C0531: FORMAL LANGUAGES AND AUTOMATA THEORY

III Year B.Tech. CSE II-Sem

Course Objectives

1. To provide introduction to some of the central ideas of theoretical computer science from the perspective of formal languages.
2. To introduce the fundamental concepts of formal languages, grammars and automata theory.
3. Classify machines by their power to recognize languages.
4. Employ finite state machines to solve problems in computing.
5. To understand deterministic and non-deterministic machines.
6. To understand the differences between decidability and undecidability.

Course Outcomes

1. Able to understand the concept of abstract machines and their power to recognize the languages.
2. Able to employ finite state machines for modeling and solving computing problems.
3. Able to design context free grammars for formal languages.
4. Able to distinguish between decidability and undecidability.

UNIT - I

Introduction to Finite Automata: Structural Representations, Automata and Complexity, the Central Concepts of Automata Theory – Alphabets, Strings, Languages, 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

UNIT - II

Regular Expressions: Finite Automata and Regular Expressions, Applications of Regular Expressions, Algebraic Laws for Regular Expressions, Conversion of Finite Automata to Regular Expressions.

Pumping Lemma for Regular Languages, Statement of the pumping lemma, Applications of the Pumping Lemma.

Closure Properties of Regular Languages: Closure properties of Regular languages, Decision Properties of Regular Languages, Equivalence and Minimization of Automata.

UNIT - III

Context-Free Grammars: Definition of Context-Free Grammars, Derivations Using a Grammar, Leftmost and Rightmost Derivations, the Language of a Grammar, Sentential Forms, Parse Tress, Applications of Context-Free Grammars, Ambiguity in Grammars and Languages. **Push Down Automata**: Definition of the Pushdown Automaton, the Languages of a PDA, Equivalence of PDA's and CFG's, Acceptance by final state, Acceptance by empty stack, Deterministic Pushdown Automata. From CFG to PDA, From PDA to CFG

UNIT - IV

Normal Forms for Context- Free Grammars: Eliminating useless symbols, Eliminating €-Productions. Chomsky Normal form Griebech Normal form.

Pumping Lemma for Context-Free Languages: Statement of pumping lemma, Applications
Closure Properties of Context-Free Languages: Closure properties of CFL's, Decision Properties of CFL's

Turing Machines: Introduction to Turing Machine, Formal Description, Instantaneous description, The language of a Turing machine

UNIT - V

Types of Turing machine: Turing machines and halting

Undecidability: Undecidability, A Language that is Not Recursively Enumerable, An Undecidable Problem That is RE, Undecidable Problems about Turing Machines, Recursive languages, Properties of recursive languages, Post's Correspondence Problem, Modified Post Correspondence problem, Other Undecidable Problems, Counter machines.

TEXT BOOKS:

1. Introduction to Automata Theory, Languages, and Computation, 3nd Edition, John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Pearson Education.
2. Theory of Computer Science – Automata languages and computation, Mishra and Chandrashekaran, 2nd edition, PHI.

REFERENCE BOOKS:

1. Introduction to Languages and The Theory of Computation, John C Martin, TMH.
2. Introduction to Computer Theory, Daniel I.A. Cohen, John Wiley.
3. A Text book on Automata Theory, P. K. Srimani, Nasir S. F. B, Cambridge University Press.
4. Introduction to the Theory of Computation, Michael Sipser, 3rd edition, Cengage Learning.
5. Introduction to Formal languages Automata Theory and Computation Kamala Krithivasan, Rama R, Pearson.

Formal Languages and Automata Theory

UNIT-I

CONTENTS

INTRODUCTION TO FINITE AUTOMATA

Automata – What is it?

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

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 $(Q, \sum, \delta, q_0, F)$, where −

- **Q** is a finite set of states.
- $\sum$ is a finite set of symbols, called the **alphabet** of the automaton.
- **δ** is the transition function.
- $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Structural Representations

The purpose of a logical framework such as LF is to provide a language for defining logical systems suitable for use in a logic-independent proof development environment. All inferential activity in an object logic (in particular, proof search) is to be conducted in the logical framework via the representation of that logic in the framework. An important tool for controlling search in an object logic, the need for which is motivated by the difficulty of reasoning about large and complex systems, is the use of structured theory presentations. In this paper a rudimentary language of structured theory presentations is presented, and the use of this structure in proof search for an arbitrary object logic is explored. The behaviour of structured theory presentations under representation in a logical framework is studied, focusing on the problem of "lifting" presentations from the object logic to the metalogic of the framework. The topic of imposing structure on logic presentations, so that logical systems may themselves be defined in a modular fashion, is also briefly consider.

Automata & Complexity

Automata Theory

- Finite automata: Computers with no memory

  Motivation: Numbers, names, in Prog. Languages

  Example: $x = -0.0565$
- Context-free grammars: Memory = stack

  Motivation: Syntax, grammar of Prog. Languages

  Example: *if (...) then (...) else (...)*

  Computability Theory
- Turing Machines: "Compute until the sun dies"
- Motivation: What problems can be solved at all?
- Example: Given a program, does it have a bug?

  We will prove impossible to determine!

Complexity Theory

- P, NP: Model your laptop
- Motivation: What problems can be solved fast?

● Example: Given a formula with 1000 variables like

(X OR Y) AND (X OR NOT Z) AND (Z OR Y) …

Is it satisfiable or not?

Does it take 1 thousand years or 1 second to know?

Recap

● Automata theory: Finite automata, grammars

● Computability Theory: Turing Machines

● Complexity Theory: P, NP

● Theme: Computation has many guises:

Automata, grammar, Turing Machine, formula ...

The Central Concepts Of Automata Theory-

Alphabets , Strings , Languages, Problems

Alphabet

- **Definition** − An **alphabet** is any finite set of symbols.
- **Example** − ∑ = {a, b, c, d} is an **alphabet set** where 'a', 'b', 'c', and 'd' are **symbols**.
- An alphabet is a finite, non-empty set of symbols We use the symbol ∑ (sigma) to denote an alphabet Examples: Binary: ∑ = {0,1}¬ All lower case letters: ∑ = {a,b,c,..z}¬
- Alphanumeric: ∑ = {a-z, A-Z, 0-9}

String

- **Definition** − A **string** is a finite sequence of symbols taken from ∑.
- **Example** − 'cabcad' is a valid string on the alphabet set ∑ = {a, b, c, d}

A string or word is a finite sequence of symbols chosen from ∑ (epsilon)εEmpty string is donated by λ (lambda).

Length of a String

It is the number of meaningful) present in aεsymbols/alphabets (non- string.

Length of a string w, denoted by "|w|".

E.g., if x = 010100 then |x| = 6 then|x| = ?ε 00 ε 1 ε 0 ε– If x = 01 If |W|= 0, it is called an empty string (Denoted by λ or ε) xy = concatenation of two strings x and y

Powers of an alphabet

Let ∑ be an alphabet. ∑k = the set of all strings of length k

∑* = ∑0 U ∑1 U ∑2 U …□ ∑+ = ∑1 U ∑2 U ∑3 U …

- **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 Closure / Star

- **Definition** − The Kleene star, **∑***, is a unary operator on a set of symbols or strings, ∑, that gives the infinite set of all possible strings of all possible lengths over ∑ including **λ**.
- **Representation** − $\sum^* = \sum_0 \cup \sum_1 \cup \sum_2 \cup \ldots\ldots$ where $\sum_p$ is the set of all possible strings of length p.
- **Example** − If ∑ = {a, b}, ∑* = {λ, a, b, aa, ab, ba, bb,……..}

Kleene Closure Plus

- **Definition** − The set $\sum^+$ is the infinite set of all possible strings of all possible lengths over ∑ excluding λ.
- **Representation** − $\sum^+ = \sum_1 \cup \sum_2 \cup \sum_3 \cup \ldots\ldots$

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

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

Language

- **Definition** − A language is a subset of ∑* for some alphabet ∑. It can be finite or infinite.

Example –

1. If the language takes all possible strings of length 2 over ∑ = {a, b}, then L = { ab, bb, ba, bb}

2. Let L be the language of all strings consisting of n 0's followed by n 1's: ,01,0011,000111,…}εL = { 2. Let L be the language of all strings of with equal number of 0's and 1's: ,01,10,0011,1100,0101,1010,1001,…}εL = { 3. If the language takes all possible strings of length 2 over ∑ = {a, b}, then L = { ab, bb, ba, bb} Ø denotes the Empty language }; Is L=Ø?ε•

The Membership Problem ∑*and a language L over ∑,∈Given a string w L.∈decide whether or not w Example: Let w = 100011 the language of strings with equal∈Q) Is w number of 0s and 1s?

Operations on languages: –Union: L∪M= {w| w ∈ Lor w∈M}.
–Intersection: L∩M={w |w ∈ L and w∈ M}.
–Subtraction:L−M={w|w∈Landw /∈M}.
–Complementation:L={w|w∈Σ∗−L}.
–Concatenation:LM={w|w∈Landx/∈M}.
Note: if L and M are over different alphabets, say Σ1 and Σ2, then the resulting
language can be taken to be over Σ1∪Σ2. In this course, it will not happen.
Notation: we will write LM instead of L ·M .
Also, concatenation is associative (i.e.L(MN) = (LM)N), so we drop the brackets and write LMN.
Note: for any language L ,Ø L=LØ=Ø.
And also {ε}L=L{ε}=L.

– Lk = LL . . .L| {z }k times for k > 0; L0 = {"}.
Exercise: Prove that for any language L, and any m _ 0, n _ 0, LmLn = Lm+n.
– Kleene closure (or star closure): L_ = L0 [L1 [L2 [. . . .

– Reverse: LR = {wR|w 2 L}.

NON DETERMINISTIC FINITE AUTOMATA:

Finite Automaton can be classified into two types −

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

Non deterministic finite automata(NFA)

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

An NDFA is represented by digraphs called state diagram.

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

Formal Definition of an NDFA

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

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

  (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: (same as DFA)

- An NFA is a five-tuple: $M = (Q, \Sigma, \delta, q_0, F)$

Q	A finite set of states
Σ	A finite input alphabet
q_0	The initial/starting state, q_0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, which is a total function from Q x Σ to 2^Q

$\delta: (Q \times \Sigma) \rightarrow \mathbf{2^Q}$ -2^Q is the power set of Q, the set of all subsets of Q $\delta(q,s)$ -The set of all states p such that there is a transition

l

abeled s from q to p $\delta(q,s)$ is a

function from Q x S to 2^Q (but

not to Q)

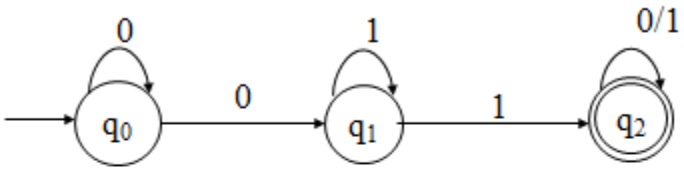
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let w be in Σ^*. Then w is *accepted* by M iff $\delta(\{q_0\}, w)$ contains at least one state in F.

- Let M = (Q, Σ, δ,q_0,F) be an NFA. Then the *language accepted* by M is the set: L(M) = {w | w is in Σ* and δ({q_0},w) contains at least one state in F}

- Another equivalent definition:
  L(M) = {w | w is in Σ* and w is accepted by M}

- Example: some 0's followed by some 1's

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{0, 1\}$
Start state is q_0
$F = \{q_2\}$



δ:

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

Example

An Application - Text Search

Automata theory is the study of abstract machines and automata. It also includes the computational problems that can be solved using them . In the theory of computation,the simpler abstract machine is finite automata. The otherimportant abstract machines are 1. Pushdown Automata2.Turing Machine. The finite automata proposed to model brain function of the human. The simplest example for finite automata is the switch with two states "on" and "off" [1].The Finite Automata is the five tuples combination focusing on states and transition through input characters. Infigure 1 the ending state is ON, starting state is OFF and collection of states are OFF and ON. It is having only a single input PUSH for making the transition from the state OFF to ON, then ON to OFF. The switch is the simplest practical application of finite automata.

Acceptors, Classifiers, and Transducers

Acceptor (Recognizer)

An automaton that computes a Boolean function is called an **acceptor**. All the states of an acceptor is either accepting or rejecting the inputs given to it.

Classifier

A **classifier** has more than two final states and it gives a single output when it terminates.

Transducer

An automaton that produces outputs based on current input and/or previous state is called a **transducer**. Transducers can be of two types −

- **Mealy Machine** − The output depends both on the current state and the current input.
- **Moore Machine** − The output depends only on the current state.

Finite automata With Epsilon- Transition

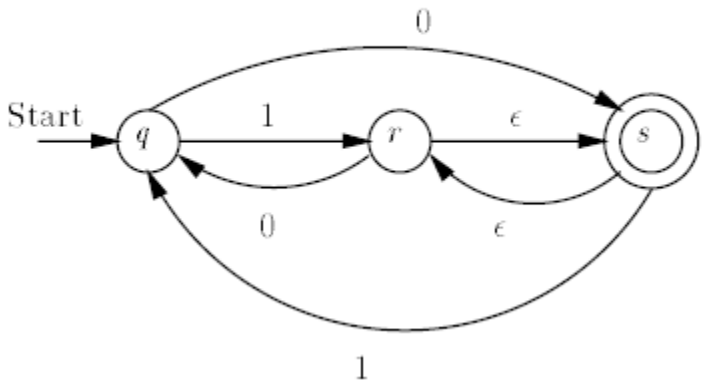
Allow to be a label on arcs.

Nothing else changes: acceptance of w is still the existence of a path from the start state to an accepting state with label w.

✦ But can appear on arcs, and means the

empty string (i.e., no visible contribution to w).

Example



001 is accepted by the path q; s; r; q; r; s, with
label 0 01 = 001.

DETERMINISTIC FINITE AUTOMATA :

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

Formal Definition of a DFA

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

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

Graphical Representation of a DFA

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

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

Example

Let a deterministic finite automaton be →

- Q = {a, b, c},
- ∑ = {0, 1},
- q_0 = {a},
- F = {c}, and

Transition function δ as shown by the following table –

Present state	next state for input 0	next state for input 1
A	A	B
B	C	A
C	B	C

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.

How A DFA Processing Strings

Acceptors, Classifiers, and Transducers

Acceptor (Recognizer)

An automaton that computes a Boolean function is called an **acceptor**. All the states of an acceptor is either accepting or rejecting the inputs given to it.

Classifier

A **classifier** has more than two final states and it gives a single output when it terminates.

Transducer

An automaton that produces outputs based on current input and/or previous state is called a **transducer**. Transducers can be of two types −

- **Mealy Machine** − The output depends both on the current state and the current input.
- **Moore Machine** − The output depends only on the current state.

Acceptability by DFA and NDFA

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 string wholly.

A string S is accepted by a DFA/NDFA $(Q, \sum, \delta, q_0, F)$, iff

$\delta^*(q_0, S) \in F$

The language **L** accepted by DFA/NDFA is

$\{S \mid S \in \sum^* \text{ and } \delta^*(q_0, S) \in F\}$

A string S′ is not accepted by a DFA/NDFA $(Q, \sum, \delta, q_0, F)$, iff

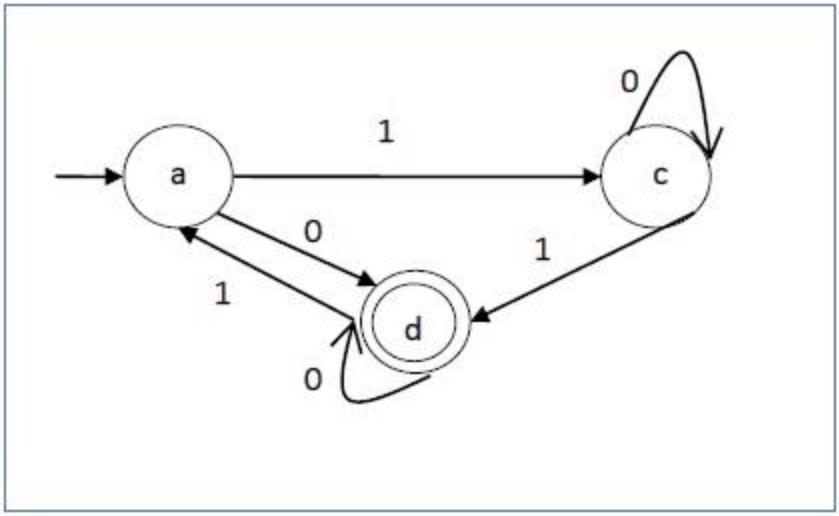
$\delta^*(q_0, S') \notin F$

The language L′ not accepted by DFA/NDFA (Complement of accepted language L) is

$\{S \mid S \in \sum^* \text{ and } \delta^*(q_0, S) \notin F\}$

Example

Let us consider the DFA shown in Figure 1.3. From the DFA, the acceptable strings can be derived.



Strings accepted by the above DFA: {0, 00, 11, 010, 101, ...}

Strings not accepted by the above DFA: {1, 011, 111, ...}

The Language of DFA

- A DFA is a five-tuple: $M = (Q, \Sigma, \delta, q_0, F)$

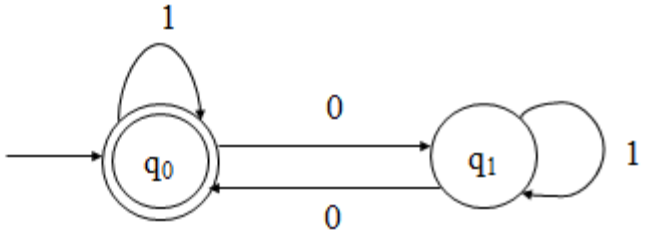
Q	A finite set of states
Σ	A finite input alphabet
q_0	The initial/starting state, q_0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, which is a total function from Q x Σ to Q

$\delta: (Q \times \Sigma) \rightarrow Q$ δ is defined for any q in Q and s in Σ, and $\delta(q,s) = q'$ is equal to another state q' in Q.

Intuitively, $\delta(q,s)$ is the state entered by M after reading symbol s while in state q.

- For Example #1:

$Q = \{q_0, q_1\}$
$\Sigma = \{0, 1\}$
Start state is q_0
$F = \{q_0\}$



δ:

	0	1
q_0	q_1	q_0
q_1	q_0	q_1

- Let M = (Q, Σ, δ, q_0, F) be a DFA and let w be in Σ*. Then w is *accepted* by M iff

$\delta(q_0, w) = p$ for some state p in F.

- Let M = (Q, Σ, δ, q_0, F) be a DFA. Then the *language accepted* by M is the set:

$$L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } \delta(q_0, w) \text{ is in } F\}$$

- Another equivalent definition:

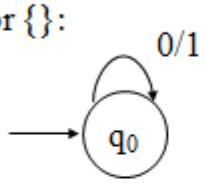
$$L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } w \text{ is accepted by } M\}$$

- Let L be a language. Then L is a *regular language* iff there exists a DFA M such that L = L(M).

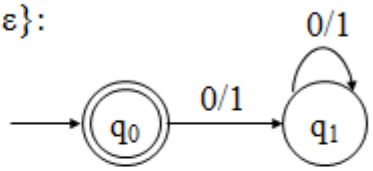
- Notes:
  - A DFA M = (Q, Σ, δ,q_0,F) partitions the set Σ* into two sets: L(M) and Σ* - L(M).
  - If L = L(M) then L is a subset of L(M) and L(M) is a subset of L.
  - Similarly, if $L(M_1) = L(M_2)$ then $L(M_1)$ is a subset of $L(M_2)$ and $L(M_2)$ is a subset of $L(M_1)$.

- Let $\Sigma = \{0, 1\}$. Give DFAs for $\{\}$, $\{\varepsilon\}$, Σ^*, and Σ^+.

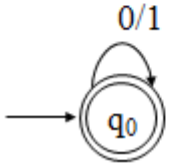
For $\{\}$:



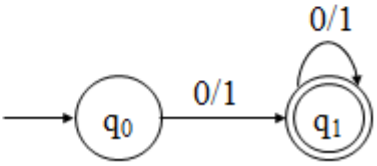
For $\{\varepsilon\}$:



For Σ^*:

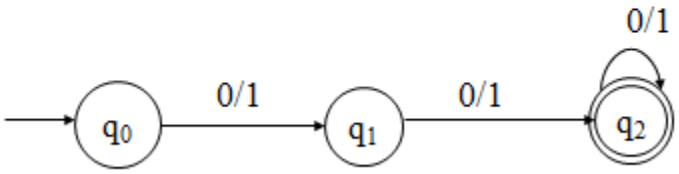


For Σ^+:

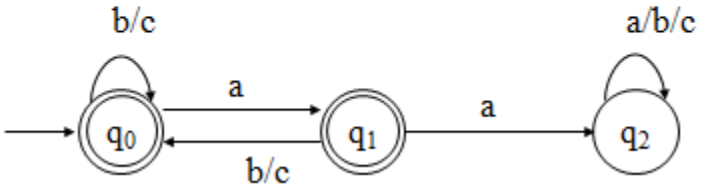


- Give a DFA M such that:

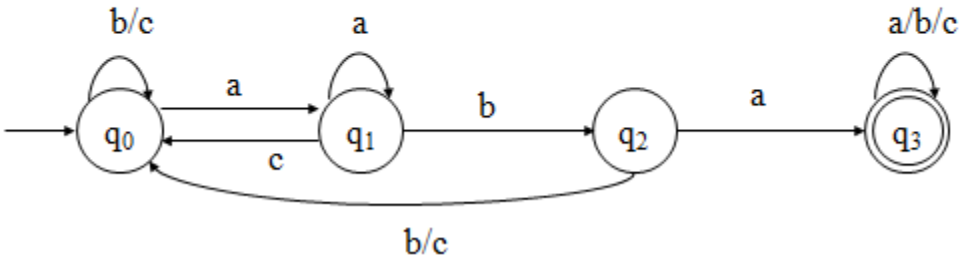
L(M) = {x | x is a string of 0's and 1's and |x| >= 2}



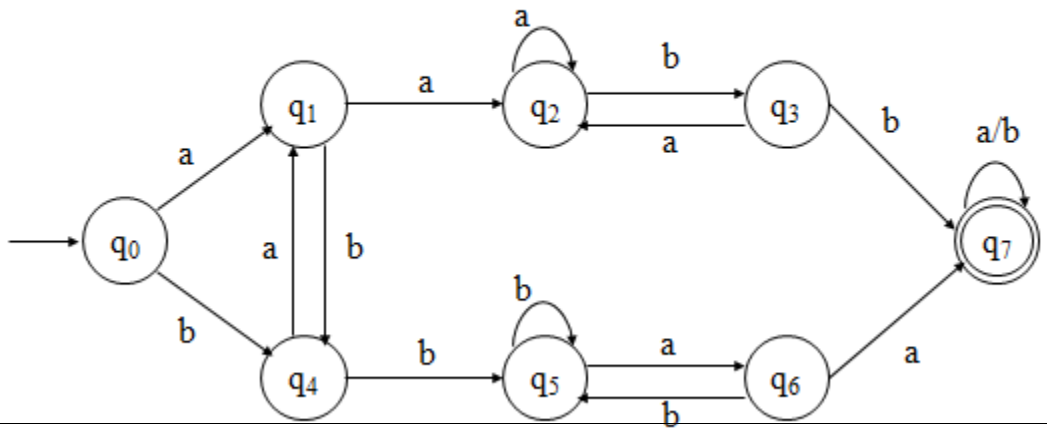
L(M) = {x | x is a string of (zero or more) a's, b's and c's such that x does not contain the substring *aa*}



L(M) = {x | x is a string of a's, b's and c's such that x contains the substring *aba*}



L(M) = {x | x is a string of a's and b's such that x contains both *aa* and *bb*}



Convertion of NFA with Epsilon Transition to NFA without

Epsilon Transition

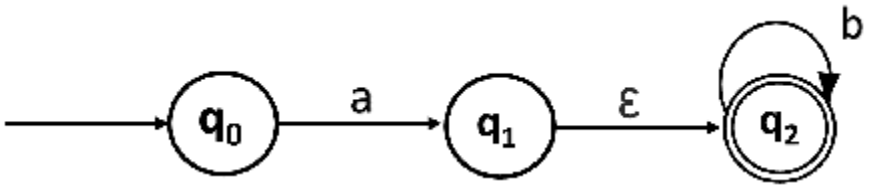
Eliminating ε Transitions

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

1. Find out all the ε transitions from each state from Q. That will be called as ε-closure{q1} where qi ∈ Q.
2. Then δ' transitions can be obtained. The δ' transitions mean a ε-closure on δ moves.
3. Repeat Step-2 for each input symbol and each state of given NFA.
4. Using the resultant states, the transition table for equivalent NFA without ε can be built.

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(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}

Now we will summarize all the computed δ' transitions:

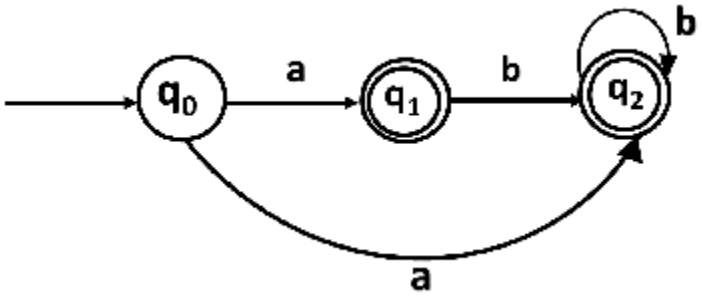
δ'(q0, a) = {q0, q1}
δ'(q0, b) = Φ
δ'(q1, a) = Φ

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:



Conversion from NFA to DFA

In this section, we will discuss the method of converting NFA to its equivalent DFA. In NFA, when a specific input is given to the current state, the machine goes to multiple states. It can have zero, one or more than one move on a given input symbol. On the other hand, in DFA, when a specific input is given to the current state, the machine goes to only one state. DFA has only one move on a given input symbol.

Let, M = (Q, Σ, δ, q0, F) is an NFA which accepts the language L(M). There should be equivalent DFA denoted by M' = (Q', Σ', q0', δ', F') such that L(M) = L(M').

Steps for converting NFA to DFA:

Step 1: Initially Q' = φ

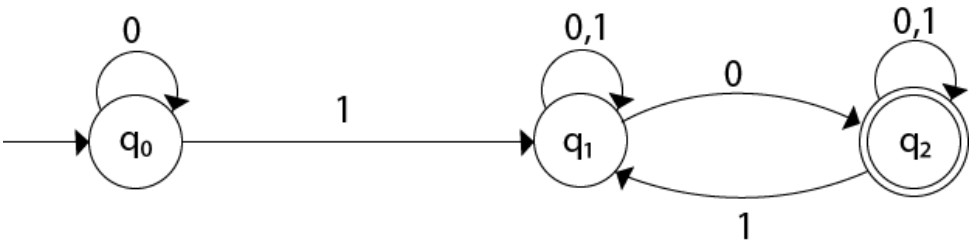
Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

Step 3: In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.

Step 4: In DFA, the final state will be all the states which contain F(final states of NFA)

Example 1:

Convert the NFA to DFA.



Solution: For the given transition diagram we will first construct the transition table.

State	0	1
→q0	q0	q1

q1	{q1, q2}	q1
*q2	q2	{q1, q2}

Now we will obtain δ' transition for state q0.

δ'([q0], 0) = [q0]
δ'([q0], 1) = [q1]

The δ' transition for state q1 is obtained as:

δ'([q1], 0) = [q1, q2] (**new** state generated)
δ'([q1], 1) = [q1]

The δ' transition for state q2 is obtained as:

δ'([q2], 0) = [q2]
δ'([q2], 1) = [q1, q2]

Now we will obtain δ' transition on [q1, q2].

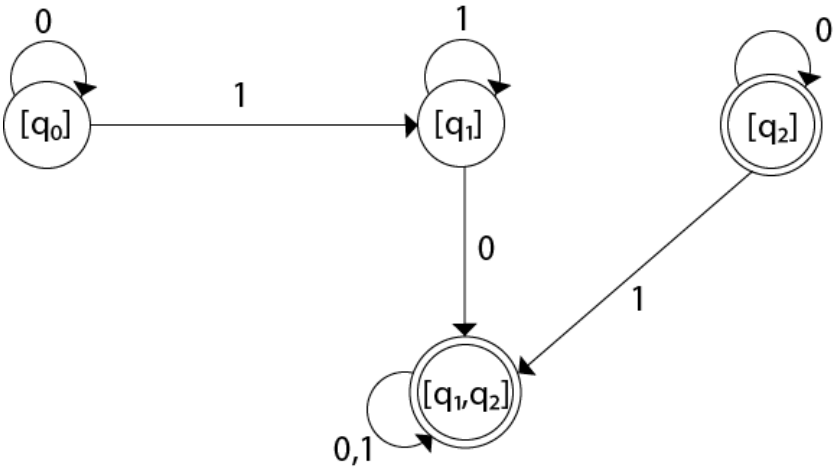
$$\delta'([q1, q2], 0) = \delta(q1, 0) \cup \delta(q2, 0)$$
$$= \{q1, q2\} \cup \{q2\}$$
$$= [q1, q2]$$
$$\delta'([q1, q2], 1) = \delta(q1, 1) \cup \delta(q2, 1)$$
$$= \{q1\} \cup \{q1, q2\}$$
$$= \{q1, q2\}$$
$$= [q1, q2]$$

The state [q1, q2] is the final state as well because it contains a final state q2. The transition table for the constructed DFA will be:

State	0	1
→[q0]	[q0]	[q1]

[q1]	[q1, q2]	[q1]
*[q2]	[q2]	[q1, q2]
*[q1, q2]	[q1, q2]	[q1, q2]

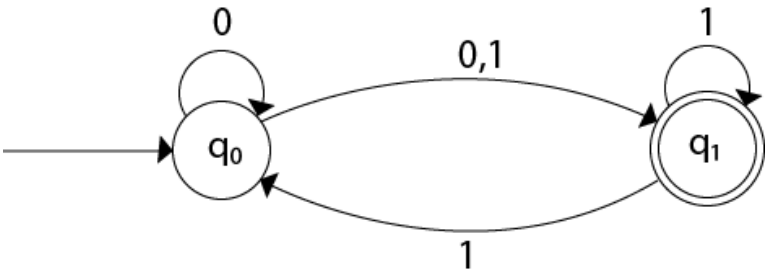
The Transition diagram will be:



The state q2 can be eliminated because q2 is an unreachable state.

Example 2:

Convert the given NFA to DFA.



Solution: For the given transition diagram we will first construct the transition table.

State	0	1
→q0	{q0, q1}	{q1}
*q1	φ	{q0, q1}

Now we will obtain δ' transition for state q0.

δ'([q0], 0) = {q0, q1}
= [q0, q1] (**new** state generated)
δ'([q0], 1) = {q1} = [q1]

The δ' transition for state q1 is obtained as:

δ'([q1], 0) = φ
δ'([q1], 1) = [q0, q1]

Now we will obtain δ' transition on [q0, q1].

δ'([q0, q1], 0) = δ(q0, 0) ∪ δ(q1, 0)
= {q0, q1} ∪ φ
= {q0, q1}
= [q0, q1]

Similarly,

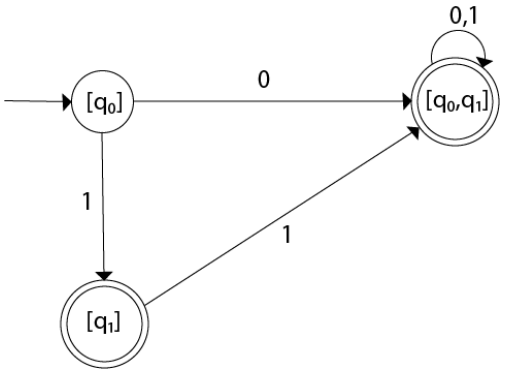
δ'([q0, q1], 1) = δ(q0, 1) ∪ δ(q1, 1)
= {q1} ∪ {q0, q1}
= {q0, q1}
= [q0, q1]

As in the given NFA, q1 is a final state, then in DFA wherever, q1 exists that state becomes a final state. Hence in the DFA, final states are [q1] and [q0, q1]. Therefore set of final states F = {[q1], [q0, q1]}.

The transition table for the constructed DFA will be:

State	0	1
→[q0]	[q0, q1]	[q1]
*[q1]	φ	[q0, q1]
*[q0, q1]	[q0, q1]	[q0, q1]

The Transition diagram will be:

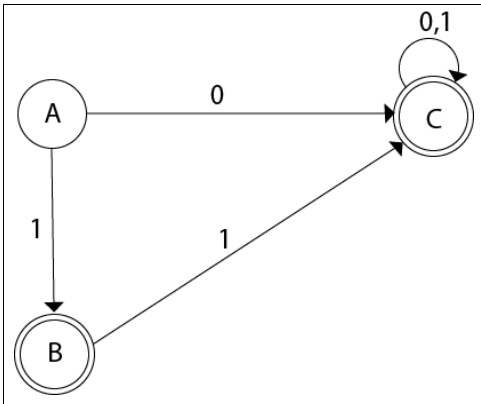


Even we can change the name of the states of DFA.

Suppose

A = [q0]
B = [q1]
C = [q0, q1]

With these new names the DFA will be as follows:



Conversion from NFA with ε to DFA

Non-deterministic finite automata(NFA) is a finite automata where for some cases when a specific input is given to the current state, the machine goes to multiple states or more than 1 states. It can contain ε move. It can be represented as M = { Q, Σ, δ, q0, F}.

Where

Q: finite set of states

Σ: finite set of the input symbol

q0: initial state

F: **final** state

δ: Transition function

NFA with ∈ move: If any FA contains ε transaction or move, the finite automata is called NFA with ∈ move.

ε-closure: ε-closure for a given state A means a set of states which can be reached from the state A with only ε(null) move including the state A itself.

Steps for converting NFA with ε to DFA:

Step 1: We will take the ε-closure for the starting state of NFA as a starting state of DFA.

Step 2: Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closures for each state of NFA present in the current state of DFA.

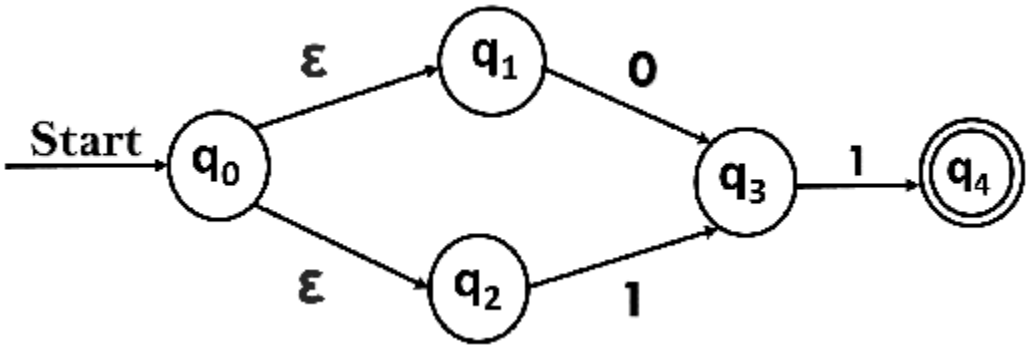
Step 3: If we found a new state, take it as current state and repeat step 2.

Step 4: Repeat Step 2 and Step 3 until there is no new state present in the transition table of DFA.

Step 5: Mark the states of DFA as a final state which contains the final state of NFA.

Example 1:

Convert the NFA with ε into its equivalent DFA.



Solution:

Let us obtain ε-closure of each state.

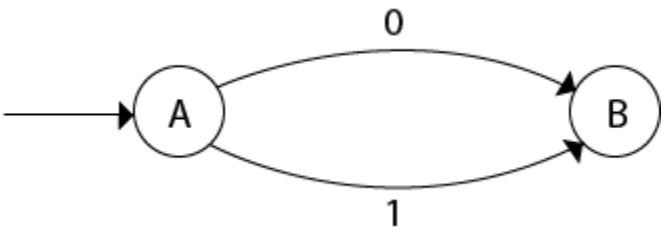
ε-closure {q0} = {q0, q1, q2}
ε-closure {q1} = {q1}
ε-closure {q2} = {q2}
ε-closure {q3} = {q3}
ε-closure {q4} = {q4}

Now, let ε-closure {q0} = {q0, q1, q2} be state A.

Hence

```
δ'(A, 0) = ε-closure {δ((q0, q1, q2), 0) }
         = ε-closure {δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0) }
         = ε-closure {q3}
         = {q3}            call it as state B.

δ'(A, 1) = ε-closure {δ((q0, q1, q2), 1) }
         = ε-closure {δ((q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1) }
         = ε-closure {q3}
         = {q3} = B.
```

The partial DFA will be



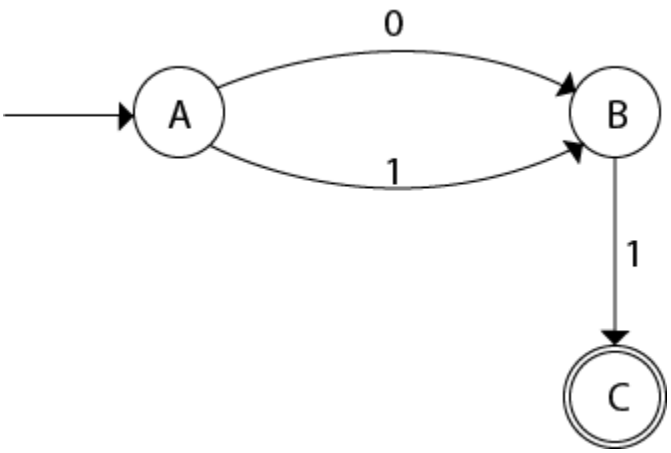
Now,

```
δ'(B, 0) = ε-closure {δ(q3, 0) }
         = ϕ
δ'(B, 1) = ε-closure {δ(q3, 1) }
         = ε-closure {q4}
         = {q4}            i.e. state C
```

For state C:

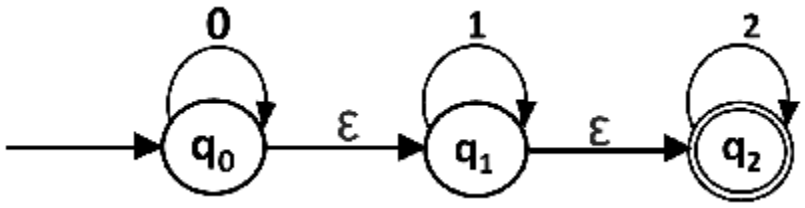
δ'(C, 0) = ε-closure {δ(q4, 0) }
= φ
δ'(C, 1) = ε-closure {δ(q4, 1) }
= φ

The DFA will be,



Example 2:

Convert the given NFA into its equivalent DFA.



Solution: Let us obtain the ε-closure of each state.

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

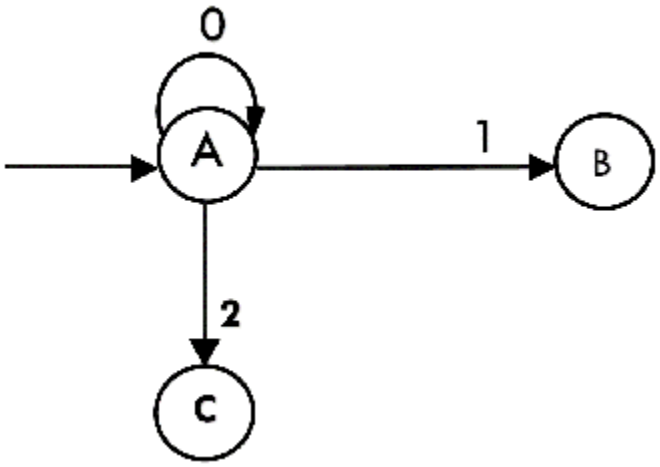
Now we will obtain δ' transition. Let ε-closure(q0) = {q0, q1, q2} call it as **state A**.

```
δ'(A, 0) = ε-closure{δ((q0, q1, q2), 0)}
         = ε-closure{δ(q0, 0) ∪ δ(q1, 0) ∪ δ(q2, 0)}
         = ε-closure{q0}
         = {q0, q1, q2}

δ'(A, 1) = ε-closure{δ((q0, q1, q2), 1)}
         = ε-closure{δ(q0, 1) ∪ δ(q1, 1) ∪ δ(q2, 1)}
         = ε-closure{q1}
         = {q1, q2}          call it as state B

δ'(A, 2) = ε-closure{δ((q0, q1, q2), 2)}
         = ε-closure{δ(q0, 2) ∪ δ(q1, 2) ∪ δ(q2, 2)}
         = ε-closure{q2}
         = {q2}          call it state C
```

Thus we have obtained

δ'(A, 0) = A
δ'(A, 1) = B
δ'(A, 2) = C

The partial DFA will be:



Now we will find the transitions on states B and C for each input.

Hence

$$\delta'(B, 0) = \varepsilon\text{-closure}\{\delta((q1, q2), 0)\}$$
$$= \varepsilon\text{-closure}\{\delta(q1, 0) \cup \delta(q2, 0)\}$$
$$= \varepsilon\text{-closure}\{\phi\}$$
$$= \phi$$

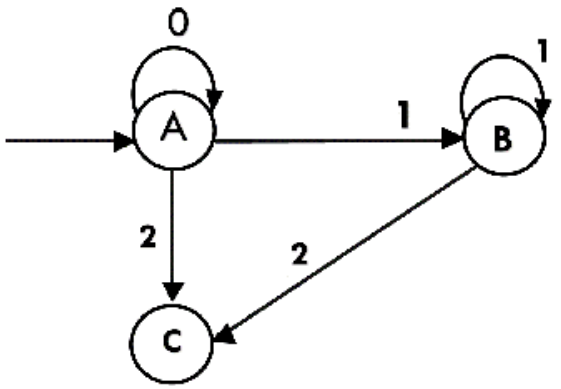
$$\delta'(B, 1) = \varepsilon\text{-closure}\{\delta((q1, q2), 1)\}$$
$$= \varepsilon\text{-closure}\{\delta(q1, 1) \cup \delta(q2, 1)\}$$
$$= \varepsilon\text{-closure}\{q1\}$$
$$= \{q1, q2\}$$ **i.e. state B itself**

$$\delta'(B, 2) = \varepsilon\text{-closure}\{\delta((q1, q2), 2)\}$$
$$= \varepsilon\text{-closure}\{\delta(q1, 2) \cup \delta(q2, 2)\}$$
$$= \varepsilon\text{-closure}\{q2\}$$
$$= \{q2\}$$ **i.e. state C itself**

Thus we have obtained

$\delta'(B, 0) = \varphi$
$\delta'(B, 1) = B$
$\delta'(B, 2) = C$

The partial transition diagram will be



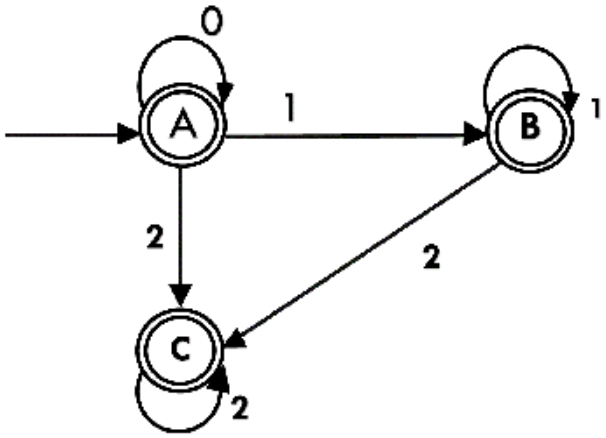
Now we will obtain transitions for C:

$$\delta'(C, 0) = \varepsilon\text{-closure}\{\delta(q2, 0)\}$$
$$= \varepsilon\text{-closure}\{\phi\}$$
$$= \phi$$

$$\delta'(C, 1) = \varepsilon\text{-closure}\{\delta(q2, 1)\}$$
$$= \varepsilon\text{-closure}\{\phi\}$$
$$= \phi$$

$$\delta'(C, 2) = \varepsilon\text{-closure}\{\delta(q2, 2)\}$$
$$= \{q2\}$$

Hence the DFA is



As A = {q0, q1, q2} in which final state q2 lies hence A is final state. B = {q1, q2} in which the state q2 lies hence B is also final state. C = {q2}, the state q2 lies hence C is also a final state.

FINITE AUTOMATA WITH OUTPUT

Moore and Melay machins

Moore Machine

Moore machine is a finite state machine in which the next state is decided by the current state and current input symbol. The output symbol at a given time depends only on the present state of the machine. Moore machine can be described by 6 tuples (Q, q0, Σ, O, δ, λ) where,

Q: finite set of states

q0: initial state of machine

Σ: finite set of input symbols

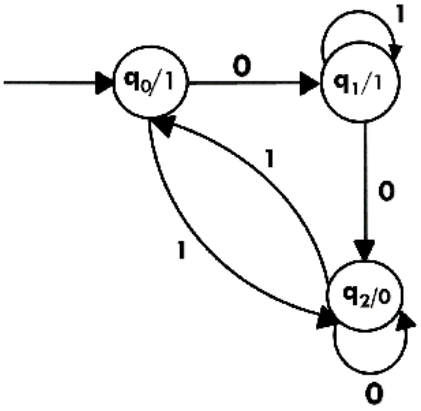
O: output alphabet

δ: transition function where $Q \times \Sigma \rightarrow Q$

λ: output function where $Q \rightarrow O$

Example 1:

The state diagram for Moore Machine is



Transition table for Moore Machine is:

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

In the above Moore machine, the output is represented with each input state separated by /. The output length for a Moore machine is greater than input by 1.

Input: 010

Transition: δ (q0,0) => δ(q1,1) => δ(q1,0) => q2

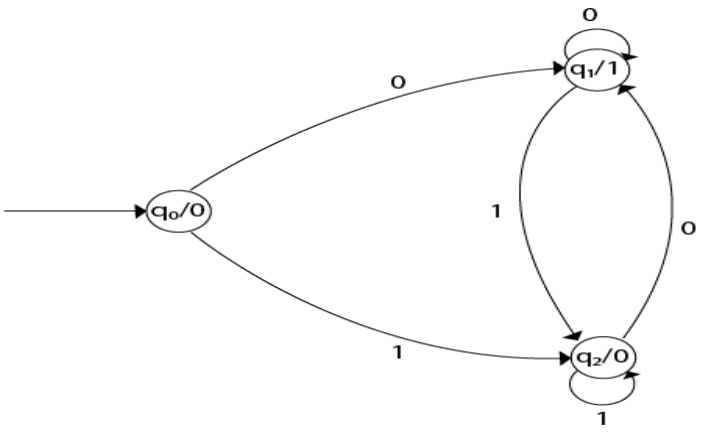
Output: 1110(1 for q0, 1 for q1, again 1 for q1, 0 for q2)

Example 2:

Design a Moore machine to generate 1's complement of a given binary number.

Solution: To generate 1's complement of a given binary number the simple logic is that if the input is 0 then the output will be 1 and if the input is 1 then the output will be 0. That means there are three states. One state is start state. The second state is for taking 0's as input and produces output as 1. The third state is for taking 1's as input and producing output as 0.

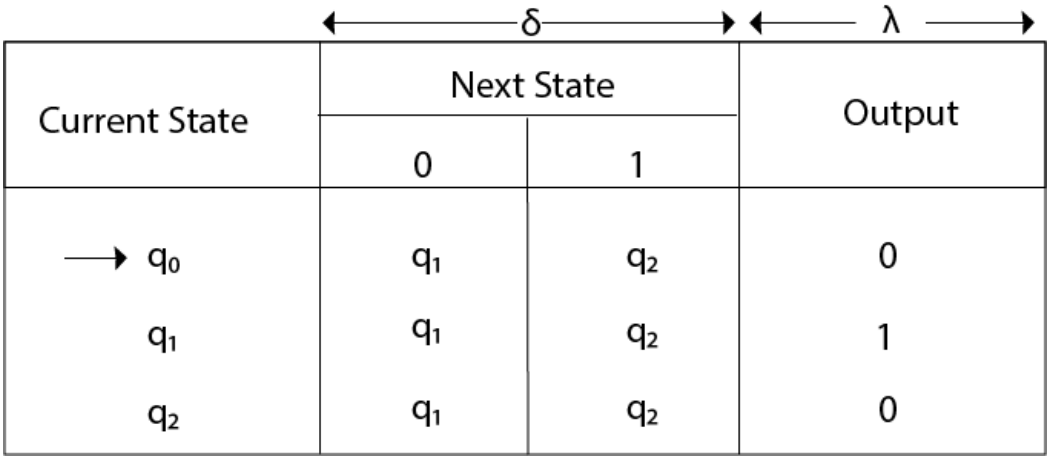
Hence the Moore machine will be,



For instance, take one binary number 1011 then

Input		1	0	1	1
State	q0	q2	q1	q2	q2
Output	0	0	1	0	0

Thus we get 00100 as 1's complement of 1011, we can neglect the initial 0 and the output which we get is 0100 which is 1's complement of 1011. The transaction table is as follows:



Thus Moore machine M = (Q, q0, Σ, O, δ, λ); where Q = {q0, q1, q2}, Σ = {0, 1}, O = {0, 1}. the transition table shows the δ and λ functions.

Mealy Machine

A Mealy machine is a machine in which output symbol depends upon the present input symbol and present state of the machine. In the Mealy machine, the output is represented with each input symbol for each state separated by /. The Mealy machine can be described by 6 tuples (Q, q0, Σ, O, δ, λ') where

Q: finite set of states

q0: initial state of machine

Σ: finite set of input alphabet

O: output alphabet

δ: transition function where Q × Σ → Q

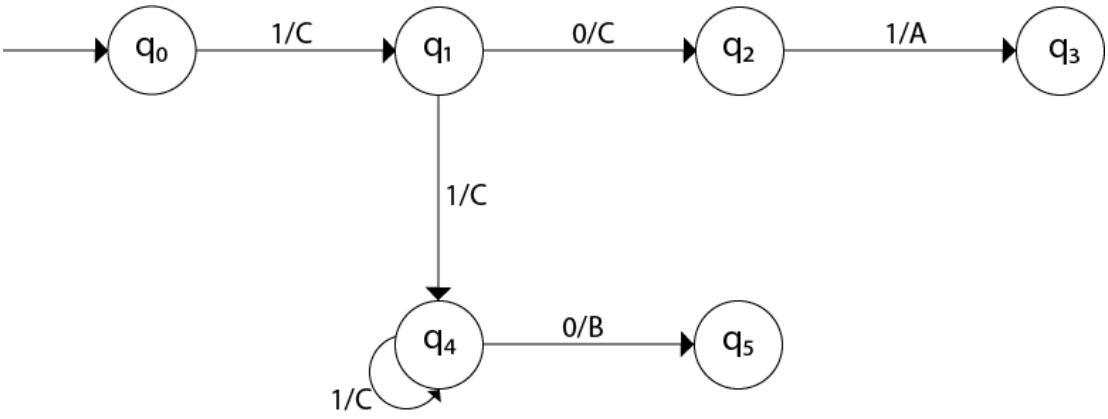
λ': output function where Q × Σ →O

Example 1:

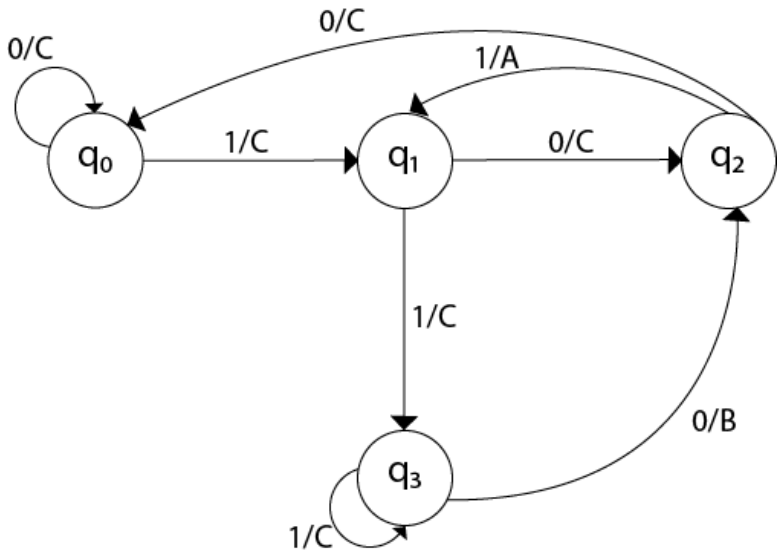
Design a Mealy machine for a binary input sequence such that if it has a substring 101, the machine output A, if the input has substring 110, it outputs B otherwise it outputs C.

Solution: For designing such a machine, we will check two conditions, and those are 101 and 110. If we get 101, the output will be A. If we recognize 110, the output will be B. For other strings the output will be C.

The partial diagram will be:



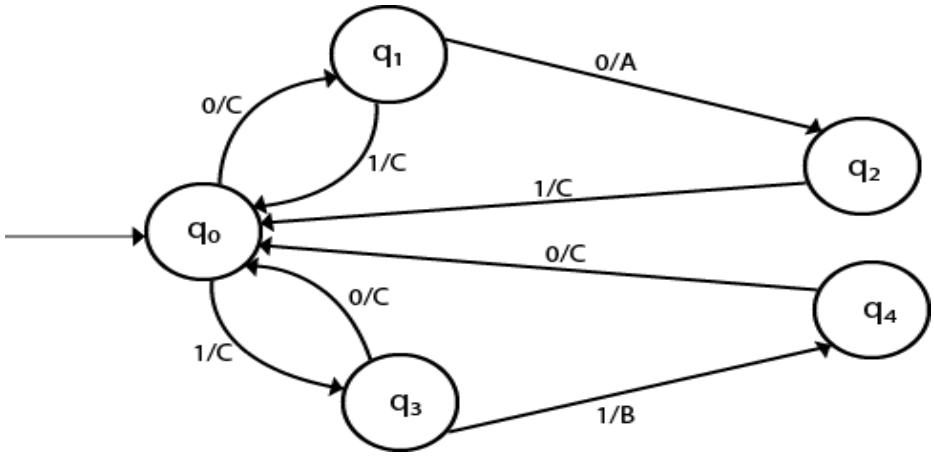
Now we will insert the possibilities of 0's and 1's for each state. Thus the Mealy machine becomes:



Example 2:

Design a 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.

Solution: The mealy machine will be:



Conversion from Mealy machine to Moore Machine

In Moore machine, the output is associated with every state, and in Mealy machine, the output is given along the edge with input symbol. To convert Moore machine to Mealy machine, state output symbols are distributed to input symbol paths. But while converting the Mealy machine to Moore machine, we will create a separate state for every new output symbol and according to incoming and outgoing edges are distributed.

The following steps are used for converting Mealy machine to the Moore machine:

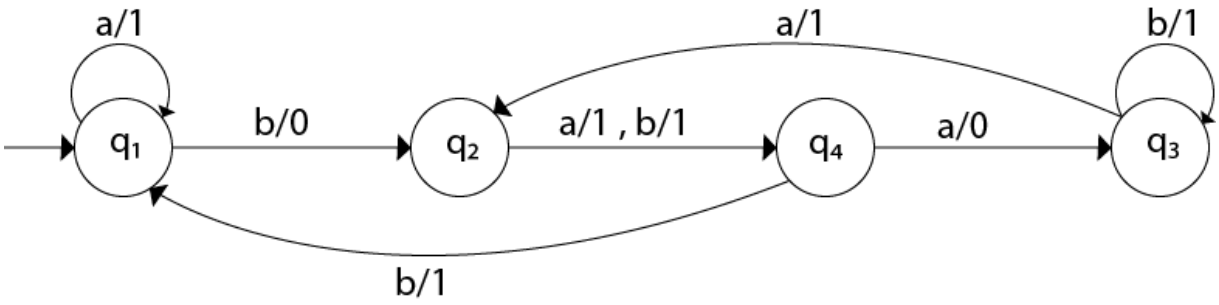
Step 1: For each state(Qi), calculate the number of different outputs that are available in the transition table of the Mealy machine.

Step 2: Copy state Qi, if all the outputs of Qi are the same. Break qi into n states as Qin, if it has n distinct outputs where n = 0, 1, 2....

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

Example 1:

Convert the following Mealy machine into equivalent Moore machine.



Solution:

Transition table for above Mealy machine is as follows:

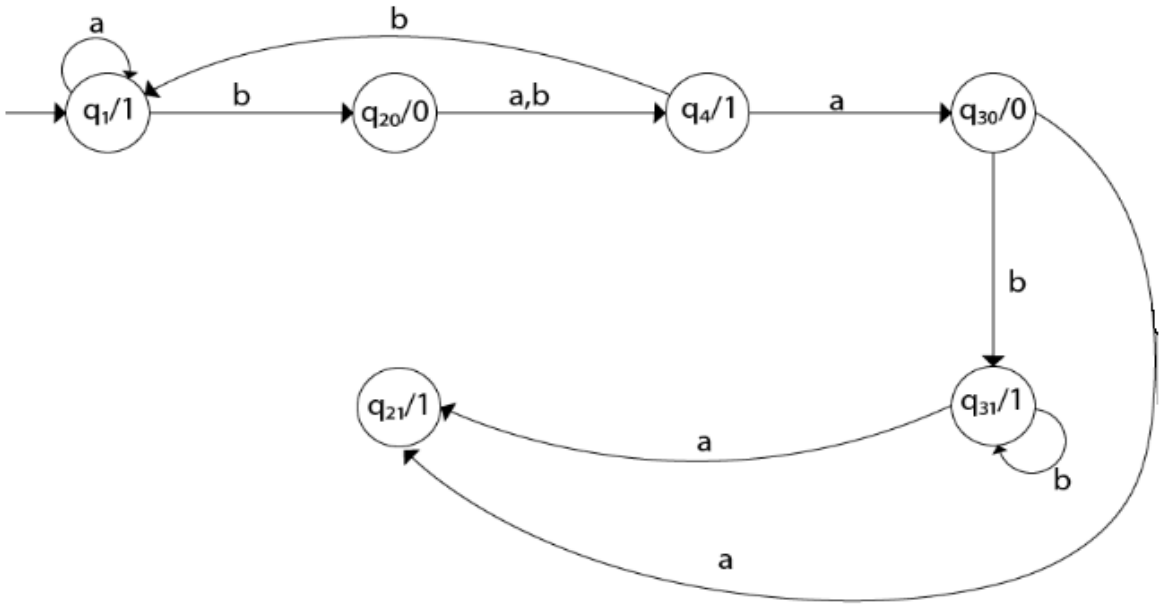
Present State	Next State			
	a		b	
	State	O/P	State	O/P
q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

- For state q1, there is only one incident edge with output 0. So, we don't need to split this state in Moore machine.
- For state q2, there is 2 incident edge with output 0 and 1. So, we will split this state into two states q20(state with output 0) and q21(with output 1).
- For state q3, there is 2 incident edge with output 0 and 1. So, we will split this state into two states q30(state with output 0) and q31(state with output 1).
- For state q4, there is only one incident edge with output 0. So, we don't need to split this state in Moore machine.

Transition table for Moore machine will be:

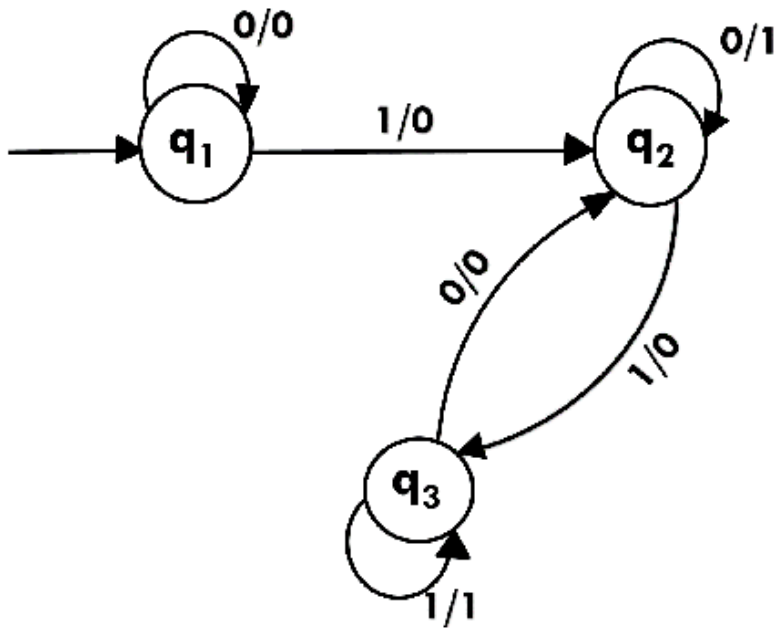
Present State	Next State		Output
	a=0	a=1	
q_1	q_1	q_2	1
q_{20}	q_4	q_4	0
q_{21}	ø	ø	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{21}	q_{31}	1
q_4	q_3	q_4	1

Transition diagram for Moore machine will be:



Example 2:

Convert the following Mealy machine into equivalent Moore machine.



Solution:

Transition table for above Mealy machine is as follows:

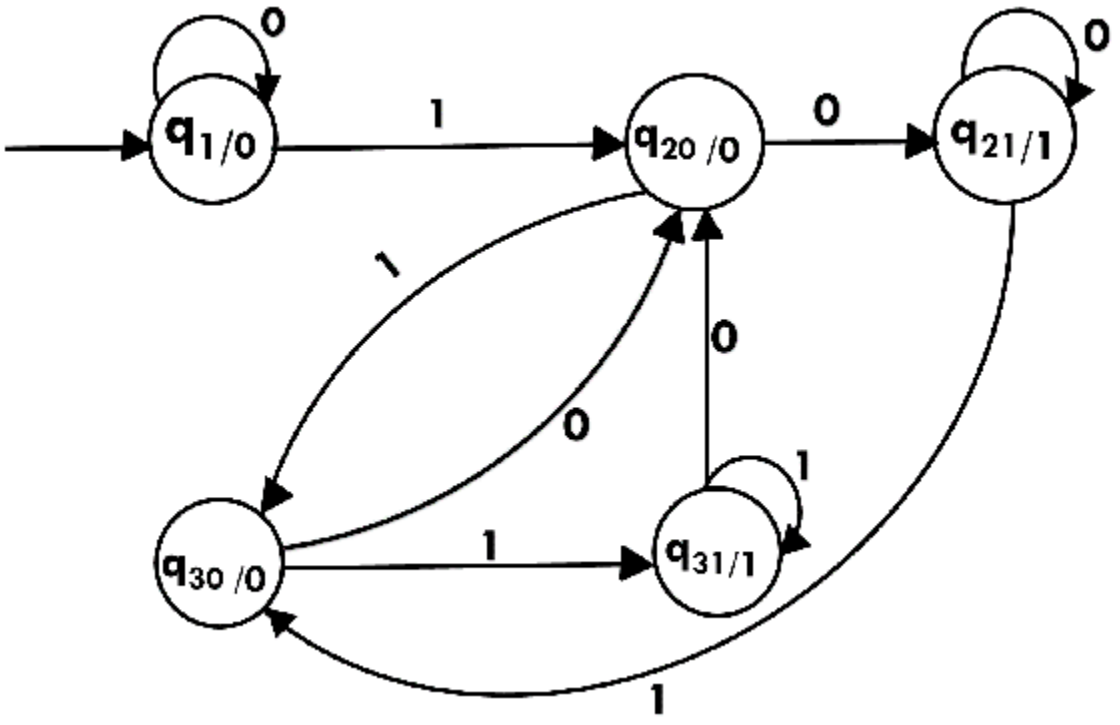
Present State	Next State 0		Next State 1	
	State	o/P	State	o/P
q_1	q_1	0	q_2	0
q_2	q_2	1	q_3	0
q_3	q_2	0	q_3	1

The state q1 has only one output. The state q2 and q3 have both output 0 and 1. So we will create two states for these states. For q2, two states will be q20(with output 0) and q21(with output 1). Similarly, for q3 two states will be q30(with output 0) and q31(with output 1).

Transition table for Moore machine will be:

Present State	Next State 0	Next State 1	o/P
q_1	q_1	q_{20}	0
q_{20}	q_{21}	q_{30}	0
q_{21}	q_{21}	q_{30}	1
q_{30}	q_{20}	q_{31}	0
q_{31}	q_{20}	q_{31}	1

Transition diagram for Moore machine will be:



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.

We cannot directly convert Moore machine to its equivalent Mealy machine because the length of the Moore machine is one longer than the Mealy machine for the given input. To convert Moore machine to Mealy machine, state output symbols are distributed into input symbol paths. We are going to use the following method to convert the Moore machine to Mealy machine.

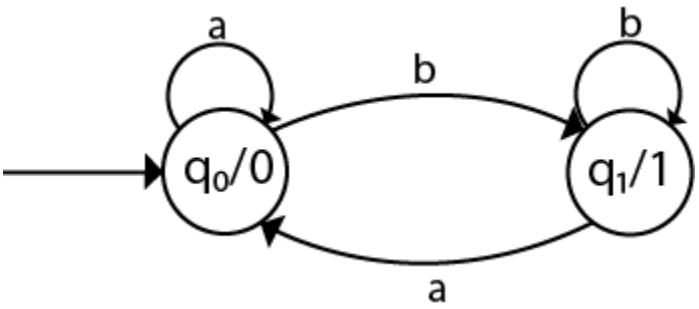
Method for conversion of Moore machine to Mealy machine

Let M = (Q, Σ, δ, λ, q0) be a Moore machine. The equivalent Mealy machine can be represented by M' = (Q, Σ, 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:

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:

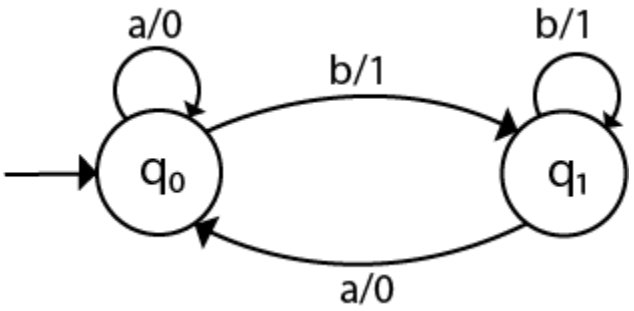
λ' (q1, a) = λ(δ(q1, a))
= λ(q0)
= 0

λ' (q1, b) = λ(δ(q1, b))
= λ(q1)
= 1

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

Q \ Σ	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

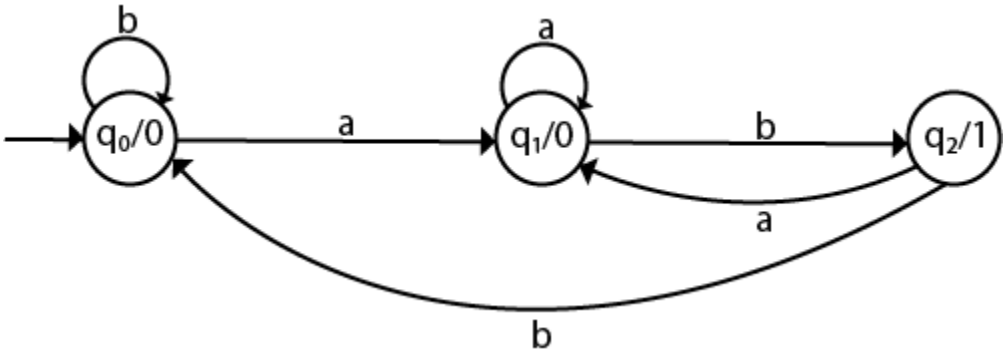
The equivalent Mealy machine will be,



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

λ' (q1, b) = λ(δ(q1, b))
= λ(q2)
= 1

The λ for state q2 is as follows:

λ' (q2, a) = λ(δ(q2, a))
= λ(q1)
= 0

λ' (q2, b) = λ(δ(q2, b))
= λ(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,