Unit-II

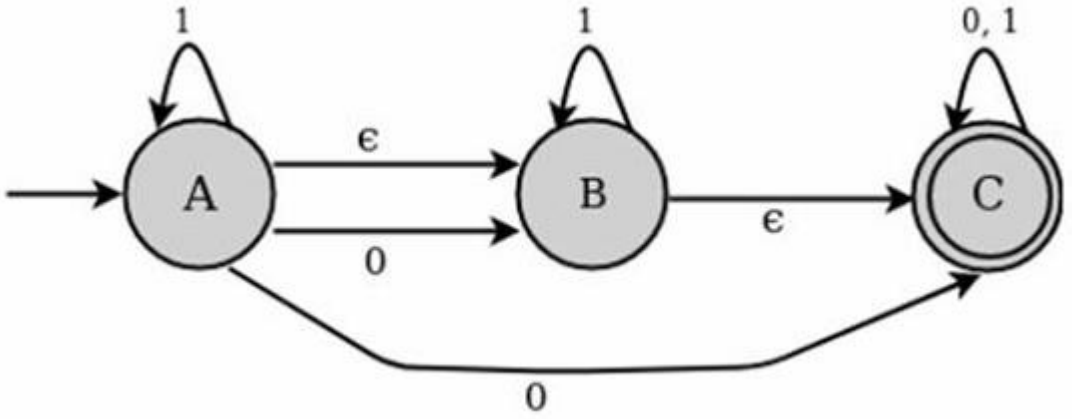
FA

NFA with epsilon transition

- Automaton change its state without reading the input symbol.
- Transitions are depicted by labeling the appropriate arcs with ε.
- Both NFA and ε -NFA recognize exactly the same languages
- ε-NFA - $(Q, \sum, \delta, q_0, F)$ where –

  δ is the transition function where $\delta: Q \times \sum U\{\in\} \rightarrow 2^Q$
- Every state on ∈ goes to itself.

State	0	1	epsilon
A	B,C	A	B
B	-	B	C
C	C	C	-



Conversion of Epsilon NFA to NFA

- The method is mentioned below stepwise −

  Step 1 − Find out all the ε-transitions from each state from Q. That will be called as ε-closure(qi) where, qi ∈Q.

  Step 2 − Then, $\delta 1$ transitions can be obtained. The $\delta 1$ transitions means an ε-closure on δ moves.

  Step 3 − Step 2 is repeated for each input symbol and for each state of given NFA.

  Step 4 − By using the resultant status, the transition table for equivalent NFA without ε can be built.

- NFA with ε to without ε is as follows −

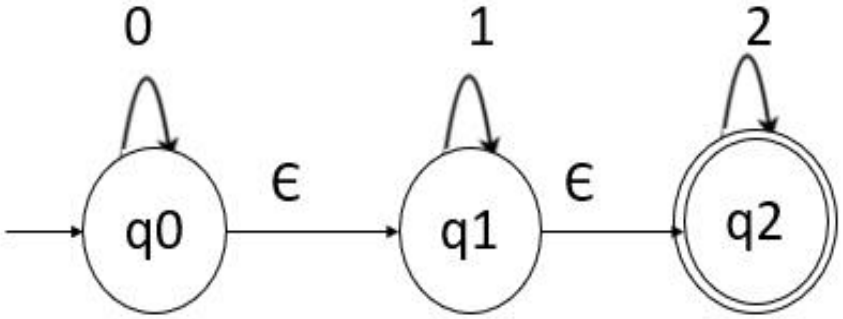
δ1(q,a) = ∈ - closure (δ (δ^(q,**ϵ**),a)) where, δ^(q,**ϵ**) = ∈ - closure(q)

Convert the given NFA with epsilon to NFA without epsilon.

Given NFA **with** ∈ moves is given by M= ({q0,q1,q2}, {0,1,2}, δ, q_0, F) where δ: Q × ∑U{∈} → 2^Q
can be converted to equivalent
NFA **without** ∈ moves M1= ({q0,q1,q2}, {0,1,2}, δ', q_0, F') where δ': Q × ∑ → 2^Q

Step1: − Find out all the ε-transitions from each state from Q
ε-closure(q0) means with null input (no input symbol) we can reach q0, q1, q2.

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



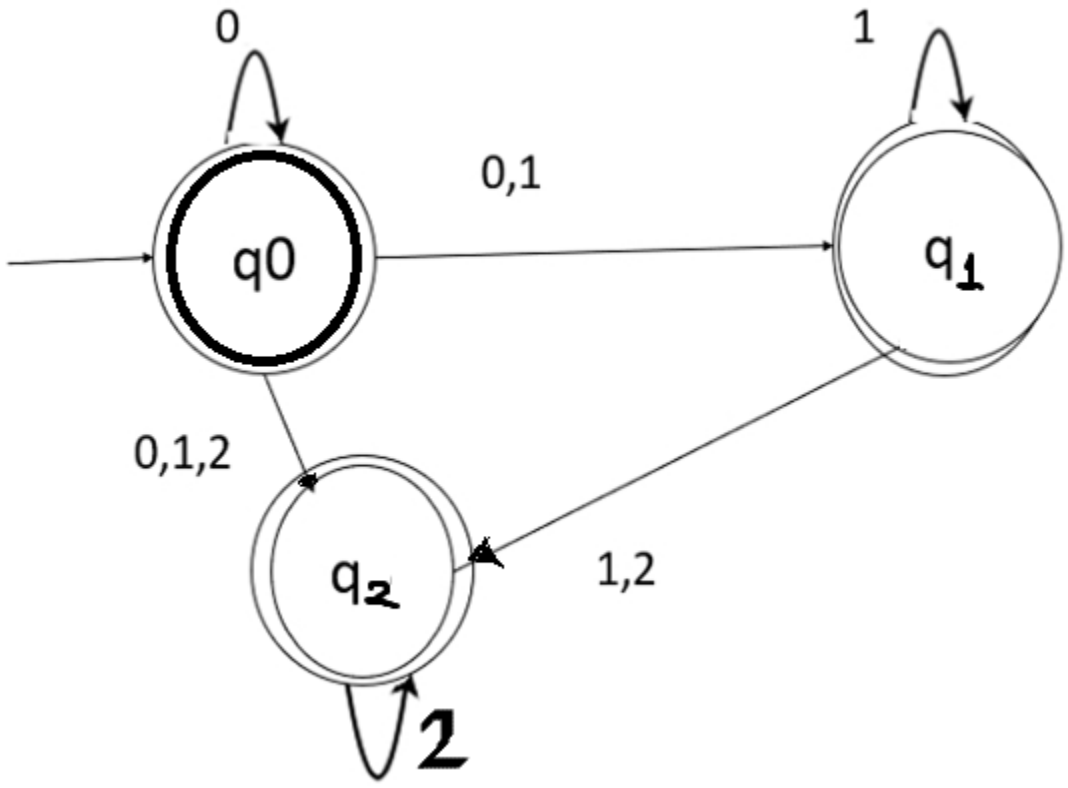
Step 2 − Find out $\delta 1$ transitions for M1

δ'(q0, 0) = ε-closure(δ(δ^(q0, ε),0))
= ε-closure(δ(ε-closure(q0),0))
= ε-closure(δ(q0,q1,q2), 0))
= ε-closure(δ(q0, 0) ∪ δ(q1, 0) U δ(q2, 0))
= ε-closure(q0 U Φ ∪ Φ)
= ε-closure(q0)
= {q0,q1, q2}

δ'(q0, 1) = ε-closure(δ(δ^(q0, ε),1))
= ε-closure(δ(q0,q1,q2), 1))
= ε-closure(δ(q0, 1) ∪ δ(q1, 1) U δ(q2, 1))
= ε-closure(Φ ∪q1 U Φ)
= ε-closure(q1)
= {q1, q2}
δ'(q0, 2) = ε-closure(δ(δ^(q0, ε),2))
= ε-closure(δ(q0,q1,q2), 2))
= ε-closure(δ(q0, 2) ∪ δ(q1, 2) U δ(q2, 2))
= ε-closure(Φ U ΦU q2)
= ε-closure(q2)
= {q2}
δ'(q1, 0) = ε-closure(δ(δ^(q1, ε),0))
= ε-closure(δ(q1,q2), 0))
= ε-closure(δ(q1, 0) U δ(q2, 0))
= ε-closure(Φ ∪ Φ)
= ε-closure(Φ)
= Φ
δ'(q1,1) = ε-closure(δ(δ^(q1, ε),1))= {q1,q2}
δ'(q1, 2) = ε-closure(δ(δ^(q1, ε),2)) ={q2}
δ'(q2, 0)=δ'(q2, 1)= Φ
δ'(q2, 2)={q2}

Step 4: Transition Table for NFA **without** ∈ moves i.e. M1 is

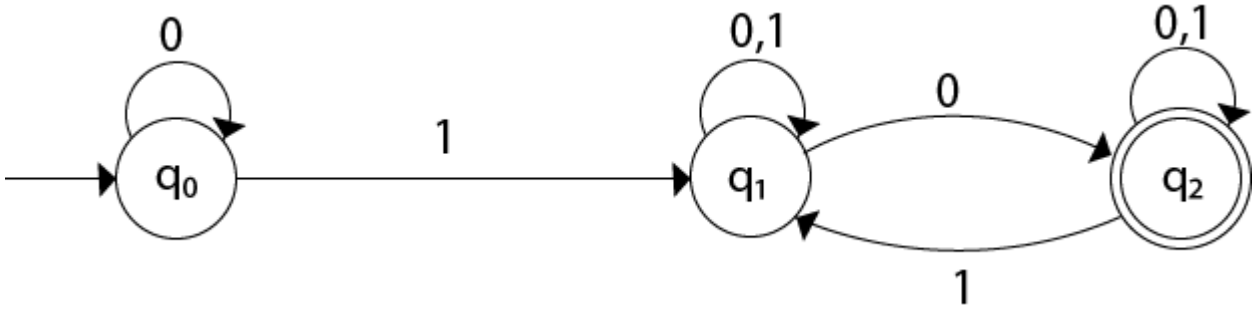
States\inputs	0	1	2
q0	{q0,q1,q2}	{q1,q2}	{q2}
q1	Φ	{q1,q2}	{q2}
q2	Φ	Φ	{q2}



NFA to DFA Conversion

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

Convert the given NFA to DFA.



State	0	1
→q0	q0	q1
q1	{q1, q2}	q1
*q2	q2	{q1, q2}

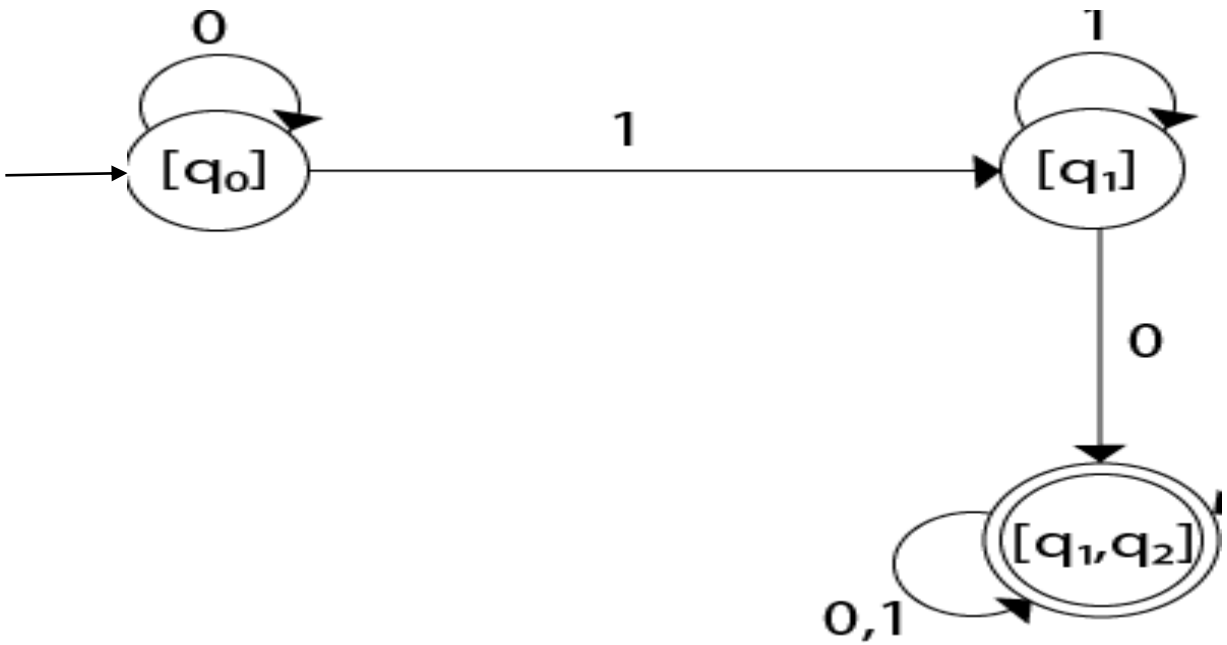
Let, M = ({qo,q1,q2}, {0,1}, δ, q0, {q2}) is an NFA

equivalent DFA denoted by M' = (Q', {0,1}, δ, [q0], F') where δ is constructed as follows

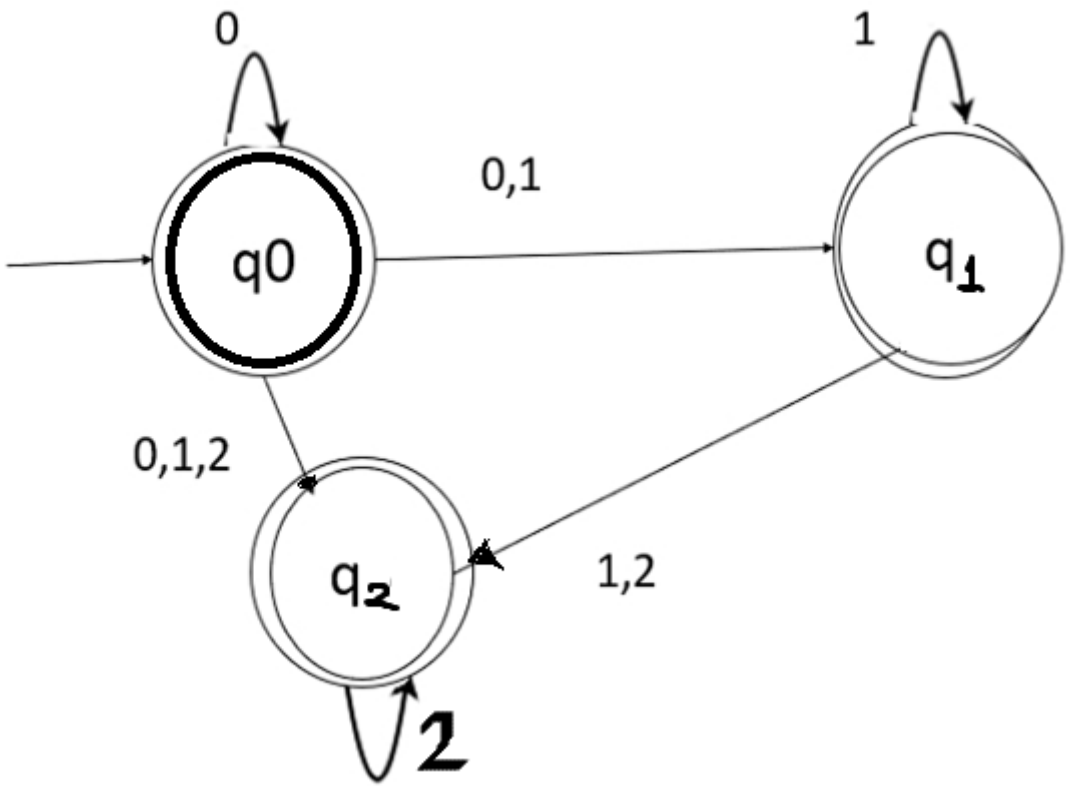
1.δ([q0], 0) = [q0]
2.δ([q0], 1) = [q1]
3.δ([q1], 0) = [q1, q2]
4. δ([q1, q2], 0) = δ(q1, 0) ∪ δ(q2, 0)
= {q1, q2} ∪ {q2}
= [q1, q2]
5. δ([q1, q2], 1) = δ(q1, 1) ∪ δ(q2, 1)
= {q1} ∪ {q1, q2}
= {q1, q2}
= [q1, q2]

State	0	1
→[q0]	[q0]	[q1]
[q1]	[q1, q2]	[q1]
*[q1, q2]	[q1, q2]	[q1, q2]

DFA

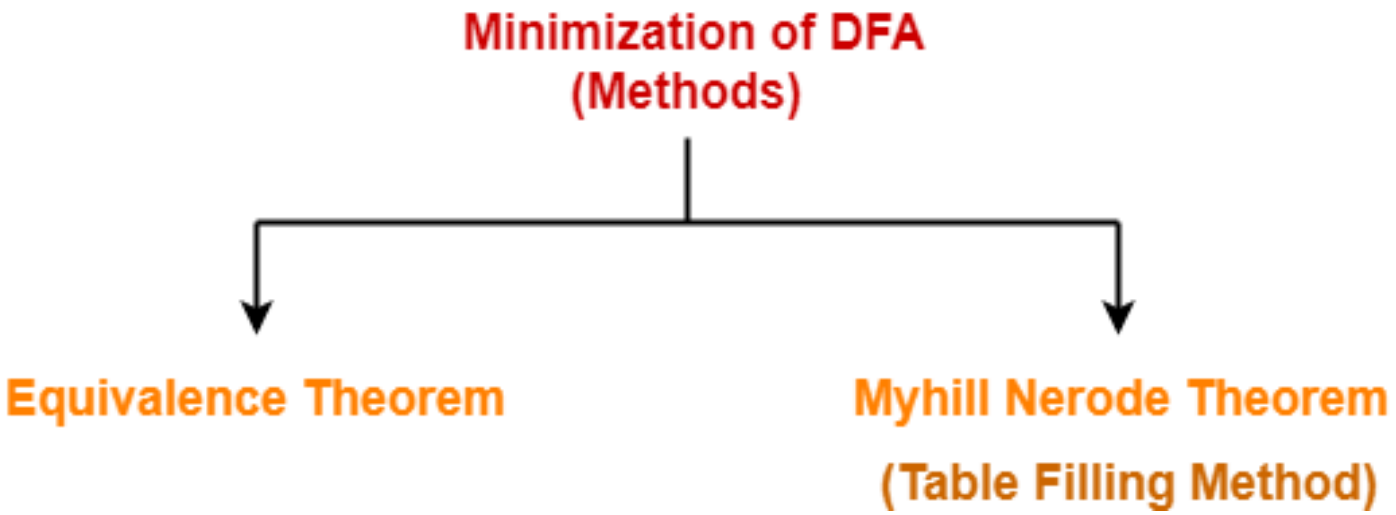


Convert the given NFA to DFA



Minimization of FA

The two popular methods for minimizing a DFA are-



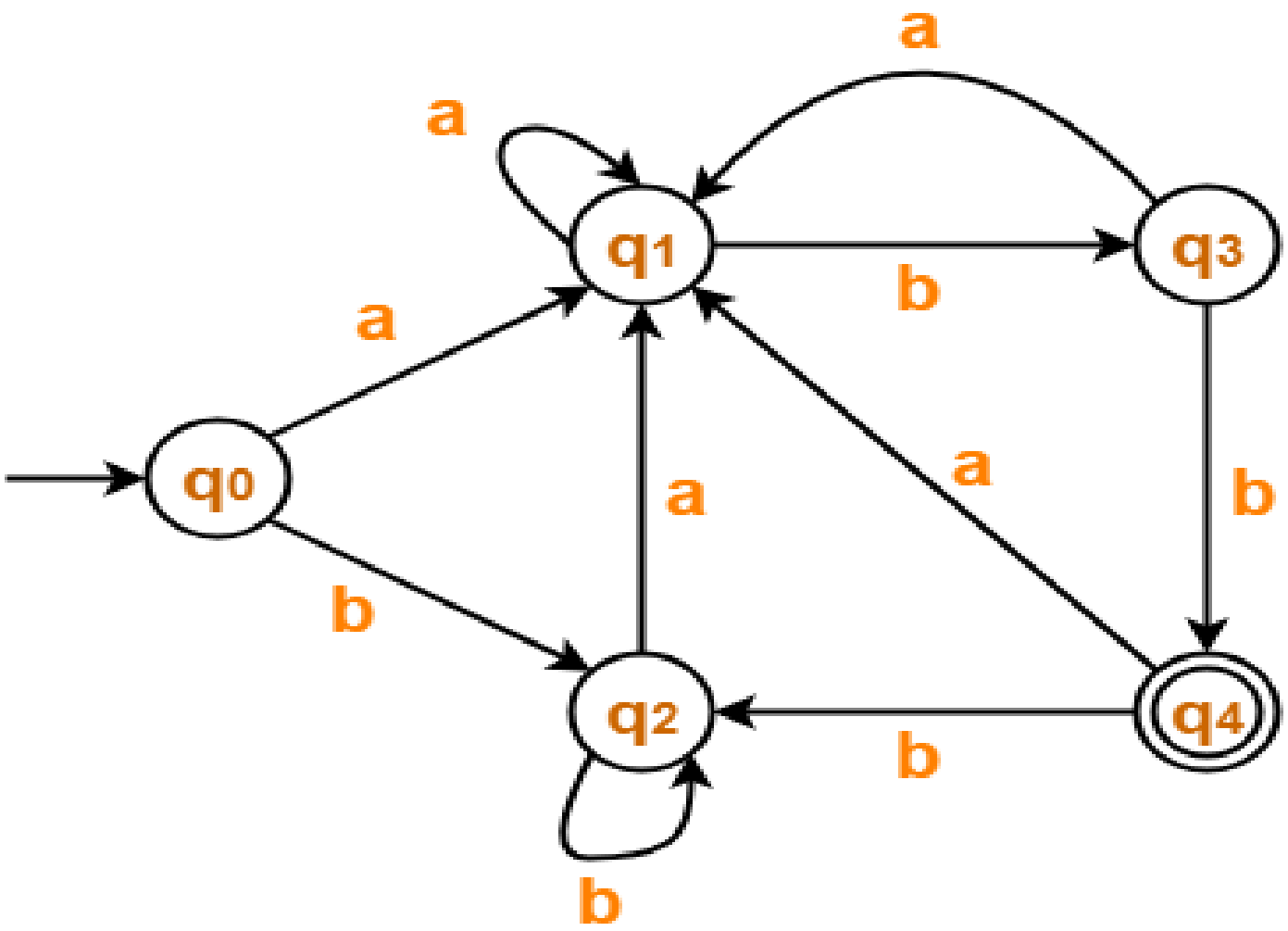
Equivalence Theorem

- Construct automaton with minimum number of states equivalent to a given automaton M.
- Interest lies only in strings accepted by M.
- Two states q1 and q2 are equivalent (denoted by q1$\equiv q2$) if both $\delta(q1, x) and \delta(q2, x)$ are final states or both are non-final states for all $x \in \sum *$
- Any two final states are 0-equivalent and any two non-final states are also 0-equivalent.

Construction of Minimum Automaton

1. (Construction of $\prod_0$)- $\prod_0 = (Q_1^0, Q_2^0)$ where Q_1^0=set of all final states and Q_2^0=Q-Q_1^0.
2. (Construction of $\prod_{k+1}\ from\ \prod_k$)
3. (Construct $\prod_n\ for$ n=1,2…… until $\prod_n = \prod n + 1$
4. Construction of minimum automaton- the state table is obtained by replacing a state q by the corresponding equivalence class [q].

Minimize the given DFA



Given DFA M=({q_0,q_1,q_2,q_3,q_4},{a,b}, δ,q0,{q_4}) where transition table is

	a	**b**
→**q0**	q1	q2
q1	q1	q3
q2	q1	q2
q3	q1	*q4
***q4**	q1	q2

Can be converted to equivalent minimum FA

M' =(Q', {a, b}, δ', *qo', F')*

Now using Equivalence Theorem, we have-

$\prod_0 = \{ q_0 , q_1 , q_2 , q_3 \} \{ q_4 \}$

$\prod_1 = \{ q_0 , q_1 , q_2 \} \{ q_3 \} \{ q_4 \}$

$\prod_2 = \{ q_0 , q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$

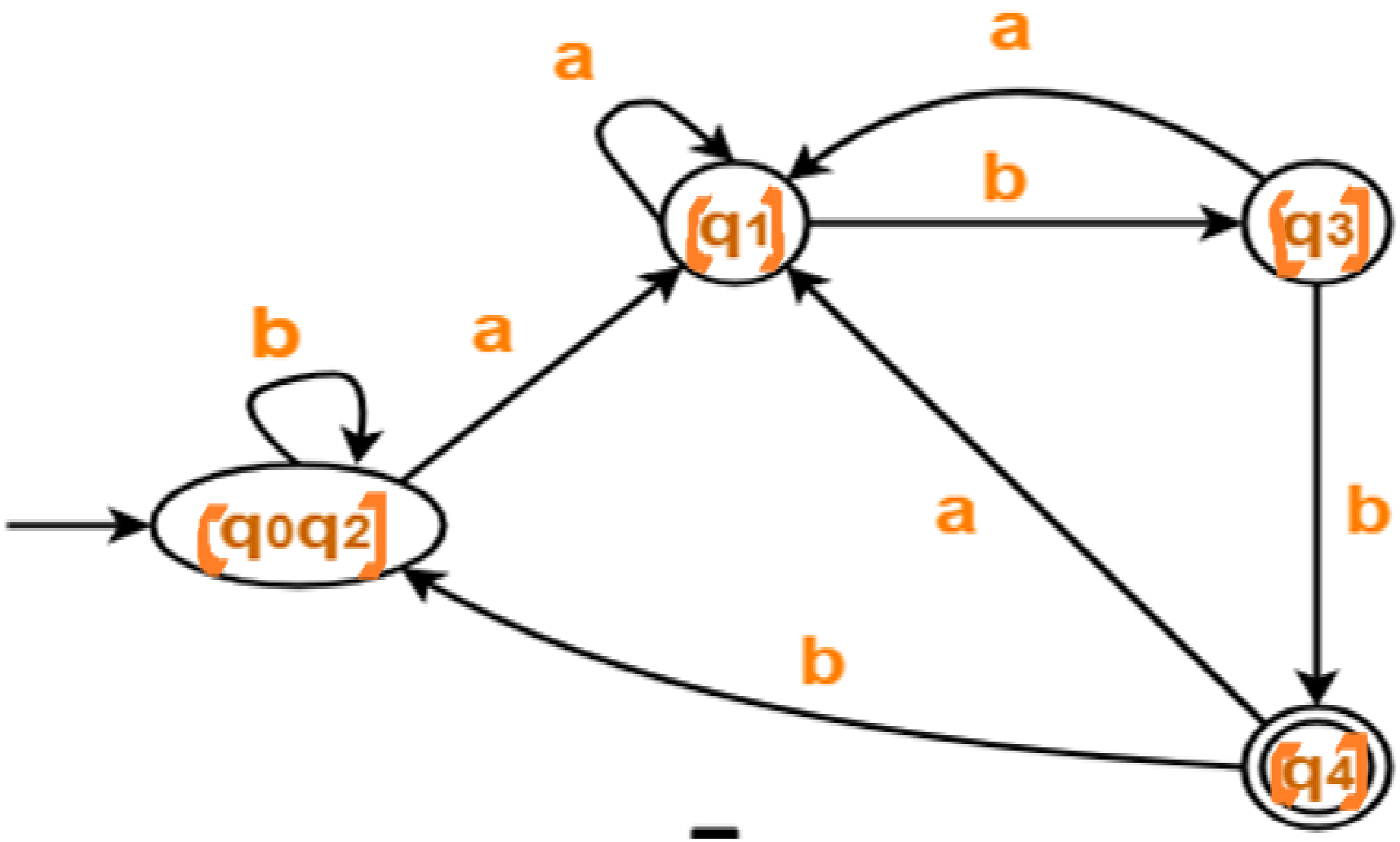
$\prod_3 = \{ q_0 , q_2 \} \{ q_1 \} \{ q_3 \} \{ q_4 \}$

Since $\prod_2 = \prod_3$, so we stop.

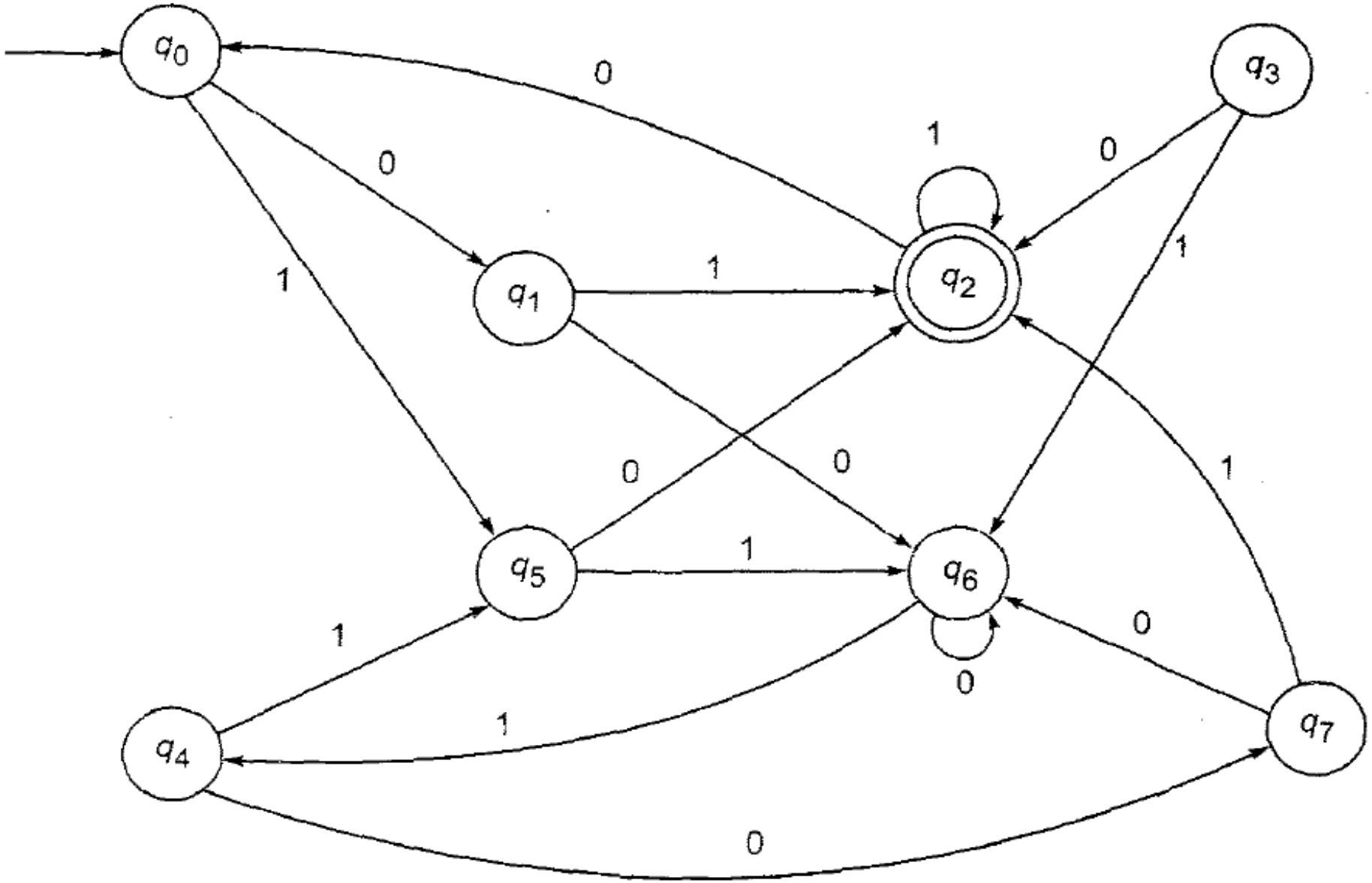
From $\prod_3$, we infer that states q_0 and q_2 are equivalent and can be merged together.

So, Our minimal DFA is-

M'=({[q0,q2],[q1],[q3],[q4]},{a,b}, δ,[q0,q2],[,{[q4]})


a
a
b
{q1}
{q3}
b
a
{q0q2}
a
b
b
{q4}

Find minimum FA



Minimum FA

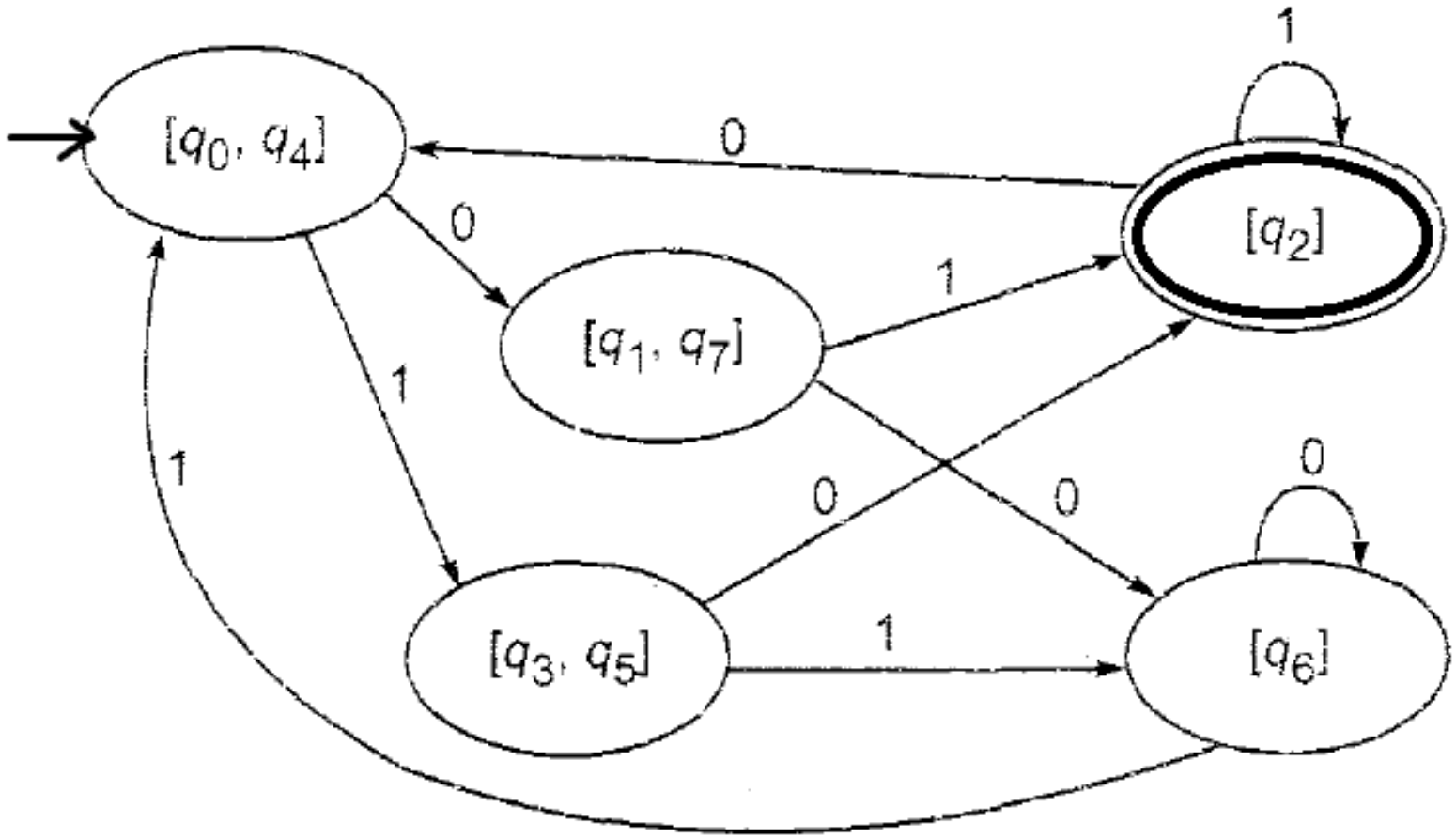
M' =(Q', {0, 1}, δ', *qo', F')*

where

Q' = {*[qo,* q4], [q2], *[q6], [q1,q7], [q3,q5]}*

qo' = [qo, q4]

F' = {[q2]}

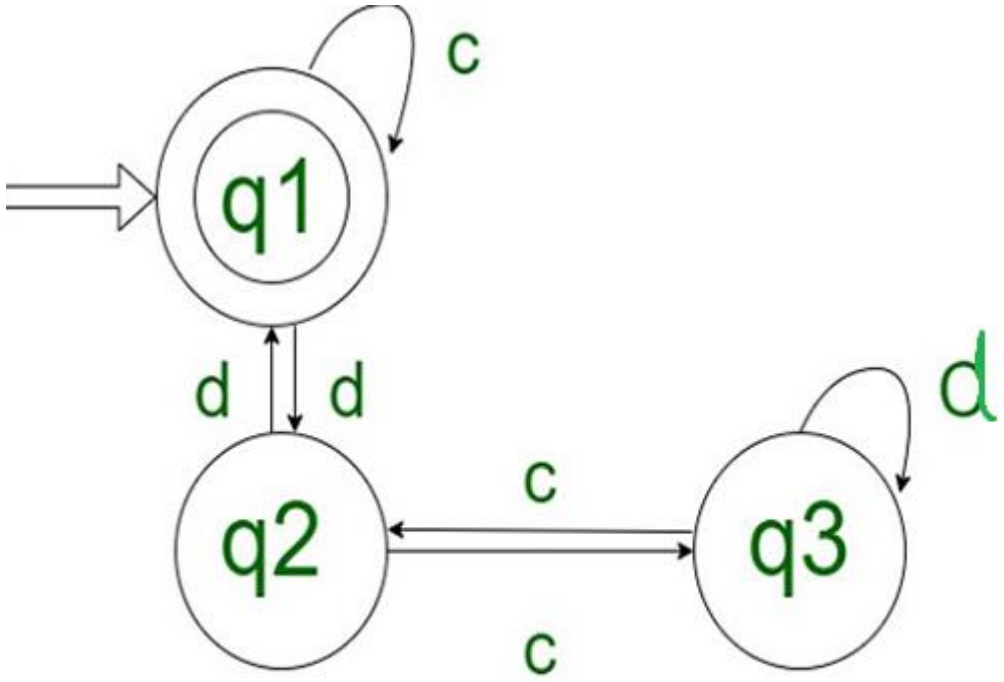


Equivalence between two FA

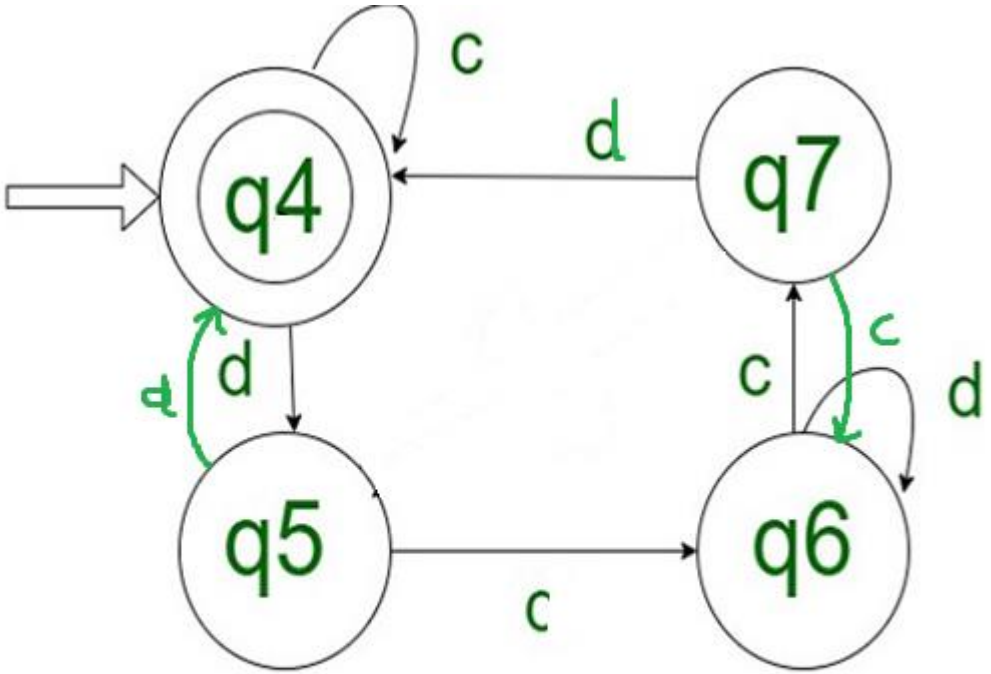
- Any Two Automaton is equivalent if both accept exactly the same set of input strings.
- Steps

  1. The initial and final states of both the automatons must be same.
  2. Every pair of states chosen is from a different automaton only.
  3. While combining the states with the input alphabets, the pair results must be either both final states or intermediate states.(i.e both should lie either in the final state or in the non-final state).
  4. If the resultant pair has different types of states, then it will be non-equivalent. (i.e. One lies in the final state and the other lies in the intermediate state).

Determine whether both FA are equivalent?



AUTOMATON-1

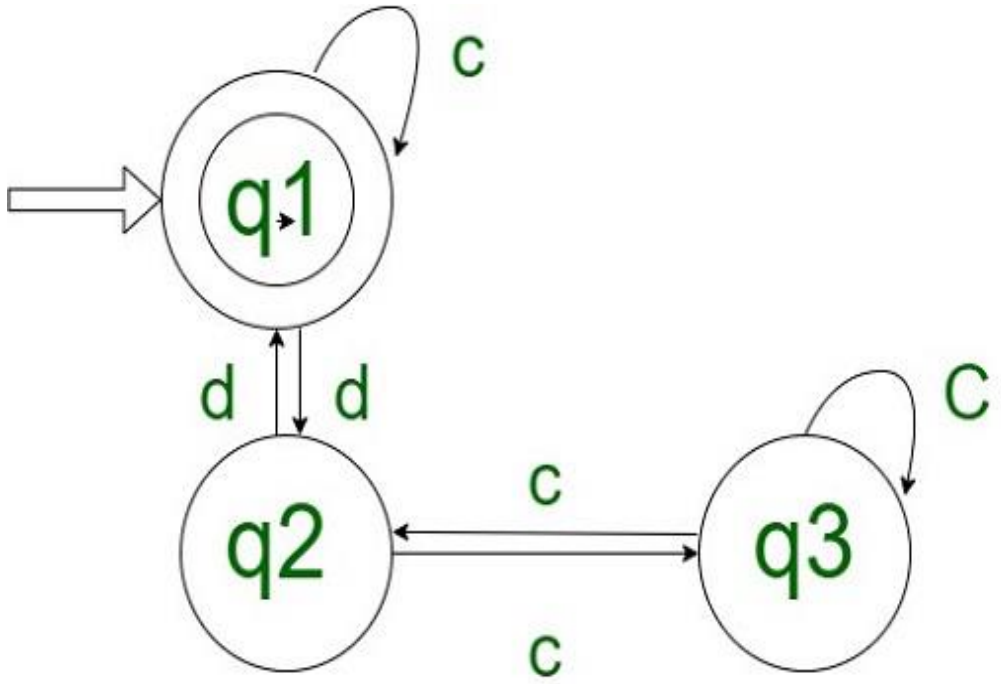


AUTOMATON-2

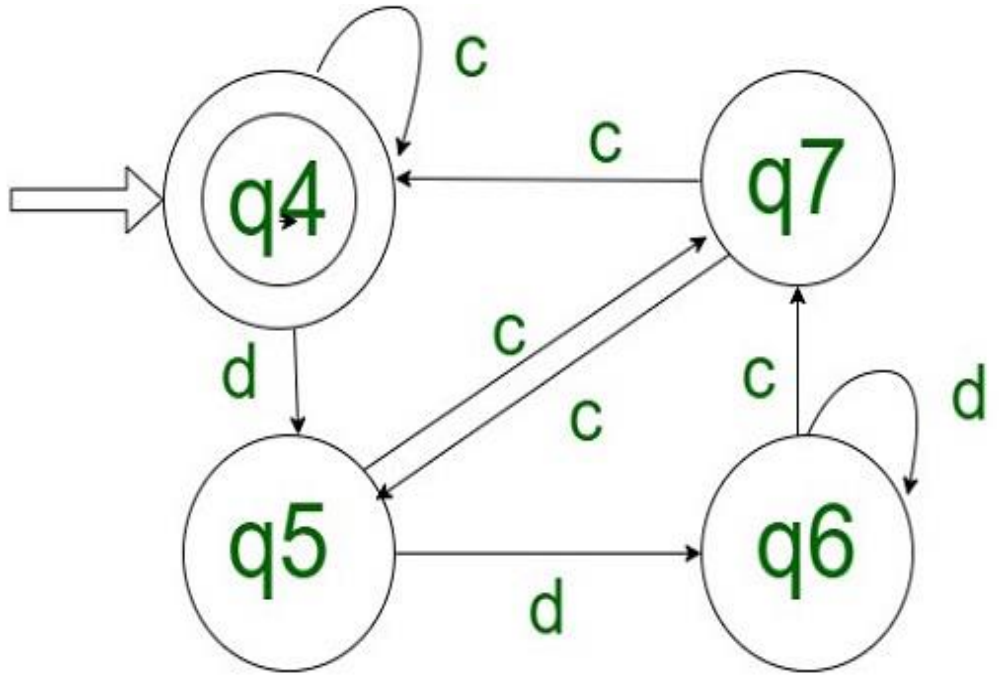
Comparison Table

States	c	d
(q1,q4)	(q1,q4)=>(F.S,F.S)	(q2,q5)=>(I.S,I.S)
(q2,q5)	(q3,q6)=>(I.S,I.S)	(q1,q4)=>(F.S,F.S)
(q3,q6)	(q2,q7)=>(I.S,I.S)	(q3,q6)=>(I.S,I.S)
(q2,q7)	(q3,q6)=>(I.S,I.S)	(q1,q4)=>(F.S,F.S)

Determine whether both FA are equivalent?



AUTOMATON-1



AUTOMATON-2

FIGURE -2

FA with output: Mealy and Moore Machine

- **Mealy Machine** is defined as a machine in the theory of computation whose output values are determined by both its current state and current inputs. In this machine at most one transition is possible. It has 6 tuples: (Q, ∑, ▲, δ, λ, q0)
  - Q is a finite set of states
  - q0 is the initial state
  - ∑ is the input alphabet
  - ▲ is the output alphabet
  - δ is the transition function that maps Q×∑ → Q
  - 'λ' is the output function that maps Q×∑→ ▲

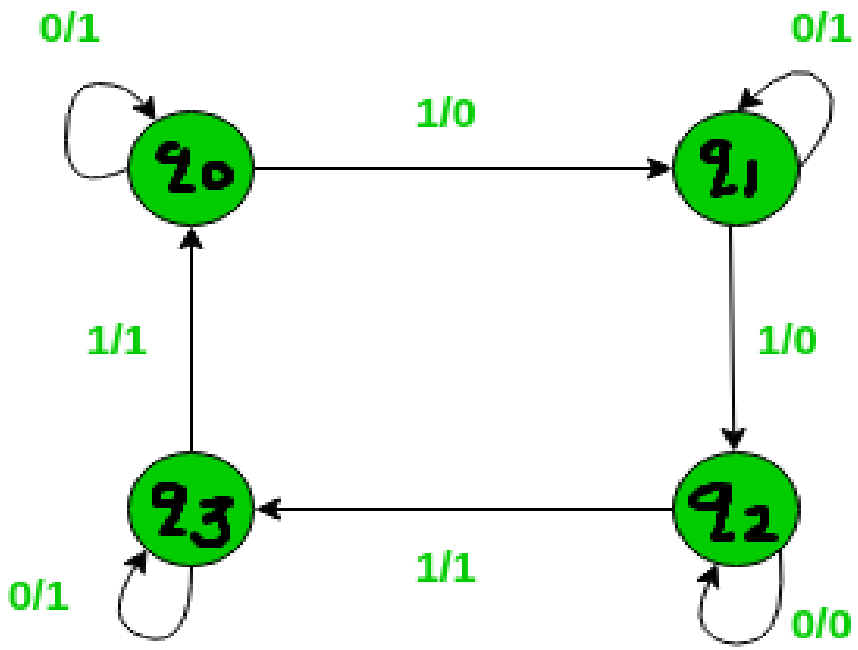


Figure - Mealy machine

Present state	Next state			
	input = 0		input = 1	
	State	Output	State	Output
→ q0	q0	1	q1	0
q1	q1	1	q2	0
q2	q2	0	q3	1
q3	q3	1	q0	1

Moore's machine is defined as a machine in the theory of computation whose output values are determined only by its current state. It has also 6 tuples

(Q, ∑, ▲, δ, λ, q0)

1. Q is a finite set of states
2. q0 is the initial state
3. ∑ is the input alphabet
4. ▲ is the output alphabet
5. δ is the transition function that maps Q×∑ → Q
6. λ is the output function that maps Q → ▲

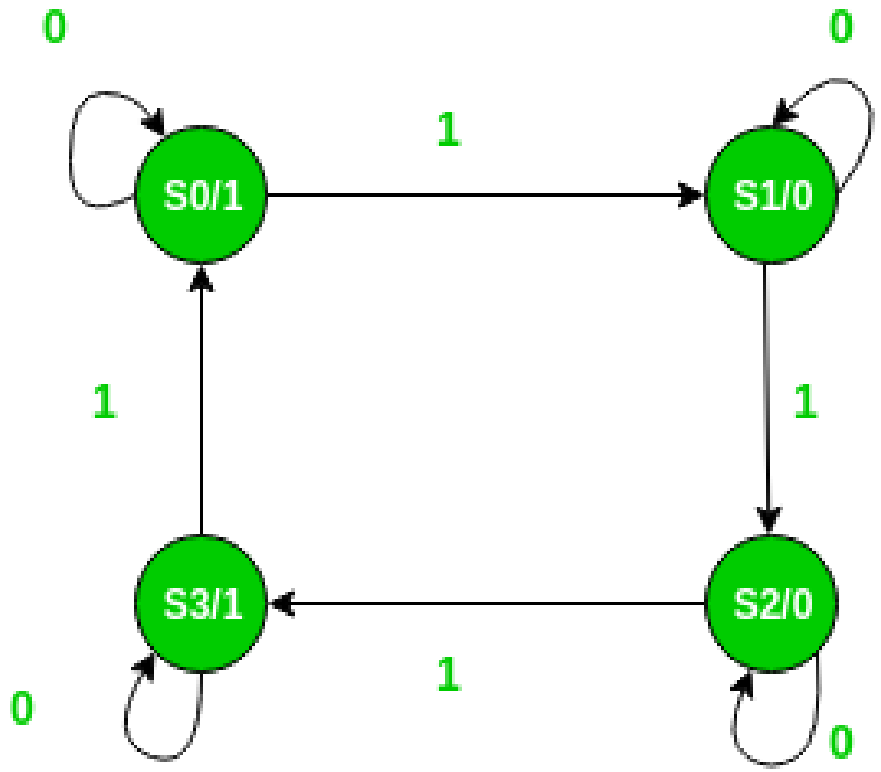


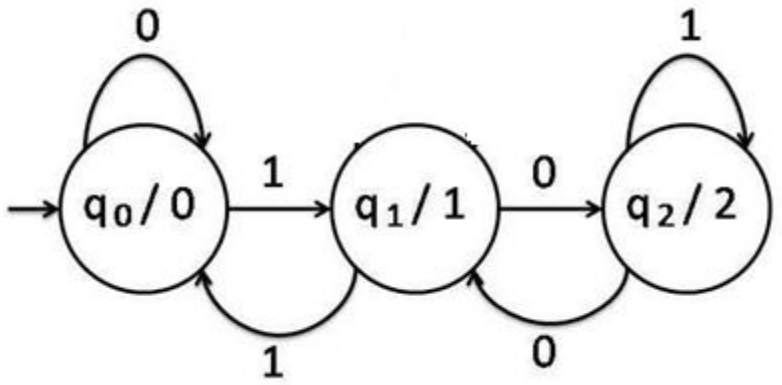
Figure - Moore machine

Present state	Next State		Output
	Input = 0	Input = 1	
→ s0	s0	S1	1
S1	S1	S2	0
S2	S2	s3	0
s3	s3	s0	1

Design a Moore machine to determine the residue mod 3 of a binary number

M=({q0,q1,q2}, {0,1}, {0,1,2}, δ, λ, q0)

Present state	Input = 0	Input = 1	Output
→ q0	q0	q1	0
q1	q1	q2	1
q2	q2	q0	2

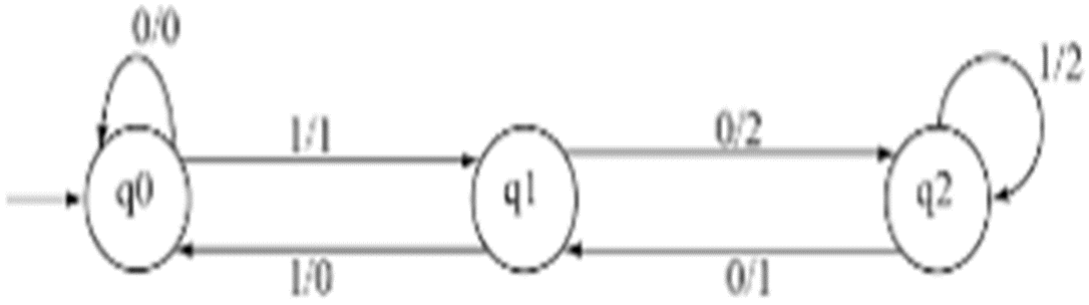


Validate for input 1000

Design a Mealy machine to determine the residue mod 3 of a binary number

M=({q0,q1,q2}, {0,1}, {0,1,2}, δ, λ, q0)

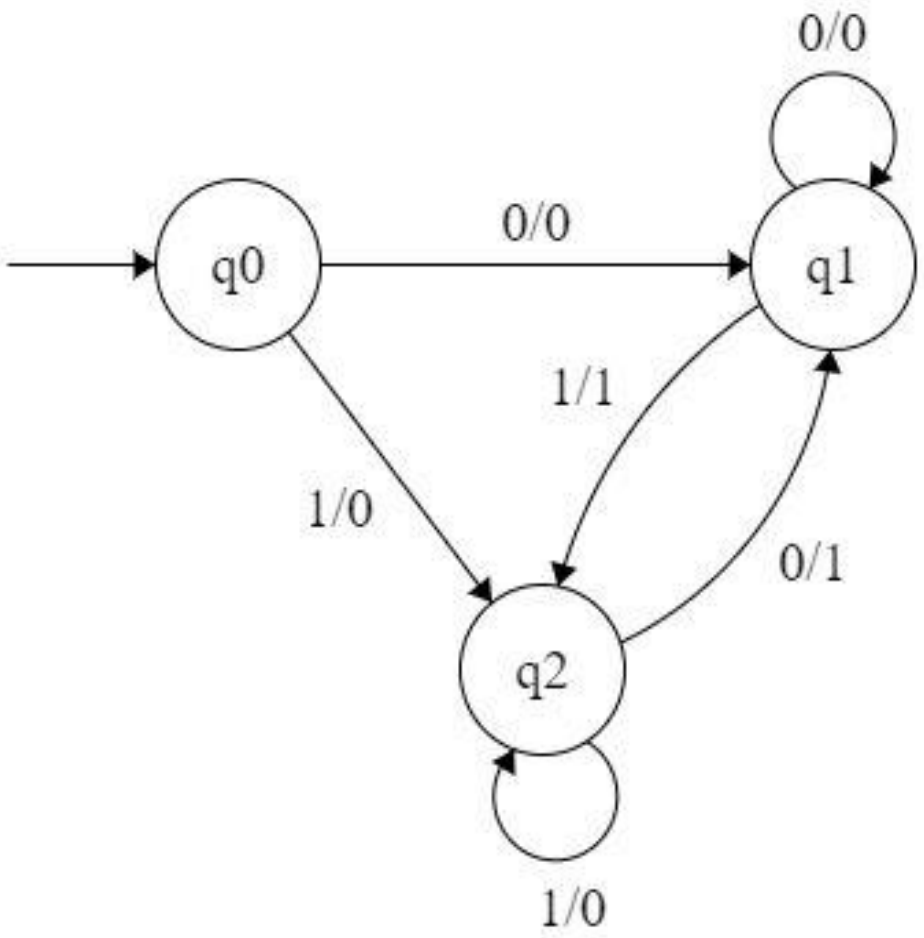
Present state	Next state			
	input = 0		input = 1	
	State	Output	State	Output
→ q0	q0	0	q1	1
q1	q2	2	q0	0
q2	q1	1	q2	2



Validate for input 1000

Conversion between Mealy and Moore Machine

- **Convert following Mealy Machine to Moore Machine**



Let given Mealy Machine M=({q0,q1,q2}, {0,1}, {0,1}, δ, λ, q0) where δ and λ is given in table

Present state	Next state			
	input = 0		input = 1	
	State	Output	State	Output
→ q0	q1	0	q2	0
q1	q1	0	q2	1
q2	q1	1	q2	0

Can be converted in Moore Machine
M'=({q0,q10, q11,q20, q21}, {0,1}, {0,1}, δ', λ', q0)
where δ' and λ' is given in table

	Input=0	Input=1	
Present State	Next State	Next State	Output
→q0	q10	q20	0
q10	q10	q21	0
q11	q10	q21	1
q20	q11	q20	0
q21	q11	q20	1

Convert following moore machine to mealy machine

Convert following Moore machine to mealy machine

TABLE 3.14 Moore Machine of Example 3.10

Present state	*Next state*		*Output*
	$a = 0$	$a = 1$	
$\rightarrow q_0$	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

TABLE 3.15 Mealy Machine of Example 3.10

Present state	*Next state*			
	$a = 0$		$a = 1$	
	state	*output*	*state*	*output*
$\rightarrow q_0$	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

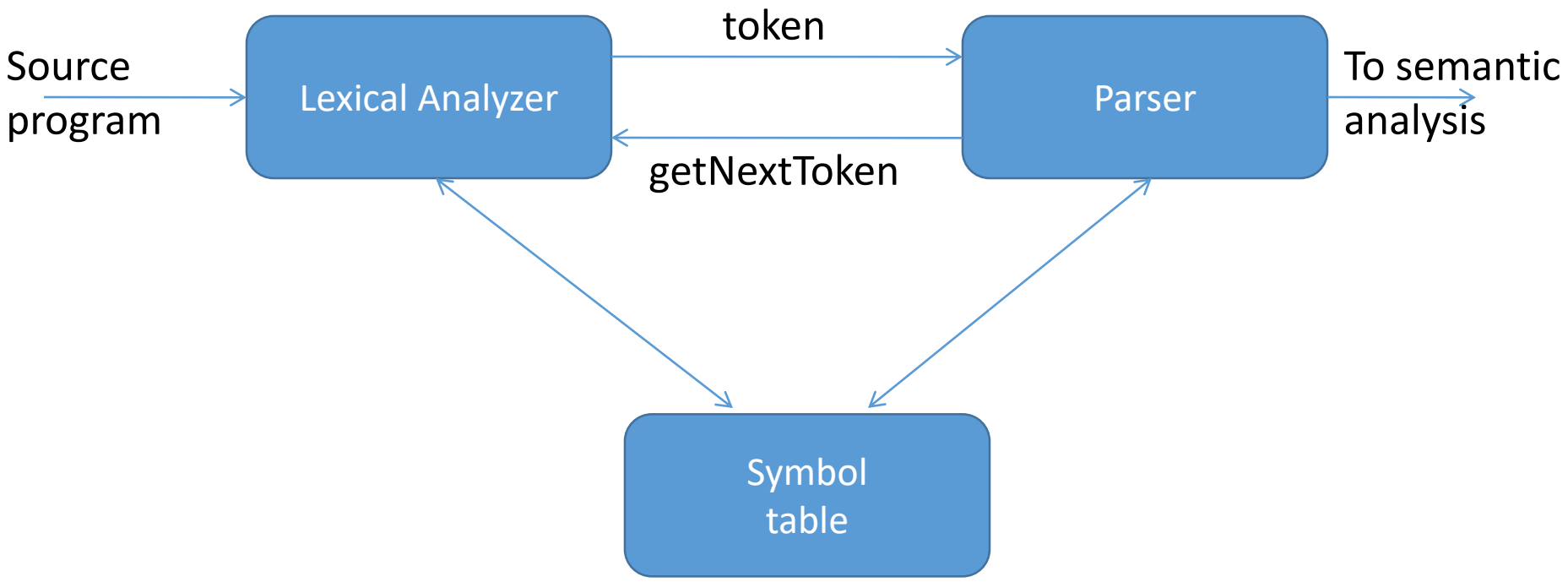
Applications of Finite Automata

- Automata is a machine that can accept the Strings of a *Language L* over an *input alphabet*. So far we are familiar with the Types of Automata .
- The Expressive Power of any machine can be determined from the class or set of Languages accepted by that particular type of Machine.
- Here is the increasing sequence of expressive power of machines

$$FA < DPDA < PDA < LBA < TM$$

- As we can observe that FA is less powerful than any other machine. It is important to note that DFA and NFA are of same power because every NFA can be converted into DFA and every DFA can be converted into NFA .
- The applications of Finite Automata are as follows −
  - Design of the lexical analysis of a compiler.
  - Recognize the pattern by using regular expressions.
  - Use of the Mealy and Moore Machines for designing the combination and sequential circuits.
  - Helpful in text editors.
  - Used for spell checkers.

The role of lexical analyzer



Tokens, Patterns and Lexemes

- A token is a pair, a token name and an optional token value ex.
  - 1) Identifiers
  - 2) keywords
  - 3) operators
  - 4) special symbols
  - 5)constants
- A pattern is a description of the form that the lexemes of a token may take
- A lexeme is a sequence of characters in the source program that matches the pattern for a token

Example tokens, lexemes, patterns

Token	Sample Lexemes	Informal description of pattern
if	if	if
While	While	while
Relation	<, <=, = , <>, > >=	< or <= or = or <> or > or >=
Id	count, sun, i, j, pi, D2	Letter followed by letters and digits
Num	0, 12, 3.1416, 6.02E23	Any numeric constant

Lexical Analyzer

- A lexical analyzer takes source code as a string, and outputs sequence of tokens.

For example,

for i = 1 to max
do x[i] = 0;

- might have token sequence

for id = num to id do id [id] = num sep

- As a token is identified, there may be an action.

For example, when a number is identified, its value is calculated,