Unit-II

Finite Automata with E-Transition

<u>NFA with epsilon (ε) transitions</u>

NFA's with ε −Transitions

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

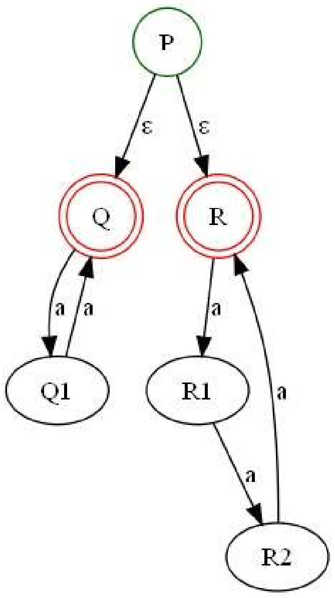
1. The automaton may be allowed to change its state without reading the input symbol.

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.

Example

- { a^n | n is even or divisible by 3 }



Definition

• A ε-NFA is a quintuple A=(Q,Σ,δ,q0,F)

Where

– Q is a set of states

– Σ is the alphabet of input symbols

– q0∈Q is the initial state

– F ⊆ Q is the set of final states

– δ: Q × Σε □→ P(Q) is the transition function

• Note ε is never a member of Σ • Σε is defined to be (Σ ∪ ε)

ε-NFA

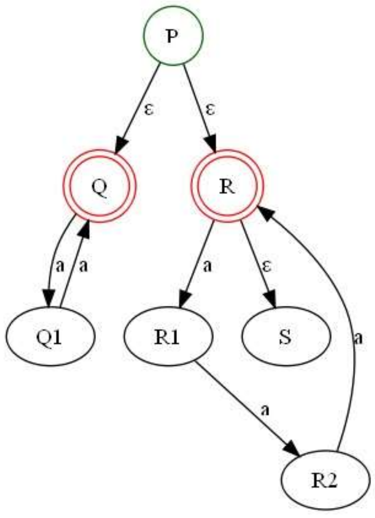
• ε -NFAs add a convenient feature but (in a sense) they bring us nothing new: they do not extend the class of languages that can be represented. Both NFAs and ε-NFAs recognize exactly the same languages.

• ε-transitions are a convenient feature: try to design an NFA for the even or divisible by 3 language that does not use them! – Hint, you need to use something like the product construction from union-closure of DFAs

ε-Closure

• ε-closure of a state

• The ε-closure of the state q, denoted ECLOSE(q), is the set that contains q, together with all states that can be reached starting at q by following only ε-transitions.



• In the above example:

• ECLOSE(P) ={P,Q,R,S}

• ECLOSE(R)={R,S}

• ECLOSE(x)={x} for the remaining 5 states {Q,Q1,R1,R2,R2}

Elimination of ε-Transitions

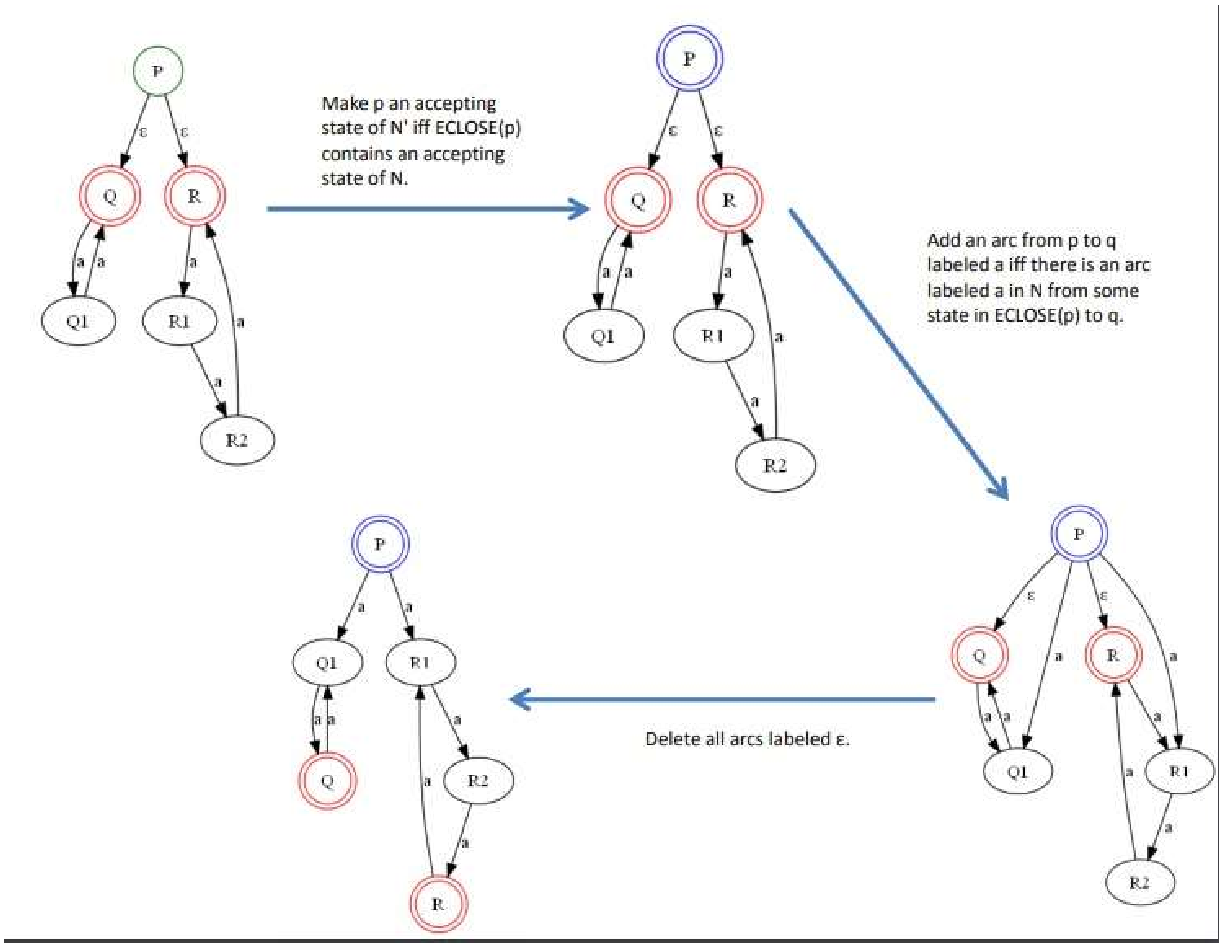
• Given an ε-NFA N, this construction produces an NFA N' such that L(N')=L(N).

• The construction of N' begins with N as input, and takes 3 steps:

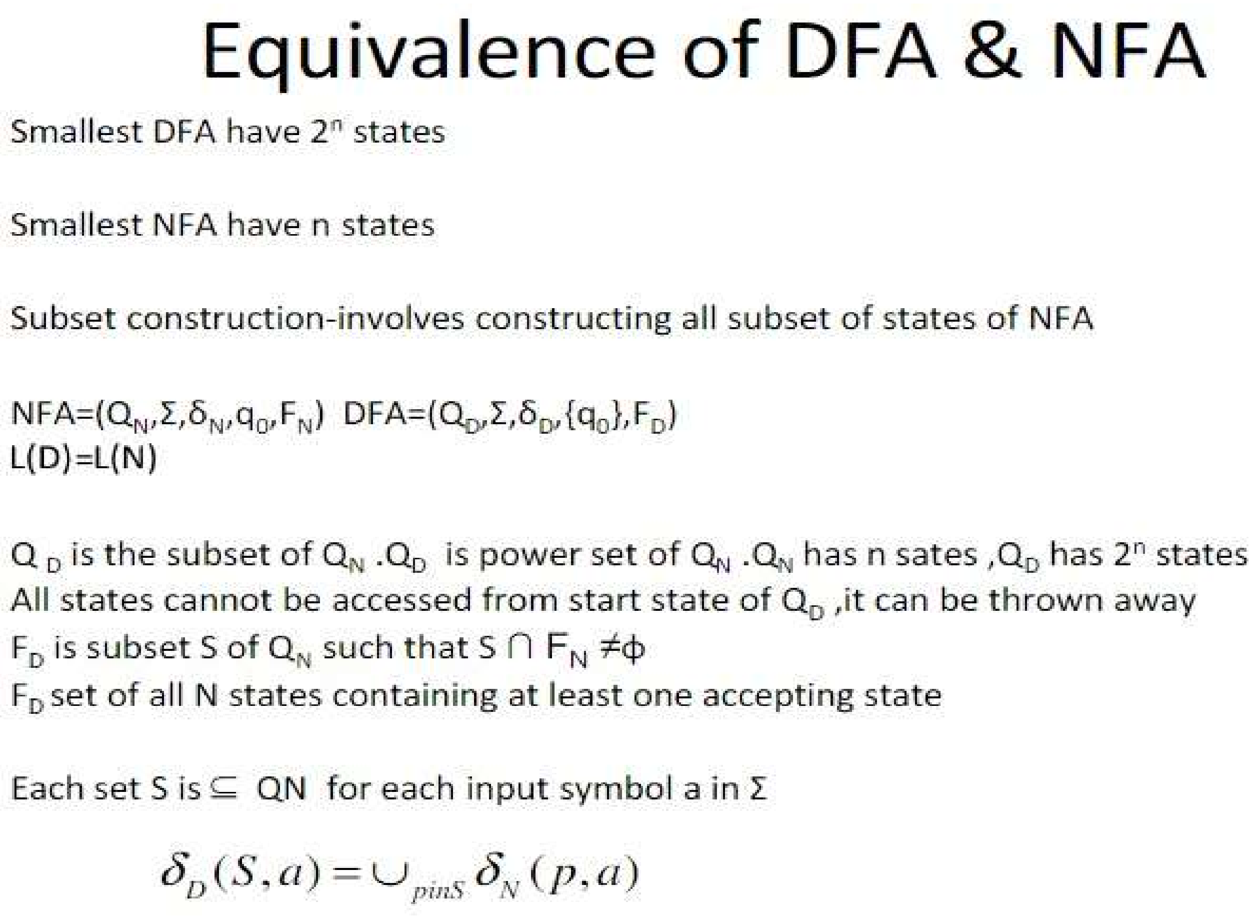
1. Make p an accepting state of N' iff ECLOSE(p) contains an accepting state of N.

2. Add an arc from p to q labeled a iff there is an arc labeled a in N from some state in ECLOSE(p) to q.

3. Delete all arcs labeled ε.


Make p an accepting state of N' iff ECLOSE(p) contains an accepting state of N.
Add an arc from p to q labeled a iff there is an arc labeled a in N from some state in ECLOSE(p) to q.
Delete all arcs labeled ε.
P
Q
R
Q1
R1
R2
ε
a

Equivalence of DFA and NFA



Equivalence of DFA & NFA

Smallest DFA have 2^n states

Smallest NFA have n states

Subset construction-involves constructing all subset of states of NFA

NFA=$(Q_N,\Sigma,\delta_N,q_0,F_N)$ DFA=$(Q_D,\Sigma,\delta_D,\{q_0\},F_D)$
L(D)=L(N)

Q_D is the subset of Q_N .Q_D is power set of Q_N .Q_N has n sates ,Q_D has 2^n states
All states cannot be accessed from start state of Q_D ,it can be thrown away
F_D is subset S of Q_N such that $S \cap F_N \neq \phi$
F_D set of all N states containing at least one accepting state

Each set S is $\subseteq$ QN for each input symbol a in Σ

$$\delta_D(S,a) = \cup_{pinS}\, \delta_N(p,a)$$

Conversion of NFA into DFA

Problem Statement

Let **X = (Q_x, $\sum$, δ_x, q_0, F_x)** be an NDFA which accepts the language L(X). We have to design an equivalent DFA **Y = (Q_y, $\sum$, δ_y, q_0, F_y)** 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).

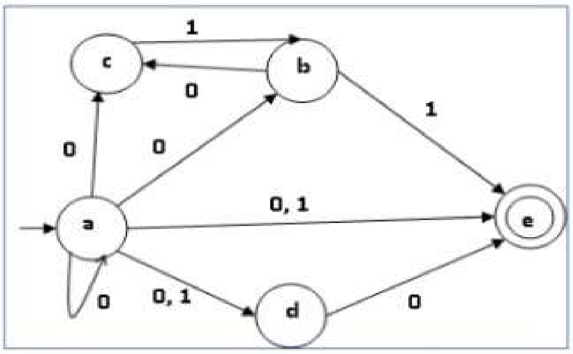
Step 4 − Find out the combination of States $\{Q_0, Q_1, \ldots, Q_n\}$ for each possible input alphabet.

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.

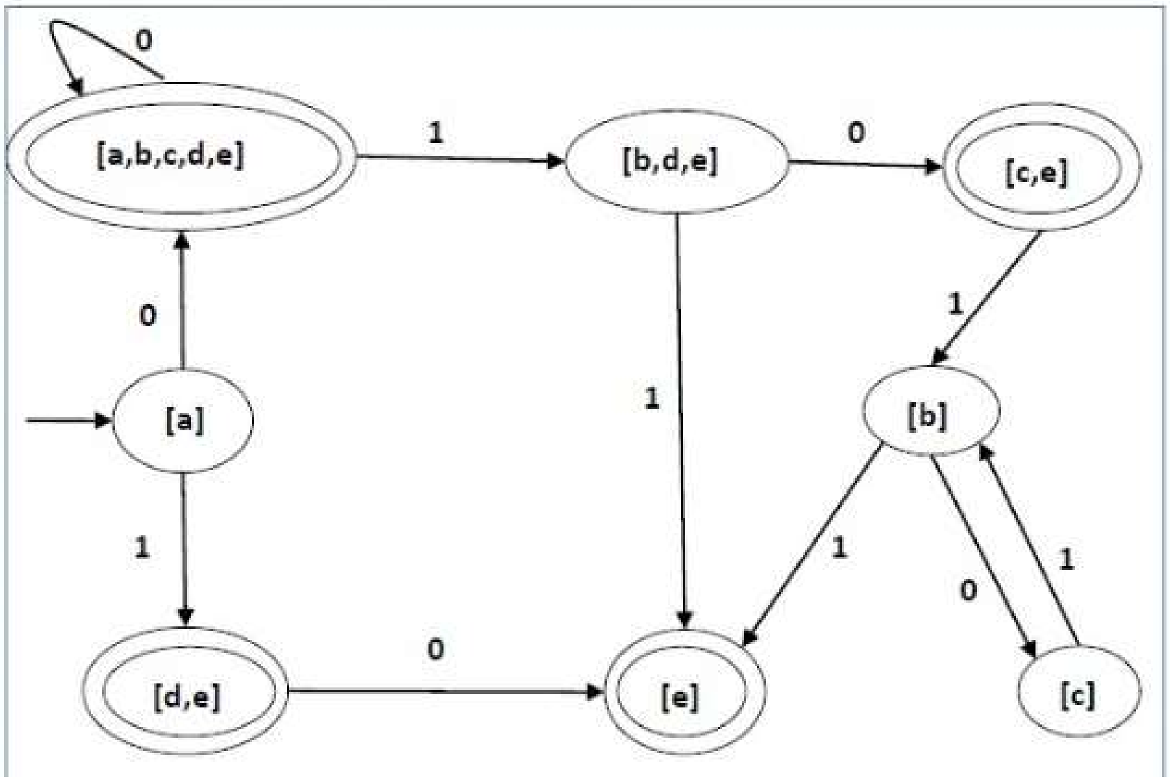


q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	∅	{b}
d	{e}	∅
e	∅	∅

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]

The state diagram of the DFA is as follows −



ClassWork:

Minimization of Finite Automata

DFA Minimization using Equivalence Theorem

If X and Y are two states in a DFA, we can combine these two states into {X, Y} if they are not distinguishable. Two states are distinguishable, if there is at least one string S, such that one of δ (X, S) and δ (Y, S) is accepting and another is not accepting. Hence, a DFA is minimal if and only if all the states are distinguishable.

Algorithm 3

Step 1 − All the states **Q** are divided in two partitions − **final states** and **non-final states** and are denoted by $\mathbf{P_0}$. All the states in a partition are 0thequivalent. Take a counter **k** and initialize it with 0.

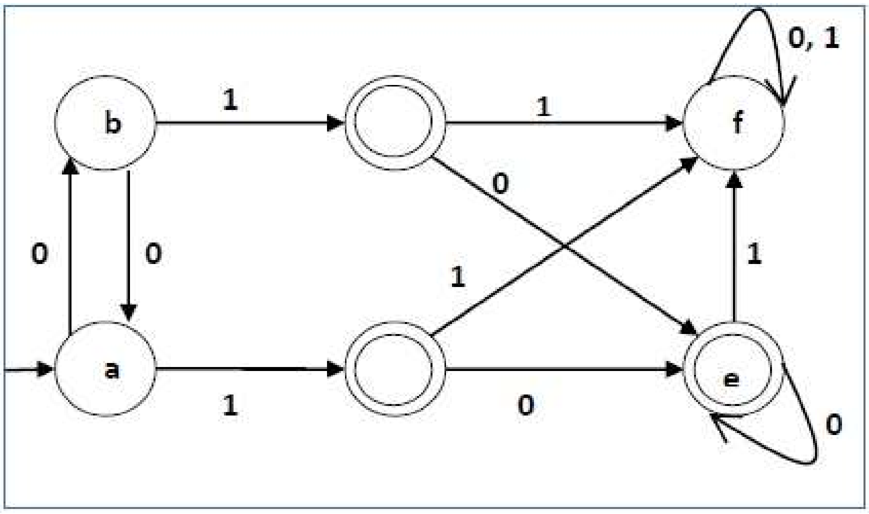
Step 2 − Increment k by 1. For each partition in P_k, divide the states in $P_k$into two partitions if they are k-distinguishable. Two states within this partition X and Y are k-distinguishable if there is an input **S** such that **δ(X, S)** and **δ(Y, S)** are (k-1)-distinguishable.

Step 3 − If $P_k \neq P_{k-1}$, repeat Step 2, otherwise go to Step 4.

Step 4 − Combine kth equivalent sets and make them the new states of the reduced DFA.

Example

Let us consider the following DFA −



q	δ(q,0)	δ(q,1)
a	B	C
b	A	D
c	E	F
d	E	F
e	E	F

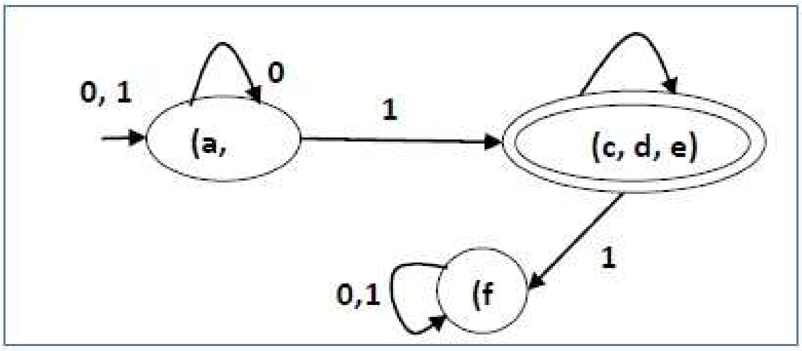
f	F	F

Let us apply the above algorithm to the above DFA −

- P_0 = {(c,d,e), (a,b,f)}
- P_1 = {(c,d,e), (a,b),(f)}
- P_2 = {(c,d,e), (a,b),(f)}

Hence, $P_1 = P_2$.

There are three states in the reduced DFA. The reduced DFA is as follows −



Q	**δ(q,0)**	**δ(q,1)**
(a, b)	(a, b)	(c,d,e)
(c,d,e)	(c,d,e)	(f)

(f)	(f)	(f)

Mealy and Moore Machines

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

- Mealy Machine
- Moore machine

Mealy Machine

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

It can be described by a 6 tuple $(Q, \sum, O, \delta, X, q_0)$ where −

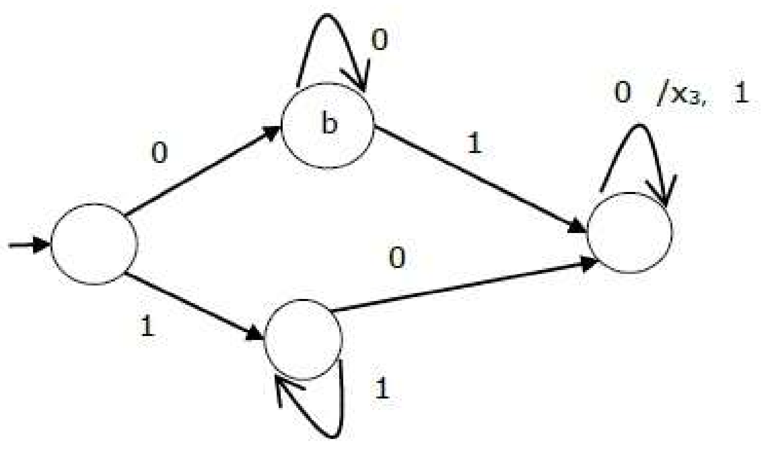
- **Q** is a finite set of states.
- $\sum$ is a finite set of symbols called the input alphabet.
- **O** is a finite set of symbols called the output alphabet.
- **δ** is the input transition function where $\delta: Q \times \sum \rightarrow Q$

- **X** is the output transition function where X: Q × ∑ → O
- **q_0** is the initial state from where any input is processed (q_0 ∈ Q).

The state table of a Mealy 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 Mealy 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, ∑, O, δ, X, q_0) where −

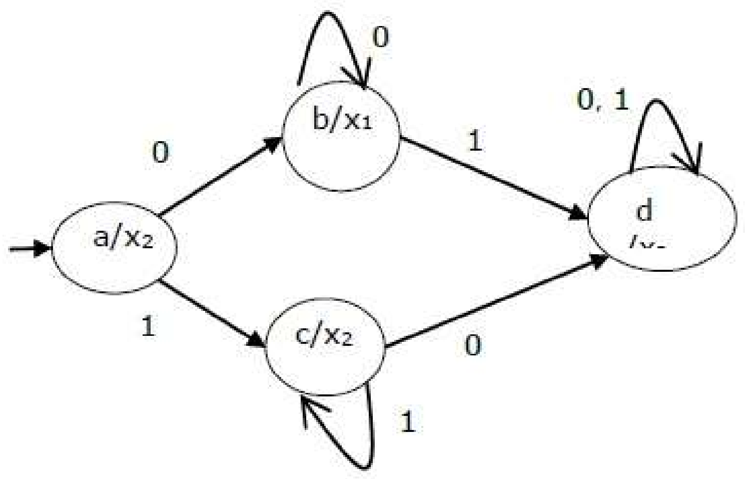
- **Q** is a finite set of states.
- ∑ is a finite set of symbols called the input alphabet.
- **O** is a finite set of symbols called the output alphabet.
- **δ** is the input transition function where δ: Q × ∑ → Q
- **X** is the output transition function where X: Q → O

- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).

The state table of a Moore Machine is shown below –

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

The state diagram of the above Moore Machine is –



Mealy Machine vs. Moore Machine

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

Mealy 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	Generally, it has more states than Mealy Machine.

Moore Machine.	
Output changes at the clock edges.	Input change can cause change in output change as soon as logic is done.
Mealy 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 Mealy Machine

Algorithm 4

Input − Moore Machine

Output − Mealy Machine

Step 1 − Take a blank Mealy Machine transition table format.

Step 2 − Copy all the Moore Machine transition states into this table format.

Step 3 − Check the present states and their corresponding outputs in the Moore Machine state table; if for a state Q_i output is m, copy it into the output columns of the Mealy Machine state table wherever Q_i appears in the next state.

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

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

Step 1 & 2 –

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

Step 3 −

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

Mealy Machine to Moore Machine

Algorithm 5

Input − Mealy Machine

Output − Moore Machine

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

Step 2 − If all the outputs of Qi are same, copy state Q_i. If it has n distinct outputs, break Q_i into n states as Q_{in} where **n** = 0, 1, 2.......

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

Example

Let us consider the following Mealy Machine −

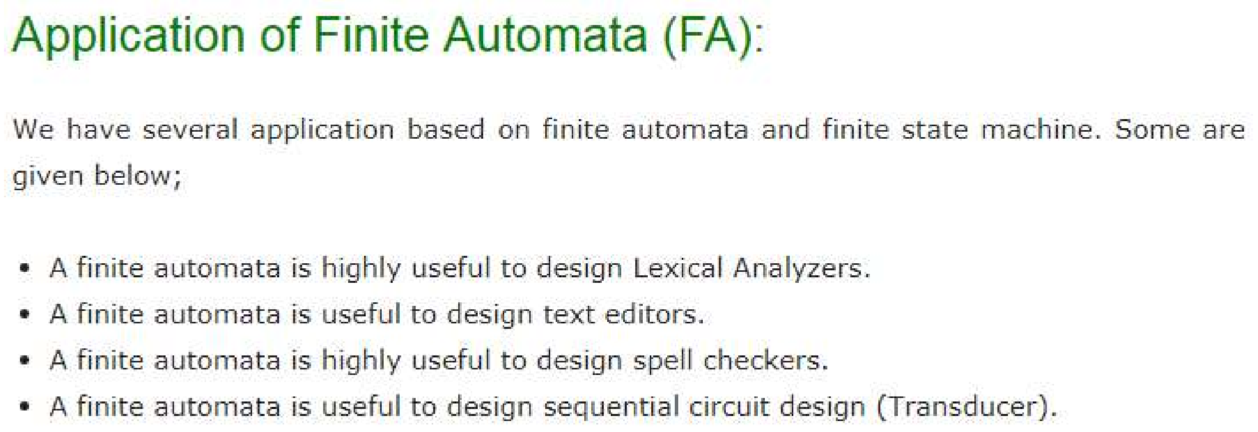
Present State	**Next State**			
	a = 0		**a = 1**	
	Next State	**Output**	**Next State**	**Output**
→a	d	0	b	1
b	a	1	d	0
c	c	1	c	0

d	b	0	a	1

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 **b_0, b_1** and **c** into **c_0, c_1**.

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

Applications of Finite Automata



Application of Finite Automata (FA):

We have several application based on finite automata and finite state machine. Some are given below;

- A finite automata is highly useful to design Lexical Analyzers.
- A finite automata is useful to design text editors.
- A finite automata is highly useful to design spell checkers.
- A finite automata is useful to design sequential circuit design (Transducer).

String matching algorithms :

String Processing

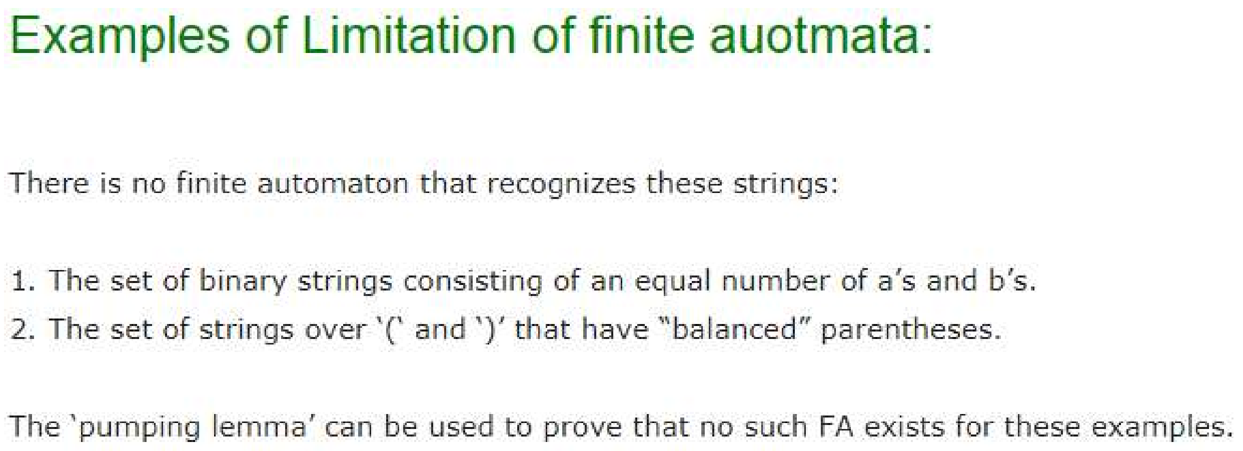
Consider finding all occurrences of a short string (pattern string) within a long string (text string).

This can be done by processing the text through a DFA: the DFA for all strings that end with the pattern string. Each time the accept state is reached, the current position in the text is out- put.

Example:

Finding 1001

To find all occurrences of pattern 1001, con- struct the DFA for all strings ending in 1001.



Examples of Limitation of finite auotmata:

There is no finite automaton that recognizes these strings:

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

The 'pumping lemma' can be used to prove that no such FA exists for these examples.