Formal Language and Automata Theory

Branch:CSE **Sem:5th**

Faculty:A.K.Rout

Alphabet:

- **Definition** − An **alphabet** is any finite set of symbols.
- **Example** − ∑ = {a, b, c, d} is an **alphabet set** where ‘a’, ‘b’, ‘c’, and ‘d’ are **symbols**.

String:

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

Length of a String:

- **Definition** − It is the number of symbols present in a string. (Denoted by **|S|**).
- **Examples** −
  - If S = ‘cabcad’, |S|= 6
  - If |S|= 0, it is called an **empty string** (Denoted by **λ** or **ε**)

Kleene Star:

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

- **Example** − If ∑ = {a, b}, ∑* = {λ, a, b, aa, ab, ba, bb,…….}

Kleene Closure / Plus

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

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

Language:

- **Definition** − A language is a subset of ∑* for some alphabet ∑. It can be finite or infinite.
- **Example** − If the language takes all possible strings of length 2 over ∑ = {a, b}, then L = { ab, aa, ba, bb }

NB:Treat lamda(λ) as epsilon(€)

Automata:

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

An automaton with a finite number of states is called a **Finite Automaton** (FA) or **Finite State Machine** (FSM).

Formal definition of a Finite Automaton

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

- **Q** is a finite set of states.
- ∑ is a finite set of symbols, called the **alphabet** of the automaton.
- **δ** is the transition function.
- **q_0** is the initial state from where any input is processed ($q_0 \in Q$).

- **F** is a set of final state/states of Q ($F \subseteq Q$).

Finite Automaton can be classified into two types −

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

Deterministic Finite Automaton (DFA)

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

Formal Definition of a DFA

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

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

Graphical Representation of a DFA

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

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

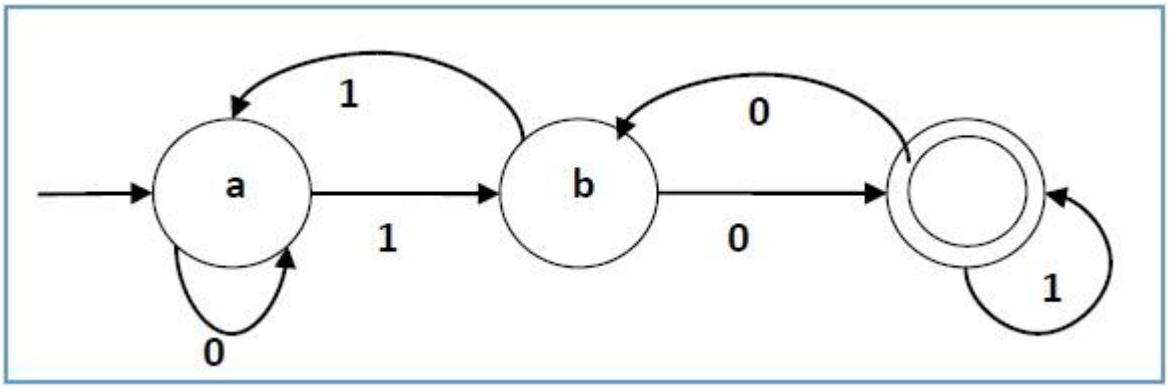
Example

Let a deterministic finite automaton be →

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

Transition function δ as shown by the following table –(Transition Table)

Present State(δ/∑)	Next State for Input 0	Next State for Input 1
a	a	b
b	c	a
c	b	c



Non-deterministic Finite Automaton

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

the machine is called **Non-deterministic Finite Machine** or **Non-deterministic Finite Automaton**.

Formal Definition of an NDFA

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

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

  (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).
- **F** is a set of final state/states of Q ($F \subseteq Q$).

Graphical Representation of an NDFA: (same as DFA)

An NDFA is represented by digraphs called state diagram.

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

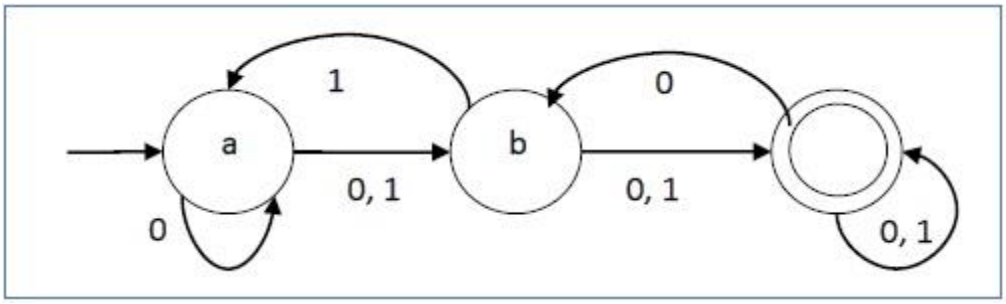
Example

Let a non-deterministic finite automaton be →

- $Q = \{a, b, c\}$
- $\sum = \{0, 1\}$
- $q_0 = \{a\}$
- $F = \{c\}$

The transition function δ as shown below –

Present State(δ/∑)	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c



DFA vs NDFA

The following table lists the differences between DFA and NDFA.

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is called *deterministic*.	The transition from a state can be to multiple next states for each input symbol. Hence it is called *non-deterministic*.
Empty string transitions are not seen in DFA.	NDFA permits empty string transitions.
Backtracking is allowed in DFA	In NDFA, backtracking is not always possible.
Requires more space.	Requires less space.
A string is accepted by a DFA, if it transits to a final state.	A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state

Acceptors, Classifiers:

Acceptor (Recognizer)

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

Classifier

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

Acceptability by DFA and NDFA

A string is accepted by a DFA/NDFA iff the DFA/NDFA starting at the initial state ends in an accepting state (any of the final states) after reading the string wholly.

A string S is accepted by a DFA/NDFA (Q, ∑, δ, q_0, F), iff

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

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

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

A string S′ is not accepted by a DFA/NDFA (Q, ∑, δ, q_0, F), iff

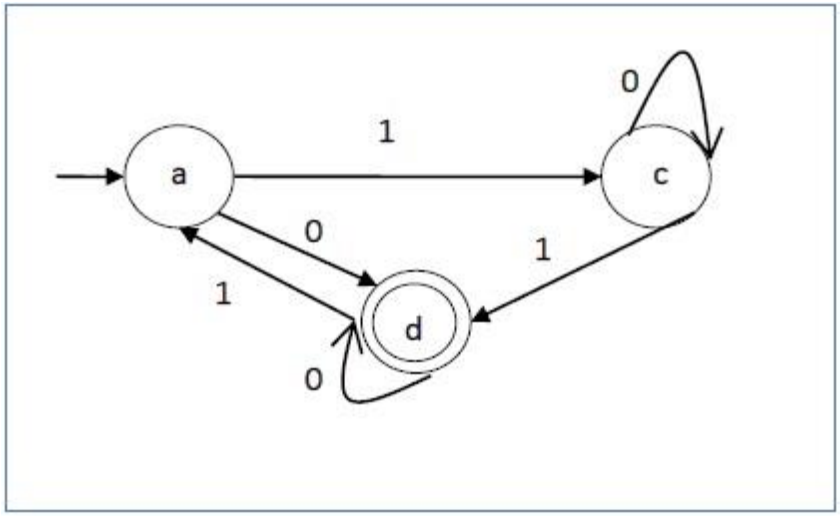
$\delta^*(q_0, S') \notin F$

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

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

Example

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



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

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

Equivalence of NFA and DFA:

Let $\mathbf{X = (Q_x, \sum, \delta_x, q_0, F_x)}$ be an NDFA which accepts the language L(X). We have to design an equivalent DFA $\mathbf{Y = (Q_y, \sum, \delta_y, q_0, F_y)}$ such that **L(Y) = L(X)**. The following procedure converts the NDFA to its equivalent DFA −

Algorithm:

Input − An NDFA

Output − An equivalent DFA

Step 1 − Create state table from the given NDFA.

Step 2 − Create a blank state table under possible input alphabets for the equivalent DFA.

Step 3 − Mark the start state of the DFA by q0 (Same as the NDFA).

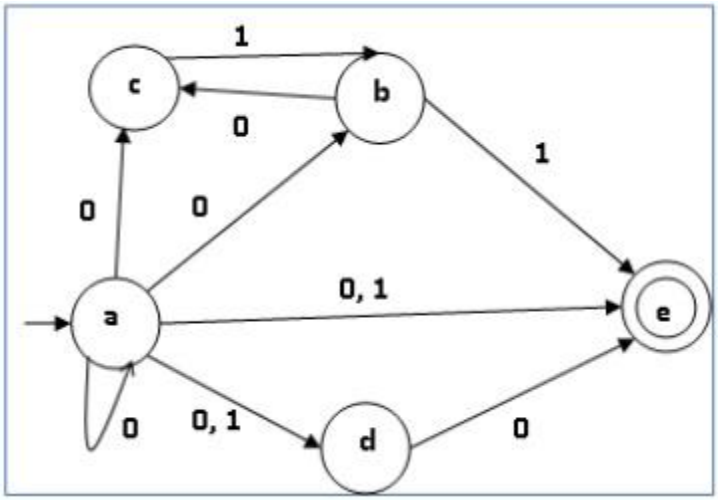
Step 4 − Find out the combination of States $\{Q_0, Q_1, \ldots, Q_n\}$ for each possible input alphabet.

Step 5 − Each time we generate a new DFA state under the input alphabet columns, we have to apply step 4 again, otherwise go to step 6.

Step 6 − The states which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example

Let us consider the NDFA shown in the figure below.

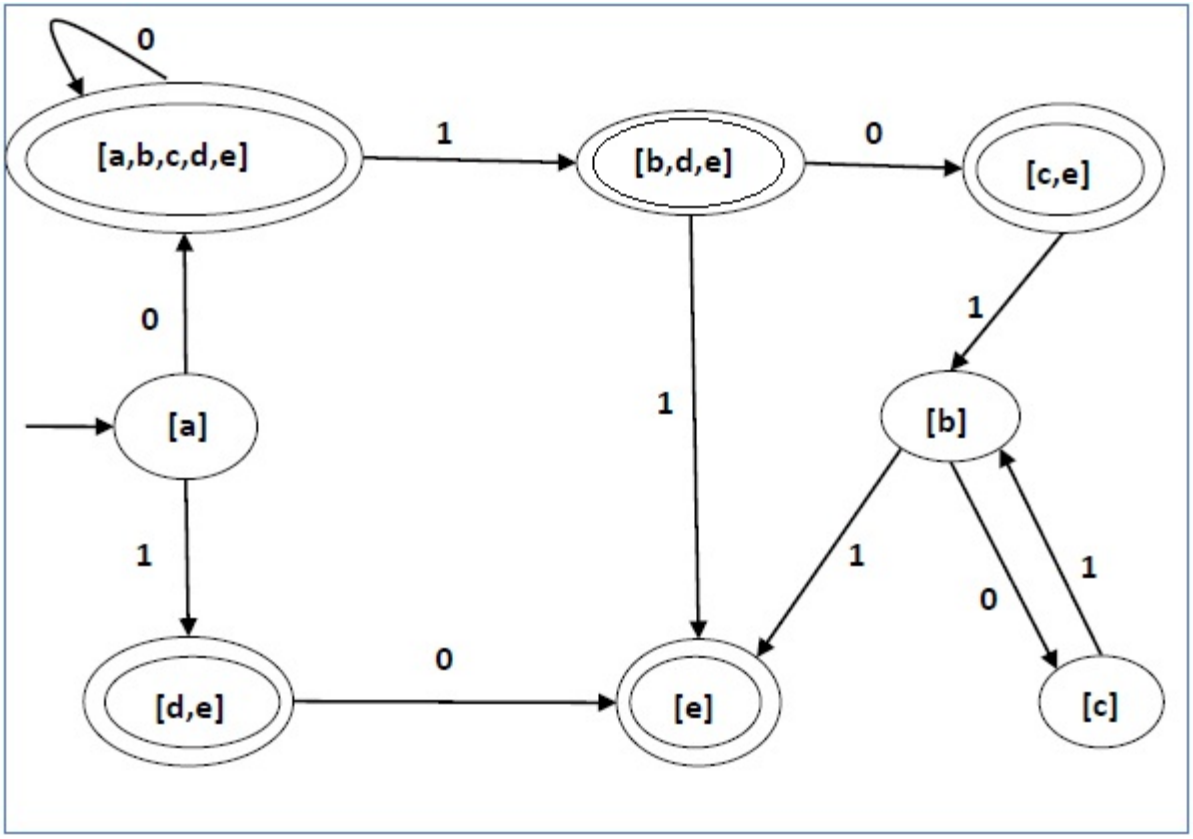


q	δ(q,0)	δ(q,1)
a	{a,b,c,d,e}	{d,e}
b	{c}	{e}
c	∅	{b}
d	{e}	∅
e	∅	∅

Using the above algorithm, we find its equivalent DFA. The state table of the DFA is shown in below.

q	δ(q,0)	δ(q,1)
[a]	[a,b,c,d,e]	[d,e]
[a,b,c,d,e]	[a,b,c,d,e]	[b,d,e]
[d,e]	[e]	∅
[b,d,e]	[c,e]	[e]
[e]	∅	∅
[c, e]	∅	[b]
[b]	[c]	[e]
[c]	∅	[b]

The state diagram of the DFA is as follows –



DFA Minimization

DFA Minimization using Myphill-Nerode Theorem

Algorithm:

Input − DFA

Output − Minimized DFA

Step 1 − Draw a table for all pairs of states (Q_i, Q_j) not necessarily connected directly [All are unmarked initially]

Step 2 − Consider every state pair (Q_i, Q_j) in the DFA where $Q_i \in F$ and $Q_j \notin F$ or vice versa and mark them. [Here F is the set of final states]

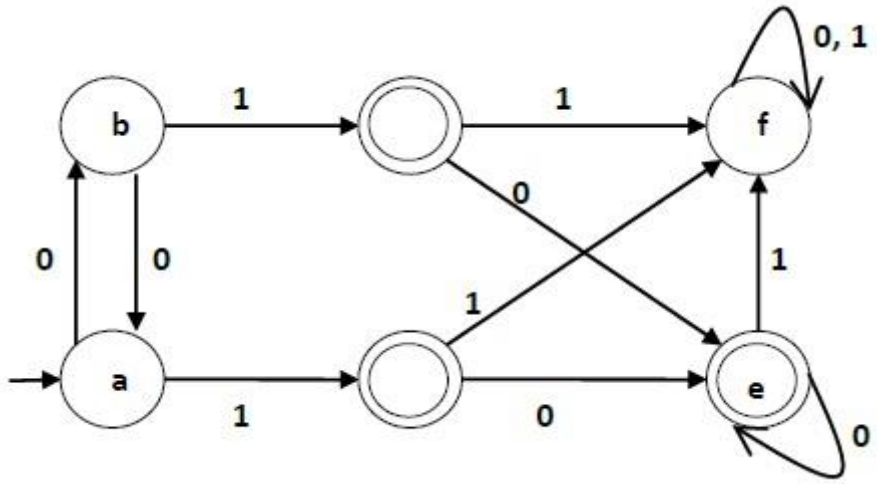
Step 3 − Repeat this step until we cannot mark anymore states −

If there is an unmarked pair (Q_i, Q_j), mark it if the pair {δ (Q_i, A), δ (Q_i, A)} is marked for some input alphabet.

Step 4 − Combine all the unmarked pair (Q_i, Q_j) and make them a single state in the reduced DFA.

Example:

Let us use Algorithm 2 to minimize the DFA shown below.



Step 1 − We draw a table for all pair of states.

	a	**b**	**c**	**d**	**e**	**f**
a						
b						
c						
d						
e						
f						

Step 2 − We mark the state pairs.

	a	b	c	d	e	f
a						
b						
c	✔	✔				
d	✔	✔				
e	✔	✔				
f			✔	✔	✔	

Step 3 − We will try to mark the state pairs, with green colored check mark, transitively. If we input 1 to state ‘a’ and ‘f’, it will go to state ‘c’ and ‘f’ respectively. (c, f) is already marked, hence we will mark pair (a, f). Now, we input 1 to state ‘b’ and ‘f’; it will go to state ‘d’ and ‘f’ respectively. (d, f) is already marked, hence we will mark pair (b, f).

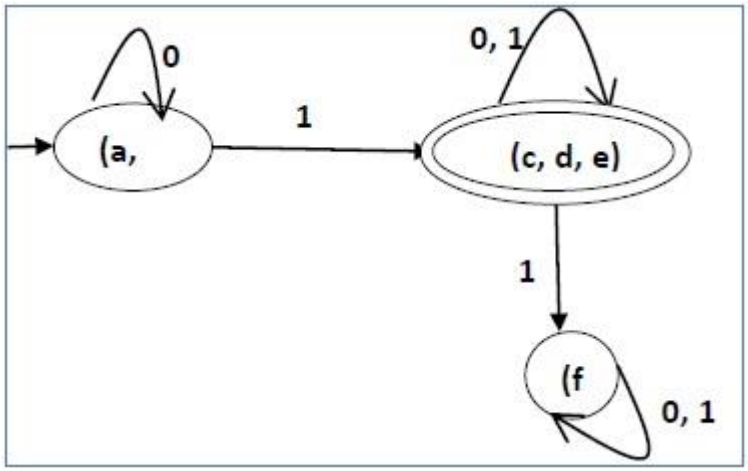
	a	b	c	d	e	f
a						
b						
c	✔	✔				
d	✔	✔				
e	✔	✔				
f	✔	✔	✔	✔	✔	

After step 3, we have got state combinations {a, b} {c, d} {c, e} {d, e} that are unmarked.

We can recombine {c, d} {c, e} {d, e} into {c, d, e}

Hence we got two combined states as − {a, b} and {c, d, e}

So the final minimized DFA will contain three states {f}, {a, b} and {c, d, e}



DFA Minimization using Equivalence Theorem

If X and Y are two states in a DFA, we can combine these two states into {X, Y} if they are not distinguishable. Two states are distinguishable, if there is at least one string S, such that one of δ (X, S) and δ (Y, S) is accepting and another is not accepting. Hence, a DFA is minimal if and only if all the states are distinguishable.

Algorithm:

Step 1 − All the states **Q** are divided in two partitions − **final states** and **non-final states** and are denoted by $\mathbf{P_0}$. All the states in a partition are 0^{th} equivalent. Take a counter **k** and initialize it with 0.

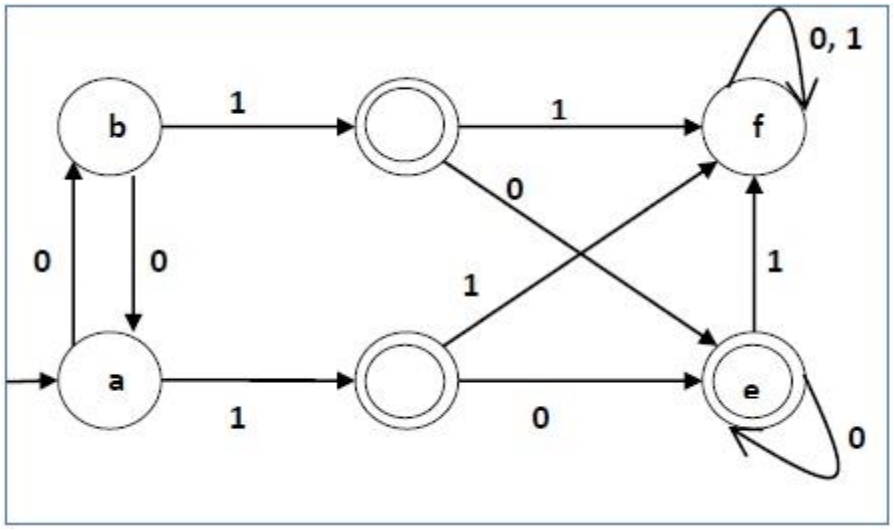
Step 2 − Increment k by 1. For each partition in P_k, divide the states in P_k into two partitions if they are k-distinguishable. Two states within this partition X and Y are k-distinguishable if there is an input **S** such that **δ(X, S)** and **δ(Y, S)** are (k-1)-distinguishable.

Step 3 − If $P_k \neq P_{k-1}$, repeat Step 2, otherwise go to Step 4.

Step 4 − Combine k^{th} equivalent sets and make them the new states of the reduced DFA.

Example

Let us consider the following DFA −



q	δ(q,0)	δ(q,1)
a	b	c
b	a	d
c	e	f
d	e	f
e	e	f
f	f	f

Let us apply the above algorithm to the above DFA −

- P_0 = {(c,d,e), (a,b,f)}
- P_1 = {(c,d,e), (a,b),(f)}
- P_2 = {(c,d,e), (a,b),(f)}

Hence, $P_1 = P_2$.

There are three states in the reduced DFA. The reduced DFA is as follows –

Q	**δ(q,0)**	**δ(q,1)**
(a, b)	(a, b)	(c,d,e)
(c,d,e)	(c,d,e)	(f)
(f)	(f)	(f)

Nondeterministic Finite Automata with ε transitions (ε-NFA) :

• For both DFAs and NFAs, you must read a symbol in order for the machine to make a move.

• In Nondeterministic Finite Automata with ε transitions (ε-NFA) – Can make move without reading a symbol of the read tape – Such a move is called a ε transition

1. The automaton may be allowed to change its state without reading the input symbol.

2. In diagrams, such transitions are depicted by labeling the appropriate arcs with ε.

3. Note that this does not mean that ε has become an input symbol. On the contrary, we assume that the symbol ε does not belong to any alphabet.