UNIT-1
Finite Automata

Syllabus:

UNIT-I:

Automata: Need for Automata Theory, Chomsky hierarchy, Acceptance of a string, Design of NFA with Є, NFA without Є, DFA, Equivalence of NFA, DFA

Finite Automata Conversions: Conversion from NFA Є to NFA, NFA to DFA, Minimization of DFA, Moore and Mealy Machines

Automata Theory

→ Automata theory is the study of abstract computing devices or machines.
i.e., the study of what can be done by computers in principle and what can be done in practice.

→ In automata theory, we study about computers by building abstract mathematical model, study their limitations by analyzing the types of inputs which they operate successfully.

Need for Automata Theory

→ Automata theory provides concepts and principles that help us understand the general nature of the computer.

→ There are immediate and important applications of automata theory such as digital design, programming languages, and compilers and so on.

→ Through automata theory we are able to understand how computer compute function and solve problems, what are computable or decidable problems.

→ Finite automata are useful for building several different kinds of software including the lexical analysis component of a compiler and systems for verifying the correctness of circuits or protocols.

→ Automata theory concepts help us understand what we can expect from our software.

Three Central concepts of Automata Theory:

Language: a system suitable for the expression of ideas, facts, or concepts.

Automata: an abstract mathematical model (theoretical machine) of a computer.
(Language Acceptor or Language Recognizer)

Grammar: a mathematical mechanism to describe language (Language Generator).

Three basic concepts of Automata Theory:

Alphabet (∑): a finite, non-empty set of symbols.

String (w): a finite sequence of symbols chosen from some alphabet.

Language (L): a set of strings over the given alphabet.

Terminalgy used in Automata theory

Alphabet

→ A finite, nonempty set of symbols is called alphabet.

→ Conventionally we use the symbol ∑ for an alphabet

→ Common alphabets include:

1. $\Sigma = \{0, 1\}$, the *binary* alphabet.
2. $\Sigma = \{a, b, \ldots, z\}$, the set of all lower-case letters.
3. The set of all ASCII characters, or the set of all printable ASCII characters.

String

→ A finite sequence of symbols chosen from some alphabet is called string.

→ For example, if the alphabet Σ = {*a*, *b*}, then *abab* and *aaabbba* are strings on Σ.

→ For example, if the alphabet Σ = {0, 1}, then *011100, 00000,11111,1010101* are strings on Σ.

→ The set of all strings over $\sum$ will be written $\sum^*$

Language

→ A set of strings drawn from some alphabet is called language.

→ A language over $\sum$ is a subset of $\sum^*$

→ Language can be empty or non-empty and finite or non-finite.

1. The language of all strings consisting of n 0's followed by n 1's, for some $n \geq 0$: $\{\epsilon, 01, 0011, 000111, \ldots\}$.
2. The set of strings of 0's and 1's with an equal number of each:

$$\{\epsilon, 01, 10, 0011, 0101, 1001, \ldots\}$$

3. The set of binary numbers whose value is a prime:

$$\{10, 11, 101, 111, 1011, \ldots\}$$

4. Σ^* is a language for any alphabet Σ.
5. $\emptyset$, the empty language, is a language over any alphabet.
6. $\{\epsilon\}$, the language consisting of only the empty string, is also a language over any alphabet. Notice that $\emptyset \neq \{\epsilon\}$; the former has no strings and the latter has one string.

NOTE:

→ Alphabet and string must be finite but language can be finite or infinite.

→ Alphabet must be non-empty but string and language can be empty or non-empty.

Basic Operations on Strings:

1. Length of string
2. Concatenation of string
3. Reverse of string
4. Substring
5. Prefix and Suffix of string

Length:

→ The **length** of a string *w*, denoted by |*w*|, is the number occurrence of symbols in the string. i.e.; Number of positions for symbols in the string.

→ For instance string **01101** has length 5

→ We will frequently need to refer to the **empty string**, which is a string with no symbols at all. It will be denoted by ε .

→ For example |01101| = 5 , |ε| = 0

Concatenation:

→ If *x* and *y* are two strings over an alphabet, the concatenation of *x* and *y* is written *xy* and consists of the symbols of *x* followed by those of *y*.

→ If *x* = *ab* and *y* = *bab*, for example, then *xy* = *abbab* and *yx* = *babab*.

→ When we concatenate the null string ε with another string, the result is just the other string (for every string *x*, *x*ε = ε*x* = *x*);

→ In general, for two strings *x* and *y*, |*xy*| = |*x*| + |*y*|.

NOTE

→ If Σ is an alphabet, then we use Σ* to denote the set of strings obtained by concatenating zero or more symbols from Σ. The set Σ* always contains ε

→ We denote $\sum^k$ to be the set of strings of length k

→ $\sum^0 = \{\varepsilon\}$ regardless of what alphabet $\sum$ is. That is ε is the only string whose length is 0.

→ For example

$$\text{If } \Sigma = \{0,1\}, \text{ then } \Sigma^1 = \{0,1\}, \Sigma^2 = \{00,01,10,11\},$$

$$\Sigma^3 = \{000,001,010,011,100,101,110,111\}$$

→ Note that

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$$

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots.$$

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}.$$

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

Reverse:

→ The **reverse** of a string is obtained by writing the symbols in reverse order.

→ If *w* is a string $w = a_1a_2 \ldots a_n$, then its reverse w^R is $w^R = a_n \ldots a_2\, a_1$.

→ For example, if **abbaaa** is string, then its reverse is **aaabba**

Substring

→ Any string of consecutive symbols in some string *w* is said to be a substring of w.

→ For example, if *w* = abcd then {ε, a, b, c, d, ab, bc, cd, abc, bcd, abcd} is the set of all its substrings.

Prefix and Suffix:

→ A prefix of a string is a substring of leading symbols of that string.
→ A suffix of a string is a substring of trailing symbols of that string,
→ For example, if *w* = *abbab*, then {ε, a, ab, abb, abba, abbab} is the set of all prefixes of *w*, while {abbab, bbab, bab, ab, b,ε } is the set of all its suffixes.

Basic Operations on Languages:

1. Union
2. Intersection
3. Difference
4. Complement
5. Concatenation
6. Reverse:
7. Kleene Closure
8. Positive Closure

Complement:

→ The complement of a language is defined with respect to Σ*; that is, the complement of *L* is

$$\overline{L} = \Sigma^* - L.$$

Reverse:

→ The reverse of a language is the set of all string reversals, that is,

$$L^R = \left\{ w^R : w \in L \right\}.$$

Concatenation:

→ The concatenation of two languages *L*1 and *L*2 is the set of all strings obtained by concatenating any element of *L*1 with any element of *L*2; specifically,

$$L_1 L_2 = \left\{ xy : x \in L_1, y \in L_2 \right\}.$$

→ For example L_1 = {hope, fear} and L_2 = {less, fully} then
L_1L_2= {hopeless, hopefully, fearless, fearfully}
→ For example, {*a, aa*}{ε, *b, ab*} = {*a, ab, aab, aa, aaab*}. Because *εx* = *xε* for every string *x*.
→ We have {ε}L = L{ε} = L

Kleene Closure:

→ For a language *L* over an alphabet ∑, we use the notation L^* to denote the language of all strings that can be obtained by concatenating zero or more strings in *L*.
→ This operation on a language *L* is known as the *Kleene star*, or Kleene closure, after the mathematician Stephen Kleene.

$$L^* = L^0 \cup L^1 \cup L^2 \cdots$$

Positive Closure

→ For a language L over an alphabet ∑, we use the notation L^+ to denote the language of all strings that can be obtained by concatenating one or more strings in L.

→ This operation on a language L is known as the positive closure.

$$L^+ = L^1 \cup L^2 \cdots .$$

Chomsky Hierarchy

→ Chomsky Hierarchy is a broad classification of the various types of grammars available. These include un restricted grammar, context-free grammar, context-sensitive grammar and regular gramma

→ Each category represents a class of languages that can be recognized by a different automaton

→ The classes are nested, with type 0 being the largest and most general, and type 3 being the smallest and most restricted.

→ The four classes of languages: regular, context-free, context-sensitive, and recursively enumerable—make up what is often referred to as the *Chomsky hierarchy*.

→ Chomsky Hierarchy contains 4 classes of languages, each having a corresponding model of computation and a corresponding type of grammar.

→ Chomsky himself designated the four types as type 3, type 2, type 1, and type 0, in order from the most restrictive to the most general.

→ Each level of the hierarchy can be characterized by a class of grammars and by a specific model of computation.

→ Regular Grammar(RG) generates regular language(RL) which is accepted by Finite Automata(FA)

→ Context Free Grammar(CFG) generates Context Free Language(CFL) which is accepted by Push Down Automata(PDA)

→ Context sensitive Grammar(CSG) generates Context Sensitive Language(CSL) which is accepted by Linear Bounded Automata(LBA)

→ Recursive Enumerable Grammar(REG) generates Recursive Enumerable Language(REL) which is accepted by Turing Machine(TM)

→ A language *L* is *recursively enumerable* if there is a TM that accepts *L*.

→ A language is a context-sensitive language (CSL) if it can be generated by a context-sensitive grammar.

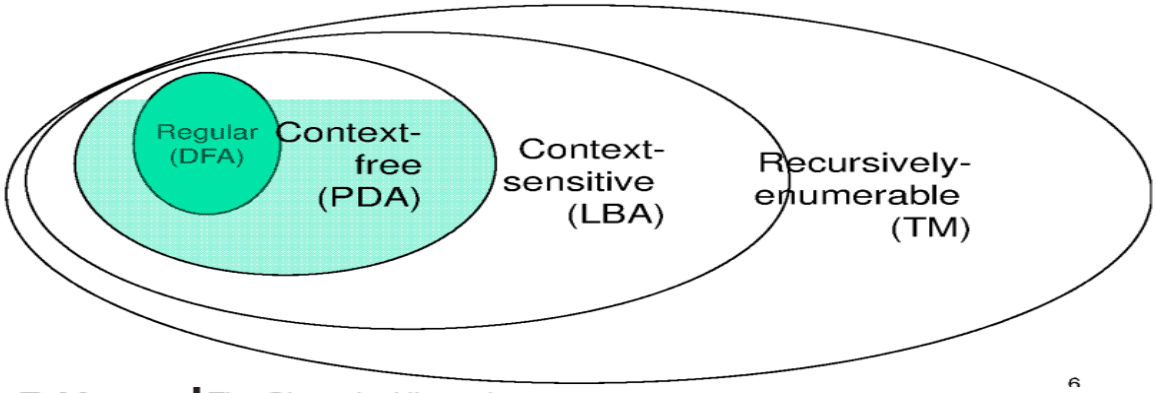
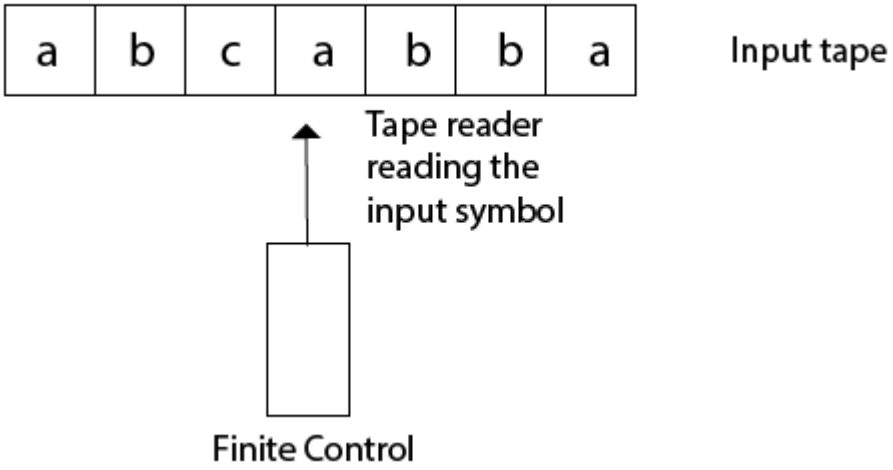


Table 8.21 | The Chomsky Hierarchy

Type	Languages (Grammars)	Form of Productions in Grammar	Accepting Device
3	Regular	$A \to aB$, $A \to \Lambda$ $(A, B \in V, a \in \Sigma)$	Finite automaton
2	Context-free	$A \to \alpha$ $(A \in V, \alpha \in (V \cup \Sigma)^*)$	Pushdown automaton
1	Context-sensitive	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup \Sigma)^*, \lvert\beta\rvert \geq \lvert\alpha\rvert,$ α contains a variable)	Linear-bounded automaton
0	Recursively enumerable (unrestricted)	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup \Sigma)^*,$ α contains a variable)	Turing machine

Finite Automata(FA) Model



The various components of FA are explained as follows:

- **Input tape:** The input tape is divided into squares (or cells), each square containing a single symbol from the input alphabet.
- **Reading head:** The head examines only one square at a time and can move one square to the right side. Initially the head is places at the leftmost square of the tape

- **Finite control:** It can be in any of a finite number of states and change the state in some defined manner. Initially finite control is set to initial state(starting sate)

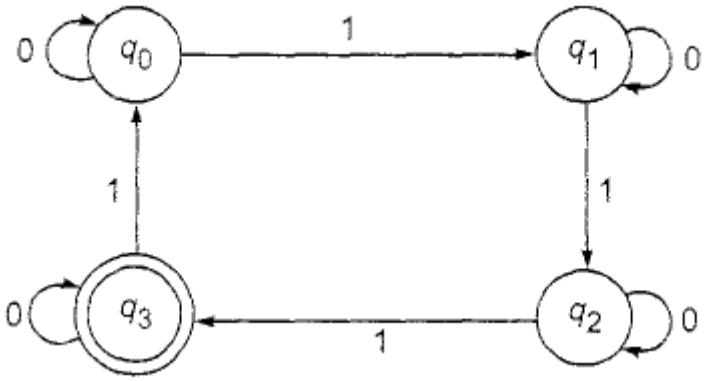
Deterministic Finite Automata(DFA)

→ A **deterministic finite automata** or **DFA** is defined by the quintuple
$M = (Q,\Sigma,\delta,q_0, F)$, where
Q is a finite, nonempty set of **states**,
Σ is a finite, nonempty set of symbols called the **input alphabet**,
$\delta : Q \times \Sigma \rightarrow Q$ is a **transition function**,
$q_0 \in Q$ is the **initial state (starting state)**, Always one
$F \subseteq Q$ is a set of **final states(accepting states).**

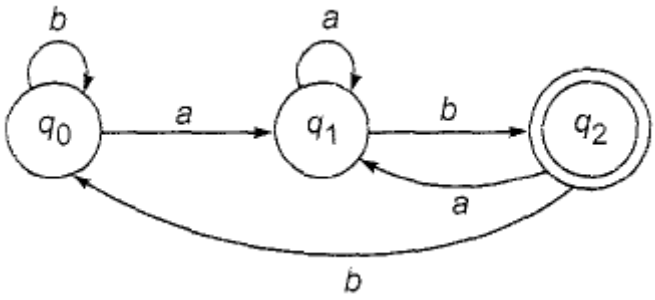
→ A deterministic finite automata operates in the following manner:

- At the initial time, it is assumed to be in the initial state $q0$, with its input mechanism on the leftmost symbol of the input string. During each move of the automaton, the input mechanism advances one position to the right, so each move consumes one input symbol. When the end of the string is reached, the string is accepted if the automaton is in one of its final states. Otherwise the string is rejected.
- The input mechanism can move only from left to right and reads exactly one symbol on each step. The transitions from one internal state to another are governed by the transition function δ

→ Example1: Construct a DFA accepting all strings w over {0,1} such that the number of l' s in w is 3 mod 4.

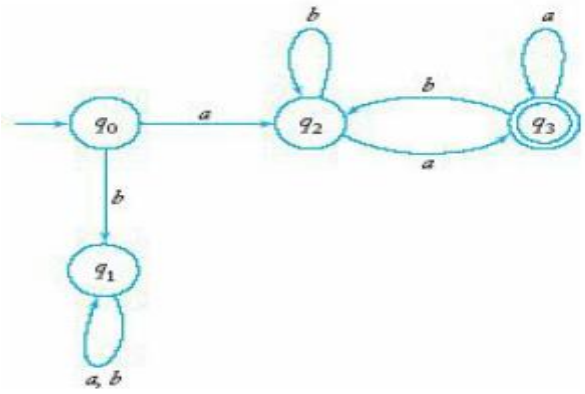


→ Example 2: Construct a DFA accepting all strings over *{a. b}* ending in ***ab***

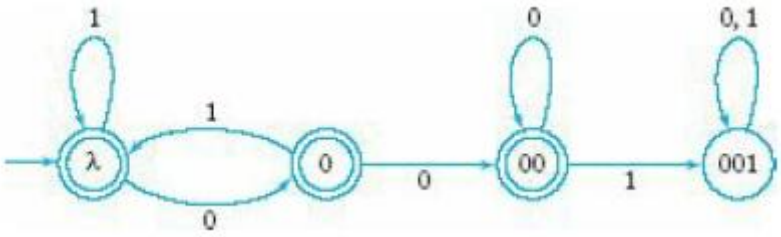


→ Example 3: Design a DFA for the language

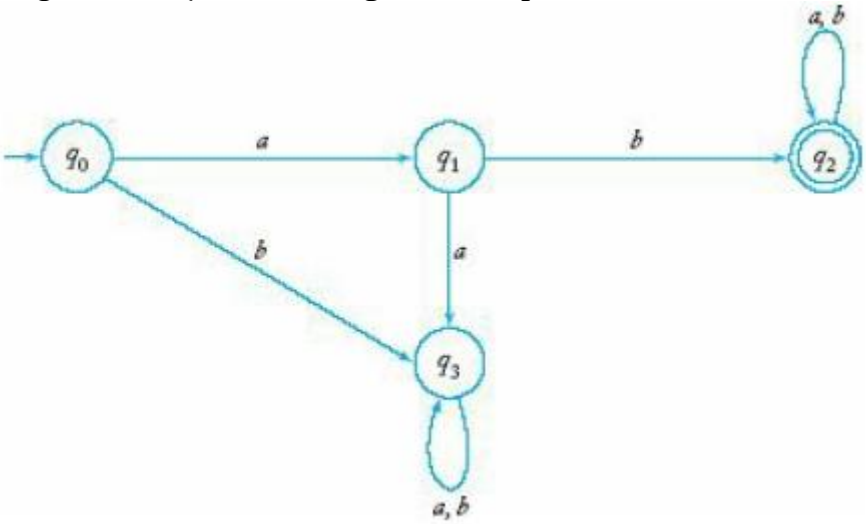
$$L = \{awa : w \in \{a,b\}^*\}$$



→ Example4: Find a DFA that accepts all the strings on {0,1}, except those containing the substring 001.

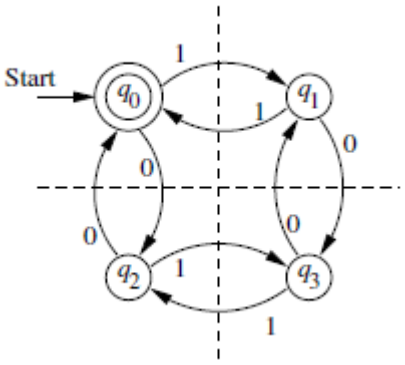


→ Example5: Find a deterministic finite accepter that recognizes the set of all strings on Σ= {*a*,*b*} starting with the prefix *ab*.

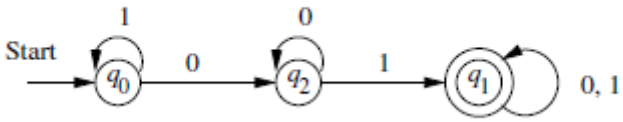


→ Example6: Design a DFA to accept the language

$L = \{w \mid w \text{ has both an even number of 0's and an even number of 1's}\}$



→ Example7: Find DFA that accepts all strings over {0, 1} with substring **01**



Nondeterministic Finite Automaton (NFA or NDFA)

→ A FA in which some moves can't be determined uniquely is called NFA

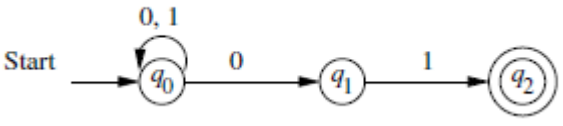
→ NFA allows multiple transitions (zero, one or more) from the same combination of state and input symbol

→ Formally, a nondeterministic finite automaton (NFA or NDFA) is defined as a 5-tuple $(Q, \sum, \delta, q_0, F)$, where

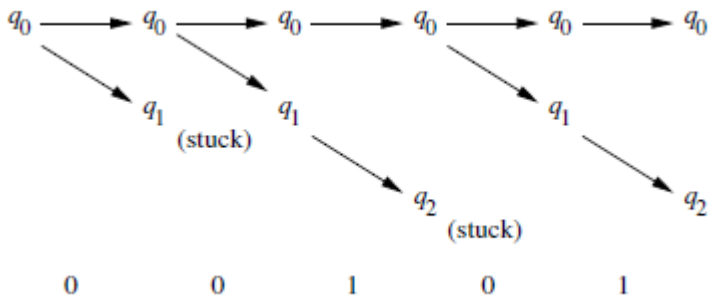
- Q is a finite nonempty set of states;
- $\sum$ is a finite nonempty set of inputs;
- δ is the transition function mapping from Q x $\sum$ into 2^Q which is the power set of Q, the set of all subsets of Q;
- $q_0 \in Q$ is the initial(starting) state; and
- $F \subseteq$ Q is the set of final (accepting) states.

→ Example1:

An NFA accepting all strings that end in 01



The states an NFA is in during the processing of input sequence 00101



→ Transition Table:

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

→ The NFA can be specified formally as

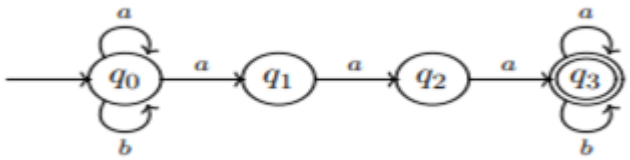
$$(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

Where is δ defined as

$\delta(q_0,0) = \{q_0,q_1\}$
$\delta(q_0,1) = \{q_0\}$
$\delta(q_1,0) = \phi$
$\delta(q_1,1) = \{q_2\}$
$\delta(q_2,0) = \phi$
$\delta(q_2,1) = \phi$

Example2: NFA for the language

$\{u \in \{a, b\}^* \mid u \text{ contains three consecutive } a\text{'s}\}$.



Significance of NFA:

→ NFA is an extension of the behaviour of DFA and in some cases greatly simplifies the description of automata

→ It is useful because constructing an NFA to recognize a given language is sometimes much easier than constructing a DFA for that language.

→ Using greater flexibility of NFA, language recognizer can be design without actually extending the class of language.

→ Certain theoretical results are more easily established for NFA's than for DFA's.

→ Nondeterminism is sometimes helpful in solving problems easily.

→ Nondeterminism is an effective mechanism for describing some complicated languages concisely.
→ A nondeterministic algorithm that can make the best choice would be able to solve the problem without backtracking. For this reason, Non-deterministic mechanisms can serve as models of search and backtrack algorithms.

Differences between NFA and DFA:

NFA	DFA
1.NFA stands for Nondeterministic Finite Automata.	DFA stands for Deterministic Finite Automata.
2.Some moves of NFA can't be determined uniquely by the input symbol and present state.	All moves are uniquely determined
3. Allows multiple transitions from the same combination of sate and input symbol	For each input symbol there is exactly one transition out of each state
4.It is not fixed that with a specific input where to go next on which state.	it is fixed that with a specific input where to go next on which state.
5. NFA can use Empty String transition. i.e can have transition on ε	DFA cannot use Empty String transition. i.e no transition on ε
6.There may not be any transition defined for the specific pair of current state and input symbol. i.e the set δ(q,a) may be empty	Transition is defined for all pairs of state and input symbol i.e. the set δ(q,a) must be nonempty
7.It is generalization of DFA i.e. Not all NFAs are DFA	It is specialization of NFA i.e. All DFAs are NFA
8.δ: Q X Σ → 2^Q next possible state belongs to power set of Q i.e. outcome of transition function is subset of Q	δ: Q x ∑→Q next possible state belongs to Q. i.e. outcome of transition function is element of Q
9.Each entry in transition table can have multiple states	Each entry in transition table must have single state
10.NFA is easier to design	DFA is more strict with rules and so difficult to design

Equivalence of NFA and DFA

Statement: For every NFA, there exists a DFA which simulates the behaviour of NFA. Alternatively, if L is the set accepted by NDFA, then there exists a DFA which also accepts L.

i.e. For every NFA, there exists an equivalent DFA

Proof: Let M = (Q, ∑, δ, q_0, F) be an NFA accepting L. We construct a DFA M' as: M' = (Q', ∑, δ', q_0 ', F') where

(i) Q' =2^Q (any state in Q' is denoted by [q_1 q_2, ... ,q_i], where q_1,q_2,.... ,q_i ∈ Q)

(ii) q_0' = [q_0]; and

(iii) F'is the set of all subsets of Q containing an element of *F*.

(iv) δ'([q_1 q_2, ... ,q_i],a) = [p_1 ,p_2, ..., p_j] iff δ({q_1 q_2, ... q_i},a) = {p_1 ,p_2, ..., p_j}

This can proved by induction on the length of input string x

That δ'(q_0 ',x) = [q_1 ,q_2, ..., q_i] iff δ({q_0 ,x) = {q_1 ,q_2, ..., q_i}

Basis: the result is trivial for |x| = 0 since q_0'= [q_0] and x must be ε

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

Induction: Let xa be a string of length m+1 with a in ∑.then

δ'(q_0',xa) = δ'(δ'(q_0',x),a)

By the induction hypothesis

δ'(q_0',x) = [p_1 ,p_2, ..., p_j] iff δ(q_0 ,x) = {p_1 ,p_2, ..., p_j}

By the definition of δ'

δ'([p_1 p_2, ... ,p_j],a) = [r_1 ,r_2, ..., r_k] iff δ({p_1 p_2, ... p_j},a) = {r_1 ,r_2, ..., r_k}

Thus

δ'(q_0',xa) = [r_1 ,r_2, ..., r_k] iff δ(q_0 ,xa) = {r_1 ,r_2, ..., r_k}

and also δ'(q_0',x) is in F' exactly when δ(q_0,x) contain a state of Q that is in F

Thus L(M) = L(M')

Conversion of NFA to DFA

→ Let given NFA is M=(Q,∑,δ,q_0,F),the equivalent DFA M'=(Q',∑, δ',q_0',F') can be constructed by subset construction method as fallows

i) From the states that are subset of states of given NFA ,start construction of δ' for[q_0] i,e starting state is [q_0]

ii) Continue the process of constructing δ' by considering only new states appearing earlier under input columns using

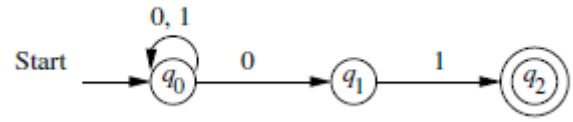
$$\delta'(S,a) = \bigcup_{P \text{ in } S} \delta(p,a)$$

iii) Stop the process of constructing when no new state appear under input columns

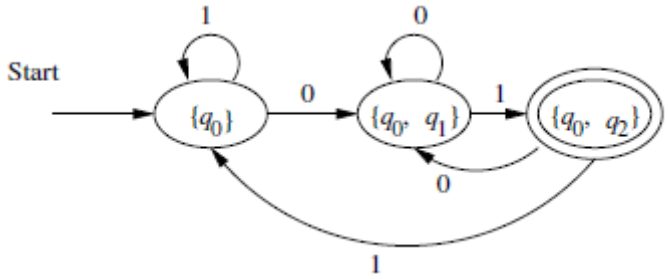
iv) F'(set of final sates in DFA) is the set of all subsets of Q which are having any final state of given NFA

Example:

An NFA accepting all strings that end in 01



The DFA constructed from the NFA



Minimization of Finite Automata

(Reducing Number of States of DFA)

→ *The minimization of FA refers to the* construction of finite automata(FA) with a minimum number of states**,** which is equivalent to the given finite automata.

→ *Ea*ch DFA defines a unique language but converse is not true. For a given language there may be many DFAs that accept it.

→ For each DFA, there exist an equivalent FA that has few states as any DFA accept the same language

→ Larger number of states in FA requires higher memory and computing power.

→ An NFA of n states result to 2^n maximum number of states in an equivalent DFA, therefore design of DFA is crucial.

→ Minimization of a DFA refers to detecting and eliminating those states whose absence does not affect the language acceptability of DFA. A reduced automata, called minimum-state DFA or minimal DFA consumes lesser memory, and complexity of implementation is reduced. This results to faster execution time, easier to analyze.

→ Minimum-state DFA(MDFA) is unique for the language

→ Table filling algorithm(also called Myhill-Nerode Algorithm) can be used to find the minimum-state DFA

Procedure

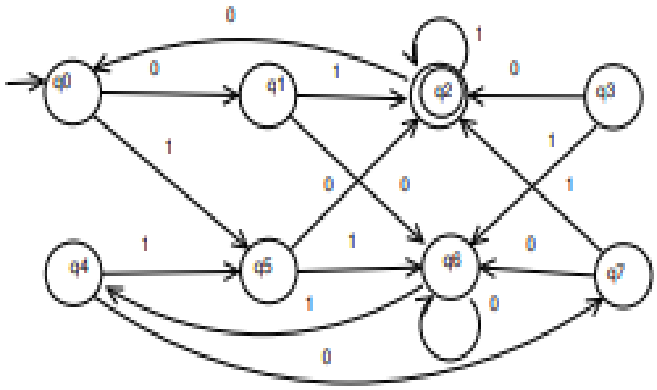
(Table filling algorithm or Myhill-Nerode Algorithm)

1. Remove all the states that are unreachable from the initial state via any set of the transition of DFA.

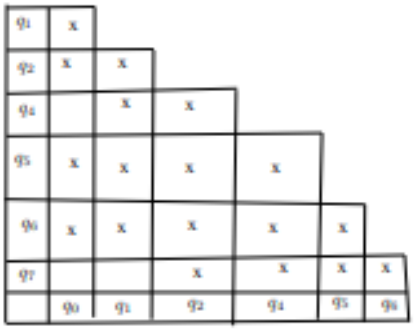
2. Draw the transition table for DFA obtained in step-1.
3. A table is constructed with an entry for each pair of states.
4. Initially X place in each entry corresponding to one final state and one non-final state.
5. Next, X is placed in entry corresponding to the pair (p,q) if δ (q, a) = r , δ (r, a) = s and pair(r,s) is marked as X, for any a $\varepsilon \sum$
6. After applying above steps for all the pairs remaining pairs are equivalent. All equivalent pairs are merged and used with single state

Example:

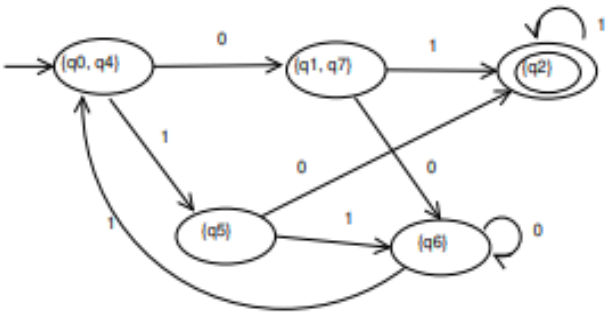
Given DFA:



Minimization Table:



Minimal DFA:



Types of states:

- **Unreachable states** are the states that are not reachable from the initial state of the DFA, for any input string. These states can be removed.
- **Dead states** are the states from which no final state is reachable. These states can be removed unless the automaton is required to be complete
- **Nondistinguishable states** (Equivalent states) are those that cannot be distinguished from one another for any input string. These states can be merged.
  i.e. two states p, q ∈ Q are equivalent, if for every string x ∈ Σ∗ the state that DFA reaches from p given x is accepting if and only if the state that DFA reaches from q given x is accepting.

- **Distinguishable(distinct) states:**
  Two states *p* and *q* are distinct if
  - $p \in F$ and $q \notin F$ or vice versa, or
  - for some $\alpha \in \Sigma$, $\delta(p, \alpha)$ and $\delta(q, \alpha)$ are distinct

NFA with ε-transition

(NFA-ε or ε-NFA)

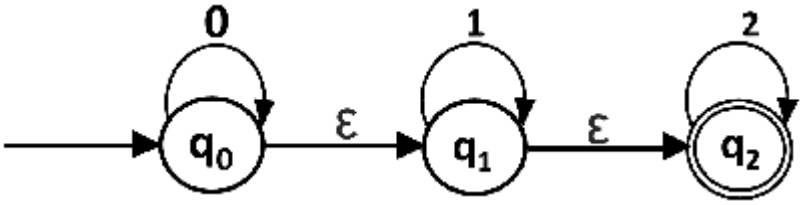
→ The NFA with epsilon-transition is a finite state machine in which the transition from one state to another state is allowed without any input symbol i.e. empty string ε.

→ FA that allows transition without consuming any input symbol (transition on empty input or transition on ε or transition with no input symbol at all) along with zero, one or more transition on input symbols is called ε-NFA or NFA- ε.

→ Formally, ε-NFA is defined as quintuple $M=(Q,\sum,\delta,q_0,F)$ where all components have their same interpretation as for an NFA, except that δ is mapping from $Q \times \sum U\{\varepsilon\}$ to 2^Q

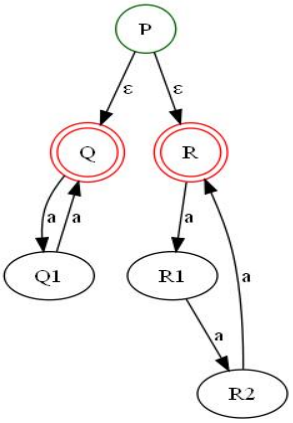
→ Example 1: ε-NFA accepting language consisting of any number of 0's followed by any number of 1's followed by any number of 2's

i.e. $L= \{0^m1^n2^p / m, n, p>=0\}$



→ Example 2:

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



Significance of ε-NFA

→ ε–NFA allows ε-transition but each ε along a path is invisible i.e. It contributes nothing to the string along the path.

→ ε -NFA simplifies the design of FA

→ ε –NFA's can be converted DFA's accepting the same language

→ This is very helpful when we study regular expression (RE) and prove the equivalence between the class of language accepted by RE and finite automata.

Converting NFA with ε to NFA:

→ NFA with ε can be converted to NFA without ε.To do this, we will use a method, which can remove all the ε transition from given NFA. The method will be:

→ Let M=(*(Q,* ∑,*δ, q0,* F) be an ε-NFA, then equivalent NFA without ε-transition is M' =(*Q,* ∑,*δ', q0,* F')

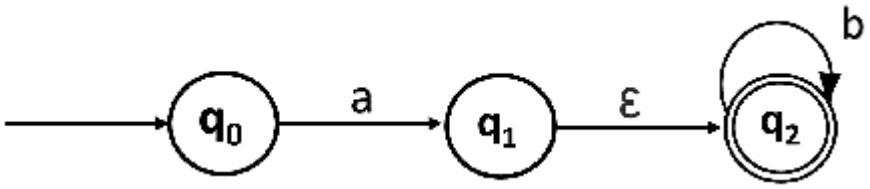
1. Find out ε-closure of each state of given ε-NFA.
2. Then δ'(q,a) for each state q and each input symbol using

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

3. F^1 = set of all sates whose ε-closure contains a final state of ε-NFA

Example:

Convert the following NFA with ε to NFA without ε.



Solutions: 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}

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

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

δ'(q0, 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(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(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(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}

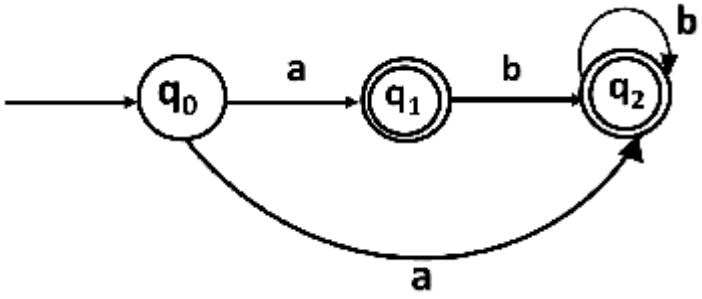
Now we will summarize all the computed δ' transitions:

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}

The transition table can be:

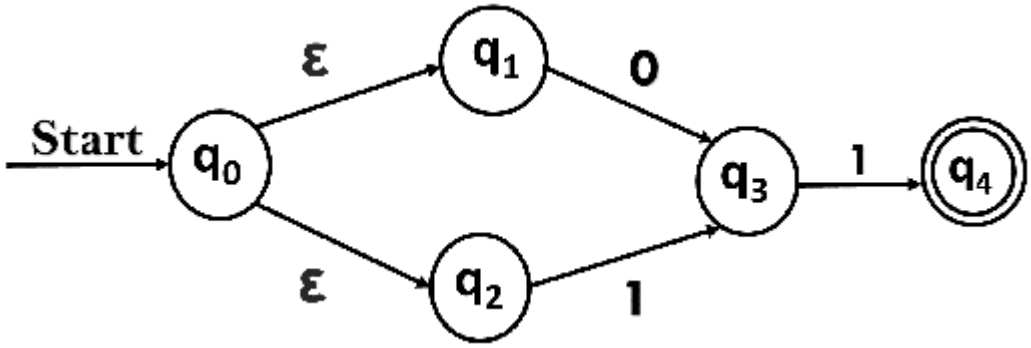
States	a	b
→q0	{q1, q2}	Φ
*q1	Φ	{q2}
*q2	Φ	{q2}

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:



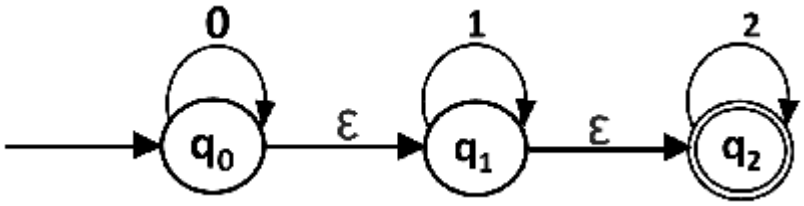
Exercise 1:

Convert the NFA with ε into its equivalent NFA.



Exercise 1:

Convert the given NFA into its equivalent DFA



Moore and Mealy Machines

(Finite Automata with output)

→ An FA that accepts input strings and translates them into output strings is called FA with outputs.
→ Both Moore and Mealy machines are special case of DFA.
→ Both acts like output producers rather than language acceptors
→ In both no need to define final state
→ In both no concept of dead state
→ Both are equivalent in power
→ Moore machine is created by E.F.Moore in 1956 and Mealy machine is created by G.H.Mealy in 1955

Moore Machine

→ Moore machine is a FA whose outputs depend on only the present state.
→ The output is associated with the state i.e. Output is placed on state.
→ The output at a given time depends only upon the present state of the machine
→ On input ε it gives output $\lambda(q_0)$ i.e. output of starting state.

i.e. It responses for empty string

→ If the input string is of length n, the output string is of length n+1

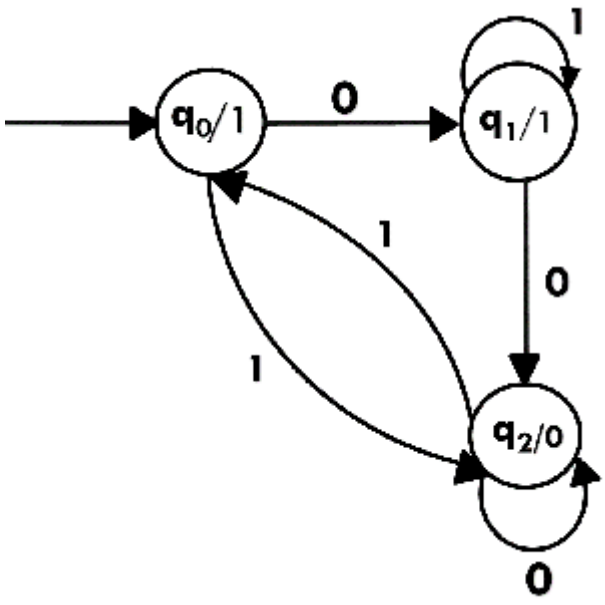
→ A Moore machine can be described by a 6 tuple (Q, ∑, Δ, δ, λ, q_0) where

Q is a finite set of states.
∑ is a finite set of symbols called the input alphabet.
Δ is a finite set of symbols called the output alphabet.
δ is the input transition function where δ: Q × ∑ → Q
λ is the output transition function where λ: Q → Δ
q_0 is the initial state from where any input is processed (q_0 ∈ Q).

→ The state table of a Moore Machine is shown below –

Current State	Next State (δ)		Output(λ)
	0	1	
q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

→ The state diagram of the above Moore Machine is –



Mealy Machine

→ A Mealy Machine is a FA whose output depends on the present state as well as the present input.

→ The output is associated with the transition i.e. Output is placed on transition.

→ The output at a given time is function of the present input as well as the present state of the machine

→ On input ε it gives output ε i.e. It does not responses for empty string

→ If the input string is of length n, the output string is also of length n

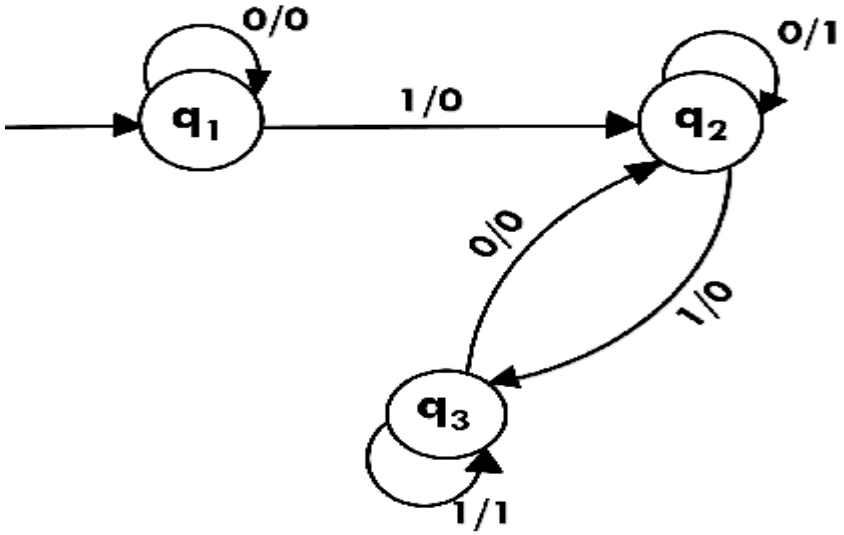
→ It can be described by a 6 tuple $(Q, \sum, \Delta, \delta, X, q_0)$ where

Q is a finite, non-empty set of states.
$\sum$ is a finite, non-empty set of symbols called the input alphabet.
Δ is a finite, non-empty set of symbols called the output alphabet.
δ is the transition function where $\delta: Q \times \sum \rightarrow Q$
λ is the output function where $\lambda: Q \times \sum \rightarrow \Delta$
q_0 is the initial state from where any input is processed ($q_0 \in Q$).

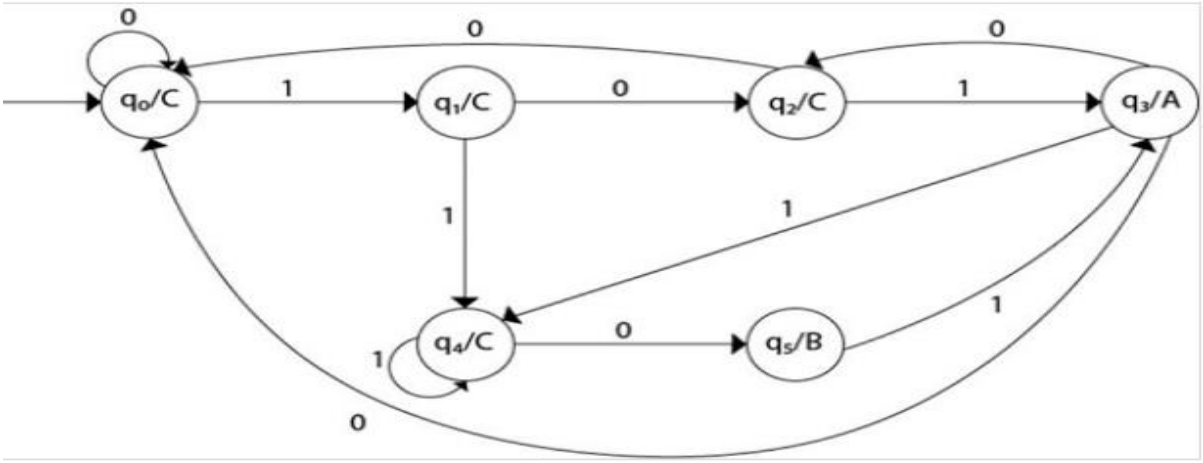
→ The state table of a Mealy Machine is shown below –

Present State	Next State 0		Next State 1	
	State	o/P	State	o/P
q_1	q_1	0	q_2	0
q_2	q_2	1	q_3	0
q_3	q_2	0	q_3	1

→ The state diagram of the above Mealy Machine is –



Problem: Design a Mealy machine for a binary input sequence such that if it has a substring 101, the machine output A, if the input has substring 110, it outputs B otherwise it outputs C.



Problem: Design Mealy and Moore machine to find 2's complement of binary number.

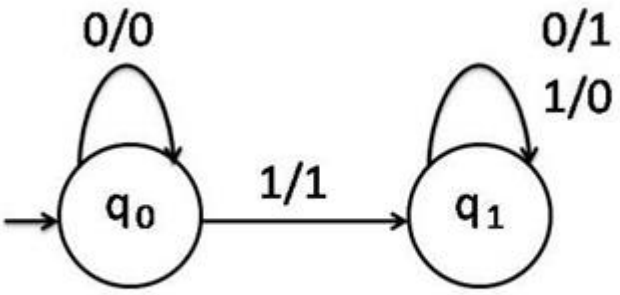
2's complement means 1's complement and adds 1 at LSB. But here one shortcut method to generates 2's complement.

1 0 1 0 0 0

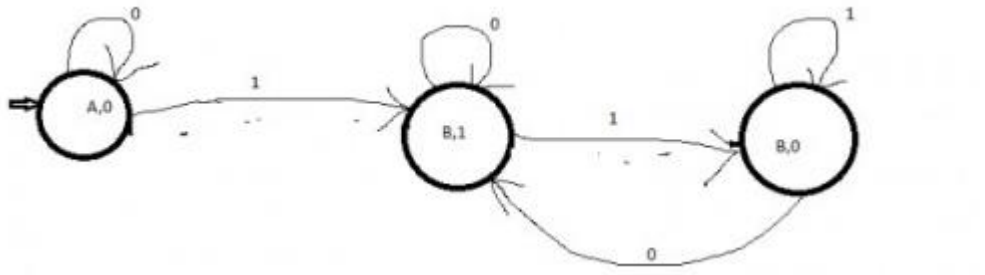
Left all zero as it is from LSB

First 1 from LSB keep same

Change the remaining bit 0 to 1 and 1 to 0



Mealy Machine of 2's Complement



Moore machine for 2's complement

Conversion from Moore Machine to Mealy Machine

In the Moore machine, the output is associated with every state, and in the mealy machine, the output is given along the edge with input symbol. The equivalence of the Moore machine and Mealy machine means both the machines generate the same output string for same input string.

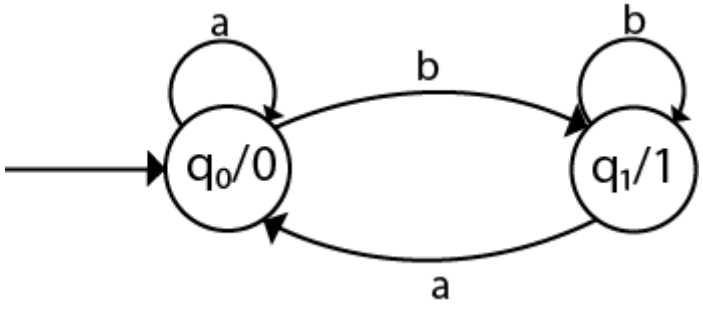
Method:

Let M = (Q, ∑, δ, λ, q0) be a Moore machine. The equivalent Mealy machine can be represented by M' = (Q, ∑, δ, λ', q0). The output function λ' can be obtained as:

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

Example 1:

Convert the following Moore machine into its equivalent Mealy machine.



Solution:

The transition table of given Moore machine is as follows:

Q	A	b	Output(λ)
q0	q0	q1	0
q1	q0	q1	1

The equivalent Mealy machine can be obtained as follows:

$\lambda'(q0, a) = \lambda(\delta(q0, a))$
$= \lambda(q0)$
$= 0$

$\lambda'(q0, b) = \lambda(\delta(q0, b))$
$= \lambda(q1)$
$= 1$

The λ for state q1 is as follows:

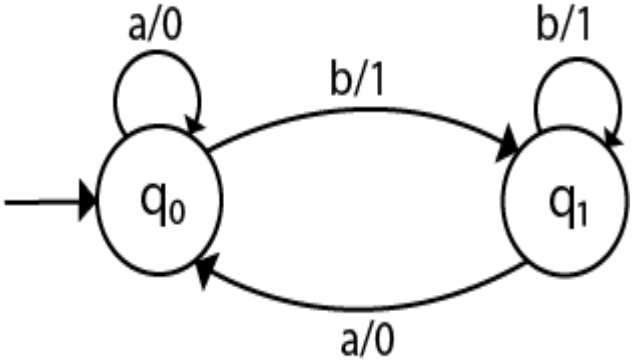
$\lambda'(q1, a) = \lambda(\delta(q1, a))$
$= \lambda(q0)$
$= 0$

$\lambda'(q1, b) = \lambda(\delta(q1, b))$
$= \lambda(q1)$
$= 1$

Hence the transition table for the Mealy machine can be drawn as follows:

Σ / Q	Input 0		Input 1	
	State	O/P	State	O/P
q_0	q_0	0	q_1	1
q_1	q_0	0	q_1	1

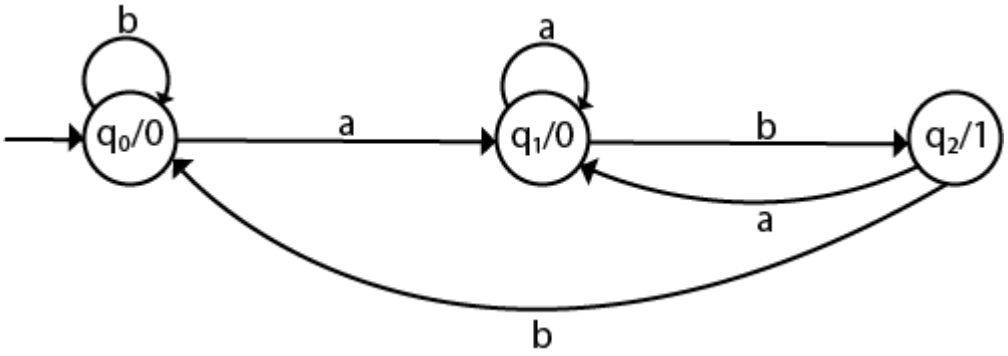
The equivalent Mealy machine will be,



Note: The length of output sequence is 'n+1' in Moore machine and is 'n' in the Mealy machine.

Example 2:

Convert the given Moore machine into its equivalent Mealy machine.



Solution:

The transition table of given Moore machine is as follows:

Q	A	b	Output(λ)
q0	q1	q0	0
q1	q1	q2	0
q2	q1	q0	1

The equivalent Mealy machine can be obtained as follows:

λ' (q0, a) = λ(δ(q0, a))
= λ(q1)
= 0

λ' (q0, b) = λ(δ(q0, b))
= λ(q0)
= 0

The λ for state q1 is as follows:

λ' (q1, a) = λ(δ(q1, a))
= λ(q1)
= 0

$\lambda'(q1, b) = \lambda(\delta(q1, b))$
$= \lambda(q2)$
$= 1$

The λ for state q2 is as follows:

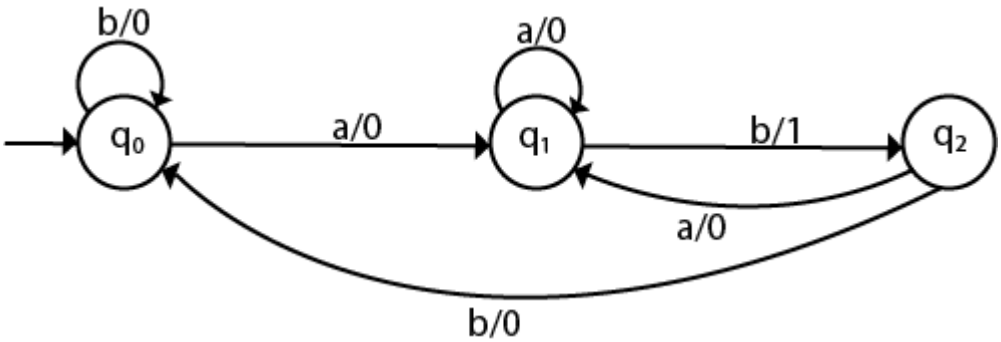
$\lambda'(q2, a) = \lambda(\delta(q2, a))$
$= \lambda(q1)$
$= 0$

$\lambda'(q2, b) = \lambda(\delta(q2, b))$
$= \lambda(q0)$
$= 0$

Hence the transition table for the Mealy machine can be drawn as follows:

Σ / Q	Input a		Input b	
	State	Output	State	Output
q_0	q_1	0	q_0	0
q_1	q_1	0	q_2	1
q_2	q_1	0	q_0	0

The equivalent Mealy machine will be,



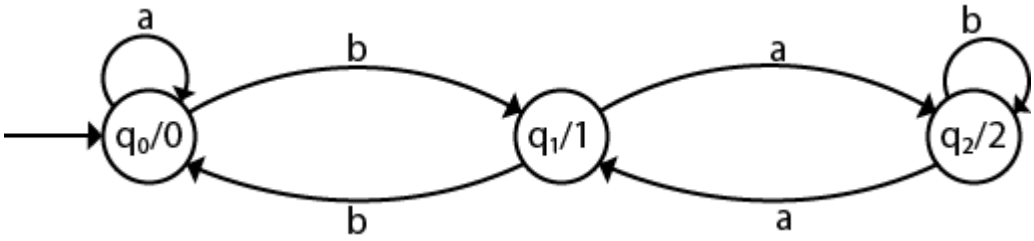
Example 3:

Convert the given Moore machine into its equivalent Mealy machine.

Q	a	b	Output(λ)
q0	q0	q1	0
q1	q2	q0	1
q2	q1	q2	2

Solution:

The transaction diagram for the given problem can be drawn as:



The equivalent Mealy machine can be obtained as follows:

λ' (q0, a) = λ(δ(q0, a))
= λ(q0)
= 0
λ' (q0, b) = λ(δ(q0, b))
= λ(q1)
= 1

The λ for state q1 is as follows:

λ' (q1, a) = λ(δ(q1, a))
= λ(q2)
= 2

λ' (q1, b) = λ(δ(q1, b))
= λ(q0)
= 0

The λ for state q2 is as follows:

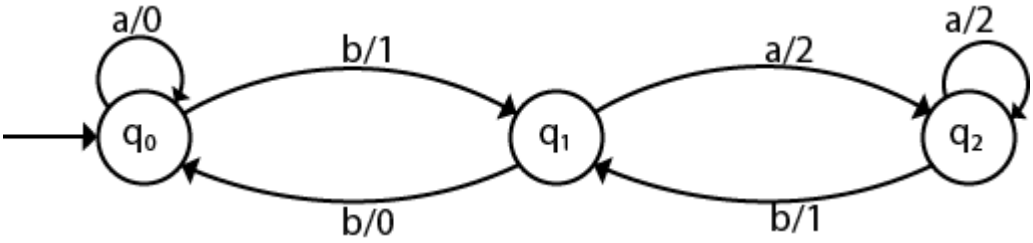
λ' (q2, a) = λ(δ(q2, a))
= λ(q1)
= 1

λ' (q2, b) = λ(δ(q2, b))
= λ(q2)
= 2

Hence the transition table for the Mealy machine can be drawn as follows:

Σ / Q	Input a		Input b	
	State	O/P	State	O/P
q_0	q_0	0	q_1	1
q_1	q_2	2	q_0	0
q_2	q_1	1	q_2	2

The equivalent Mealy machine will be,



Conversion from Mealy machine to Moore Machine

→ Let M=((Q, ∑, Δ, δ, X, q_0) be a mealy machine then there exist equivalent moore machine M'=(Q×Δ ,∑,Δ,δ',λ',[q_0 ,b_0) where
b_0 - arbitrarily selected member of Δ
Q×Δ - states of M' are pairs [q,b] consisting of states of M and an output symbol
δ'([q,b],a) = [δ(q,a), λ(q,a)]
λ'([q,b]) = b

Example:

Given Mealy machine:

Present state	Next state			
	Input a = 0		Input a = 1	
	state	output	state	output
$\rightarrow q_1$	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

$b_0 = 0$

$Q \times \Delta = \{[q_1,0],[q_1,1][q_2,0][q_2,1],[q_3,0],[q_3,1][q_4,0][q_4,1]\ \}$

$\delta'([q,b],a) = [\delta(q,a), \lambda(q,a)]$

$\delta'([q_1,0],0) = [\delta(q_1,0), \lambda(q_1,0)] = [q_3,0]$

$\delta'([q_1,0],1) = [\delta(q_1,1), \lambda(q_1,1)] = [q_2,0]$

$\delta'([q_1,1],0) = [\delta(q_1,0), \lambda(q_1,0)] = [q_3,0]$

$\delta'([q_1,1],1) = [\delta(q_1,1), \lambda(q_1,1)] = [q_2,0]$

$\delta'([q_2,0],0) = [\delta(q_2,0), \lambda(q_2,0)] = [q_1,1]$

$\delta'([q_2,0],1) = [\delta(q_2,1), \lambda(q_2,1)] = [q_4,0]$

$\delta'([q_2,1],0) = [\delta(q_2,0), \lambda(q_2,0)] = [q_1,1]$

$\delta'([q_2,1],1) = [\delta(q_2,1), \lambda(q_2,1)] = [q_4,0]$

$\delta'([q_3,0],0) = [\delta(q_3,0), \lambda(q_3,0)] = [q_2,1]$

$\delta'([q_3,0],1) = [\delta(q_3,1), \lambda(q_3,1)] = [q_1,1]$

$\delta'([q_3,1],0) = [\delta(q_3,0), \lambda(q_3,0)] = [q_2,1]$

$\delta'([q_3,1],1) = [\delta(q_3,1), \lambda(q_3,1)] = [q_1,1]$

$\delta'([q_4,0],0) = [\delta(q_4,0), \lambda(q_4,0)] = [q_4,1]$

$\delta'([q_4,0],1) = [\delta(q_4,1), \lambda(q_4,1)] = [q_3,0]$

$\delta'([q_4,1],0) = [\delta(q_4,0), \lambda(q_4,0)] = [q_4,1]$

$\delta'([q_4,1],1) = [\delta(q_4,1), \lambda(q_4,1)] = [q_3,0]$

$\lambda'([q,b]) = b$

$\lambda'([q_0,0]) = 0$
$\lambda'([q_0,1]) = 1$
$\lambda'([q_1,0]) = 0$
$\lambda'([q_1,1]) = 1$
$\lambda'([q_2,0]) = 0$
$\lambda'([q_2,1]) = 1$
$\lambda'([q_3,0]) = 0$
$\lambda'([q_3,1]) = 1$
$\lambda'([q_4,0]) = 0$
$\lambda'([q_4,1]) = 1$

Q×Δ	0	1	Output(λ)
$[q_1,0]$	$[q_3,0]$	$[q_2,0]$	0
$[q_1,1]$	$[q_3,1]$	$[q_2,0]$	1
$[q_2,0]$	$[q_1,0]$	$[q_4,1]$	0
$[q_2,1]$	$[q_1,0]$	$[q_4,0]$	0
$[q_3,0]$	$[q_2,0]$	$[q_1,0]$	0
$[q_3,1]$	$[q_2,1]$	$[q_1,1]$	1
$[q_4,0]$	$[q_4,1]$	$[q_3,0]$	0
$[q_4,1]$	$[q_4,0]$	$[q_3,0]$	0

Moore Machine vs Mealy Machine

The difference between the Mealy machine and Moore machine are as follows:

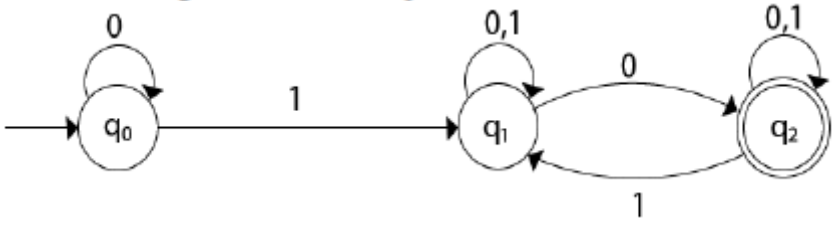
Moore Machine	Mealy Machine
The output is associated with the state i.e. Output is placed on state.	The output is associated with the transition i.e. Output is placed on transition.
The output at a given time depends only upon the present state of the machine	The output at a given time is function of the present input as well as the present state of the machine
The output function maps from Q to Δ	The output function maps from Q×∑ to Δ
On input ε it gives output $\lambda(q_0)$ i.e. output on starting state. i.e. It responses for empty string	On input ε it gives output ε i.e. It does not responses for empty string
If the input string is of length n, the output string is of length n+1	If the input string is of length n, the output string is also of length n
Generally, it has more states than Mealy Machine.	Generally, it has fewer states than Moore Machine.

Tutorial Questions:

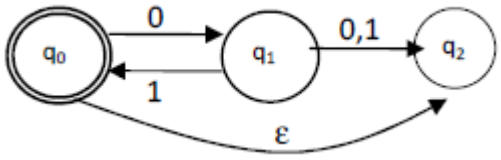
1. Why do we need to study automata theory and formal languages?
2. What is DFA? Find the DFA for the following languages over {0, 1}*
   i) The set of all strings such that number of 0's is odd
   ii) The set of all strings that contain exactly three 1's
   iii) The set of all strings that do not contain 1101
3. Design DFA which accepts language L={0,000,00000…….} over {0}
4. Design a DFA L(M) = {w | w ε {0, 1}*} and W is a string that does not contain three consecutive 1's.Process the string **0011001** on machine.
5. Design DFA to accept strings with 'c' and'd' such that number of d's are divisible by 4.Show the moves of DFA for the string **cdcddcd**
6. Design a DFA to accept the language
   L = {w |w has both an even number of 0's and an even number of 1's}.
   Represent obtained DFA by transition table. Show the transitions of DFA for the string **110101.**
7. What is DFA? Find the DFA's accepting for the following languages over the alphabet {a,b,c}:
   i) The set of all strings with **abc** as a substring
   ii) The set of all strings ending in **bac**

iii) The set of all strings starting with **cab**

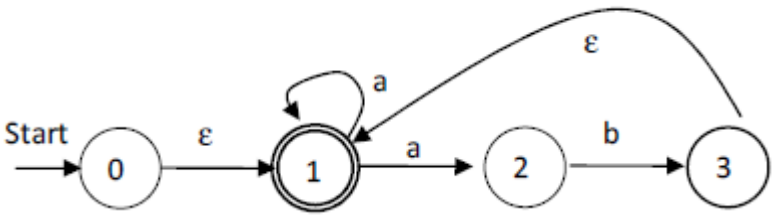
8. Draw a deterministic and non-deterministic finite automata for ∑= {A-Z} which accept a string containing "CSE" at the end of a string of {A-Z}.
9. Design DFA for the following over {a,b}.
   i) All string containing not more than three a's.
   ii) All strings that has at least two occurrences of b between any two occurrences of a.
   iii) All strings ending with aa
10. What is DFA? Construct a DFA accepting the language
    { W ∈ {a,b}*| W has neither aa nor bb as substring}
11. Draw the DFA for the following
    i) To accept decimal strings divisible by 3 over the alphabet ∑ = {0,1,2,3,4,5,6,7,8,9}
    ii) To accept odd number of a's and even number of b's over alphabet ∑ = {a,b}
12. Design a DFA that reads strings made up of letters in the word 'CHARIOT' and recognizes these strings that contain the word 'CAT' as a substring.
13. Define Deterministic and Non-deterministic finite automaton.
14. Differentiate between NFA and DFA?
15. Design an NFA with Σ = {0, 1} accepts all string in which the third symbol from the right end is always 0.
16. Construct an NFA that accepts the set of all strings over {0,1} that start with 0 or 1 and end with 10 or 01.
17. Design a NFA for the following language L={0101^n where n>0}
18. Construct DFA for the language L={$a^m cb^n$ / m,n>=0)
19. Describe the procedure of converting NFA to DFA with a suitable example.
20. Convert the given NFA to equivalent DFA



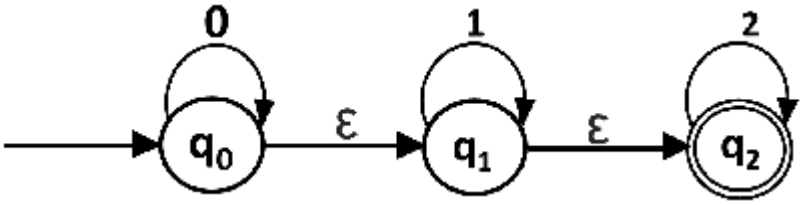
21. Construct a DFA equivalent to the NFA given below



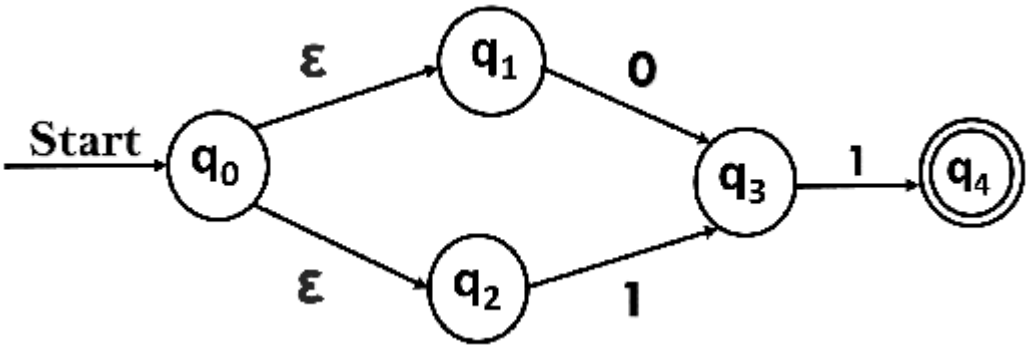
22. Define NFA with ∈ - moves and give example.
23. Depict the steps in converting an NFA with ∈ into NFA without ∈ with an example.
24. Show with an example equivalence between NFA with and without Є-transitions
25. Convert the following NFA-ε to NFA



26. Convert the following NFA-ε to NFA



27. Convert the following NFA-ε to NFA



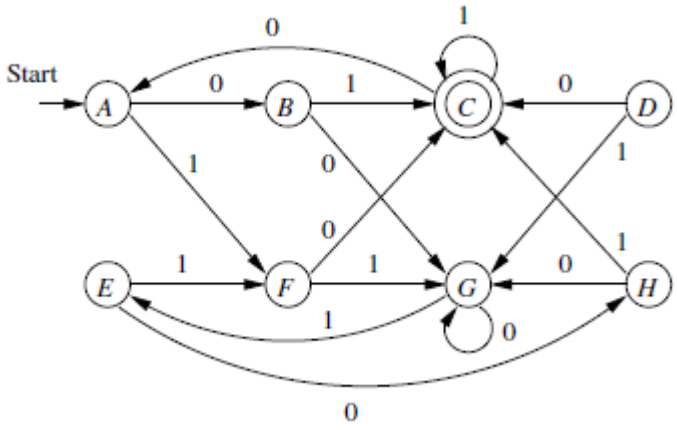
28. Explain the procedure for constructing minimum state DFA with an example.
    (OR) What is minimal DFA? Write the minimization Algorithm for DFA?
29. Let ∑= {a, b}, Give DFA that accepts any string with *aababb* as a substring. Minimize the DFA obtained.
30. Reduce the following DFA where q1 is the start state and q6 is the final state.

δ	0	1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_6	q_7
q_4	q_4	q_5
q_5	q_6	q_7
q_6	q_4	q_5
q_7	q_6	q_7

31. Construct Minimum state Automata for the following DFA?

δ	0	1
→ q1	q2	q6
q2	q1	q3
*q 3	q2	q4
q 4	q4	q2
q5	q4	q5
*q6	q5	q4

32. Reduce the DFA given belo**w**



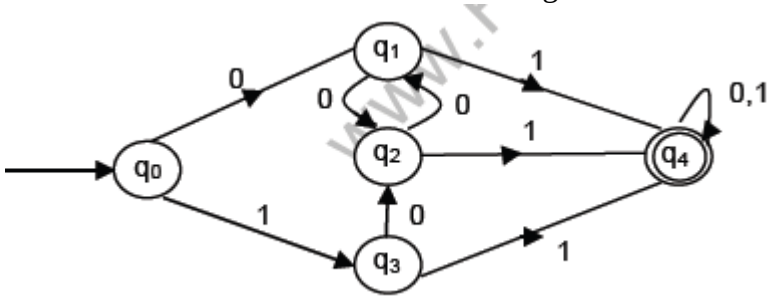
33. Construct Minimum state Automata for the following DFA

	0	1
$\rightarrow A$	B	A
B	A	C
C	D	B
$*D$	D	A
E	D	F
F	G	E
G	F	G
H	G	D

34. Construct Minimum state Automata for the following DFA

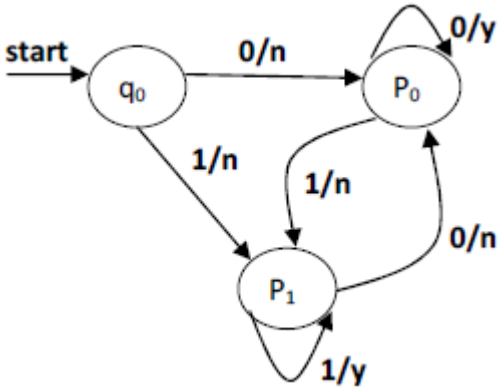
	0	1
$\rightarrow A$	B	E
B	C	F
$*C$	D	H
D	E	H
E	F	I
$*F$	G	B
G	H	B
H	I	C
$*I$	A	E

35. Minimize the finite automaton shown in figure below.



36. Bring out the differences between Moore and Mealy machines?
37. Design a Moore machine with the input alphabet {0, 1} and output alphabet {Y, N} which produces Y as output if input sequence contains 1010 as a substring otherwise, it produces N as output.

38. Design a mealy machine to print out 1's complement of an input bit string?
39. Design a Moore machine for 2's complement of binary number.
40. Construct the Moore machine to compute residue modulo 5 and finds to its equivalent Mealy machine.
41. Draw a Moore machine for calculating mod 3 of a given binary number. Find its equivalent mealy machine.
42. Construct a Moore machine that determines whether an input string contains an even or odd number of 1's. The machine should give 1 as output if an even number of 1's is in the string and 0 otherwise.
43. Design a Moore and Mealy machines for a binary input sequence such that if it has a substring 101, the machine outputs A, if the input has substring 110, it outputs B otherwise it outputs C.
44. Design a Moore and Mealy machine that scans sequence of input of 0 and 1 and generates output 'A' if the input string terminates in 00, output 'B' if the string terminates in 11, and output 'C' otherwise.
45. Design Moore and Mealy Machine to increment binary number by 1
46. Give Mealy and Moore machine for the following: For input from ∑*, where ∑= {0,1,2}, print the residue modulo 5 of the input treated as a ternary (base 3, with digits 0, 1 and 2) number.
47. Design a Moore Machine that will read sequences made up of letters a, e, i, o, u and will give as output, same characters except when an 'i' is followed by 'e', it will be changed to 'u'.
48. Design a Mealy machine to add two binary numbers of the form x1x2…xk, y1y2…yk?
49. Convert the following Mealy machine to an equivalent Moore machine



50. Convert the following Mealy machine into equivalent Moore machine.