Mohsin Iban Hossain

AIUB, Theory of Computation Notes

THEORY OF COMPUTATION

Table of Contents

Sl. No	Topic	Pages
1	Introduction to Theory of computation	03-05
2	Deterministic Finite Automaton (DFA)	06-25
3	Non-deterministic Finite Automaton (NFA)	26-35
4	Regular Languages	36-42
5	Conversion of NFA to DFA	43-47
6	Conversion of Epsilon (ε)NFA to DFA	48-54
7	Regular Expression (RE)	55-63
8	Conversion of RE to NFA	64-68
10	Conversion of DFA TO RE	69-82
11	Conversion of NFA TO RE	69-82

INTRODUCTION TO THEORY OF COMPUTATION

“Theory” of “Computation”

⇨ Theory:

- **General & Abstract**: Applies to many systems, independent of technology.
- **Precision**: Used to prove formal mathematical theorems.
- **Positive Results**: Proves what can be computed (e.g., correctness of algorithms).
- **Negative Results**: Proves what cannot be computed (e.g., unsolvable problems with certain costs).

⇨ Computation:

- Any task can be performed by machine that is known as computation.

Automata Theory

➔ Automata Overview:

- **Meaning**: From Greek (Αυτόματα) – "**self-operating**."
- **Study of**: Abstract machines & computational problems (**formal languages**).
- **Role**: Finite representation of infinite sets (**language theory**).
- **Purpose**: Models for computing machines (input → process → output).
- **Use Cases**:
    - **Finite Automaton**: Text processing, compilers design, hardware design.
    - **Context-Free Grammar**: Programming languages, AI.

Computability Theory

➔ Computability Theory Overview:

- **Unsolvable Problems**: Some problems cannot be solved by computers.
- **Example**: Determining if a mathematical statement is true or false (undecidable).
- **Theoretical Models**: Ideas that led to actual computer development.
- **Problem Classification**: Divides problems into solvable and unsolvable categories.

Complexity Theory

➔ Computability & Complexity Theory

- **Computability Theory**: Determines if a problem is **Solvable / Not Solvable**.
- **Complexity Theory**: If solvable, classifies it as **Easy / Hard**.
- **Problem Classification**:
  - Modify the problem to make it easier.
  - Settle for approximate solutions.
  - Some problems are hard only in worst-case scenarios.
- **Cryptography**: Requires problems that are **hard to solve** without a key.

Summary

Theory	Concept	Examples / Applications
Computability Theory	Determines if a problem is **Solvable / Not Solvable**	If solvable → moves to Complexity Theory
Complexity Theory	Classifies solvable problems as **Easy / Hard**	- **Easy**: Sorting (e.g., arranging millions of numbers) - **Hard**: Scheduling (e.g., conflict-free class routine)
Automata Theory	Deals with **Mathematical Models**	**Classification**: - **Finite Automaton**: Used in **text processing, compiler design, hardware design** - **Context-Free Grammar**: Used in **programming languages, AI**

DETERMINISTIC FINITE AUTOMATON (DFA)

Automata Theory

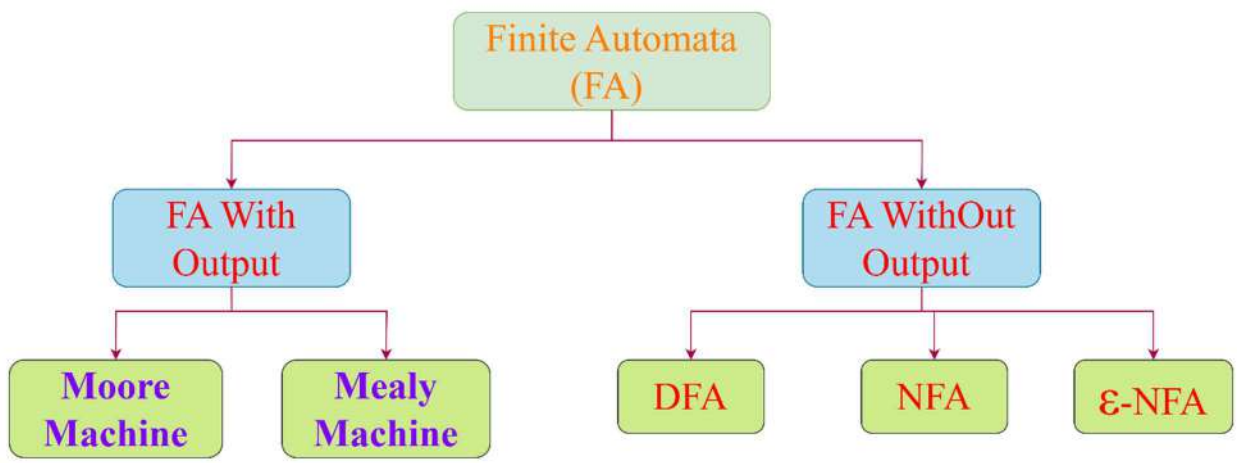
Concept	Description	Examples / Applications
Origin	Comes from the Greek word **"Αυτόματα"** meaning "self-operating"	—
Definition	Study of **abstract mathematical machines** and **computational problems**	—
Purpose	Theoretical models for **computing machines** (Input → Process → Output)	**Used for proofs** about computability
Formal Languages	Automaton represents **infinite language sets**	Formal languages specify problems for computation
Types of Automata	**- Finite Automaton** **- Context-Free Grammar**	- **Finite Automaton**: Used in **text processing, compiler design, hardware design** - **Context-Free Grammar**: Used in **programming languages, AI**

Terminology/Prerequisites

Concept	Definition	Examples
Symbol	A **single character** or element from an alphabet.	a, b, c, … z; 0, 1, 2, … 9; A, 3, f, 5, …
Alphabet (Σ)	A **non-empty finite set** of symbols, denoted by **Σ**.	**Σ** = {a, b}, **Σ** = {0, 1}, **Σ** ={0, 1, … 9}
String (W)	A **finite sequence** of symbols from an alphabet.	Σ = {a, b}, **W = ab, ba**

Concept	Definition	Examples
Language (L)	A **collection of all possible strings** over an alphabet.	**Finite Example**: $\Sigma = \{a, b\}$, L_1 = {Set of all strings of length 2} = {aa, ab, bb, ba} L_2 = {Set of all strings of length 3} = {aaa, aab, aba, abb, baa, bab, bba, bbb} **Infinite Example**: $\Sigma = \{a, b\}$, L_3 = {Set of all strings starting with 'a'} = {a, aa, ab, abb, aba, …}
Power of Alphabet (Σ^n)	Represents **all possible strings of a given length** over an alphabet.	$\Sigma^0 = \{\varepsilon\}$ (empty string) $\Sigma^1 = \{a, b\}$ (strings of length 1) $\Sigma^2 = \{aa, ab, ba, bb\}$ (strings of length 2) $\Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$ (strings of length 3) $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots^*$ (all possible strings of any length)

Finite State Machine



Finite Automata

➔ Finite Automaton Overview:

- **Simple Model:** Used for devices with limited memory.
- **Common Applications:** Dishwashers, electronic thermostats, digital watches, calculators.
- **Key Focus:** Limited states and transitions based on input.
- **Goal:** Design and understand the behavior of devices using finite memory.

➔ Finite Representation of an Infinite Language

Finite Example:

- **Alphabet:** ∑ = {a, b}
- **Language (L(M)):** Set of all strings of length 2
  - **L(M) = {aa, ab, bb, ba}**
  - **Accepted String:** "ab" (Machine will accept)
  - **Rejected String:** "abc" (Machine will not accept)

Infinite Example:

- **Alphabet:** ∑ = {a, b}
- **Language (L(M)):** Set of all strings starting with 'a'
  - **L(M) = {a, aa, ab, abb, aba, ...}**
  - **Machine Behavior:** Will search indefinitely for strings like "baaaa."

➔ **Problem:** An infinite language requires infinite time for the machine to search through every possible string. This is impractical.

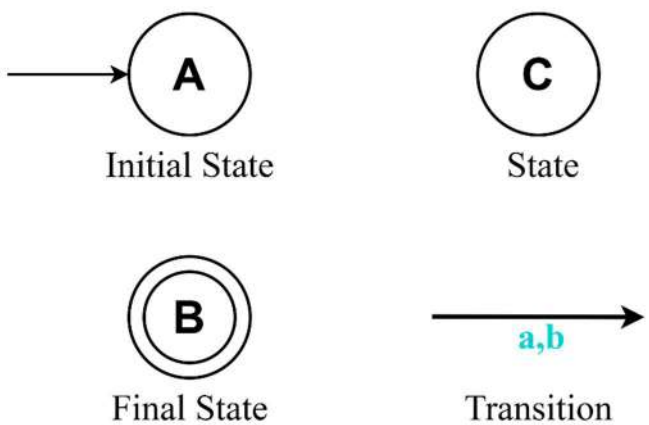
➔ Solution: Finite Representation (FR)

- **Goal:** Use a finite model (e.g., regular expression, finite automaton) to represent infinite languages.
  - **Example FR:** `a(a|b)*` for strings starting with **'a'**.
  - **Approach:** Use compact representations (like regular expressions) to avoid infinite search.

Finite Representation (FR) / Finite Automaton (FA)

- **Definition:** If a language can be recognized with a finite number of states, it's called Finite Representation (FR) or Finite Automaton (FA).
- **Key Point:** FA uses a finite number of states to represent and accept or reject strings from a language.

➔ Finite Representation:

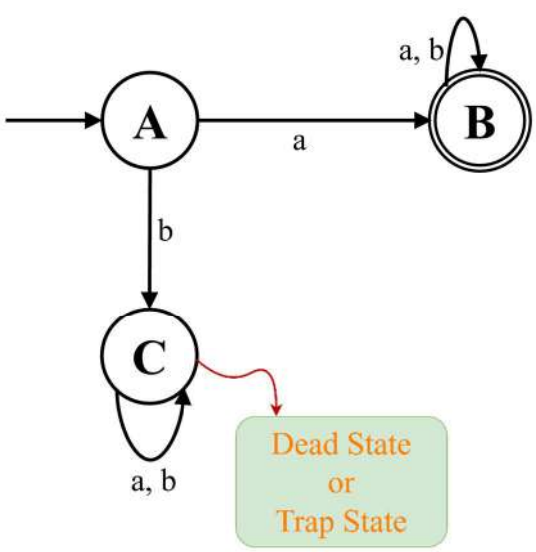


Example 1:

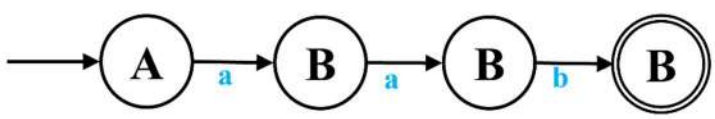
➔∑ = {a, b}

➔L1 = Set of all strings starting with 'a'

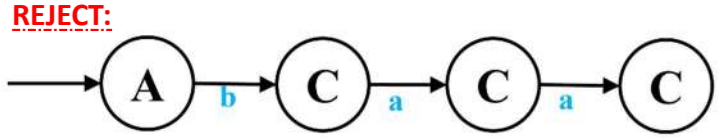
➔L1 = {a, aa, ab, abb, aba, ...}



ACCEPT:

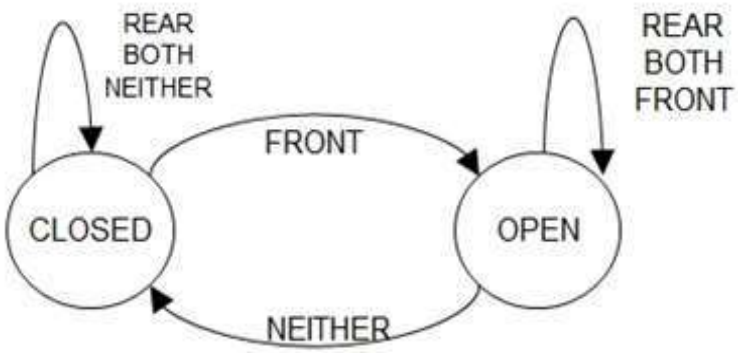


REJECT:



Parts of a Finite Automaton (FA)

Example:



	Input Signal				
		Front	Rare	Both	Neither
State	Closed	Open	Closed	Closed	Closed
	Open	Open	Open	Open	Closed

Fig: State Transition for Automatic Door Controller

1. **Input (Language):** Defined set of inputs.
   - ***Example:*** **FRONT, REAR, BOTH, NEITHER.**
2. **Set of States:** Different conditions the system can be in.
   - ***Example:*** **OPEN, CLOSED (for an automatic door).**
3. **Initial (Start) State:** The starting condition.
   - *Example:* CLOSED (default state of the door).
4. **Transition Rules:** Define state changes based on input.
   - *Example:* If in CLOSED state and input is FRONT, move to OPEN state.
5. **Final/Accept States:** The goal state(s).
   - *Example:* CLOSED when the last input is NEITHER.

Deterministic Finite Automaton (DFA)

1. **Deterministic Computation:**
   - Each step follows uniquely from the previous one.
   - Given a state and input, the next state is determined.
2. **Transition Rules:**
   - Each state has exactly **one** transition per input.
   - If there are **n** inputs, each state has **n** transitions.
3. **Key Properties:**
   - **One Start State** → Begins the transition.
   - **One or More Final States** → Marks the completion.

Formal Definition - DFA

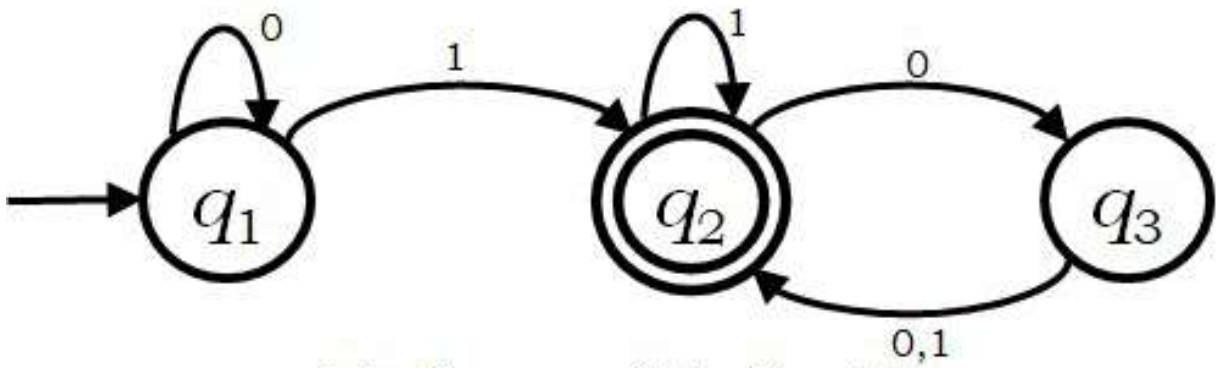
➔ A DFA is a **5-tuple**: **$(Q, \Sigma, \delta, q_0, F)$**

1. **Q** → Finite set of states.
2. **Σ** → Finite input alphabet.
3. **$\delta: Q \times \Sigma \to Q$** → Transition function (maps state & input to next state).
4. **$q_0 \in Q$** → Start state.
5. **$F \subseteq Q$** → Set of accept (final) states.

Language Recognition:

- If **A** is the set of strings accepted by DFA machine **M**, then:
  - **L(M) = A** → DFA recognizes **A**.
  - **M accepts A** if it reaches a final state after reading **input.**

Example 1:



state diagram of Machine M_1

➔Here,

- $M_1 = (Q, \Sigma, d, q_0, F)$,
  - Where:

$Q = \{q1, q2, q3\}$,

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

$q0 = \{q1\}$,

$F = \{q2\}$,

δ is describe as:

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

$\delta(q_2,0) = q_3, \quad \delta(q_2,1) = q_2,$

$\delta(q_3,0) = q_2, \quad \delta(q_3,1) = q_2.$

Transition Function

or

δ	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Transition Table

Mathematical Formalization of DFA Computation

⇨ Let **M = (Q, Σ, δ, q_0, F)** be a DFA.

For a string **w = $w_1w_2\ldots w_n$ ∈ Σ*** (where each w_i ∈ Σ), **M** accepts **w** if a sequence of states $r_0, r_1, \ldots, r_n$ exists in **Q**, satisfying:

1. **Start State:** $r_0 = q_0$
2. **State Transition:** $\delta(r_i, w_{i+1}) = r_{i+1}$ **for** $i = 0, 1, \ldots, n-1$
3. **Final State:** $r_n \in F$

Language Recognition:

- **M recognizes language L** if **L = {w | M accepts w}** (i.e., all strings leading to a final state).

Example 2:

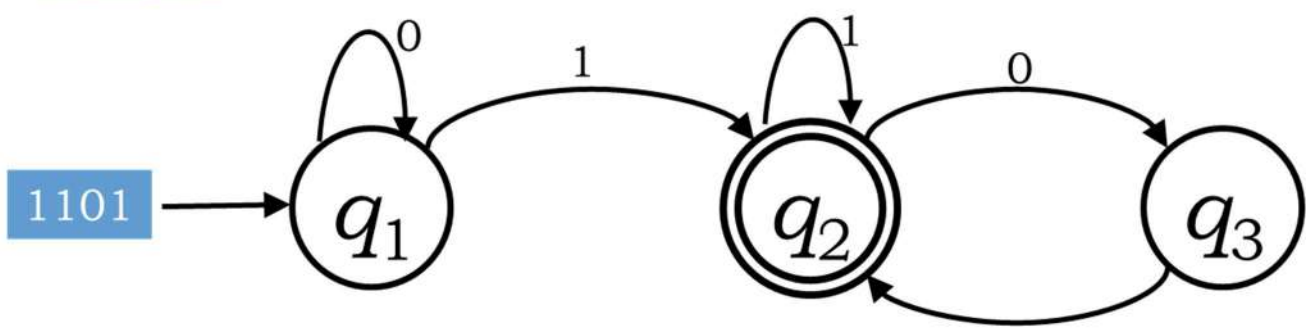


Figure: Finite Automaton M_1.

➔We know:

➢ M= (Q, Σ, δ, q_0, F),

➢ Where:

- ✓ $Q = \{q_1, q_2, q_3\}$,
- ✓ $\Sigma = \{0, 1\}$,
- ✓ $q_0 = q_1$,
- ✓ $F = \{q_2\}$,

$$\delta = \boxed{\begin{array}{ll} \delta(q_1,0) = q_1, & \delta(q_1,1) = q_2, \\ \delta(q_2,0) = q_3, & \delta(q_2,1) = q_2, \\ \delta(q_3,0) = q_2, & \delta(q_3,1) = q_2. \end{array}}$$

- Input string $w = w_1w_2w_3w_4$ = **1101** to M_1 gives a sequence of states r_0, r_1, r_2, r_3, r_4 in the following computation (here $n = 4$) –
- **Start in state $r_0 = q_1$;** → $r_0 = q_0$,
- $\delta(r_0, w_1) = \delta(q_1, 1) = q_2 = r_1$; → $\delta(r_i, w_{i+1}) = r_{i+1}$, **for i = 0, 1, 2, … , n-1,**
- $\delta(r_1, w_2) = \delta(q_2, 1) = q_2 = r_2$; → $\delta(r_i, w_{i+1}) = r_{i+1}$, **for i = 0, 1, 2, … , n-1,**
- $\delta(r_2, w_3) = \delta(q_2, 0) = q_3 = r_3$; → $\delta(r_i, w_{i+1}) = r_{i+1}$, **for i = 0, 1, 2, … , n-1,**
- $\delta(r_3, w_4) = \delta(q_3, 1) = q_2 = r_4$; → $r_n \in F$.

Step	Current State (r_i)	Input Symbol (w_{i+1})	Next State (r_{i+1})	Transition Used (δ)
0	**q1** (Start)	1	**q2**	δ(q1,1) = q2
1	**q2**	1	**q2**	δ(q2,1) = q2
2	**q2**	0	**q3**	δ(q2,0) = q3
3	**q3**	1	**q2**	δ(q3,1) = q2
4	**q2** (Accept)	-	-	**w is Accepted**

Conclusion:

- The input **w = 1101** leads to **q2**, which is an **accept state**.
- **M_1 accepts w**, meaning **w ∈ L(M_1)**.
- **Language L = {w | M_1 accepts w}**.

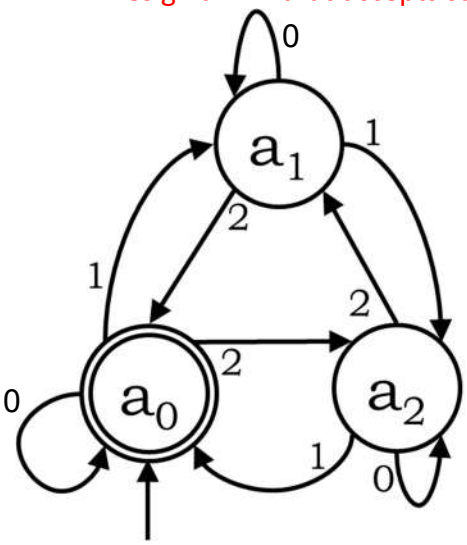
DFA design issues

- Identify the **pattern/type** of input strings in the language.
- Match input pattern **step-by-step** with states & transitions.
- Ensure **necessary conditions** before completing the state diagram.
- **Rules to follow:**
  - Each state must have **one transition per input symbol**.
  - **Self-loops** allow zero or more occurrences of an input symbol.
  - There can be **one or more final states**.
- Create **test input strings** (both **ACCEPTED** & **REJECTED**).
- Test strings **while drawing/updating** the state diagram.

Designing DFA

Example 1:

➔ Design a DFA that accepts strings where the sum of all symbols (0,1,2) is a multiple of 3.



➔ DFA for Sum of Symbols ≡ 0,1,2 (mod 3):

- **Alphabet**: Σ = {0,1,2}
- **Language**: Strings where the sum of symbols is a **multiple of 3**.
- **States**: Q = {a_0, a_1, a_2}, where **a_i represents S mod 3 = i**.
- **Start State**: $q_0 = a_0$
- **Final State**: F = {a_0}
- **Transition Table (δ_1):**

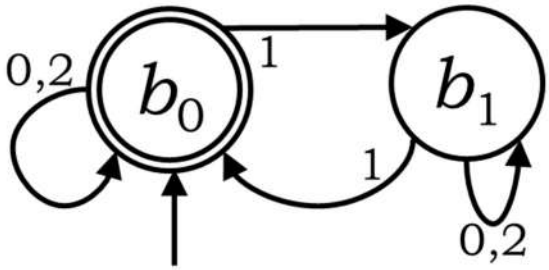
δ	0	1	2
a_0	a_0	a_1	a_2
a_1	a_1	a_2	a_0
a_2	a_2	a_0	a_1

Input example: 01120101

Input symbol: Accepted

Example 2:

➔ Design a DFA that accepts strings where the sum of all symbols (0,1,2) is an even number.



DFA for Sum of Symbols ≡ 0,1,2 (mod 2) (Even Sum):

- **Alphabet**: Σ = {0,1,2}
- **Language**: Strings where the sum of symbols is an **even number**.
- **States**: Q = {b_0, b_1}, where **b_i represents S mod 2 = i**.
- **Start State**: $q_0 = b_0$
- **Final State**: F = {b_0}
- **Transition Table (δ_2):**

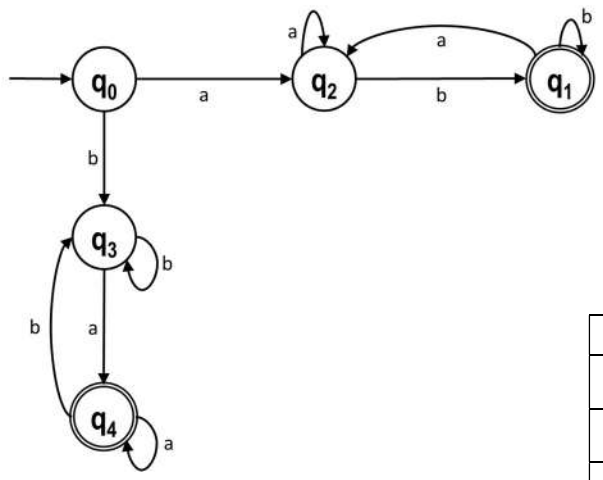
δ	0	1	2
b_0	b_0	b_1	b_0
b_1	b_1	b_0	b_1

Input example: 01120101

Input symbol: Accepted

Example 3:

➔ Identify the Components of the DFA



Components of the DFA:

- **Alphabet**: Σ = {a, b}
- **States**: Q = {q_0, q_1, q_2, q_3, q_4},
- **Start State**: q_0 = q_0
- **Final State**: F = {q_1, q_4}
- **Transition Table (δ):**

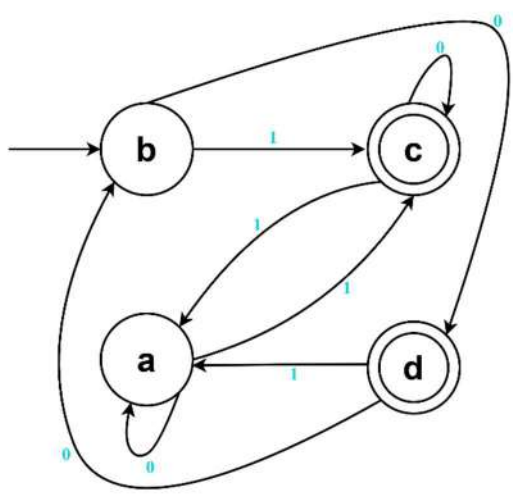
δ	a	b
q_0	q_2	q_3
q_1	q_2	q_1
q_2	q_2	q_1
q_3	q_4	q_3
q_4	q_4	q_3

Example 4:

➔ Design a DFA that Q = {a,b,c,d}, Σ = {0, 1}, q_0 = b, F = {c,d} ,

δ	0	1
a	a	c
b	d	c
c	c	a
d	b	a

➔ Solution:



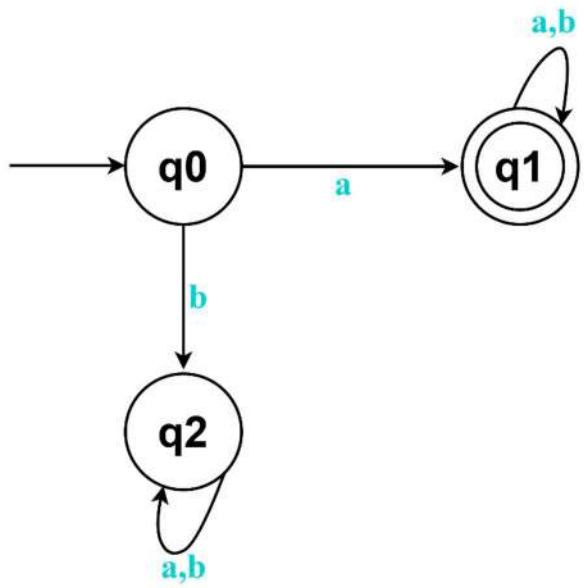
Example 5:

➔ Given the state diagram of the machine ($\{q_0, q_1, q_2\}$, $\{a, b\}$, δ, q_0, $\{q_1\}$) Where δ **is given the following table**

δ	a	b
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_2	q_2

➔Design The DFA

➔ Solution:

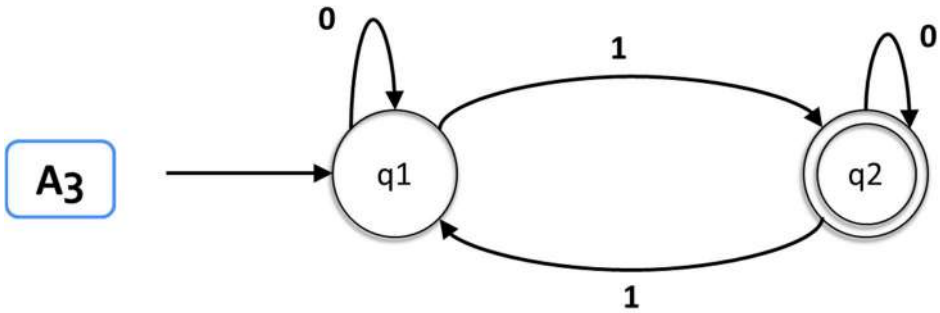


Example 6:

➔ **A3 = {w : w is a binary string containing an odd number of *1*s}. $\sum = \{0,1\}$**

➔ **So, $w_1 = 1$, $w_2 = 10$, $w_3 = 101$, $w_4 = 10101$, $w_5 = 10101$…**

Now, L = {1, 100, 10101, 11010, 11100…}



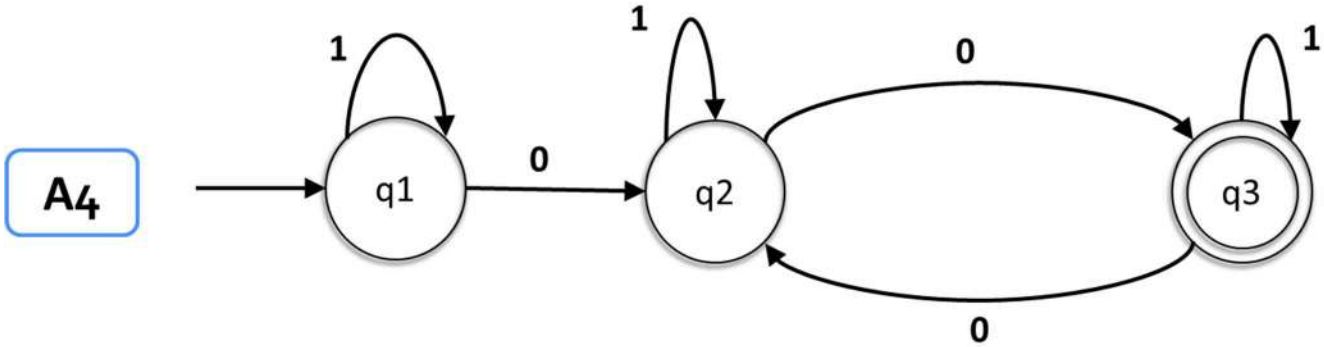
Accepted

Example 7:

➔ A4 = {w : w is a binary string containing an even number of *0*s}. ∑ = {0,1}

➔ So, w_1 = 00, w_2 = 001, w_3 = 0101, w_4 = 01010, w_5 = 0101010…

Now, L = {00, 100, 1010, 11010, 11100…}

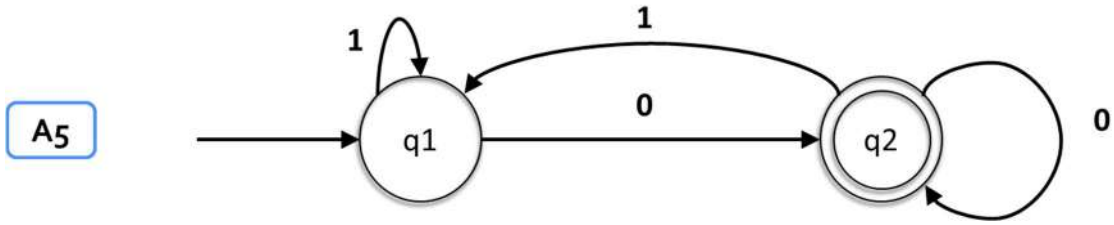


Example 8:

➔ A5 = {w| w ends with a *0*}. ∑ = {0,1}

➔ So, w_1 = 010, w_2 = 000, w_3 = 10, w_4 = 010101, w_5 = 0101010…

Now, L = {0, 10, 100, 11010, 11100…}

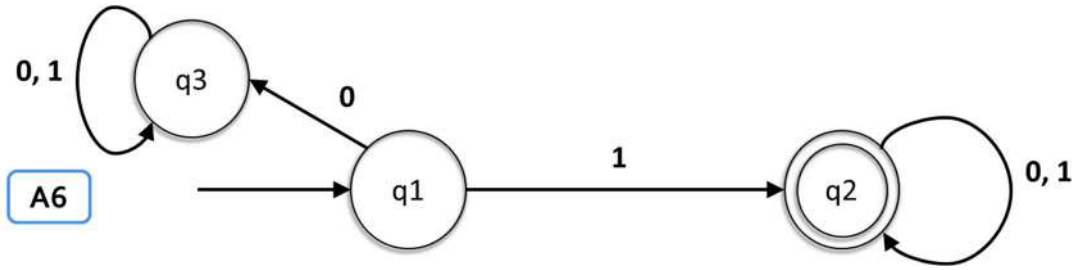


Example 9:

➔ A6 = {w| w begins with a *1*}. ∑ = {0,1}

➔ So, w_1 = 100, w_2 = 001, w_3 = 10, w_4 = 01010, w_5 = 101010…

Now, L = {1, 11, 101, 1001, 11100…}

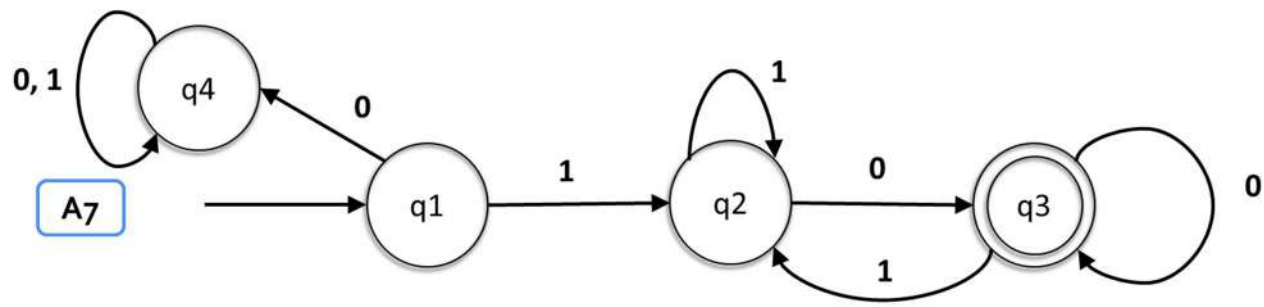


Example 10:

➔ A7 ={w| w begins with a *1* and ends with a *0*}. $\sum = \{0,1\}$

➔ So, $w_1 = 100$, $w_2 = 001$, $w_3 = 10$, $w_4 = 01010$, $w_5 = 101010\ldots$

Now, L = {1, 100, 1010, 11010, 11100…}

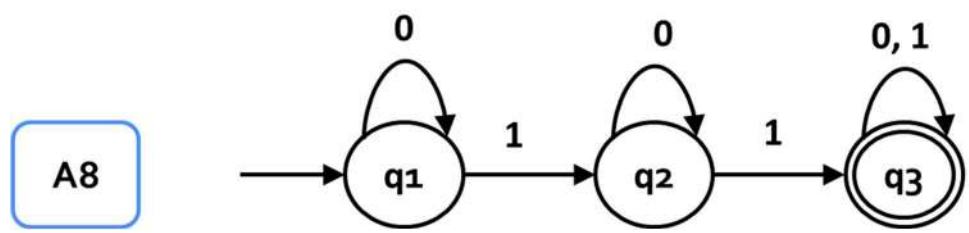


Example 11:

➔ A8 ={w| w has at least two *1*s}. $\sum = \{0,1\}$

➔ So, $w_1 = 1010$, $w_2 = 001$, $w_3 = 101$, $w_4 = 01010$, $w_5 = 101010\ldots$

Now, L = {11, 1100, 1010, 11010, 11100…}

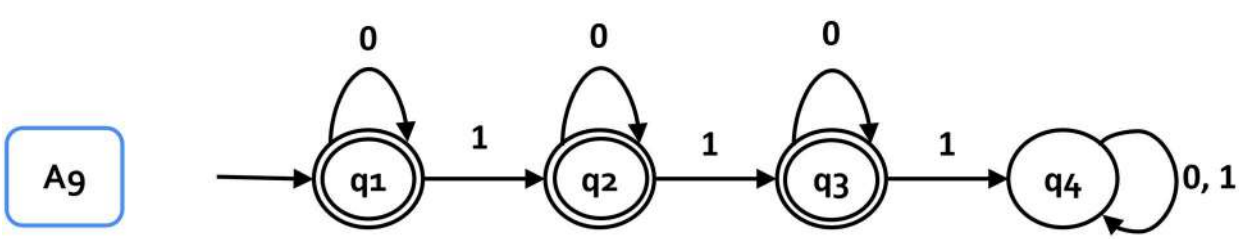


Example 12:

➔ A9 ={w| w has at most two *1*s}. $\sum = \{0,1\}$

➔ So, $w_1 = 1010$, $w_2 = 001$, $w_3 = 000$, $w_4 = 010110$, $w_5 = 10100\ldots$

Now, L = {11, 1100, 1010, 11010, 11100…}

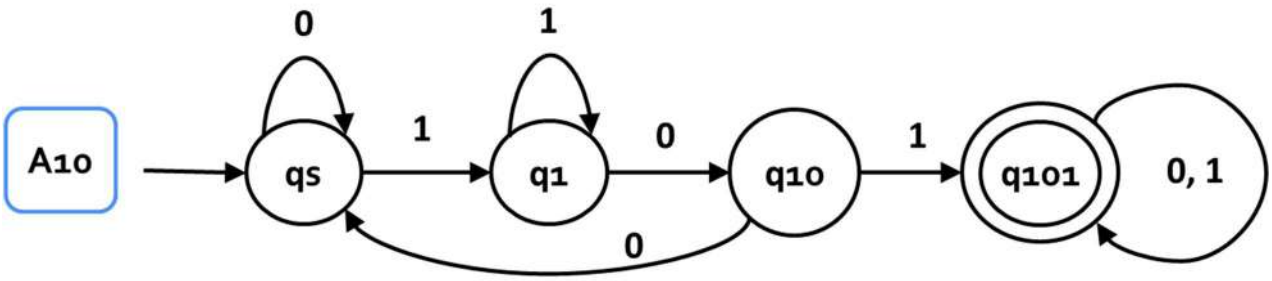


Example 13:

➔ A10 ={w| w has substring 101}. $\sum = \{0,1\}$

➔ So, $w_1 = 1010$, $w_2 = 001$, $w_3 = 000$, $w_4 = 000110$, $w_5 = 10100$…

Now, L = {101, 1101, 10101, 11010, 111010…}

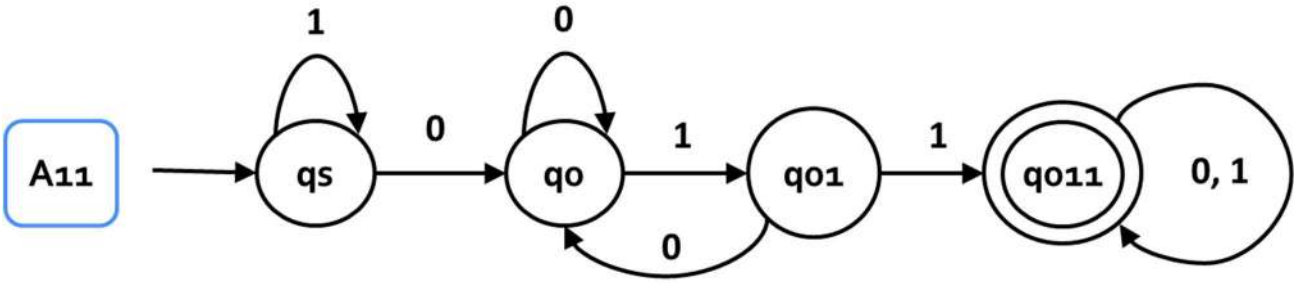


Example 14:

➔ A11 ={w| w has substring 011}. $\sum = \{0,1\}$

➔ So, $w_1 = 10110$, $w_2 = 001$, $w_3 = 000$, $w_4 = 000110$, $w_5 = 101100$…

Now, L = {011, 11011, 10101011, 110110, 1011010…}



➔ What happens for the language, **A11 ={w| w does not have substring 011}**?

➔ For A_{11} = **{w | w has substring "011"}**, the DFA **accepts** strings containing "011".

➔ For A'_{11} = **{w | w does not have substring "011"}**, the DFA **rejects** any string with "011" and stays in states ensuring "011" never appears.

➔ A **dead state** is used for rejection.

Problem 1:

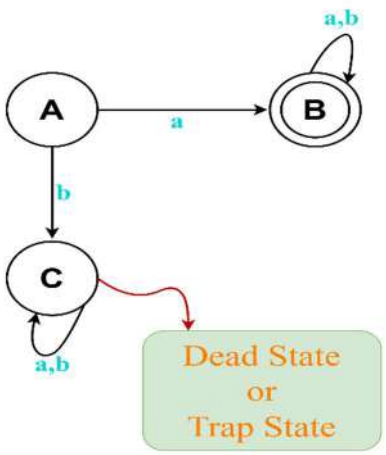
Construct a DFA which accepts set of all string over the ∑ = {a,b}, where each string start with an **"a"**

Or L(M) = {w| w Start with an "a"}

➔ **Is Given,**

∑ = {a,b},

L(M) ={ a, aa, aba, aabba,….}



Problem 2:

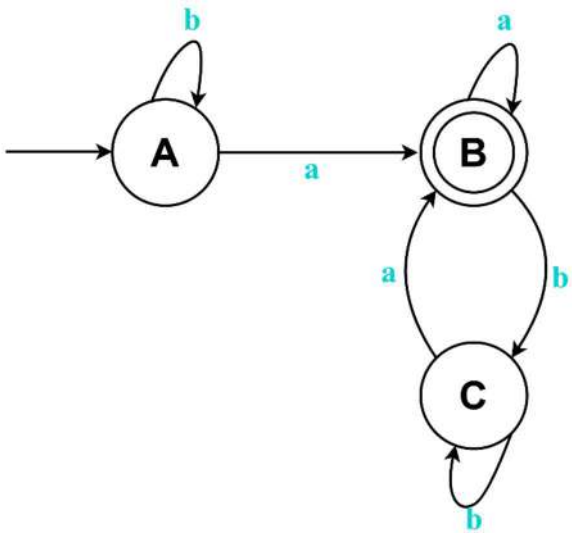
Construct a DFA which accepts set of all string over the ∑ = {a,b}, where each string end with an **"a"**

Or L(M) = {w| w End with an "a"}

➔ **Is Given,**

∑ = {a,b},

L(M) ={a, aa, aba, aabba, abababa, bba, ….}

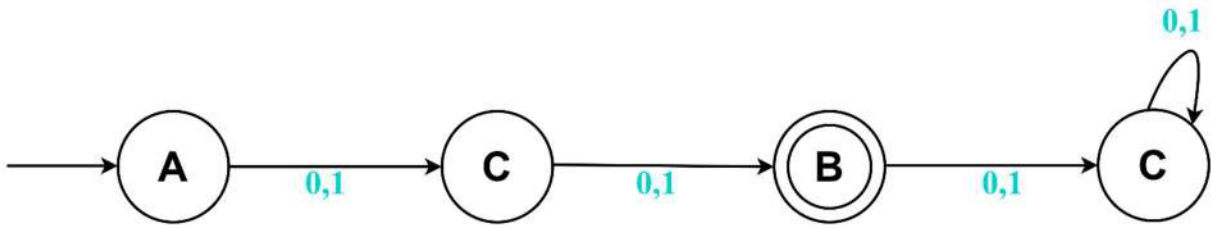


Problem 3:

Construct a DFA that accepts set of all string over the ∑ = {0,1}, of length 2.

➔ **Is Given,**

∑ = {0,1},
L(M) ={00,01,10,11}



Problem 4:

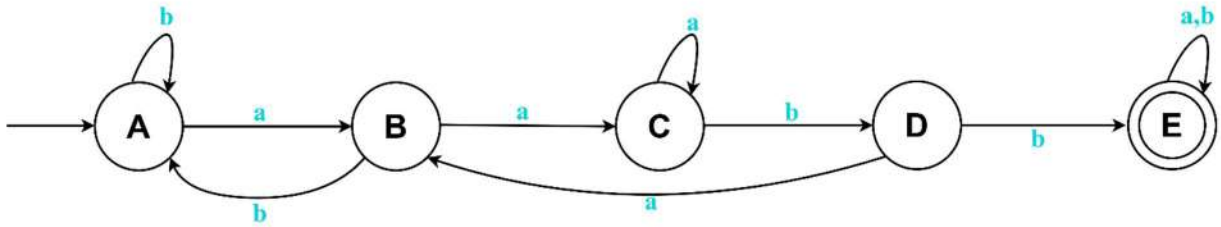
Construct a DFA that accepts any string over the ∑ = {a,b}, that **does not contain** the string "aabb" in it.

➔ **Is Given,**

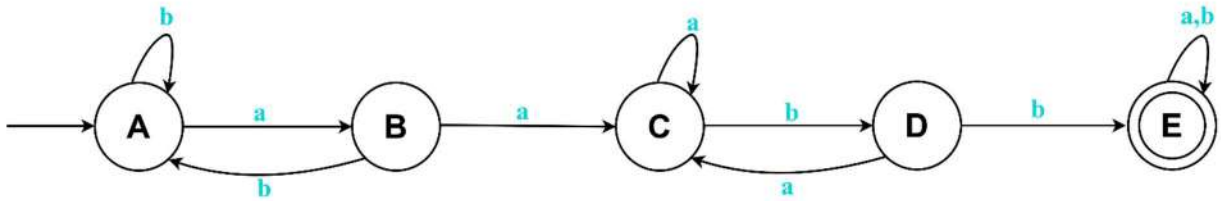
∑ = {a,b},

➔Let, Construct a DFA that accepts any string over the ∑ = {a,b}, that **contains** the string "aabb" in it.

L(M) = {aabb, baabb, babaabb, babaaabb, aabaabb, aababaabb,…}



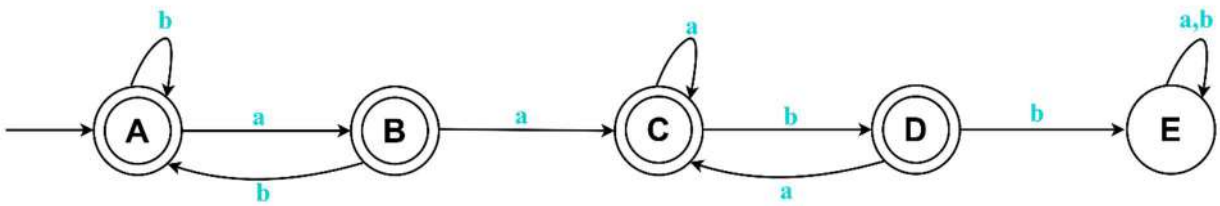
OR



➔ Okay, This is not our desired output since it does contain the string "aabb."

➔ So, To have our desired output, we just need to:

- Flip the states.
- Make the final states non-final state.
- Make the non-final states final state.



Problem 5:

Design DFA for the following Language. Where ∑ = {a,b} and give the formal definition of your machine. (Any-ONE)

i. L= {w | ends with 'abbab' substring}

ii. L= { w| w contains exactly two a's and odd number of b }

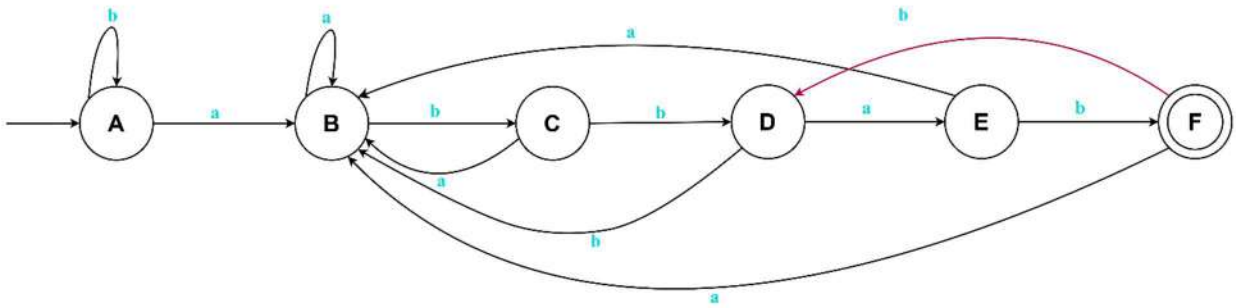
Mode1#

(i)

➔ **Is Given,**

∑ = {a,b},

L(M1) ={abbab, babbab, aabbab, ababbab, abbbabbab, abbabbab, abbababbab,…….}

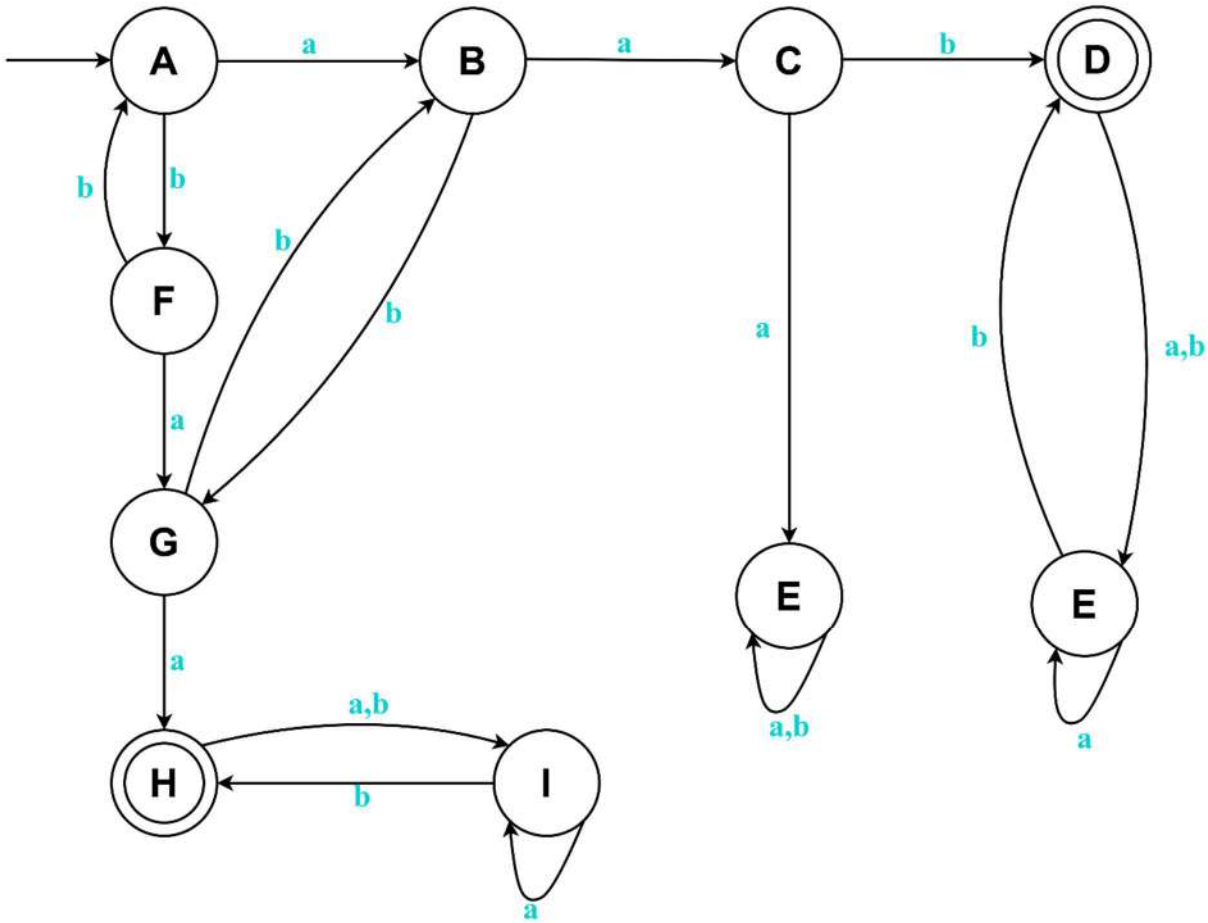


(ii)

➔ **Is Given,**

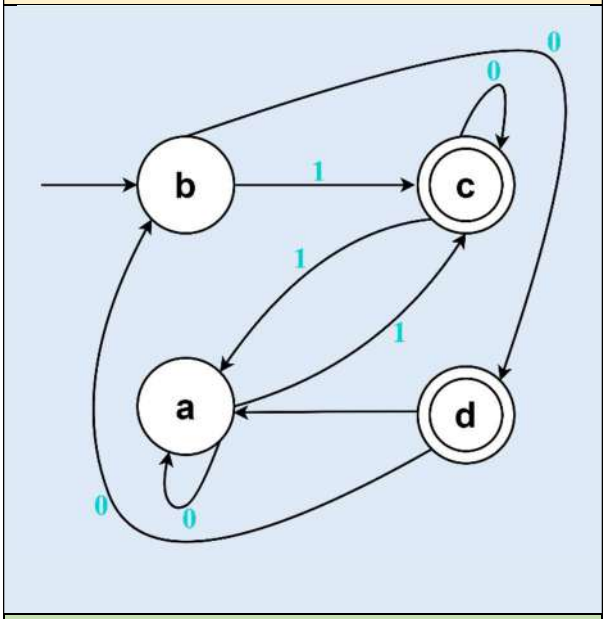
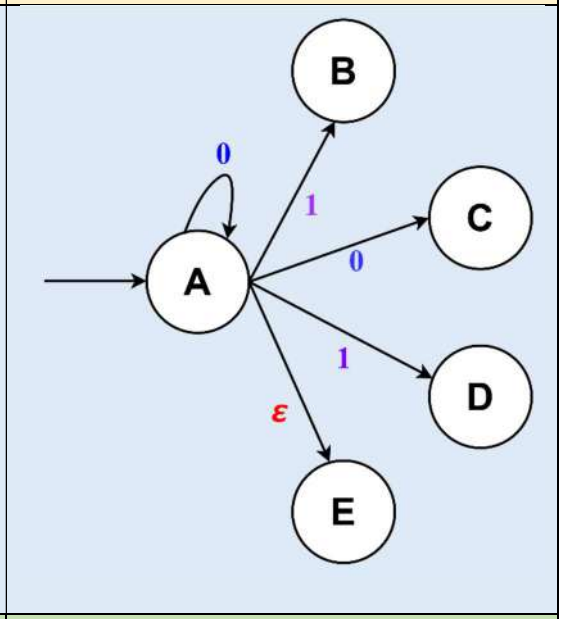
∑ = {a,b},

L(M1) ={baa, aab, baabb, babab, bbababb, bbababb, abbab,…}



NON-DETERMINISTIC FINITE AUTOMATA (NFA)

DFA vs NFA

DFA	NFA
	
»In DFA, given the current state we know what the next state will be » It has only one unique next state » It has no choices or randomness » It is simple and easy to design	»In NFA, given the current state there could be multiple next states. » The next state may be chosen at random. » All the next states may be chosen in parallel » **ε-moves** are allowed. (ε = empty string)

Running an NFA

➔ NFA Computation:

- Machine **splits** into multiple copies for each choice.
- Each copy follows a different path **in parallel**.
- **ε-moves** allow transitions without consuming input.
- Copies **die** if no valid transition exists.
- If **any copy** reaches an accept state, the input is accepted.
- **Nondeterminism = parallel computation.**

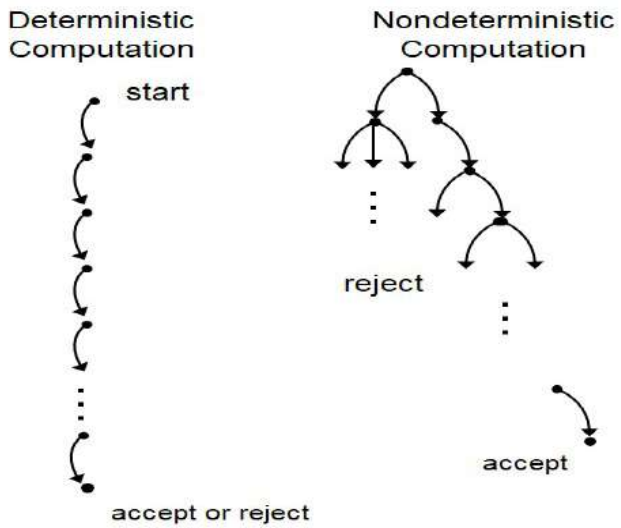
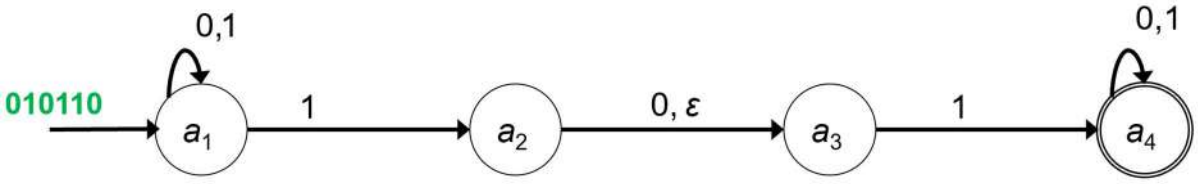
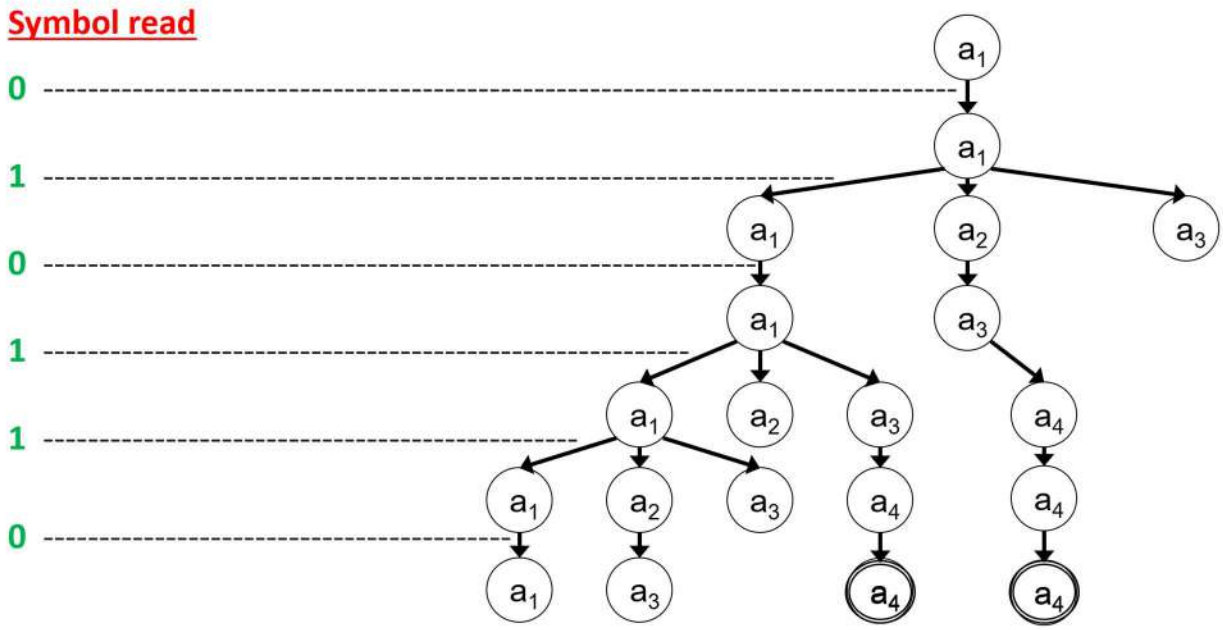


Figure: Deterministic and nondeterministic computations with an accepting branch

Simulation – NFA Tree



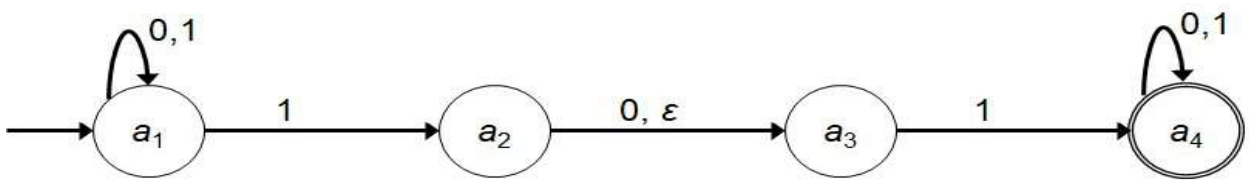


Formal Definition of NFA & Epsilon (ε) NFA

➔ NFA as a 5-tuple M= (Q, Σ, δ, q_0, F):

- **Q:** Finite set of states.
- **Σ:** Finite alphabet.
- **δ:** Transition function, mapping **(state × symbol/ε) → set of states.** ($Q \times \Sigma_\varepsilon \rightarrow P(Q) = 2^Q$)
- **q_0:** Start state.
- **F:** Set of accepting states.

Example 1:



➔ Here,

»$Q = \{a_1, a_2, a_3, a_4\}$,

»$\Sigma = \{0, 1\}$,

»$q_0 = a_1$,

»$F = \{a_4\}$,

»δ=

δ	0	1	ε
a_1	$\{a_1\}$	$\{a_1, a_2\}$	ϕ
a_2	$\{a_3\}$	ϕ	$\{a_3\}$
a_3	ϕ	$\{a_4\}$	ϕ
a_4	$\{a_4\}$	$\{a_4\}$	ϕ

Fig: Transition table

δ_1 in form $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$

$\delta_1(a_1, 0) = \{a_1\}$
$\delta_1(a_1, 1) = \{a_1, a_2\}$
$\delta_1(a_1, \varepsilon) = \phi$
$\delta_1(a_2, 0) = \{a_3\}$
$\delta_1(a_2, 1) = \phi$
$\delta_1(a_2, \varepsilon) = \{a_3\}$
$\delta_1(a_3, 0) = \phi$
$\delta_1(a_3, 1) = \{a_4\}$
$\delta_1(a_3, \varepsilon) = \phi$
$\delta_1(a_4, 0) = \{a_4\}$
$\delta_1(a_4, 1) = \{a_4\}$
$\delta_1(a_4, \varepsilon) = \phi$

Fig: Transition Functions

»$Q \times \Sigma_\varepsilon = \{(a_1, 0), (a_1, 1), (a_1, \varepsilon), (a_2, 0), (a_2, 1), (a_2, \varepsilon), (a_3, 0), (a_3, 1), (a_3, \varepsilon), (a_4, 0), (a_4, 1), (a_4, \varepsilon)\}$

»$\mathcal{P}(Q) = \{\phi, \{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}, \{a_1,a_2\}, \{a_1,a_3\}, \{a_1,a_4\}, \{a_2,a_3\}, \{a_2,a_4\}, \{a_3,a_4\}, \{a_1,a_2,a_3\},$

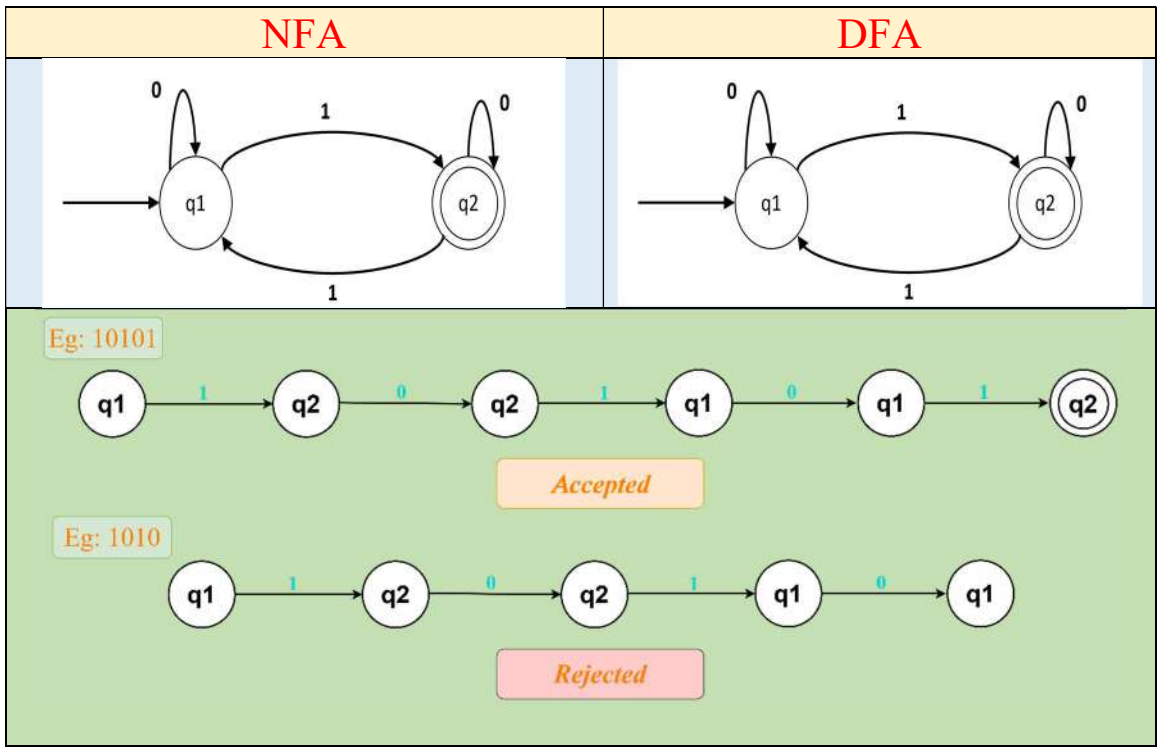
$\{a_1,a_2,a_4\}, \{a_1,a_3,a_4\}, \{a_2,a_3,a_4\}, \{a_1,a_2,a_3, a_4\}\}$

Practice NFA

Problem 1:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **containing** an **odd number of "1s"**
- **Or L(M) = {w: w is a binary string containing an odd number of "1s"}.**

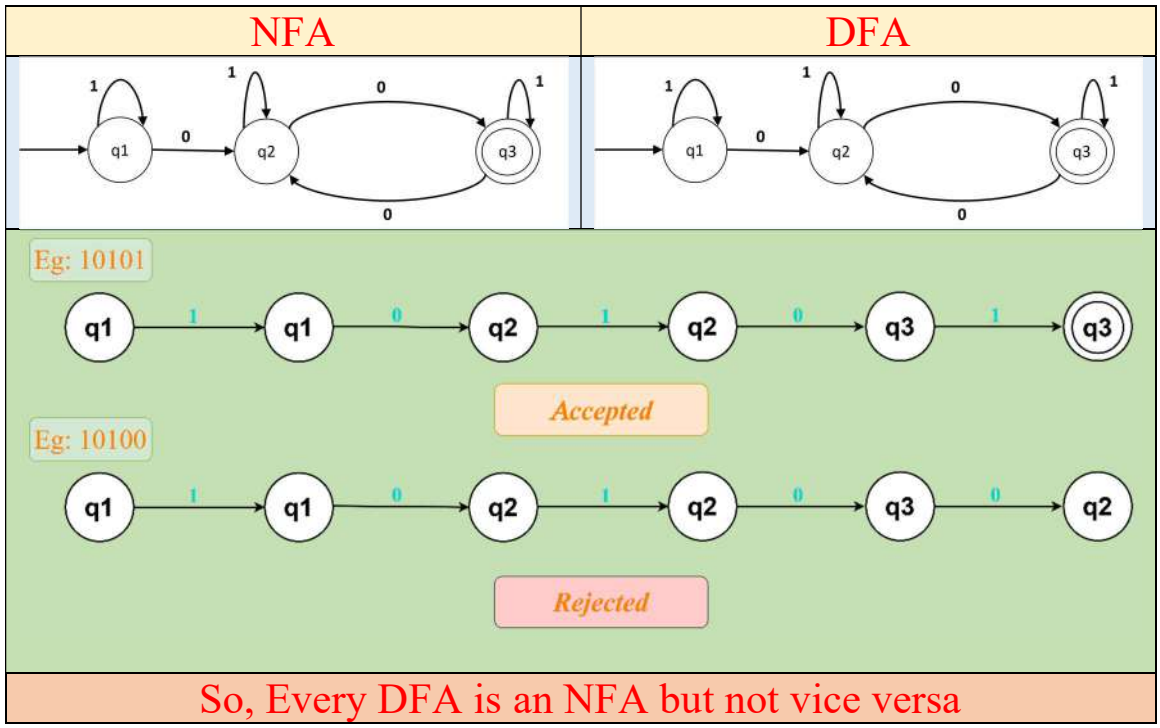
➔ **Is Given,**
∑ = {0,1},
L(M1) = {1, 01, 10, 101, 1101, …}



Problem 2:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **containing** an **even number of "0s"**
- **Or L(M) = {w: w is a binary string containing even number of "0s"}.**

➔ **Is Given,**
∑ = {0,1},
L(M1) = {00, 001, 100, 1001, 11010, …}



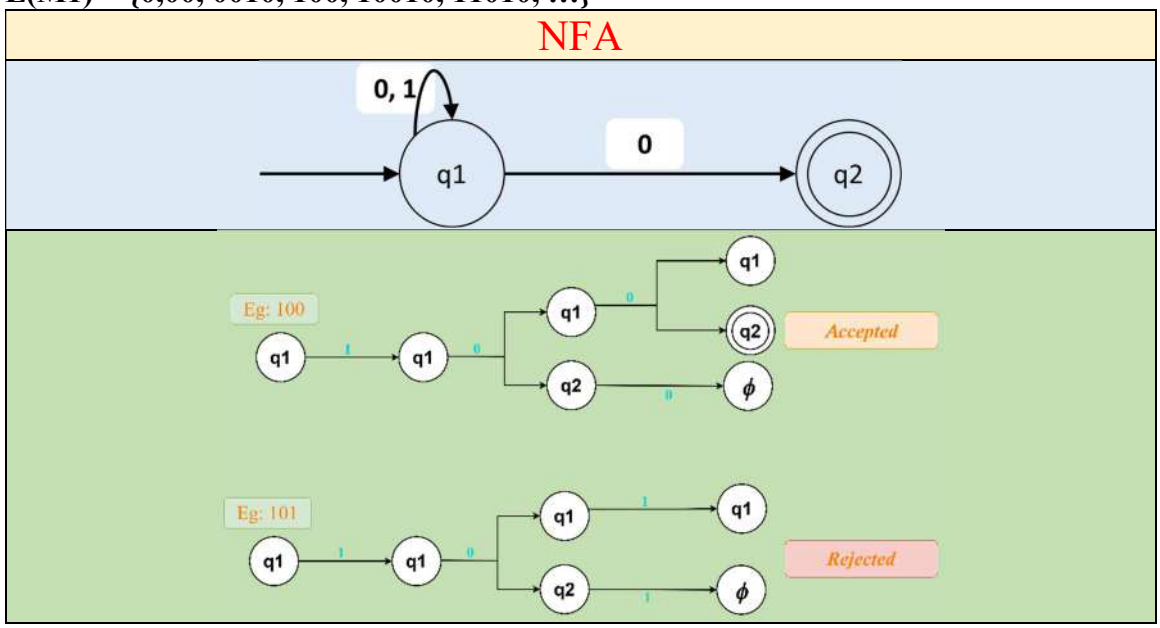
Problem 3:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **ends with a "0"**
- **Or L(M) ={w| w ends with a 0}.**

➔ **Is Given,**

∑ = {0,1},

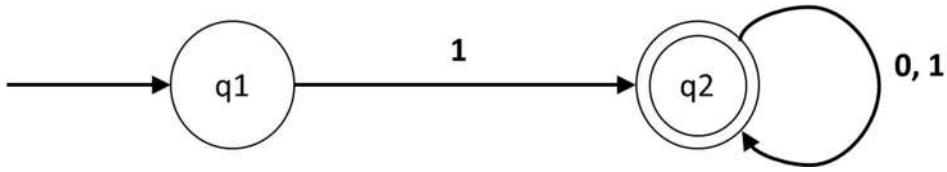
L(M1) = {0,00, 0010, 100, 10010, 11010, …}



Problem 4:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **begins with a "1"**
- **Or L(M) = {w| w begins with a 1}.**

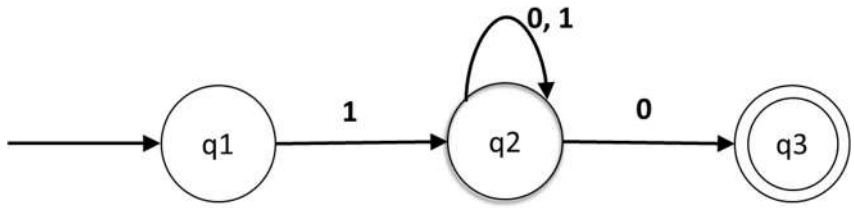
➔ **Is Given,**
∑ = {0,1},
L(M1) = {1, 10010, 100, 10010, 11010, …}



Problem 5:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **begins with a 1 and ends with a 0.**
- **Or L(M) ={w| w begins with a 1 and ends with a 0}.**

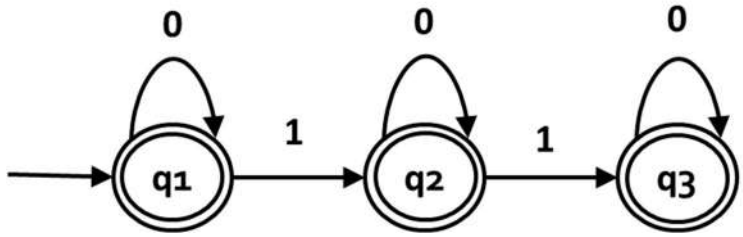
➔ **Is Given,**
∑ = {0,1},
L(M1) = {10, 10010, 100, 10010, 11010, …}



Problem 6:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **has at most two 1s**
- **Or L(M) = {w| w has at most two 1s}.**

➔ **Is Given,**
∑ = {0,1},
L(M1) = {0, 10, 1100, 1010, 10010, 01010, …}

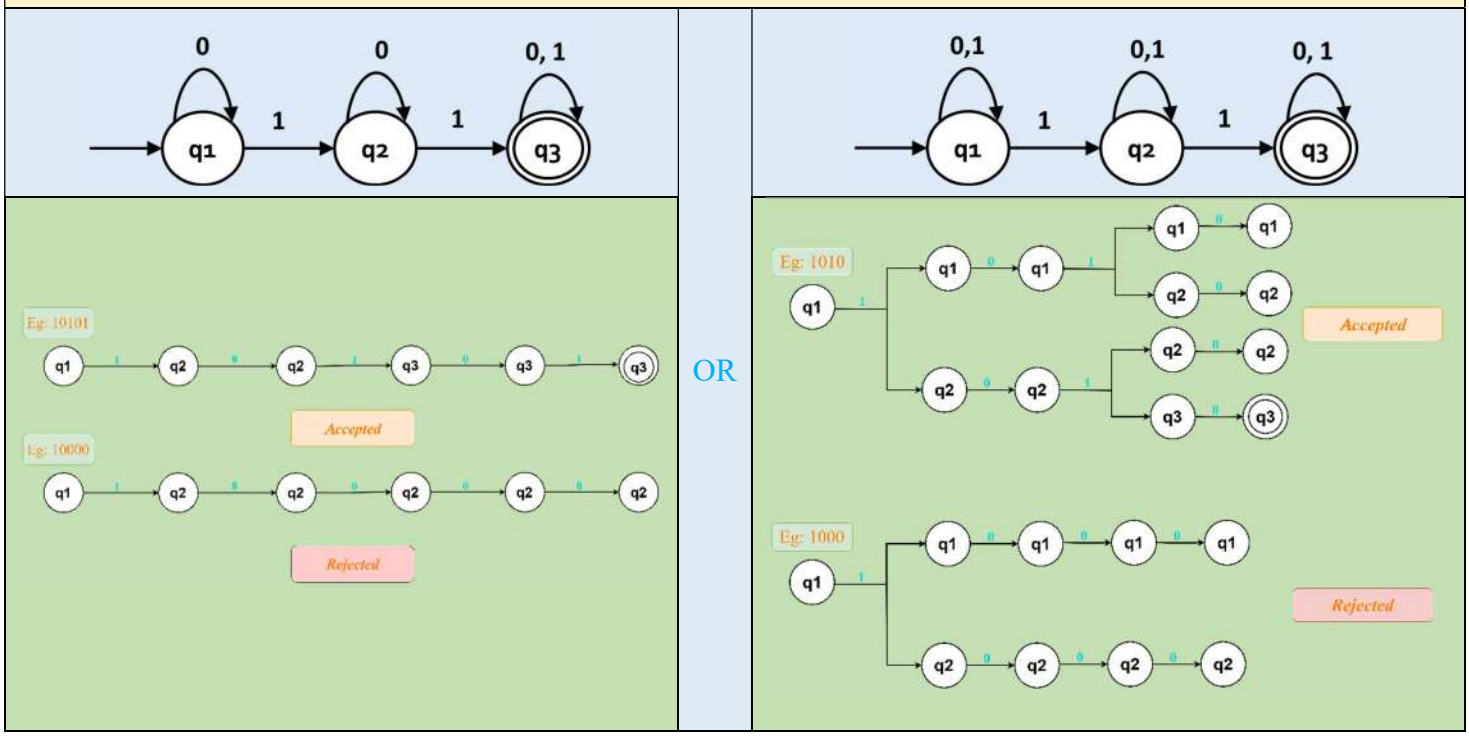


Problem 7:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **has at least two 1s**
- **Or L(M) ={w| w has at least two 1s}.**

➔ **Is Given,**
∑ = {0,1},
L(M1) = {11, 110, 01100, 011010, 10010, 01010, …}

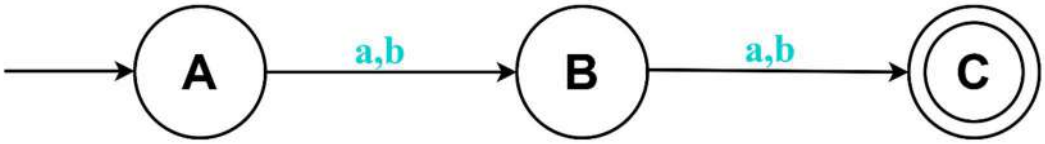
NFA



Problem 8:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, OF Length 2

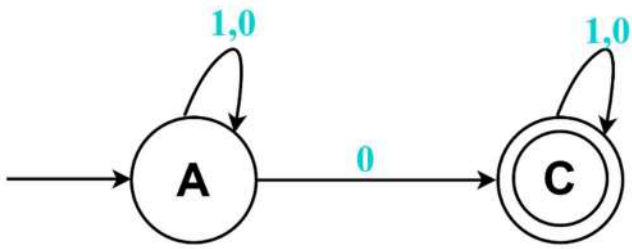
➔ **Is Given,**
∑ = {0,1},
L(M1) = {11, 00, 01,10}



Problem 9:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **Contain "0"**
- **Or L(M) = {w| w Contain "0"}.**

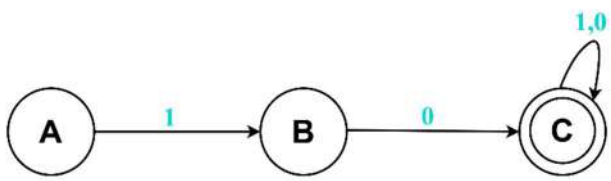
➔ **Is Given,**
∑ = {0,1},
L(M1) = {0, 010, 011, 10000, …}



Problem 10:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **begins with "10"**
- **Or L(M) = {w| begins with "10"}.**

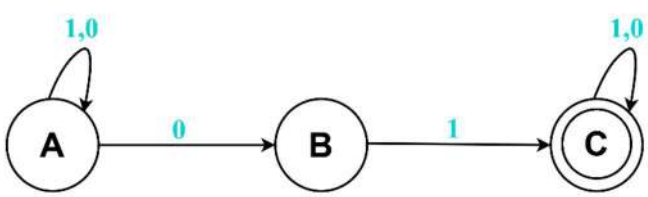
➔ **Is Given,**
∑ = {0,1},
L(M1) = {10, 1011, 101, 10000, …}



Problem 11:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **Contain "01"**
- **Or L(M) = {w| w Contain "01"}.**

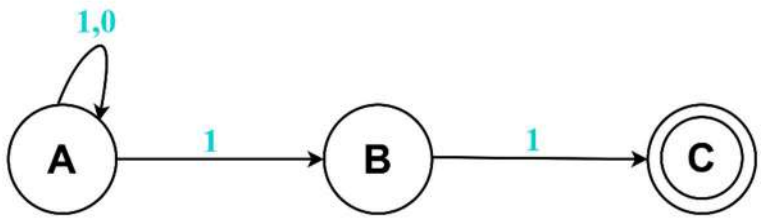
➔ **Is Given,**
∑ = {0,1},
L(M1) = {01, 010, 011, 100001, …}



Problem 12:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **ends with "11"**
- **Or L(M) = {w| w ends with "11"}.**

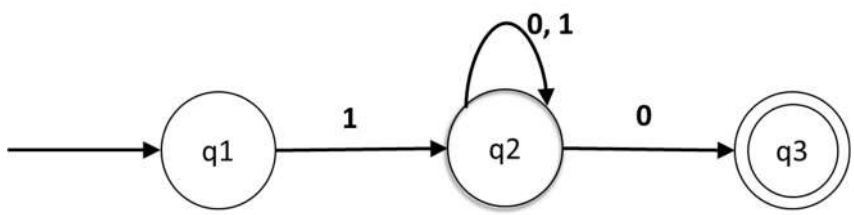
➔ **Is Given,**
∑ = {0,1},
L(M1) = {11, 011, 0111, 1000011, …}



Problem 13:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **begins with a 1 and ends with a 0.**
- **Or L(M) ={w| w begins with a 1 and ends with a 0}.**

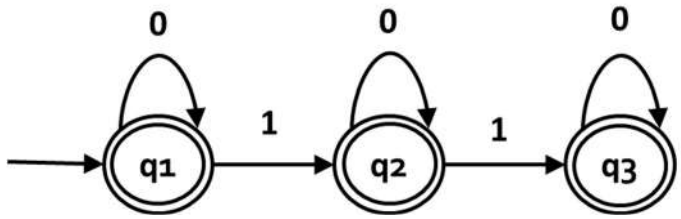
➔ **Is Given,**
∑ = {0,1},
L(M1) = {10, 10010, 100, 10010, 11010, …}



Problem 14:

- Construct a NFA which accepts set of all string over the ∑ = {0,1}, where a binary string **has at most two 1s**
- **Or L(M) = {w| w has at most two 1s}.**

➔ **Is Given,**
∑ = {0,1},
L(M1) = {0, 10, 1100, 1010, 10010, 01010, …}



REGULAR LANGUAGES

Closure

Definition: A collection of objects is *closed* under an operation if applying that operation to its members results in an object still within the collection.

Theorem: The class of **regular languages** is closed under the following operations:

- **Regular operations:** Union, Concatenation, Star
- **Other operations:** Complement, Intersection, Reverse

Regular Languages and Operations

➔ A language is **regular** if a finite automaton recognizes it.

Three Basic Regular Operations:

1. **Union (A ∪ B)** → Combines all strings from both sets.
   - Example: A = {good, bad}, B = {boy, girl}
   - **A ∪ B** = {good, bad, boy, girl}
   - $A \cup B = \{x : x \in A \text{ or } x \in B\}$
2. **Concatenation (A • B)** → Attaches each string from A to a string from B.
   - **A • B** = {goodboy, goodgirl, badboy, badgirl}
   - $\{xy : x \in A \text{ and } y \in B\}$
3. **Star** $(A)^*$ → Forms new strings by repeating elements of A any number of times (including 0).
   - **A*** = {ε, good, bad, goodgood, goodbad, badgood, badbad, …}
   - **ε (empty string) is always included.**
   - $\{x_1x_2\ldots x_k : k \geq 0 \text{ and each } x_i \in A\}$

Regular Language closed under Union in DFA

➔ We construct **M** that recognizes $\mathbf{A_1 \cup A_2}$ using $\mathbf{M_1}$ and $\mathbf{M_2}$:

- **Given:**
  - $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognizes A_1
  - $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognizes A_2
- **Constructing M:**
  - **States:** $Q = Q_1 \times Q_2$ (all combinations of states from M_1 and M_2)
  - **Alphabet:** $\Sigma = \Sigma_1 \cup \Sigma_2$ (assumed same for simplicity)
  - **Transition function:**
    - $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
  - **Start state:** $q_0 = (q_1, q_2)$
  - **Final states:** $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$ (if either machine reaches an accept state)

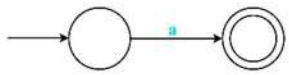
Thus, **M** recognizes $\mathbf{A_1 \cup A_2}$ by simulating both M_1 and M_2 in parallel.

Closure under Concatenation

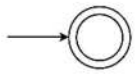
- **Problem:** M must determine where to split the input for M_1 and M_2.
- **Solution:** Use a **nondeterministic automaton (NFA)** to guess the split point.
- **Construction of $M = (Q, \Sigma, \delta, q_0, F)$:**
  - **States:** $Q = Q_1 \times Q_2 \cup \{q_0\}$
  - **Start State:** q_0 (new initial state)
  - **Transition Function:**
    - Simulate M_1 on the input.
    - At any step, nondeterministically transition to M_2 if M_1 reaches an accepting state.
    - Simulate M_2 from that point onward.
  - **Final States:** $F = \{(q_1, q_2) \mid q_1 \in F_1 \text{ and } q_2 \in F_2\}$
- **Key Idea:** M nondeterministically decides where M_1 ends and M_2 begins, ensuring acceptance if such a split exists.

Formal Definition of Regular Language

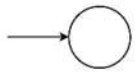
1. $R = a \quad ; \; a \in \sum$



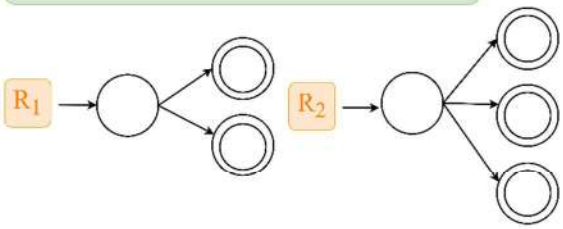
2. $R = \epsilon$; initial state is the final state

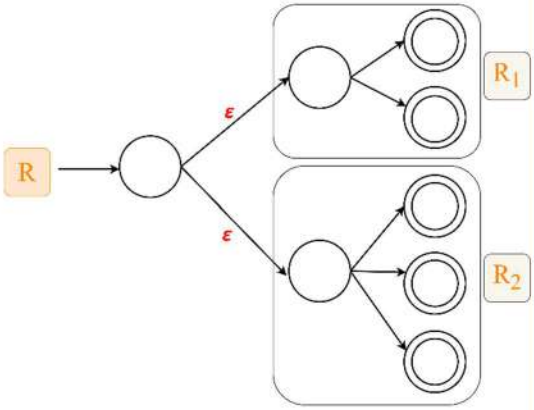


3. $R = \phi$

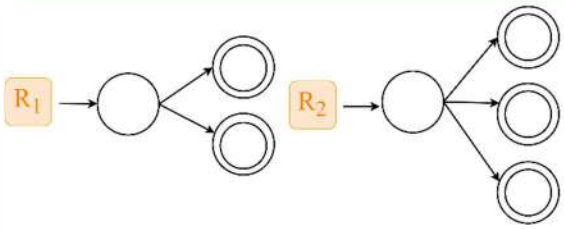


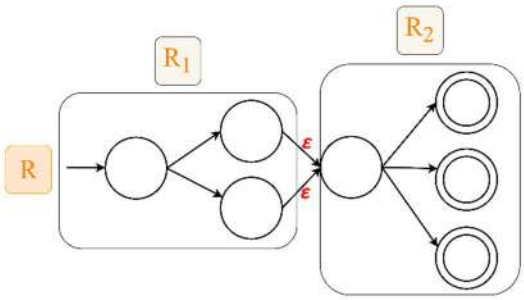
4. $R = R_1 \cup R_2$ **or** $R = R_1 | R_2$ **or** $R = R_1$ or R_2



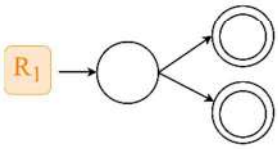


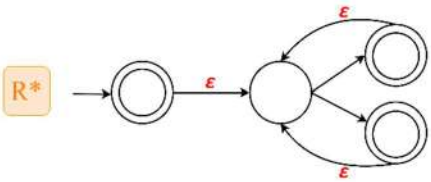
5. $R = R_1 . R_2$ **or** $R = R_1 R_2$ **or** $R = R_1$ Followed by R_2





6. $R^* = R_1^*$



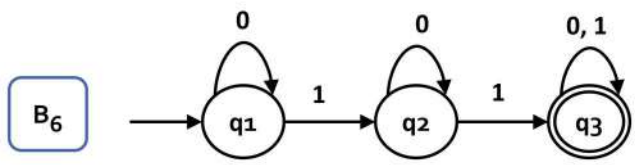


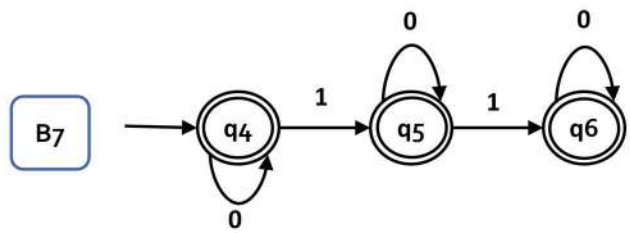
Practice Problem

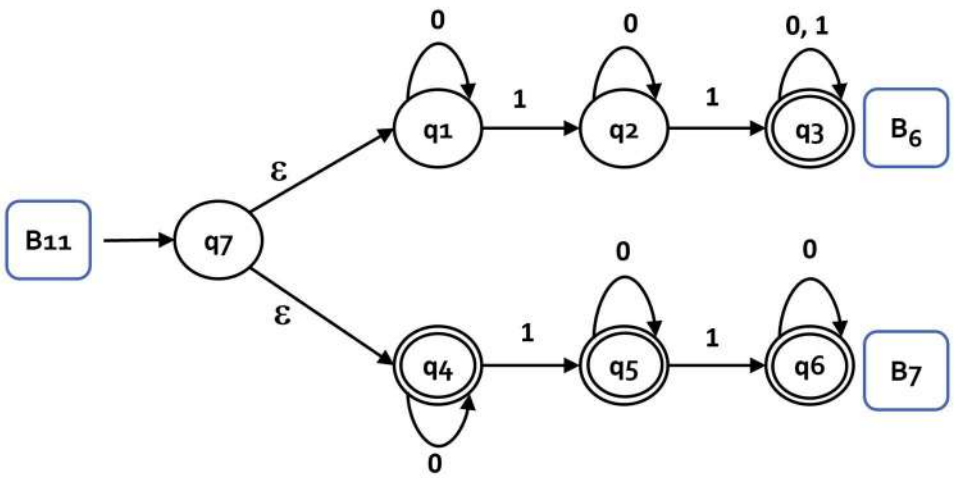
Example 1:

if B_6 = {w| w has at least two 1s}.
B7 = {w| w has at most two 1s}.
B11 = {w| w has at least two 1s **or** w has at most two 1s}.
Where, $\sum = \{0,1\}$
Then Draw the NFA machine.

➔ **Is Given,**
$\sum = \{0,1\}$,
$L(B_6)$ = {1, 01, 10, 1100, 1010, 10010, 01010, …}
$L(B_7)$ = {1, 01, 10, 1100, 1010, 10010, 01010, …}
$L(B_{11}) = (L(B_6) \cup L(B_7))$



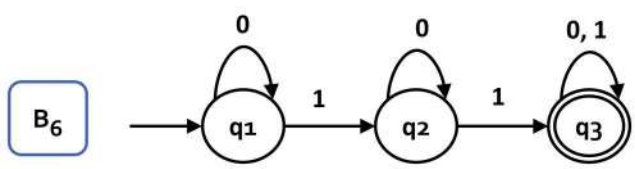


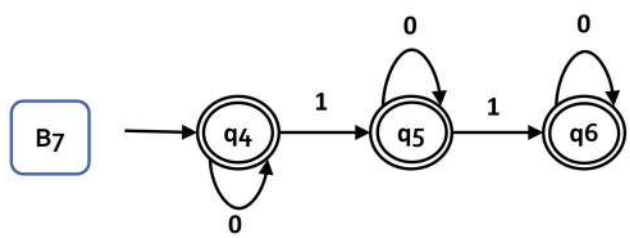


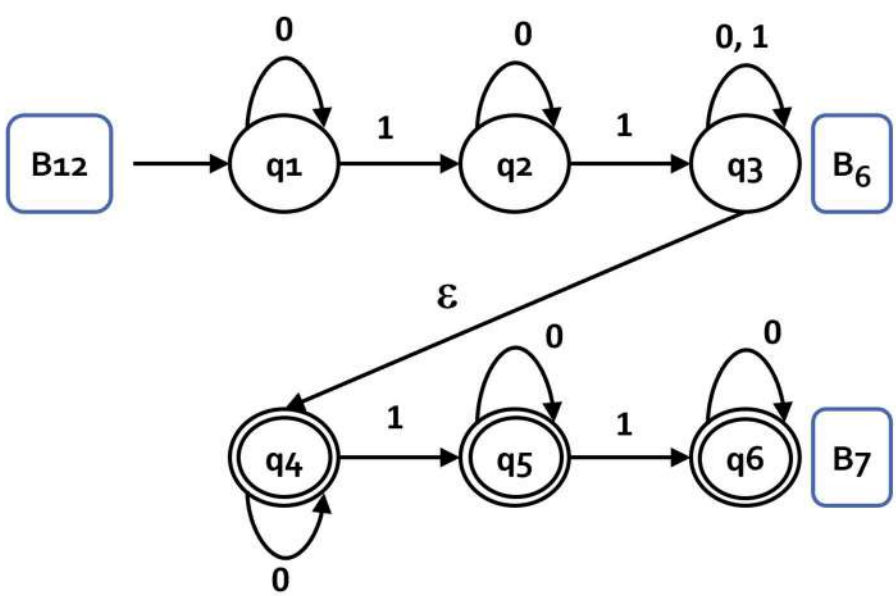
Example 2:

if B6 = {w| w has at least two 1s}.
B7 = {w| w has at most two 1s}.
B12 = {w| w has at least two 1s **followed by** w has at most two 1s}.
Where, $\sum = \{0,1\}$
Then Draw the NFA machine.

➔ **Is Given,**
$\sum = \{0,1\}$,
$L(B_6) = \{1, 01, 10, 1100, 1010, 10010, 01010, \ldots\}$
$L(B_7) = \{1, 01, 10, 1100, 1010, 10010, 01010, \ldots\}$
$L(B_{12}) = (L(B_6) \circ L(B_7))$







☙ Example 3:

if NFA B10 = {w| w starts with a 1 and has even length or
w starts with a 0 and has odd length}.

Find B10* for ∑ = {0,1}

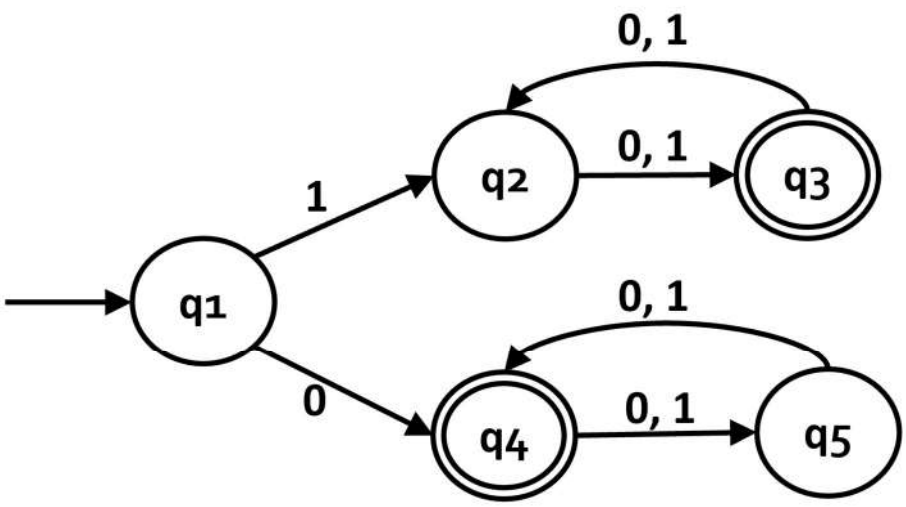
➔ **Is Given,**

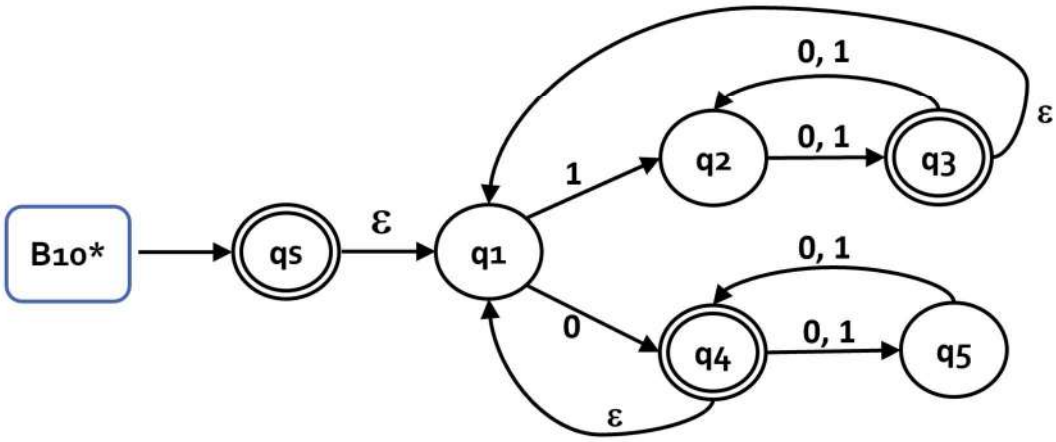
∑ = {0,1},

L(M) = {11, 10, 1000, 1100, 1010, 101110, 1100, …}

L(M) = {0, 010, 01110, 01100, 01010, 00010, 01010, …}

L(B_{10})* = (L(B_6) or L(B_7))*





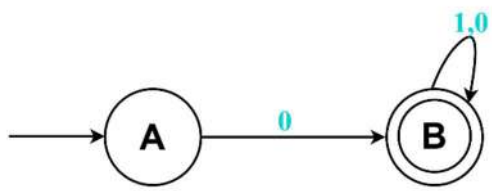
CONVERSION OF NFA TO DFA

Conversion of NFA to DFA

- Every DFA is an NFA, but not vice versa, But there is an **equivalent** DFA for every NFA

Example1: Construct a NFA which accepts set of all string over the $\sum = \{0,1\}$, where a binary string **Start with a "0".** Construct the **equivalent** DFA.

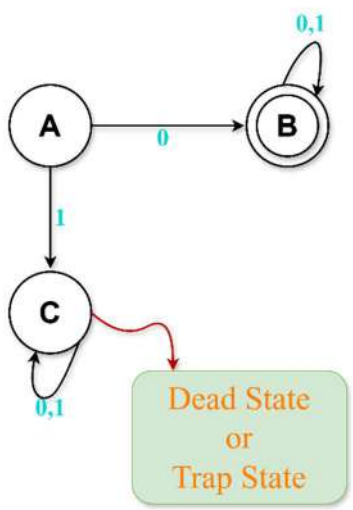
Step1: Construct the NFA



Step2: Write the transition table of **NFA**

δ	0	1
→ A	{B}	ϕ
B	{B}	{B}

Step4: Now, Construct the DFA from T.table



Step3: Convert this NFA transition table to its equivalent **DFA** transition table.

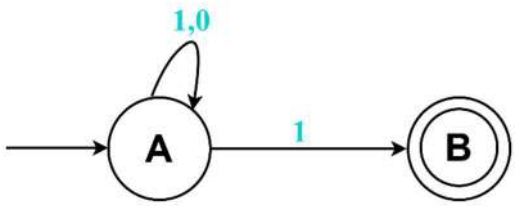
δ	0	1
→ A	{B}	{C}
B	{B}	{B}
C	{C}	{C}

Dead State
or
Trap State

Practice Problem

Problem1: Construct a NFA which accepts set of all string over the $\sum = \{0,1\}$, where a binary string **End with a "1".** Also Construct the **equivalent** DFA.

Step1: Construct the NFA

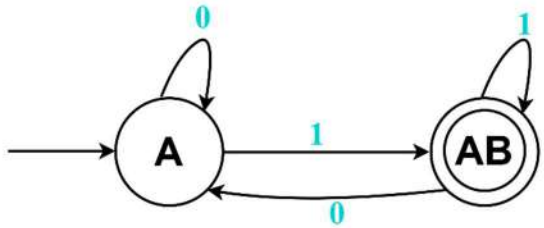


Step2: Write the transition table of **NFA**

δ	0	1
→ A	{A}	{A, B}
B	ϕ	ϕ

Explanations

Step4: Now, Construct the DFA from T.table



Step3: Convert this NFA transition table to its equivalent **DFA** transition table.

δ	0	1
→ A	{A}	{AB}
AB	{A}	{AB}

AB = new Single State

$A \xrightarrow{1} (AUB) = AB$

$A \xrightarrow{0} (AUB) = A$

Problem2: Find the equivalent DFA for the NFA given by M= [{A, B, C}, (a,b), δ, A, {C}] where δ is given by:

δ	a	b
A	{A, B}	{C}
B	{A}	{B}
C	-	{A, B}

➔ Here,

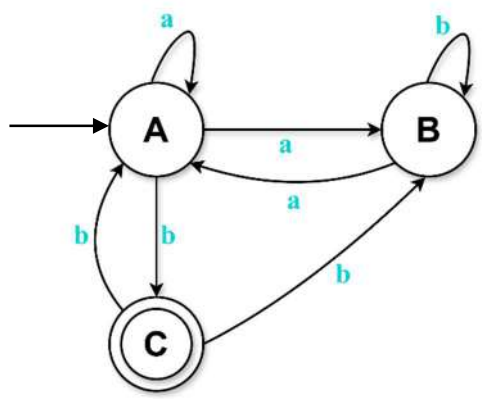
»$Q = \{A, B, C\}$,

»$\Sigma = \{a, b\}$,

»$q_0 = A$,

»$F = \{C\}$,

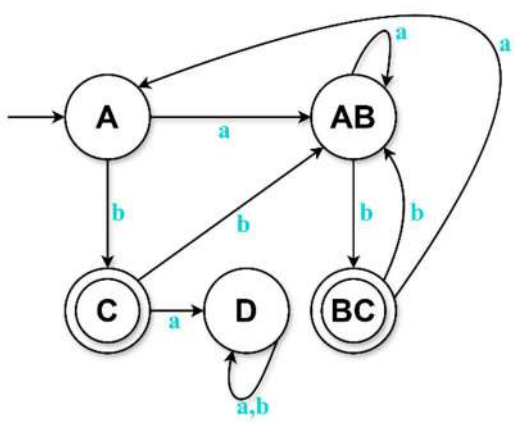
Step1: Construct the NFA



Step2: Write the transition table of **NFA**

δ	a	b
→ A	{A, B}	{C}
B	{A}	{B}
C	ϕ	{A, B}

Step4: Now, Construct the DFA from T.table

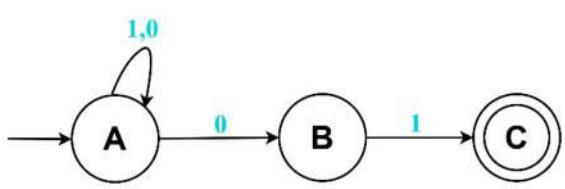


Step3: Convert this NFA transition table to its equivalent **DFA** transition table.

δ	a	b
→ A	{AB}	{C}
AB	{AB}	{BC}
BC	{A}	{AB}
C	{D}	{AB}
D	{D}	{D}

Problem3: Construct a NFA which accepts set of all string over the $\sum = \{0,1\}$, where a binary string **Ends with a "01".** Also Construct the **equivalent** DFA.

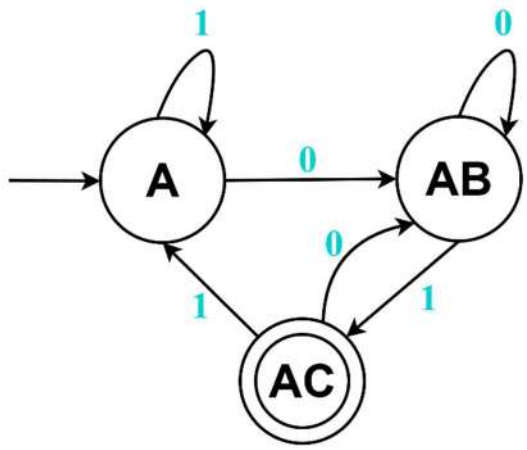
Step1: Construct the NFA



Step2: Write the transition table of **NFA**

δ	0	1
→ A	{A, B}	{A}
B	ϕ	{C}
C	ϕ	ϕ

Step4: Now, Construct the DFA from T.table

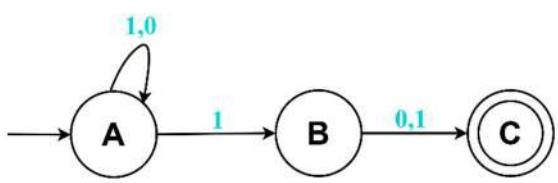


Step3: Convert this NFA transition table to its equivalent **DFA** transition table.

δ	0	1
→A	{AB}	{A}
AB	{AB}	{AC}
AC	{AB}	{A}

Problem4: Construct a NFA which accepts set of all string over the $\sum = \{0,1\}$, where a binary string **Second last symbol always "1".** Also Construct the **equivalent** DFA.

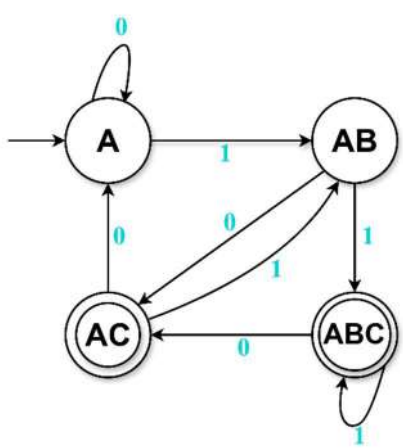
Step1: Construct the NFA



Step2: Write the transition table of **NFA**

δ	0	1
→A	{A}	{A, B}
B	{C}	{C}
C	ϕ	ϕ

Step4: Now, Construct the DFA from T.table



Step3: Convert this NFA transition table to its equivalent **DFA** transition table.

δ	0	1
→A	{A}	{AB}
AB	{AC}	{ABC}
ABC	{AC}	{ABC}
AC	{A}	{AB}

Epsilon (ε) NFA to DFA conversion

Example1: Find the equivalent DFA for the NFA given by M= [$\{b_1, b_2, b_3\}$, (0,1), δ, b_1, $\{b_1\}$] where δ is given by:

δ	0	1	ε
b_1	ϕ	$\{b_2\}$	$\{b_3\}$
b_2	$\{b_2, b_3\}$	$\{b_3\}$	ϕ
b_3	$\{b_1\}$	ϕ	ϕ

➔ Here,

»$Q = \{b_1, b_2, b_3\}$,

»$\Sigma = \{0, 1\}$,

»$q_0 = b_1$,

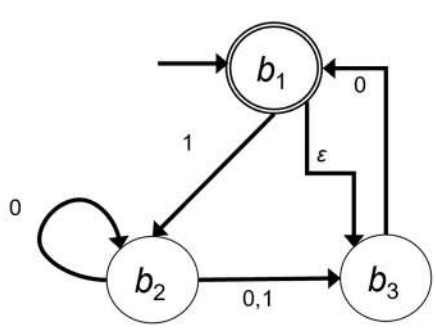
»$F = \{b_1\}$,

➔ For Equivalent DFA:

»$Q = P(Q) = P(\{b_1, b_2, b_3\})$

»$\mathcal{P}(Q) = \{ \phi, \{b_1\}, \{b_2\}, \{b_3\}, \{b_1, b_2\}, \{b_1, b_3\}, \{b_2, b_3\}, \{b_1, b_2, b_3\}\}$

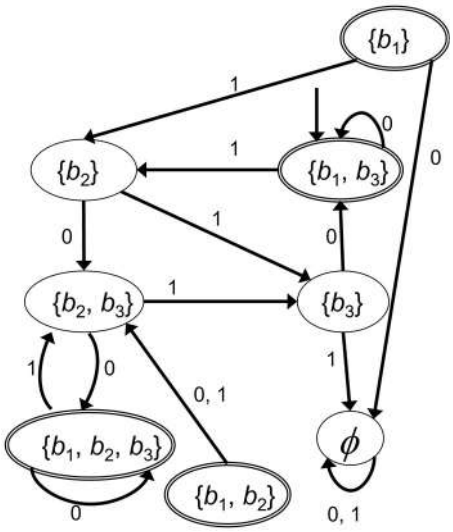
Step1: Construct the NFA



Step2: Write the transition table of **NFA**

δ	0	1	ε
b_1	ϕ	$\{b_2\}$	$\{b_3\}$
b_2	$\{b_2, b_3\}$	$\{b_3\}$	ϕ
b_3	$\{b_1\}$	ϕ	ϕ

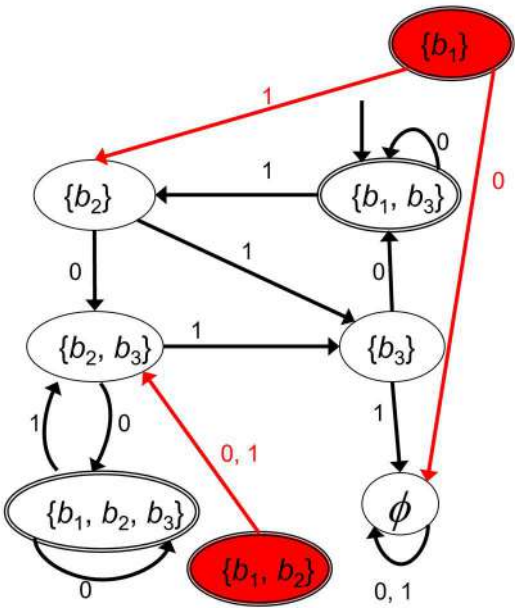
Step4: Now, Construct the DFA from T.table



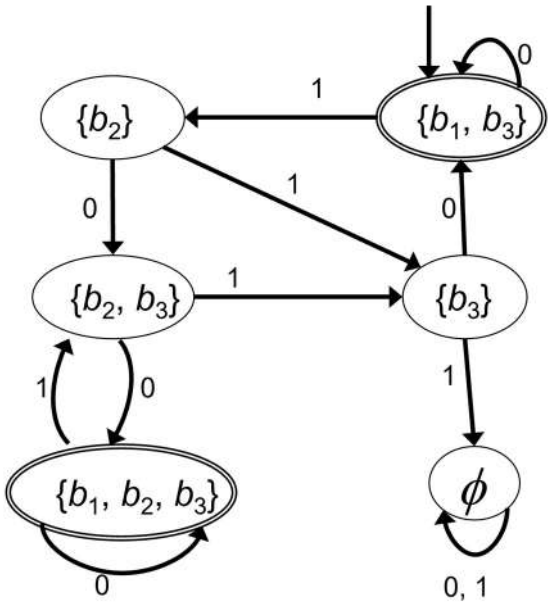
Step3: Convert this NFA transition table to its equivalent **DFA** transition table.

δ	0	1
ϕ	ϕ	ϕ
b_1	ϕ	$\{b_2\}$
b_2	$\{b_2, b_3\}$	$\{b_3\}$
b_3	$\{b_1, b_3\}$	ϕ
$\{b_1, b_2\}$	$\{b_2, b_3\}$	$\{b_2, b_3\}$
$\{b_1, b_3\}$	$\{b_1, b_3\}$	$\{b_2\}$
$\{b_2, b_3\}$	$\{b_1, b_2, b_3\}$	$\{b_3\}$
$\{b_1, b_2, b_3\}$	$\{b_1, b_2, b_3\}$	$\{b_2, b_3\}$

Step5: Remove the states with no incoming arrows.

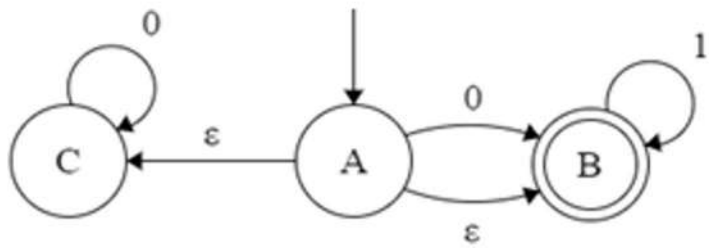


Step6: After Remove the states with no incoming arrows.



Practice Problem

Problem1: Convert the following NFA into equivalent DFA using the formal procedure.



➔ Here,

»$Q = \{A, B, C\}$,

»$\Sigma = \{0, 1\}$,

»$q_0 = A$,

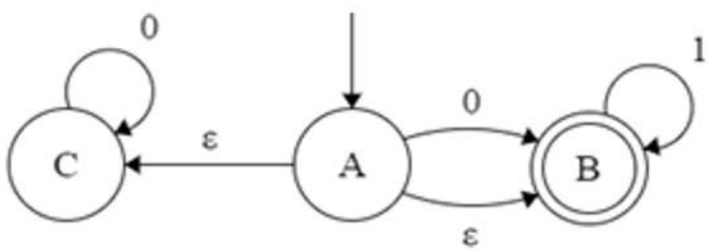
»$F = \{B\}$,

➔ For Equivalent DFA:

»$Q = P(Q) = P(\{A, B, C\})$

»$\mathcal{P}(Q) = \{ \phi, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}$

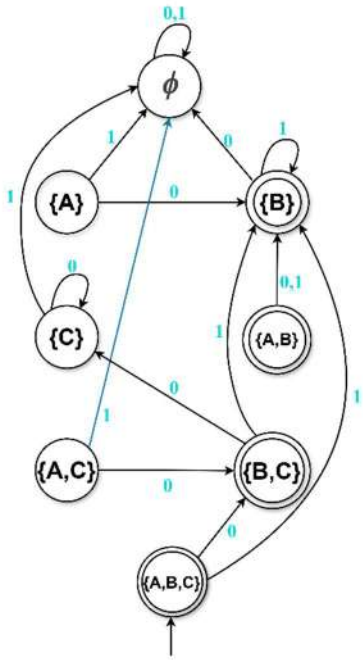
Step1: Construct the NFA



Step2: Write the transition table of **NFA**

δ	0	1	ε
A	{B}	ϕ	$\{B, C\}$
B	ϕ	{B}	ϕ
C	{C}	ϕ	ϕ

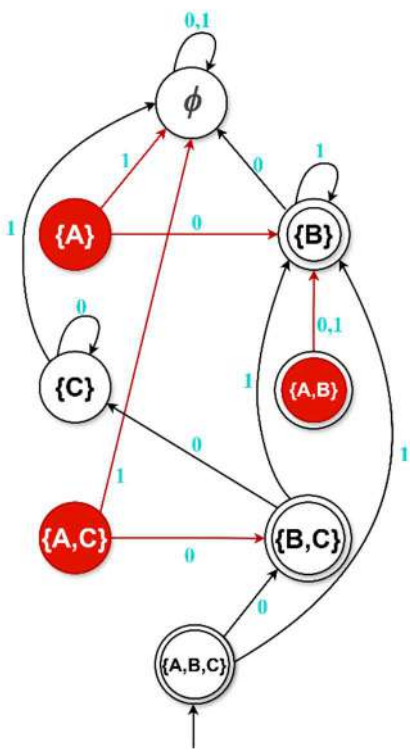
Step4: Now, Construct the DFA from T.table



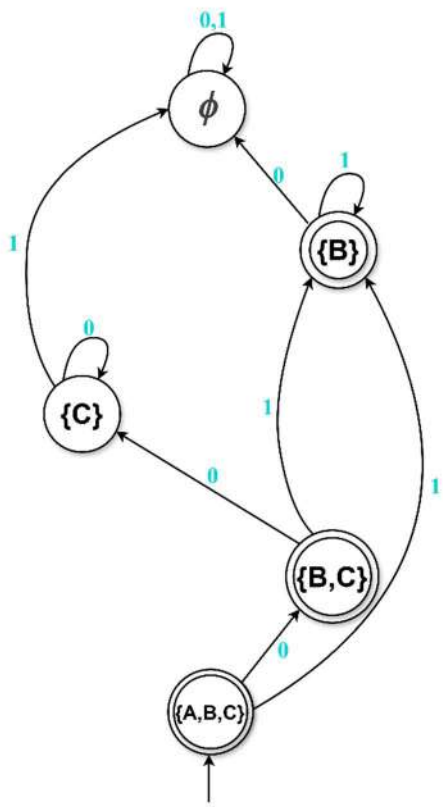
Step3: Convert this NFA transition table to its equivalent **DFA** transition table.

δ	0	1
ϕ	ϕ	ϕ
{A}	{B}	ϕ
{B}	ϕ	{B}
{C}	{C}	ϕ
{A, B}	{B}	{B}
{A, C}	{B, C}	ϕ
{B, C}	{C}	{B}
{A, B, C}	{B, C}	{B}

Step5: Remove the states with no incoming arrows.

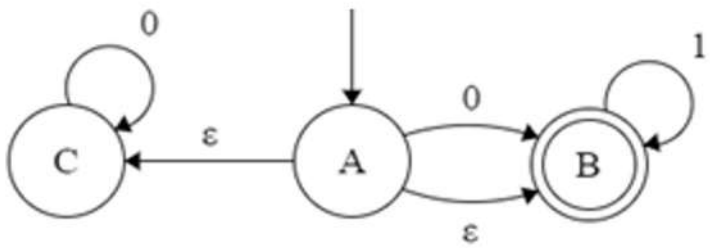


Step6: After Remove the states with no incoming arrows.



"An Alternative Approach to Solving It"

Step1: Construct the NFA



Step2: Write the transition table of **NFA**

δ	0	1	ε
A	{B}	ϕ	{*B, C*}
B	ϕ	{B}	ϕ
C	{C}	ϕ	ϕ

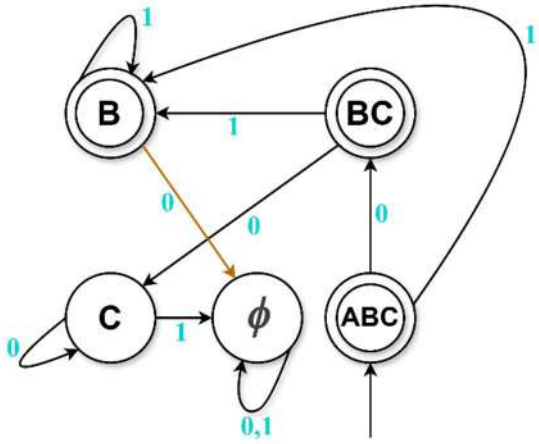
Step3: Find the **Epsilon (ε) Closure** of all the state

- **ε-Closure for {A} = {A, B, C}**
- **ε-Closure for {B} = {B}**
- **ε-Closure for {C} = {C}**

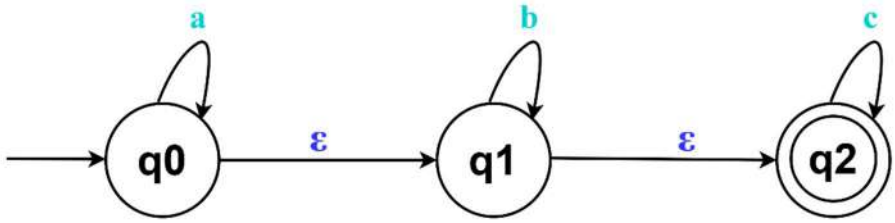
Step4: Convert this NFA transition table to its equivalent **DFA** transition table.

δ	0	1
{*A, B, C*}	{B, C}	{B}
{*B, C*}	{C}	{B}
{*B*}	ϕ	{B}
{*C*}	{C}	ϕ
ϕ	ϕ	ϕ

Step5: Now, Construct the DFA from transition table.



Problem2: Convert the following NFA into equivalent DFA using the formal procedure.



➔ Here,

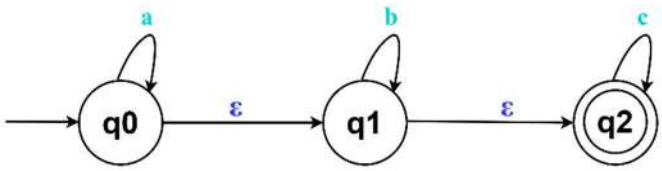
»$Q = \{q_0, q_1, q_2\}$,

»$\Sigma = \{a,b\}$,

»$q_0 = q_0$,

»$F = \{q_2\}$,

Step1: Construct the NFA



Step2: Write the transition table of **NFA**

δ	a	b	c	ε
q_0	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

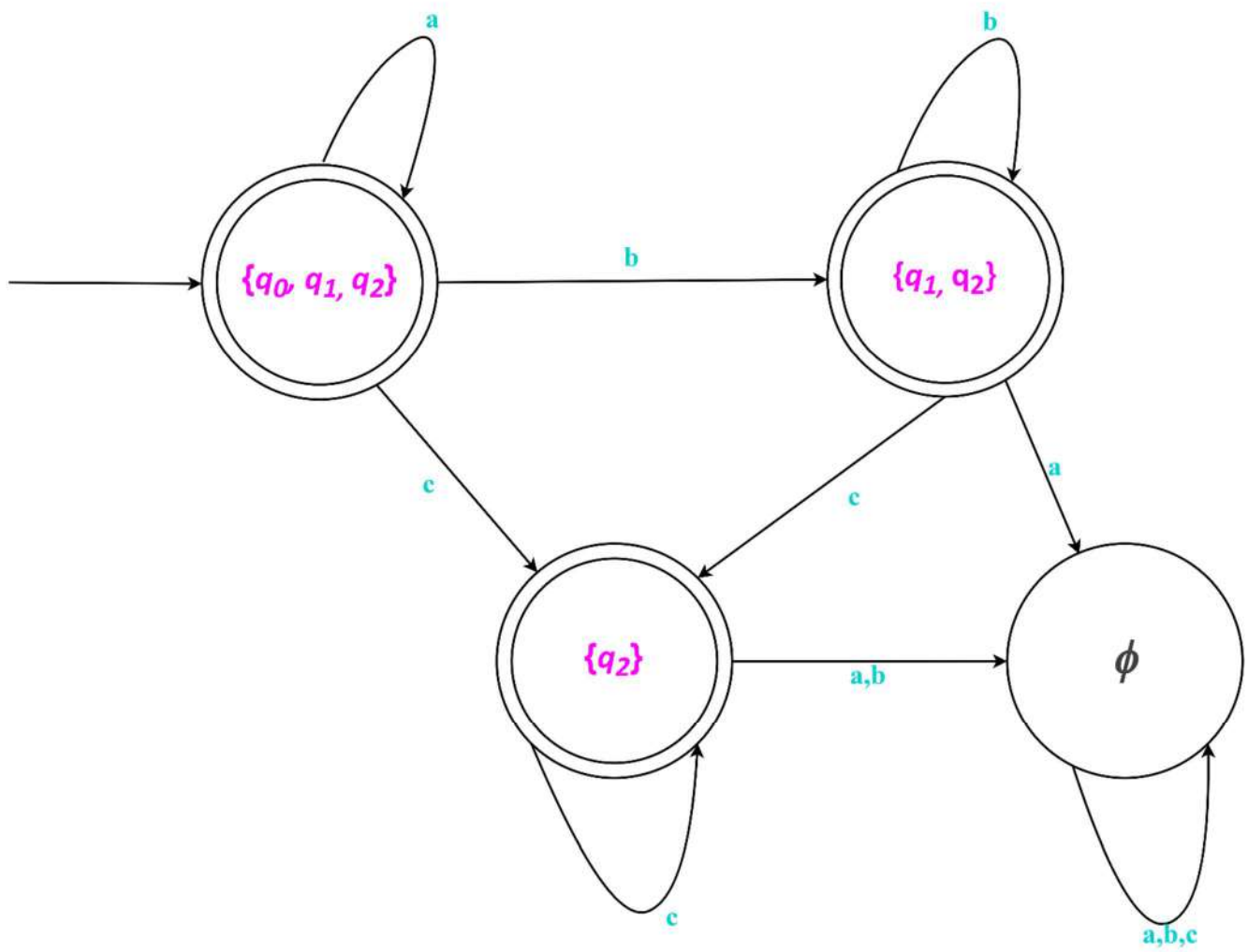
Step3: Find the **Epsilon (ε) Closure** of all the state

- ε-Closure for $\{q_0\} = \{q_0, q_1, q_2\}$
- ε-Closure for $\{q_1\} = \{q_1, q_2\}$
- ε-Closure for $\{q_2\} = \{q_2\}$

Step4: Convert this NFA transition table to its equivalent **DFA** transition table.

δ	a	b	c
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_1, q_2\}$	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_2\}$	ϕ	ϕ	$\{q_2\}$
ϕ	ϕ	ϕ	ϕ

Step5: Now, Construct the DFA from transition table.

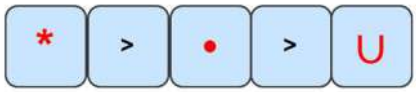


REGULAR EXPRESSION

Regular Expression

≫Regular expressions are used to represent certain sets of strings in an **algebraic** fashion, moving beyond simple English representations of strings.

- **Structure**:
  Built using alphabets and **regular operations**:
  - * (star/**Closure**) – Represents zero or more occurrences of a string.
  - • (Concatenation) – joining two patterns
  - ∪ (Union) – either/or (can also be written as |)
- **Precedence Order**:



≫Examples:

1. **(0∪1)0***
   - Means: Starts with 0 or 1, followed by any number of 0s
   - Language: `{w | w starts with 0 or 1, followed by 0s}`
2. **(0∪1)***
   - Means: Any combination of 0s and 1s
   - Language: `{all binary strings}`
3. **(0∪1)1**
   - **Means**: Either 0 or 1, followed by 1
   - Language: `{w | w ends with 1, and starts with 0 or 1}`

Designing Regular Expressions

🧠 Example1: Write the Regular Expression of the following languages, where ∑= {a,b},
"Language Accepting **Strings of Length Exactly Two**"

Step 1: Understand the Alphabet	**Step 2:** Understand the Language Requirement
We are working with: Σ = {a, b} This means only the characters a and b can appear in our strings.	≫The language accepts only strings of **length exactly two**. This means strings like: • aa • ab • ba • bb Only these four strings are valid. Any string with **length ≠ 2** is not allowed.
Step 3: List All Valid Strings	**Step 4:** Write the Regular Expression
≫As we saw above, valid strings of length 2 over {a, b} are: • **aa** • **ab** • **ba** • **bb**	∴R = (aa ∪ ab ∪ ba ∪ bb) Or ∴R = (aa | ab | ba | bb) Or ∴R = (aa + ab + ba + bb)

🧠 Example2: Write the Regular Expression of the following languages, where ∑= {a,b},
"which accepts **all strings of length at least two**."

Step 1: Understand the Alphabet	**Step 2:** Understand the Language Requirement
We are working with: Σ = {a, b} This means only the characters a and b can appear in our strings.	≫The language accepts **all strings of length at least two** This means: • The string must be **two or more characters long**. • This includes all strings of length 2, 3, 4, and so on.
Step 3: List All Valid Strings	**Step 4:** Write the Regular Expression
≫ For strings of length at least two: • **Valid examples: aa, ab, ba, bb, aaa, aab, abb, baa, bab, bba, bbb, etc.** • **Invalid examples** (strings of length less than two): a, b	∴R = (a | b)(a | b)(a | b)*

Explanation:
• `(a | b)`: The first character can be either `a` or `b`. • `(a | b)`: The second character can also be either `a` or `b`. • `(a | b)*`: After the second character, any number (including zero) of `a` or `b` characters can follow. This makes the string longer than two characters if needed.

🧠 Example3: Write the Regular Expression of the following languages, where ∑= {a,b}, "which accepts **Strings of Length At Most Two**."

Step 1: Understand the Alphabet	**Step 2:** Understand the Language Requirement
We are working with: Σ = {a, b} This means only the characters `a` and `b` can appear in our strings.	≫The language accepts **all strings of length at Most two.** This means: • Length 0 (the **empty string**, denoted by **ε**) • Length 1 (like `a`, `b`) • Length 2 (like `aa`, `ab`, `ba`, `bb`)
Step 3: List All Valid Strings	**Step 4:** Write the Regular Expression
≫ For strings of length at **Most** two: • **ε** • **a** • **b** • **aa** • **ab** • **ba** • **bb**	∴R = (ε ∪ a ∪ b ∪ aa ∪ ab ∪ ba ∪ bb)

Practice Problem

Problem1: Write the Regular Expression of the following languages, where ∑= {a,b}, "Language Accepting **Strings of Length Exactly Two**"

➔ Here,

»Σ = {a,b},

» L(M) = {aa, ab, ba, bb}

$$\therefore R = (aa \cup ab \cup ba \cup bb)$$

Or

$$\therefore R = (aa \mid ab \mid ba \mid bb)$$

Or

$$\therefore R = (aa + ab + ba + bb)$$

Problem2: Write the Regular Expression of the following languages, where ∑= {a,b}, "which accepts **all strings of length at least two**."

➔ Here,

»Σ = {a,b},

» L(M) = {aa, ab, ba, bb, aaa, aab, abab…, bbaa..,}

$$\therefore R = (a \cup b)\,(b \cup a)\,(a \cup b)^*$$

Problem3: Write the Regular Expression of the following languages, where ∑= {a,b}, "which accepts **all strings of length at Most two**."

➔ Here,

»Σ = {a,b},

» L(M) = {ε, a, b, aa, bb, ab, ba}

$$\therefore R = (\varepsilon \cup a \cup b \cup aa \cup ab \cup ba \cup bb)$$

Problem4: Write the Regular Expression of the following languages, where ∑= {a,b}, "L={w| each a in w is followed by at least two b}"

➔ Here,

»Σ = {a,b},

» L(M) = {abb, b…abb, abbb…, abbabb…, abbbabb…}

$$\therefore R = b^* \text{ abb } b^* \text{ (abb } b^*)^*$$

Let's check whether the string "**abbabb**" matches the regular expression for the language:
1. **Breakdown of** abbabb**:** ○ The string can be grouped as: a b b a b b (Each a is followed by exactly two bs, which satisfies the condition.) 2. **Matching with** b* a b b b* (a b b b*)***:** ○ b* → Matches 0 leading bs (empty). ○ a b b b* → Matches a b b (the first b* matches 0 trailing bs here). ○ (a b b b*)* → Matches the remaining a b b (again, b* matches 0 trailing bs). ○ **Result:** The entire string abbabb is matched.

Problem5: Write the Regular Expression of the following languages, where ∑= {a,b}, "L={w| w contains odd number of b}"

➔ Here,

»Σ = {a,b},

» L(M) = {b, a…b, ba…, bababa…,ba…ba…ba…}

$$\therefore R = a^*b \text{ (}a^*b\ a^*b\ a^*\text{) }^*$$

Let's check whether the string "**ababab**" matches the regular expression for the language:

$$\therefore R = a^*b\ (a^*b\ a^*b\ a^*)^*$$

Attempt to match "abab":

1. **First b:**
   - a* matches a.
   - b matches the first b (count = 1).
   - **Remaining string: abab**
2. **First repetition of (a* b a* b a*)*:**
   - a* matches a.
   - b matches the second b (count = 2).
   - a* matches a.
   - b matches the second b (count = 3).
   - **The remaining part of the group (a* b a* b a*)* is optional (due to *), so the match succeeds.**

Its correct **because it accepts** abab **(and other even-**b **strings).**

Problem5: Write the Regular Expression of the following languages, where ∑= {a,b},
"L={w| w contains ***abbab*** substring}"

➔ Here,

»Σ = {a,b},

» L(M) = {abbab, a..abbab, b..abbab, abbaba…, abbabb…}

$$\therefore R = (a \cup b)^*\ abbab\ (a \cup b)^*$$

Explanation:

1. (a|b)* → Matches any sequence of as and bs (including none) before "abbab".
2. abbab → Matches the exact required substring.
3. (a|b)* → Matches any sequence of as and bs (including none) after "abbab".

Examples:

1. **String "abbab":**
   - (a|b)* matches the empty string (before "abbab").
   - abbab matches the entire string.
   - (a|b)* matches the empty string (after "abbab").
   - ☑ Accepted.

1. **String "abba":**
   - The substring "abbab" is missing.
   - ❌ Rejected.
2. **String "abbabb":**
   - (a|b)* matches the empty string.
   - abbab matches "abbab".
   - (a|b)* matches "b".
   - ☑ Accepted.

Problem6: Write the Regular Expression of the following languages, where ∑= {m,n}

i. "{w | each m in w is followed by at least three n}"
ii. "{w | w has odd length and starts with m}"
iii. "{w | w contains 'mnnmmn' substring}"

(i)

➔ Here,

»**Σ = {m, n},**

» **L(M) = {mnnn, n…mnnn, mnnnn…, mnnmnn… }**

$$\therefore R = n^* \text{ mnnn } n^* \text{ (mnnn } n^*)^*$$

(ii)

➔ Here,

»**Σ = {m, n},**

» **L(M) = {m, mmn, mnn,mnnnm,… }**

$$\therefore R = m((m \cup n)(m \cup n))^*$$

(iii)

➔ Here,

»**Σ = {m, n},**

» **L(M) = { mnnmmn, mmn…mnnmmn, mnnmmn…mnnnm,… }**

$$\therefore R = (m|n)^* \text{ mnnmmn } (m|n)^*$$

Problem7: Give the **Description** of the following Regular Expressions.

Consider $\sum = \{x,y\}$

i. "(x∪y)* xyyxxy"
ii. "x* y x* y x* y y*"
iii. "(x ∪ y). ((x ∪ y). (x ∪ y))*"

(i)

➔(x∪y)* xyyxxy:

- Means: Strings that end with the substring "xyyxxy"..
- Language: {w | w is a string over ∑ = {x, y} that ends with "xyyxxy"}.

(ii)

➔x* y x* y x* y y*:

- Means: Strings containing minimum three 'y's, where the third 'y' may be followed by any number of additional 'y's..
- Language: {w | w is a string over ∑ = {x, y} with at least three 'y's, and the third 'y' is followed by zero or more 'y's}.

(iii)

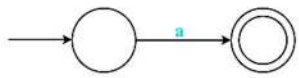
➔(x ∪ y). ((x ∪ y). (x ∪ y))*

- Means: Starts with x or y, and string has odd number of length.
- Language: {w | w string starts with x or y, string has odd number of length}

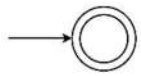
CONVERSION OF RE TO NFA

Formal procedure of Regular Language

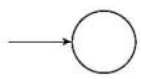
1. $R = a \quad ; \; a \in \Sigma$



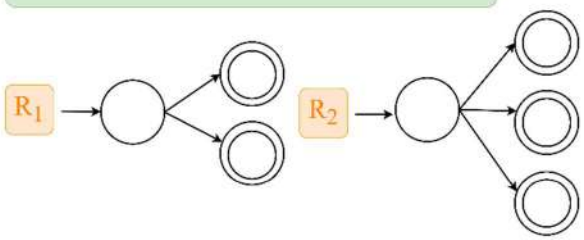
2. $R = \epsilon$; initial state is the final state

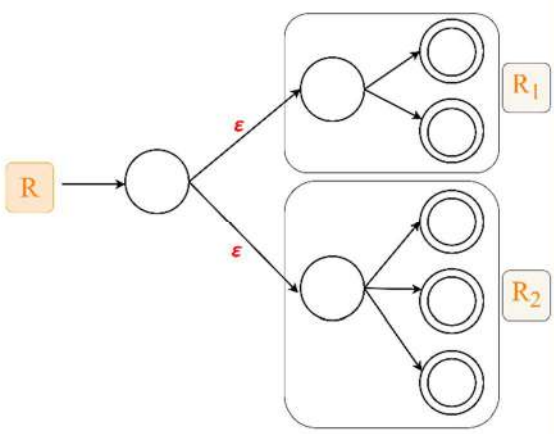


3. $R = \phi$

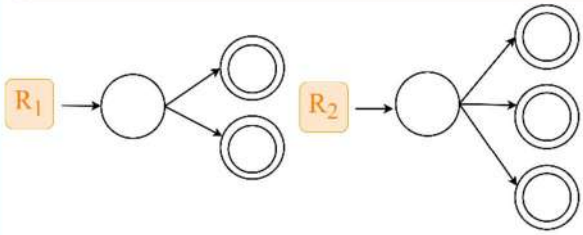


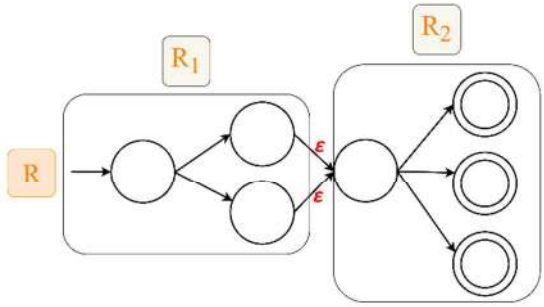
4. $R = R_1 \cup R_2$ or $R = R_1 | R_2$ or $R = R_1$ or R_2



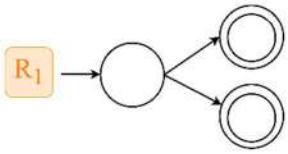


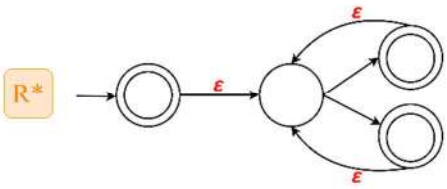
5. $R = R_1 \circ R_2$ or $R = R_1 R_2$ or $R = R_1$ Followed by R_2





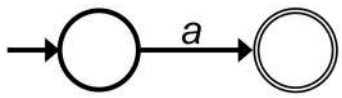
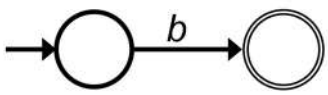
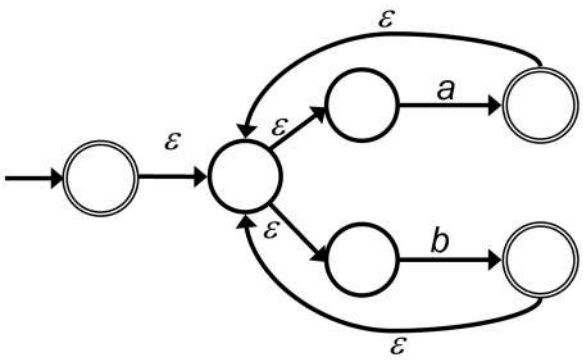
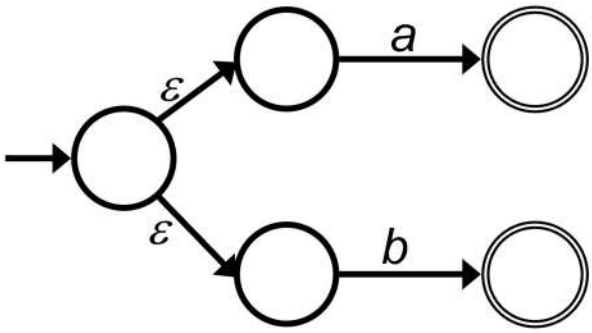
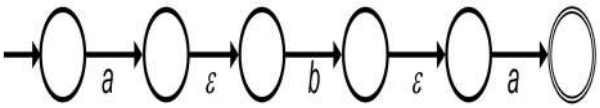
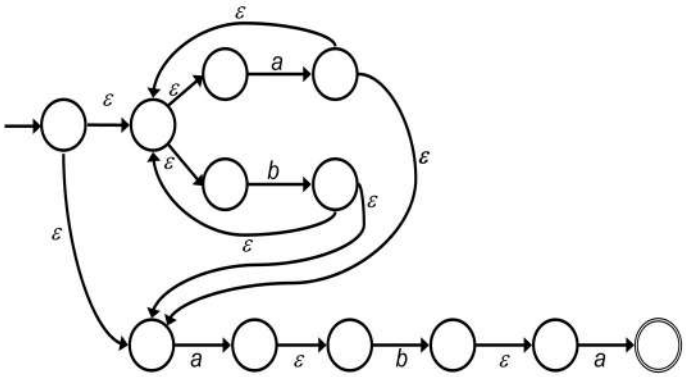
6. $R^* = R_1^*$



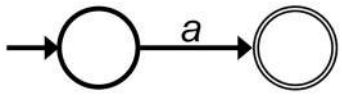
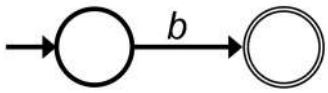
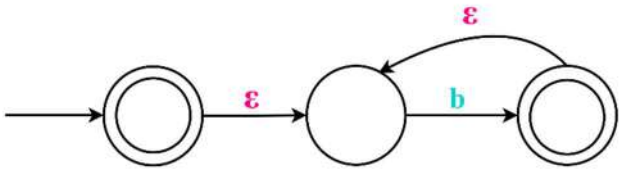
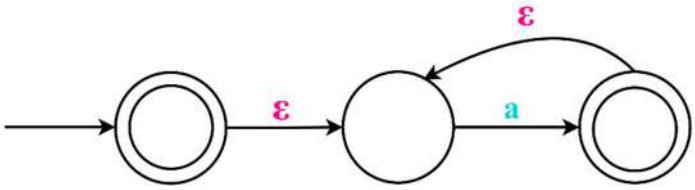
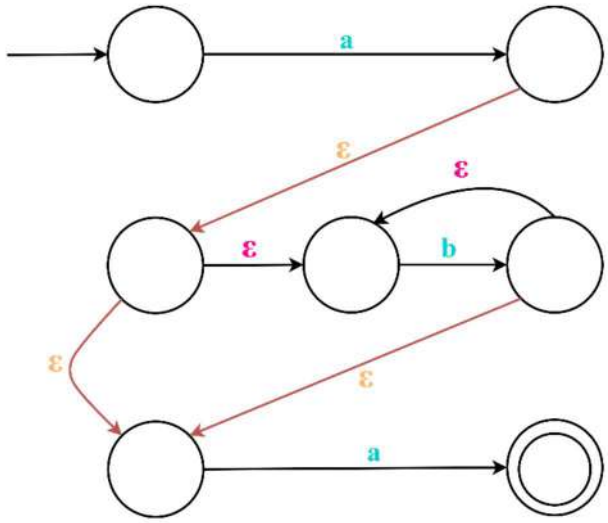
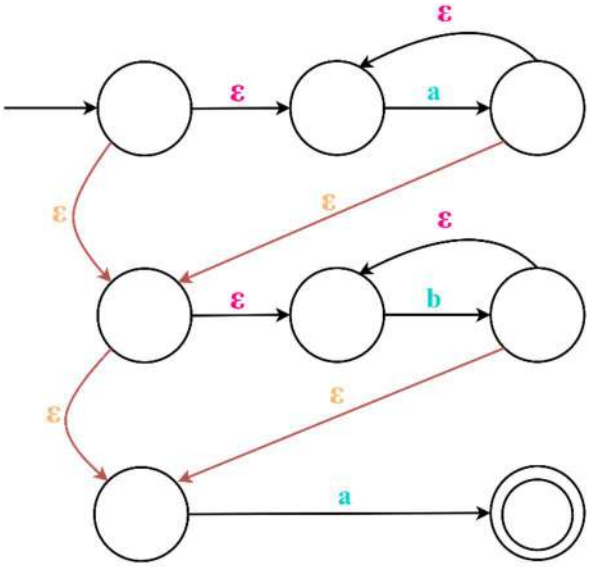


Practice Problem

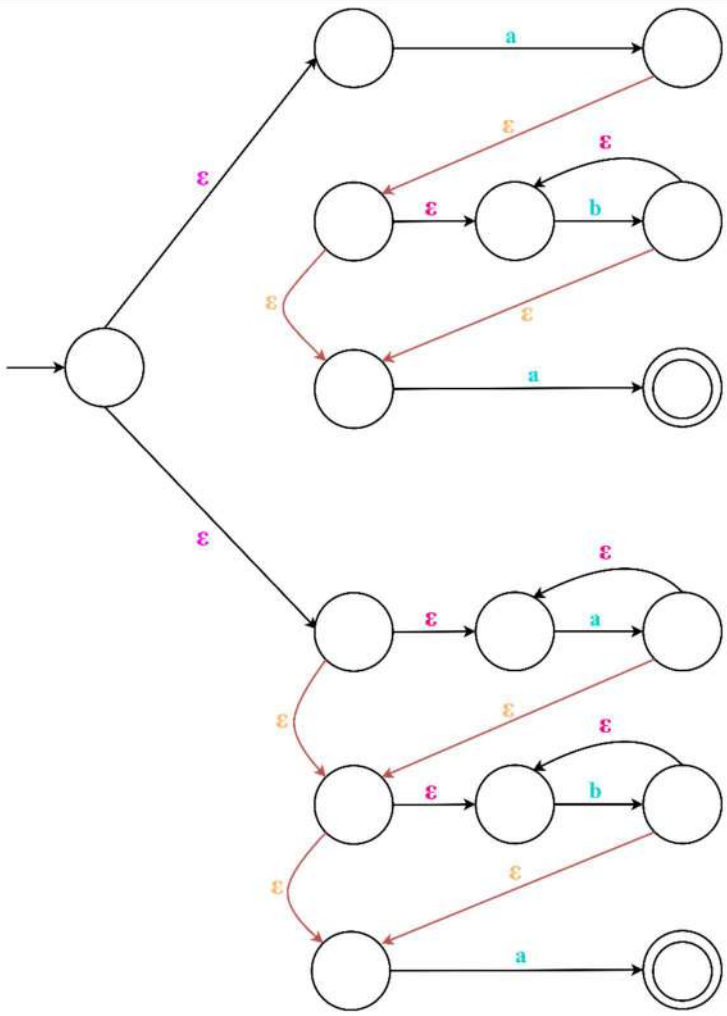
🧠 Problem1: Convert the following Regular Expression into equivalent NFA using formal procedure. "***(a∪b)*aba***".

Step 1: Parse the RE into Subexpressions	**Step 2:** Build NFAs for Basic Symbols	
≫ ***The* RE** "(a|b)*aba" **can be decomposed as:** 1. (a|b)* 2. a 3. b 4. a We'll build NFAs for each component and then concatenate them.	NFA for a	NFA for b
		
Step 4: NFA for (a|b)* (Star)	**Step 3:** NFA for (a|b) (Union)	
		
Step 5: NFA for "aba" (Concatenation)	**Step 6:** Combine (a|b)* and aba (Concatenation)	
		

🧠 Problem2: Convert the following Regular Expression into equivalent NFA using formal procedure. "**ab*a U a*b*a**".

Step 1: Parse the RE into Subexpressions	**Step 2:** Build NFAs for Basic Symbols	
≫ ***The* RE** "ab*a U a*b*a" **can be decomposed as:** 1. ab*a 2. a*b*a 3. a 5.a* 4. b 6.b* We'll build NFAs for each component and then concatenate them.	NFA for a	NFA for b
		
Step 4: NFA for b* (Star)	**Step 3:** NFA for a* (Star)	
		
Step 5: NFA for ab*a (Concatenation)	**Step 6:** NFA for a*b*a (Concatenation)	
		

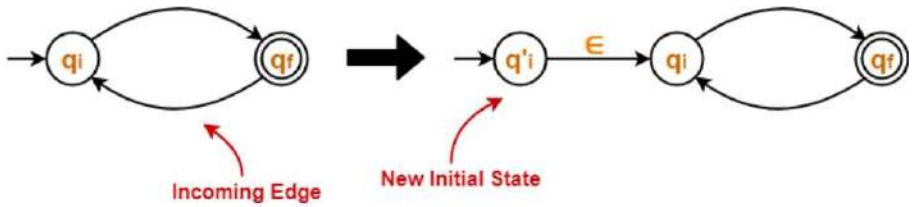
Step 7: Final NFA for ab*a U a*b*a (Union)



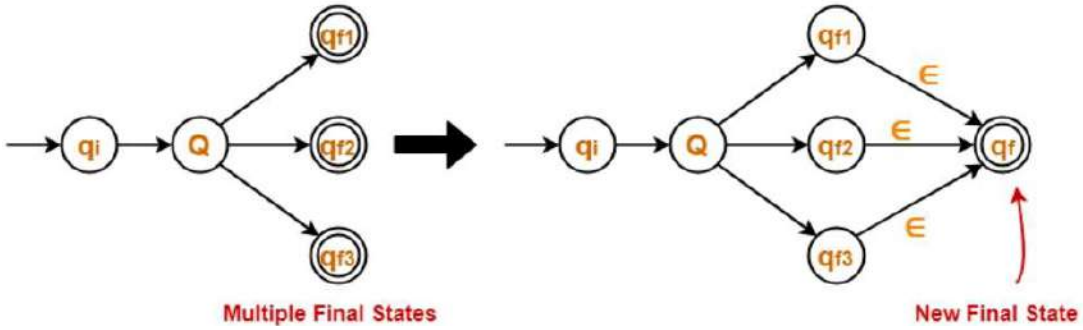
CONVERSION OF DFA TO RE

State Elimination Method & Conversion of GNFA

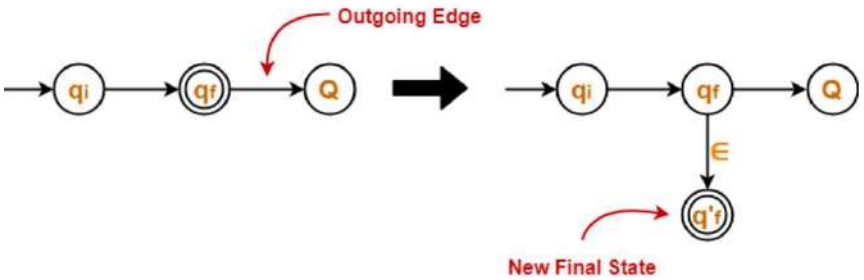
Rule1: There must be only one initial state, and there should not be any incoming edges to it.



Rule2: There must be only one final state.



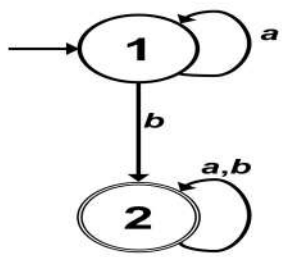
Rule3: From the final state, there should be no outgoing edges.

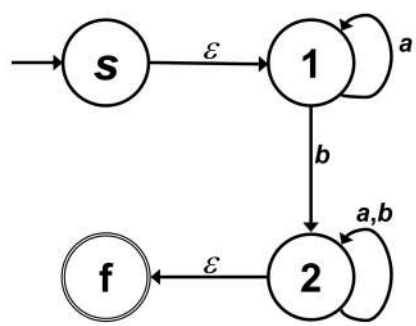


Rule4: After verifying rules 1 to 3, all intermediate states are eliminated one by one. We can eliminate these intermediate states in any order.

Step-by-Step Conversion of DFA to Regular Expression (RE)

🧠 Problem1: Convert the following DFA into equivalent Regular Expression using Formal Procedure.



Step1: Represent the DFA	**Step2:** Converting the DFA to GNFA
➢ Q ={1,2}, ➢ Σ ={a,b}, ➢ q0 =1, ➢ F ={2}	≫**Add a new unique start state q_s and final state q_f** with ε-transitions: ○ $q_s \xrightarrow{\varepsilon} q_0$ ○ All original final states $q_f \xrightarrow{\varepsilon}$ new q_f ≫**Ensure only one final state** (merge if multiple exist). 

Step 3: Eliminate States One by One

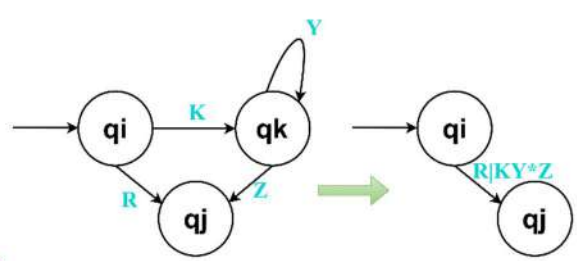
≫General Rule for Eliminating State q_k:

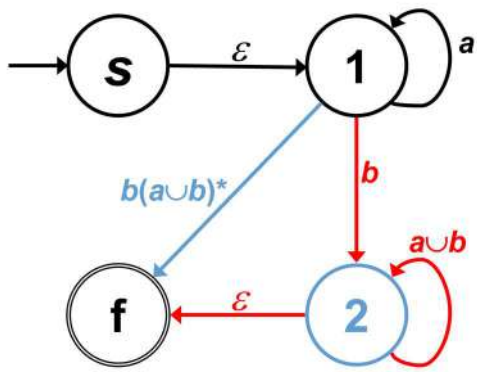
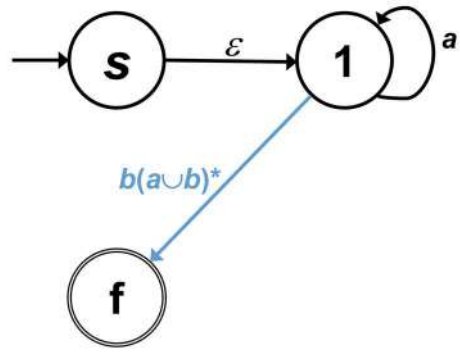
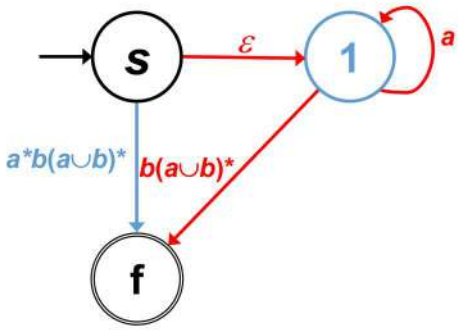
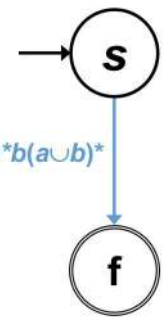
For every pair of states q_i and q_j:

1. Existing transition: $q_i \xrightarrow{R} q_j$
2. Incoming to q_k: $q_i \xrightarrow{X} q_K$
3. Loop on q_k: $q_k \xrightarrow{Y} q_K$
4. Outgoing from q_k: $q_k \xrightarrow{Z} q_j$

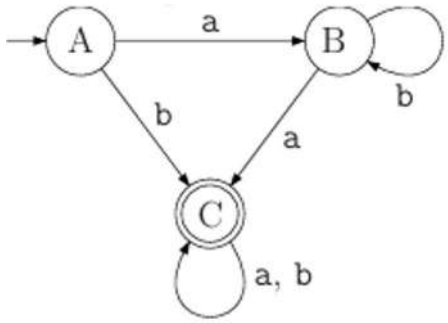
After eliminating q_k add a new transition:

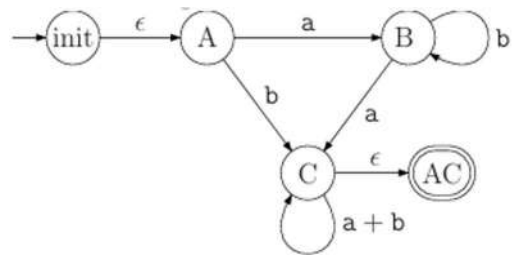
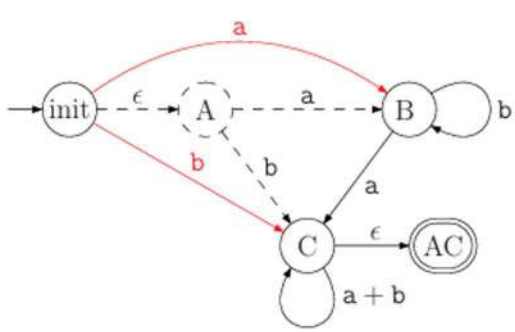
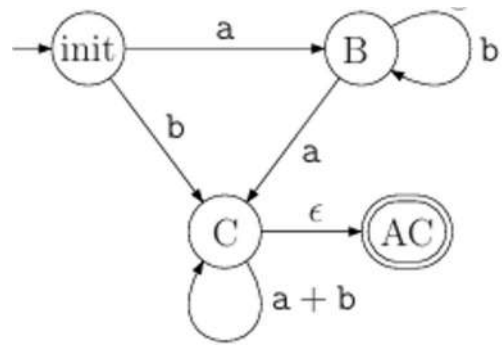
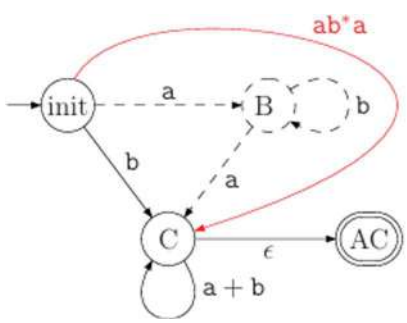
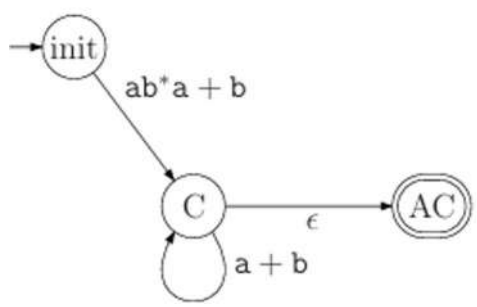
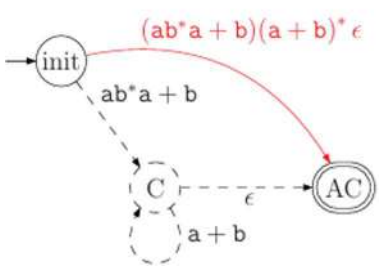
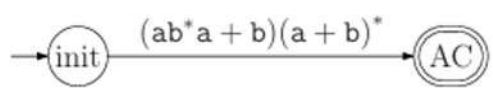
$$q_i \xrightarrow{R|XY^*Z} q_j$$



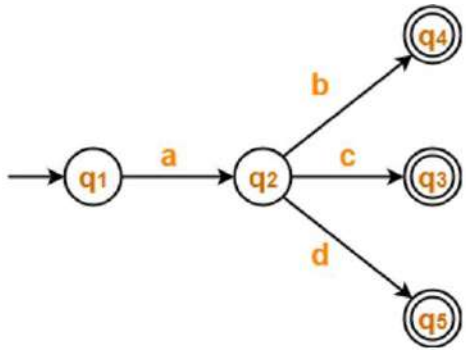
Step 3.1.1: Start Eliminate state 2	**Step 3.1.2:** After Eliminate state 2
	
Step 3.2.1: Start Eliminate state 1	**Step 3.2.2:** After Eliminate state 1
	

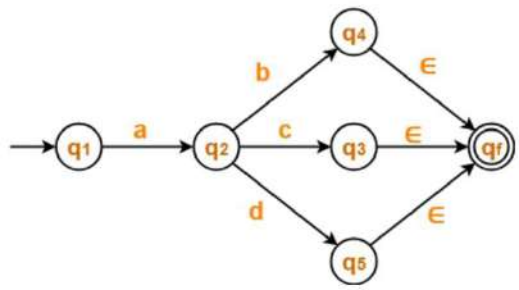
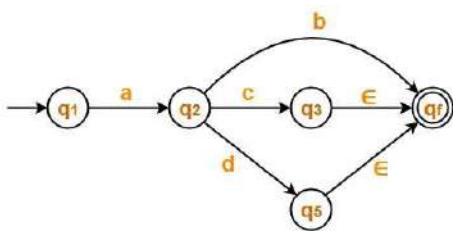
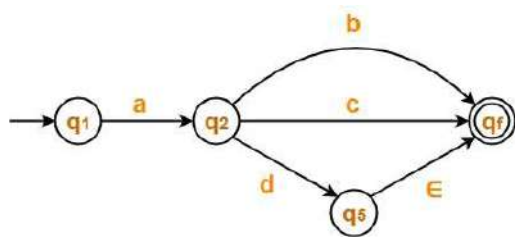
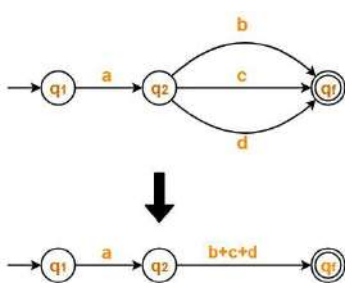
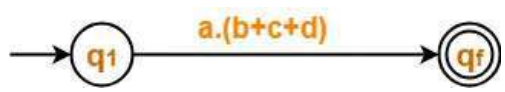
Problem2: Convert the following DFA into equivalent Regular Expression using Formal Procedure.



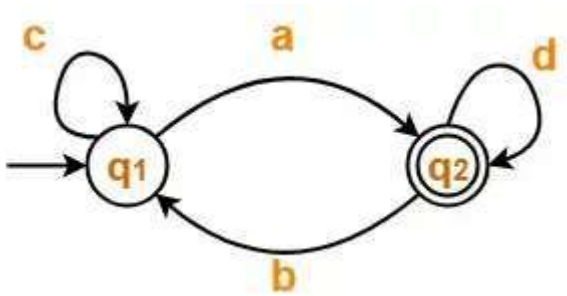
Step1: Represent the DFA	**Step2:** Converting the DFA to GNFA
➢ $Q = \{A,B,C\}$, ➢ $\Sigma = \{a,b\}$, ➢ q0 =A, ➢ F ={C}	
Step 3: Eliminate States One by One	
Step 3.1.1: Start Eliminate state A	**Step 3.1.2:** After Eliminate state A
	
Step 3.2.1: Start Eliminate state B	**Step 3.2.2:** After Eliminate state B
	
Step 3.3.1: Start Eliminate state C	**Step 3.3.2:** After Eliminate state C
	

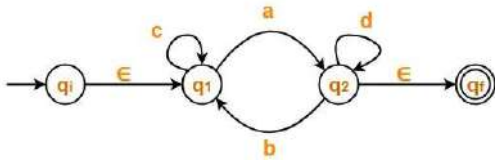
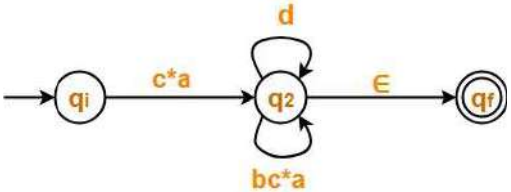
🧠 Problem3: Convert the following DFA into equivalent Regular Expression using Formal Procedure.



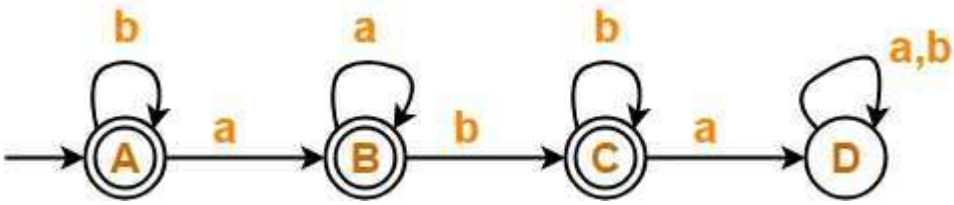
Step1: Represent the DFA	**Step2:** Converting the DFA to GNFA
➢ Q ={q1,q2,q3,q4,q5}, ➢ Σ ={a,b,c,d}, ➢ q0 =q1, ➢ F ={q3,q4,q5}	
Step 3: Eliminate States One by One	
Step 3.1: Eliminate state q4	**Step 3.2:** Eliminate state q3
	
Step 3.3: Eliminate state q5	**Step 3.4:** Eliminate state q2
	

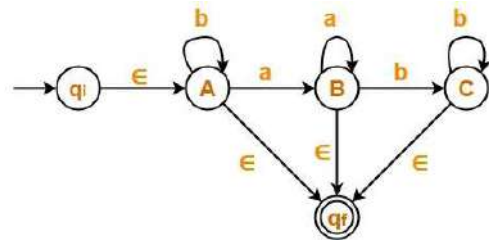
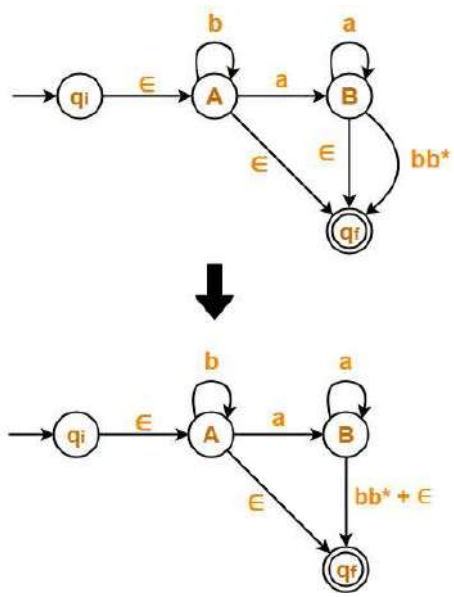
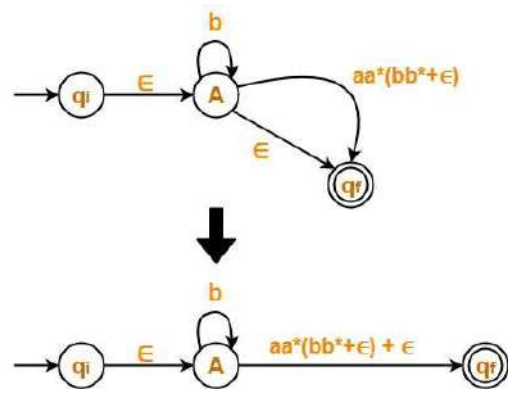
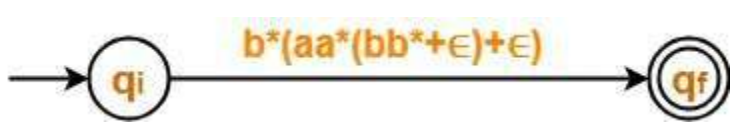
Problem4: Convert the following DFA into equivalent Regular Expression using Formal Procedure.



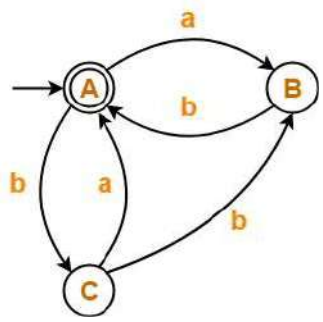
Step1: Represent the DFA	**Step2:** Converting the DFA to GNFA
➢ Q ={q1,q2 }, ➢ Σ ={a,b,c,d}, ➢ q0 =q1, ➢ F ={q2}	
Step 3: Eliminate States One by One	
Step 3.1: Eliminate state q1	**Step 3.2:** Eliminate state q2
	

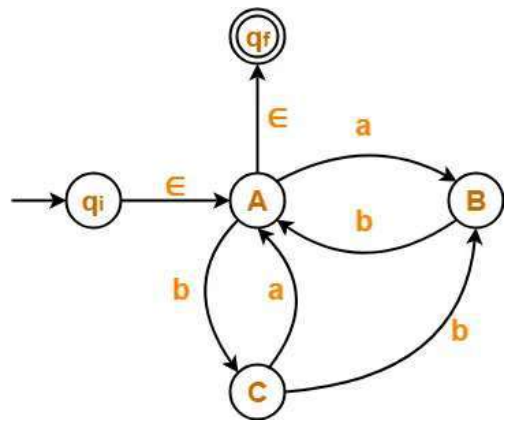
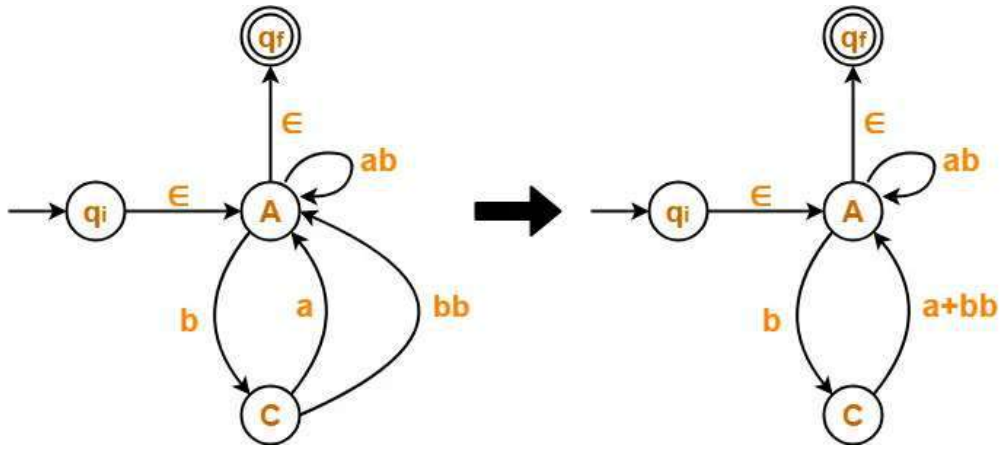
Problem5: Convert the following DFA into equivalent Regular Expression using Formal Procedure.

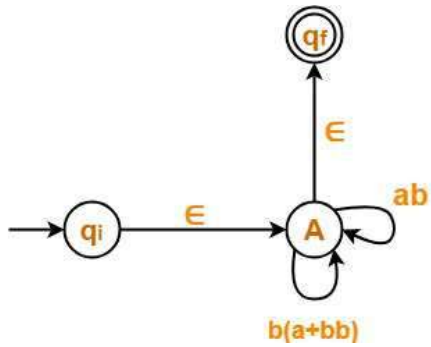


Step1: Represent the DFA	**Step2:** Converting the DFA to GNFA
➢ Q ={A,B,C,D}, ➢ Σ ={a,b}, ➢ q0 =A, ➢ F ={A,B,C}	
Step 3: Eliminate States One by One	
Step 3.1: Eliminate state C	**Step 3.2:** Eliminate state B
	
Step 3.3: Eliminate state A	
	

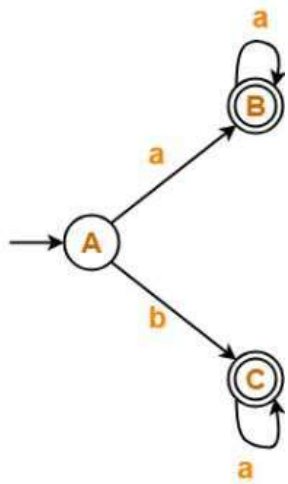
🧠 Problem6: Convert the following DFA into equivalent Regular Expression using Formal Procedure.

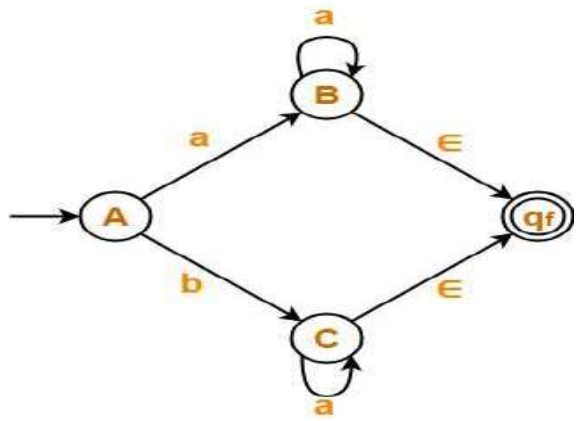


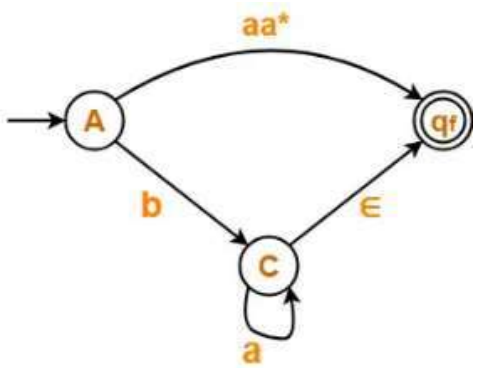
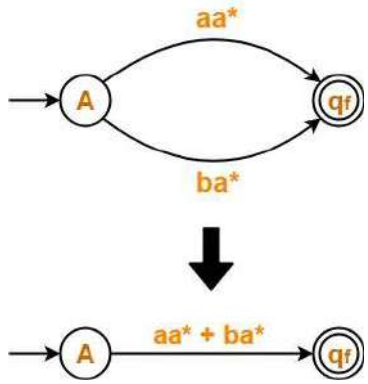
Step1: Represent the DFA	**Step2:** Converting the DFA to GNFA
➢ Q ={A,B,C}, ➢ Σ ={a,b}, ➢ q0 =A, ➢ F ={A}	
Step 3: Eliminate States One by One	
Step 3.1: Eliminate state B	
	

Step 3.2: Eliminate state C	**Step 3.3:** Eliminate state A
	

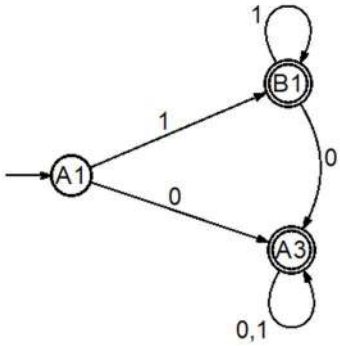
Problem7: Convert the following DFA into equivalent Regular Expression using Formal Procedure.

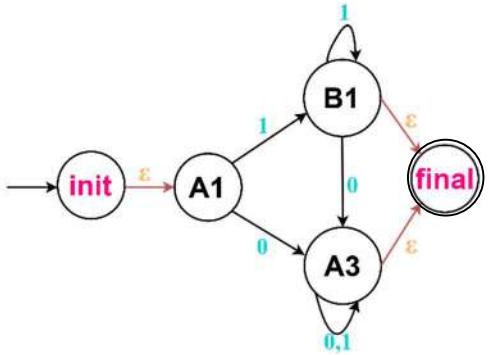


Step1: Represent the DFA	**Step2:** Converting the DFA to GNFA
➢ Q ={A,B,C }, ➢ Σ ={a,b}, ➢ q0 = A, ➢ F ={B,C}	

Step 3: Eliminate States One by One	
Step 3.1: Eliminate state B	**Step 3.2:** Eliminate state q2
	

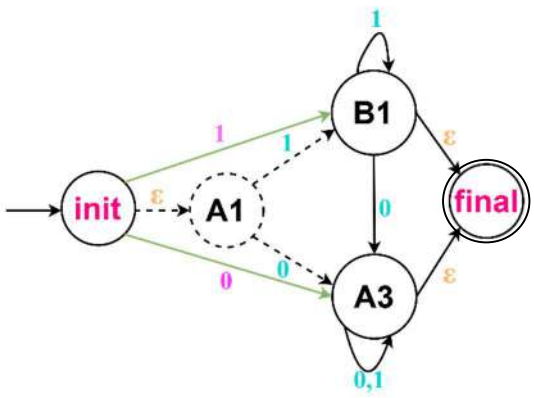
Problem8: Convert the following DFA into equivalent Regular Expression using Formal Procedure.



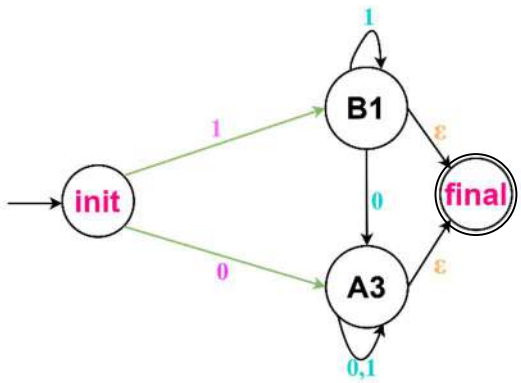
Step1: Represent the DFA	**Step2:** Converting the DFA to GNFA
➢ Q ={A1,A3,B1}, ➢ Σ ={0,1}, ➢ q0 =A1, ➢ F ={A3,B1}	

Step 3: Eliminate States One by One

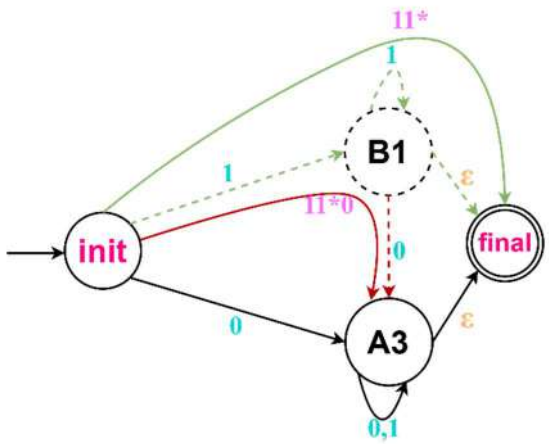
Step 3.1.1: Start Eliminate state A1



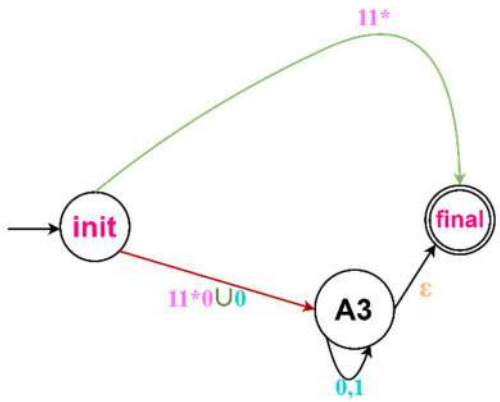
Step 3.1.2: After Eliminate state A1



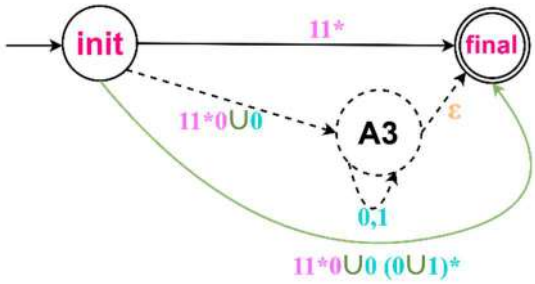
Step 3.2.1: Start Eliminate state B1



Step 3.2.2: After Eliminate state B1



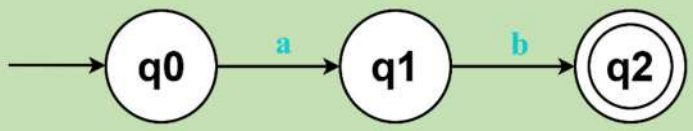
Step 3.3.1: Start Eliminate state A3



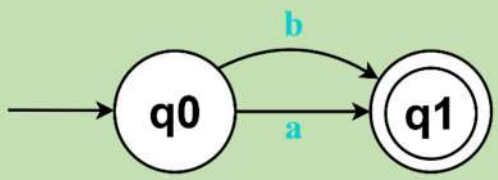
Step 3.3.2: After Eliminate state A3

Basics of Conversion: Practice Exercises

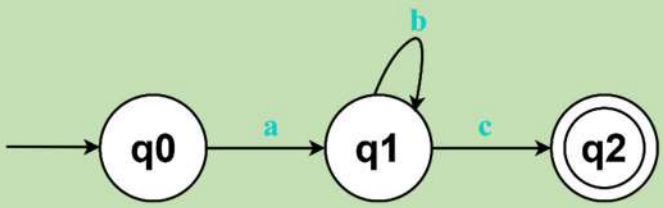
Conversion 1:

DFA	RE
	$\therefore R = ab$

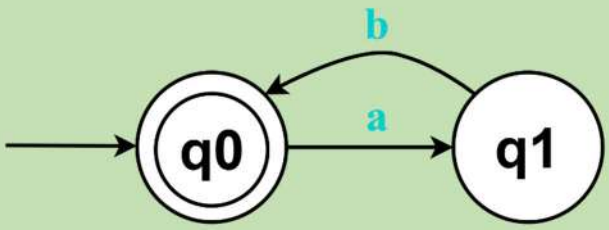
Conversion 2:

DFA	RE
	$\therefore R = a\|b$

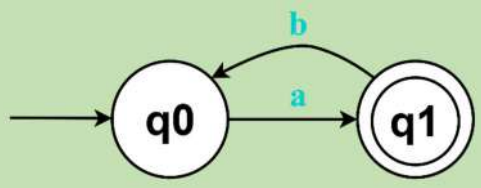
Conversion 3:

DFA	RE
	$\therefore R = ab^*c$

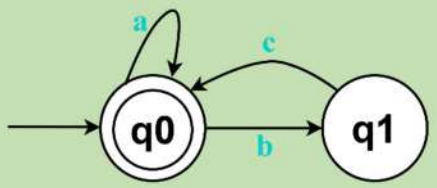
Conversion 4:

DFA	RE
	$\therefore R = (ab)^*$

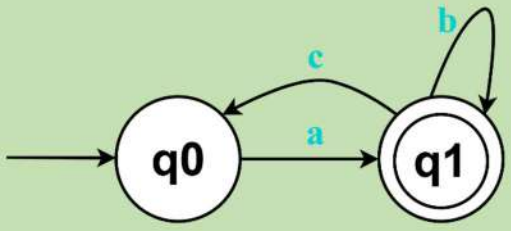
Conversion 5:

DFA	RE
	$\therefore R = a\ (ba)^*$

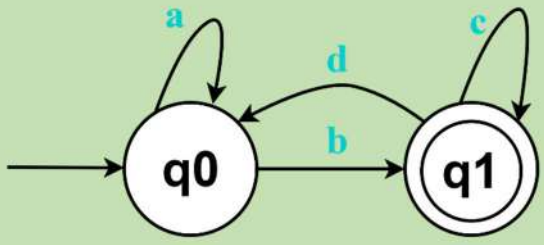
Conversion 6:

DFA	RE
	$\therefore R = (a^* \mid (bc)^*)^*$

Conversion 7:

DFA	RE
	$\therefore R = a(b^* \mid (ca)^*)^*$

Conversion 8:

DFA	RE
	$\therefore R = a^* \mid b(c^* \mid (d(a^* \mid b))^*)^*$