

# Formal Languages and Automata Theory

**23CSPC304**

# Syllabus

## **Module 1: Introduction to formal languages and Finite Automata**

- Introduction to formal languages: Need for Automata Theory, The central concepts of Automata theory- Alphabet, String, Language, A machine-based hierarchy of language class Finite Automata: Deterministic Finite Automata, Nondeterministic Finite Automata, Finite Automata with Epsilon Transition, Equivalence and Minimization of Automata
- **Text book1: Chapter 1.1, 1.5, Chapter 2.2, 2.3.1 to 2.3.5, 2.5, 4.4**
- **Text book2: Chapter 3.3**

## **Module 2: Regular Expression, Properties of Regular Languages**

- Regular Expression, Properties of Regular Languages: Regular Expressions, Finite Automata and Regular Expressions, Proving Languages Not to Be Regular, Closure Properties of Regular Language
- **Text book1: Chapter 3.1, 3.2.2, 3.2.3, 4.1, 4.2**

## **Module 3: Context-Free Grammars and Languages**

- Context-Free Grammars and Languages: Context –Free Grammars, Parse Trees, Ambiguity in Grammars and Languages, Closure properties of Context- Free Languages
- **Text book1: Chapter 5.1, 5.2, 5.4, 7.3**

## **Module 4: Properties of Context Free Languages and Pushdown Automata**

- Properties of Context Free Languages: Normal forms for Context- Free Grammar Pushdown Automata: Definition of the Pushdown automata, The languages of a PDA, Equivalence of PDA's and CFG's: From Grammars to Pushdown Automata, Deterministic Pushdown Automata
- **Text book1: Chapter 7.1, 6.1 to 6.3.1, 6.4**

## **Module 5: Introduction to Turing Machine and Undecidability**

- Introduction to Turing Machine: Problems That Computers Cannot Solve, The Turing Machine, Programming Techniques for Turing Machines, Extensions to the Basic Turing Machines Undecidability: A Language That Is Not Recursively Enumerable, An Undecidable Problem That is RE, Post Correspondence Problem
- **Text book1: Chapter 8.1 to 8.4, Chapter 9.1, 9.2, 9.4.1**

# Text Books

1. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman: Introduction to Automata Theory, Languages and Computation, 3rd Edition, Pearson Education, 2007.
2. Elaine Rich: Automata, Computability and Complexity, Theory and Applications, Pearsons Education, 2018.

# Reference Books

1. K.L.P. Mishra: Theory of Computer Science, Automata, Languages, and Computation, 3rd Edition, PHI Learning, 2009.
2. Raymond Greenlaw, H. James Hoover: Fundamentals of the Theory of Computation, Principles and Practice, Elsevier, 1998.
3. John C Martin: Introduction to Languages and Automata Theory, 3rd Edition, Tata McGraw-Hill, 2007.
4. Thomas A. Sudkamp: An Introduction to the Theory of Computer Science, Languages and Machines, 3rd Edition, Pearson Education, 2006.

# Course Outcomes

At the end of the course, the student will be able to

1. Apply automata concepts to construct and optimize finite automata using equivalence and minimization techniques.
2. Apply regular expressions and finite automata to recognize languages, prove non-regularity, and utilize closure properties.
3. Apply context-free grammars to generate languages, construct parse trees, resolve ambiguity, and explore closure properties.
4. Apply context-free grammar transformations, design pushdown automata, and establish PDA-CFG equivalence for language recognition.
5. Apply Turing machines to develop computation models and solve undecidable problems.

# Module 1: Introduction to formal languages and Finite Automata

**Text book1: Chapter 1.1, 1.5, Chapter 2.2, 2.3.1 to 2.3.5, 2.5, 4.4**

**Text book2: Chapter 3.3**

## What is automata

- The term "Automata" is derived from the Greek word "αὐτόματα" which means "self-acting".
- Automata are abstract models of machines that perform computations of an input by moving through a series of states.

# Finite Automata(FA)

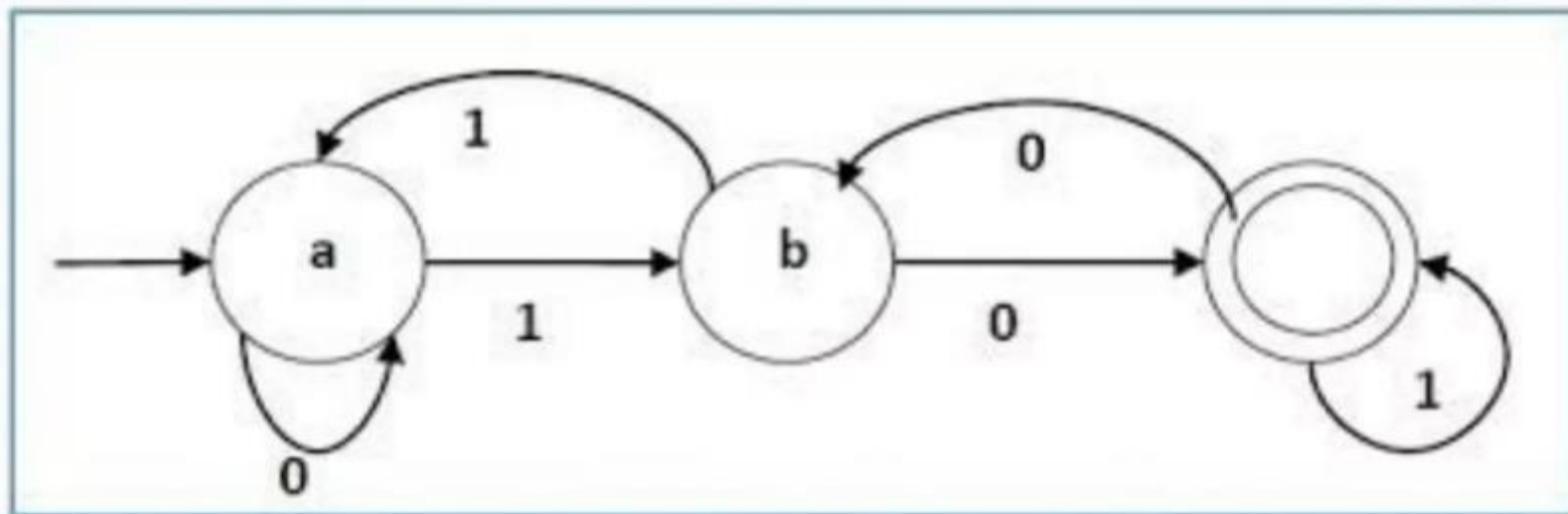
- An automaton with a finite number of states is called a **Finite Automaton** (FA) or **Finite State Machine** (FSM).

- Formal definition of a Finite Automaton

An automaton can be represented by a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where –

- $Q$  is a finite set of states.
- $\Sigma$  is a finite set of symbols, called the **alphabet** of the automaton.
- $\delta$  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  $Q$  ( $F \subseteq Q$ ).

- **Example:**



- The Circle 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.

$$Q = \{a, b, c\},$$

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

$$q_0 = \{a\},$$

$$F = \{c\}$$

# Structural Representation

- FA is represented in 2 ways

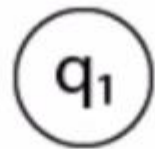
A. Transition Diagram

B. Transition Table

States/ Input	0	1
→A	A	B
B	C	A
*C	B	C

# Transition Diagram

- A transition diagram or state transition diagram is a directed graph which can be constructed as follows:
  - There is a node for each state in  $Q$ , which is represented by the circle.
  - There is a directed edge from node  $q$  to node  $p$  labeled  $a$  if  $\delta(q, a) = p$ .
  - In the start state, there is an arrow with no source.
  - Accepting states or final states are indicating by a double circle.



State



Transition from one  
state to another



Start State



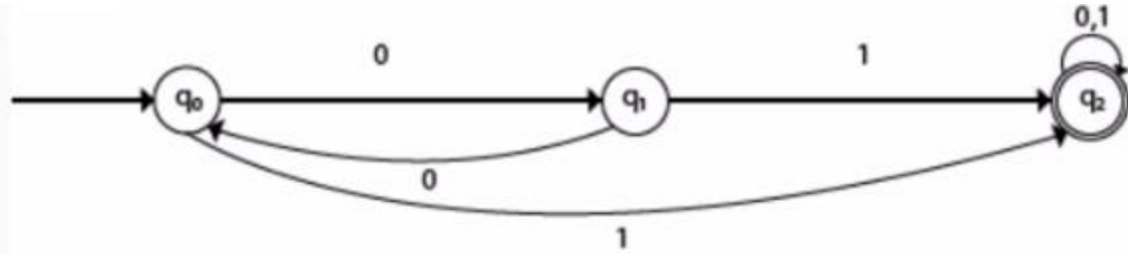
Final State

# Transition Table

- The transition table is basically a tabular representation of the transition function. It takes two arguments (a state and a symbol) and returns a state (the "next state").

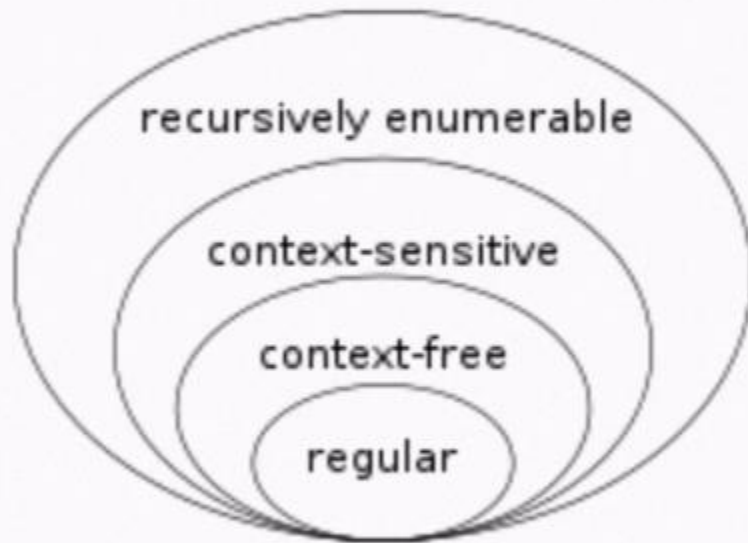
A transition table is represented by the following things:

- Columns correspond to input symbols.
- Rows correspond to states.
- Entries correspond to the next state.
- The start state is denoted by an arrow with no source.
- The accept state is denoted by a star.



Input/states	0	1
→Q0	Q1	Q2
Q1	Q0	Q2
*Q2	Q2	Q2

## ● Chomsky Normal Form



Grammar	Languages	Automaton
Type-0	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine
Type-2	Context-free	pushdown automaton
Type-3	Regular	Finite state automaton

# Alphabets, Strings, Languages, Problems

- Alphabet: An alphabet is a finite collection of symbols.
- Example:
  - $\Sigma = \{a, b\}$  be an alphabet
  - $\Sigma = \{0,1\}$  be an alphabet
  - $\Sigma = \{a, b, \dots, z\}$  be an alphabets (set of lower case letters)

- String: A string over an alphabet  $\Sigma$  is a finite sequence of symbols of  $\Sigma$ .
- Example:
  - Let  $\Sigma = \{a, b\}$  be an alphabet; then  $\{a, b, aa, ab, bba, baaba, \dots\}$  are some examples of strings over  $\Sigma$ . A infinite number of strings can be generated.
  - Let  $\Sigma = \{0,1\}$  be an alphabet; then  $\{0,1,01,0011,0101,100000,\dots\}$

- Language: It is collection of appropriate strings.
- Examples:
  - L= The set of all strings over  $\{0, 1\}$  that start with 0  
 $\Sigma = \{0, 00, 010, 011, 0111, 0101, \dots\}$
  - The set of all strings over  $\{a, b, c\}$  having ac as a substring  
 $\Sigma = \{ac, abaca, acba, bcac, abac, \dots\}$

Since languages are sets, we can apply various well known set operations such as union, concatenation, Kleene star or Kleene closure, complement, difference, intersection on languages

## Kleene Star/Kleene Closure

- The Kleene star,  $\Sigma^*$ , is a unary operator on a set of symbols or strings  $\Sigma$ .
- For example, if  $\Sigma = \{0, 1\}$ , then  $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$
- $\Sigma^0 = \{\epsilon\}$ . The length of string
- $\Sigma^1 = \{0, 1\}$
- $\Sigma^2 = \{00, 11, 10, 01\}$
- $\Sigma^3 = \{000, 001, 011, 010, 111, 110, 101, 100\}$
- The Kleene closure of a language  $L$  is denoted by  $L^*$  is defined as

- The Kleene star of the language  $L = \{0, 1\}$  over the alphabet  $\Sigma = \{0, 1\}$  is
- $L^* = L^0 \cup L^1 \cup L^2 \cup \dots$
- $= \{\varepsilon\} \cup \{0, 1\} \cup \{00, 01, 10, 11\} \cup \dots$
- $= \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\}$
- $=$  the set of all strings over  $\Sigma$ .

## Kleene Plus

- The positive closure of a language  $L$  is denoted by  $L^+$  is defined as over the alphabet  $\Sigma = \{0, 1\}$  is
- $L^+ = L^1 \cup L^2 \cup \dots$
- $= \{0, 1\} \cup \{00, 01, 10, 11\} \cup \dots$
- $= \{0, 1, 00, 01, 10, 11, \dots\}$
- $=$  the set of all strings over  $\Sigma$ .

# Types of FA

- 2 types of FA. They are
  - Deterministic Finite Automaton(DFA) or Deterministic Finite Machine
  - Non-Deterministic Finite Automaton(NFA) or Non-Deterministic Finite Machine
-

# DFA

## Formal Definition of a DFA

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

- $Q$  is a finite set of states.
- $\Sigma$  is a finite set of symbols called the alphabet.
- $\delta$  is the transition function where  $\delta: Q \times \Sigma \rightarrow Q$
- $q_0$  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 **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.

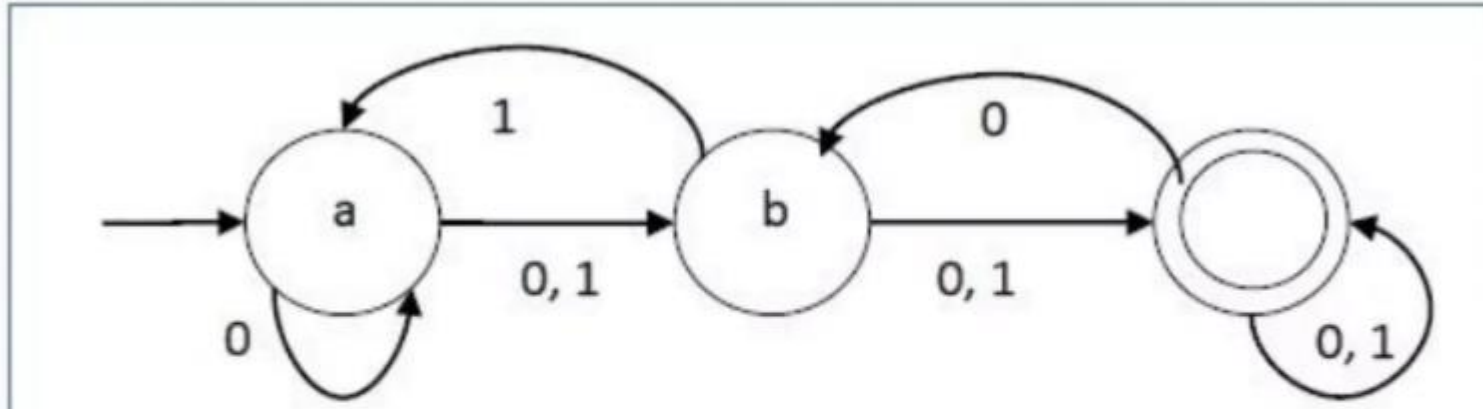
# NFA

An NFA can be represented by a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$  where –

- $Q$  is a finite set of states.
- $\Sigma$  is a finite set of symbols called the alphabets.
- $\delta$  is the transition function where  $\delta: Q \times \Sigma \rightarrow 2Q$   
(Here the power set of  $Q$  ( $2Q$ ) 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  $Q$  ( $F \subseteq Q$ ).

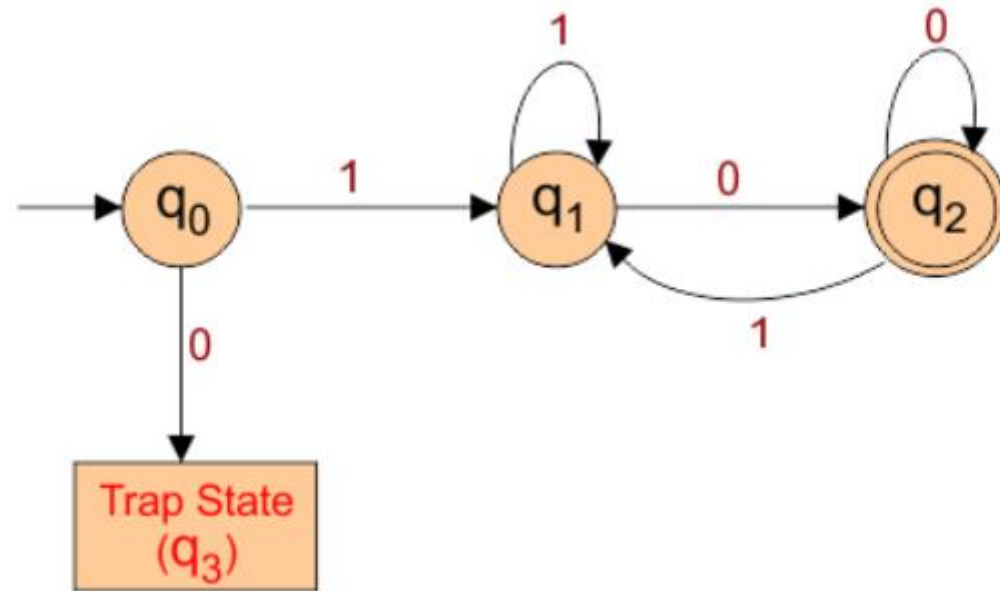
## Graphical Representation of a NFA

- 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.

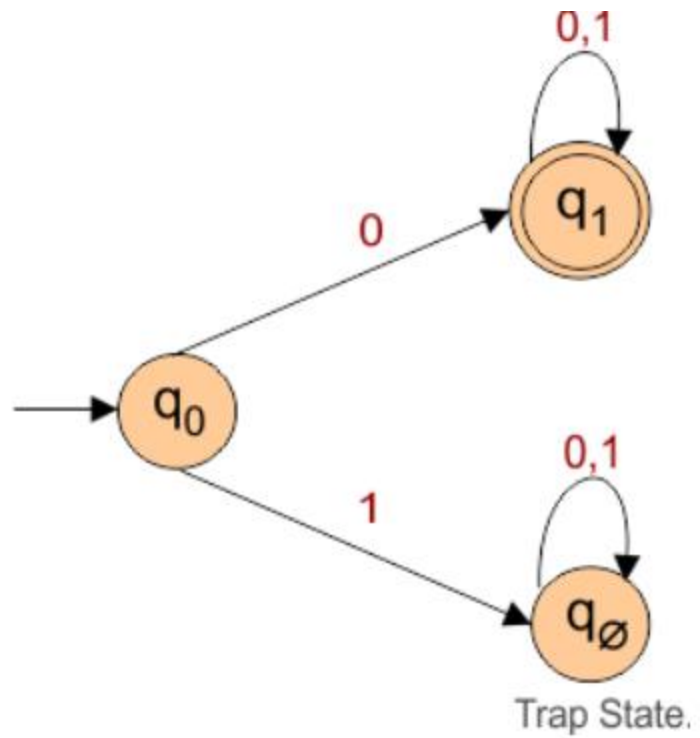


Present State	Next State for Input 0	Next State for Input 1
a	a, b	b
b	c	a, c
c	b, c	c

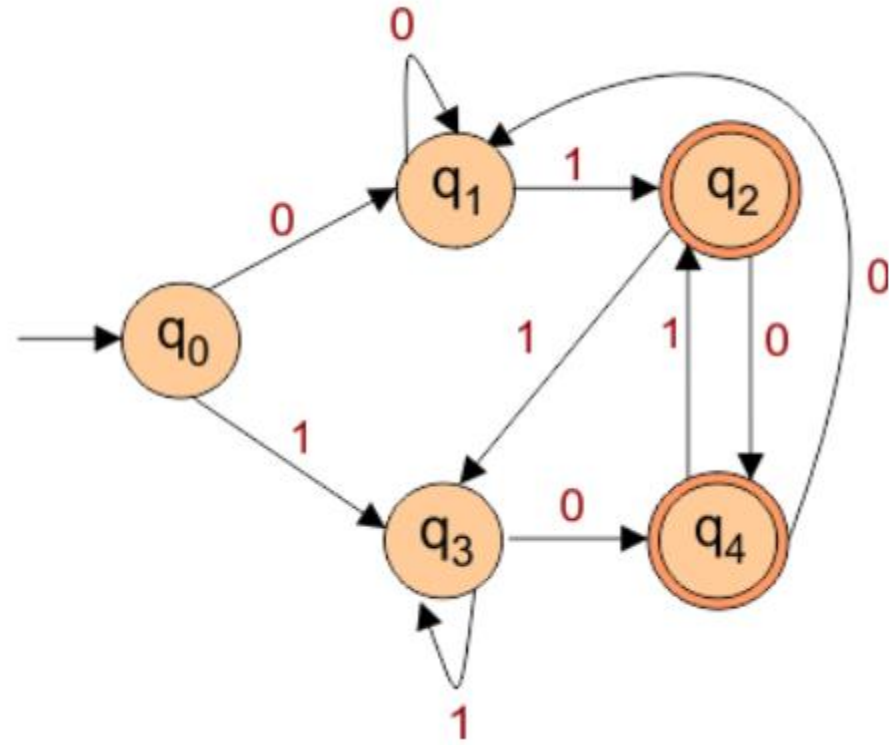
Construct a DFA with sigma  $\Sigma = \{0, 1\}$ , accepts those string which **starts with one and ends with 0**.



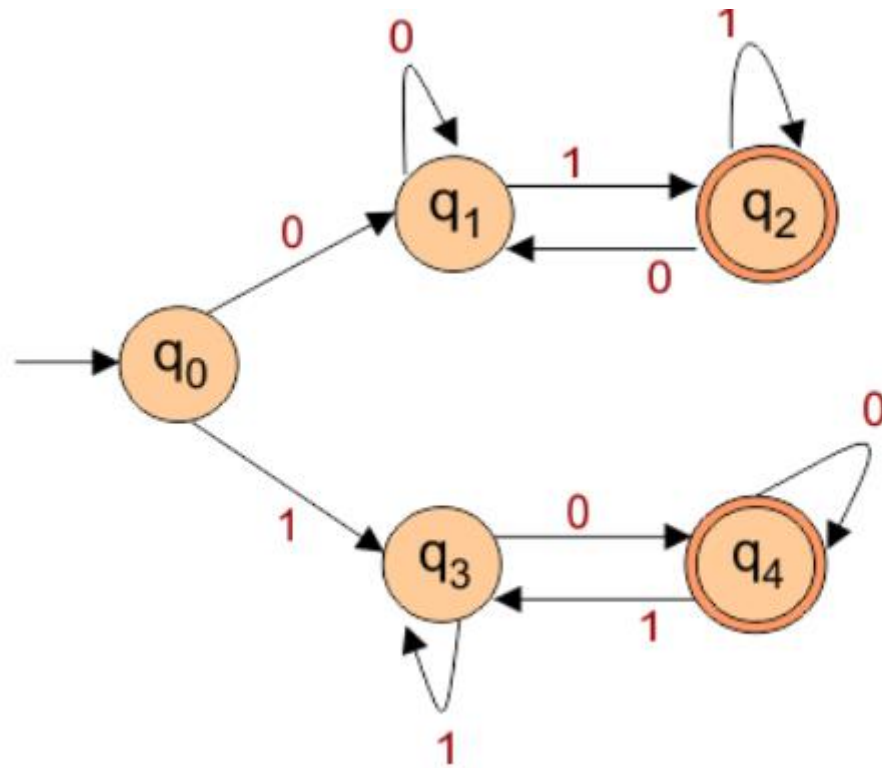
Construct a DFA that accepts all the strings over the alphabets  $\Sigma \{0,1\}$  that **start with “0”**.



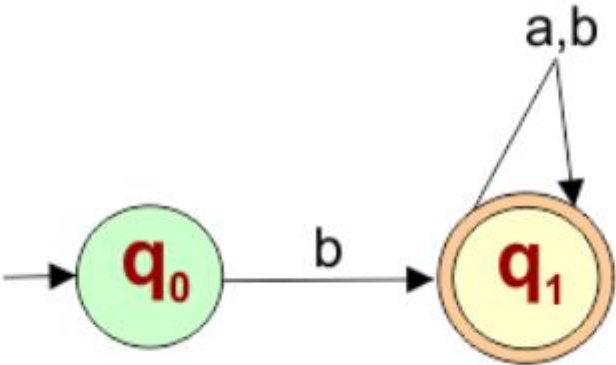
Construct a DFA with sigma  $\Sigma = \{0, 1\}$  for the language accepting **strings ending in either '01' or '10'**.



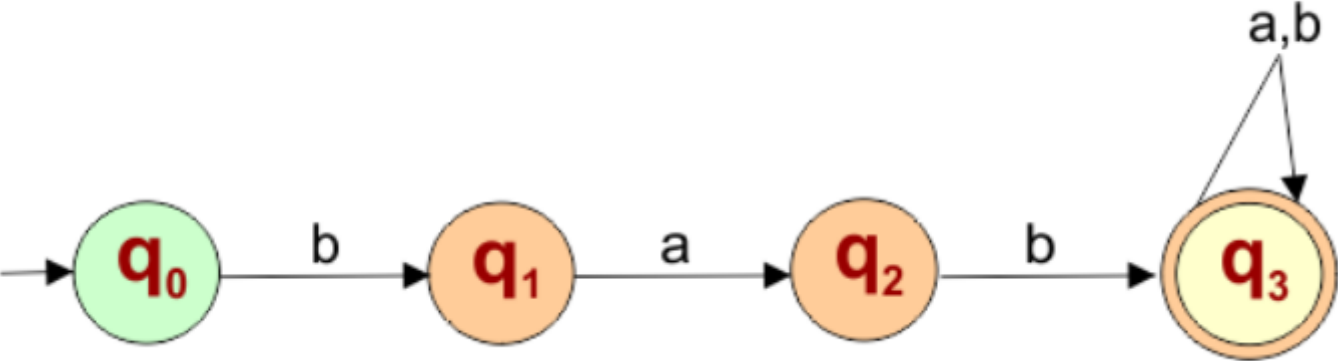
Design a DFA with sigma  $\Sigma = \{0, 1\}$  for the language accepting strings **starting and ending with different characters**.



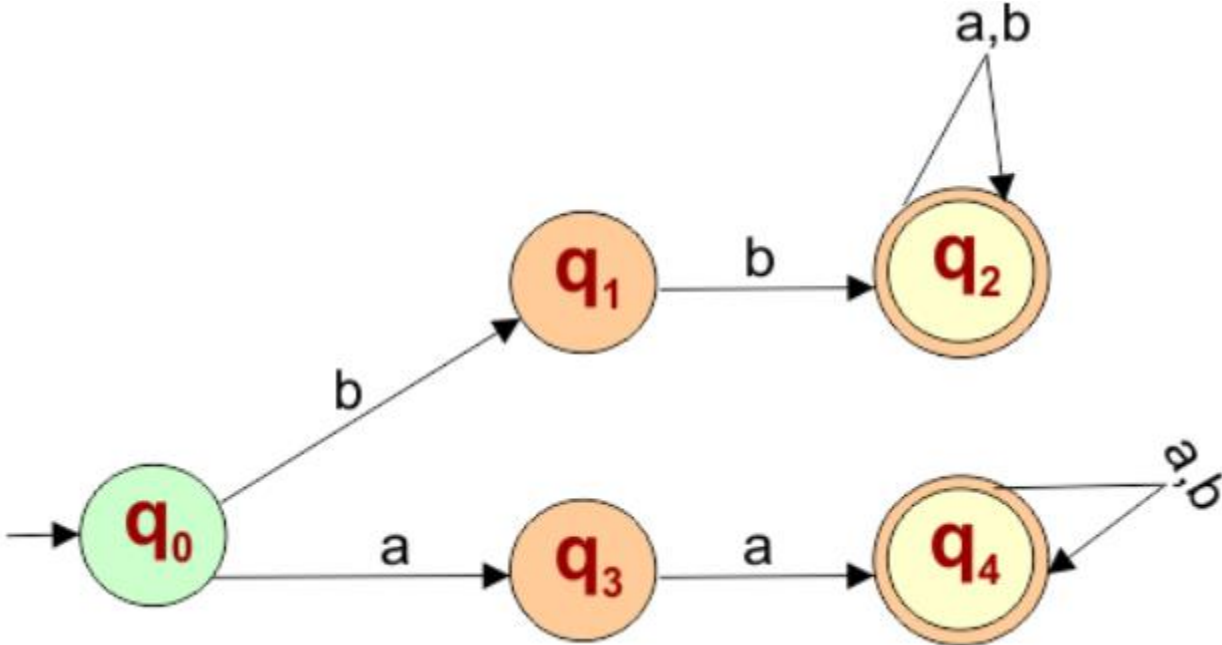
Design an NFA over the alphabet  $\Sigma = \{a, b\}$  that accepts only the strings that start with "b"



Design an NFA over the alphabet  $\Sigma = \{a, b\}$  that accepts only the strings that start with "bab"

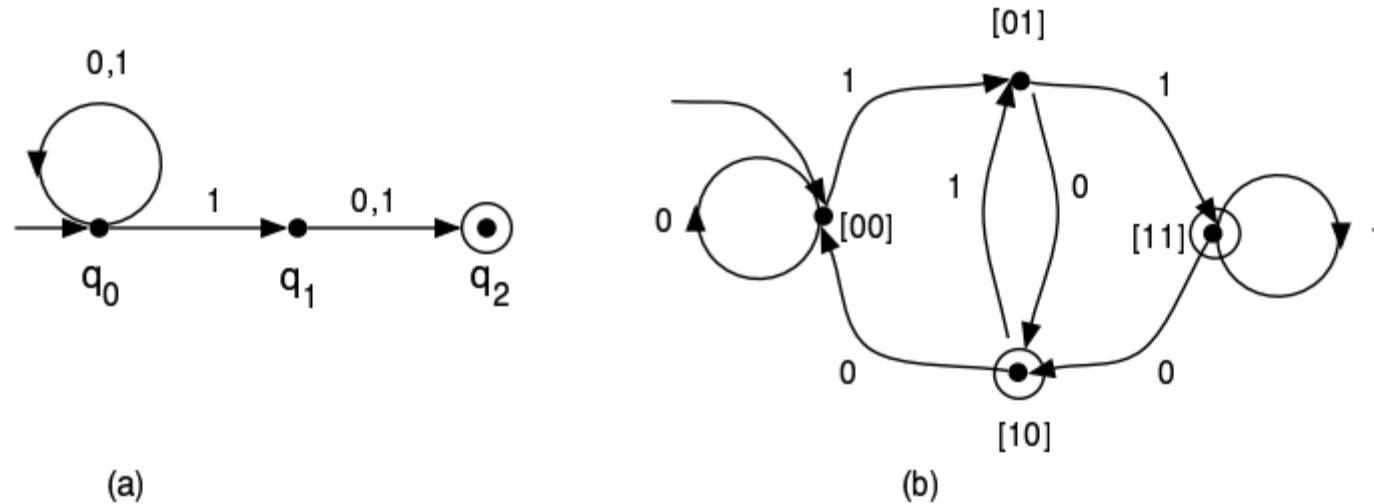


Design an NFA over the alphabet  $\Sigma = \{a, b\}$  that accepts only the strings that start with "aa" or "bb"



# NFA to DFA Subset construction method

Consider the set  $A = \{x \in \{0,1\}^* \mid \text{2nd symbol from right is } 1\}$ .



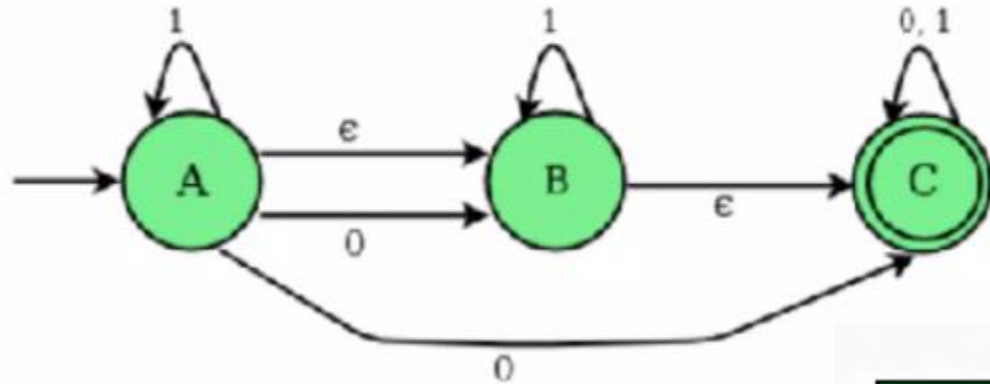
A 3-state non-deterministic machine (NFA) with the set  $A$  of accepted strings is given in the figure above (a), with the states labelled  $q_0, q_1, q_2$ . The transition function for an NFA returns sets of possible states rather than particular states. Each such subset can be regarded as a particular state of an associated DFA. The “subset construction” thus associates to the above NFA a deterministic machine (DFA) whose states are the subsets  $\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}$  of the set  $\{q_0, q_1, q_2\}$  of states of the NFA.

	0	1			0	1
$\phi$	$\phi$	$\phi$			$\phi$	$\phi$
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$			$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	$\{q_2\}$	$\{q_2\}$		$\{q_0\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_2\}$ F	$\phi$	$\phi$	$\Rightarrow$	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$		$\{q_0, q_2\}$ F	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_0, q_2\}$ F	$\{q_0\}$	$\{q_0, q_1\}$		$\{q_0, q_1, q_2\}$ F	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_1, q_2\}$ F	$\{q_2\}$	$\{q_2\}$				
$\{q_0, q_1, q_2\}$ F	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$				

where F designates accepted final states, those containing  $q_2$ . Note however that the states  $\phi, \{q_1\}, \{q_2\}, \{q_1, q_2\}$  are inaccessible, i.e., cannot be reached from the start state  $q_0$  via any string of transitions. Excluding those four leaves the state transitions at right above, defining a 4-state DFA.

## • $\epsilon$ -NFA

- **NFA with (null) or  $\epsilon$  move** : If any finite automata contains  $\epsilon$  (null) move or transaction, then that finite automata is called NFA with  $\epsilon$  moves



STATES	0	1	EPSILON
A	B, C	A	B
B	-	B	C
C	C	C	-

## Epsilon ( $\epsilon$ ) – 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  $\epsilon$  moves including the state  $X$  itself. In other words,  $\epsilon$ -closure for a state can be obtained by union operation of the  $\epsilon$ -closure of the states which can be reached from  $X$  with a single  $\epsilon$  move in recursive manner.

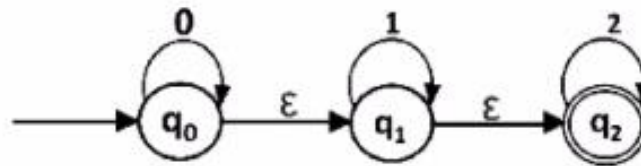
For the above example  $\epsilon$  closure are as follows :

$\epsilon$  closure(A) : {A, B, C}

$\epsilon$  closure(B) : {B, C}

$\epsilon$  closure(C) : {C}

Convert the given NFA- $\epsilon$  into its equivalent DFA.



• **Solution:**

Let us obtain the  $\epsilon$ -closure of each state.

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$$

$$\epsilon\text{-closure}(q_1) = \{q_1, q_2\}$$

$$\epsilon\text{-closure}(q_2) = \{q_2\}$$

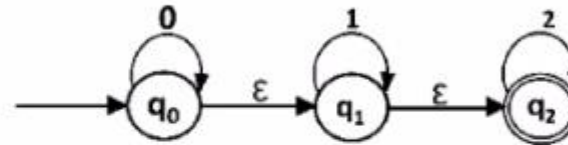
**Let the initial state of DFA be  $\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$   
 $= [q_0, q_1, q_2]$  ----- A.**

## Find transitions of A on inputs 0,1,2

$$\begin{aligned}\delta'(A, 0) &= \varepsilon\text{-closure}\{\delta((q_0, q_1, q_2), 0)\} \\ &= \varepsilon\text{-closure}\{\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)\} \\ &= \varepsilon\text{-closure}\{q_0\} \\ &= \{q_0, q_1, q_2\} \\ &= [q_0, q_1, q_2] \text{ ----- A}\end{aligned}$$

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

$$\begin{aligned}\delta'(A, 2) &= \varepsilon\text{-closure}\{\delta((q_0, q_1, q_2), 2)\} \\ &= \varepsilon\text{-closure}\{\delta(q_0, 2) \cup \delta(q_1, 2) \cup \delta(q_2, 2)\} \\ &= \varepsilon\text{-closure}\{q_2\} \\ &= \{q_2\} \\ &= [q_2] \text{ ----- C}\end{aligned}$$

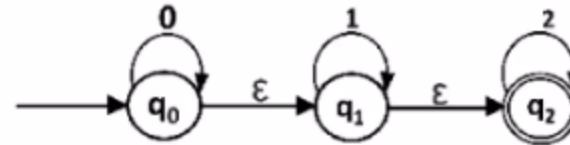


## Find transitions of B on inputs 0,1,2

$$\begin{aligned}\delta'(B, 0) &= \epsilon\text{-closure}\{\delta((q_1, q_2), 0)\} \\ &= \epsilon\text{-closure}\{\delta(q_1, 0) \cup \delta(q_2, 0)\} \\ &= \epsilon\text{-closure}\{\emptyset\} \\ &= \emptyset \text{ ----- } \emptyset\end{aligned}$$

$$\begin{aligned}\delta'(B, 1) &= \epsilon\text{-closure}\{\delta((q_1, q_2), 1)\} \\ &= \epsilon\text{-closure}\{\delta(q_1, 1) \cup \delta(q_2, 1)\} \\ &= \epsilon\text{-closure}\{q_1\} \\ &= \{q_1, q_2\} \\ &= [q_1, q_2] \text{ ----- } B\end{aligned}$$

$$\begin{aligned}\delta'(B, 2) &= \epsilon\text{-closure}\{\delta((q_1, q_2), 2)\} \\ &= \epsilon\text{-closure}\{\delta(q_1, 2) \cup \delta(q_2, 2)\} \\ &= \epsilon\text{-closure}\{q_2\} \\ &= \{q_2\} \\ &= [q_2] \text{ ----- } C\end{aligned}$$

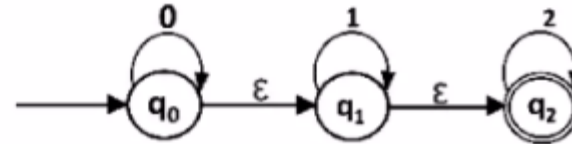


## Find transition of C on inputs 0,1,2

$$\begin{aligned}\delta'(C, 0) &= \varepsilon\text{-closure}\{\delta(q_2, 0)\} \\ &= \varepsilon\text{-closure}\{\phi\} \\ &= \phi \text{ ----- } \phi\end{aligned}$$

$$\begin{aligned}\delta'(C, 1) &= \varepsilon\text{-closure}\{\delta(q_2, 1)\} \\ &= \varepsilon\text{-closure}\{\phi\} \\ &= \phi \text{ ----- } \phi\end{aligned}$$

$$\begin{aligned}\delta'(C, 2) &= \varepsilon\text{-closure}\{\delta(q_2, 2)\} \\ &= \{q_2\} \\ &= [q_2] \text{ ----- } C\end{aligned}$$

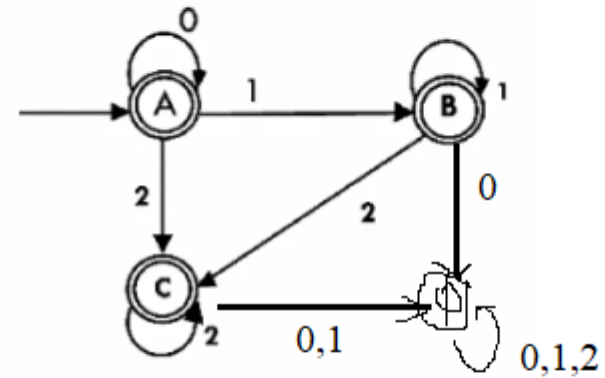


## Resultant DFA :

DFA - Transition table

State \ input	0	1	2
*A	A	B	C
• B	$\phi$	B	C
*C	$\phi$	$\phi$	C

DFA - Transition diagram

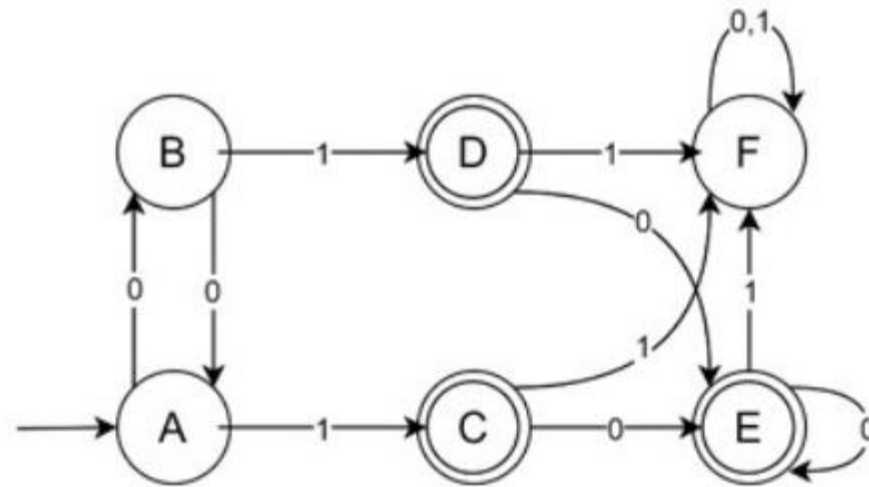


**Note:** Final state given nfa is  $\{q_2\}$   
Since  $q_2$  lies in the states  $A = \{q_0, q_1, q_2\}$   
 $B = \{q_1, q_2\}$   
 $C = \{q_2\}$   
A, B, C becomes final states of resultant DFA.

## Minimisation of DFA

- Create a table with all possible pairs of states from the given DFA. Each cell in the table will represent a pair of states, like (A, B), (C, D), etc.
- Mark all pairs where P belongs to the final state, and Q does not belong to the final state. This means that for each pair of states in the table, check whether one state is a final state, and the other is not. If this condition is met, we mark that pair in the table.
- If there are any unmarked pairs PQ, such that the transition of P on X and the transition of Q on X is marked, then mark PQ, where X is an input symbol. After marking the pairs in the second step, we move on to the unmarked pairs. For each unmarked pair (P, Q), check their transitions on every input symbol (X). If both P and Q transition to a pair that is already marked in the table, then mark the pair (P, Q).
- Repeat step 3 until you cannot make any new markings in the table. This means we have exhausted all possible combinations and identified all the distinguishable pairs of states.
- At the end combine all the unmarked pairs and make them a single state in the minimized DFA.

Consider we have a DFA like below –



	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

After the first step, we jump to the second step to fill the table –

	A	B	C	D	E	F
A						
B						
C	*	*				
D	*	*				
E	*	*				
F			*	*	*	

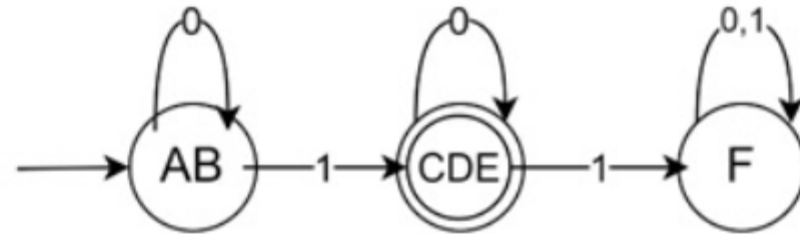
- Based on the 3<sup>rd</sup> step,

	A	B	C	D	E	F
A						
B						
C	*	*				
D	*	*				
E	*	*				
F	*	*	*	*	*	

In step 4, it states repeat step 3 until there is no other marks. You can check, but here we do not have any new marking in the next iteration. So go to the next step.

The unmarked pairs are (B, A), (D, C), (E, C), (E, D)

Now let us draw the **minimized DFA** –



(D, C), (E, C), (E, D) contains the common elements so these are merged. And we are getting the minimized DFA.