

Course Title: Formal Languages and Automata

Semester III Term Odd

Teaching Scheme

Total 75

Credits:3

Course Code

UCSL203



UNIT –I

INTRODUCTION TO FLA

Introduction- Basic Mathematical Notation and techniques- Finite State systems – Basic Definitions – Finite Automaton – DFA & NDFA – Finite Automaton with ϵ - moves – Regular Languages- Regular Expression – Equivalence of NFA and DFA – Equivalence of NDFA's with and without ϵ -moves – Equivalence of finite Automaton

Prof. Padmavati Sarode
S.Y B.Tech
GHRCEM

What is Formal Language

In logic ,mathematics, computer science and linguistics, a formal language consists of words whose letters are taken from an alphabet and are well-formed according to a specific set of rules.

The alphabet of a formal language consists of symbols, letters or tokens that concatenate into strings of the language.

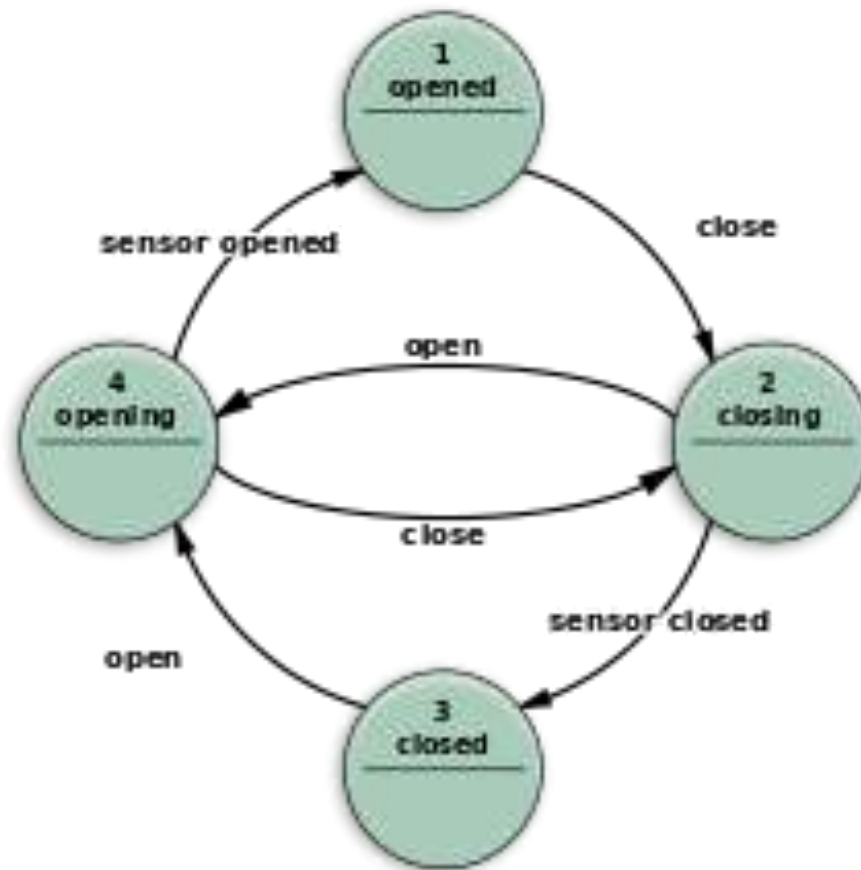
- natural (human) languages and programming languages.
 1. Learning its alphabet - the symbols that are used in the language.
 2. Its words - as various sequences of symbols of its alphabet.
 3. Formation of sentences - sequence of various words that follow certain rules of the language.

Mathematical notations

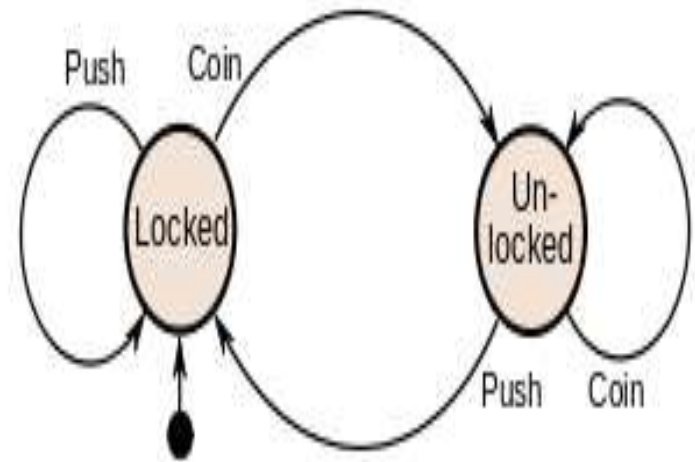
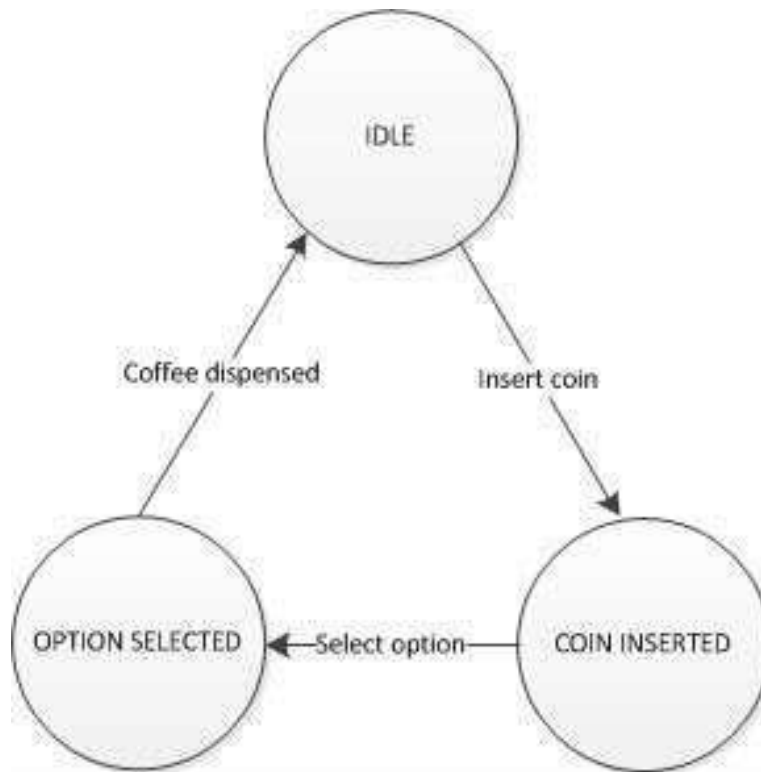
- See in pdf
- [Mathematical notations.pdf](#)

What is finite state system?

- A finite-state machine (FSM) or finite-state automaton (FSA, plural: automata), finite automaton, or simply a state machine, is a **mathematical model of computation**. It is an abstract machine that can be in exactly one of a finite number of states at any given time.



Implementing finite state machines



A finite automaton (FA)

is a simple idealized machine used to recognize patterns within input taken from some character set (or alphabet) C .

The job of an FA is to accept or reject an input depending on whether the pattern defined by the FA occurs in the input. A finite automaton consists of: a finite set S of N states.

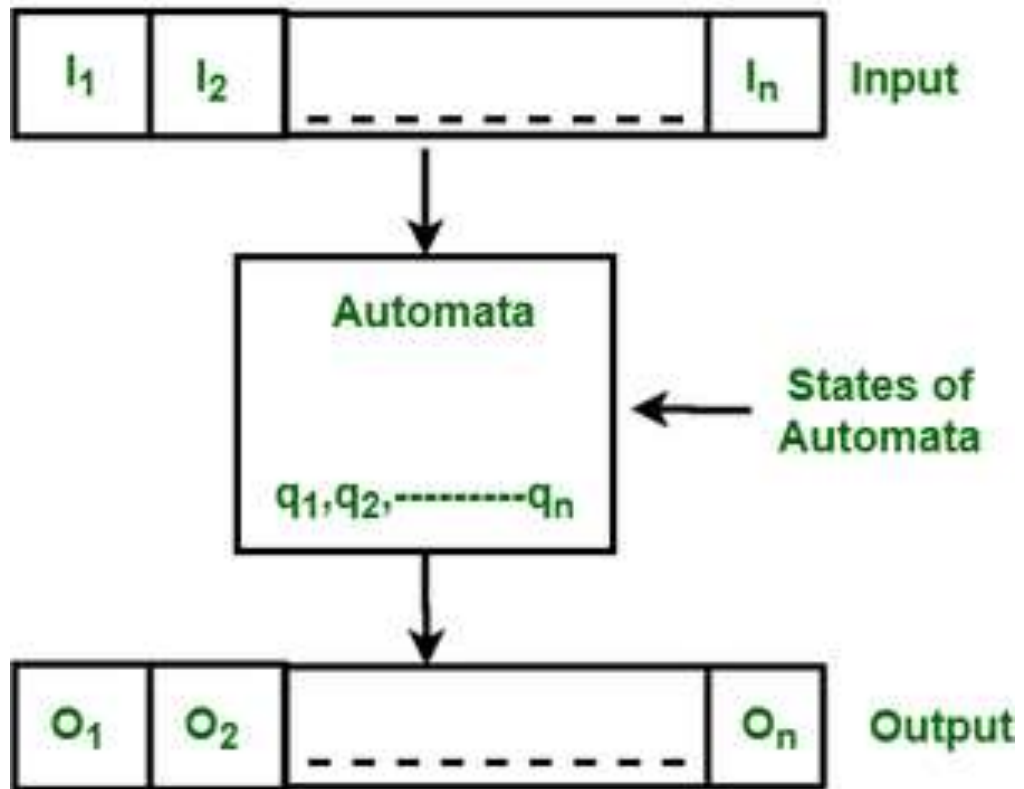
Formal Definition of a Finite Automaton

1. Finite set of states, typically Q .
 2. Alphabet of input symbols, typically Σ
 3. One state is the start/initial state, typically q_0 // $q_0 \in Q$
 4. Zero or more final/accepting states; the set is typically F . // $F \subseteq Q$
 5. A transition function, typically δ . This function
 - Takes a state and input symbol as arguments.
 - Returns a state.
 - One “rule” would be written $\delta(q, a) = p$, where q and p are states, and a is an input symbol.
 - Intuitively: if the FA is in state q , and input a is received, then the FA goes to state p (note: $q = p$ OK).
1. A FA is represented as the five-tuple: $A = (Q, \Sigma, \delta, q_0, F)$. Here, F is a set of accepting states.

FA

A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the *states*,
2. Σ is a finite set called the *alphabet*,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the *transition function*,¹
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the *set of accept states*.²



DFA

DFA refers to **Deterministic Finite Automaton**. A Finite Automata (FA) is said to be deterministic, if corresponding to an input symbol, there is single resultant state i.e. there is only one transition.

Deterministic Finite Automata

Definition: A deterministic finite automaton (DFA) consists of

1. a finite set of *states* (often denoted Q)
2. a finite set Σ of *symbols* (alphabet)
3. a *transition function* that takes as argument a state and a symbol and returns a state (often denoted δ)
4. a *start state* often denoted q_0
5. a set of *final* or *accepting* states (often denoted F) We have $q_0 \in Q$ and $F \subseteq Q$

How do you define DFA?

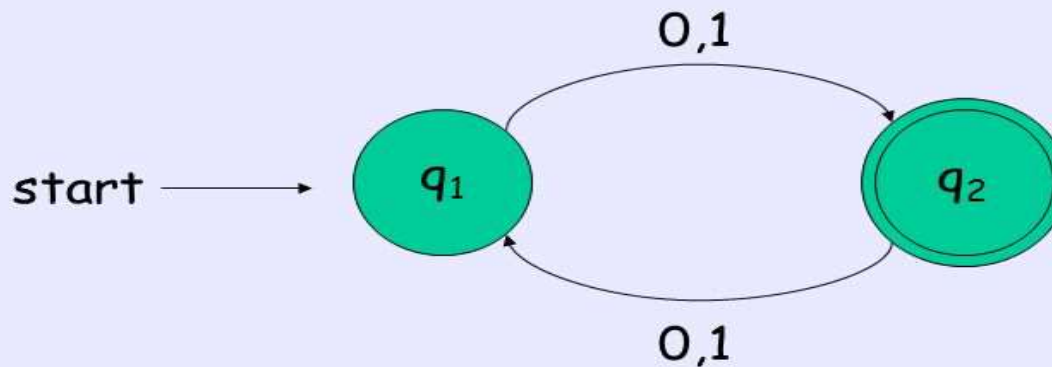
- DFA Formal Definition (reminder) A **deterministic finite automaton** (DFA) is a 5-tuple. $(Q, \Sigma, \delta, q_0, F)$, where. Q is a finite set called the states, Σ is a finite set called the alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, $q_0 \in Q$ is the start state, and $F \subseteq Q$ is the set of accept states.

DFA (Deterministic finite automata)

- DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.
- In DFA, there is only one path for specific input from the current state to the next state.
- DFA does not accept the null move, i.e., the DFA cannot change state without any input character.
- DFA can contain multiple final states. It is used in Lexical Analysis in Compiler.

DFA

Formal Definition of DFA



$Q = \{q_1, q_2\}, \Sigma = \{0, 1\}, q_{\text{start}} = q_1, F = \{q_2\}$

$\delta(q_1, 0) = q_2, \delta(q_1, 1) = q_2$

$\delta(q_2, 0) = q_1, \delta(q_2, 1) = q_1$

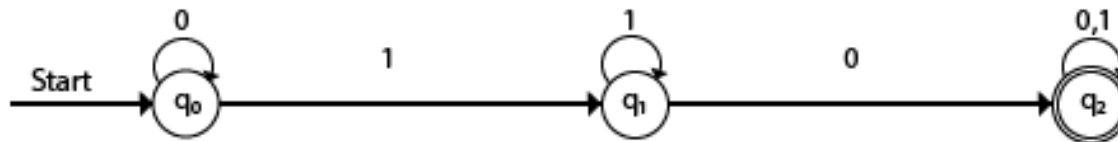
Graphical Representation of DFA

A DFA can be represented by digraphs called state diagram. In which:

- The state is represented by vertices.
- The arc labeled with an input character show the transitions.
- The initial state is marked with an arrow.
- The final state is denoted by a double circle.

Ex: $Q = \{q_0, q_1, q_2\}$
 $\Sigma = \{0, 1\}$
 $q_0 = \{q_0\}$
 $F = \{q_2\}$

Transition Diagram:



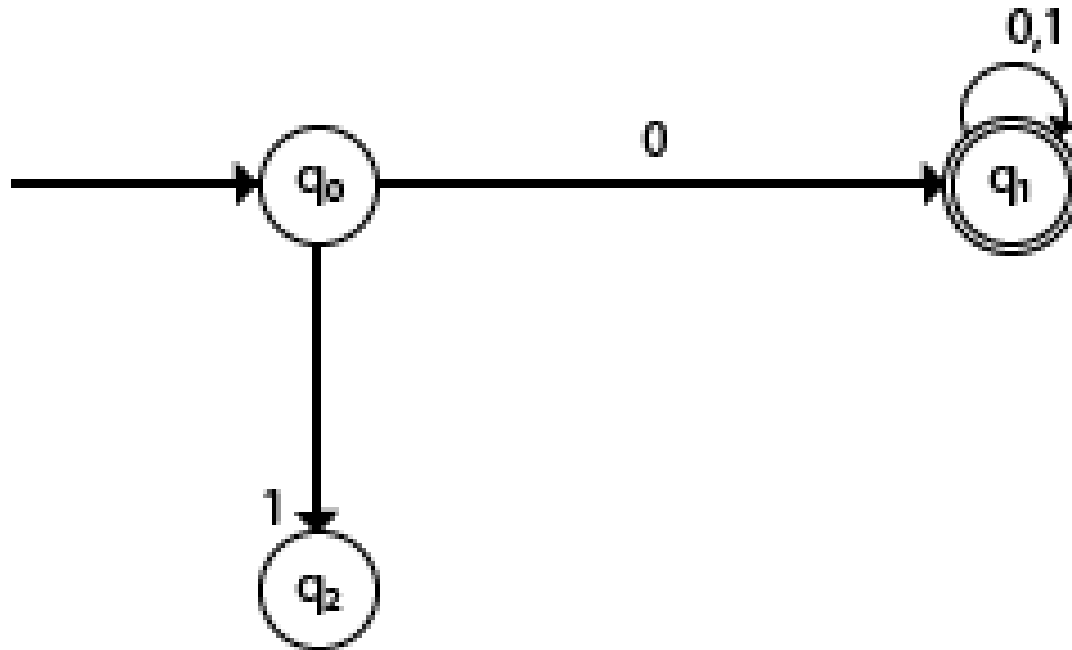
Transition Table:

Present State	Next state for Input 0	Next State of Input 1
→q0	q0	q1
q1	q2	q1
*q2	q2	q2

Example 2:

Example 2:

DFA with $\Sigma = \{0, 1\}$ accepts all input starting with 0.

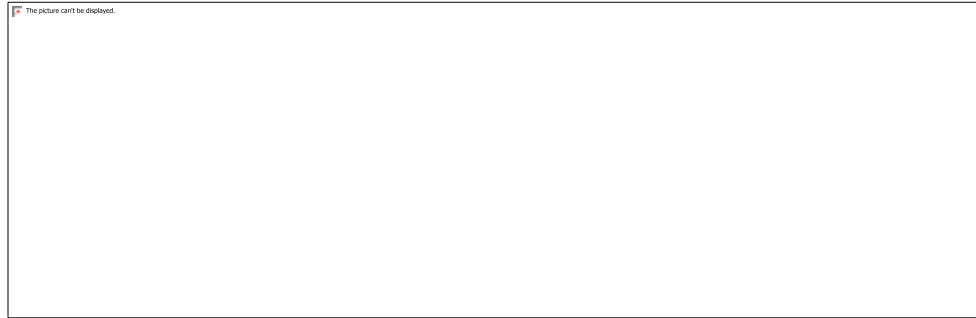


In the above diagram, we can see that on given 0 as input to DFA in state q_0 the DFA changes state to q_1 and always go to final state q_1 on starting input 0. It can accept 00, 01, 000, 001....etc. It can't accept any string which starts with 1, because it will never go to final state on a string starting with 1. see state q_2

Trap state is **one from which we cannot escape**, that is, it has either no transitions defined or transitions for all symbols goes to itself. I feel, definition on this page for dead state is more correct: Once the machine enters a dead state, there is no way for it to reach an accepting state.

As q2 state in above example.

Example :2 with trap state.

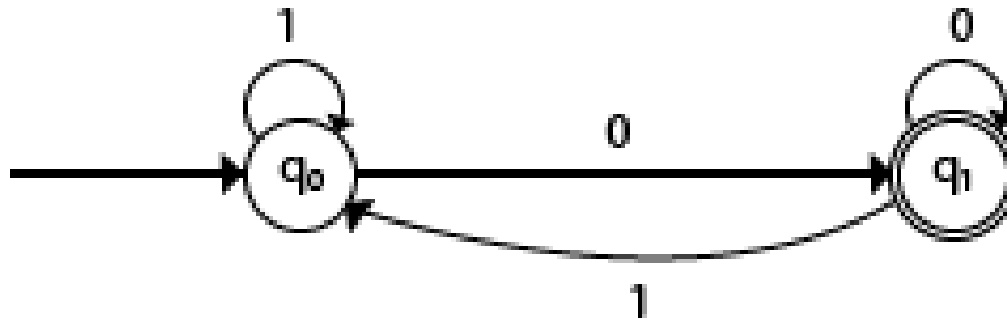


What is a dead state?

Dead State - **A rejecting state that is essentially a dead end.** Once the machine enters a dead state, there is no way for it to reach an accepting state, so we already know that the string is going to be rejected.

Example 3:

DFA with $\Sigma = \{0, 1\}$ See whether given transition diagram accepts all ending with 0 or not.



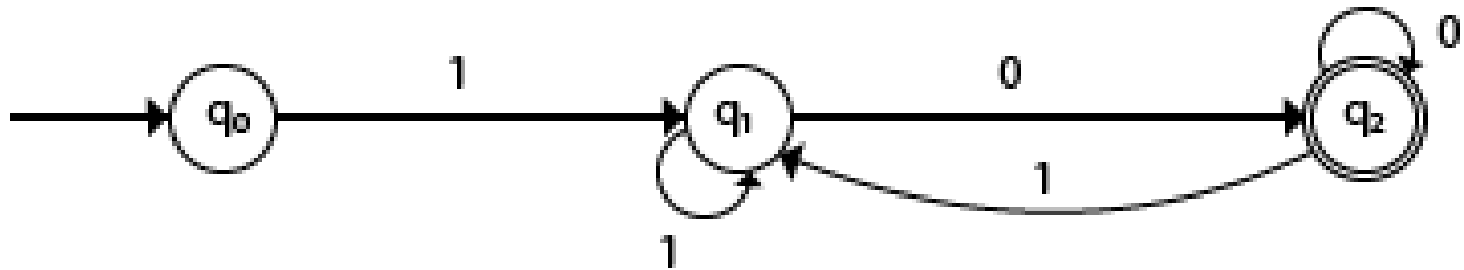
In the above diagram, we can see that on given 0 as input to DFA in state q_0 , the DFA changes state to q_1 . It can accept any string which ends with 0 like 00, 10, 110, 100....etc. It can't accept any string which ends with 1, because it will never go to the final state q_1 on 1 input, so the string ending with 1, will not be accepted or will be rejected.

Example to solve:

Design a FA with $\Sigma = \{0, 1\}$ accepts those string which starts with 1 and ends with 0.

Solution:

The FA will have a start state q_0 from which only the edge with input 1 will go to the next state.



In state q_1 , if we read 1, we will be in state q_1 , but if we read 0 at state q_1 , we will reach to state q_2 which is the final state. In state q_2 , if we read either 0 or 1, we will go to q_2 state or q_1 state respectively. Note that if the input ends with 0, it will be in the final state.

Example: Design a FA with $\Sigma = \{0, 1\}$ accepts the only input 101.

Example :

Design FA with $\Sigma = \{0, 1\}$ accepts even number of 0's and even number of 1's.

Solution:

1

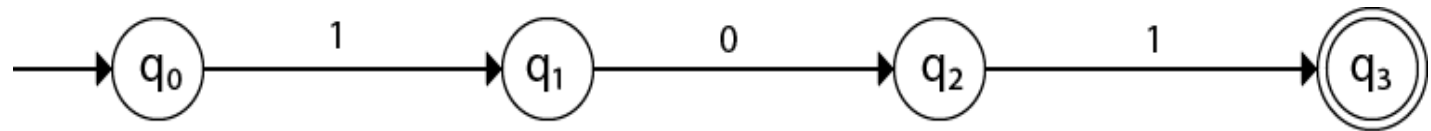
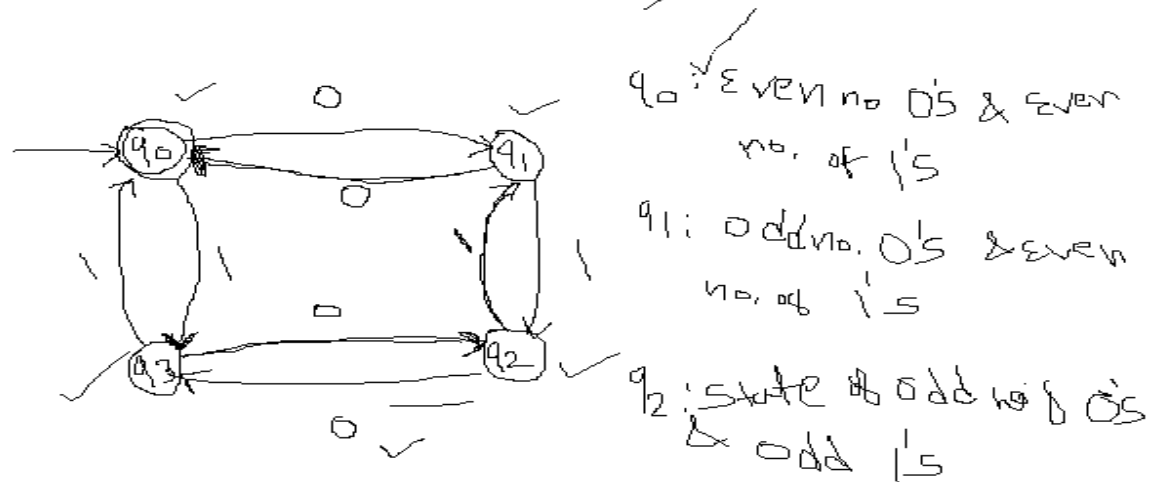


Fig: FA

$$\Sigma = \{0, 1\} \quad L = \{001, 110, 100\}$$

2



q_0 : EVEN no 0's & EVEN no. of 1's

q_1 : ODD no. 0's & EVEN no. of 1's

q_2 : STATE of ODD no. of 0's & ODD 1's

q_3 : STATE with EVEN no. of 0's & ODD no. 1's

NFA or NDFA

- NFA stands for **non-deterministic finite automata**. It is easy to construct an NFA than DFA for a given regular language. The finite automata are called NFA when there exist many paths for specific input from the current state to the next state. Every NFA is not DFA, but each NFA can be translated into DFA.

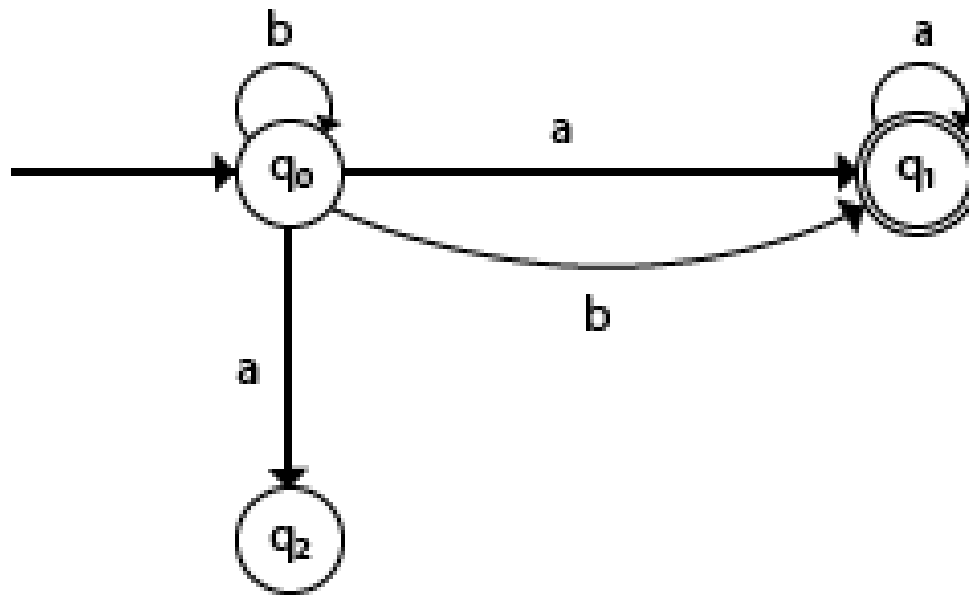


Fig:- NFA

NFA

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

Formal Definition of an NDFA

- Formal Definition of an NDFA
- An NDFA can be represented by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where –
- Q is a finite set of states.
- Σ is a finite set of symbols called the alphabets.
- δ is the transition function where $\delta: Q \times \Sigma \rightarrow 2^Q$
- (Here the power set of Q (2^Q) has been taken because in case of NDFA, from a state, transition can occur to any combination of Q states)
- q_0 is the initial state from where any input is processed ($q_0 \in Q$).
- F is a set of final state/states of Q ($F \subseteq Q$).

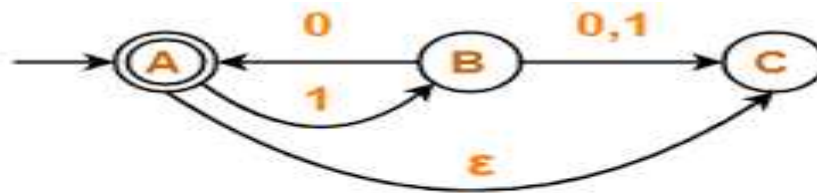
Nondeterministic finite automaton(NDFA)

In automata theory, a finite-state machine is called a deterministic finite automaton, if each of its transitions is uniquely determined by its source state and input symbol, and reading an input symbol is required for each state transition.

Formal Definition of an NDFA

Q is a **finite set of states**. Σ is a finite set of symbols called the alphabets. 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 ($F \subseteq Q$).

DFA stands for **Deterministic Finite Automata** and NDFA stands for Non-Deterministic Finite Automata.



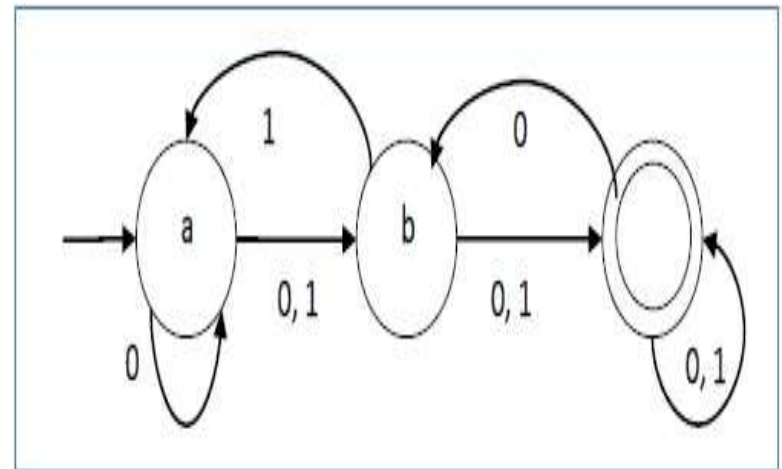
Example of Non-Deterministic Finite Automata
(With Epsilon)

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

The transition function δ as shown below –

Present State	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

DFA	NDFA
The transition from a state is to a single particular next state for each input symbol. Hence it is called <i>deterministic</i> .	The transition from a state can be to multiple next states for each input symbol. Hence it is called <i>non-deterministic</i> .
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, and Transducers

- **Acceptor (Recognizer)**

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

- **Classifier**

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

- **Transducer**

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

Mealy Machine – The output depends both on the current state and the current input.

Moore Machine – The output depends only on the current state.

Acceptability by DFA and NDFA

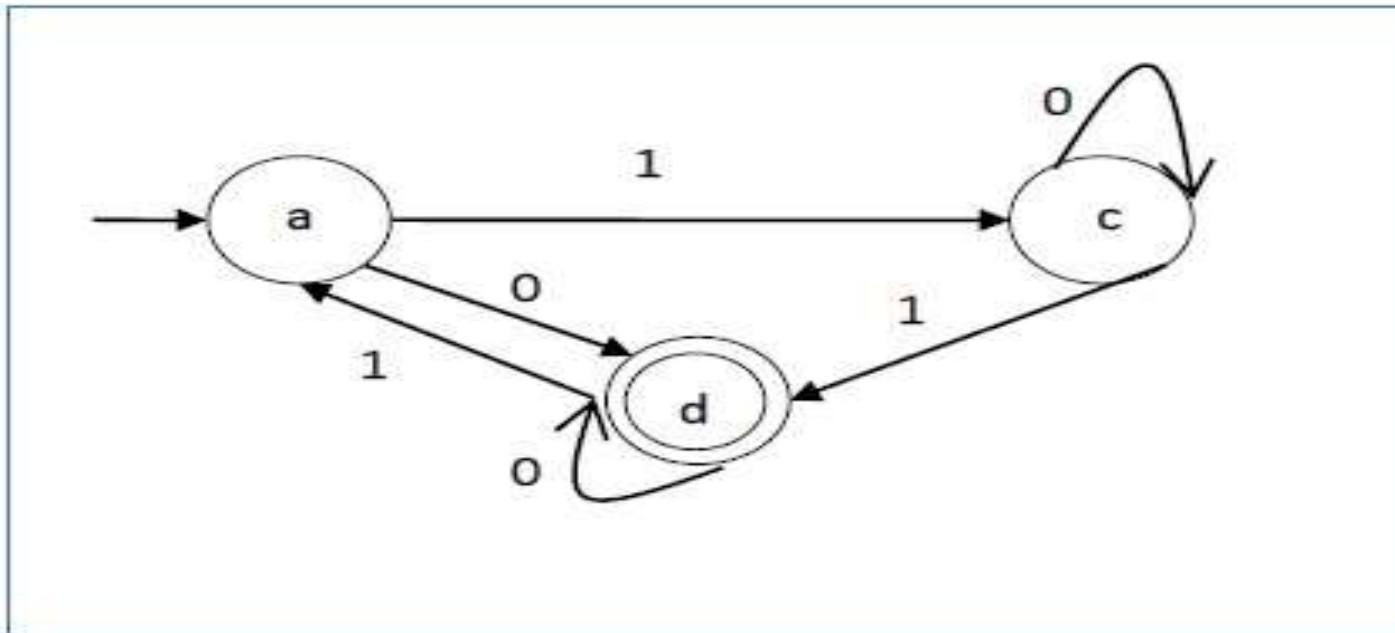
A string is accepted by a DFA/NDFA if the DFA/NDFA starting at the initial state ends in an accepting state (any of the final states) after reading the string wholly.

A string S is accepted by a DFA/NDFA $(Q, \Sigma, \delta, q_0, F)$, if
 $\delta^*(q_0, S) \in F$

- The language **L** accepted by DFA/NDFA is
 $\{S \mid S \in \Sigma^* \text{ and } \delta^*(q_0, S) \in F\}$
- A string S' is not accepted by a DFA/NDFA $(Q, \Sigma, \delta, q_0, F)$, iff
 $\delta^*(q_0, S') \notin F$

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

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



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

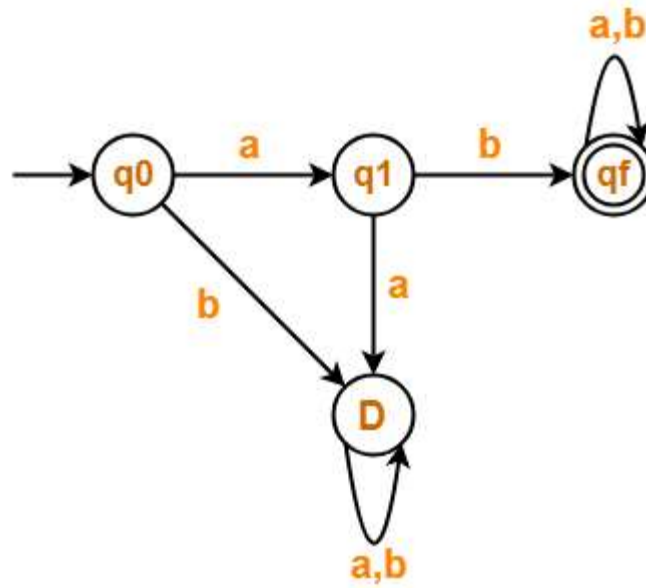
Strings not accepted by the above DFA: {1, 011, 111,}

Q3.construct DFA for the following strings-

ab

aba

abab



DFA

Converting NFA to DFA-

The following steps are followed to convert a given NFA to a DFA

Step-01:

Let Q' be a new set of states of the DFA. Q' is null in the starting.
Let T' be a new transition table of the DFA.

Step-02:

Add start state of the NFA to Q' .

Add transitions of the start state to the transition table T' .

If start state makes transition to multiple states for some input alphabet, then treat those multiple states as a single state in the DFA.

* In NFA, if the transition of start state over some input alphabet is null,
then perform the transition of start state over that input alphabet to a dead state in the DFA.

Step-03:

If any new state is present in the transition table T' ,

Add the new state in Q' .

Add transitions of that state in the transition table T' .

Step-04:

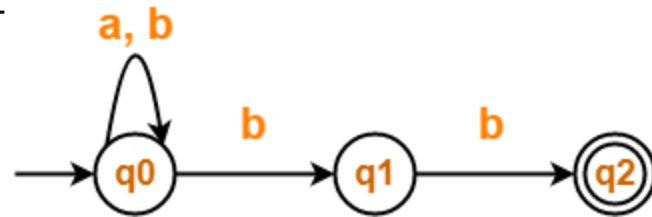
Keep repeating Step-03 until no new state is present in the transition table T' .

Finally, the transition table T' so obtained is the complete transition table of the required DFA.

PRACTICE PROBLEMS BASED ON CONVERTING NFA TO DFA-

Problem-01:

Convert the following Non-Deterministic Finite Automata (NFA) to Deterministic Finite Automata (DFA)-



Solution-

Transition table for the given Non-Deterministic Finite Automata (NFA) is-

State / Alphabet	a	b
→q0	q0	q0, q1
q1	–	*q2
*q2	–	–

Step-01:

Let Q' be a new set of states of the Deterministic Finite Automata (DFA).

Let T' be a new transition table of the DFA.

Step-02:

Add transitions of start state q_0 to the transition table T' .

State / Alphabet	a	b
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$

Step-03:

New state present in state Q' is $\{q_0, q_1\}$.

Add transitions for set of states $\{q_0, q_1\}$ to the transition table T' .

State / Alphabet	a	b
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1\}$	q_0	$\{q_0, q_1, q_2\}$

Step-04:

New state present in state Q' is $\{q_0, q_1, q_2\}$.

Add transitions for set of states $\{q_0, q_1, q_2\}$ to the transition table T' .

State / Alphabet	a	b
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1\}$	q_0	$\{q_0, q_1, q_2\}$
$\{q_0, q_1, q_2\}$	q_0	$\{q_0, q_1, q_2\}$

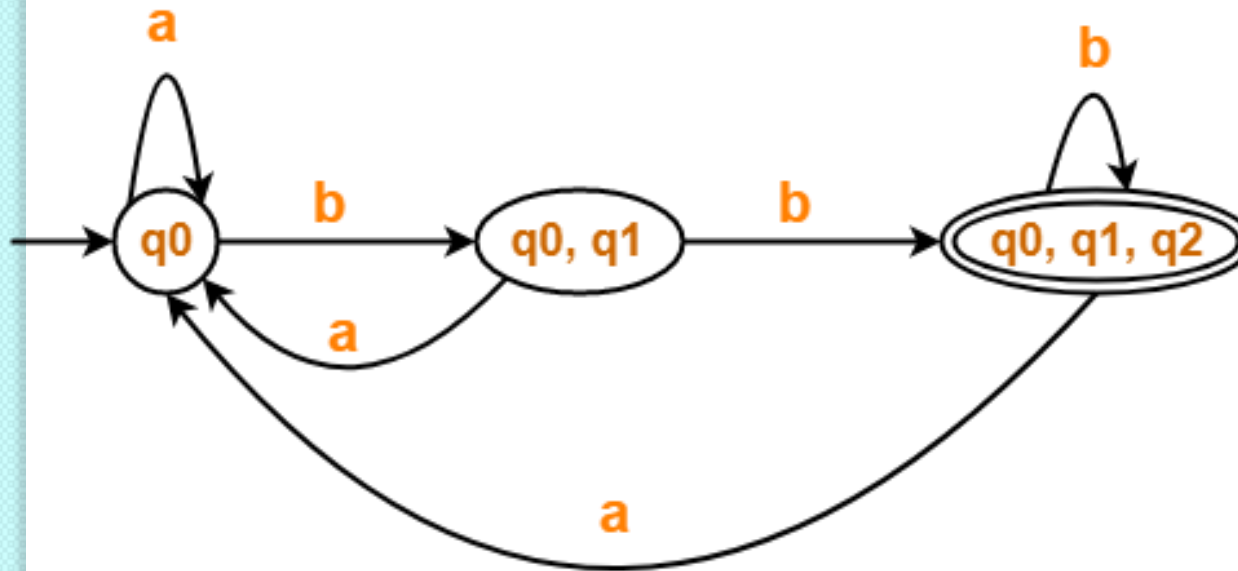
Step-05:

Since no new states are left to be added in the transition table T' , so we stop. States containing q_2 as its component are treated as final states of the DFA.

Finally, Transition table for Deterministic Finite Automata (DFA) is-

State / Alphabet	a	b
$\rightarrow q_0$	q_0	$\{q_0, q_1\}$
$\{q_0, q_1\}$	q_0	$\ast\{q_0, q_1, q_2\}$
$\ast\{q_0, q_1, q_2\}$	q_0	$\ast\{q_0, q_1, q_2\}$

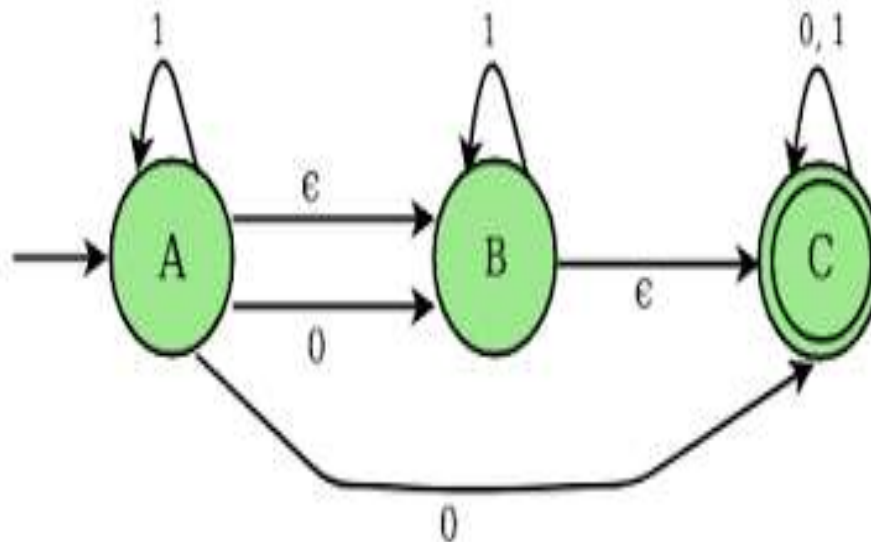
Now, Deterministic Finite Automata (DFA) may be drawn as-



Deterministic Finite Automata (DFA)

Finite Automaton with ϵ -moves

Nondeterministic finite automaton with ϵ -moves (NFA- ϵ) is a **further generalization to NFA**. This automaton replaces the transition function with the one that allows the empty string ϵ as a possible input. The transitions without consuming an input symbol are called ϵ -transitions.



Conversion from NFA with ϵ to DFA

Non-deterministic finite automata(NFA) is a finite automata where for some cases when a specific input is given to the current state, the machine goes to multiple states or more than 1 states. It can contain ϵ move. It can be represented as $M = \{ Q, \Sigma, \delta, q_0, F \}$.

Where

Q: finite set of states

Σ : finite set of the input symbol

q_0 : initial state

F: **final** state

δ : Transition function

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

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

Steps for converting NFA with ϵ to DFA:

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

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

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

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

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

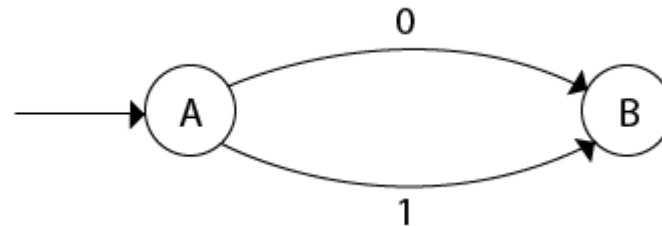
Now, let ε -closure $\{q_0\} = \{q_0, q_1, q_2\}$ be state A.

Hence

$$\begin{aligned}\delta'(A, 0) &= \varepsilon\text{-closure } \{\delta((q_0, q_1, q_2), 0) \} \\ &= \varepsilon\text{-closure } \{\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0) \} \\ &= \varepsilon\text{-closure } \{q_3\} \\ &= \{q_3\} \quad \text{call it as state B.}\end{aligned}$$

$$\begin{aligned}\delta'(A, 1) &= \varepsilon\text{-closure } \{\delta((q_0, q_1, q_2), 1) \} \\ &= \varepsilon\text{-closure } \{\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1) \} \\ &= \varepsilon\text{-closure } \{q_3\} \\ &= \{q_3\} = B.\end{aligned}$$

The partial DFA will be



Now,

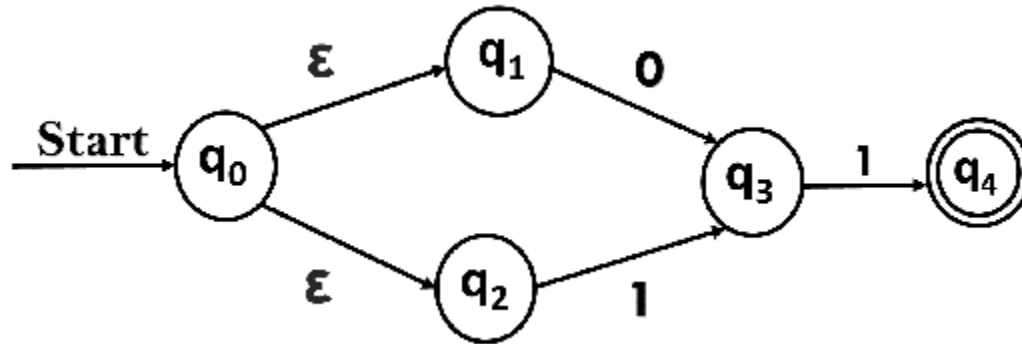
$$\begin{aligned}\delta'(B, 0) &= \varepsilon\text{-closure } \{\delta(q3, 0) \} \\ &= \phi \ \delta'(B, 1) \\ &= \varepsilon\text{-closure } \{\delta(q3, 1) \} \\ &= \varepsilon\text{-closure } \{q4\} \\ &= \{q4\} \quad \textbf{i.e. state C}\end{aligned}$$

For state C:

$$\begin{aligned}\delta'(C, 0) &= \varepsilon\text{-closure } \{\delta(q4, 0) \} \\ &= \varphi \\ \delta'(C, 1) &= \varepsilon\text{-closure } \{\delta(q4, 1) \} \\ &= \varphi\end{aligned}$$

Example 1

Convert the NFA with ϵ into its equivalent DFA.



Solution:

Let us obtain ϵ -closure of each state.

ϵ -closure $\{q_0\} = \{q_0, q_1, q_2\}$

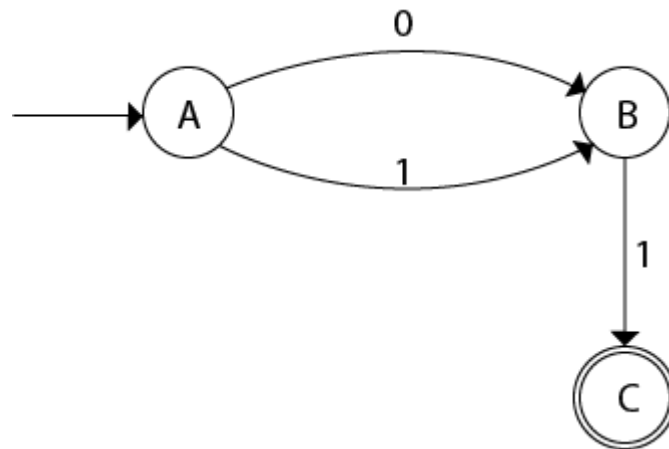
ϵ -closure $\{q_1\} = \{q_1\}$

ϵ -closure $\{q_2\} = \{q_2\}$

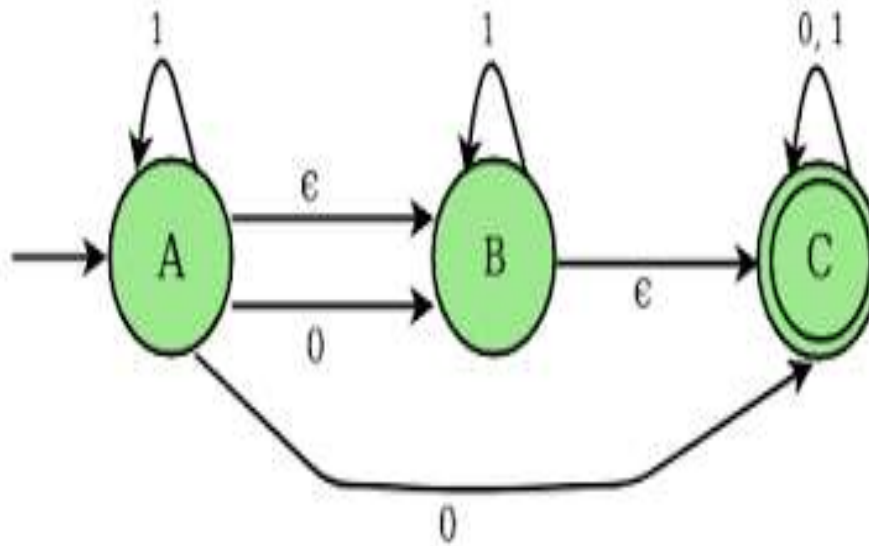
ϵ -closure $\{q_3\} = \{q_3\}$

ϵ -closure $\{q_4\} = \{q_4\}$

The DFA will be,



Example 2: Consider the following figure of epsilon NFA convert it to DFA :



Transition state table for the above NFA

STATES	0	1	epsilon
A	B, C	A	B
B	–	B	C
C	C	C	–

Epsilon (ϵ) – closure : Epsilon closure for a given state X is a set of states which can be reached from the states X with only (null) or ϵ moves including the state X itself. In other words, ϵ -closure for a state can be obtained by union operation of the ϵ -closure of the states which can be reached from X with a single ϵ move in recursive manner.

For the above example ϵ closure are as follows :

ϵ closure(A) : {A, B, C}

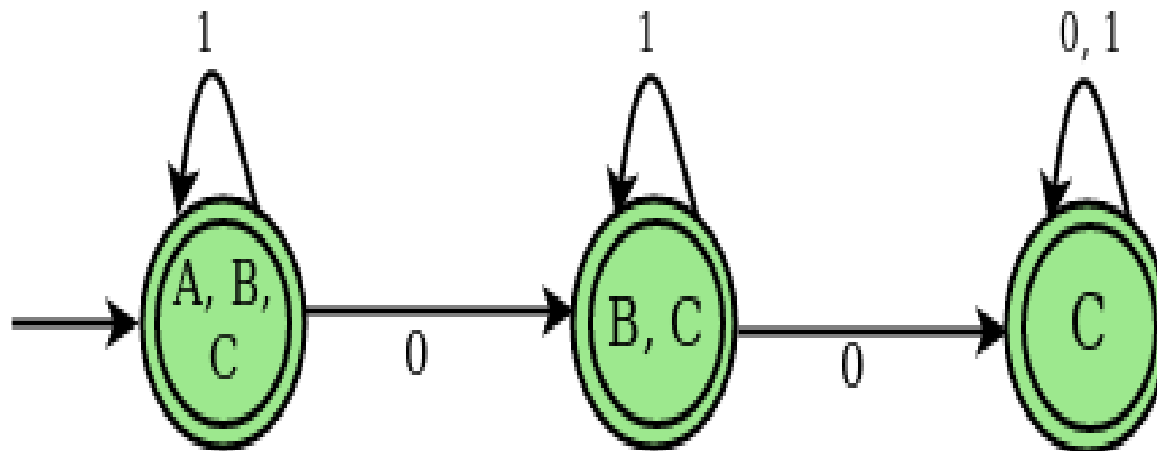
ϵ closure(B) : {B, C}

ϵ closure(C) : {C}

Transition State Table for DFA corresponding to above NFA

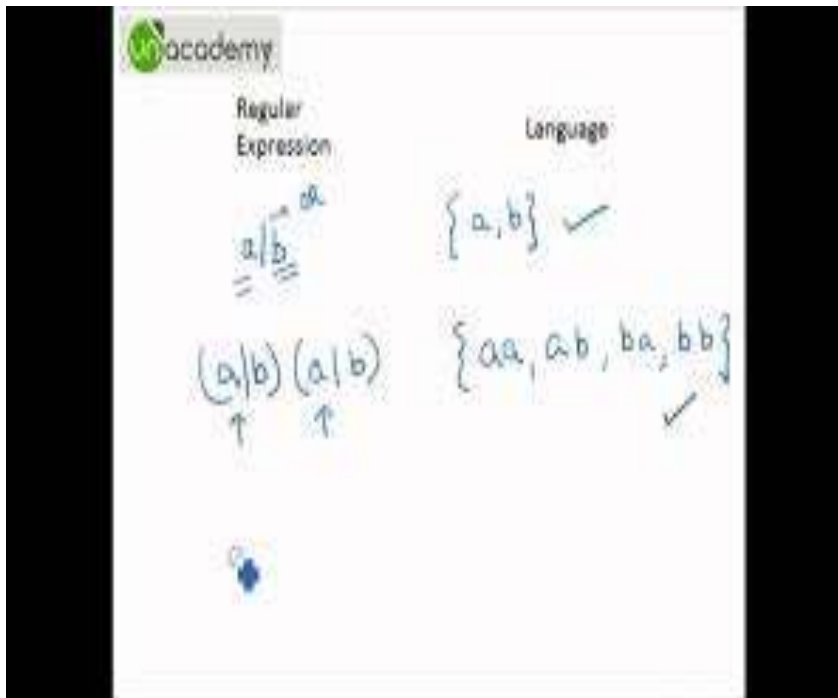
STATES	0	1
A, B, C	B, C	A, B, C
B, C	C	B, C
C	C	C

DFA STATE DIAGRAM



Regular Languages- Regular Expression

Regular Expressions are **an algebraic way to describe languages**. Regular Expressions describe exactly the regular languages. If E is a regular expression, then $L(E)$ is the regular language it defines.



The image shows a whiteboard with handwritten notes. At the top left is a logo for 'academy'. The whiteboard is divided into two columns: 'Regular Expression' and 'Language'. The first row shows the regular expression $a|b$ with an arrow pointing to the a and another arrow pointing to the b , and the language $\{a, b\}$ with a checkmark. The second row shows the regular expression $(a|b)(a|b)$ with arrows pointing to each $(a|b)$ group, and the language $\{aa, ab, ba, bb\}$ with a checkmark.

Regular Expression	Language
$a b$	$\{a, b\}$ ✓
$(a b)(a b)$	$\{aa, ab, ba, bb\}$ ✓

1. Regular language

The set of regular languages over an alphabet Σ is defined recursively as below. Any language belonging to this set is a **regular language** over Σ .

Definition of Set of Regular Languages :

Basis Clause: \emptyset , $\{\Lambda\}$ and $\{a\}$ for any symbol $a \in \Sigma$ are regular languages.

Inductive Clause: If L_r and L_s are regular languages, then $L_r \cup L_s$, $L_r L_s$ and L_r^* are regular languages.

Extremal Clause: Nothing is a regular language unless it is obtained from the above two clauses.

For example, let $\Sigma = \{a, b\}$. Then since $\{a\}$ and $\{b\}$ are regular languages, $\{a, b\} (= \{a\} \cup \{b\})$ and $\{ab\} (= \{a\}\{b\})$ are regular languages. Also since $\{a\}$ is regular, $\{a\}^*$ is a regular language which is the set of strings consisting of a's such as Λ , a, aa, aaa, aaaa etc. Note also that Σ^* , which is the set of strings consisting of a's and b's, is a regular language because $\{a, b\}$ is regular.

2. Regular expression

Regular expressions are used to denote regular languages. They can represent regular languages and operations on them succinctly.

The set of regular expressions over an alphabet Σ is defined recursively as below. Any element of that set is a **regular expression**.

Basis Clause: \emptyset , Λ and a are regular expressions corresponding to languages \emptyset , $\{\Lambda\}$ and $\{a\}$, respectively, where a is an element of Σ .

Inductive Clause: If r and s are regular expressions corresponding to languages L_r and L_s , then $(r + s)$, (rs) and (r^*) are regular expressions corresponding to languages $L_r \cup L_s$, $L_r L_s$ and L_r^* , respectively.

expression which is one of the ways to describe regular languages.

Operations on Languages

Remember: A *language* is a set of strings

Union: $L \cup M = \{w : w \in L \text{ or } w \in M\}$

Concatenation: $L.M = \{w : w = xy, x \in L, y \in M\}$

Powers: $L^0 = \{\epsilon\}$, $L^1 = L$, $L^{k+1} = L.L^k$

Kleene Closure: $L^* = \bigcup_{i=0}^{\infty} L^i$

Regular Expressions

- Regular Expressions are an algebraic way to describe languages.
- Regular Expressions describe exactly the regular languages.
- If E is a regular expression, then $L(E)$ is the regular language it defines.
- A regular expression is built up of simpler regular expressions (using defining rules)
- For each regular expression E , we can create a DFA A such that $L(E) = L(A)$.
- For each a DFA A , we can create a regular expression E such that $L(A) = L(E)$

Regular Expressions - Definition

Regular expressions over alphabet Σ

	<u>Reg. Expr. E</u>	<u>Language it denotes L(E)</u>
Basis 1:	Φ	$\{\}$
Basis 2:	ϵ	$\{\epsilon\}$
Basis 3:	$a \in \Sigma$	$\{a\}$

Note:

$\{a\}$ is the language containing one string, and that string is of length 1.

Regular Expressions - Definition

Induction 1 – or : If E_1 and E_2 are regular expressions, then $E_1 + E_2$ is a regular expression, and $L(E_1 + E_2) = L(E_1) \cup L(E_2)$.

Induction 2 – concatenation: If E_1 and E_2 are regular expressions, then $E_1 E_2$ is a regular expression, and $L(E_1 E_2) = L(E_1) L(E_2)$ where $L(E_1) L(E_2)$ is the set of strings wx such that w is in $L(E_1)$ and x is in $L(E_2)$.

Induction 3 – Kleene Closure: If E is a regular expression, then E^* is a regular expression, and $L(E^*) = (L(E))^*$.

Induction 4 – Pranteheses: If E is a regular expression, then (E) is a regular expression, and $L((E)) = L(E)$.

Regular Expressions - Parentheses

- Parentheses may be used wherever needed to influence the grouping of operators.
- We may remove parentheses by using precedence and associativity rules.

<u>Operator</u>	<u>Precedence</u>	<u>Associativity</u>
*	highest	
concatenation	next	left associative
+	lowest	left associative

ab^*+c means **$(a((b)^*))+(c)$**

Regular Expressions - Examples

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

- $L(\mathbf{01}) = \{01\}$. $L(\mathbf{01}) = L(\mathbf{0}) L(\mathbf{1}) = \{0\} \{1\} = \{01\}$
- $L(\mathbf{01+0}) = \{01, 0\}$. $L(\mathbf{01+0}) = L(\mathbf{01}) \cup L(\mathbf{0}) = (L(\mathbf{0}) L(\mathbf{1})) \cup L(\mathbf{0})$
 $= (\{0\} \{1\}) \cup \{0\} = \{01\} \cup \{0\} = \{01, 0\}$
- $L(\mathbf{0(1+0)}) = \{01, 00\}$.
 - Note order of precedence of operators.
- $L(\mathbf{0^*}) = \{\epsilon, 0, 00, 000, \dots\}$.
- $L(\mathbf{(0+10)^*(\epsilon+1)})$ = all strings of 0's and 1's without two consecutive 1's.
- $L(\mathbf{(0+1)(0+1)}) = \{00, 01, 10, 11\}$
- $L(\mathbf{(0+1)^*})$ = all strings with 0 and 1, including the empty string

Regular Expressions - Examples

All strings of 0's and 1's starting with 0 and ending with 1

$0(0+1)^*1$

All strings of 0's and 1's with even number of 0's

$1^*(01^*01^*)^*$

All strings of 0's and 1's with at least two consecutive 0's

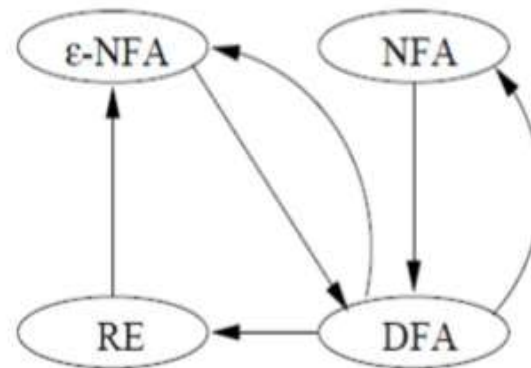
$(0+1)^*00(0+1)^*$

All strings of 0's and 1's without two consecutive 0's

$((1+01)^*(\epsilon+0))$

Equivalence of FA's and Regular Expressions

- We have already shown that DFA's, NFA's, and ϵ -NFA's all are equivalent.
- To show FA's equivalent to regular expressions we need to establish that
 1. For every DFA A we can construct a regular expression R, s.t. $L(R) = L(A)$.
 2. For every regular expression R there is a ϵ -NFA A (a DFA A), s.t. $L(A) = L(R)$.





Equivalence of NFA and DFA

Why NFA is equivalent to DFA?

An **NFA** can have zero, one or more than one move from a **given state on a given input** symbol. An NFA can also have NULL moves (moves without input symbol). On the other hand, DFA has one and only one move from a given state on a given input symbol.