6
    - If |S|= 0, it is called an empty string (Denoted by λ or ε)

Kleene Star

- Definition: The Kleene star, Σ^*, is a unary operator on a set of symbols or strings, Σ, that gives the infinite set of all possible strings of all possible lengths over Σ including λ.

- Representation: $\Sigma^* = \Sigma 0 \cup \Sigma 1 \cup \Sigma 2 \cup \ldots\ldots$ where Σp is the set of all possible strings of length p.

- Example: If $\Sigma = \{a, b\}$, $\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, \ldots \ldots\ldots\ldots\}$

Positive Closure

- Definition: The set Σ^+ is the infinite set of all possible strings of all possible lengths over Σ excluding λ.

- Representation: $\Sigma^+ = \Sigma 1 \cup \Sigma 2 \cup \Sigma 3 \cup \ldots\ldots$
  $\Sigma^+ = \Sigma^* - \{ \lambda \}$

- Example: If $\Sigma = \{ a, b \}$, $\Sigma^+ = \{ a, b, aa, ab, ba, bb, \ldots \ldots\ldots\ldots \}$

Language

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

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

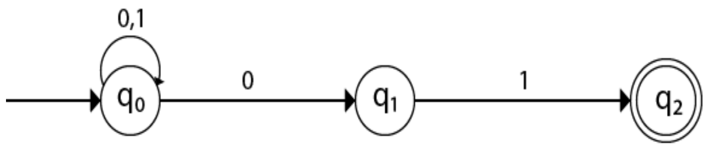
Problems

- Construct a Language consisting of a set of strings over {a, b} in which each string of the language starts with ‘01’
  L = {01,011,010,…….}

Graphical Representation of a FA: (same as NDFA, NFA, ε-NFA)

Any Finite Automata is represented by directed graphs called state transition diagram or transition diagram. Which includes the following

- The vertices represent the states.
- The arcs labeled with an input alphabet shows the transitions i.e $\delta(q0, 1) = q0$.
- In the initial state , there is an arrow with no source.
- The final states or accepting states are indicated by double circles.



Extended transition function:

➢ We define the extended transition function ˆδ.

➢ It takes a state q and an input string w to the resulting state.

➢ The definition proceeds by induction over the length of w.

➢ We define ˆδ (q, x) by induction ˆδ : $Q \times \Sigma^* \rightarrow Q$

- BASIS $\hat{\delta}(q, \varepsilon) = q$ for $|x| = 0$
- INDUCTION suppose $x = ay$ (y is a string, a is a symbol)

  $\hat{\delta}(q, ay) = \hat{\delta}(\delta(q, a), y)$
- Notice that if $x = a$ we have $\hat{\delta}(q, a) = \delta(q, a)$ since $a = a\varepsilon$ and

  $\hat{\delta}(\delta(q, a), \varepsilon) = \delta(q, a)$
- $\delta : Q \times \Sigma^* \rightarrow Q$ We write q.x instead of $\hat{\delta}(q, x)$ We can now define mathematically the language accepted by a given automaton (Q, Σ, δ, q0, F)

  $L = \{ x \in \Sigma * \mid q\,0.x \in F \}$

Finite Automaton can be classified into two types:

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

Non-deterministic Finite Automaton

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, Σ, δ, q0, 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)
- **q0** is the initial state from where any input is processed ($q0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

The language of an NFA:

A= (Q, Σ, δ, q0, F)

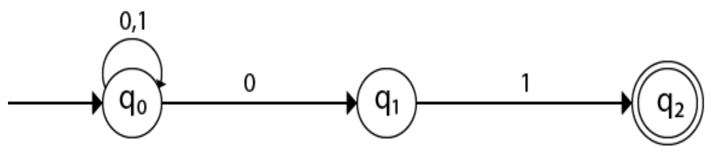
$L(A) = \{w / \hat{\delta}(q_0 , w) \Pi F \neq \Phi\}$

L(A) is the Language accepted by NFA A.

Problem 1: Construct a NFA accepting a set of strings over {a, b} in which each string of the language ends with ‘01’

Solution: The desired language is L1 = {01, 001, 101, 1101 ....}

The state transition diagram of the desired language is



The transition table for the above NFA is

δ	0	1
->q_0	{q_0, q_1}	q_0
q_1	Φ	q_2
*q_2	Φ	Φ

The 5-tuple representation for the desired automata is

M = {{q_0, q_1, q_2},{0, 1}, **δ** , {q_0} , {q_2}}

String acceptance :

Take any string from the language L , let w=001

$$\begin{tikzcd}
\hat{\delta}(q_0,001) = q0,0 \arrow[r] \arrow[dr] & q0,0 \arrow[r] \arrow[dr] & q0,1 \arrow[dr] & \\
& q1,0 \arrow[dr] & q1,1 \arrow[dr] & q0 \\
& & {} & q2
\end{tikzcd}$$

$\hat{\delta}$ (q_0, 001) = {q0,q2}

So the above string w=001 is accepted by the automata. Since { q_0, q_2 }∩{q_2}(i.e Final state F) ≠Φ

An Application: Text Search

- **Finding Strings in Text**

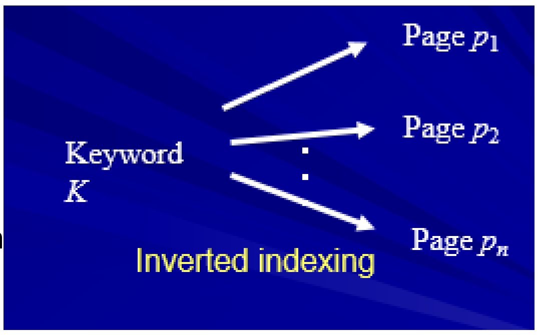
– Searching Google for a set of words is equivalent to just finding strings in documents

– Techniques

  - Using inverted indexes
  - Using finite automata

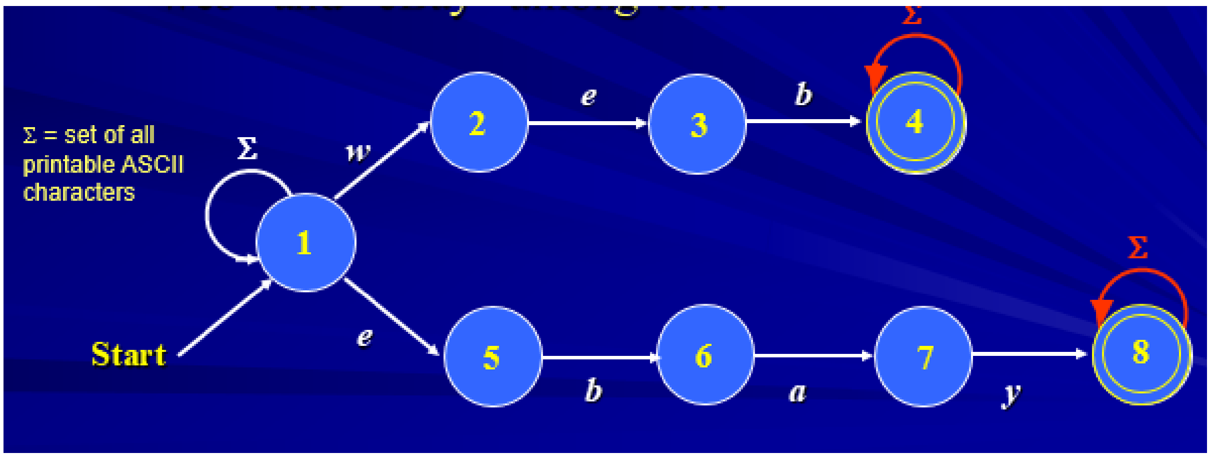
– Applications unsuitable to use inverted in

  - Document repository change rapid
  - Documents to be searched cannot be catalogued.



NFA's for Text Search

– use an NFA to search two keywords "web" and "eBay" among text



Non-Deterministic Finite Automata with ε transitions (ε-NFA)

• A Non-Deterministic Finite Automata with ε transitions is a 5-tuple (Q, Σ, qo, δ, F)

where – Q is a finite set (of states)

– Σ is a finite alphabet of symbols

– qo ∈ Q is the start state

– F ⊆ Q is the set of accepting states

– δ is a function from Q x (Σ ∪ {ε}) -> 2Q (transition function)

- We extend the class of NFAs by allowing instantaneous (ε) transitions:

  1. The automaton may be allowed to change its state without reading the input symbol.
  2. In diagrams, such transitions are depicted by labeling the appropriate arcs with ε.
  3. Note that this does not mean that ε has become an input symbol. On the contrary, we assume that the symbol ε does not belong to any alphabet.

The language of an NFA with epsilon:

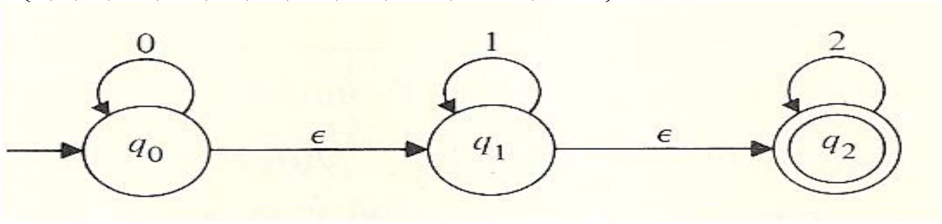
A= (Q, Σ, δ, q0, F)

L(A) ={w/^ δ(q_0 , w) Π F ≠ Φ}

L(A) is the Language accepted by NFA with epsilon A.

Problem: Construct a NFA with epsilon, accepting a Language consists a set of strings with any number of 0's followed by any number of 1's followed by any number of 2's over {0, 1, 2}.

L = {0,1, 2, 00, 11,22,01, 02, 12, 012, 0012,……}



Epsilon (∈) – closure :

- Epsilon closure for a given state X is a set of states which can be reached from the states X with only (null) or ε moves including the state X itself.

- In other words, ε- closure for a state can be obtained by union operation of the ε-closure of the states which can be reached from X with a single ε move in recursive manner.
- ∈ – closures for the above automata are
  - ∈ – closure(q0) = {q0,q1,q2}
  - ∈ – closure(q1) = { q1,q2}
  - ∈ – closure(q2) = {q2}

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, Σ, δ, q0, 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 δ: $Q \times \Sigma \rightarrow Q$
- **q0** is the initial state from where any input is processed ($q0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

The language of an DFA:

A= (Q, Σ, δ, q0, F)

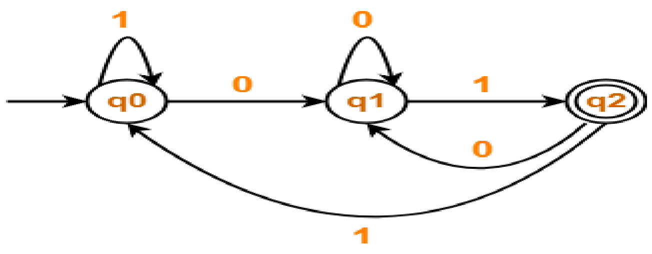
L(A) ={w/ $\hat{\delta}(q_0, w)$ Π F ≠ Φ}

L(A) is the Language accepted by DFA A.

Problem : Construct a DFA accepting a set of strings over {0, 1} in which each string of the language ends with ‘01’

Solution: The desired language is L1 = {01, 001, 101, 1101 ....}

The state transition diagram of the desired language is



The transition table for the above automata is

δ	0	1
->q_0	q_1	q_0
q_1	q_1	q_2
*q_2	q_1	q_0

The 5-tuple representation for the desired automata is

$M = \{\{q_0, q_1, q_2\},\{0, 1\}, \boldsymbol{\delta}, \{q_0\}, \{q_2\}\}$

String acceptance :

Take any string from the language L , let w=001

$\hat{}\delta(q_0,001) = q0,0 \longrightarrow q1,0 \longrightarrow q1,1 \longrightarrow q2$

$\hat{}\delta(q_0,001) = \{q2\}$

So the above string w=001 is accepted by the automata. Since $\{q_2\} \cap \{q_2\} \neq \Phi$

Processing of Strings:

A string is accepted by a DFA/NDFA iff the DFA/NDFA starting at the initial state ends in an accepting state (any of the final states) after reading the entire string.

A string S is accepted by a DFA/NDFA (Q, Σ, δ, q0, F), iff $\hat{}\delta(q0, S) \in F$

The language L accepted by DFA/NDFA is

$\{S \mid S \in \Sigma^* \text{ and } \hat{}\delta(q0, S) \in F\}$

A string S′ is not accepted by a DFA/NDFA (Q, Σ, δ, q0, F), iff $\delta^*(q0, S) \notin F$

The language L′ not accepted by DFA/NDFA (Complement of accepted language L) is
$\{S \mid S \in \Sigma^* \text{ and } \hat{}\delta(q0, S) \notin F\}$

Conversion of NFA with €-transitions to NFA without €-transitions

Problem Statement

Let X = (Qx, Σ, δx, q0, Fx) be an €-NFA which accepts the language L(X). We have to design an equivalent NDFA Y = (Qy, Σ, δy, q0, Fy) such that L(Y) = L(X). The following procedure converts the NFA with €-transitions to NFA without €-transitions

Algorithm

Input:	An NFA with €-transitions
Output:	An equivalent NFA without €-transitions

Step 1: Find out all the ε transitions from each state from Q. That will be called as ε-closure{q1} where qi ∈ Q.

Step 2: Then δ' transitions can be obtained. The δ' transitions mean a ε-closure on δ moves.

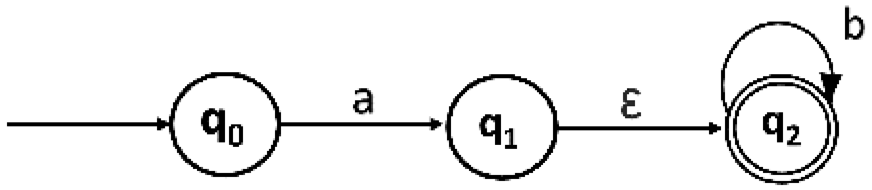
$\delta'(q, a) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q),a))$

step 3: Repeat Step-2 for each input symbol and each state of given €-NFA.

Step 4: Find the final states F for the NFA (F={q}, where ε-closure(q) Π Final states of €-NFA)

Step 5: Using the resultant states, the transition table and transition diagram for equivalent NFA without ε can be built.

Problem: Convert the following NFA with ε to NFA without ε.



Solution:

Step 1 : We will first obtain ε-closures of q0, q1 and q2 as follows:

ε-closure(q0) = {q0}

ε-closure(q1) = {q1, q2}

ε-closure(q2) = {q2}

step2: Now the δ' transition on each input symbol is obtained as:

δ'(q0, a) = ε-closure(δ(δ^(q0, ε),a))

= ε-closure(δ(ε-closure(q0),a))
= ε-closure(δ(q0, a))
= ε-closure(q1)
= {q1, q2}

δ'(q0, b) = ε-closure(δ(δ^(q0, ε),b))
= ε-closure(δ(ε-closure(q0),b))
= ε-closure(δ(q0, b))
= Φ

Now the δ' transition on q1 is obtained as:

δ'(q1, a) = ε-closure(δ(δ^(q1, ε),a))

= ε-closure(δ(ε-closure(q1),a))

= ε-closure(δ(q1, q2), a)

= ε-closure(δ(q1, a) ∪ δ(q2, a))

= ε-closure(Φ ∪ Φ)

= Φ

δ'(q1, b) = ε-closure(δ(δ^(q1, ε),b))

= ε-closure(δ(ε-closure(q1),b))

= ε-closure(δ(q1, q2), b)

= ε-closure(δ(q1, b) ∪ δ(q2, b))

= ε-closure(Φ ∪ q2)

= {q2}

The δ' transition on q2 is obtained as:

δ'(q2, a) = ε-closure(δ(δ^(q2, ε),a))

= ε-closure(δ(ε-closure(q2),a))

= ε-closure(δ(q2, a))

= ε-closure(Φ)

= Φ

δ'(q2, b) = ε-closure(δ(δ^(q2, ε),b))

= ε-closure(δ(ε-closure(q2),b))

= ε-closure(δ(q2, b))

= ε-closure(q2)

= {q2}

Step3: Now we will summarize all the computed δ' transitions:

δ'(q0, a) = {q0, q1}

δ'(q0, b) = Φ

δ'(q1, a) = Φ

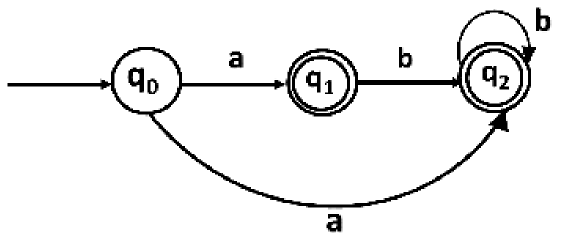
δ'(q1, b) = {q2}

δ'(q2, a) = Φ

δ'(q2, b) = {q2}

Step 4: State q1 and q2 become the final state as ε-closure of q1 and q2 contain the final state q2.

The NFA can be shown by the following transition diagram:



The transition table can be:

States	a	b
→q0	{q1, q2}	Φ
*q1	Φ	{q2}
*q2	Φ	{q2}

The 5- tuple representation of equivalent NFA is

M={{ q0,q1, q2,},{a,b}, δ, {q0},{q1, q2}}

NDFA to DFA Conversion

Problem Statement

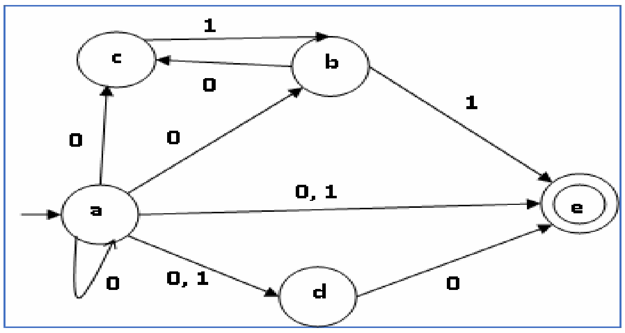
Let X = (Qx, Σ, δx, q0, Fx) be an NDFA which accepts the language L(X). We have to design an equivalent DFA Y = (Qy, Σ, δy, q0, Fy) such that L(Y) = L(X). The following procedure converts the NDFA to its equivalent DFA:

Algorithm

Input:	An NDFA
Output:	An equivalent DFA
Step 1	Create state table from the given NDFA.
Step 2	Create a blank state table under possible input alphabets for the equivalent DFA.
Step 3	Mark the start state of the DFA by q0 (Same as the NDFA).
Step 4	Find out the combination of States {Q0, Q1,... , Qn} for each possible input alphabet.
Step 5	Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.
Step 6	The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example

Let us consider the NDFA shown in the figure below.



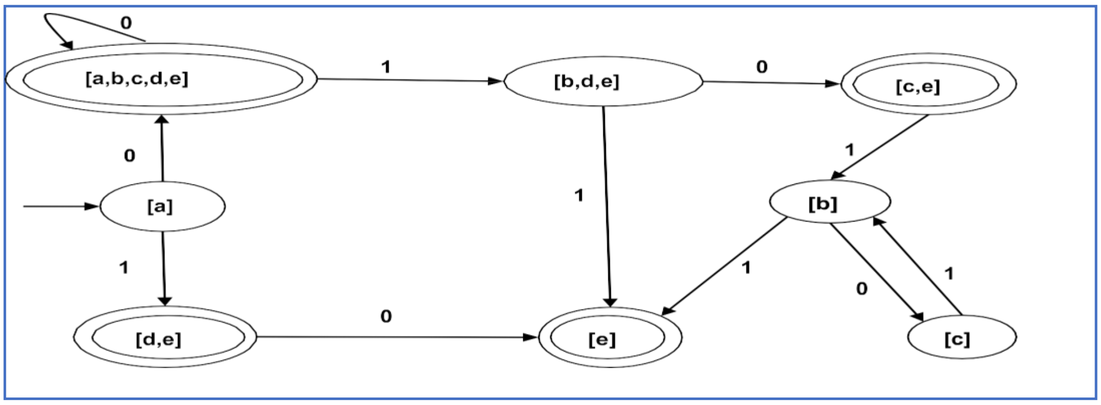
q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	∅	{b}
d	{e}	∅
e	∅	∅

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

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

State table of DFA equivalent to NDFA

The state diagram of the DFA is as follows:



State diagram of equivalent DFA

DFA vs NDFA

The following table lists the differences between DFA and NDFA.

DFA	NDFA
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 can be to multiple next states for each input symbol. Hence it is called *non-deterministic*.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
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.

Moore and Melay Machines

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

- Melay Machine
- Moore machine

Melay Machine

A Melay Machine is an FSM whose output depends on the present state as well as the present input(transition).

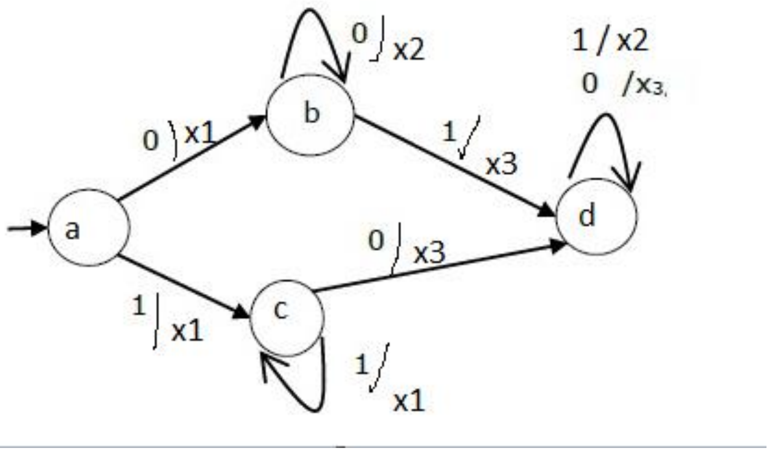
It can be described by a 6 tuple (Q, ∑, O, δ, X, q0) where −

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

The state table of a Melay Machine is shown below –

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

The state diagram of the above Melay Machine is:



Moore Machine

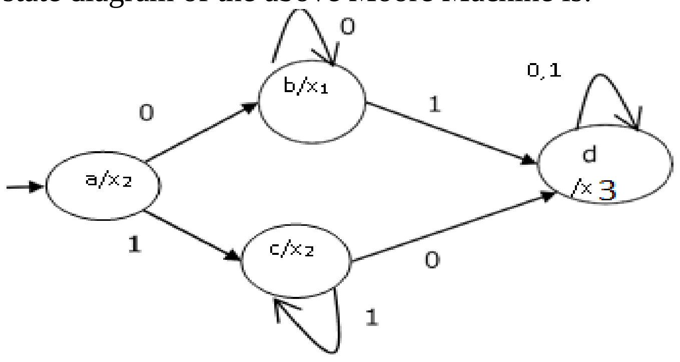
Moore machine is an FSM whose outputs depend on only the present state. A Moore machine can be described by a 6 tuple (Q, Σ, O, δ, X, q0) where:

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

The state table of a Moore Machine is shown below –

Present State	Next State		Output
	Input = 0	Input = 1	
$\rightarrow$ a	b	c	x_2
b	b	d	x_1
c	c	d	x_2
d	d	d	x_3

The state diagram of the above Moore Machine is:



Melay Machine vs. Moore Machine

The following table highlights the points that differentiate a Melay Machine from a Moore Machine.

Melay Machine	Moore Machine
Output depends both upon present state and present input.	Output depends only upon the present state.
Generally, it has fewer states than Moore Machine.	Generally, it has more states than Melay Machine.
Output changes at the clock edges.	Input change can cause change in output change as soon as logic is done.
Melay machines react faster to inputs	In Moore machines, more logic is needed to decode the outputs since it has more circuit delays.

Moore Machine to Melay Machine

Algorithm 4

Input:	Moore Machine
Output:	Melay Machine
Step 1	Take a blank Melay 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 Qi output is m, copy it into the output columns of the Mealy Machine state table wherever Qi appears in the next state.

Example

Let us consider the following Moore machine:

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

State table of a Moore Machine

Now we apply Algorithm 4 to convert it to Melay Machine.

Step 1 & 2:

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

The partial state table after steps 1 and 2

Step 3:

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

State table of an equivalent Melay Machine

Melay Machine to Moore Machine

Algorithm 5:

Input:	Melay Machine
Output:	Moore Machine
Step 1	Calculate the number of different outputs for each state (Qi) that are available in the state table of the Melay machine.
Step 2	If all the outputs of Qi are same, copy state Qi. If it has n distinct outputs, break Qi into n states as Qin where n = 0, 1,2.......
Step 3	If the output of the initial state is 1, insert a new initial state at the beginning which gives 0 output.

Example

Let us consider the following Melay Machine:

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

State table of a Melay Machine

Here, states ‘a’ and ‘d’ give only 1 and 0 outputs respectively, so we retain states ‘a’ and ‘d’. But states ‘b’ and ‘c’ produce different outputs (1 and 0). So, we divide b into b0, b1 and c into c0, c1.

Present State	**Next State**		**Output**
	a = 0	**a = 1**	
-> a	d	b1	1
b0	a	d	0
b1	a	d	1
c0	c1	c0	0
c1	c1	c0	1
d	b0	a	0

State table of equivalent Moore Machine