



# Introduction to Finite Automata

Unit 2



## Introduction

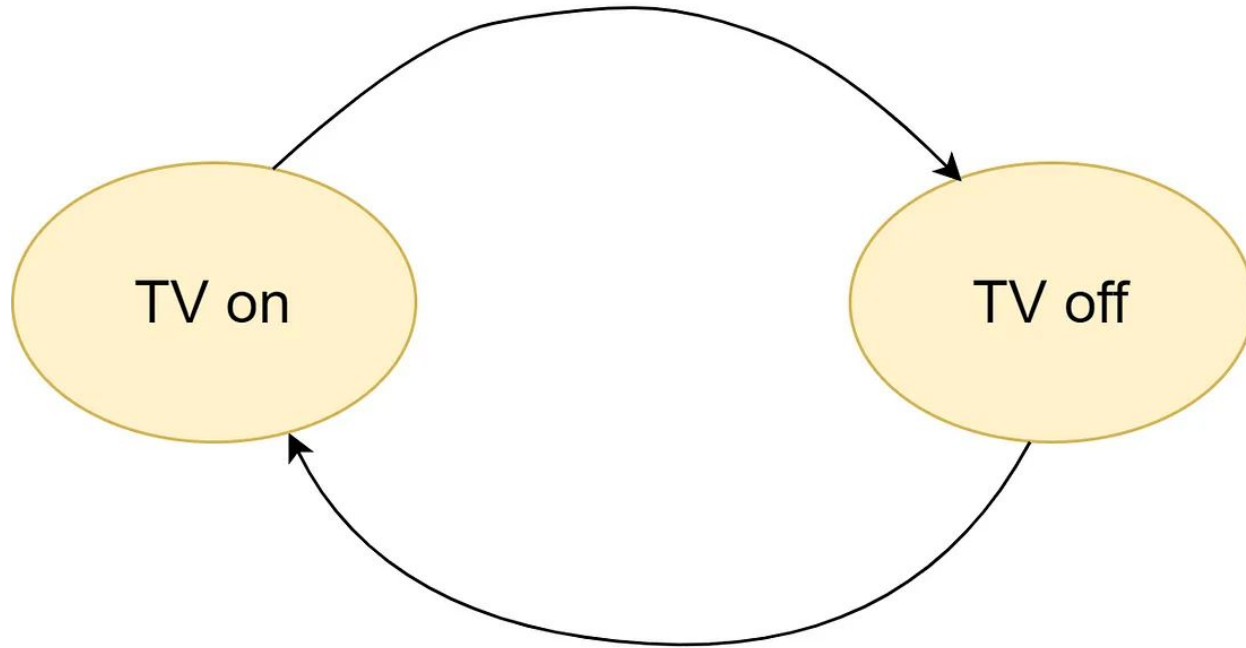
Consider a person watching TV

The TV is in **on** state; when it is switched off, it goes to **off** state

When it is switched on, it again goes to **on** state

This can be represented by the following diagram, called **state diagram**

press power button



press power button



# Introduction

A language is a subset of the set of strings over an alphabet

A language can be generated by grammar

A language can also be recognized by a machine

Such machine is called a recognition device, and a commonly used one is the finite state automaton

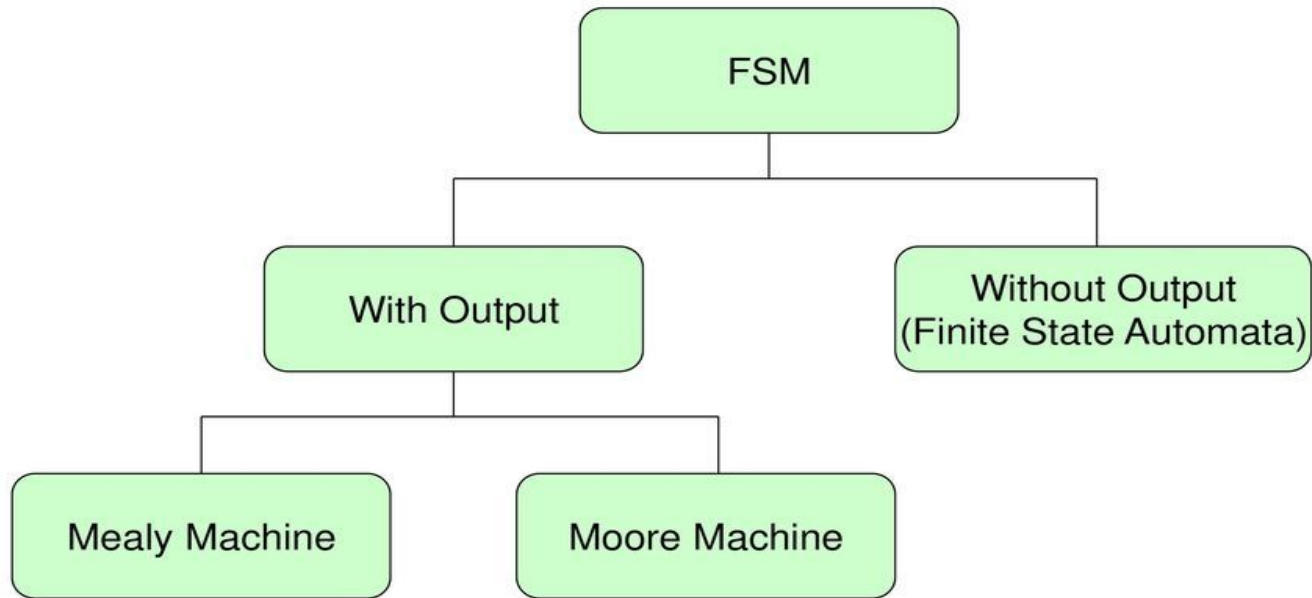


# Finite State Machine

A **finite state machine** (FSM) is a conceptual model that describes systems with a finite number of states and transitions to represent their behavior and logic

Finite state machine is used to recognize patterns

# Types of FSM





## Finite Automata

A finite automaton is a simple machine used to recognize patterns

It is an abstract machine that has a set of **states** and its control moves from state to state in response to external **inputs**

The control may either be **deterministic** (DFA) meaning that the automaton can't be in more than one state at any one time, or **non deterministic** (NFA), meaning that it may be in several states at once



## Applications

FSMs are used in applications in computer science and data networking

They are basis for programs for spell checking, indexing, grammar checking, searching large bodies of text, recognizing speech, transforming text using markup languages such as XML & HTML, and network protocols that specify how computers communicate





## Deterministic Finite Automata (DFA)

A DFA is defined by a quintuple (5-tuple) as  $(Q, \Sigma, \delta, q_0, F)$ , where

$Q$  = Finite set of states

$\Sigma$  = Finite set of input symbols

$\delta$  = A transition function that maps  $Q \times \Sigma \rightarrow Q$

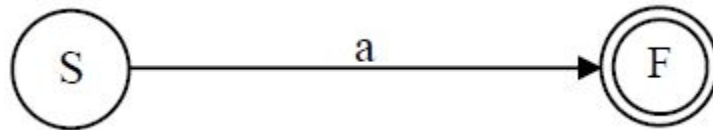
$q_0$  = A start state;  $q_0 \in Q$

$F$  = Set of final states;  $F \subseteq Q$

## Deterministic Finite Automata

A transition function  $\delta$  takes a state and an input symbol as arguments and returns a state

eg:



Here, the transition function takes state **S** and input **a** as inputs and transitions it into state **F**, i.e.  $\delta(S, a) = F$



# Deterministic Finite Automata

There are two common notations for describing this class of automata

- Transition Table
- Transition Diagram



## Transition Table

Transition table is a conventional, tabular representation of the transition function  $\delta$

It takes the arguments from  $Q \times \Sigma$  and returns a value which is one of the states of the automation

The row of the table corresponds to the states while column corresponds to the input symbol



## Transition Table

The starting state in the table is represented by  $\rightarrow$  followed by the state i.e.  $\rightarrow q$ , for  $q$  being start state, whereas final state as  $*q$ , for  $q$  being final state

The entry for a row corresponding to state  $q$  and the column corresponding to input  $a$ , is the state  $\delta(q, a)$



## Transition Table

eg: Consider a DFA;  $Q = \{q_0, q_1, q_2, q_3\}$

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

$$q_0 = q_0$$

$$F = \{q_0\}$$

$$\delta = Q \times \Sigma \rightarrow Q$$

# Transition Table



A transition table for above DFA is as follows:

$\delta$	0	1
* $\rightarrow$ q <sub>0</sub>	q <sub>2</sub>	q <sub>1</sub>
q <sub>1</sub>	q <sub>3</sub>	q <sub>0</sub>
q <sub>2</sub>	q <sub>0</sub>	q <sub>3</sub>
q <sub>3</sub>	q <sub>1</sub>	q <sub>2</sub>

This DFA accepts strings having both an even number of 0's & even number of 1's



## Transition Diagram

A transition diagram of a DFA is a graphical representation where

- For each state in  $Q$ , there is a node represented by circle
- For each state  $q$  in  $Q$  and each input  $a$  in  $\Sigma$ , if  $\delta(q, a) = p$  then there is an arc from node  $q$  to  $p$  labeled 'a' in the transition diagram
- If more than one input symbol causes the transition from state  $q$  to  $p$ , then arc from  $q$  to  $p$  is labeled by a list of those symbols

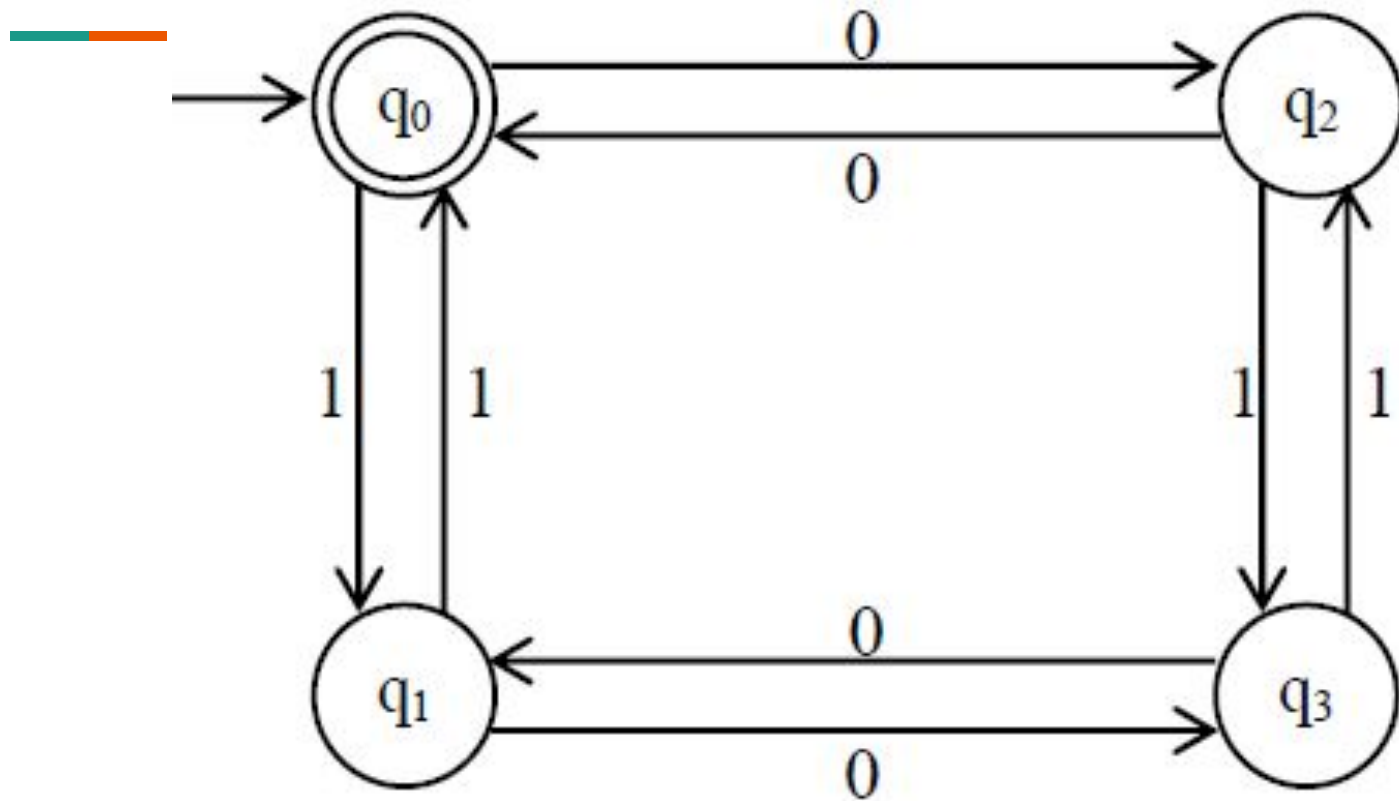




## Transition Diagram

- The start state is labeled by an arrow into it coming from nowhere
- The final or accepting state is marked by double circle

For the earlier example, the transition diagram is as follows





## String Processing by DFA

Suppose  $a_1, a_2, \dots, a_n$  is a sequence of input symbols

We start out with the DFA in its start state,  $q_0$

We also use the transition function  $\delta$  for this

Say  $\delta(q_0, a_1) = q_1$ , which is the state that the DFA enters after processing the first input symbol  $a_1$



## String Processing by DFA

We then process the next input symbol  $a_2$ , by evaluating  $\delta(q_1, a_2)$ ; suppose this state is  $q_2$

We continue in this manner, finding states  $q_3, q_4, \dots, q_n$  such that  $\delta(q_{i-1}, a_i) = q_i$  for each  $i$

If  $q_n$  is a member of  $F$ , then input  $a_1, a_2, \dots, a_n$  is accepted, if not, it is rejected



## Extended Transition Function of DFA( $\hat{\delta}$ )

The extended transition function of DFA, denoted by  $\hat{\delta}$  is a transition function that takes two arguments as input, one is the state  $q$  of  $Q$  and another is a string  $w \in \Sigma^*$ , and generates a state  $p \in Q$

This state  $p$  is one that the automaton reaches when starting in state  $q$  and processing the sequence of inputs  $w$  i.e.  $\hat{\delta}(q, w) = p$



## Extended Transition Function of DFA

We define  $\hat{\delta}$  by induction on length of input string as follows:

**Basis step:**  $\hat{\delta}(q, \epsilon) = q$ . i.e. if we are in state  $q$  and read no inputs, then we are still in state  $q$

**Induction:** Let  $w$  be a string from  $\Sigma^*$  such that  $w = xa$ , where  $x$  is substring of  $w$  without last symbol and  $a$  is the last symbol of  $w$ , then

$$\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$$



## Extended Transition Function of DFA

Thus, to compute  $\hat{\delta}(q, w)$ , we first compute  $\hat{\delta}(q, x)$ , the state the automaton is in after processing all but last symbol of  $w$

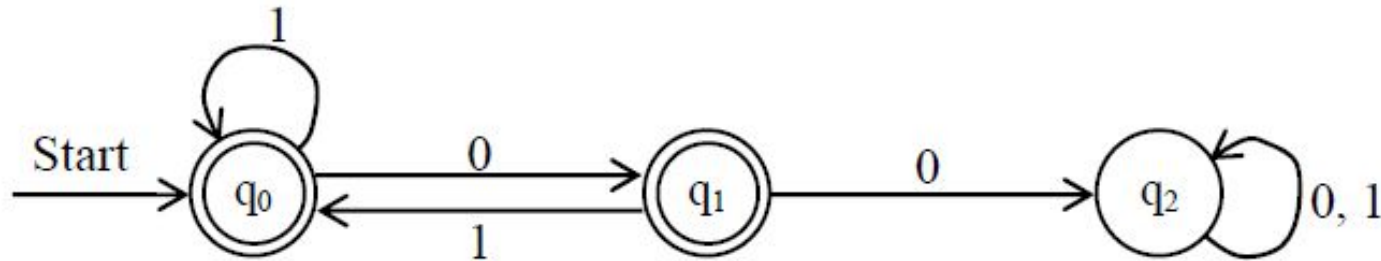
Suppose this state is  $p$ , i.e.  $\hat{\delta}(q, x) = p$

Then,  $\hat{\delta}(q, w)$  is what we get by making a transition from state  $p$  on input  $a$ , the last symbol of  $w$

i.e.  $\hat{\delta}(q, w) = \delta(p, a)$

## Extended Transition Function of DFA

Consider the following DFA:



Steps to compute  $\delta(q_0, 1001)$  is as follows



$$= \delta(\delta(q_0, 100), 1)$$

$$= \delta(\delta(\delta(q_0, 10), 0), 1)$$

$$= \delta(\delta(\delta(\delta(q_0, 1), 0), 0), 1)$$

$$= \delta(\delta(\delta(\delta(\delta(q_0, \epsilon), 1), 0), 0), 0), 1)$$

$$= \delta(\delta(\delta(\delta(q_0, 1), 0), 0), 1)$$

$$= \delta(\delta(\delta(q_0, 0), 0), 1)$$

$$= \delta(\delta(q_1, 0), 1)$$

$$= \delta(q_2, 1)$$

$$= q_2, \text{ not accepted.}$$



## Practice Question

(Q) Compute  $\hat{\delta}(q_0, 1010)$  for the above DFA



## Language of DFA

The **language** of DFA  $M = (Q, \Sigma, \delta, q_0, F)$ , denoted by  $L(M)$ , is a set of strings over  $\Sigma^*$  that are accepted by  $M$

$$\text{i.e. } L(M) = \{w \mid \hat{\delta}(q_0, w) = p \in F\}$$

That is, the language of a DFA is the set of all strings  $w$  that takes the DFA from the start state to one of the accepting states

The language of a DFA is called **regular language**

A **string  $x$  is accepted** by a DFA  $(Q, \Sigma, \delta, q_0, F)$  if  $\hat{\delta}(q, x) = p \in F$



## Practice Questions

(Q) Construct transition table and transition diagram for a DFA:

(a) for the language that accepts all the strings over  $\Sigma = \{a, b\}$  that do not end with ba

(b) for the language that accepts all strings over  $\Sigma = \{0, 1\}$  ending with 3 consecutive 0's

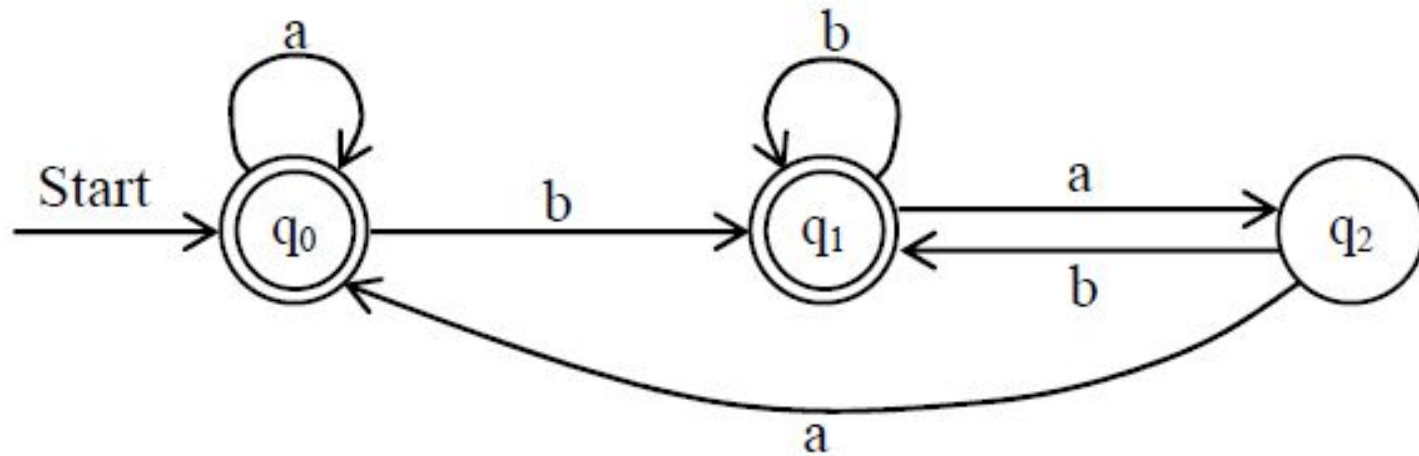
(c) for the language over  $\Sigma = \{a, b\}$  that accepts the strings {baa, ab, abb}



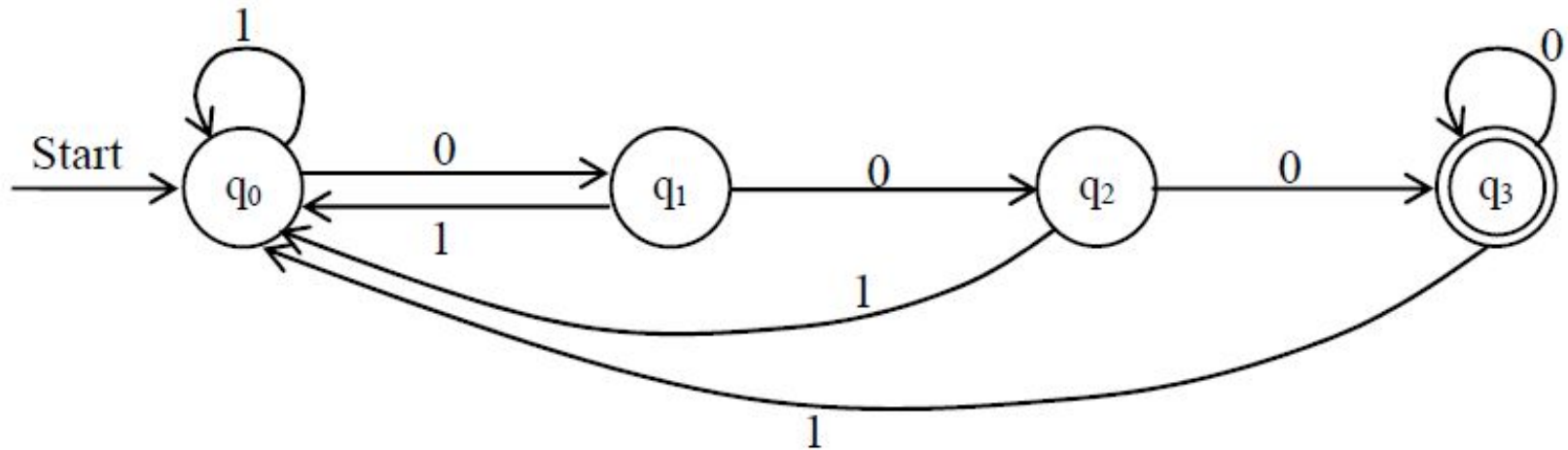
## Practice Questions

- (d) for the language over  $\Sigma = \{0, 1\}$  that accepts strings with zero or more consecutive 1's
- (e) for the language over  $\Sigma = \{0, 1\}$  that accepts the strings  $\{1, 01\}$
- (f) for the language over  $\Sigma = \{a, b\}$  that accepts the strings ending with abb

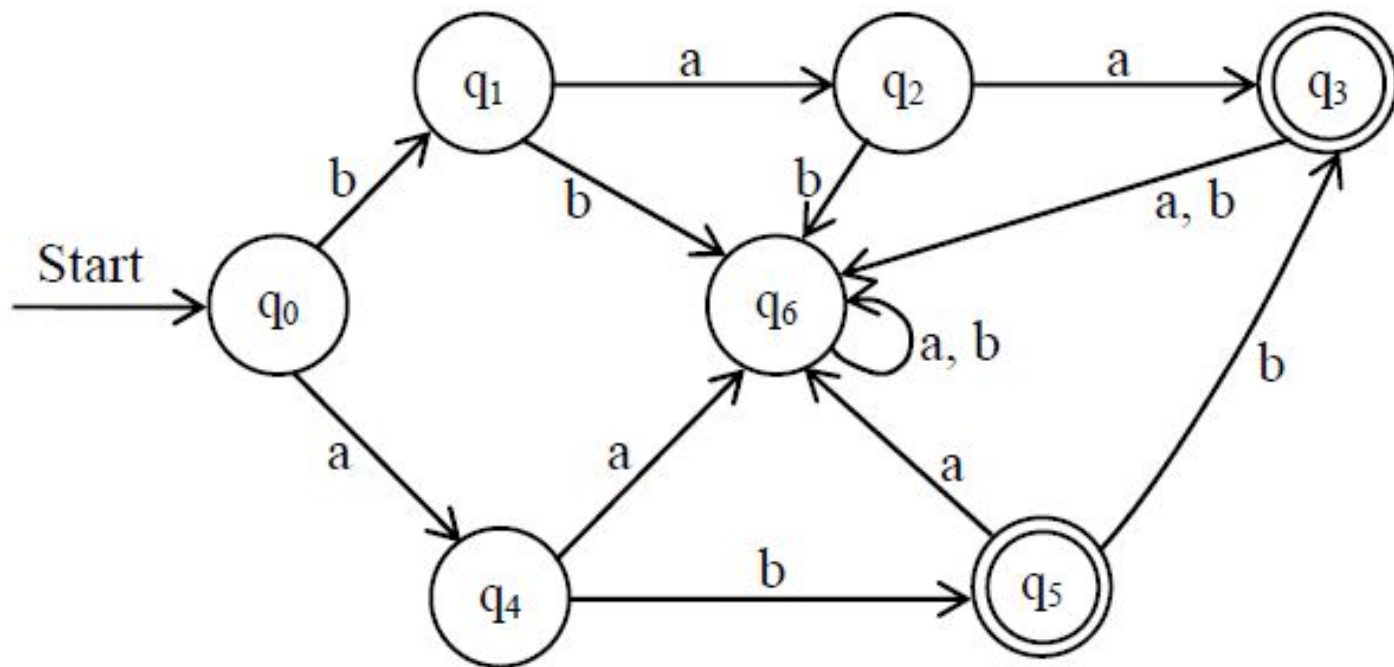
Ans (a):



Ans (b):

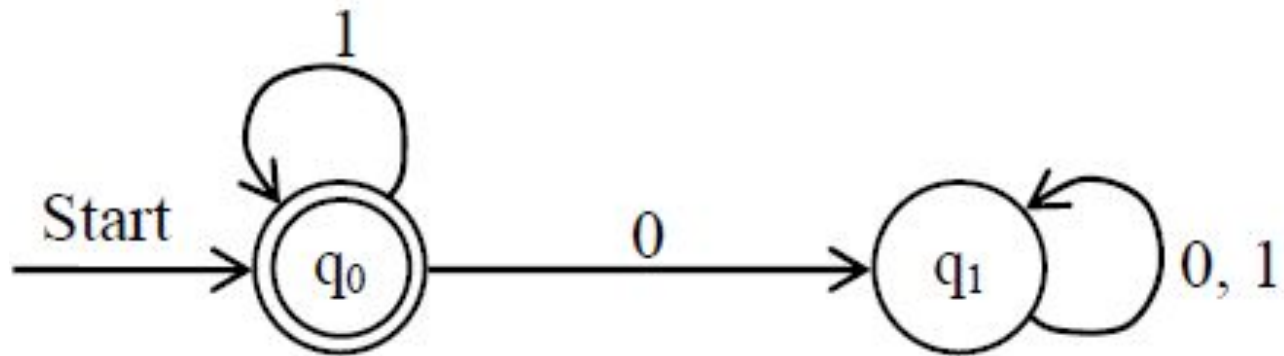


Ans (c):

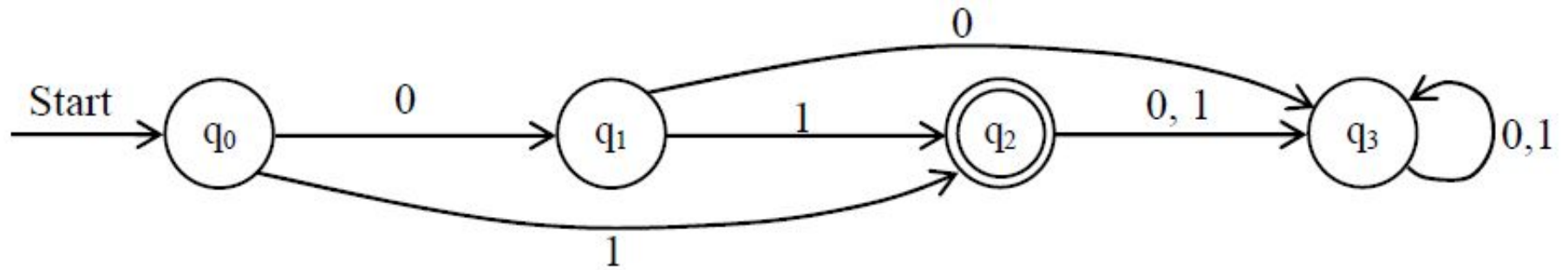




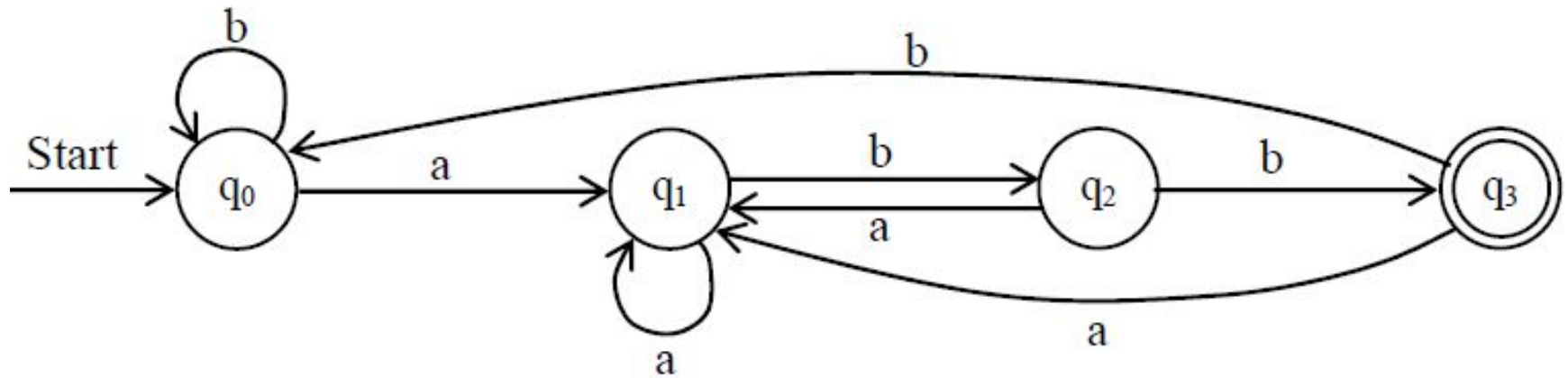
Ans (d):



Ans (e):



Ans (f):



# Non-Deterministic Finite Automata (NFA)



A non-deterministic finite automaton is a mathematical model that consists of:

A finite set of states  $Q$

A finite set of input symbols (alphabets)  $\Sigma$

A transition function that maps (state, symbol) pair to a set of states

A state  $q_0 \in Q$ , that is distinguished as start state

A set of states  $F$  distinguished as accepting/final states,  $F \subseteq Q$



## Non-Deterministic Finite Automata

Thus, NFA is defined by a quintuple  $(Q, \Sigma, \delta, q_0, F)$  where  $\delta$  is  $Q \times \Sigma \rightarrow 2^Q$

Unlike DFA, a transition function in NFA may take the NFA from one state to several states just with a single input

## Non-Deterministic Finite Automata

eg:

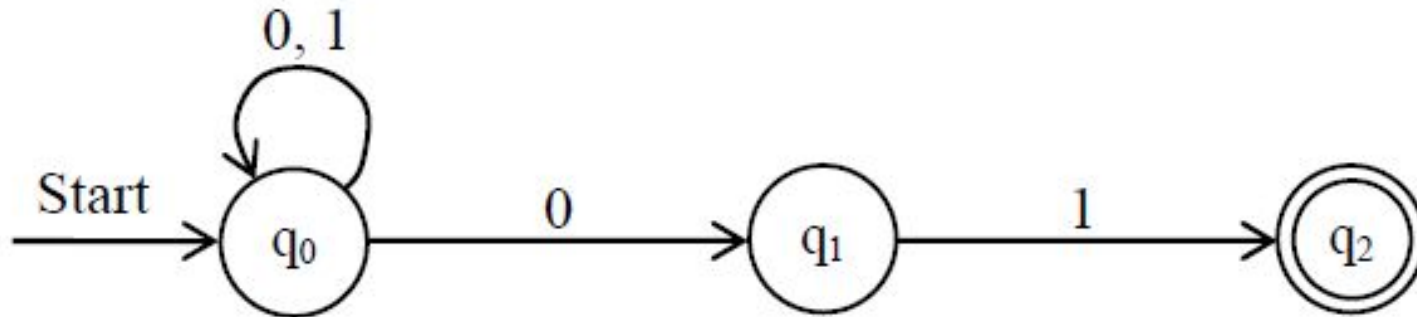


Fig: - NFA accepting all strings that end in 01.



## Non-Deterministic Finite Automata

Here, from state  $q_1$ , there is no arc for input symbol 0 and no arc out of  $q_2$  for 0 & 1

In an NFA, there may be no arc out of a state for an input symbol, or there may be more than one arc

While in a DFA, there is exactly one arc out of each state for each input symbol



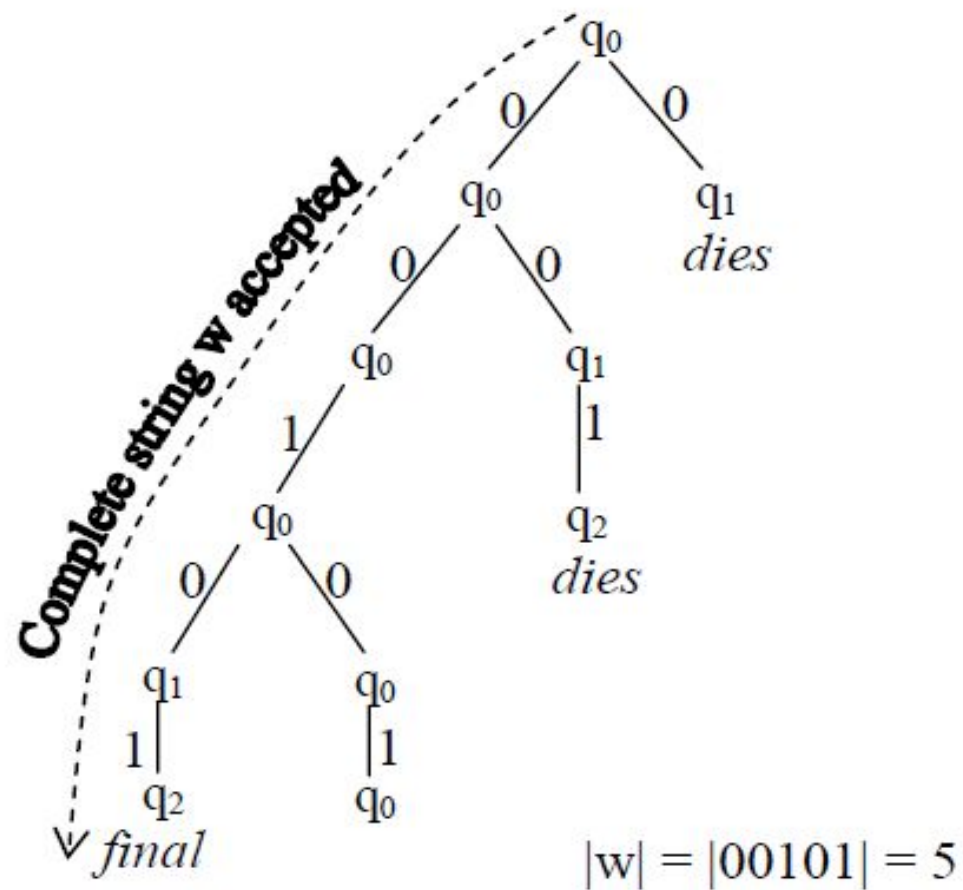
## Non-Deterministic Finite Automata

The transition function  $\delta$  takes a state in  $Q$  and an input symbol in  $\Sigma$  as arguments and returns a subset of  $Q$

The main difference between an NFA and a DFA is what  $\delta$  returns

In an NFA,  $\delta$  returns a set of states and in case of a DFA it returns a single state





$$Q = \{q_0, q_1, q_2\}$$

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

$$q_0 = \{q_0\}$$

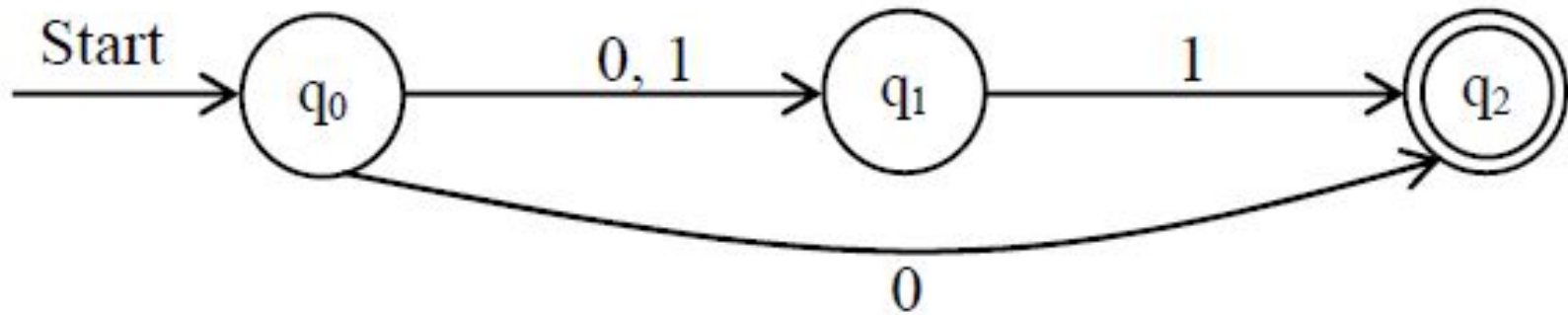
$$F = \{q_2\}$$

Transition table:

$\delta:$	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
$q_1$	$\{\phi\}$	$\{q_2\}$
$*q_2$	$\{\phi\}$	$\{\phi\}$

## Non-Deterministic Finite Automata

eg: NFA over  $\{0, 1\}$  accepting strings  $\{0, 01, 11\}$





## Non-Deterministic Finite Automata

Transition table:

$\delta:$	0	1
$\rightarrow q_0$	$\{q_0, q_2\}$	$\{q_1\}$
$q_1$	$\{\phi\}$	$\{q_2\}$
$*q_2$	$\{\phi\}$	$\{\phi\}$

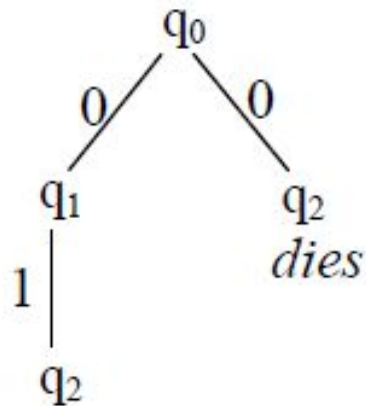


## Non-Deterministic Finite Automata

Following are the computation trees for this NFA for the strings 01 and 0110

## Non-Deterministic Finite Automata

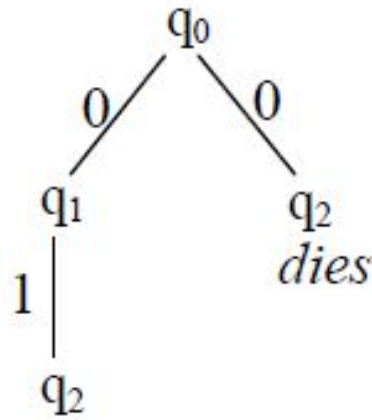
Computation tree for 01;



*Final, so 01 is accepted*

## Non-Deterministic Finite Automata

Computation tree for 0110



*dies, so 0110 is not accepted*



## Extended Transition Function of NFA

The extended transition function  $\hat{\delta}$  takes a state  $q$  and a string of input symbol  $w$  and returns the set of states that the NFA is in if it starts in state  $q$  and processes the string  $w$

Formally, we define  $\hat{\delta}$  for an NFA's transition function  $\delta$  using induction as follows



## Extended Transition Function of NFA

**Basis step:**  $\hat{\delta}(q, \varepsilon) = \{q\}$ , i.e. without reading any input symbols, we are only in the state we began in

**Induction:** Suppose  $w$  is a string from  $\Sigma^*$  such that  $w = xa$ , where  $x$  is a substring of  $w$  without last symbol  $a$ . Also suppose  $\hat{\delta}(q, x) = \{p_1, p_2, p_3, \dots, p_k\}$ . Let

$$\bigcup_{i=1}^k \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$$





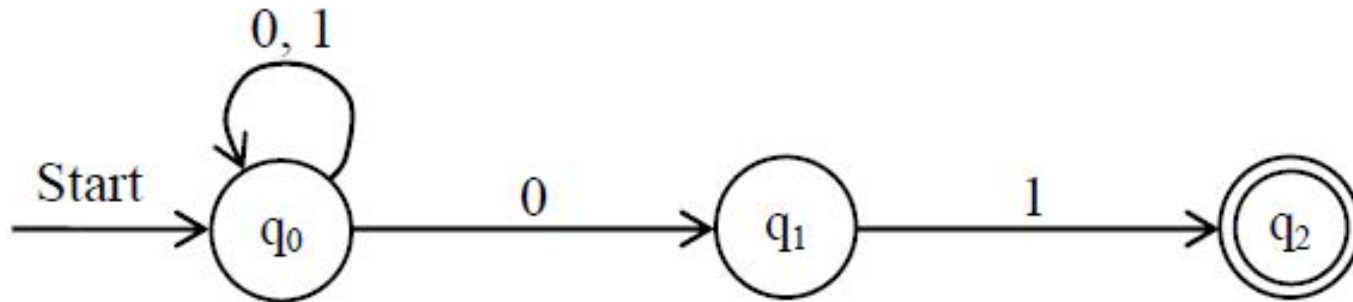
## Extended Transition Function of NFA

Then,  $\hat{\delta}(q, w) = \{r_1, r_2, r_3, \dots, r_m\}$

Thus, we compute  $\hat{\delta}(q, w)$  by first computing  $\hat{\delta}(q, x)$ , and by then following any transition from any of these states that is labeled a

## Extended Transition Function of NFA

Consider the NFA



Computing  $\hat{\delta}(q_0, 01101)$

*Solution:*

$$\hat{\delta}(q_0, 01101)$$

$$\hat{\delta}(q_0, \epsilon) = \{q_0\}$$

$$\hat{\delta}(q_0, 0) = \{q_0, q_1\}$$

$$\hat{\delta}(q_0, 01) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$$

$$\hat{\delta}(q_0, 011) = \delta(q_0, 1) \cup \delta(q_2, 1) = \{q_0\} \cup \{\phi\} = \{q_0\}$$

$$\hat{\delta}(q_0, 0110) = \delta(q_0, 0) = \{q_0\} \cup \{q_1\} = \{q_0, q_1\}$$

$$\hat{\delta}(q_0, 01101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$$



## Extended Transition Function of NFA

The result of above computation returns the set of states  $\{q_0, q_2\}$ , which includes the accepting state  $q_2$  of NFA

So, the string 01101 is accepted by the NFA



## Language of NFA

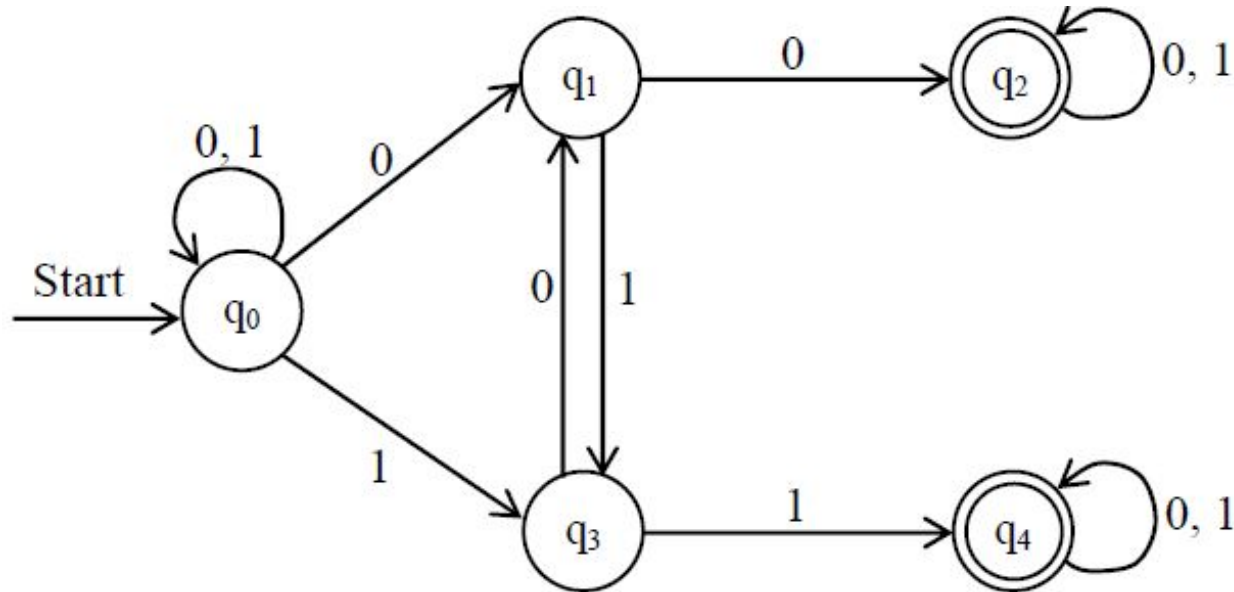
The language of NFA  $M = (Q, \Sigma, \delta, q_0, F)$ , denoted by  $L(M)$  is

$$L(M) = \{w \mid \delta(q_0, w) \cap F \neq \emptyset\}$$


i.e.  $L(M)$  is the set of strings  $w$  in  $\Sigma^*$  such that  $(q_0, w)$  contains at least one state accepting state

## Example

Design an NFA for the language over  $\{0, 1\}$  that have at least two consecutive 0's or 1's and compute for acceptance of string 10110.



## Example


$$\hat{\delta}(q_0, 10110)$$

Start with starting state

$$\hat{\delta}(q_0, \epsilon) = \{q_0\}$$

$$\hat{\delta}(q_0, 1) = \{q_0, q_3\}$$

$$\hat{\delta}(q_0, 10) = \delta(q_0, 0) \cup \delta(q_3, 0) = \{q_1, q_0\} \cup \{q_1\} = \{q_1, q_0\}$$

$$\hat{\delta}(q_0, 101) = \delta(q_1, 1) \cup \delta(q_0, 1) = \{q_3\} \cup \{q_3\} = \{q_3\}$$

$$\hat{\delta}(q_0, 1011) = \delta(q_3, 1) = \{q_4\}$$

$$\hat{\delta}(q_0, 10110) = \delta(q_4, 0) = \{q_4\} = \{q_4\} \rightarrow \text{ACCEPTED}$$



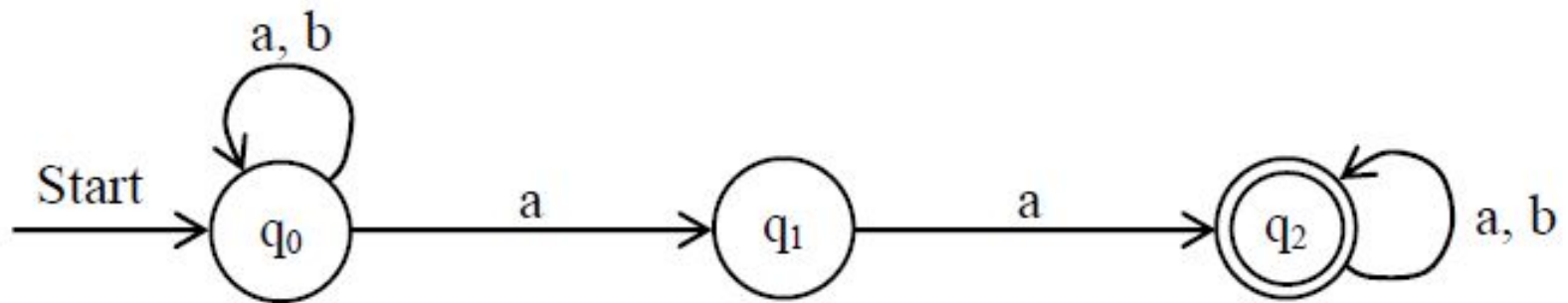
## Practice Questions

(Q) Construct NFA for the following languages:

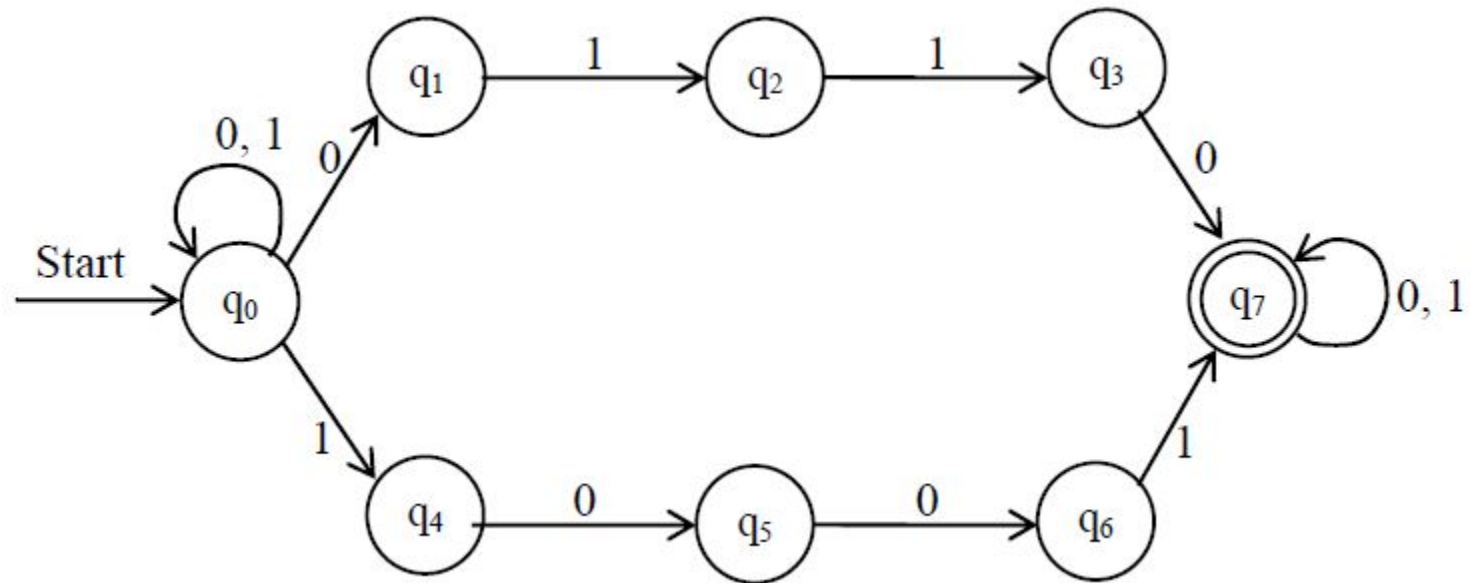
- (a) NFA over  $\{a, b\}$  that accepts strings having  $aa$  as substring
- (b) NFA for strings over  $\{0, 1\}$  that contain substring  $0110$  or  $1001$
- (c) NFA over  $\{a, b\}$  that have “a” as one of the last 3 characters
- (d) NFA over  $\{a, b\}$  that accepts strings starting with  $a$  and ending with  $b$



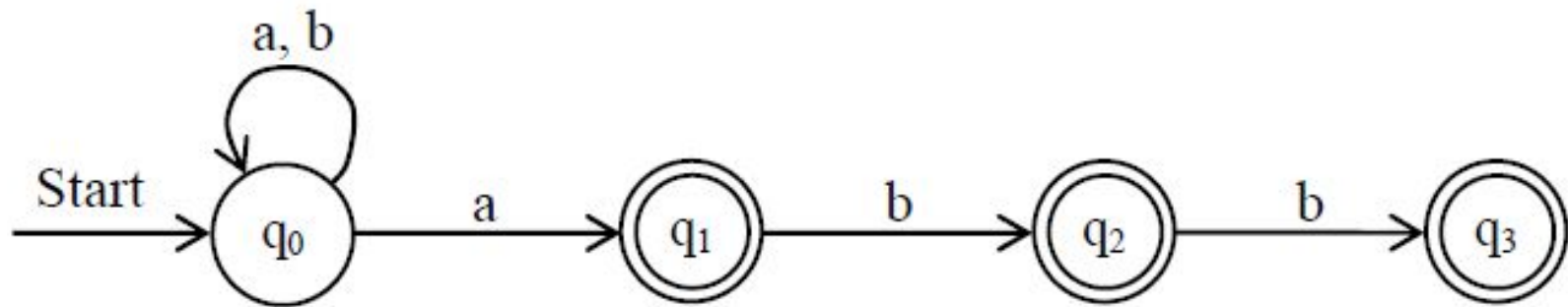
Ans (a):



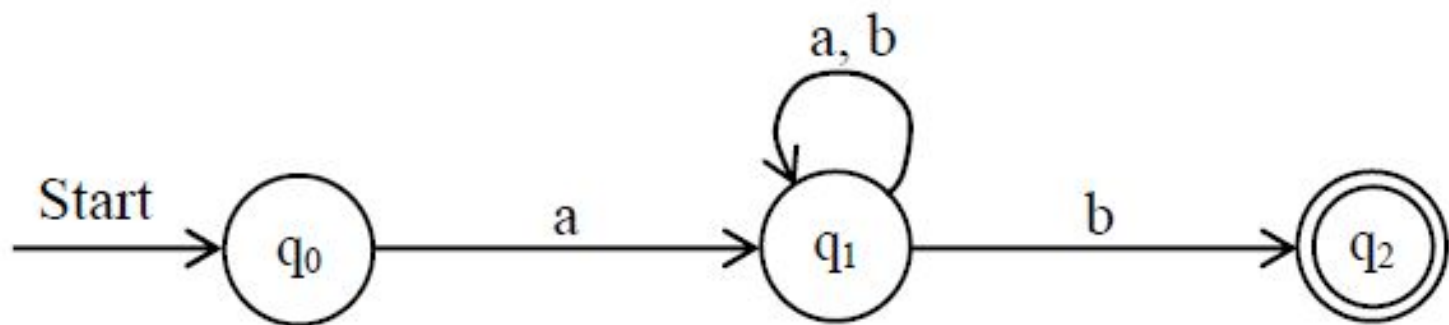
Ans (b):



Ans (c):



Ans (d):





## Equivalence of NFA and DFA

Although there are many languages for which NFA is easier to construct than DFA, it can be proved that every language that can be described by some NFA can also be described by some DFA

The DFA has more transitions than NFA and in worst case the smallest DFA can have  $2^n$  state while the smallest NFA for the same language has only  $n$  states



## Equivalence of NFA and DFA

We now show that DFAs and NFAs accept exactly the same set of languages

That is, non-determinism does not make a finite automaton more powerful

To show that NFAs and DFAs accept the same class of language, we show: **Any language accepted by an NFA can also be accepted by some DFA**



## Equivalence of NFA and DFA

For this we describe an algorithm that takes any NFA and converts it into a DFA that accepts the same language

The algorithm is called **subset construction algorithm**

The key idea behind the algorithm is that the equivalent DFA simulates the NFA by keeping track of the possible states it could be in



## Equivalence of NFA and DFA

Each state of DFA corresponds to a subset of the set of states of the NFA, hence the name of the algorithm

If NFA has  $n$  states, the DFA can have  $2^n$  states (at most), although it usually has many less





## Subset Construction Algorithm

Conversion of an NFA  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$  into an equivalent DFA  $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$  is done as follows:

$\Rightarrow$  The start state of  $D$  is the set of start states of  $N$  i.e. if  $q_0$  is start state of  $N$  then  $D$  has start state as  $\{q_0\}$ .



## Subset Construction Algorithm

$\Rightarrow Q_D$  is set of subsets of  $Q_N$  i.e.  $Q_D$  is power set of  $Q_N$ . So if  $Q_N$  has  $n$  states then  $Q_D$  will have  $2^n$  states. Often, not all these states are accessible from start state of  $Q_D$ , and hence they can be eliminated. So,  $Q_D$  will have less than  $2^n$  states.

$\Rightarrow F_D$  is set of subsets  $S$  of  $Q_N$  such that  $S \cap F_N \neq \emptyset$  i.e.  $F_D$  is all sets of  $N$ 's states that include at least one accepting state of  $N$ .



## Subset Construction Algorithm

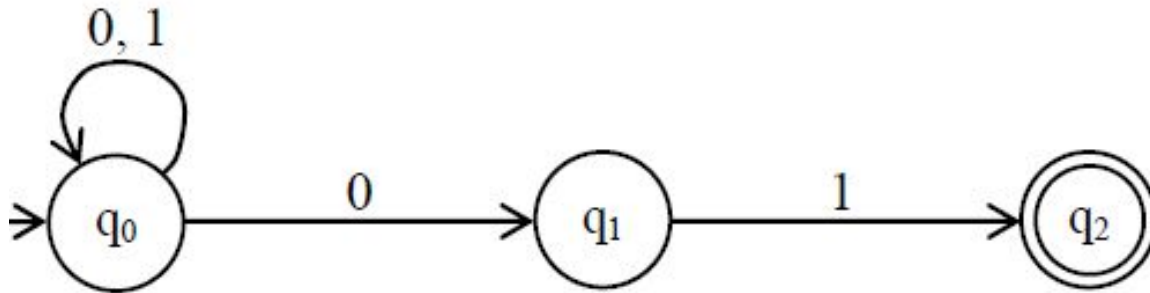
⇒ For each set  $S \subseteq Q_N$  and for each input  $a \in \Sigma$ ,

$$\delta_D(S, a) = \bigcup_{p \text{ in } S} \delta_N(p, a)$$

i.e. for any state  $\{q_0, q_1, q_2, \dots, q_k\}$  of the DFA and any input  $a$ , the next state is the set of all states of the NFA that can result as next states if the NFA is in any of the states  $q_0, q_1, q_2, \dots, q_k$  when it reads  $a$ .

## Subset Construction Algorithm

eg: Given NFA





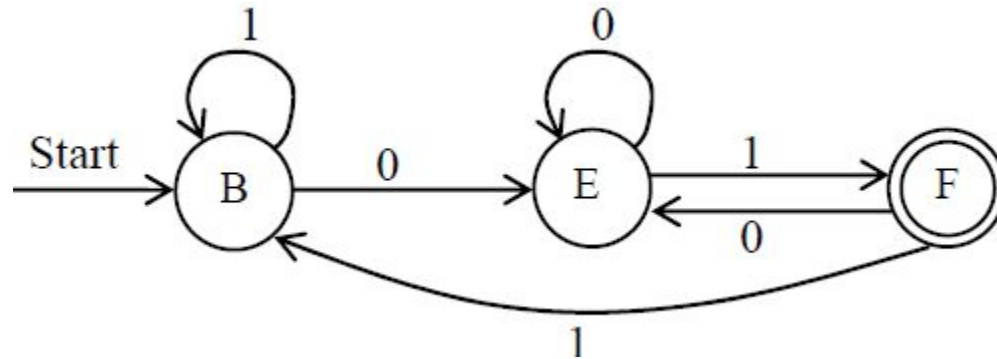
Transition  
table for  
DFA

	$\delta:$	0	1
A	$\phi$	$\phi$	$\phi$
B	$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
C	$\{q_1\}$	$\phi$	$\{q_2\}$
D	$*\{q_2\}$	$\phi$	$\phi$
E	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
F	$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
G	$*\{q_1, q_2\}$	$\phi$	$\{q_2\}$
H	$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

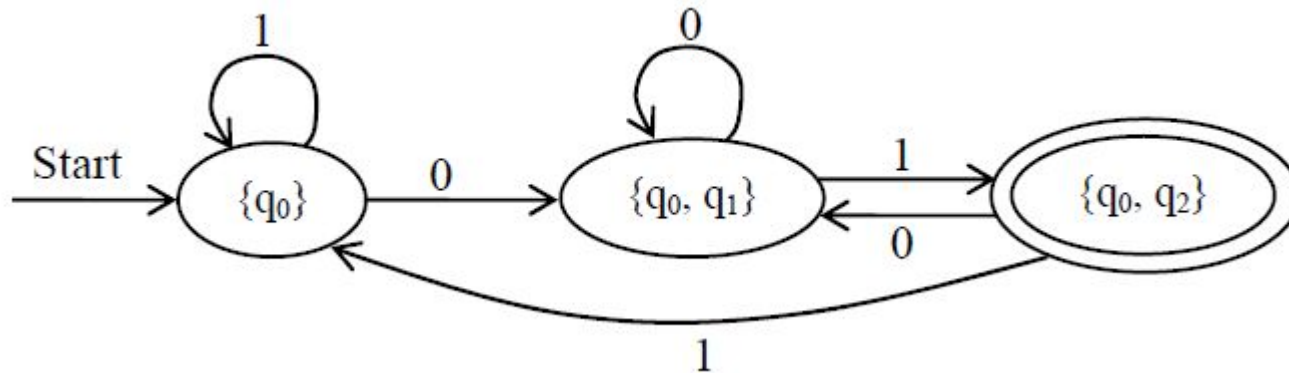
The same table can be represented with renaming the state on table entry as

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

The equivalent DFA is (states not reachable from start state are removed):

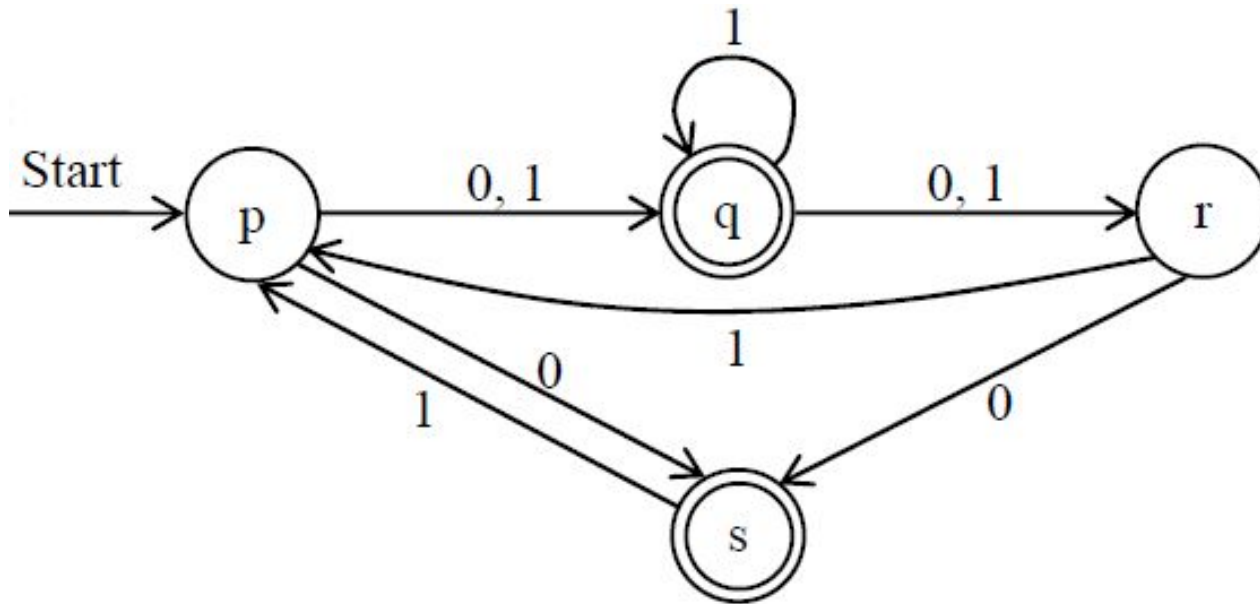


or



## Practice Question

(Q) Convert the given NFA to DFA:





## Practice Question

Ans:

Transition table  
of DFA

$\delta:$	0	1
$\phi$	$\phi$	$\phi$
$\rightarrow \{p\}$	$\{q, s\}$	$\{q\}$
$*\{q, s\}$	$\{r\}$	$\{p, q, r\}$
$*\{q\}$	$\{r\}$	$\{q, r\}$
$\{r\}$	$\{s\}$	$\{p\}$
$*\{p, q, r\}$	$\{q, r, s\}$	$\{p, q, r\}$
$*\{q, r\}$	$\{r, s\}$	$\{p, q, r\}$
$*\{s\}$	$\phi$	$\{p\}$
$*\{q, r, s\}$	$\{r, s\}$	$\{p, q, r\}$
$*\{r, s\}$	$\{s\}$	$\{p\}$

# Theorem 1



If  $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$  is the DFA constructed from NFA  $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$  by the subset construction, then  $L(D) = L(N)$ .

**Proof:** We prove by induction on  $|w|$  that  $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ .

**Basis step:** Let  $|w| = 0$ ; i.e.,  $w = \epsilon$ . By the basis definitions of  $\hat{\delta}$  for DFAs and NFAs both  $\hat{\delta}_D(\{q_0\}, \epsilon)$  and  $\hat{\delta}_N(q_0, \epsilon)$  are  $\{q_0\}$ .

# Theorem 1

**Induction:** Let  $w$  be of length  $n+1$ , and assume the statement for length  $n$ .

Break  $w$  up as  $w = xa$ , where  $a$  is the final symbol of  $w$ . By the inductive hypothesis,  $\hat{\delta}_D(\{q_0\}, x) = \hat{\delta}_N(q_0, x)$ .

Let both these sets of  $N$ 's states be  $\{p_1, p_2, \dots, p_k\}$

From the definition of  $\hat{\delta}$  for NFAs,

$$\hat{\delta}_N(q_0, w) = \bigcup_{i=1}^k \delta_N(p_i, a) \quad \dots(i)$$

# Theorem 1



And, from subset construction, we can write

$$\delta_D(\{p_1, p_2, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_N(p_i, a) \quad \dots(\text{ii})$$

Now, using (ii) and the fact that  $\hat{\delta}_D(\{q_0\}, x) = \{p_1, p_2, \dots, p_k\}$

$$\hat{\delta}_D(\{q_0\}, w) = \delta_D(\hat{\delta}_D(\{q_0\}, x), a) = \delta_D(\{p_1, p_2, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_N(p_i, a) \quad \dots(\text{iii})$$



## Theorem 1

Thus, equations (i) and (iii) demonstrate that  $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ .

Since  $D$  and  $N$  both accept  $w$  if and only if  $\hat{\delta}_D(\{q_0\}, w)$  or  $\hat{\delta}_N(q_0, w)$ , respectively, contain a state in  $F_N$ , we can conclude that  $L(D) = L(N)$ .



## Theorem 2

A language  $L$  is accepted by some DFA if and only if  $L$  is accepted by some NFA.

### Proof:

If part (A language  $L$  is accepted by some DFA if  $L$  is accepted by some NFA) is the subset construction and theorem 1.



## Theorem 2

Only if part (A language  $L$  is accepted by some NFA if  $L$  is accepted by some DFA)

Here, we have to convert the DFA into an identical NFA.

Consider we have a DFA  $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ .

This DFA can be interpreted as a NFA having the transition diagram with exactly one choice of transition for any input.

## Theorem 2



Let NFA  $N = (Q_N, \Sigma, \delta_N, q_0', F_N)$  be equivalent to  $D$ .

Where  $Q_N = Q_D$ ,  $F_N = F_D$ ,  $q_0' = q_0$  and  $\delta_N$  is defined by the rule

if  $\delta_D(p, a) = q$  then  $\delta_N(p, a) = \{q\}$ .

Then, to show if  $L$  is accepted by  $D$ , then it is also accepted by  $N$ , it is sufficient to show, for any string  $w \in \Sigma^*$ ,  $\hat{\delta}_D(q_0, w) = \hat{\delta}_N(q_0', w)$





## Theorem 2

We can prove this fact using induction

**Basis step:** Let  $|w| = 0$  i.e.  $w = \varepsilon$

$$\hat{\delta}_D(q_0, w) = \hat{\delta}_D(q_0, \varepsilon) = q_0$$

$$\hat{\delta}_N(q_0, w) = \hat{\delta}_D(q_0', \varepsilon) = \{q_0\}$$

Hence,  $\hat{\delta}_D(q_0, w) = \hat{\delta}_N(q_0', w)$  for  $|w| = 0$  is true



## Theorem 2

**Induction:** Let  $|w| = n + 1$  &  $w = xa$ ; where,  $|x| = n$  &  $|a| = 1$ ,  $a$  being the last symbol.

Let the inductive hypothesis is that it is true for  $x$ .

So,  $\hat{\delta}_D(q_0, x) = p$ ,  $\hat{\delta}_N(q_0', x) = \{p\}$ . That is,  $\hat{\delta}_D(q_0, x) = \hat{\delta}_N(q_0', x)$

Now,

$\hat{\delta}_D(q_0, w) = \hat{\delta}_D(q_0, xa) = \delta_D(\hat{\delta}_D(q_0, x), a) = \delta_D(p, a)$  [hypothesis] =  $r$ ,

say



## Theorem 2

And,

$$\hat{\delta}_N(q_0', w) = \hat{\delta}_N(q_0', xa) = \delta_N(\hat{\delta}_N(q_0', x), a) = \delta_D(\{p\}, a) [\text{hypothesis}] = r$$

This shows  $\hat{\delta}_D(q_0, w) = \hat{\delta}_N(q_0', w)$  and hence proves the theorem



## Finite Automata With Epsilon-Transitions ( $\epsilon$ -NFA)

$\epsilon$ -NFA another extension of finite automaton

$\epsilon$ -NFA allows a transition on  $\epsilon$ , the empty string, so that an NFA could make a transition spontaneously without receiving an input symbol

Similar to NFA, this feature does not expand the class of language that can be accepted by finite automata, but it does give some added programming convenience

## $\epsilon$ -NFA



An NFA with  $\epsilon$ -transitions is defined by five tuples  $(Q, \Sigma, \delta, q_0, F)$ , where;

$Q$  = set of finite states

$\Sigma$  = set of finite input symbols

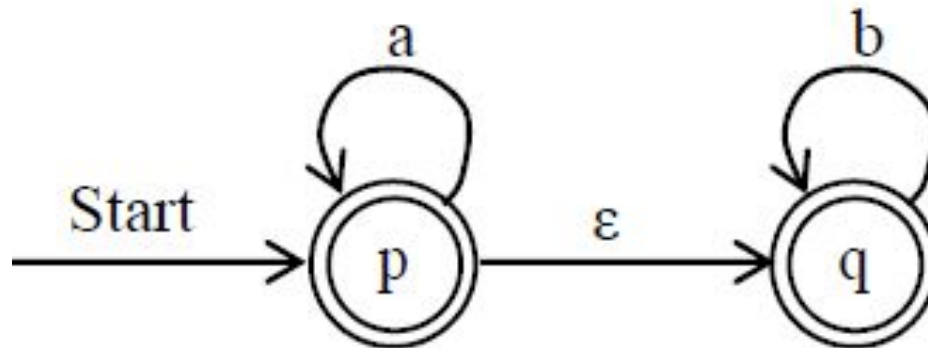
$q_0$  = initial state,  $q_0 \in Q$

$F$  = set of final states;  $F \subseteq Q$

$\delta$  = a transition function that maps  $Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$

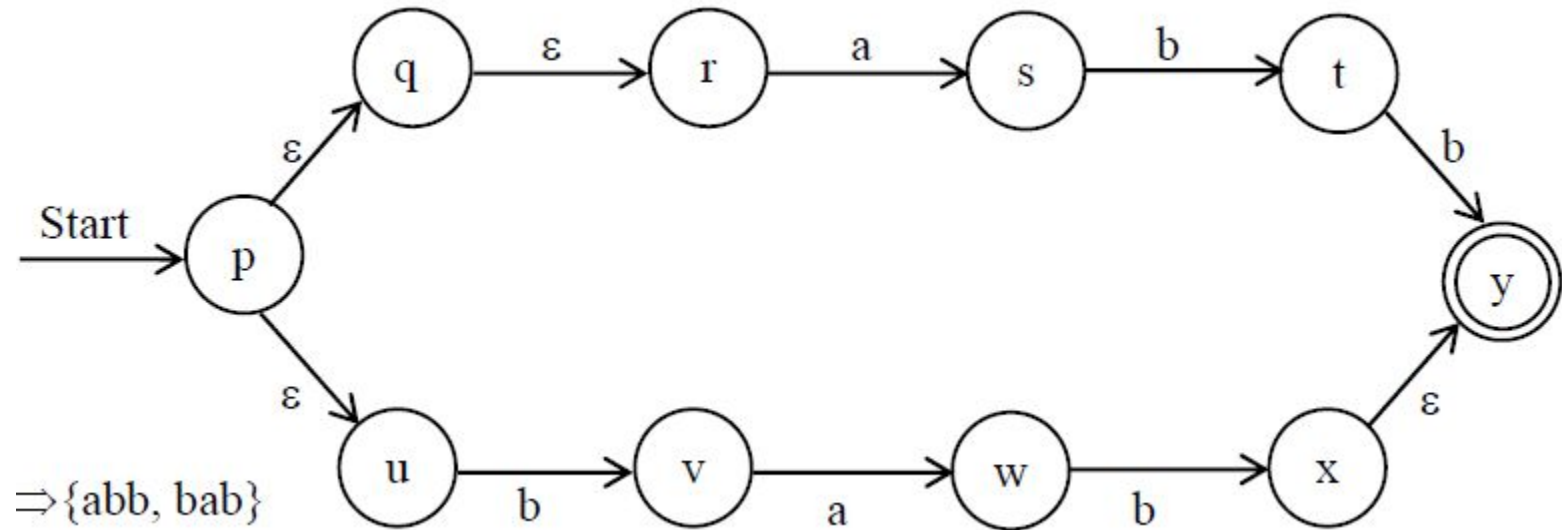
## $\epsilon$ -NFA

**eg:**  $\Sigma = \{a, b\}$ , accepts strings with 0 or more a's followed by 0 or more b's



## $\epsilon$ -NFA

eg: Language = {abb, bab}





## $\epsilon$ -Closure

$\epsilon$ -closure of a state 'q' can be obtained by following all transitions out of q that are labeled  $\epsilon$

After we get to another state by following  $\epsilon$ , we follow the  $\epsilon$ -transitions out of those states and so on, eventually finding every state that can be reached from q along any path whose arcs are all labeled  $\epsilon$





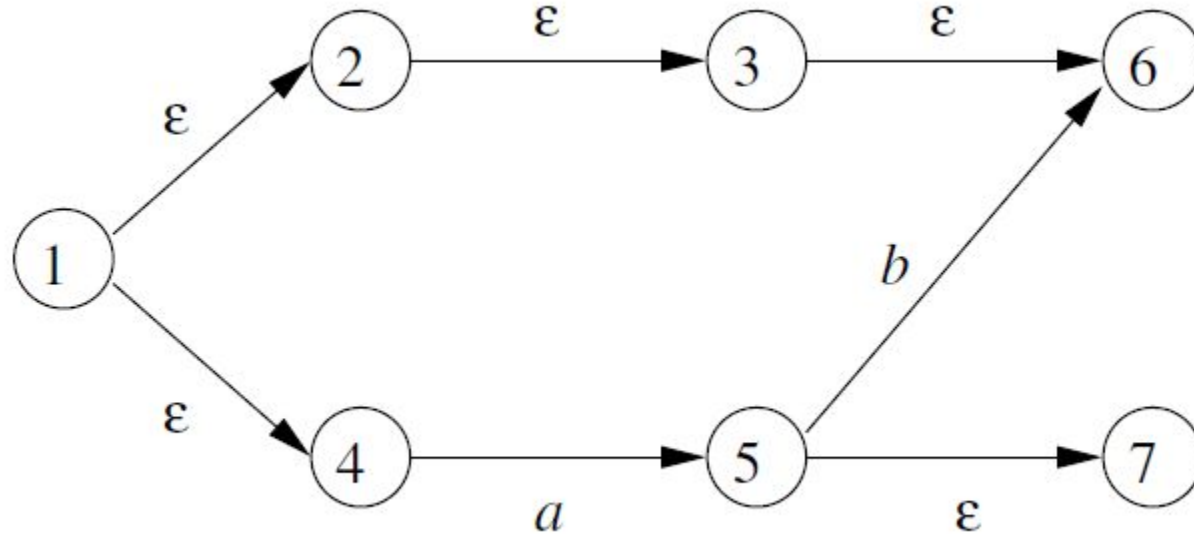
## $\epsilon$ -Closure

Formally, we can define  $\epsilon$ -closure of the state  $q$  as:

**Basis step:** State  $q$  is in  $\epsilon$ -closure( $q$ )

**Induction:** If state  $p$  is reached with  $\epsilon$ -transition from state  $q$ ,  $p$  is in  $\epsilon$ -closure ( $q$ ). And if there is an arc from  $p$  to  $r$  labeled  $\epsilon$ , then  $r$  is in  $\epsilon$ -closure ( $q$ ) and so on.

eg:



$$\epsilon\text{-closure}(1) = \{1, 2, 3, 4, 6\}$$

$$\epsilon\text{-closure}(5) = \{5, 7\}$$

$$\epsilon\text{-closure}(6) = \{6\}$$



## Extended Transition Function of $\epsilon$ -NFA

The extended transition function of  $\epsilon$ -NFA denoted by  $\hat{\delta}$ , is defined as:

**Basis Step:**  $\hat{\delta}(q, \epsilon) = \epsilon\text{-closure}(q)$

**Induction:** Let  $w = xa$  be a string where  $x$  is the substring of  $w$  without last the symbol  $a$  and  $a \in \Sigma$  but  $a \neq \epsilon$

Let  $\hat{\delta}(q, x) = \{p_1, p_2, \dots, p_k\}$  i.e.  $p_i$ 's are the states that can be reached from  $q$  following path labeled  $x$



## Extended Transition Function of $\epsilon$ -NFA

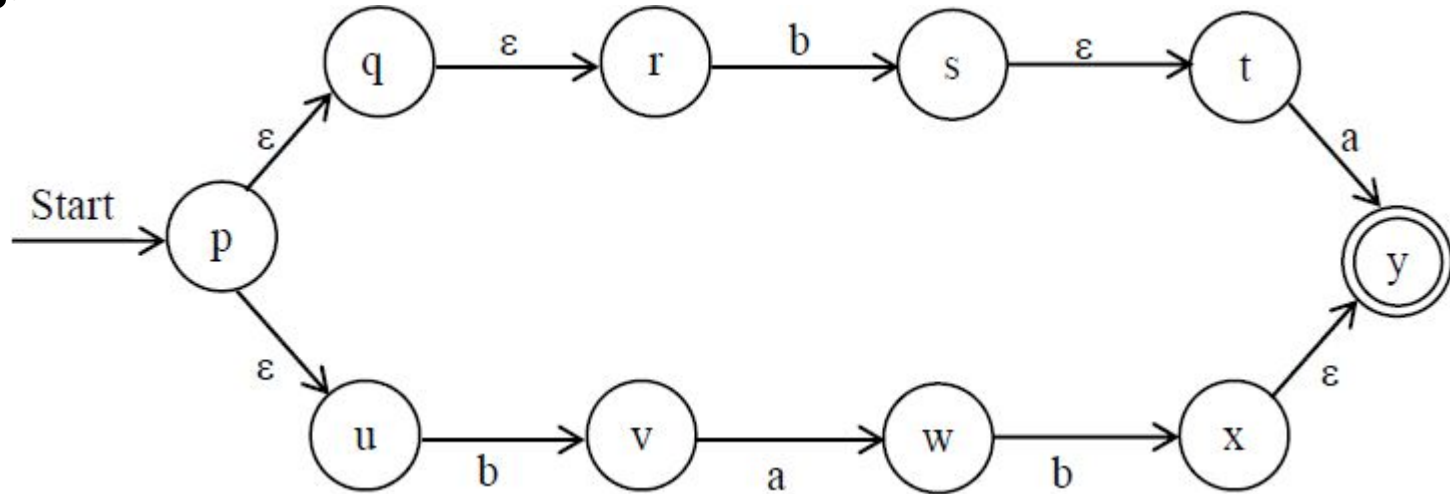
This path may end with one or more transitions labeled  $\epsilon$ , and may have other  $\epsilon$ -transitions as well

Also let 
$$\bigcup_{i=1}^k \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$$

Then,  $\hat{\delta}(q, w) = \epsilon\text{-closure}(\{r_1, r_2, \dots, r_m\})$

## Extended Transition Function of $\epsilon$ -NFA

eg:



# Extended Transition Function of $\epsilon$ -NFA

Computation of  $\hat{\delta}(p, ba)$ :

$$\hat{\delta}(p, \epsilon) = \epsilon\text{-closure}(p) = \{p, q, r, u\}$$

Compute for b:  $\delta(p, b) \cup \delta(q, b) \cup \delta(r, b) \cup \delta(u, b) = \{s, v\}$

$$\epsilon\text{-closure}(s) \cup \epsilon\text{-closure}(v) = \{s, t, v\}$$

Compute for a:  $\delta(s, a) \cup \delta(t, a) \cup \delta(v, a) = \{y, w\}$

$$\epsilon\text{-closure}(y) \cup \epsilon\text{-closure}(w) = \{y, w\}$$

The result contains state “y” so the string is accepted



## Removing $\epsilon$ -Transitions using $\epsilon$ -Closure

Using  $\epsilon$ -closure, we can convert an  $\epsilon$ -NFA to an equivalent NFA and an equivalent DFA

We first look at the equivalence of  $\epsilon$ -NFA and NFA

Any language accepted by an  $\epsilon$ -NFA can also be accepted by an NFA

The procedure given next converts an  $\epsilon$ -NFA to its equivalent NFA



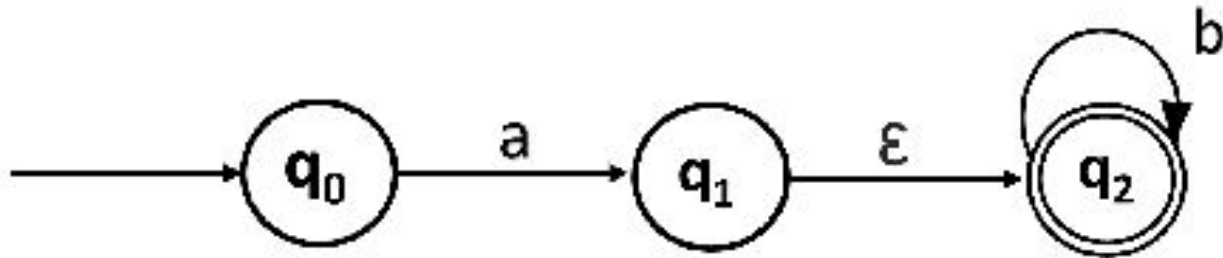
## Equivalence of NFA and $\epsilon$ -NFA

- (1) For each state, find out  $\epsilon$ -closure( $q_i$ ), where  $q_i \in Q$ .
- (2) For each resultant set of states obtained in step (1), find the result of performing transition on each input symbol.
- (3) Finally, for each resultant set of states obtained in step (2), find their  $\epsilon$ -closure.
- (4) By using the result, the transition table for required NFA can be built.



## Equivalence of NFA and $\epsilon$ -NFA

eg: Convert given  $\epsilon$ -NFA into equivalent NFA





## Equivalence of NFA and $\epsilon$ -NFA

Here,  $q_0$  is the start state

$$\begin{aligned}\delta_N(q_0, a) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), a)) \\ &= \epsilon\text{-closure}(\delta(\{q_0\}, a)) \\ &= \epsilon\text{-closure}(\{q_1\}) \\ &= \{q_1, q_2\}\end{aligned}$$



## Equivalence of NFA and $\epsilon$ -NFA

$$\begin{aligned}\delta_N(q_0, b) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), b)) \\ &= \epsilon\text{-closure}(\delta(\{q_0\}, b)) \\ &= \epsilon\text{-closure}(\emptyset) \\ &= \emptyset\end{aligned}$$



## Equivalence of NFA and $\epsilon$ -NFA

$$\begin{aligned}\delta_N(q_1, a) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), a)) \\ &= \epsilon\text{-closure}(\delta(\{q_1, q_2\}, a)) \\ &= \epsilon\text{-closure}(\emptyset) \\ &= \emptyset\end{aligned}$$



## Equivalence of NFA and $\epsilon$ -NFA

$$\begin{aligned}\delta_N(q_1, b) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_1), b)) \\ &= \epsilon\text{-closure}(\delta(\{q_1, q_2\}, b)) \\ &= \epsilon\text{-closure}(q_2) \\ &= \{q_2\}\end{aligned}$$



## Equivalence of NFA and $\epsilon$ -NFA

$$\begin{aligned}\delta_N(q_2, a) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), a)) \\ &= \epsilon\text{-closure}(\delta(\{q_2\}, a)) \\ &= \epsilon\text{-closure}(\emptyset) \\ &= \emptyset\end{aligned}$$



## Equivalence of NFA and $\epsilon$ -NFA

$$\begin{aligned}\delta_N(q_2, b) &= \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_2), b)) \\ &= \epsilon\text{-closure}(\delta(\{q_2\}, b)) \\ &= \epsilon\text{-closure}(\{q_2\}) \\ &= \{q_2\}\end{aligned}$$

$\epsilon$ -closure of  $q_1$  and  $q_2$  contain the final state  $q_2$ , hence they become final states in resultant NFA

# Equivalence of NFA and $\varepsilon$ -NFA

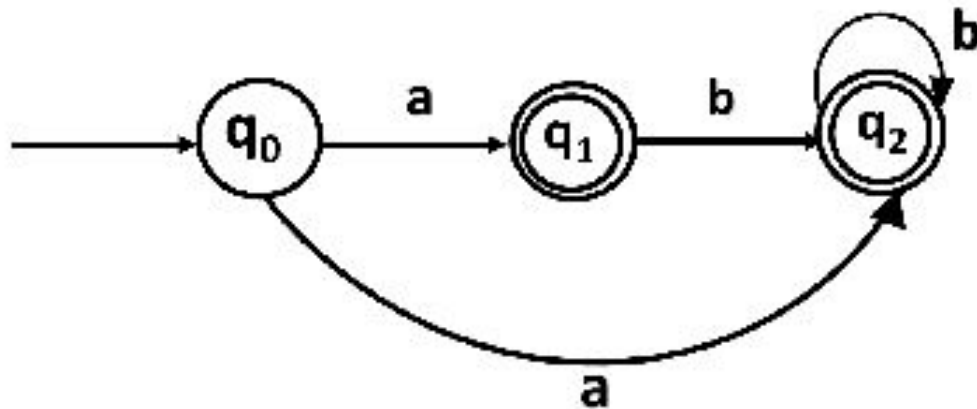
The computed transitions can be summarized in the following transition table:

States \ Inputs	a	b
$\rightarrow q_0$	$\{q_1, q_2\}$	$\emptyset$
$*q_1$	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\{q_2\}$



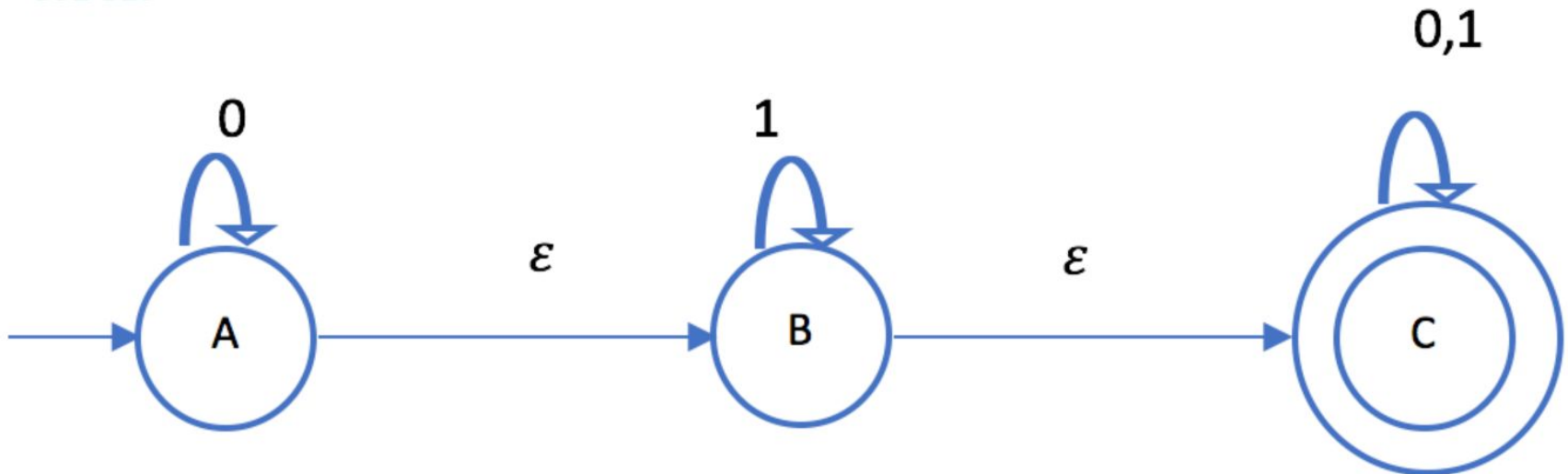
## Equivalence of NFA and $\epsilon$ -NFA

Resultant transition diagram:

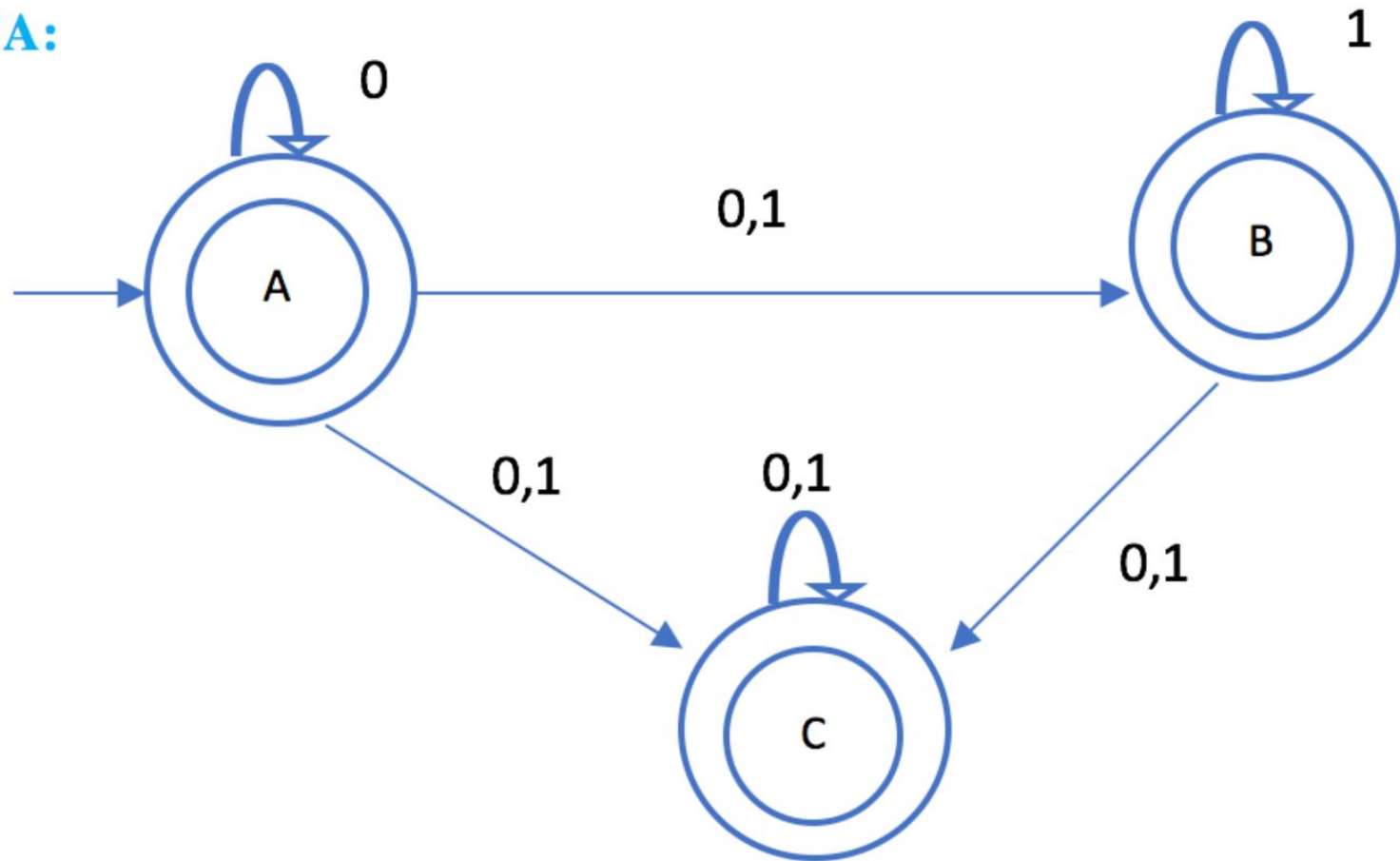


## Practice Question

(Q) Convert given  $\epsilon$ -NFA to NFA. Here,  $\Sigma = \{0, 1\}$



**NFA:**





## Equivalence of DFA and $\epsilon$ -NFA

Given any  $\epsilon$ -NFA, we can find a DFA that accepts the same language as it

That is, any  $\epsilon$ -NFA can be converted into an equivalent DFA

The construction we use is very close to the subset construction

The only difference is that we must incorporate  $\epsilon$ -transitions which we do through the mechanism of the  $\epsilon$ -closure

# Equivalence of DFA and $\epsilon$ -NFA



Put  $\epsilon$ -closure( $q_0$ ) as an unmarked state in Dstates;  $q_0$  = start state of

While there is an unmarked state T in Dstates  $\epsilon$ -NFA

Mark T

For each input symbol  $a \in \Sigma$

$U = \epsilon$ -closure( $\delta(T, a)$ )

If U is not in Dstates then

Add U as an unmarked state to Dstates

$Dtran[T, a] = U$



## Equivalence of DFA and $\epsilon$ -NFA

**Dstates** is the set of states of the DFA consisting of sets of states of the  $\epsilon$ -NFA

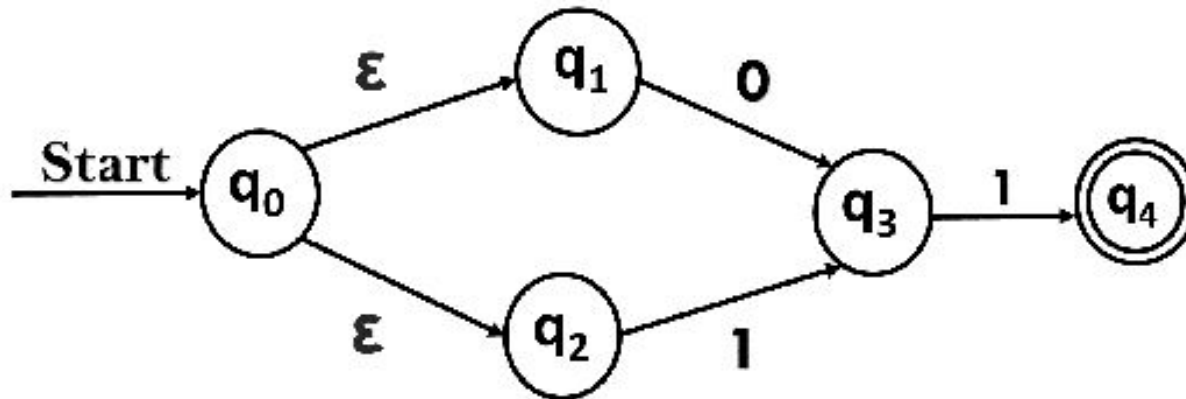
**Dtran** is the transition table of the DFA

The start state of DFA is  $\epsilon$ -closure( $q_0$ )

The final states of the DFA are the set of states containing at least one accepting state of NFA

## Equivalence of DFA and $\epsilon$ -NFA

eg: Convert the following  $\epsilon$ -NFA to equivalent DFA:



# Equivalence of DFA and $\epsilon$ -NFA

$\epsilon$ -closure( $q_0$ ) =  $\{q_0, q_1, q_2\}$  = state A (let)    A = start state

Dstates = {A}

(i) Mark A, Dstates = {**A**}

Dtran[A, 0] =  $\epsilon$ -closure ( $\delta(\{q_0, q_1, q_2\}, 0)$ )


=  $\epsilon$ -closure ( $\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)$ )

=  $\epsilon$ -closure ( $\{q_3\}$ )

=  $\{q_3\}$  = B (let);    Dstates = {**A**, B}



$$\text{Dtran}[A, 1] = \varepsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, 1))$$


$$= \varepsilon\text{-closure}(\delta((q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)))$$

$$= \varepsilon\text{-closure}(\{q_3\})$$

$$= \{q_3\} = B$$


(ii) Mark B, Dstates = **{A, B}**

$$\text{Dtran}[B, 0] = \varepsilon\text{-closure}(\delta(\{q_3\}, 0))$$

$$= \varepsilon\text{-closure}(\emptyset)$$

$$= \{\emptyset\} = C \text{ (let); } \text{Dstates} = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$$

$$Dtran[B, 1] = \varepsilon\text{-closure}(\delta(\{q_3\}, 1))$$


$$= \varepsilon\text{-closure}(\{q_4\})$$

$$= \{q_4\} = D \text{ (let); } Dstates = \{\mathbf{A}, \mathbf{B}, C, D\}; D = \text{final state}$$

(iii) Mark C,  $Dstates = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, D\}$


$$Dtran[C, 0] = Dtran[C, 1] = \varepsilon\text{-closure}(\emptyset)$$

$$= \{\emptyset\}$$

$$= C \text{ (C is known as a } \mathbf{dead\ state}, \text{ since}$$

there is no transition to move out of C)

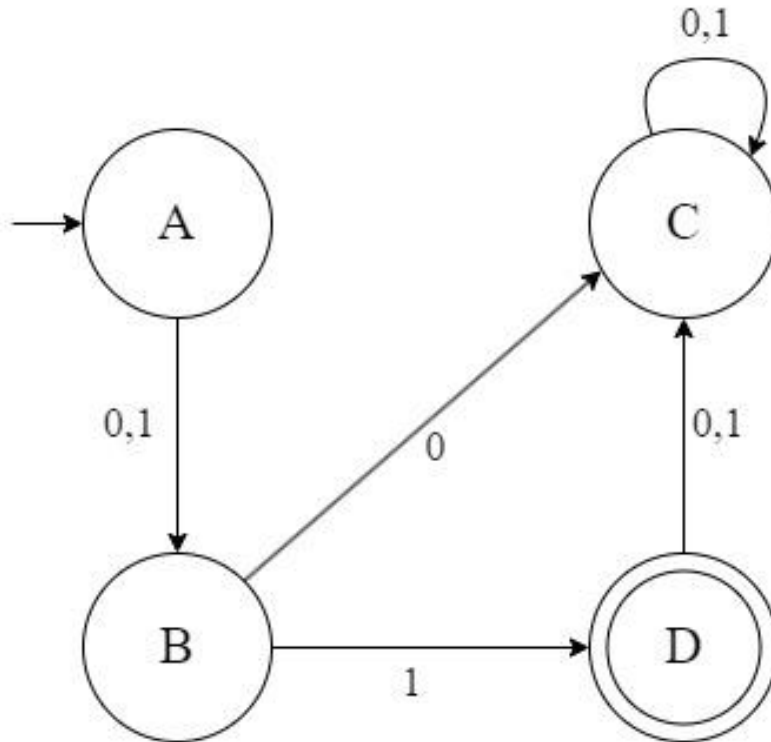
(iii) Mark D, Dstates = {**A**, **B**, **C**, **D**}


$$\begin{aligned} \text{Dtran}[D, 0] &= \varepsilon\text{-closure}(\delta(\{q_4\}, 0)) \\ &= \varepsilon\text{-closure}(\emptyset) \\ &= \{\emptyset\} = C \end{aligned}$$

$$\begin{aligned} \text{Dtran}[D, 1] &= \varepsilon\text{-closure}(\delta(\{q_4\}, 1)) \\ &= \varepsilon\text{-closure}(\emptyset) \\ &= \{\emptyset\} = C \end{aligned}$$

No unmarked states left

Required DFA:





# Finite State Machines with Output

- Moore Machine
- Mealy Machine



## 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



Moore machine can be described by 6 tuples  $(Q, q_0, \Sigma, O, \delta, \lambda)$  where,

$Q$ : finite set of states

$q_0$ : initial state of machine

$\Sigma$ : finite set of input symbols

$O$ : output alphabet

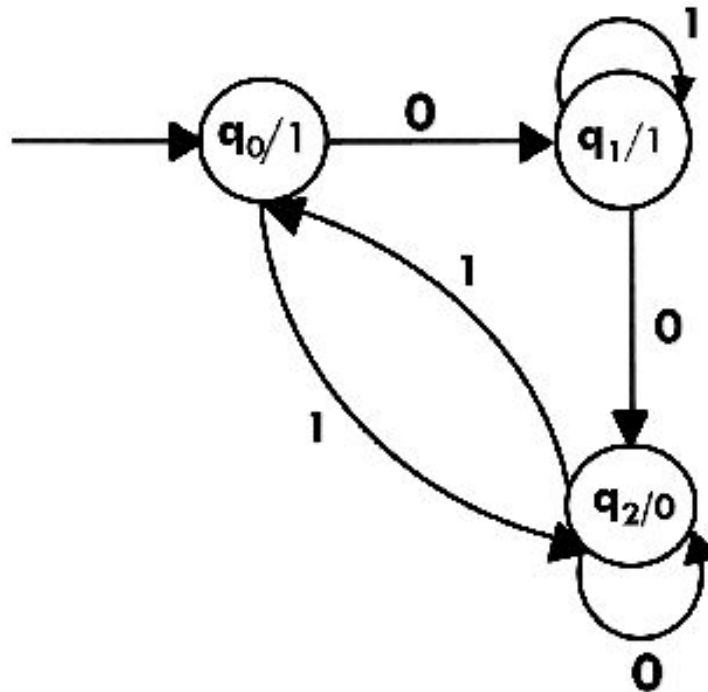
$\delta$ : transition function where  $Q \times \Sigma \rightarrow Q$

$\lambda$ : output function where  $Q \rightarrow O$

# Moore Machine

eg:

State diagram of a  
Moore Machine





Transition table:



Output is  
represented with  
each input state  
separated by “/”

Current State	Next State ( $\delta$ )		Output( $\lambda$ )
	0	1	
$q_0$	$q_1$	$q_2$	1
$q_1$	$q_2$	$q_1$	1
$q_2$	$q_2$	$q_0$	0

Output length for a Moore Machine is greater than input by 1



## Moore Machine

For the above machine,

If Input = 010,

Transition:  $\delta(q_0, 0) \Rightarrow \delta(q_1, 1) \Rightarrow \delta(q_1, 0) \Rightarrow q_2$

Output: 1110 (1 for  $q_0$ , 1 for  $q_1$ , again 1 for  $q_1$ , 0 for  $q_2$ )

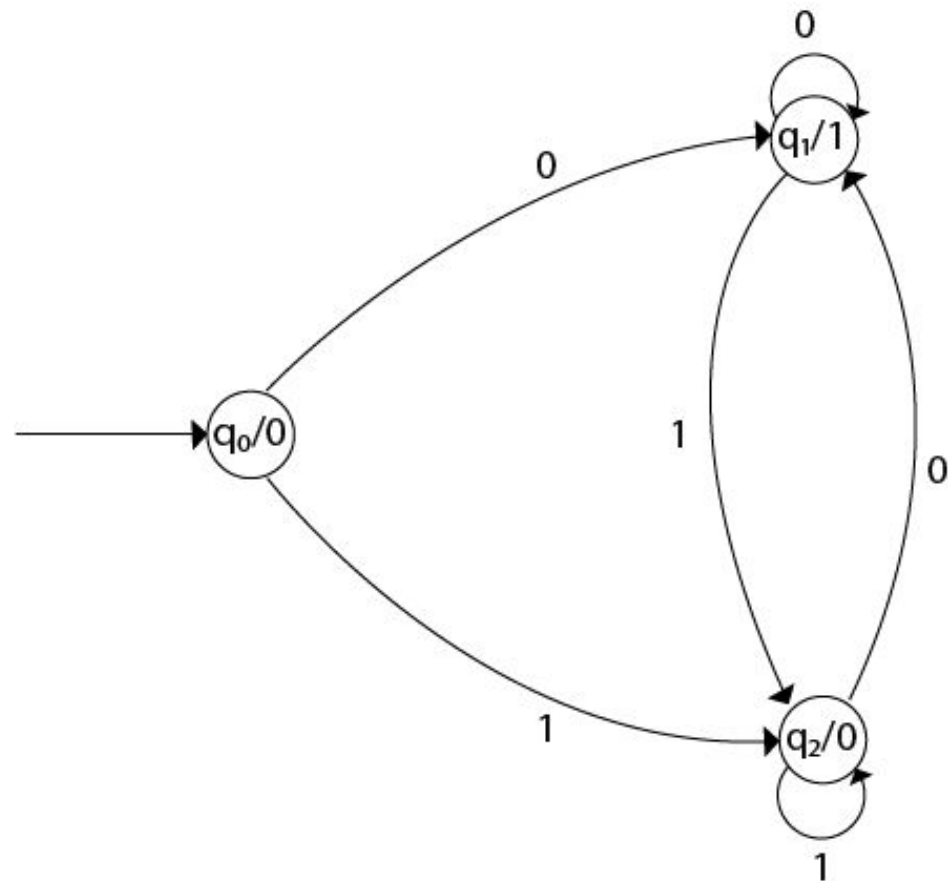


## Moore Machine


(Q) Design a Moore machine to generate 1's complement of a given binary number

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.

Transition  
diagram:



Transition table:  $Q = \{q_0, q_1, q_2\}$ ,  $\Sigma = \{0, 1\}$ ,  $O = \{0, 1\}$



Current State	$\delta$		$\lambda$
	0	1	Output
$\rightarrow q_0$	$q_1$	$q_2$	0
$q_1$	$q_1$	$q_2$	1
$q_2$	$q_1$	$q_2$	0

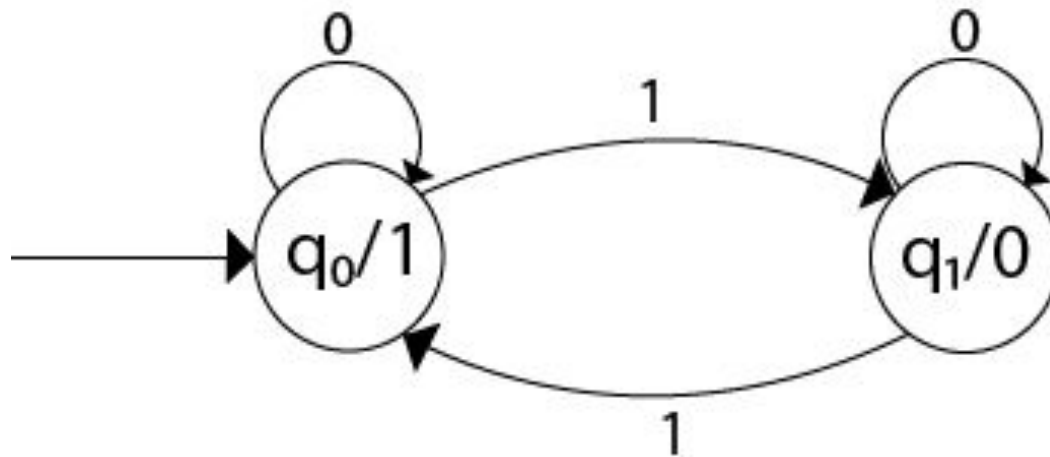


## Moore Machine

(Q) 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 are in the string and 0 otherwise.

## Moore Machine

Ans:





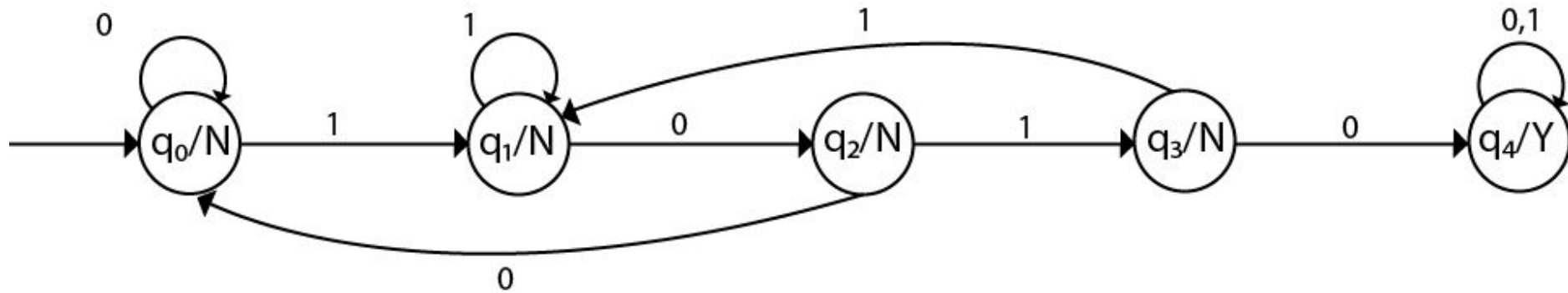
## Moore Machine

(Q) 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.



# Moore Machine

Ans:





## 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 “/”

# Mealy Machine



Mealy machine can be described by 6 tuples  $(Q, q_0, \Sigma, O, \delta, \lambda')$  where,

$Q$ : finite set of states

$q_0$ : initial state of machine

$\Sigma$ : finite set of input alphabets

$O$ : output alphabet

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

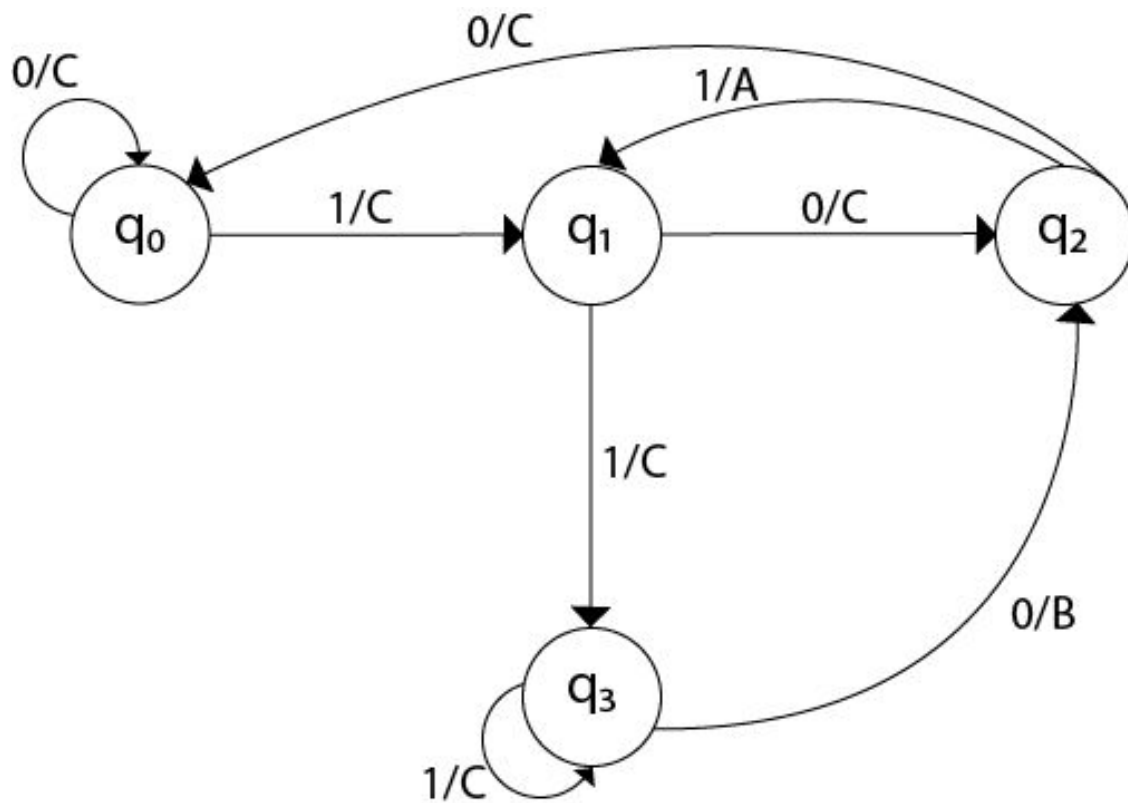
$\lambda'$ : output function where  $Q \times \Sigma \rightarrow O$



## Mealy Machine

(Q) Design a Mealy machine 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.

Ans:





## Mealy Machine

(Q) 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.

Ans:

