Unit-2

Introduction to Finite Automata

Finite Automata

- a mathematical (model) abstract machine that has a set of “states”
- and its “control” moves from state to state in response to external “inputs”.
- The control may be:
  - **Deterministic** → the automation can’t be in more than one state at any one time
  - **Non-Deterministic** → the automation t it may be in several states at once

This distinguishes the class of automata as DFA or NFA.

- The DFA, i.e. Deterministic Finite Automata can’t be in more than one state at a time.

- The NFA, i.e. Non-Deterministic Finite Automata can be in more than one state at a time.

- The finite state machines are used in applications in computer science and data networking.

For example:

finite-state machines 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.

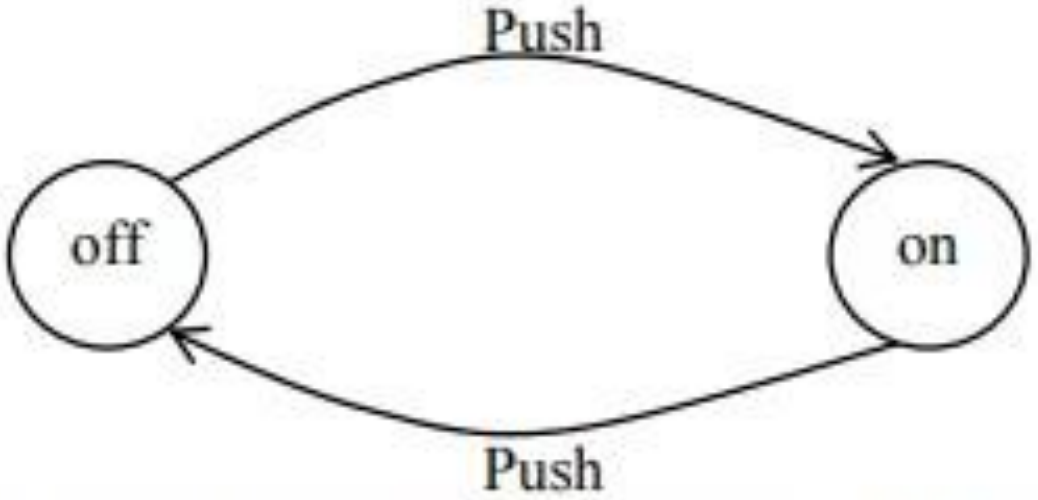


Fig: - Finite automaton modeling an on/off switch

Deterministic Finite Automata (DFA)

A deterministic finite automaton is defined by a quintuple (5-tuple) as $(Q, \sum, \delta, q0, F)$.

Where,

Q = Finite set of states,

$\sum$ = Finite set of input symbols,

δ = A transition function that maps $Q \times \sum \rightarrow Q$

q0 = A start state; $q0 \in Q$

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

General Notations of DFA

- There are two preferred notations for describing this class of automata;
  - Transition Table
  - Transition Diagram

Transition Table

- Transition table is a conventional, tubular representation of the transition function δ
- Takes the arguments from Q × ∑ & 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.
- The starting state in the table is represented by ➐ followed by the state i.e.
  ➐ q , for q being start state,
- The 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 δ (q, a).

Example: transition table

Consider a DFA;

Q = {q0, q1, q2, q3}

∑ = {0, 1}

q0 = q0

F = {q0}

δ = Q × ∑ → Q

Then the transition table for above DFA is

	0	1
* → q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

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

DFA accepting all strings over {0, 1} having substring 01

- Let, Q = {q0, q1, q2}, ∑ = {0, 1}, q0 = q0, F = { q1}

δ:	0	1
→ q_0	q_2	q_0
*q_1	q_1	q_1
q_2	q_2	q_1

Transition Diagram

- A transition diagram of a DFA is a graphical representation (or is a graph) where;

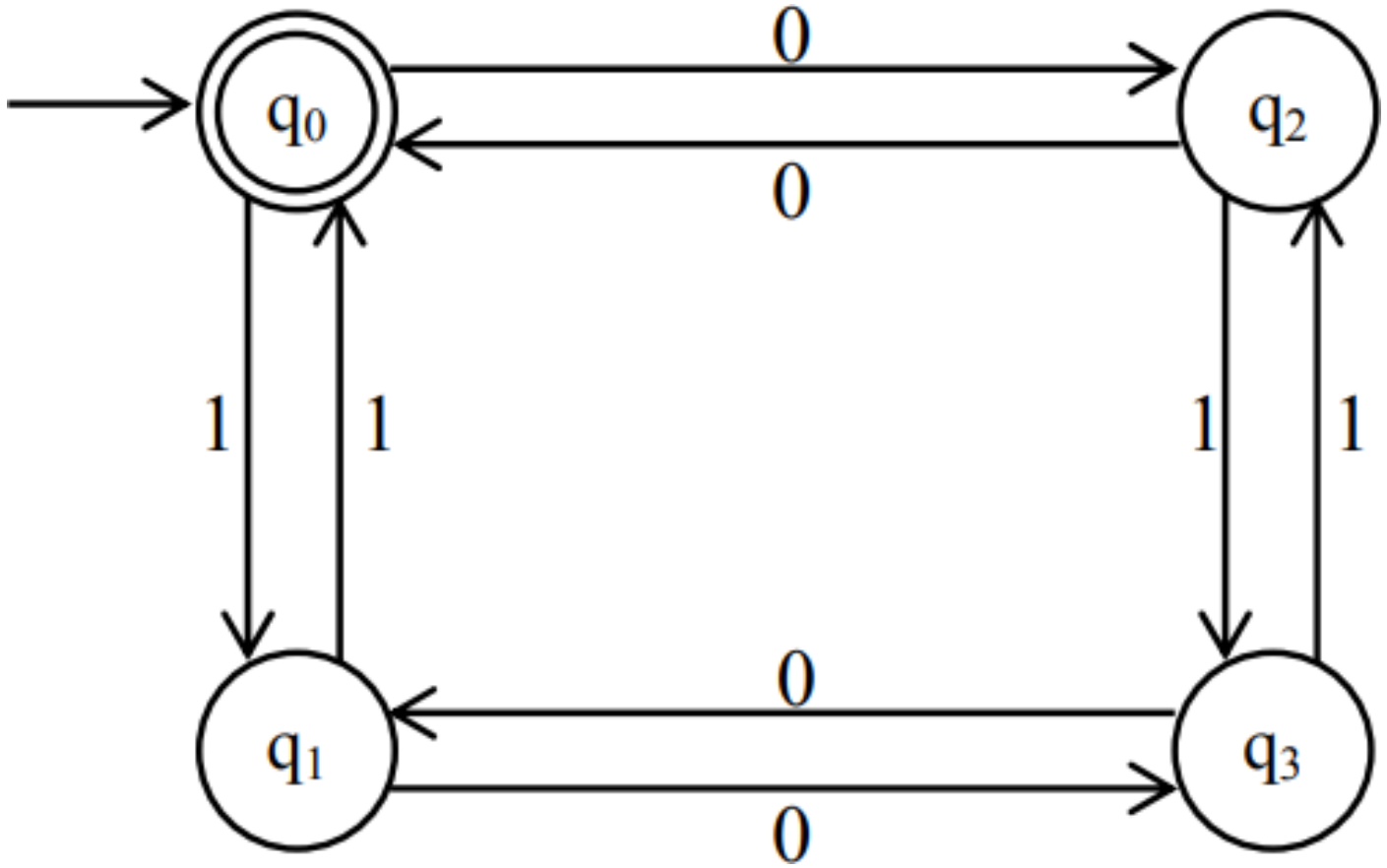
➢ For each state Q, there is a node represented by circle,

➢ For each state q in Q and each input a in ∑, if δ (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 cause the transition from state q to p then arc from q to p is labeled by a list of those symbols.

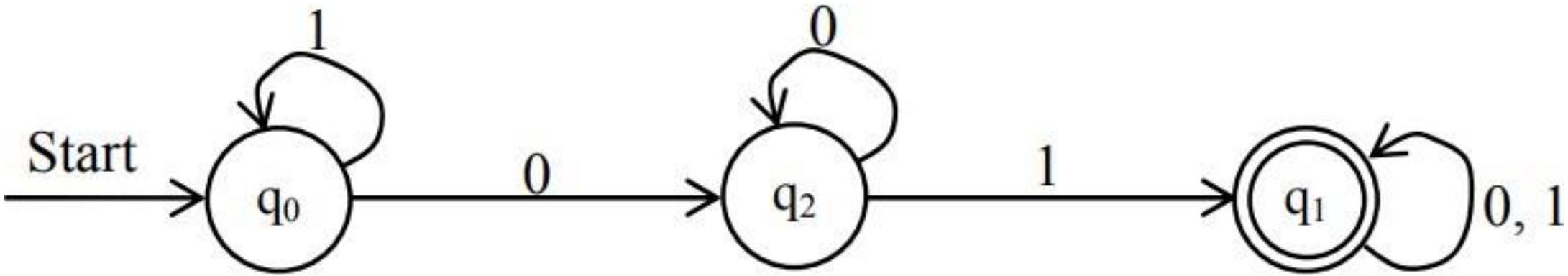
➢ The start state is labeled by an arrow written with “start” on the node.

➢ The final or accepting state is marked by double circle.

Construct a DFA over alphabet {0,1} that accepts binary string having even numbers of Zeros and Even numbers of 1



example II



How DFA process strings?

- How DFA decides whether or not to “accept” a sequence of input symbols ?
- The “language” of the DFA is the set of all symbols that the DFA accepts.

Extended Transition Function of DFA

- We define the extended transition function ˆδ. It takes a state q and an input string w to the resulting state. The definition proceeds by induction over the length of w.

Basis step(w has length 0)

In this case, w is the empty string, i.e. the string of length 0, for which we write Ɛ. We define

$$\hat{\delta}(q, \epsilon) = q.$$

Induction step (from length l to length l + 1):

- In this case, w, which has length *l + 1*, is of the form *va*, where *v* is a string of length *l* and *a* is a symbol. We define

$$\hat{\delta}(q, va) = \delta(\hat{\delta}(q, v), a).$$

- This works because, by induction hypothesis $\hat{\delta}(q, v)$ is already defined

Example:

$$\hat{\delta}(q, \epsilon) = q.$$

$$\hat{\delta}(q, va) = \delta(\hat{\delta}(q, v), a).$$

Compute $\hat{\delta}(q_0, 1001)$

$= \delta(\hat{\delta}(q_0, 100), 1)$

$= \delta(\delta(\hat{\delta}(q_0, 10), 0), 1)$

$= \delta(\delta(\delta(\hat{\delta}(q_0, 1), 0), 0), 1)$

$= \delta(\delta(\delta(\delta(\hat{\delta}(q_0, \varepsilon), 1), 0), 0), 1)$

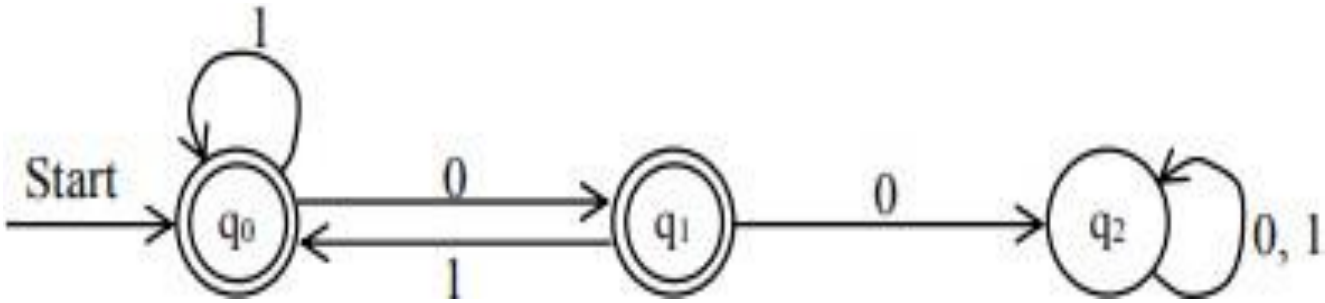
$= \delta(\delta(\delta(\delta(q_0, 1), 0), 0), 1)$

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

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

$= \delta(q_2, 1)$

$= q_2$, *so accepted.*



Compute $\hat{\delta}(q_0, 101)$

Compute $\hat{\delta}(q_0, 101)$

$= \delta(\hat{\delta}(q_0, 10), 1)$

$= \delta(\delta(\hat{\delta}(q_0, 1), 0), 1)$

$= \delta(\delta(\delta(\hat{\delta}(q_0, \varepsilon), 1), 0), 1)$

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

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

$= \delta(q_1, 1)$

$= q_0$, *so not accepted*.

String accepted by a DFA

- A string x is accepted by a DFA (Q, ∑, δ, q0, F) if; $\hat{\delta}$ (q, x) = p $\in$ F.

Language of DFA

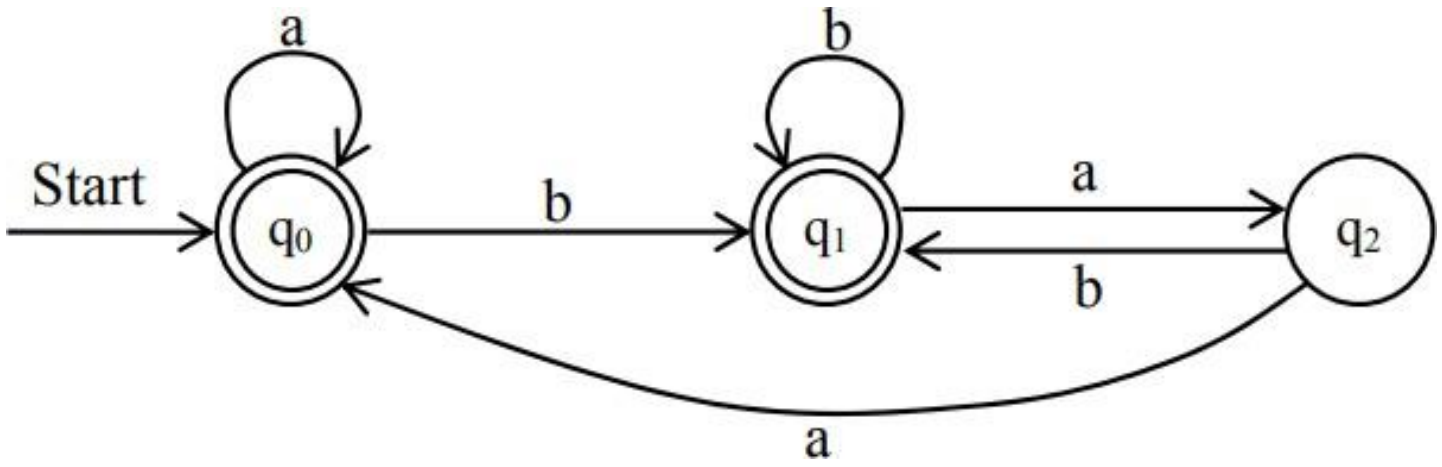
- The language of DFA ***M = (Q, ∑, δ, q0, F)*** denoted by L(M) is a set of strings over ∑* that are accepted by M.

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

That is; the language of a DFA is the set of all strings w that take DFA starting from start state to one of the accepting states. The language of DFA is called regular language.

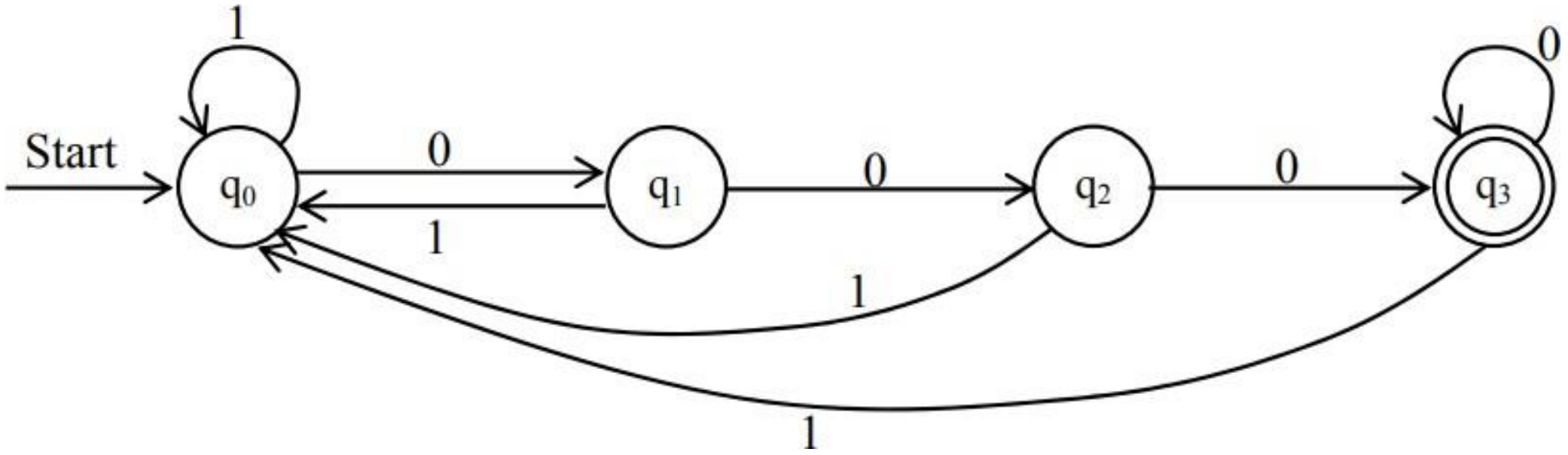
Example:1

- Construct a DFA, that accepts all the strings over ∑ = {a, b} that do not end with ba.

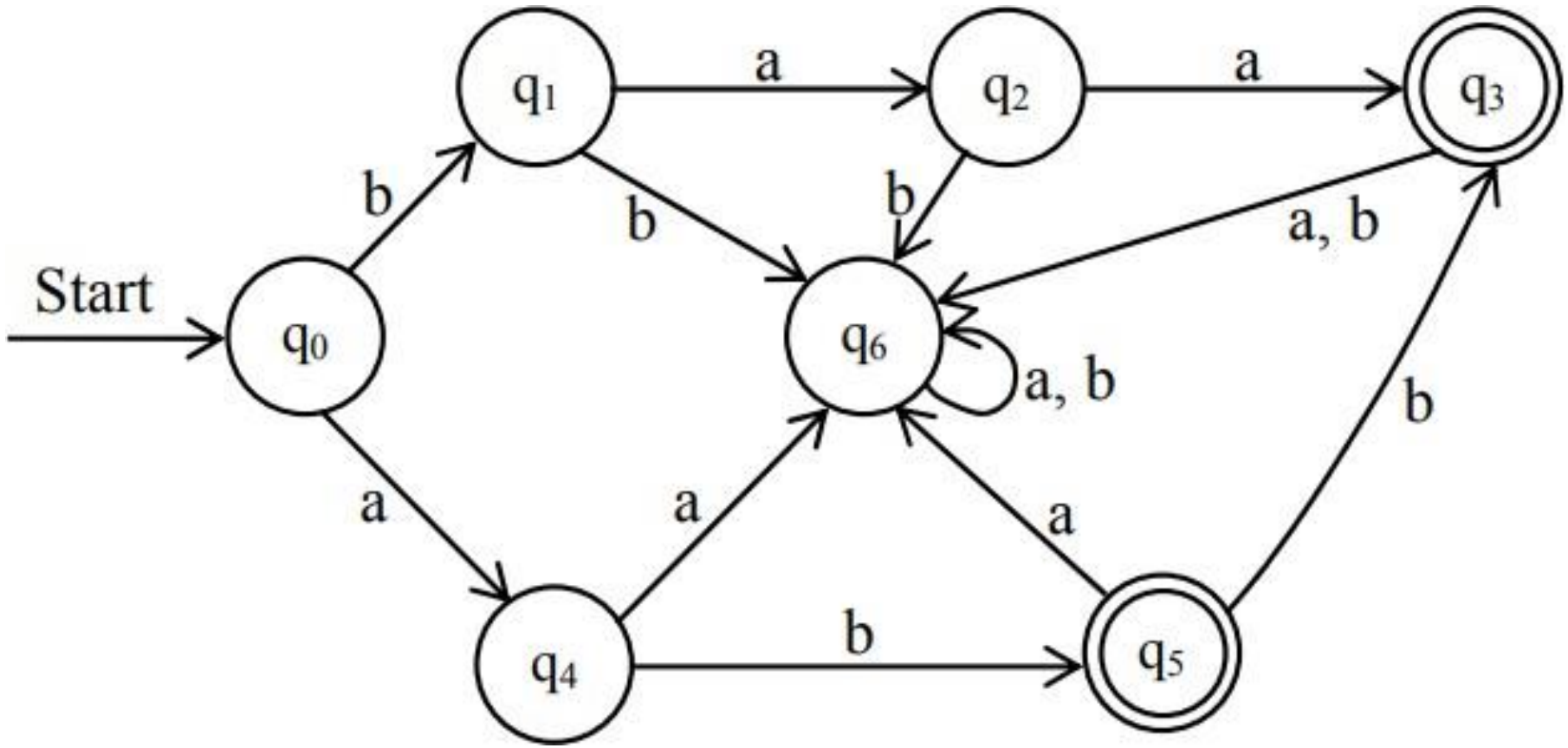


Example:2

DFA accepting all string over Σ = {0, 1} ending with 3 consecutive 0's

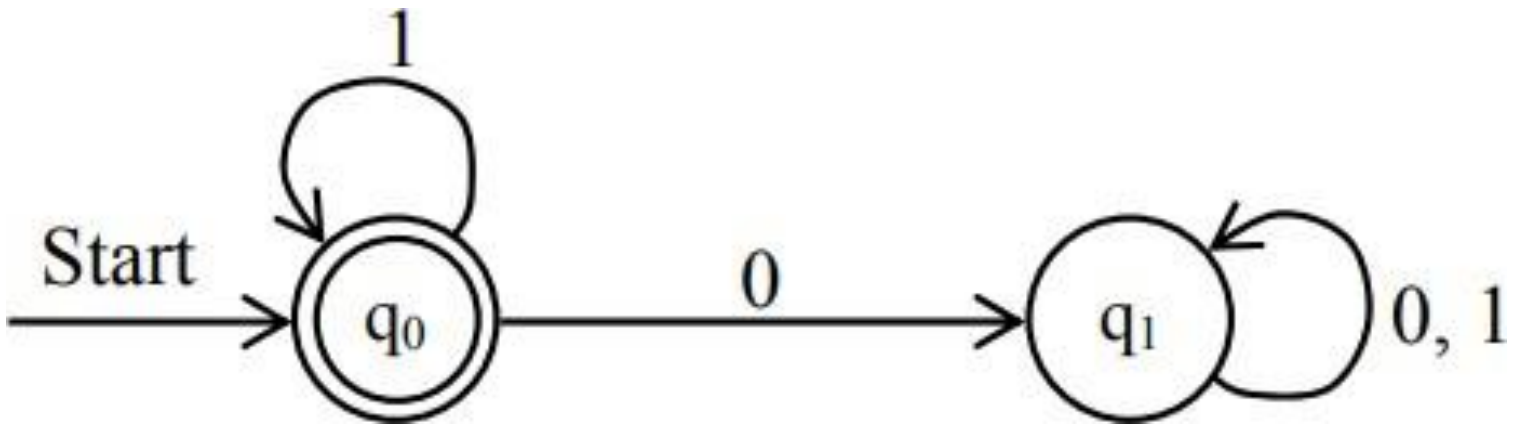


Example:3
DFA over {a, b} accepting {baa, ab, abb}

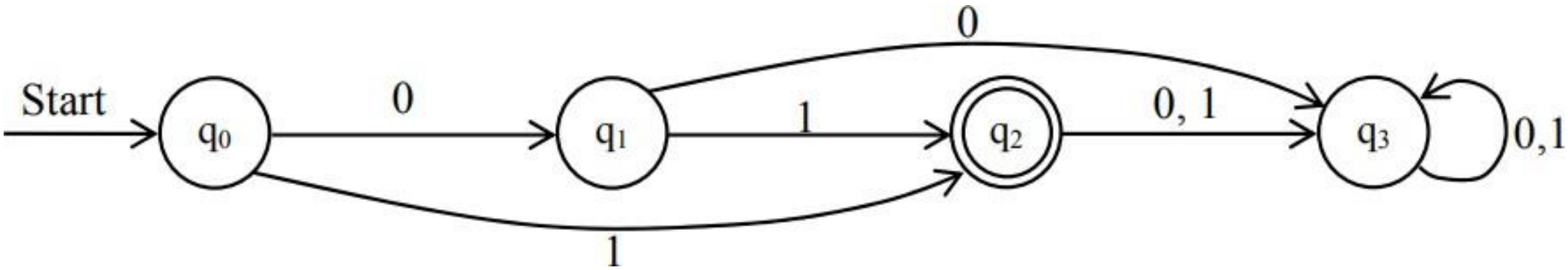


Example:4
DFA accepting zero or more consecutive 1's.

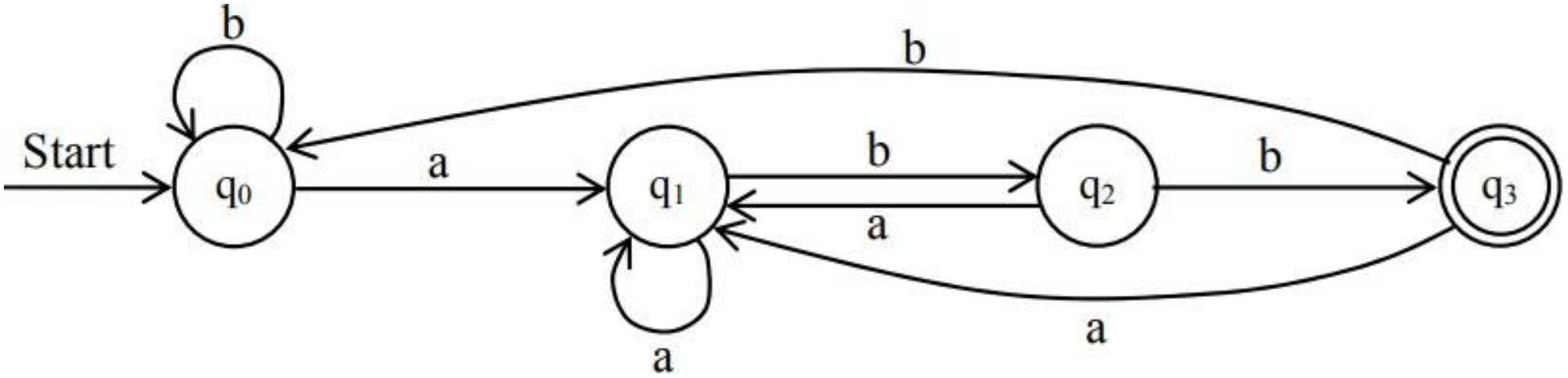
i.e. $L(M) = \{1^n / n = 0, 1, 2, \ldots\ldots\}$



Example:5
DFA over {0, 1} accepting {1, 01}

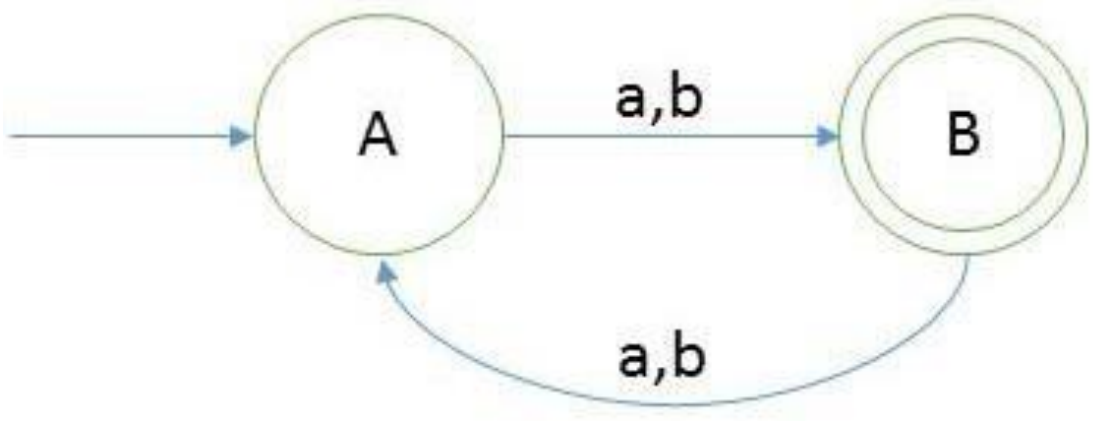


Example:5
DFA over {a, b} that accepts the strings ending with abb



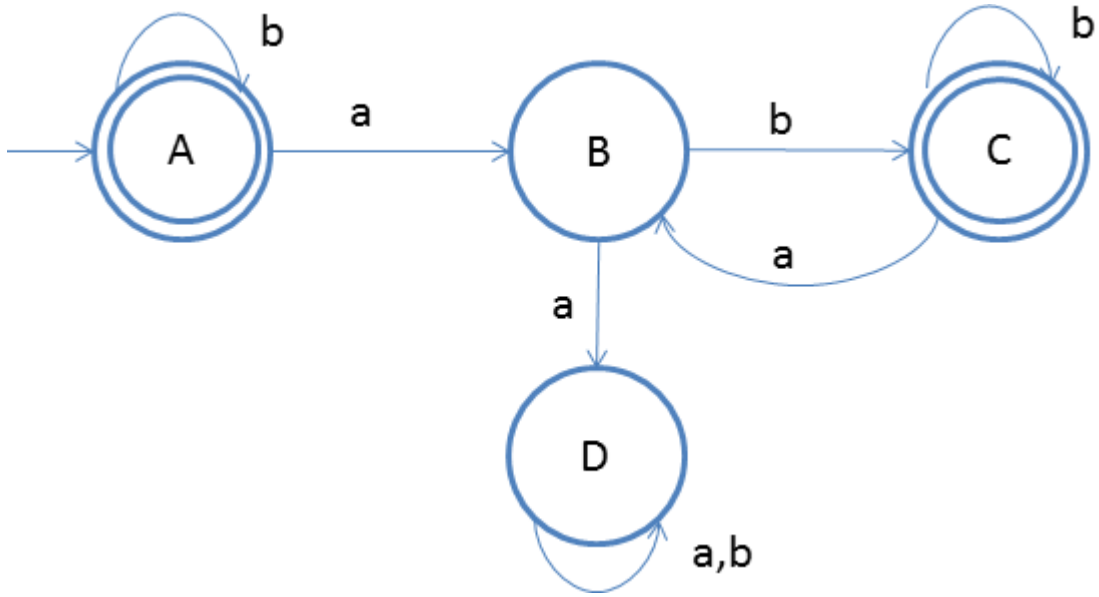
Example:7

Create a DFA which accepts strings of odd length ($\sum = \{a, b\}$)



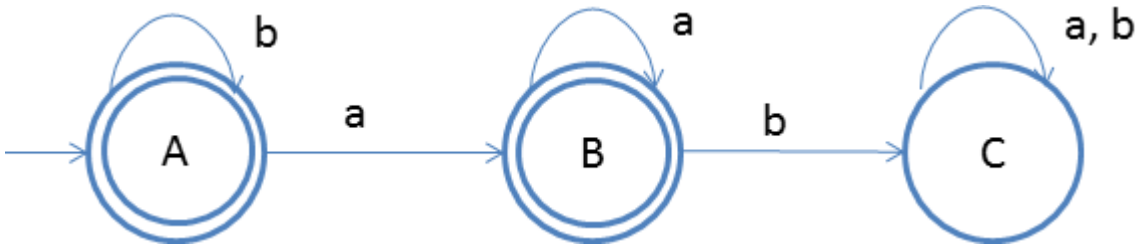
Example:8

- Design a DFA in which every 'a' should be followed by 'b'
  Given: Input alphabet, Σ={a, b}
  Language L = {ε, ab, abab, bbbb, ...}



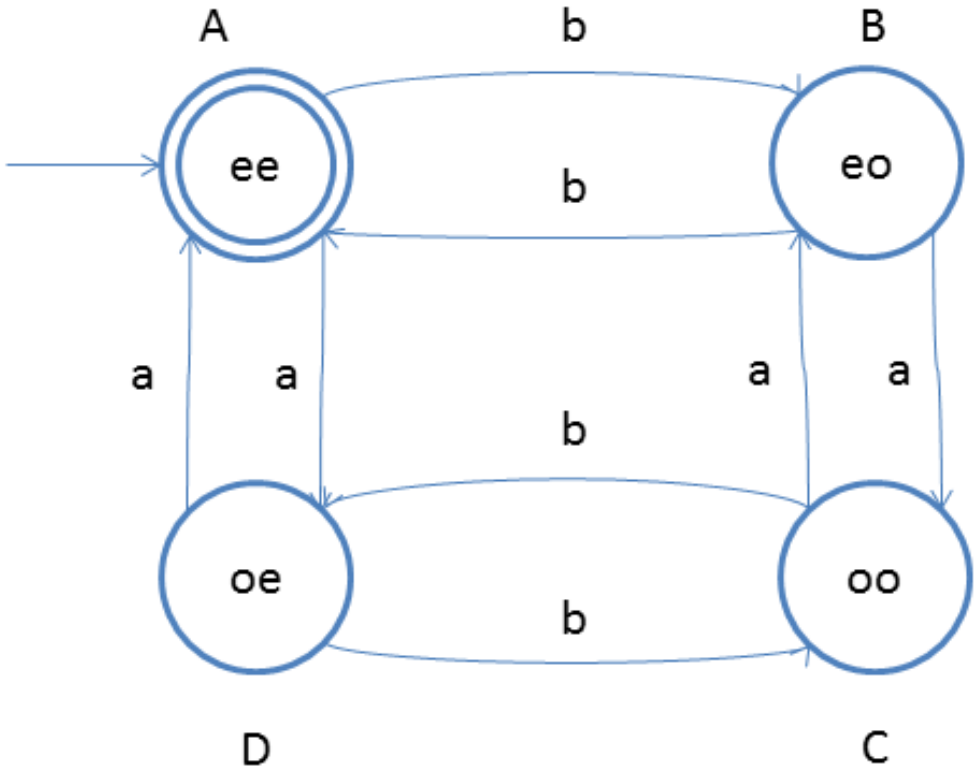
Example:9

- Design a DFA in which every 'a' should never followed by 'b'
  Given: Input alphabet, Σ={a, b}
  Language L = {ε, a, aa, aaa, ba, ...}

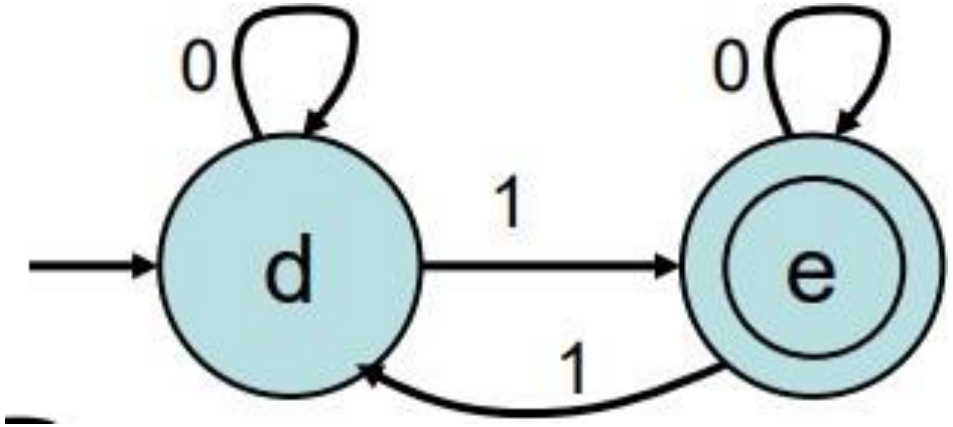


Example:10

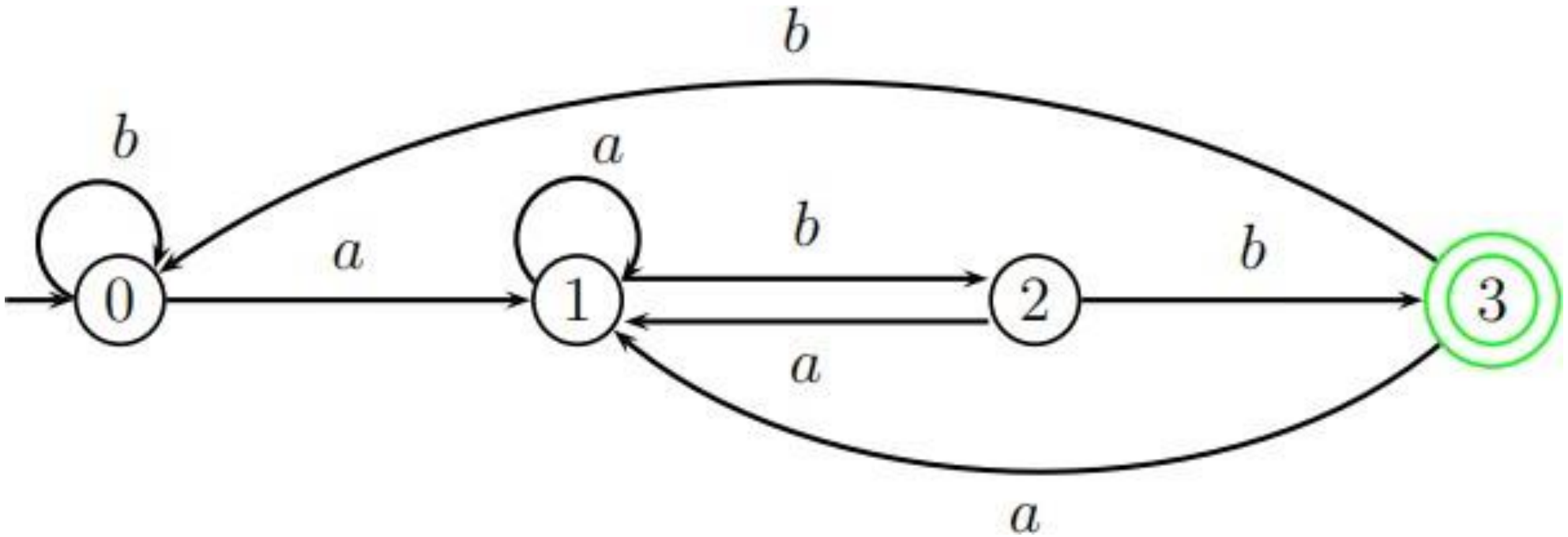
- Design a DFA Which accepts even numbers of ‘a’ and even numbers of ‘b’



Example:12
DFA: 0dd number of 1's



Example:13
DFA for {a, b}*{abb}



Non-Deterministic Finite Automata (NFA)

A deterministic finite automaton is defined by a quintuple (5-tuple) as (Q, ∑, δ, q0, F).

Where,

Q = Finite set of states,

∑ = Finite set of input symbols,

δ = A transition function that maps $Q \times \sum \rightarrow 2^{Q}$

q0 = A start state; q0 $\in$ Q

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

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

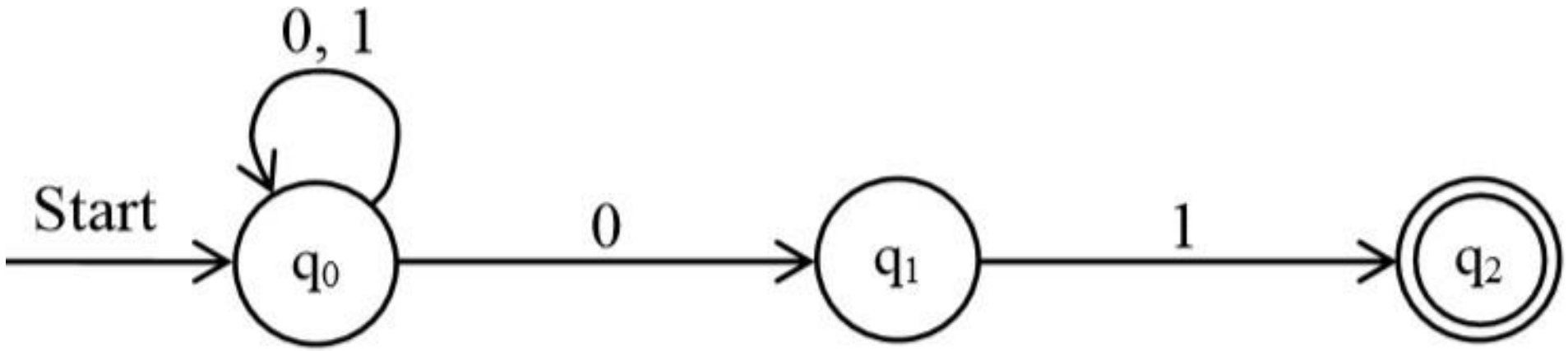
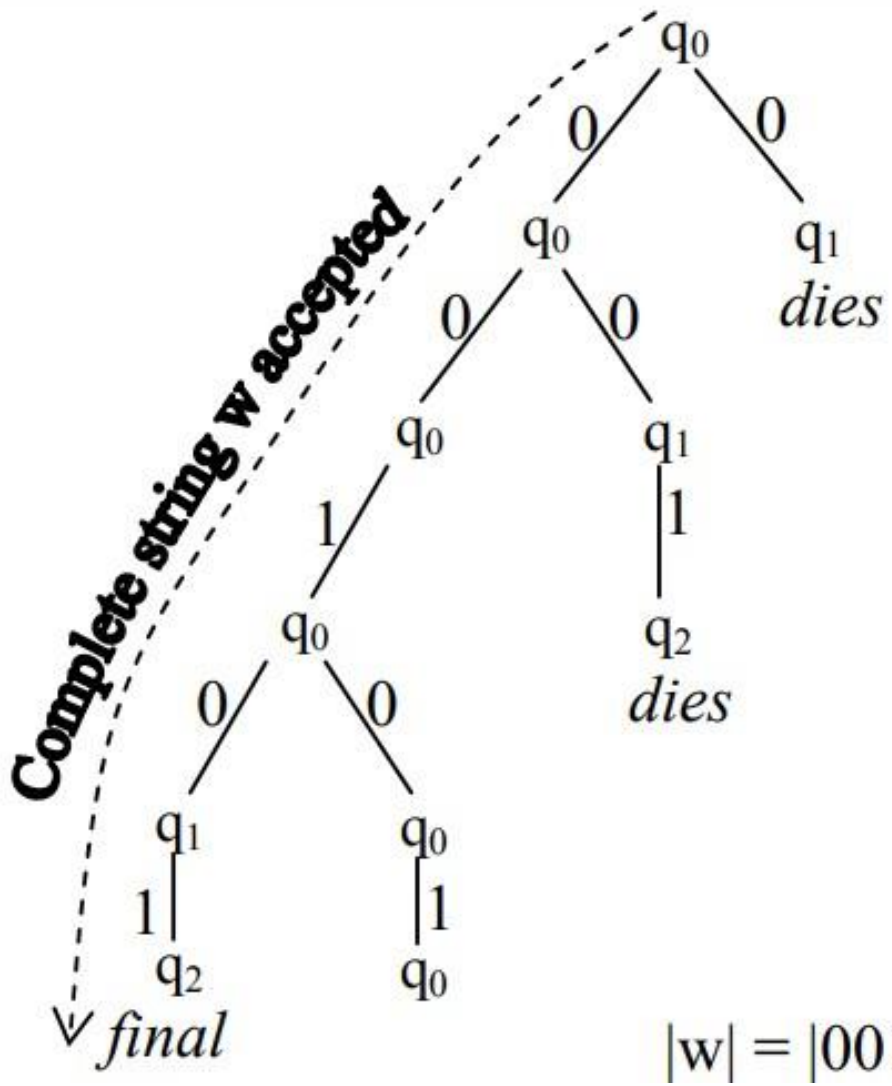


Fig: - NFA accepting all strings that end in 01.



$|w| = |00101| = 5$

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

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

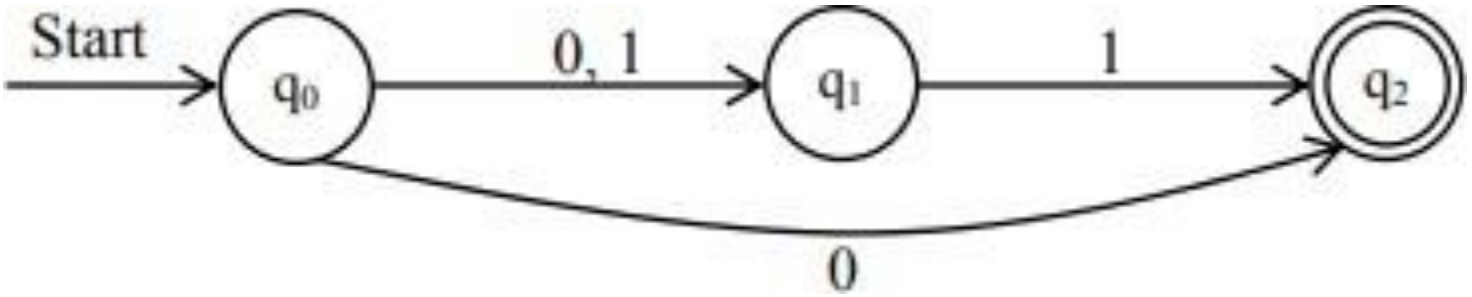
$q_0 = \{q_0\}$

$F = \{q_2\}$

Transition table:

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

Example:NFA over {0, 1} accepting strings {0, 01, 11}



δ:	0	1
→ q_0	$\{q_0, q_2\}$	$\{q_1\}$
q_1	$\{\phi\}$	$\{q_2\}$
*q_2	$\{\phi\}$	$\{\phi\}$

Extended Transition Function of NFA

- **Basis**: Without reading any input symbols, we are only in the state we began in.

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

Induction

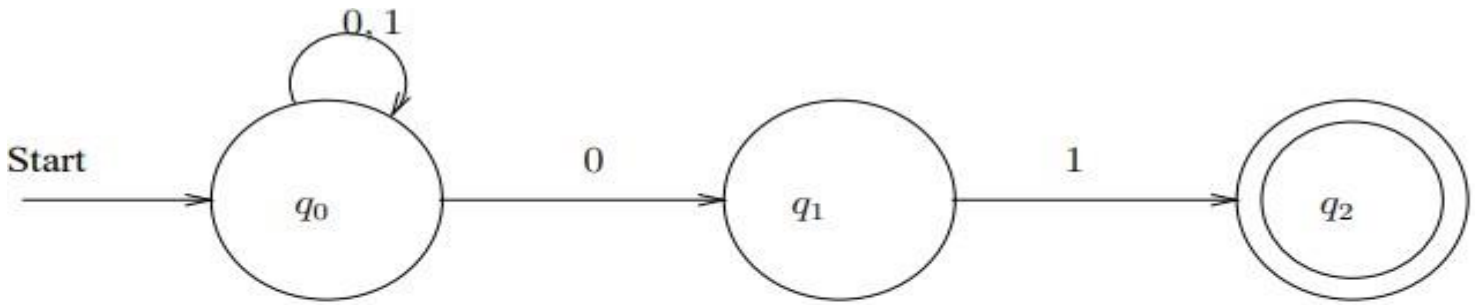
- Suppose w is a string of the form xa; that is ‘a’ is the last symbol of w, and x is the string consisting of all but not the last symbol.
- Also suppose that $\hat{\delta}(q, x) = \{p_1, p_2, \ldots p_k, \}$
- Let,

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

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

- 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.

Example: An NFA accepting strings that end in 01

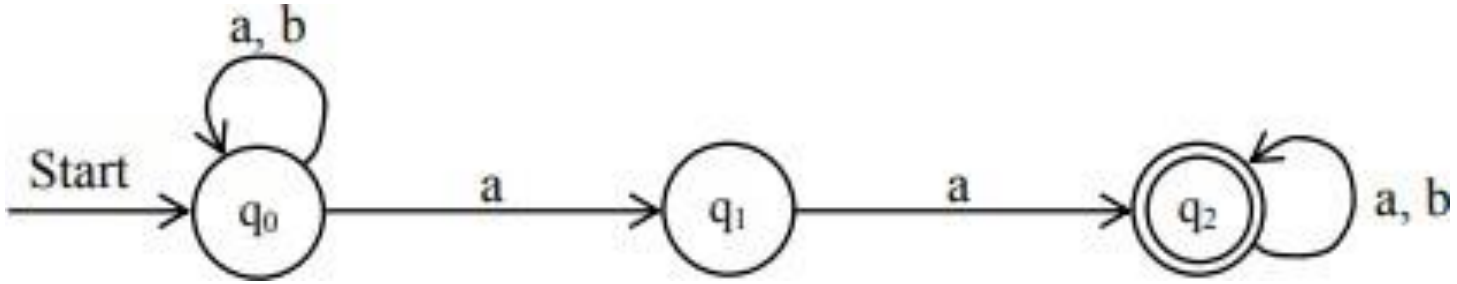


Processing $w = 00101$

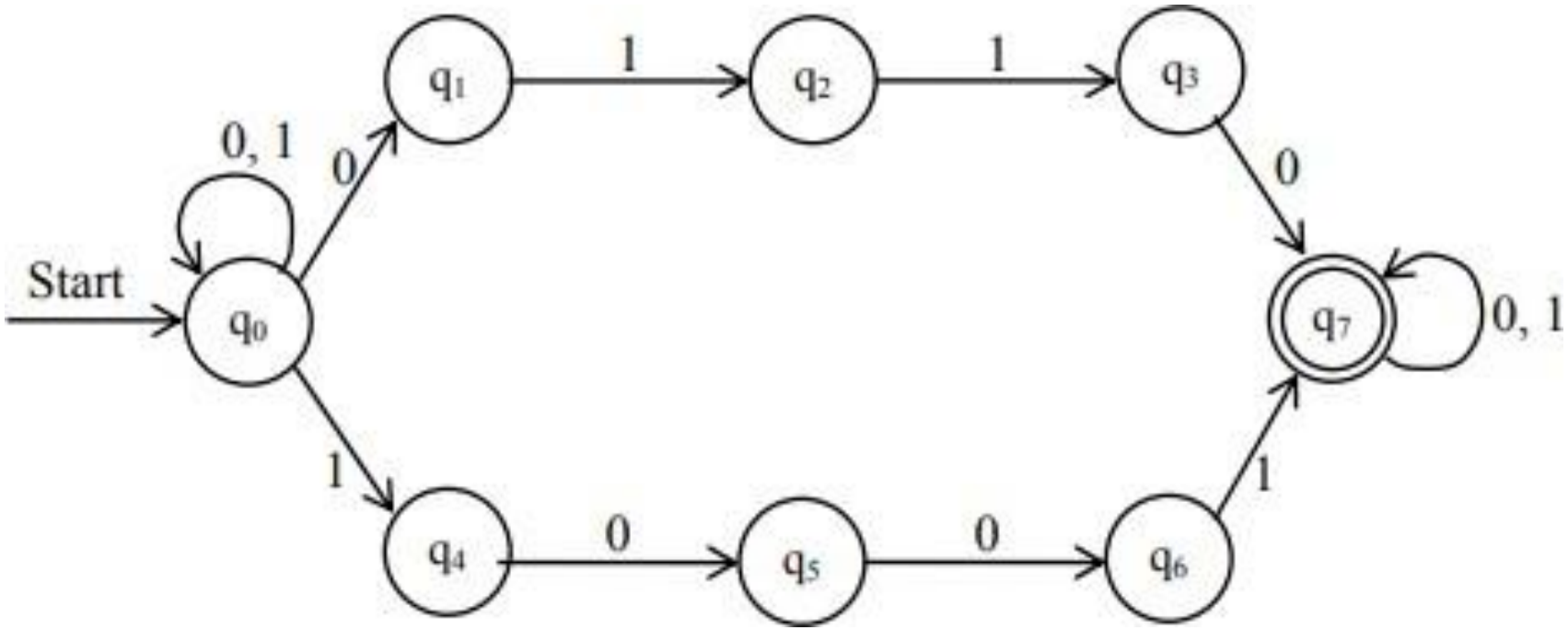
1. $\hat{\delta}(q_0, \epsilon) = \{q_0\}$
2. $\hat{\delta}(q_0, 0) = \delta(q_0, 0) = \{q_0, q_1\}$
3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
4. $\hat{\delta}(q_0, 001) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$
5. $\hat{\delta}(q_0, 0010) = \delta(q_0, 0) \cup \delta(q_2, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$
6. $\hat{\delta}(q_0, 00101) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

NFA Practice:

- Construct a NFA over {a, b} that accepts strings having aa is substring.

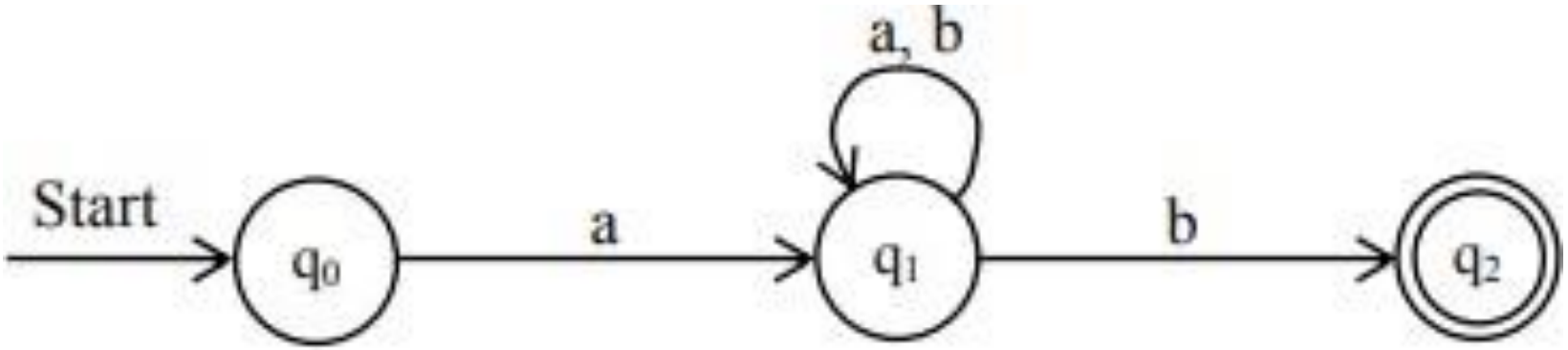


NFA for strings over {0, 1} that contain0110 or 1001.



NFA over {a, b} that have a as one of the last 3 characters.

NFA over {a, b} that accepts strings starting with a and ending with b.



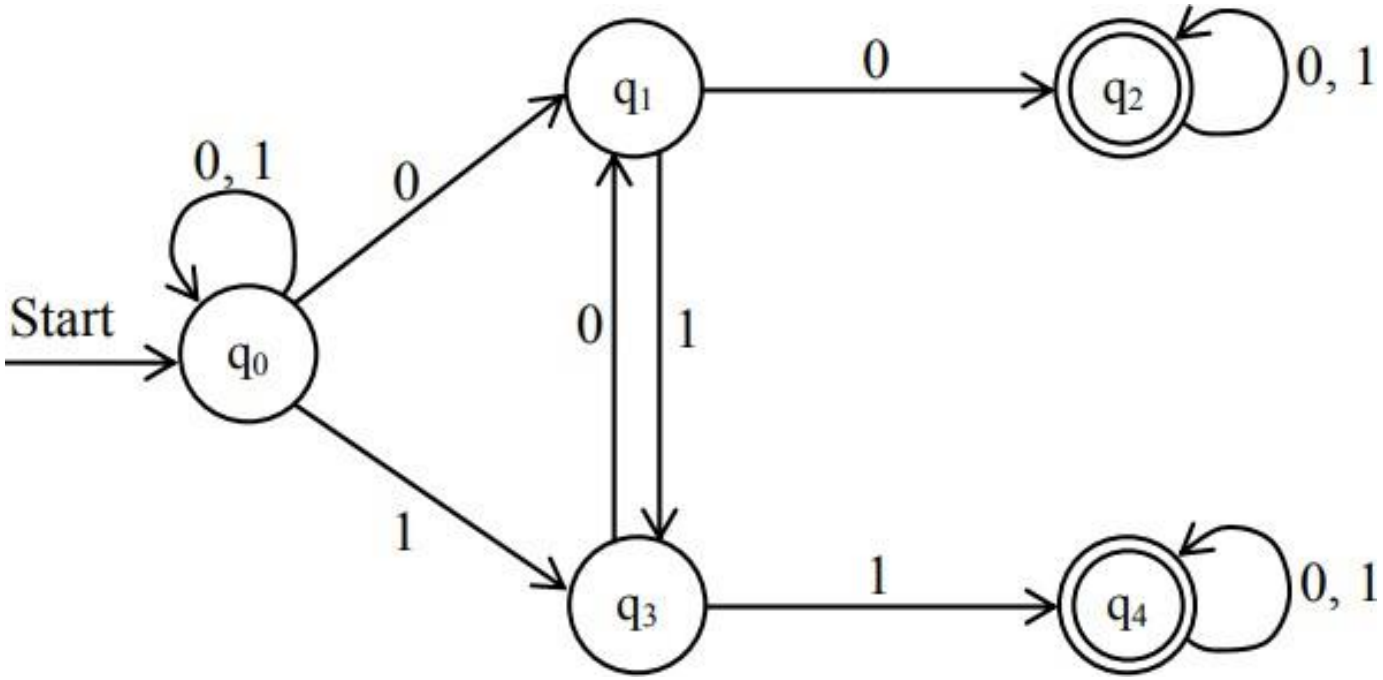
Language of NFA

- The language of a NFA A = (Q, Σ, δ, q 0, F), denoted L(A) is defined by:

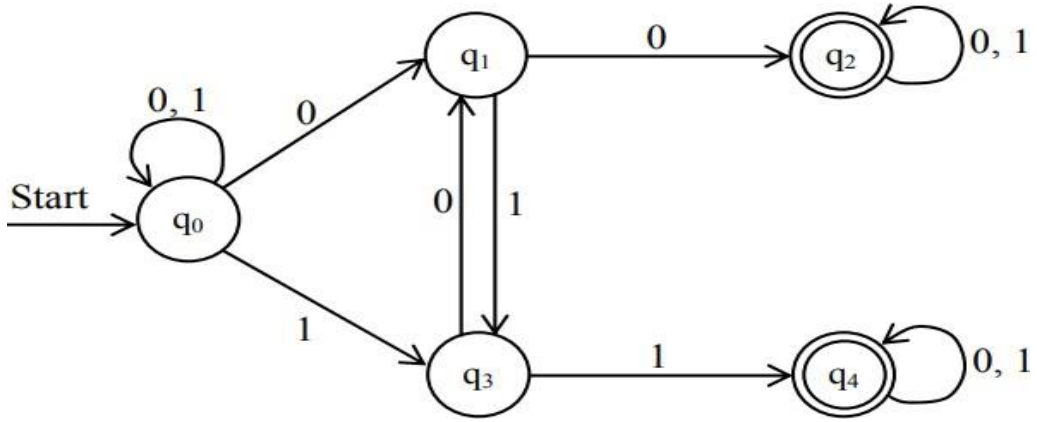
$$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

- The language of A is the set of strings $w \in \Sigma^*$ such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.
- The fact that choosing using the input symbols of w lead to a non-accepting state, or do not lead to any state at all, does not prevent w from being accepted by a NFA as a whole.

E.g. Design a NFA for the language over{0, 1} that have at least two consecutive 0's or 1's.



Now, compute 10110;

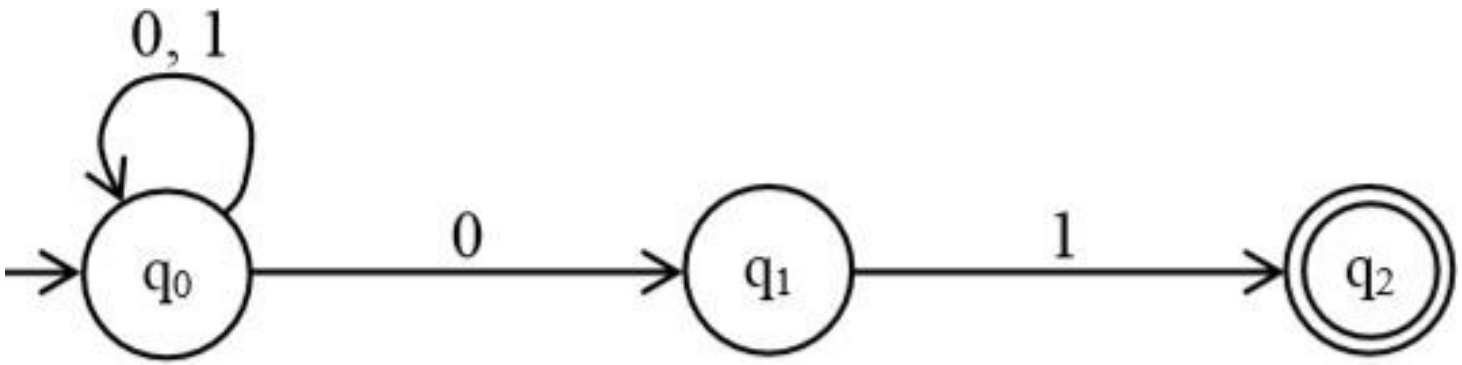


Equivalence of NFA & DFA

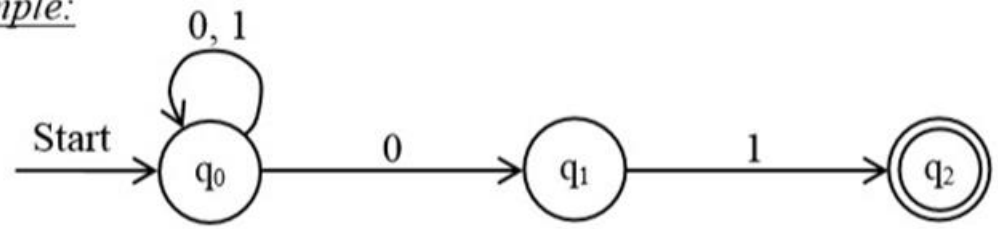
- We now show that DFAs & NFAs accept exactly the same set of languages. That is; non-determinism does not make a finite automaton more powerful.
- To show that NFAs and DFAs accept the same class of language, we show:
  - Any language accepted by a NFA can also be accepted by some DFA. For this we describe 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. 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 2n states (at most), although it usually has many less.

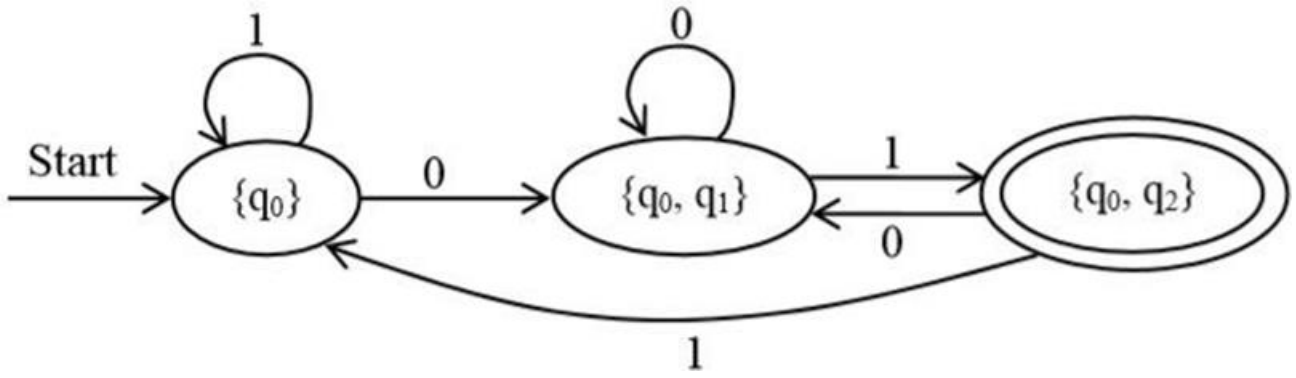
Convert following NFA into DFA(Subset construction method)



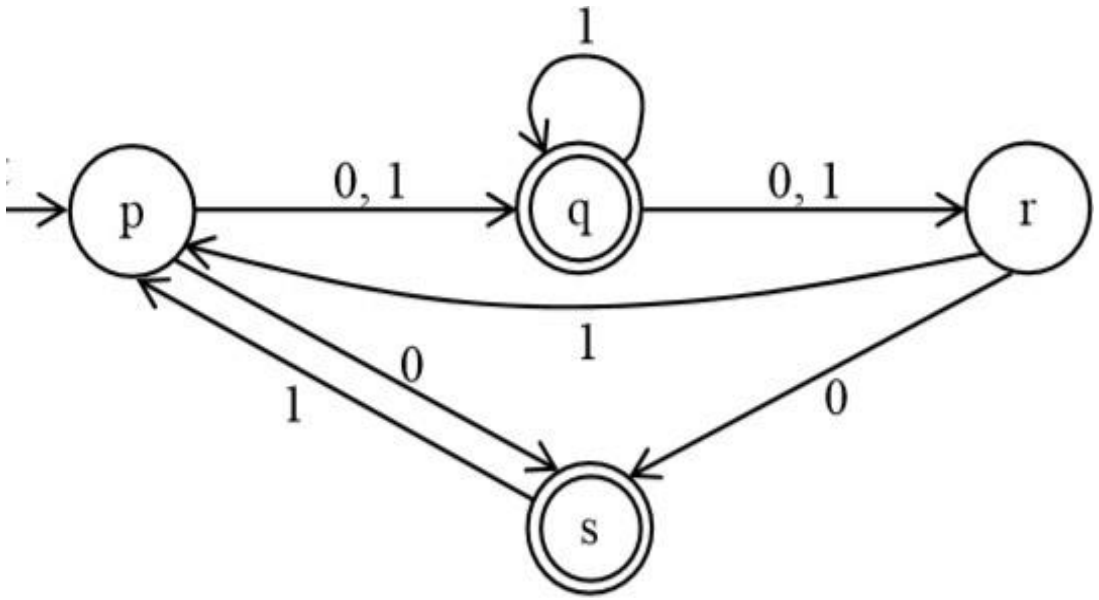
For example:



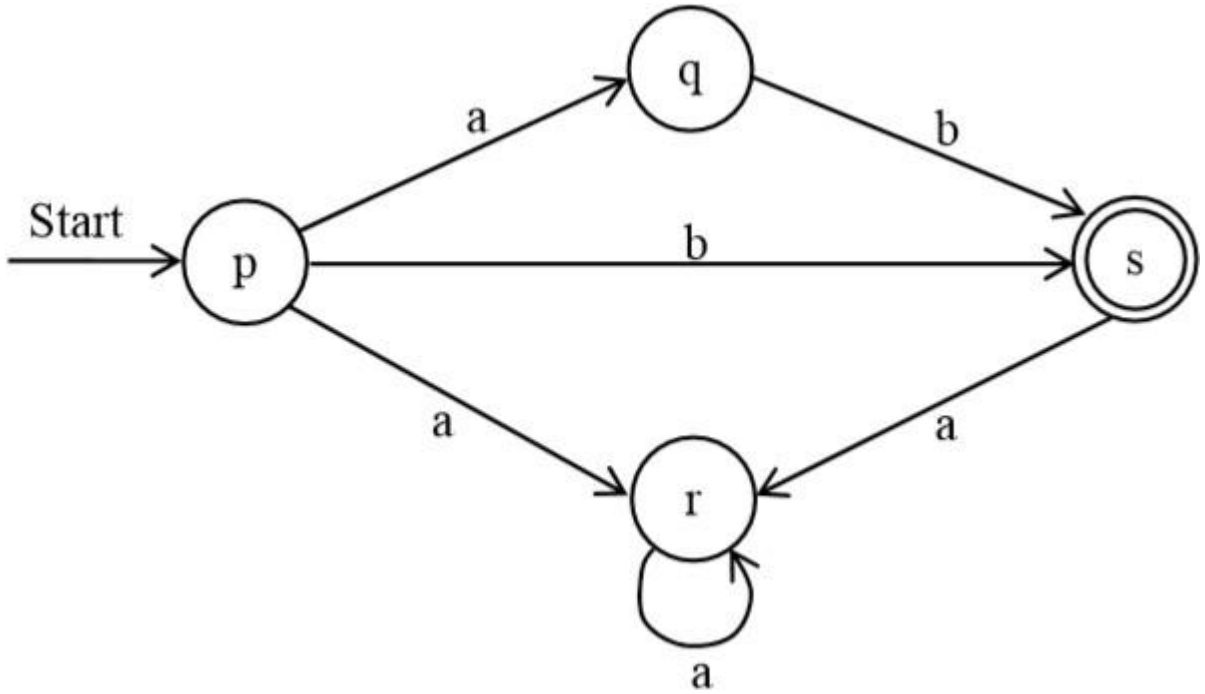
Solution



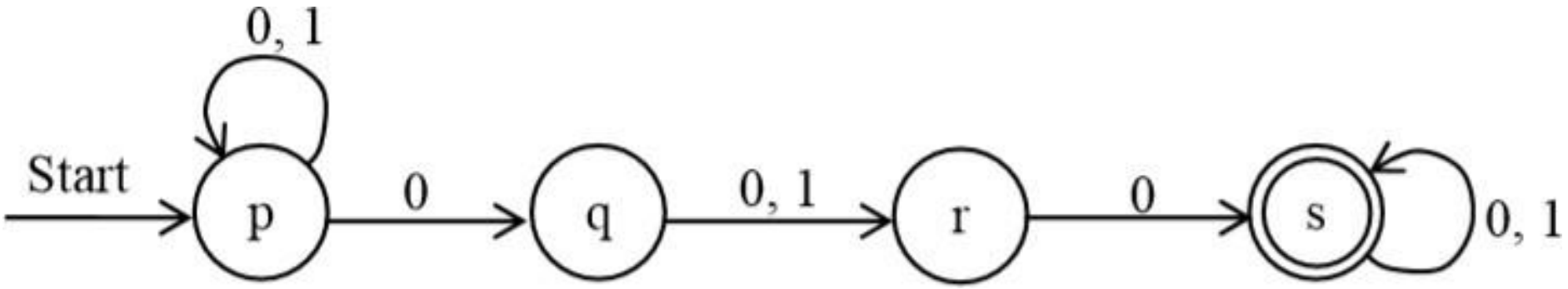
Example 1: Convert this NFA to DFA



Example 2: Convert this NFA to DFA (Assignment)



Example 3: Convert this NFA to DFA(Assignment)



NFA with ε-transition (ε-NFA)

- This is another extension of finite automation. The new feature that it incorporates is, it allows a transition on ε, the empty string, so that a NFA could make a transition spontaneously without receiving an input symbol.
- A NFA with ε-transition is defined by five tuples (Q, ∑, δ, q0, F), where;

  Q = set of finite states

  ∑ = set of finite input symbols

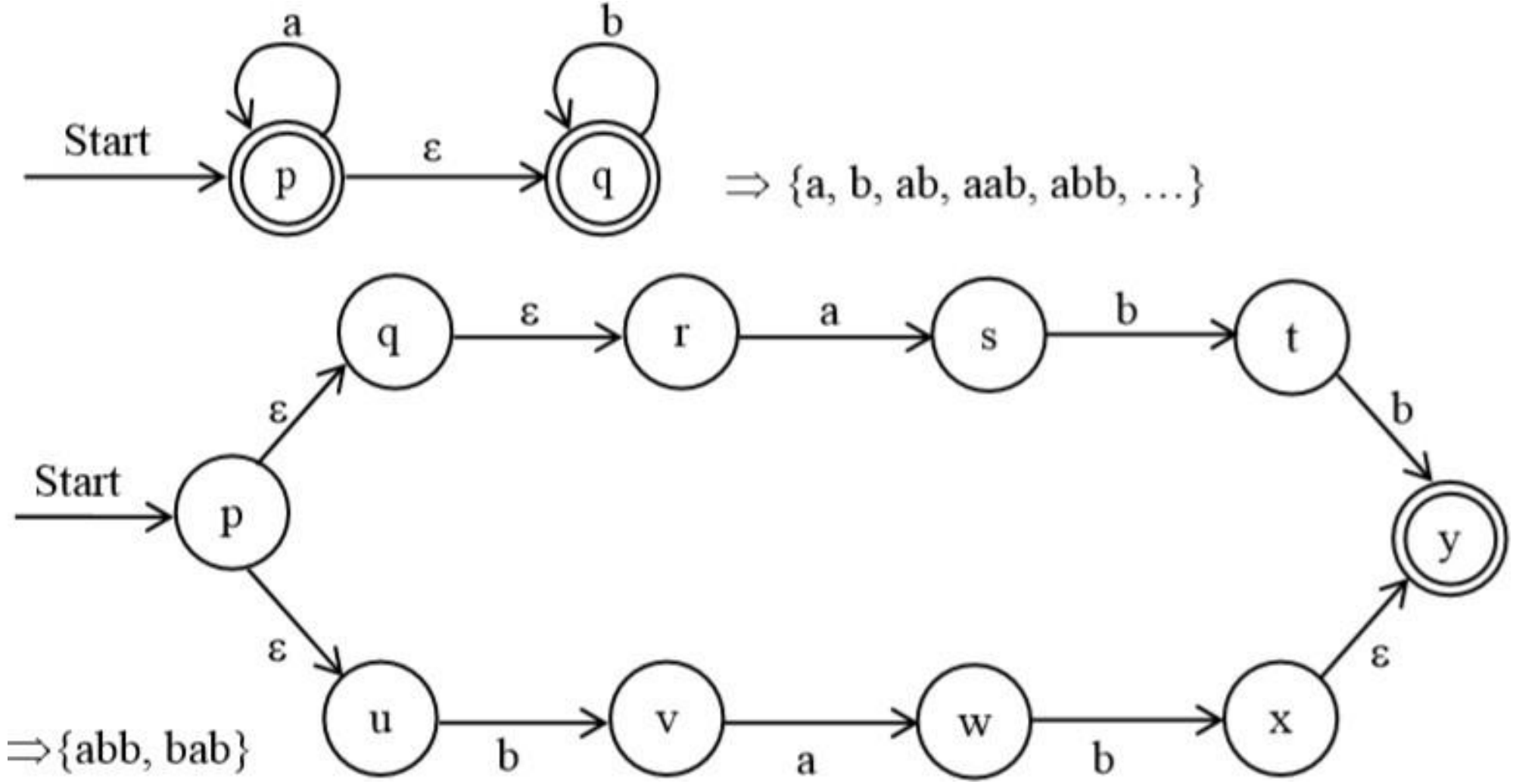
  q0 = Initial state, q0 $\in$ Q

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

  δ = a transition function that maps;

  Q × ∑ $\cup$ { ε } → 2^{Q}

  δ (q, ε) = {p1, p2, …… pk} (set of all states that can be reached from q with input ε).

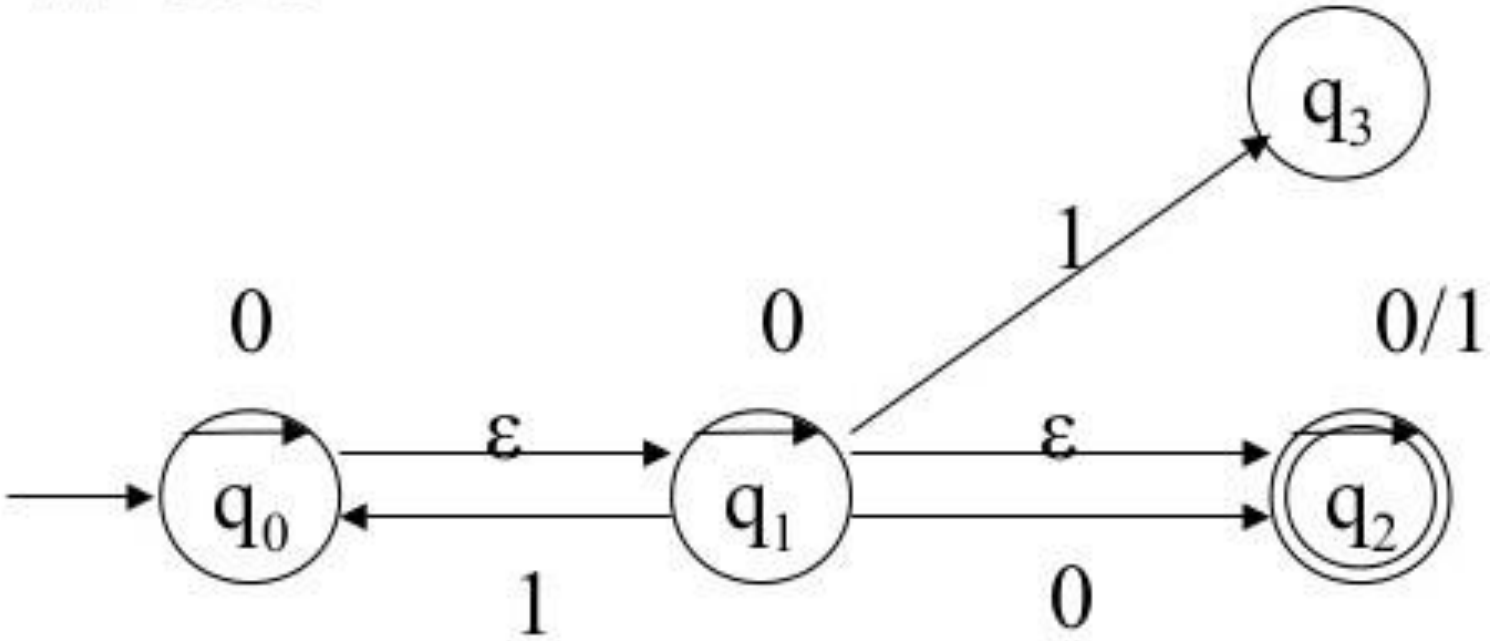

a
b
Start
p
ε
q
⇒ {a, b, ab, aab, abb, …}
q
ε
r
a
s
b
t
b
ε
Start
p
y
ε
ε
u
v
w
x
b
a
b
⇒{abb, bab}

ε-closure of a state

- ε-closure of a state ‘q’ can be obtained by following all transitions out of q that are labeled ε.
- After we get to another state by following ε, we follow the ε-transitions out of those states & so on, eventually finding every state that can be reached from q along any path whose arcs are all labeled ε.

Formal Definition

- Formally, we can define ε-closure of the state q as;
- ***Basis***: state q is in ε-closure (q).
- ***Induction***: If state q is reached with ε-transition from state q, p is in ε-closure (q). and if there is an arc from p to r labeled ε, then r is in ε-closure (q) and so on.

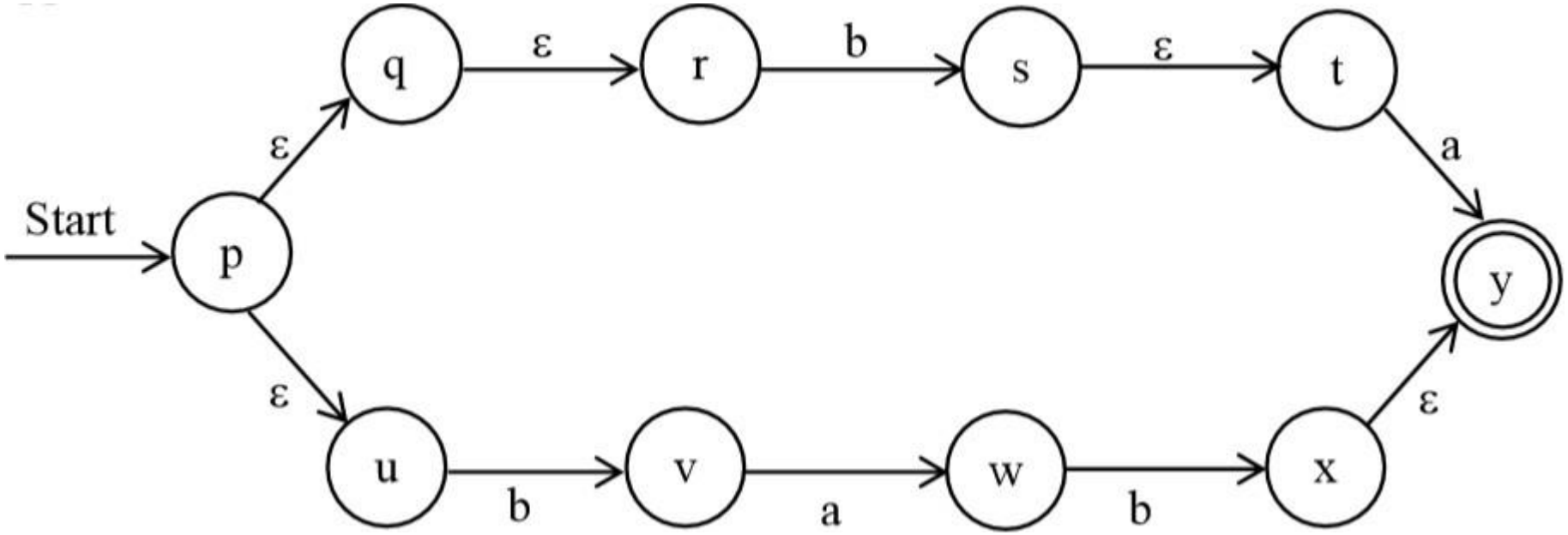


ε-closure (q0)={q0,q1,q2}

ε-closure (q1)={q1,q2}

ε-closure (q2)={q2}

Extended Transition Function of ε-NFA



Now, compute for string ba.

1) $\hat{\delta}(p, \varepsilon) = \varepsilon\text{-closure}(p) = \{p, q, r, s\}$

2) $\hat{\delta}(p, b) = \delta(p, b) \cup \delta(q, b) \cup \delta(r, b) \cup \delta(s, b)$

$= \phi \cup \phi \cup \{u\} \cup \{t\} = \{u, t\}$

$\hat{\delta}(u, \varepsilon) \cup \hat{\delta}(t, \varepsilon) = \varepsilon\text{-closure}(u) \cup \varepsilon\text{-closure}(t)$

$\therefore \hat{\delta}(p, b) = \{u\} \cup \{t, x\} = \{u, t, x\}$

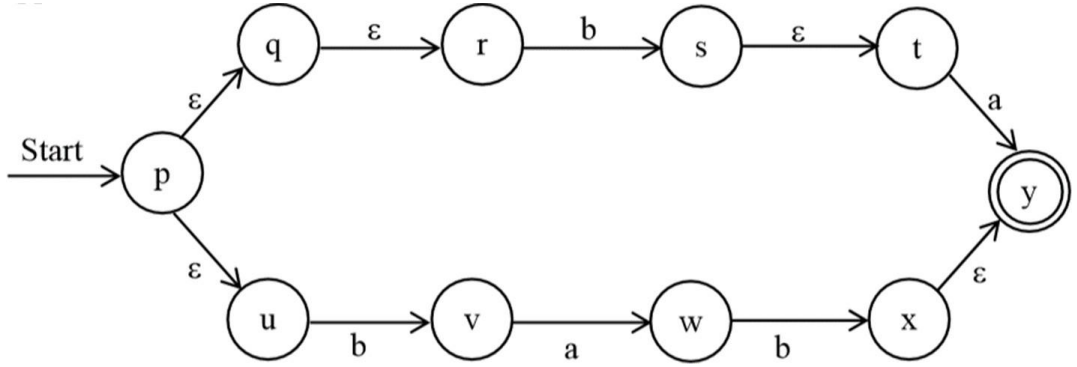
3) $\hat{\delta}(p, ba) = \delta(u, a) \cup \delta(t, a) \cup \delta(x, a)$

$= \{v\} \cup \phi \cup \{y\}$

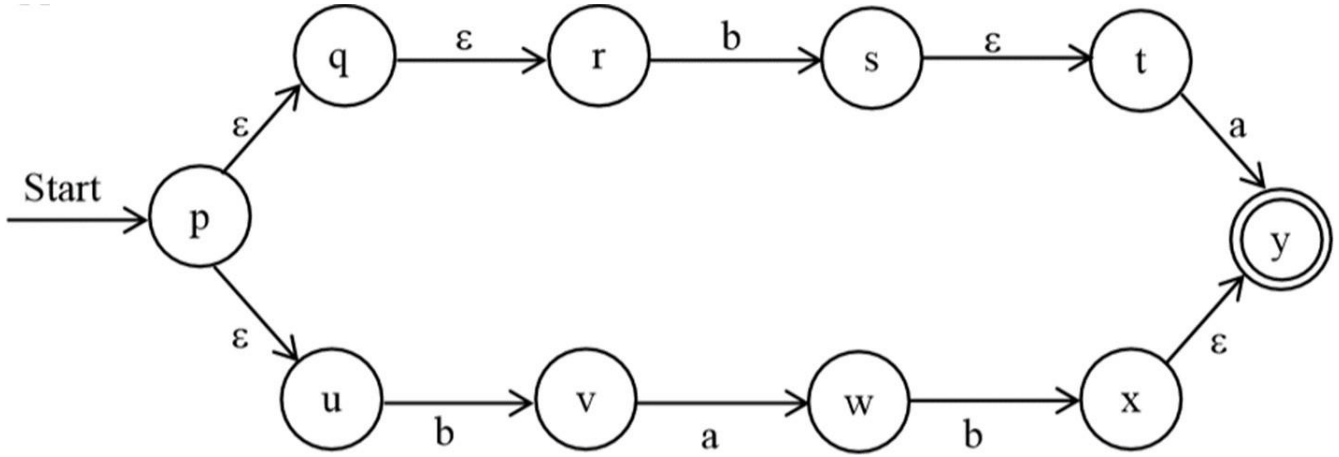
$\hat{\delta}(v, \varepsilon) \cup \hat{\delta}(y, \varepsilon) = \varepsilon\text{-closure}(v) \cup \varepsilon\text{-closure}(y)$

$= \{v\} \cup \{y\}$

$\therefore \hat{\delta}(p, ba) = \{v, y\}$ *Accepted*



Compute for stringbab



1) $\hat{\delta}(p, \varepsilon) = \varepsilon\text{-closure}(p) = \{p, q, r, s\}$

2) $\hat{\delta}(p, b) = \delta(p, b) \cup \delta(q, b) \cup \delta(r, b) \cup \delta(s, b)$

$= \phi \cup \phi \cup \{u\} \cup \{t\} = \{u, t\}$

$\hat{\delta}(u, \varepsilon) \cup \hat{\delta}(t, \varepsilon) = \varepsilon\text{-closure}(u) \cup \varepsilon\text{-closure}(t)$

$\therefore \hat{\delta}(p, b) = \{u\} \cup \{t, x\} = \{u, t, x\}$

3) $\hat{\delta}(p, ba) = \delta(u, a) \cup \delta(t, a) \cup \delta(x, a)$

$= \{v\} \cup \phi \cup \{y\} = \{v, y\}$

$\hat{\delta}(v, \varepsilon) \cup \hat{\delta}(y, \varepsilon) = \varepsilon\text{-closure}(v) \cup \varepsilon\text{-closure}(y)$

$= \{v\} \cup \{y\} = \{v, y\}$

$\therefore \hat{\delta}(p, ba) = \{v, y\}$

4) $\hat{\delta}(p, bab) = \delta(v, b) \cup \delta(y, b)$

$= \{w\} \cup \phi$

$= \{w\}$

$\hat{\delta}(w, \varepsilon) = \varepsilon\text{-closure}(w)$

$= \{y\}$ *Accepted.*

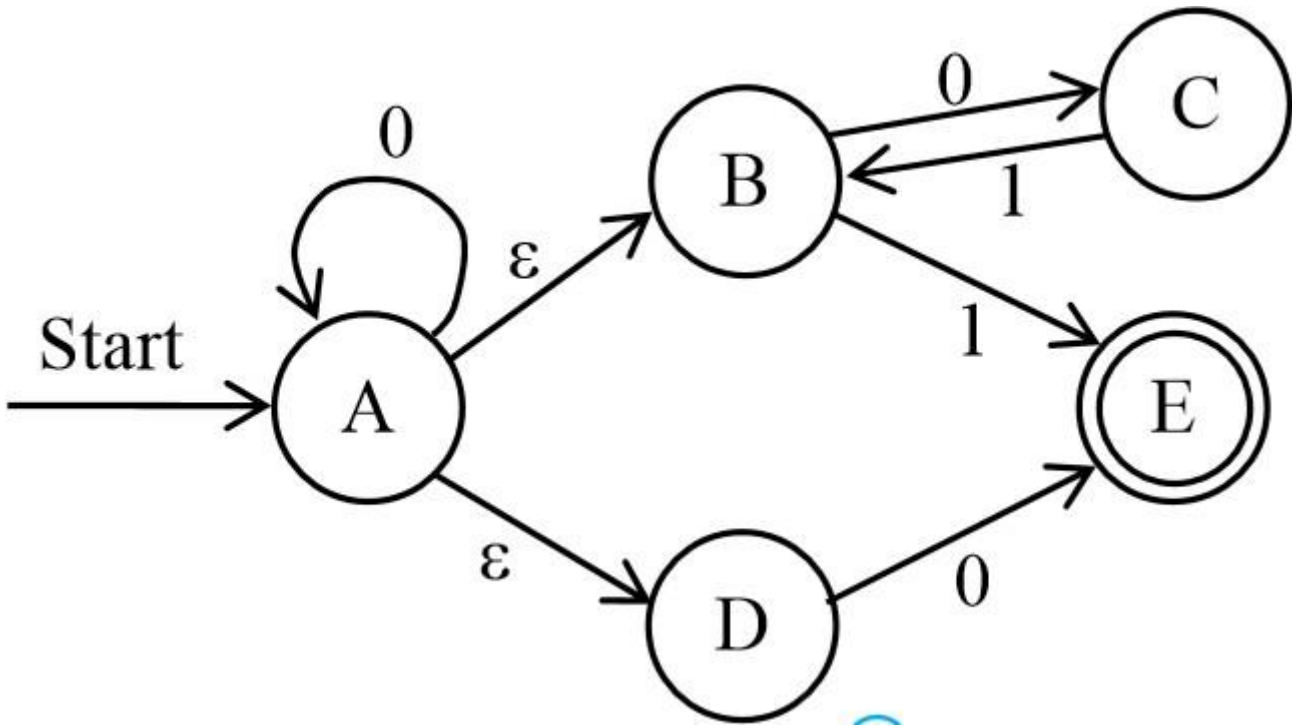
Conversion of ε-NFA into NFA & DFA
1) ε-NFA to NFA

To construct NFA from a given ε-NFA;

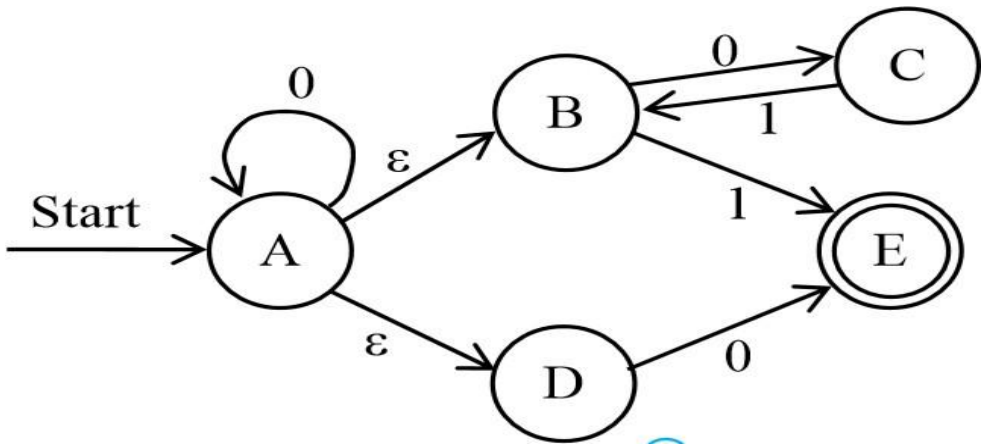
- Here, we have to do is to eliminate the ε- transitions somehow, so that the resulting NFA will have no more ε-transitions. For this we do as below:
- Take start state of ε-NFA as start state of NFA. If ε-NFA accepts ε, then mark start state as a final state.
- Take final state of ε-NFA as final state of NFA.
- Perform $\delta N(q, a) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q), a))$

  Where, δN = transition function of resulting NFA.

Example:



	ε	0	1
→ A	{B, D}	{A}	ϕ
B	ϕ	{C}	{E}
C	ϕ	ϕ	{B}
D	ϕ	{E}	ϕ
*E	ϕ	ϕ	ϕ



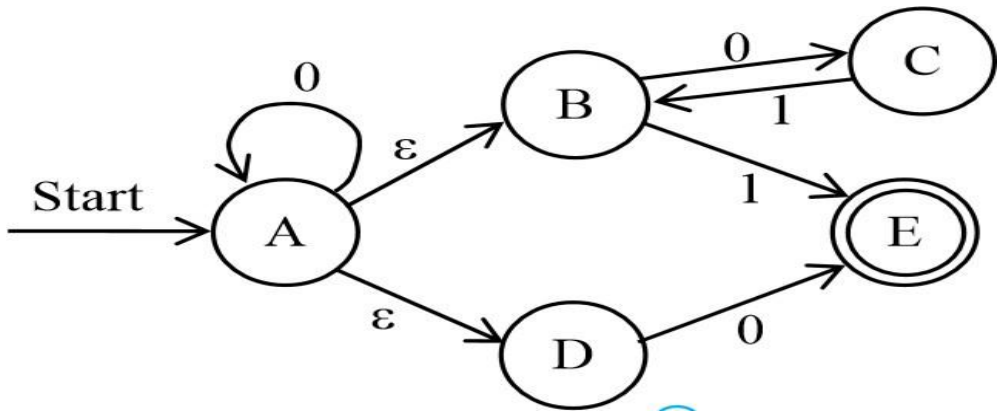
Start state = A

$$\delta_N (A, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(A), 0))$$
$$= \varepsilon\text{-closure}(\delta(\{A, B, D\}, 0))$$
$$= \varepsilon\text{-closure}(\{A, C, E\})$$
$$= \{A, B, C, D, E\}$$

$$\delta_N (A, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(A), 1))$$
$$= \varepsilon\text{-closure}(\delta(\{A, B, D\}, 1))$$
$$= \varepsilon\text{-closure}(\{E\})$$
$$= \{E\}$$

$$\delta_N (B, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(B), 0))$$
$$= \varepsilon\text{-closure}(\delta(\{B\}, 0))$$
$$= \varepsilon\text{-closure}(\{C\})$$
$$= \{C\}$$

$$\delta_N (B, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(B), 1))$$
$$= \varepsilon\text{-closure}(\delta(\{B\}, 1))$$
$$= \varepsilon\text{-closure}(\{E\})$$
$$= \{E\}$$

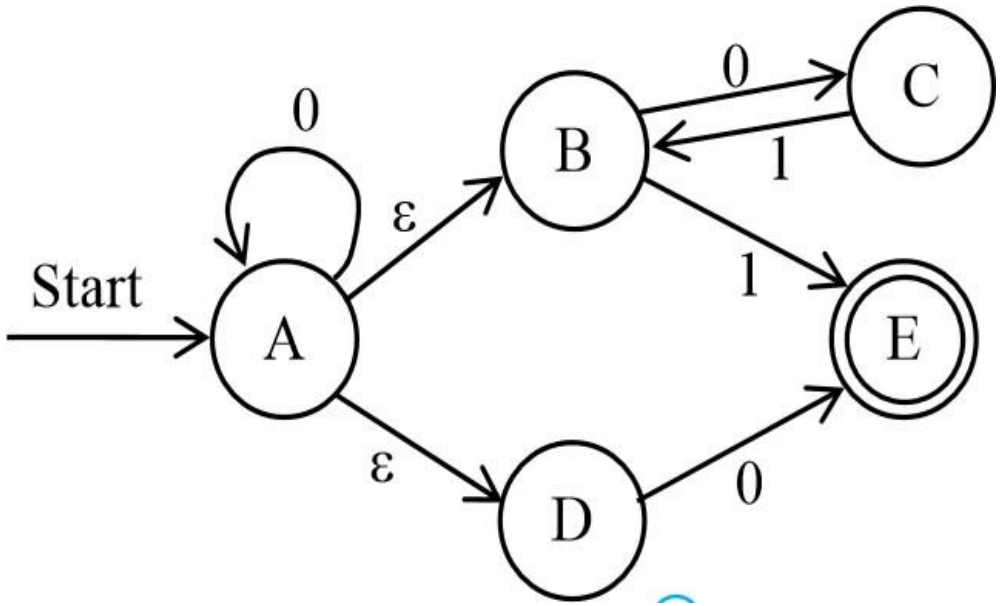


$$\delta_N (C, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(C), 1))$$
$$= \varepsilon\text{-closure}(\delta(\{C\}, 1))$$
$$= \varepsilon\text{-closure}(\{B\})$$
$$= \{B\}$$

$$\delta_N (D, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(D), 0))$$
$$= \varepsilon\text{-closure}(\delta(\{D\}, 0))$$
$$= \varepsilon\text{-closure}(\{E\})$$
$$= \{E\}$$

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

$$\delta_N (D, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(D), 1))$$
$$= \varepsilon\text{-closure}(\delta(\{D\}, 1))$$
$$= \varepsilon\text{-closure}(\{\phi\})$$
$$= \phi$$



$$\delta_N (E, 0) = \varepsilon\text{-closure}\ (\delta\ (\varepsilon\text{-closure}\ (E), 0))$$

$$= \varepsilon\text{-closure}\ (\delta\ (\{E\}, 0))$$

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

$$= \phi$$

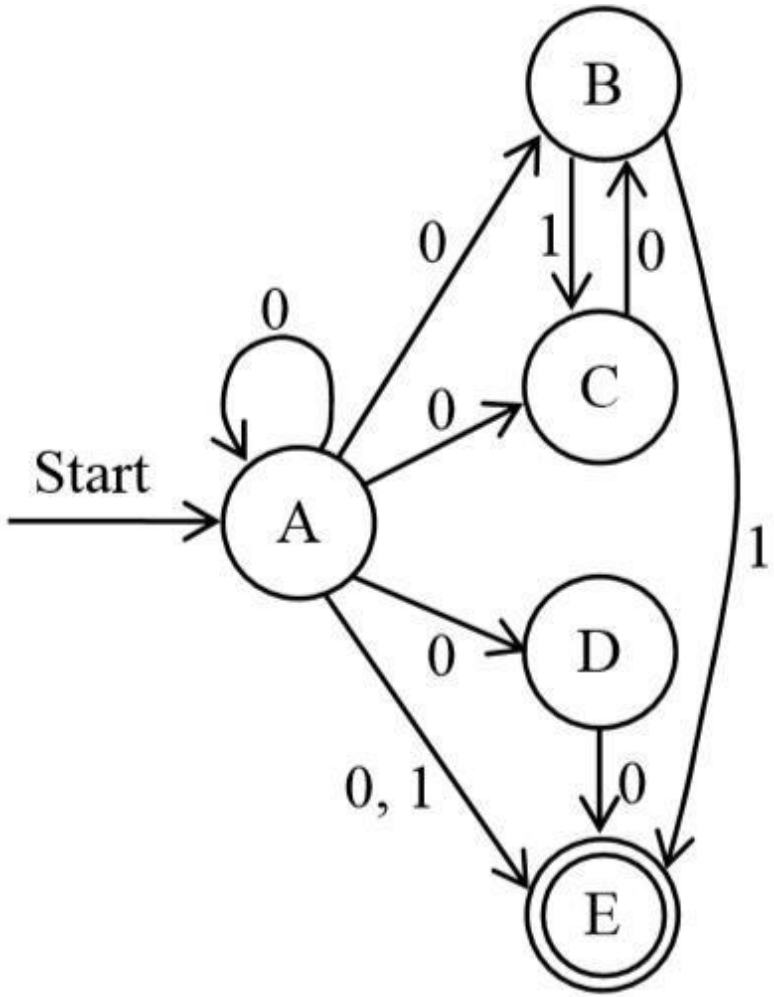
$$\delta_N (E, 1) = \varepsilon\text{-closure}\ (\delta\ (\varepsilon\text{-closure}\ (E), 1))$$

$$= \varepsilon\text{-closure}\ (\delta\ (\{E\}, 1))$$

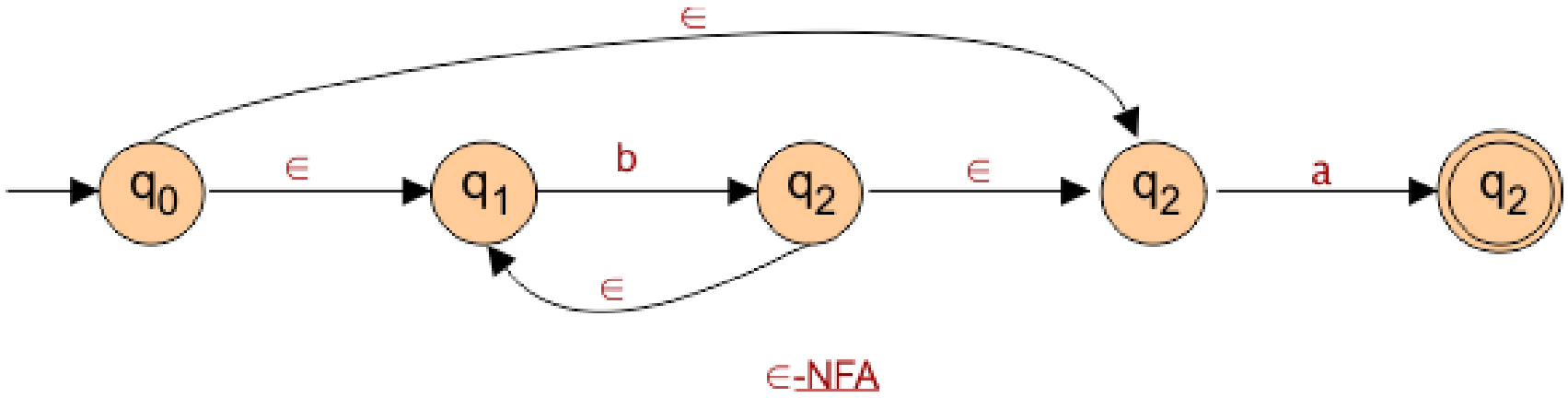
$$= \varepsilon\text{-closure}\ (\{\phi\})$$

$$= \phi$$

Thus, the resulting NFA is:

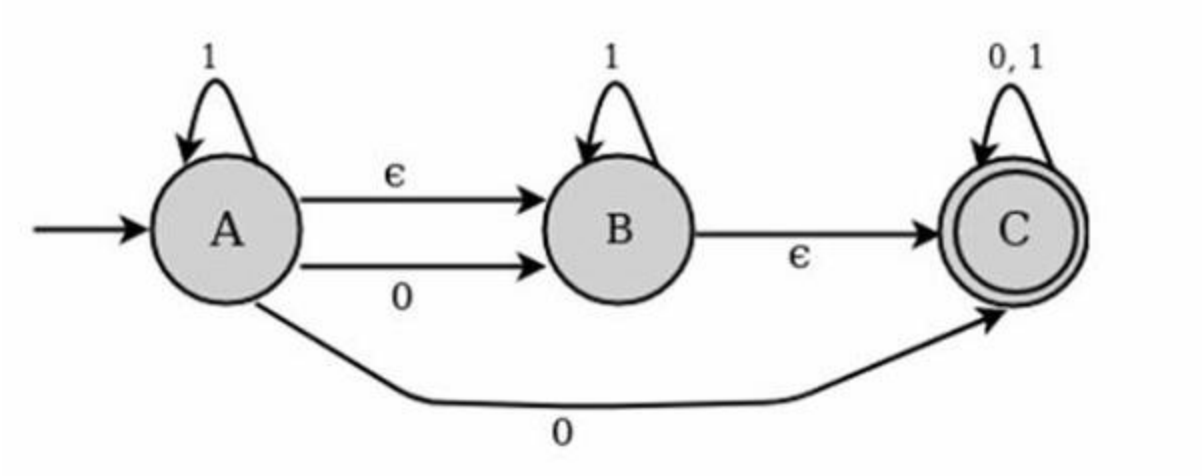


Practice1: e-NFA to NFA



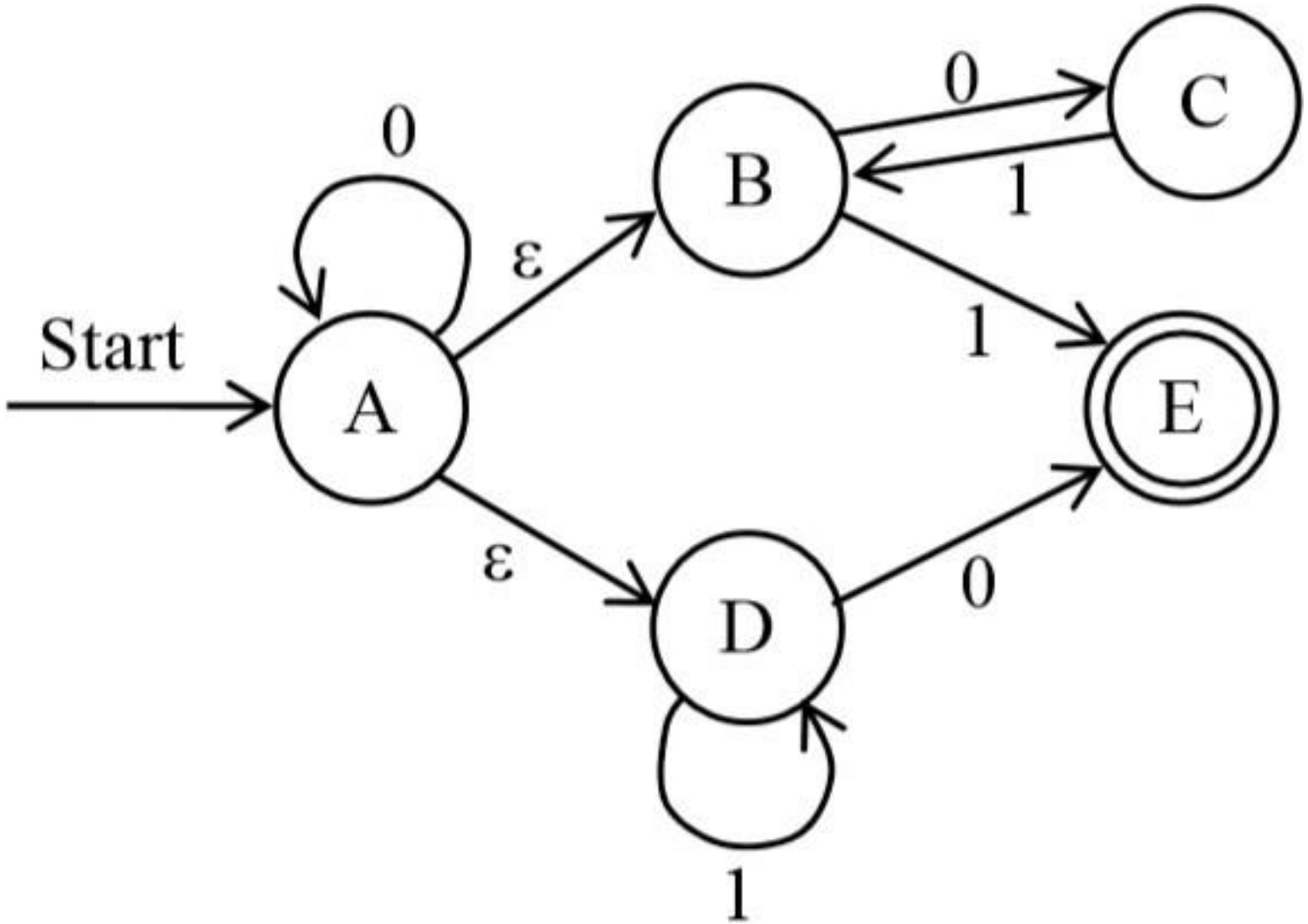
∈-NFA

Practice2: e-NFA to NFA

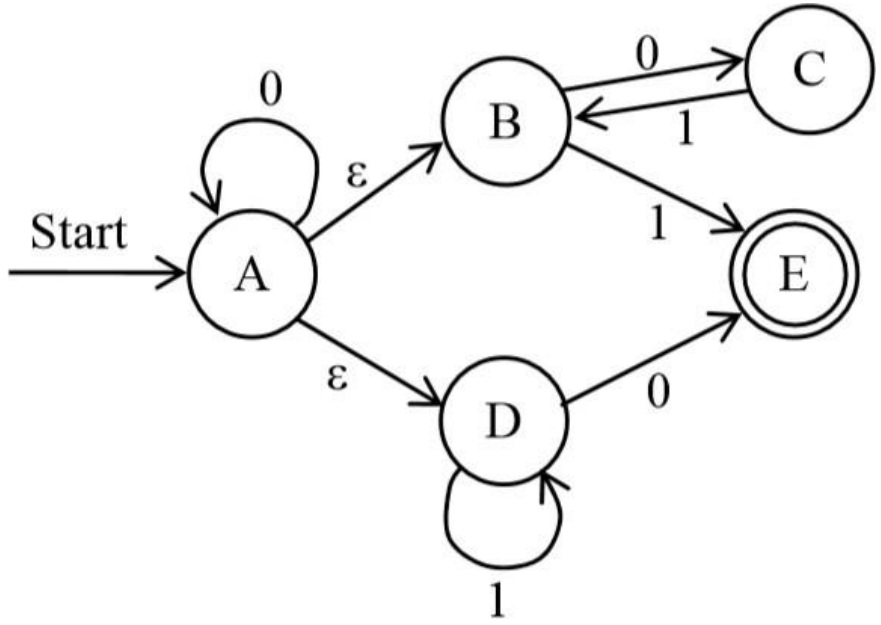


2) ε-NFA to DFA

- Given an ε-NFA E = (Q, ∑, δ, q_0 , F), to construct a DFA equivalent to E, let D = (Q', ∑, δ',q'_0 , F') is a DFA equivalent to E.


Start
A
B
C
D
E
0
0
1
ε
1
ε
0
1

- The start state of given ε-NFA is A,
- So start state for DFA will be;
- q'_0 = ε-closure (A) = {A, B, D}



$$\delta'(\{A, B, D\}, 0) = \varepsilon\text{-closure}(\delta(\{A, B, D\}, 0))$$

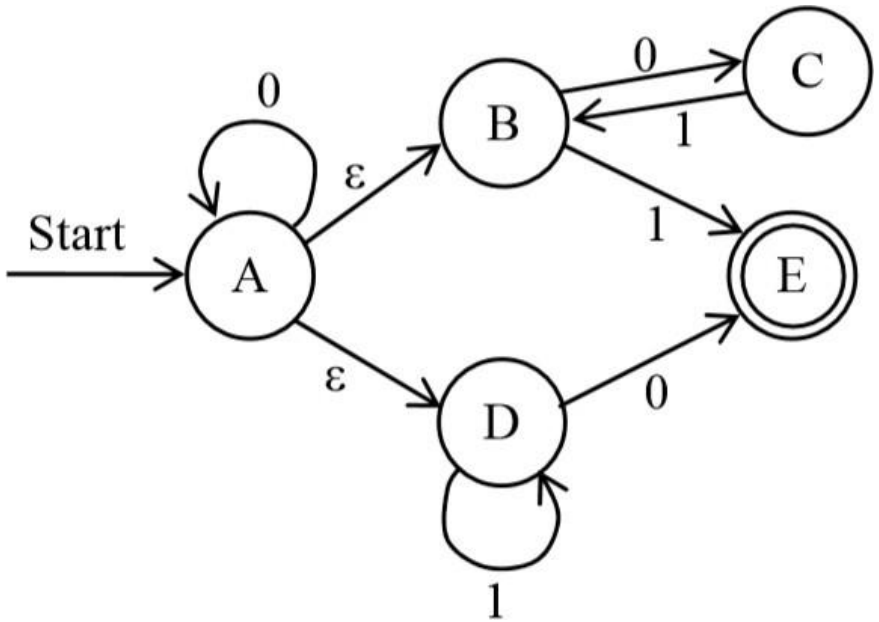
$$= \varepsilon\text{-closure}(\{A, C, E\})$$

$$= \{A, B, C, D, E\}$$

$$\delta'(\{A, B, D\}, 1) = \varepsilon\text{-closure}(\delta(\{A, B, D\}, 1))$$

$$= \varepsilon\text{-closure}(\{D, E\})$$

$$= \{D, E\}$$



$$\delta'(\{A, B, C, D, E\}, 0) = \varepsilon\text{-closure}(\delta(\{A, B, C, D, E\}, 0))$$

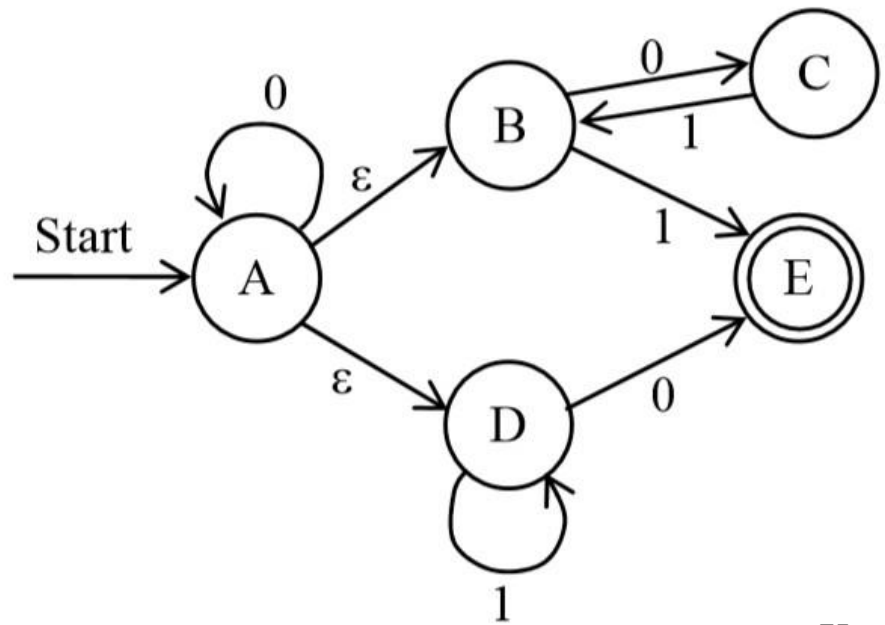
$$= \varepsilon\text{-closure}(\{A, C, E\})$$

$$= \{A, B, C, D, E\}$$

$$\delta'(\{A, B, C, D, E\}, 1) = \varepsilon\text{-closure}(\delta(\{A, B, C, D, E\}, 1))$$

$$= \varepsilon\text{-closure}(\{B, D, E\})$$

$$= \{B, D, E\}$$



$$\delta'(\{D, E\}, 0) = \varepsilon\text{-closure}(\delta(\{D, E\}, 0))$$

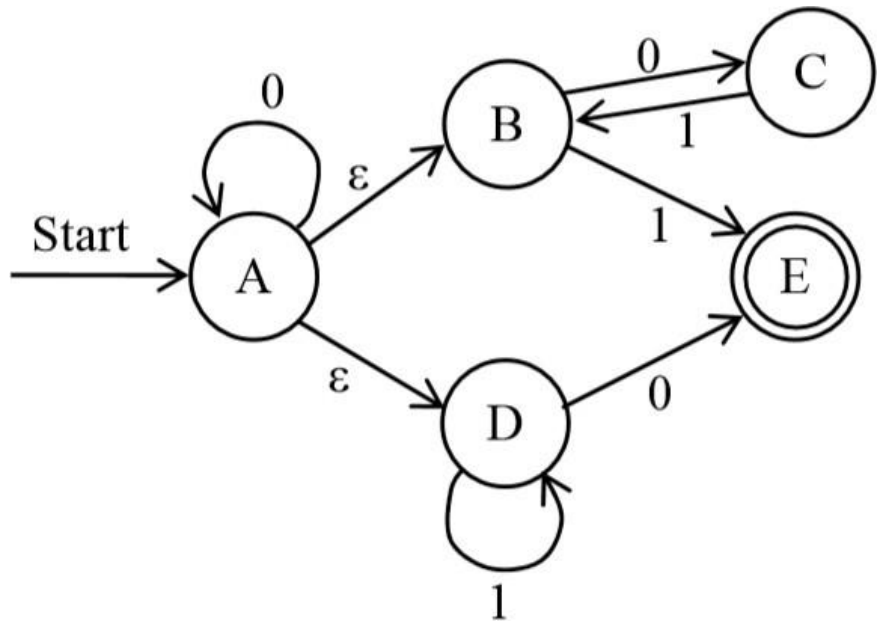
$$= \varepsilon\text{-closure}(\{E\})$$

$$= \{E\}$$

$$\delta'(\{D, E\}, 1) = \varepsilon\text{-closure}(\delta(\{D, E\}, 1))$$

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

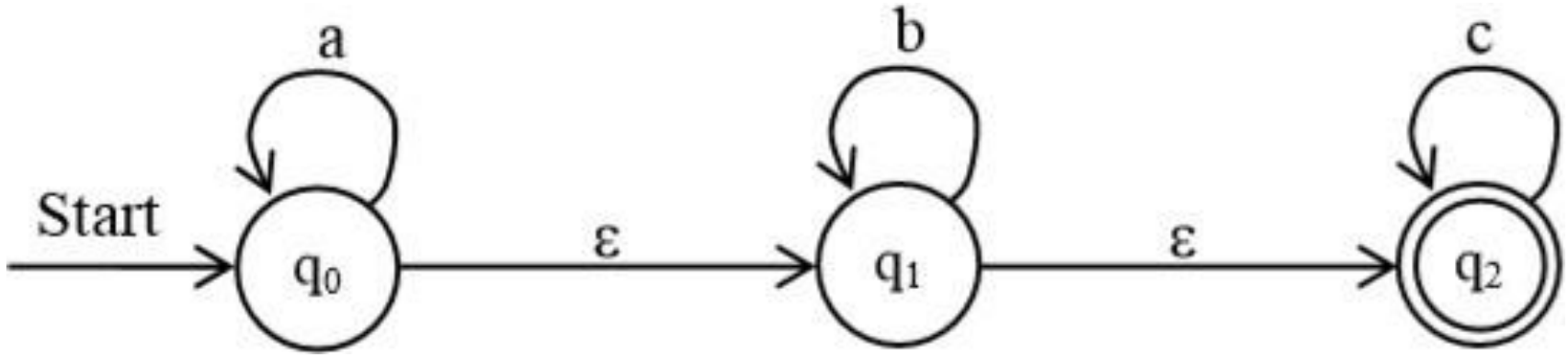
$$= \{D\}$$



we calculate remaining states in same way.

	0	1
→ {A, B, D}	{A, B, C, D, E}	{E, D}
*{A, B, C, D, E}	{A, B, C, D, E}	{B, D, E}
*{D, E}	{E}	{D}
*{B, D, E}	{C, E}	{E, D}
*{E}	ϕ	ϕ
{D}	{E}	{D}
*{C, E}	ϕ	{B}
{B}	{C}	{E}
{C}	ϕ	{B}
ϕ	ϕ	ϕ

Practice: Conversion of ε-NFA into NFA & DFA



Theorem 1:

- For any NFA, N = (QN, ∑, δ N, q0, FN) accepting language L ⊆ ∑* there is a DFA D = (QD, ∑, δ D, q0’,FD) that also accepts L i.e. L (N) = L (D).

Theorem 2:

- A language L is accepted by some NFA if L is accepted by some DFA.

Finite State Machines with output

- Moore machine and Mealy Machines

Mealy Machine

A Mealy Machine is an FSM whose output depends on the present state as well as the present input.
It can be described by a 6 tuple $(Q, \sum, O, \delta, X, q_0)$ where −

- **Q** is a finite set of states.
- $\sum$ is a finite set of symbols called the input alphabet.
- **O** is a finite set of symbols called the output alphabet.
- **δ** is the input transition function where $\delta: Q \times \sum \rightarrow Q$
- **X** is the output transition function where $X: Q \times \sum \rightarrow O$
- $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).

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

Moore Machine

Moore machine is an FSM whose outputs depend on only the present state.

A Moore machine can be described by a 6 tuple (Q, ∑, O, δ, X, q_0) where −

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

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