Unit-1 (Theory of Automata)

Theory of Computation:

Theory of Computation (ToC) deals with whether and how effecientl problems can be solved on a model of Computation using an algorithm

Types of ToC:

① Computability theory
② Computational complexity theory
③ Automata Theory

---

① **Computability theory:** This theory is given by Godel, Church, Turing, Kleene & post. This theory is used to determine which problems are Computable (solvable) and which are not-Computable (Unsolvable). It is also called recursive theory.

② **Computational Complexity theory:**

Computational Complexity theory deals with determining the Complexity of the problem during Computation. If a problem is effecientl solvable then it is "computationally easy" and if it can not be solved effecientl then it is "Computationally hard".

Automata Theory: Automata theory deals with designing a self-propelled computing device (Automaton) that follows a predetermined sequence of operations automatically.

Automaton: An automaton is a mathematical model of any computing device/machine that perform computations on an input by moving through a series of states.

* At each state of computation, a transition function determines the next state based on present state and input.
* once the computation reaches an accepting state, it accepts the input else rejects it.

Types of Automaton:

1. Finite Automaton (FA)
2. Pushdown Automaton (PDA)
3. Linear Bounded Automaton (LBA)
4. Turing Machine (TM)

* Finite Automaton (FA) is the most restricted Automaton, while Turing machine is an unrestricted Automaton.
* Turing machine is the most powerful automaton; it can compute all solvable problems.

① **Alphabet**: An alphabet is a finite, non empty set of symbols. It is represented by symbol Σ.

examples

(i) $\Sigma\{0,1\}$ is binary alphabet.
(ii) $\Sigma(A, B \cdots Z)$ is alphabet of uppercase letter.
(iii) $\Sigma(0,1,2,\cdots 9)$ is decimal alphabet.

② **Strings (words)**: It is a finite sequence of symbols from the alphabet.

examples

(i) 1101 is binary string.
(ii) 2576 is decimal string.
(iii) abcab is small case letter string.

③ **Length**:
If string has no symbol then it is called empty string (λ) (ϵ)
The length of a string is no. of symbols in that string. The standard notation for the length of string w is $|w|$.

If $w = abcab$
$|w| = 5$

Length of an empty string is zero
$|\lambda| = |\epsilon| = 0$

~~④ Kleene Closure (Kleene star)~~

④ **Kleene Closure**

(i) **Kleene Star Closure** (Σ^*): The set of all possible strings over an alphabet Σ including null string (λ)

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$$

Σ^0 → null string (λ)
Σ^1 → strings (length = 1)
Σ^2 → strings (length = 2)

If $\Sigma = \{0,1\}$
$\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001 \cdots\}$

(ii) Kleene positive closure (Σ^+):

The set of all possible strings over an alphabet Σ excluding null string(λ) is called Kleene positive closure.

So $\Sigma^+ = \Sigma^* - \{\lambda\}$

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

$\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\}$

(5) Reverse of a string (ωR):

The reverse of a string ω is obtained by writing the symbols in reverse order (ωR).

ex: If $\omega = abc$ then

$\omega^R = cba$

(6) Language: A language is a subset of strings over an alphabet Σ following a specific rule. It is represented by L.

If Σ is an alphabet then L is a subset of Σ^* following certain rule.

example: (1) If $\Sigma = \{0,1\}$

$L = \{\omega \in \{0,1\}^* \mid \omega$ has equal no. of 0's and 1's$\}$

$L = \{\lambda, 01, 10, 0011, 0110, 1001, 1010, 1100, \ldots\}$

(2) If $\Sigma = \{a,b\}$ then

$L = \{\omega \in \{a,b\}^* \mid \omega$ has all strings ending with b$\}$

$L = \{b, ab, abb, aab, bb, bab, aabb, \ldots\}$

(3) If $\Sigma = \{a,b,c\}$, then

$L = \{\omega \in \{a,b,c\}^* \mid \omega$ has all strings ending with cc$\}$

$L = \{cc, acc, bcc, abcc, bbcc, aacc, bacc, \ldots\}$

Applications of Automata Theory

1. Designing the lexical analysis of compiler (FA)
2. Design & Verification of hardware digital circuits (FA).
3. Used in text editors (FA).
4. for the implementation of spell checker (FA).
5. In pattern recognition (FA).
6. Designing the syntax analysis of compiler (PDA).
7. for implementation of genetic programming (LBA)
8. for implementation of neural networks (TM).
9. for implementation of robotics applications (TM).
10. for implementation of artificial Intelligence (TM)

Additional applications of Automata Theory

11. Automatic photo printing machine.
12. Card punching machine.
13. Automatic traffic lights.
14. Aircraft control.
15. Spacecraft.
16. Vending machine.
17. Elevators.

Grammar: Grammar specifies the rules that are required to generate strings (words) of a language. It is represented by G.

Each grammar G has 4 tuples

$$G = (N, T, S, P)$$

Where

$N \rightarrow$ Set of Non-terminals (variables) Represented by capital letters

$T \rightarrow$ Set of terminals (Constants) Represented by small letters (numeric values)

$S \rightarrow$ Start Non-terminal (Variables)

$P \rightarrow$ Production Rule(s).

$$\alpha \rightarrow \beta$$

α: Left side of rule

β: right side of rule

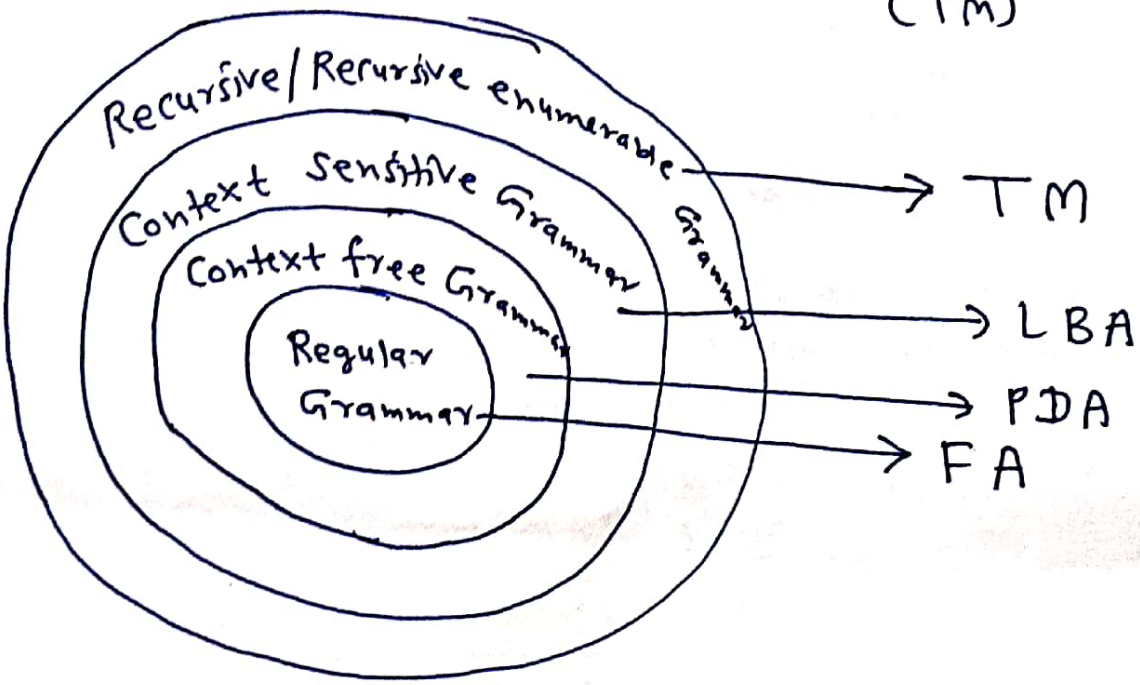
Chomsky classification of grammar:

* It is also called Chomsky hierarchy.
* This classification is given by Noam Chomsky in 1956.
* According to Chomsky hierarchy, Grammars are divided into 4 types.

(1) Regular Grammar (Type-3 Grammar)
(2) Context free Grammar (Type-2 Grammar)
(3) Context Sensitive Grammar (Type-1 Grammar)
(4) Recursive / Recursive enumerable Grammar (Type-0 Grammar)

Most Restricted Grammar → Type-3 Grammar
Unrestricted Grammar → Type-0 Grammar

Grammar	Automaton
(Type-3) Regular Grammar	→ Finite Automaton (FA)
(Type-2) Context-free Grammar	→ Pushdown Automaton (PDA)
(Type-1) Context Sensitive Grammar	→ Linear Bounded Automaton (LBA)
(Type-0) Recursive / Recursive enumerable Grammar	→ Turing Machine (TM)



① **Regular Grammar**: It is also called type-3 grammar. It is accepted by Finite Automaton (FA).

The production rule of this grammar

$$\alpha \rightarrow \beta$$

α: Single Non-terminal

β: Single terminal followed by Single Non-terminal
or
Single non-terminal followed by single terminal
or
Single terminal

examples:

① $S \rightarrow aB$
$B \rightarrow b$

② $S \rightarrow Bb$
$B \rightarrow a$
$B \rightarrow \lambda$

Regular Grammar

- Right linear Grammar
- Left Linear Grammar

(a) Right Linear Grammar:

If the productions are of the form

$\alpha \longrightarrow \beta$

- α: Single Non-terminal
- β: single terminal or single terminal followed by single Non-terminal

example:

$S \rightarrow aA$
$A \rightarrow b$
$A \rightarrow \lambda$

(b) Left Linear Grammar:

If the production are of the form

$\alpha \longrightarrow \beta$

- α: Single Non-terminal
- β: single terminal or Single Non terminal followed by a single terminal

example:

$S \rightarrow Ba$
$B \rightarrow d$

A regular Grammar is either right linear Grammar or left linear grammar.

② Context free grammar: (CFG):

* It is also called type-2 grammar
* It is accepted by Pushdown Automaton (PDA).
* The production rule of CFG is

$$\alpha \longrightarrow \beta$$

α: Single Non-terminal

β: any Combination of terminals and Non-terminals

Examples:

(i)
$S \to aAb$
$A \to aC$
$C \to b$

(ii)
$S \to AB$
$A \to aB$
$B \to bB$
$B \to \lambda$

(iii)
$S \to AbB$
$A \to \lambda$
$B \to c$

(iv)
$S \to AC$
$A \to aB$
$B \to b$
$C \to \lambda$

(v)
$S \to AB$
$A \to aC$
$B \to Bb$
$B \to \lambda$
$C \to d$

③ Context-Sensitive Grammar (CSG)

* It is also called type-1 grammar.
* It is accepted by Linear Bounded Automata.
* The production rule for this grammar is

$$\alpha \longrightarrow \beta$$

α: any combination of terminals and non-terminals except null string

β: any combination of terminals and non-terminals

(i) $\boxed{|\alpha| \leq |\beta|}$

i.e. No. of Symbols in L.H.S should be less than or equal to No. of symbols in R.H.S

(ii) There should be at least one Non-terminal on the L.H.S.

Examples

(i) $aS \rightarrow aAB$
$A \rightarrow b$
$B \rightarrow bA$

(ii) $Sb \rightarrow aSA$
$S \rightarrow bA$
$A \rightarrow d$

(iii) $S \rightarrow aAB$
$aA \rightarrow bC$
$B \rightarrow \lambda$
$C \rightarrow d$

(iv) $SaB \rightarrow aAB$
$A \rightarrow b$
$B \rightarrow \lambda$

④ Recursive / Recursive Enumerable Grammar:

* It is type-0 grammar.
* It is also called Unrestricted grammar.
* It is accepted by Turing Machine.
* The production rule are as given:

$$\alpha \longrightarrow \beta$$

α: any Combination of Non-terminals and terminals except null string

β: any Combination of non-terminals and terminals

(i) There should be at least one non-terminal on L.H.S.

(ii) No. of symbols in L.H.S can be less than/equal/greater than No. of symbols in R.H.S.

Examples: (i)
$$aS \to B$$
$$B \to C$$
$$C \to d$$

(ii)
$$SA \to AB$$
$$A \to \lambda$$
$$B \to c$$

(iii)
$$Sb \to aSA$$
$$S \to b$$
$$A \to d$$

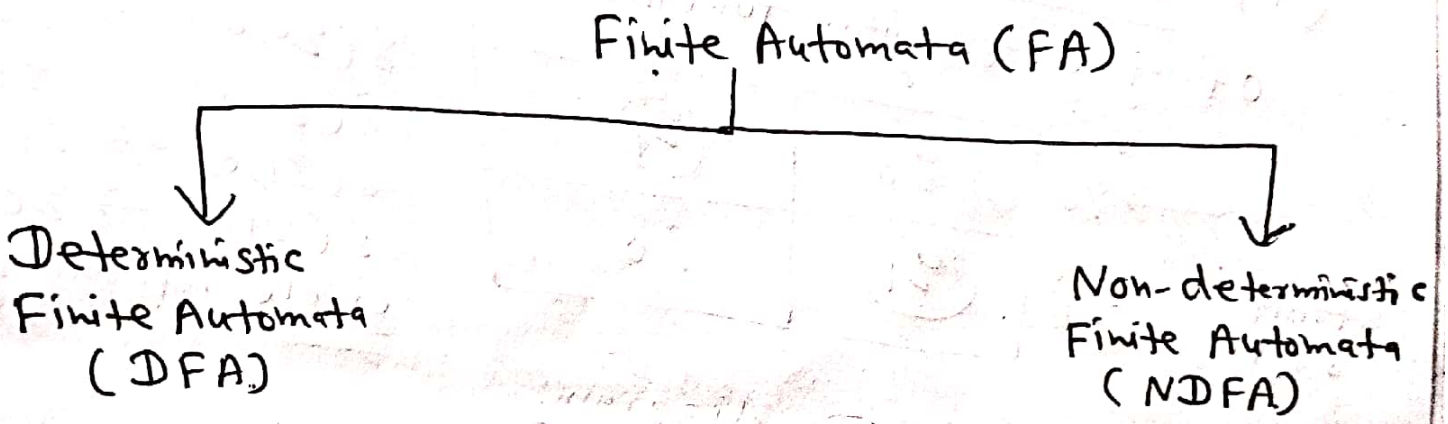
Important Note

If you want to check the type of any grammar, always start from type-3; If it is not type-3 grammar, check whether it is type-2 or not. If it is not type-2, check whether it is type-1 or not. If it is also not type-1 grammar, than it is type-0 grammar.

Finite Automaton (FA) & its types (12)

Finite Automaton (FA): * It is the most-restricted mathematical model of computing device (Automatic machine).

* It accepts type-3 (regular grammar).
* FA is used for string matching, lexical analysis phase of Compiler, pattern matching and verification of hardware digital circuits.
* It is of two types



Deterministic Finite Automata (DFA):

* The word "deterministic" in DFA refers to the fact that the transition of FA from one state to another is unique (deterministic) on any input symbol.
* A DFA consists of 5 tuples $(Q, \Sigma, q_0, F, \delta)$
  (1) A finite set of states, represented as Q.
  (2) A finite set of ~~Sym~~ alphabet, represented as Σ.
  (3) An initial state q_0.
  (4) A set of final states (accepting states). It is represented by F.
  where $F \subseteq Q$ (F is a subset of Q)
  (5) Transition function (Next state function) δ is used to show that next state depends on

present state and present input.

$$\delta: Q \times \Sigma \rightarrow Q'$$

Example 1: $\delta(q_0, a) \rightarrow q_1$

example 2: $\delta(q_1, b) \rightarrow q_1$

Representation of transition function in DFA

(i) Transition Table (ii) Transition diagram

(i) Transition Table: It is the tabular representation of the transition function.

Q / Σ	0	1
→ q_0	q_0	q_1
(q_1)	q_1	q_0

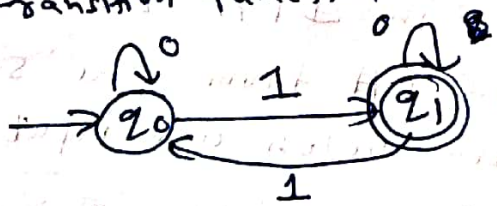
Starting state

$\delta(q_0, 0) \rightarrow q_0$
$\delta(q_0, 1) \rightarrow q_1$
$\delta(q_1, 0) \rightarrow q_1$
$\delta(q_1, 1) \rightarrow q_0$

Circle represent final states

(ii) Transition diagram:

It is the diagrammatical representation of the transition function.



Acceptance of a string by Finite Automata (FA)

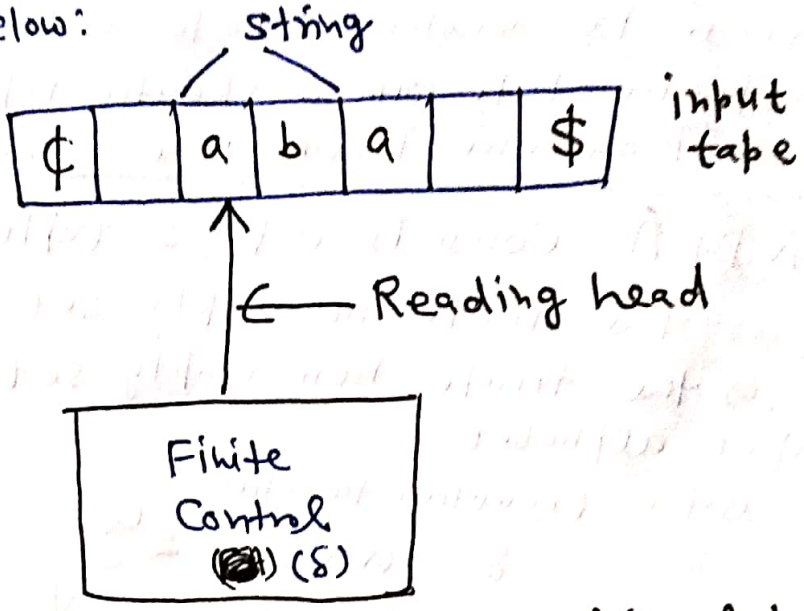
example: "10110"

$(q_0, 10110) \Rightarrow (q_1, 0110)$
$\Rightarrow (q_1, 110)$
$\Rightarrow (q_0, 10)$
$\Rightarrow (q_1, 0)$
$\Rightarrow (q_1, 1)$
$\Rightarrow q_0$ (Not accepted)

i.e. does not belong to language of DFA.

Block diagram of FA (DFA/NDFA)

The block diagram of FA (DFA or NDFA) is as shown below:



Block diagram of Finite Automaton (FA)

Components of Finite Automaton (FA)

① **input tape:**

(i) The input tape is divided into squares, each square consists of single symbol from the input alphabet Σ.

(ii) There is an endmarker (¢) at the left hand side and end marker ($) at the right hand side

(iii) The presence of endmarkers indicate that input tape is of finite length.

② **Reading head:**

(i) The Reading head is used to read symbol from the input tape.

(ii) The reading head reads one symbol at a time.

(iii) Normally the reading head reads from left to right. However in specific FA, we can move in both directions.

③ **Finite Control:** The finite control decides the next state on receiving particular input from input tape. It decides next state using transition function.

Non-deterministic Finite Automata (NDFA)

* The word "Non-deterministic" in NDFA refers to the fact that the transition of FA from one state to another state is not unique (i.e. for a specific state with a speafic input, there may be more than one next state possible.

* A NDFA Consists of 5-tuple $(Q, \Sigma, \delta, q_0, F)$

(1) Q is the finite non empty set of states.

(2) Σ is the finite non empty set of inputs called input alphabet.

(3) δ is the transition function

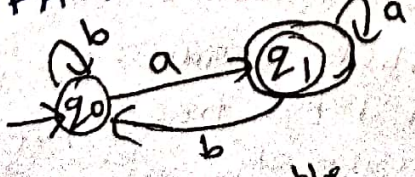
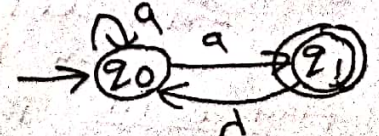
$$\delta: Q \times \Sigma \to 2^Q$$

[Present state (PS)] [input (including null string (λ))] → Total next states (NS)

(4) $q_0 \in Q$ is the initial State.

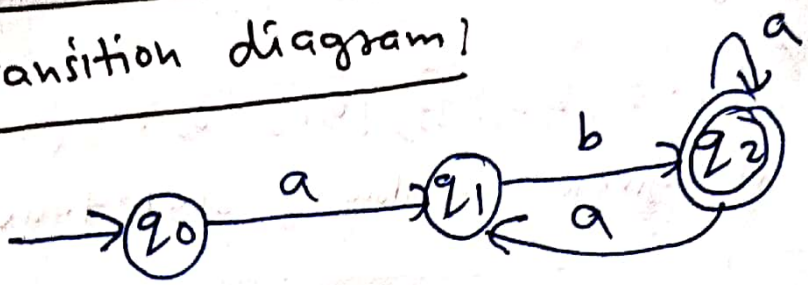
(5) $F \subseteq Q$ is the set of final states.

Difference between DFA and NDFA (NFA)

DFA	NDFA (NFA)
① In DFA, for a given input, there is only one next state. So they are deterministic in nature.	① In NDFA, for a given input there can be many next state possible, so they are nondeter-ministic in nature.
② DFA is defined for each input alphabet in each state.	② NDFA is not necessary to be defined for each input alphabet in each state.
③ null strings (d) are not allowed.	③ null strings are allowed.
④ DFA is difficult to design.	④ NFA is easy to design.
⑤ Conversion from NFA to DFA is possible.	⑤ Conversion from DFA to NFA is not possible.
⑥  DFA example	⑥ 

Representation of NDFA

(i) Transition diagram:



$\Sigma = \{a, b\}$

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

$q_0 \rightarrow$ initial state

$q_2 \rightarrow$ final state

$\delta(q_0, a) = q_1$

$\delta(q_1, b) = q_2$

$\delta(q_2, a) = (q_1, q_2)$

(ii) Transition Table:

Q	ε	a	b
→ q_0		q_1	
q_1			q_2
(q_2)		q_1, q_2	

Equivalence of NDFA and DFA
OR
Conversion of NDFA to DFA

(Ex 2.) Construct DFA (Deterministic Finite Automaton) for the given NFA

Q	ε	0	1
→ (q_0)		q_0	q_1
q_1		q_1	q_0, q_1

Ans DFA is as given

Q	ε	0	1
→ ([q_0])		[q_0]	[q_1]
[q_1]		[q_1]	[q_0, q_1]
([q_0, q_1])		[q_0, q_1]	[q_0, q_1]

δ:

~~Q=0 [q0]~~,

$F = \{[q_0], [q_0, q_1]\}$

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

initial state $= [q_0]$

Ex:2 Construct a deterministic Finite automaton equivalent to

$$M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_3\})$$

where δ is given by following state table

State \ Σ	a	b
→ q_0	q_0, q_1	q_0
q_1	q_2	q_1
q_2	q_3	q_3
(q_3)		q_2

Ans The Equivalent DFA is

State \ Σ	a	b
→[q_0]	[q_0, q_1]	[q_0]
[q_0, q_1]	[q_0, q_1, q_2]	[q_0, q_1]
[q_0, q_1, q_2]	[q_0, q_1, q_2, q_3]	[q_0, q_1, q_3]
([q_0, q_1, q_3])	[q_0, q_1, q_2]	[q_0, q_1, q_2]
([q_0, q_1, q_2, q_3])	[q_0, q_1, q_2, q_3]	[q_0, q_1, q_2, q_3]

Ex:3 Construct a deterministic acceptor equivalent to

$$M = (\{q_0, q_1, q_2\}, (a, b), \delta, q_0, \{q_2\})$$

for the given table.

State \ Σ	a	b
→ q_0	q_0, q_1	q_2
q_1	q_0	q_1
(q_2)		q_0, q_1

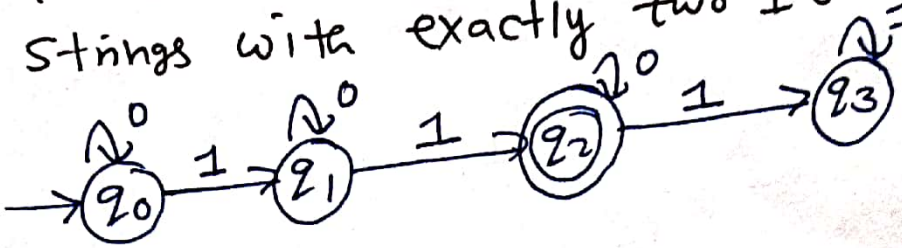
Ans DFA

State | Σ	a	b
→[q_0]	[q_0, q_1]	[q_2]
(q_2)	[ϕ]	[q_0, q_1]
[q_0, q_2]	[q_0, q_1]	[q_1, q_2]
([q_1, q_2])	[q_0]	[q_0, q_1]

Design of DFA

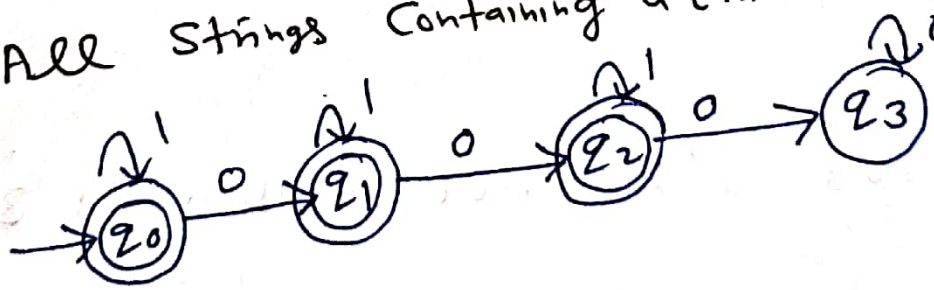
① Design a DFA for the following languages over {0,1}

(a) All strings with exactly two 1's.

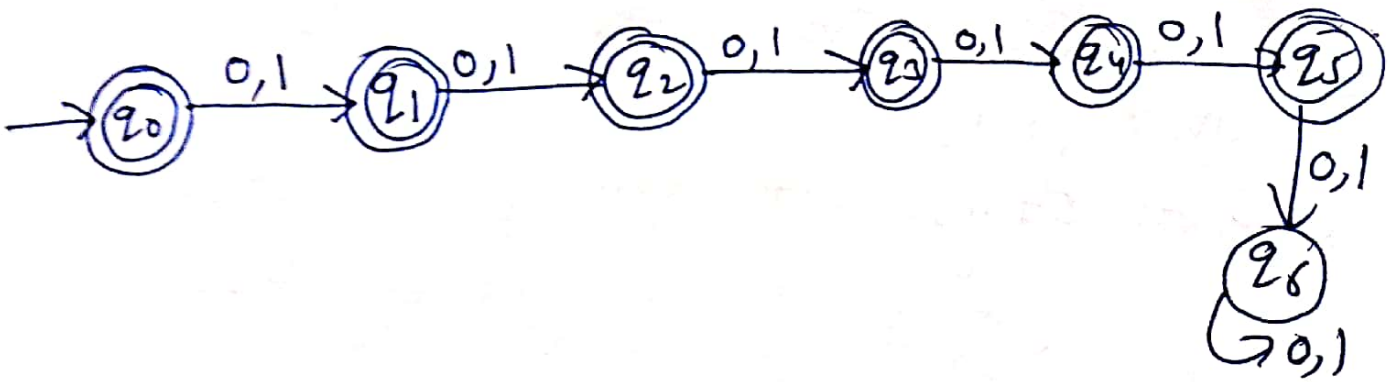


(b) All strings with atleast two 0's.

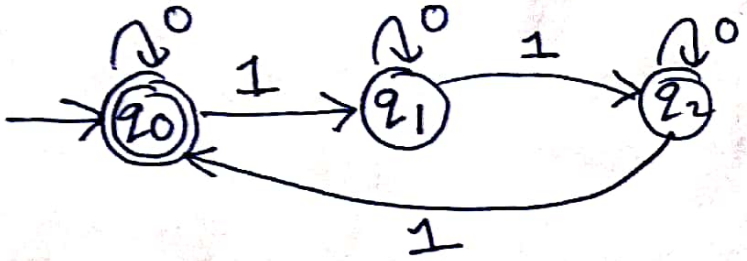
(c) All strings containing atmost two 0's.



(d) All strings of length at most Five. (1.9)

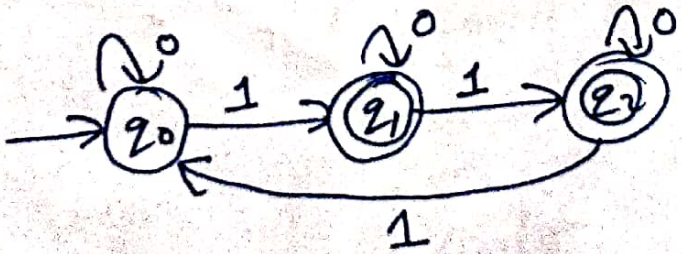


(e) No. of 1's is multiple of 3.



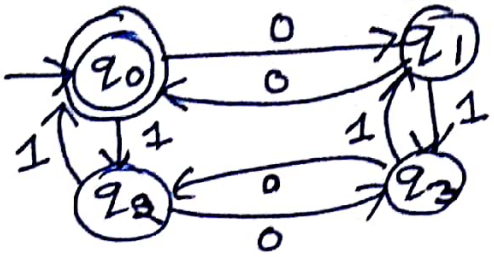
No. of 1's = 0, 3, 6, 9, 12 — —.

(f) No. of 1's not multiple of 3.

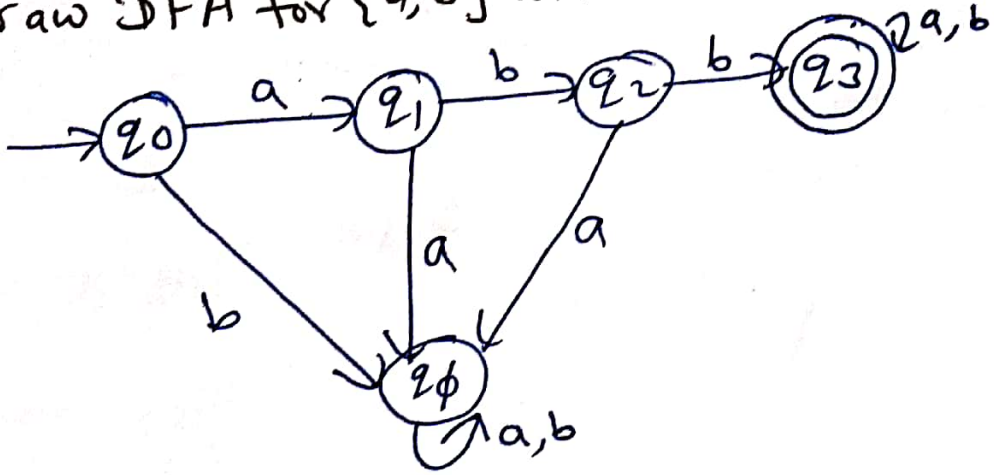


(g) No. of 1's is even and No. of 0's is even.

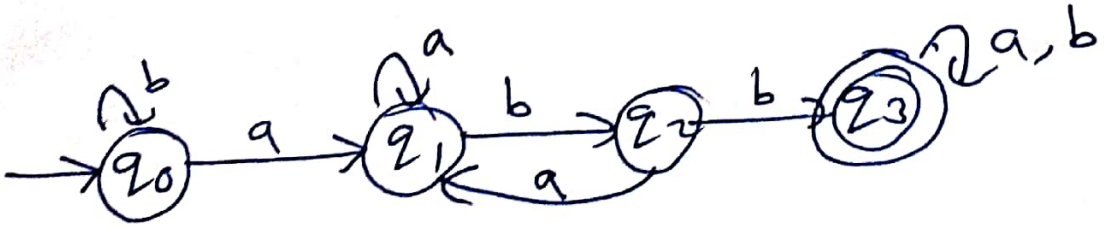
q_0 = No. of 1's (Even) & No. of 0's (Even)
q_1 = No. of 1's (Even) & No of 0's (odd)
q_2 = No. of 1's (odd) & No of 0's (Even)
q_3 = No. of 1's (odd) & No. of 0's (odd)



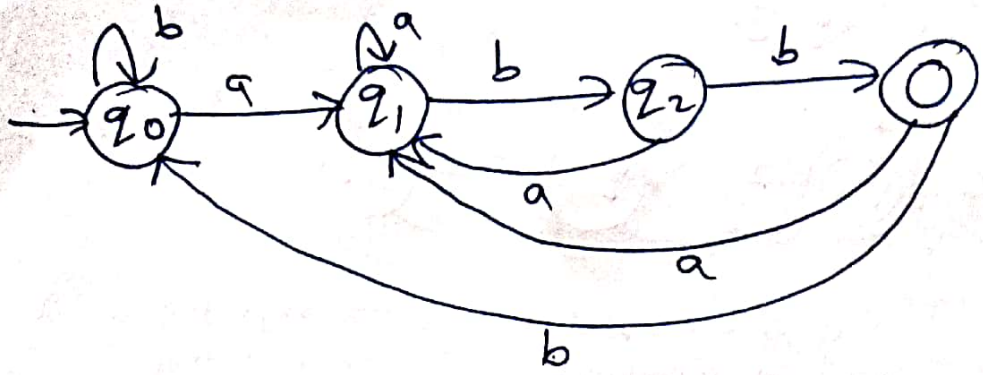
(h) Draw DFA for [a, b] with strings starting with abb.



(i) Draw DFA for [a,b] with abb as substring.

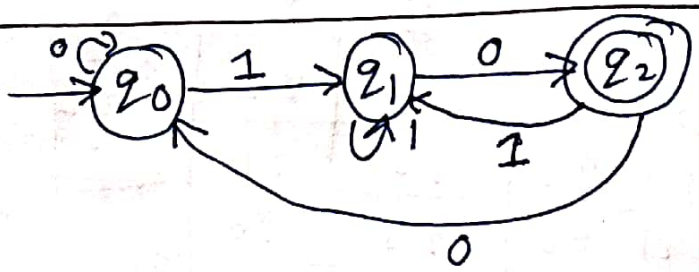


(j) Draw DFA for {a,b} with strings ending in aba. (21)



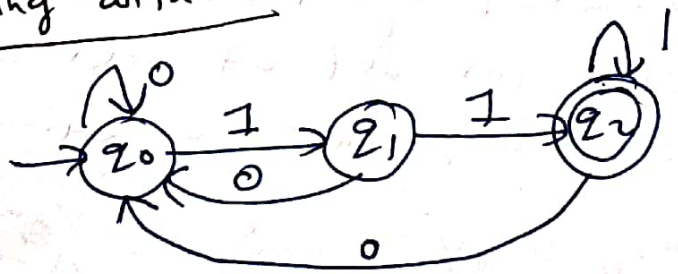
Q22 Draw DFA over {0,1} that
(i) ending with 10 (ii) ending with 11 (iii) ending with 1.

(i)

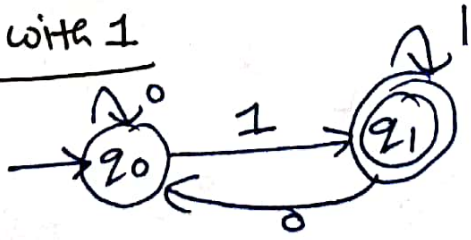


Ending with 10.

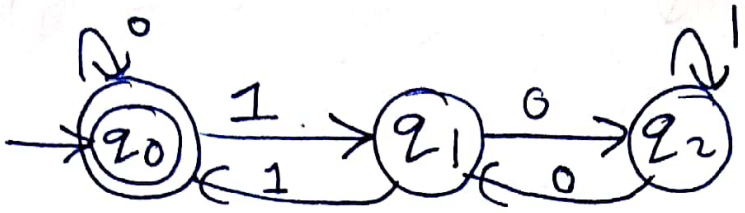
(ii) Ending with 11



(iii) Ending with 1



Q:3 Design a DFA that accept a binary No. divisible by 3.



Σ / δ	0	1
→ q0	q0	q1
q1	q2	q0
q2	q1	q2

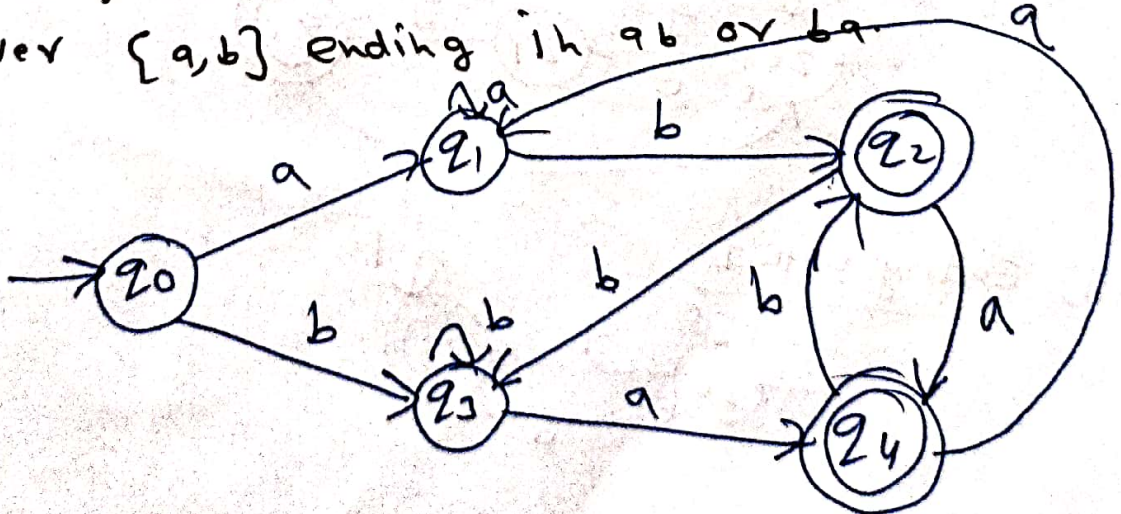
0 → accepted
11 → accepted
110 → accepted
1001 → accepted
1100 → accepted
1111 → accepted

1 → rejected
01 → rejected
10 → rejected
100 → rejected
101 → rejected
111 → rejected
1000 → rejected

Q:4 Design a DFA that accept a ternary number divisible by 4.

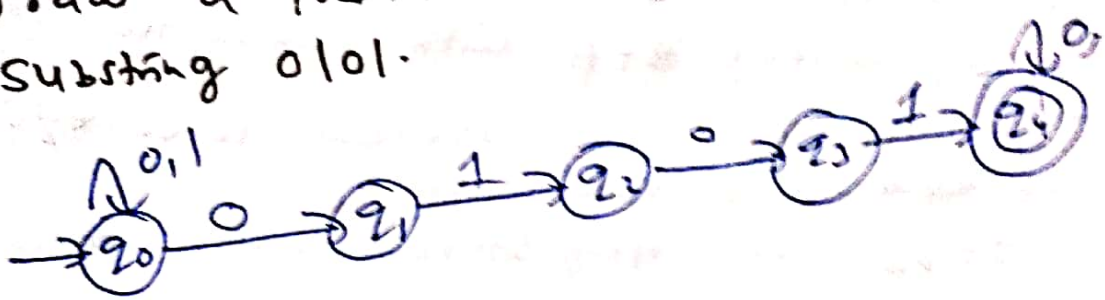
Σ / δ	0	1	2
→ q0	q0	q1	q2
q1	q3	q0	q1
q2	q2	q3	q0
q3	q1	q2	q3

Q:5 Design DFA for set of all strings over [a,b] ending in ab or ba.

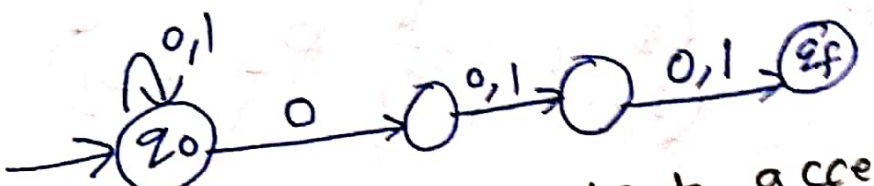


Design of NFA:

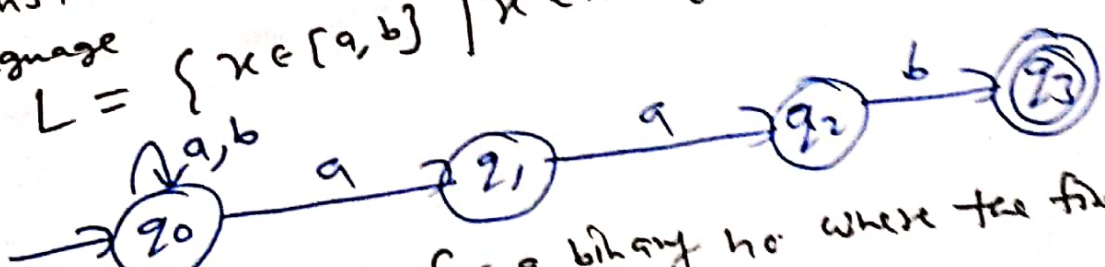
(1) Draw a NDFA to accept strings containing substring 0101.



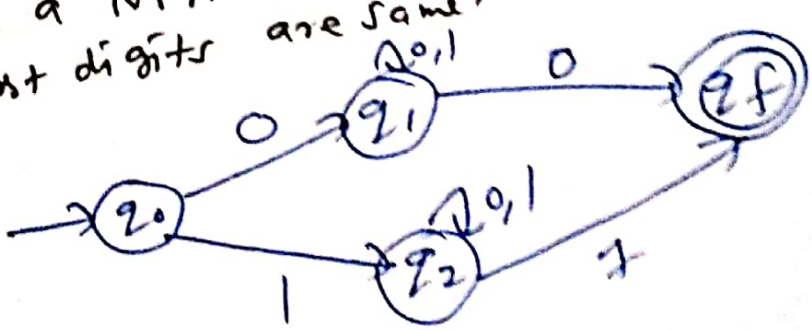
(2) Draw a NFA to accept strings over alphabet [0,1] such that third symbol from the right end is 0.



(3) Construct a NFA that accepts the following Language
$L = \{ x \in [a,b] \mid x$ ending with aab$\}$



(4) Design a NFA for a binary no where the first and last digits are same.



Minimization of DFA

* Minimization of DFA is a process of Constructi an equivalent DFA with minimum No. of States
* Two states are said to be equivalent, if either both states goes to final states or non-final states for a given input.

Minimize the given DFA.

State \ Σ	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Ans First determine 0-equivalence (Π_0) by separting final and non-final states.

$\Pi_0 = \{ \{q_2\}_{\text{final}}, \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}_{\text{Non-final}} \}$ — (1)

Since final state q_2 have no other states in its group So we can not check its equivalence.

Now Compare q_0 with q_1, q_3, q_4, q_5, q_6 and q_7 in other group to check equivalence.

[q_0 with q_1] → q_0 & q_1 have (q_1, q_6) on 0-input and (q_2, q_5) on 1-input.

(q_2 → final state, q_5 → non-final state; q_1 → non-final state, q_6 → non-final state)

since on 1-input, q_2 is final state
q_5 is non-final state

So $[q_0, q_1]$ is not equivalent

Compare [(q0 with q3)]

$q_0 \xrightarrow{0} q_1$ → Non-final
$q_3 \xrightarrow{0} q_2$ → final
] Not equivalent

$q_0 \xrightarrow{1} q_5$ → Non final
$q_3 \xrightarrow{1} q_6$ → Non final
] equivalent

since for 0-Input, they are not equivalent
so (q_0, q_3) are not equivalent.

Compare q_0 with q_4

$q_0 \xrightarrow{0} q_1$
$q_4 \xrightarrow{0} q_7$
] Non-final state] equi-valent

$q_0 \xrightarrow{1} q_5$
$q_4 \xrightarrow{1} q_5$
] Non-final state] equivalent

since (q_0, q_4) are equivalent for 0-Input and 1-Input.

so (q_0, q_4) will be equivalent

Compare q_0 with q_5

$q_0 \xrightarrow{0} q_1$ → Non-final
$q_5 \xrightarrow{0} q_2$ → final
] Not equivalent

$q_0 \xrightarrow{1} q_5$ → Non-final
$q_5 \xrightarrow{1} q_6$ → Non-final
] equivalent

for (0-Input), they are not equivalent
so q_0 is not equivalent to q_5
i.e. (q_0, q_5) are not equivalent.

Compare q_0 with q_6

$q_0 \xrightarrow{0} q_1$
$q_6 \xrightarrow{0} q_6$
] Non-final equivalent

$q_0 \xrightarrow{1} q_5$
$q_6 \xrightarrow{1} q_4$
] Non-final equivalent

so (q_0, q_6) are equivalent

Compare q_0 with q_7

$q_0 \xrightarrow{0} q_1$
$q_7 \xrightarrow{0} q_6$
] Non-final → equivalent

$q_0 \xrightarrow{1} q_5$ → Non-final
$q_7 \xrightarrow{1} q_2$ → final
] Not equivalent

So (q_0, q_7) are not-equivalent

since (q_0, q_4) and (q_0, q_6) are equivalent
so are in the same group

$\Pi_1 = \{(q_1), (q_0, q_4, q_6)\}$

Now Compare

q_1 with q_3, q_4, q_5, q_6 and q_7

(q_1, q_3)

$q_1 \xrightarrow{0} q_6 \rightarrow$ Non-final
$q_3 \xrightarrow{0} q_2 \rightarrow$ final
] Not-equivalent

$q_1 \xrightarrow{1} q_2 \rightarrow$ final
$q_3 \xrightarrow{1} q_6 \rightarrow$ Non-final
] Not equivalent

So (q_1, q_3) are not equivalent

q_1, q_4

$q_1 \xrightarrow{0} q_6 \rightarrow$ Non-final
$q_4 \xrightarrow{0} q_7 \rightarrow$ Non-final
] equivalent

$q_1 \xrightarrow{1} q_2 \rightarrow$ final
$q_4 \xrightarrow{1} q_5 \rightarrow$ Non-final
] Not-equivalent

So (q_1, q_4) are not equivalent

q_1, q_5

$q_1 \xrightarrow{0} q_6 \rightarrow$ Non final
$q_5 \xrightarrow{0} q_2 \rightarrow$ final
] Not-equivalent

$q_1 \xrightarrow{1} q_2 \rightarrow$ final
$q_5 \xrightarrow{1} q_6 \rightarrow$ Non-final
] Not-equivalent

q_1, q_7

$q_1 \xrightarrow{0} q_6 \rightarrow$ Non final
$q_7 \xrightarrow{0} q_6 \rightarrow$ Non final
] equivalent

$q_1 \xrightarrow{1} q_2 \rightarrow$ final
$q_7 \xrightarrow{1} q_2 \rightarrow$ final
] equivalent

So (q_1, q_7) are equivalent

~~Updated Π_1 of q_0, q_4, q_6~~

Updated $\Pi_1 = \{(q_2], \{q_0, q_4, q_6\}, \{q_1, q_7]\}$

Now Compare

q_3 with q_4, q_5, q_6 and q_7

q_3, q_4 → Not equivalent

q_3, q_5 → equivalent

q_3, q_6 → Not-equivalent

q_3, q_7 → Not equivalent

Updated $\Pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$ (27)

Now Compare q_4 with q_5, q_6, q_7

(q_4, q_5) → Not equivalent

(q_4, q_6) → equivalent

(q_4, q_7) → Not equivalent

Updated $\Pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

Now Compare q_5 with q_6, q_7

(q_5, q_6) → Not equivalent

(q_5, q_7) → Not equivalent

Updated $\Pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

Now Compare q_6 with q_7

(q_6, q_7) → Not equivalent

Updated $\Pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

$\Pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$ (2)

Now we will Compare among groups created in Π_1.

$\{q_2\}$ → No other state to Compare.

(q_0, q_4, q_6) → we will Compare q_0 with (q_4, q_6)

(q_0, q_4) → equivalent

(q_0, q_6) → Not equivalent

$\Pi_2 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

Compare q_4 with q_6 (q_4, q_6) → Not equivalent

Updated Π_2
$= \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

Compare q_1 with q_7

$[q_1, q_7]$ → equivalent

Updated Π_2
$= \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}\ \{q_3, q_5\}\}$

Now Compare q_3 and q_5

(q_3, q_5) → equivalent

$\Pi_2 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

Now Compare among group created in Π_2

q_2 → does not have any other member to Compare.

(q_0, q_4) → equivalent

Updated $\Pi_3 = (\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\})$

q_6 → does not have other member to Compare

(q_1, q_7) → equivalent

No change in Π_3

Updated $\Pi_3 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$

$\{q_3, q_5\}$ → equivalent

Updated $\Pi_3 = \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}\ \{q_3, q_5\}\}$

Examples!

Since $\Pi_2 = \Pi_3$

So we will stop the process

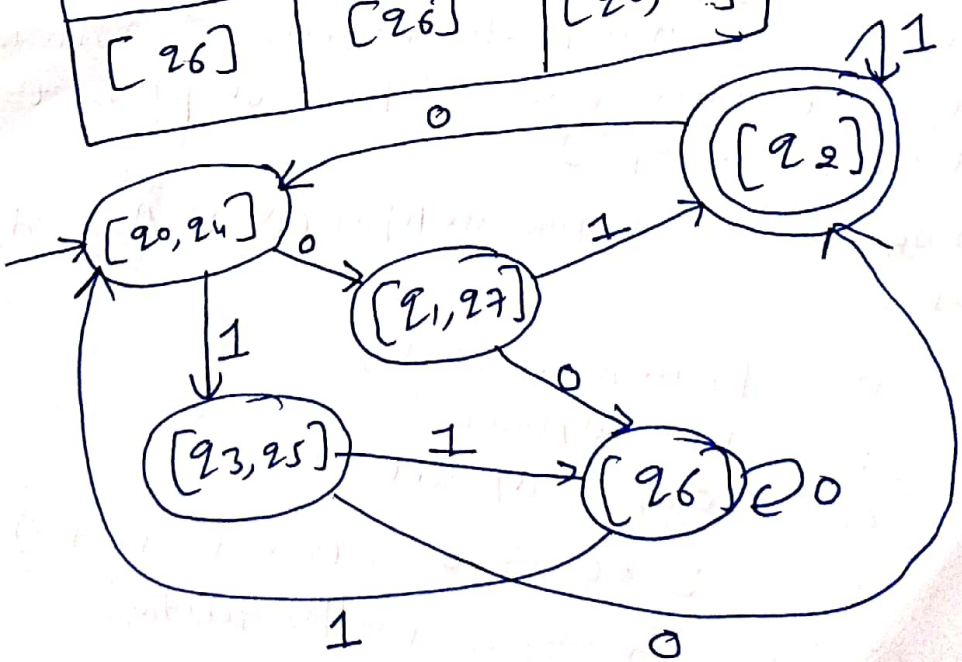
So minimized DFA has states

$\{[q_2], [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]\}$

initial state = $[q_0, q_4]$

final state = $[q_2]$

Q \ Σ	0	1
→$[q_0, q_4]$	$[q_1, q_7]$	$[q_3, q_5]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
(*)$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$



Finite Automata with outputs

~~(1) Moore~~

(1) Mealy Machine

(2) Moore Machine

Mealy Machine: Mealy Machine is a finite automata in which the output function depends on both present state and present input.

A mealy machine is a six tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where

$Q \Rightarrow$ Finite set of states

$\Sigma \Rightarrow$ input alphabet

$\Delta \Rightarrow$ output alphabet

$\delta \Rightarrow$ $\Sigma \times Q \rightarrow Q$ (transition function)

$\lambda \Rightarrow$ $Q \rightarrow \Delta$ (output function)

$q_0 \Rightarrow$ initial state

Moore Machine:

Moore Machine is a Finite Automata in which the output function only depends on present state (not on present input.)

A moore Machine has six tuples $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$

where

Q : Finite No. of states

Σ : input alphabet

Δ : output alphabet

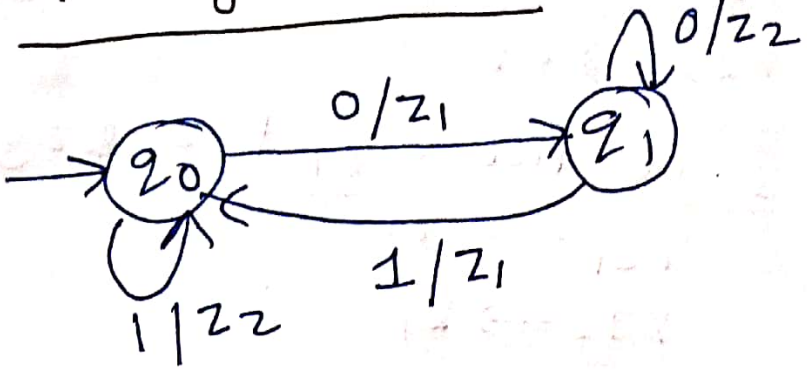
δ : $\Sigma \times Q \rightarrow Q$ (transition function)

λ : $Q \rightarrow \Delta$ (output function)

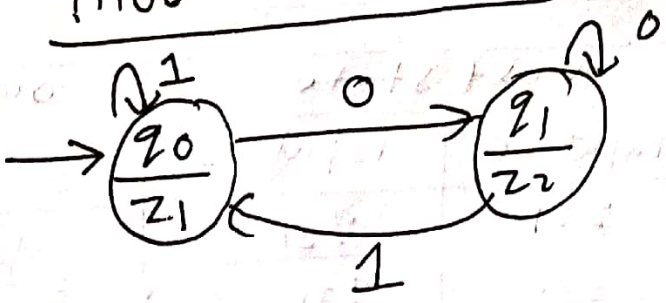
q_0 : initial state

Examples:

Mealy Machine:



Moore Machine



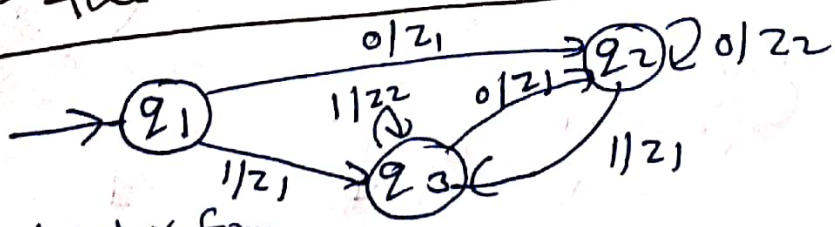
Equivalence of Mealy Machine & Moore Machine

(i) Conversion from mealy machine to moore machine.

(ii) Conversion from moore machine to mealy machine

(i) Mealy Machine to Moore Machine Conversion

Convert the following mealy machine to moore machine



Writing in tabular form

Present State (PS)	Next State (NS)			
	Input = 0		Input = 1	
	NS	output	NS	output
→ q_1	q_2	z_1	q_3	z_1
q_2	q_2	z_2	q_3	z_1
q_3	q_2	z_1	q_3	z_2

q_2 has two output z_1 & z_2

$q_{21} \rightarrow z_1$

$q_{22} \rightarrow z_2$

q_3 has two output z_1 & z_3

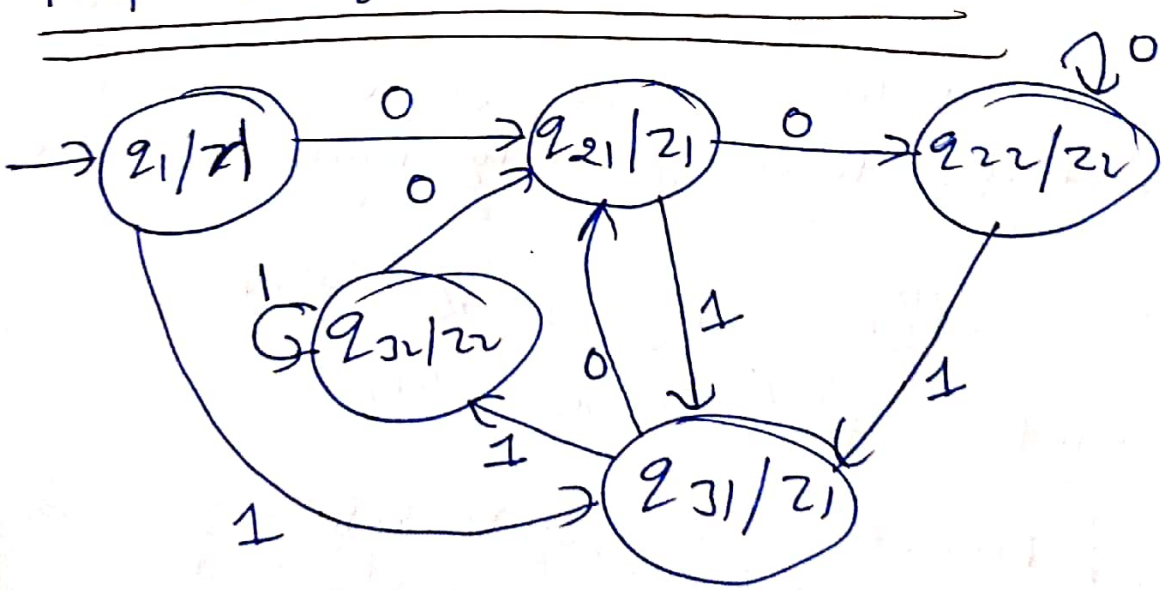
$q_{31} \rightarrow z_1$

$q_{32} \rightarrow z_2$

Since q_1 have no output, consider as λ

Present State	Next State		output
	input = 0	input = 1	
→ q_1	q_{21}	q_{31}	λ
q_{21}	q_{22}	q_{31}	z_1
q_{22}	q_{22}	q_{31}	z_2
q_{31}	q_{21}	q_{32}	z_1
q_{32}	q_{21}	q_{32}	z_2

Representing in transition diagram



Moore Machine

(ii) Conversion from Moore Machine to Mealy Machine

Construct a Mealy machine equivalent to Moore Machine given by following table:

Present state (PS)	Next State (NS)		Output
	a = 0	a = 1	
→ q_0	q_3	q_1	0
q_1	q_1	q_2	1
q_2	q_2	q_3	0
q_3	q_3	q_0	0

q_0 has output 0
q_1 has output 1
q_2 has output 0
q_3 has output 0

Present state (PS)	Next State (NS)			
	a = 0		a = 1	
	State	output	State	output
→ q_0	q_3	0	q_1	1
q_1	q_1	1	q_2	0
q_2	q_2	0	q_3	0
q_3	q_3	0	q_0	0

Note: If two present states are showing same next state and output for the given inputs, then they are equivalent and we can remove any one of them.

Difference between Mealy Machine & Moore Machine

Mealy Machine	Moore Machine
① Output depends on present state and present input.	① output depends only on the present state.
② Less No. of states are required	② More No. of States are required.
③ They reacts faster to Inputs	③ They reacts Slower to Inputs.
④ More hardware require for circuit implementation	④ Less hardware require for circuit implementation
⑤ output is placed on transitions	⑤ output is placed on states.
⑥ It is difficult to design	⑥ It is easy to design.
⑦ output length = Input length	⑦ output length = Input length + 1

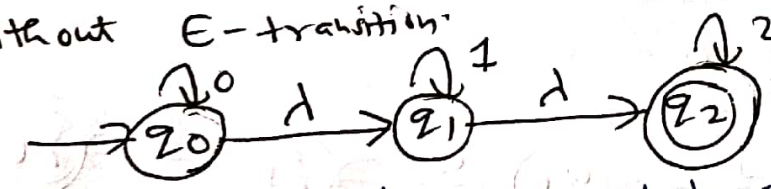
NFA with ϵ-transition (λ-transition)

A Non-deterministic Finite Automata (NFA) which perform transition on null string (ϵ/λ) is called NFA with ϵ-transition (λ-transition). i.e. transition is possible when no-input is applied.

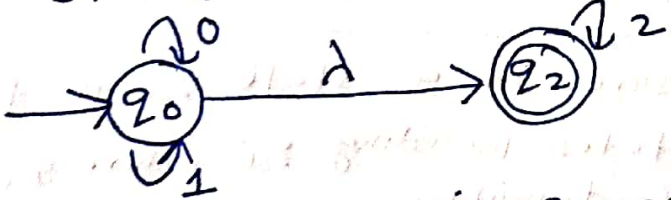
It is possible to convert NFA with ϵ-transition to NFA without transition.

Conversion of NFA with ϵ-transition to NFA without ϵ-transition.

Example-1 for the given NFA with ϵ-transition given in below figure, find an equivalent NFA without ϵ-transition.



Ans **Step-1** Since q_0 is connected to q_1 through λ-transition, so eliminate q_1 and show all the transition of q_1 on q_0. Since q_1 is also non-final state, so no change in type of state (i.e. q_0 is still Non-final state)

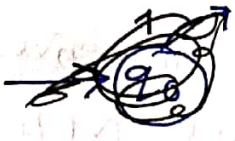


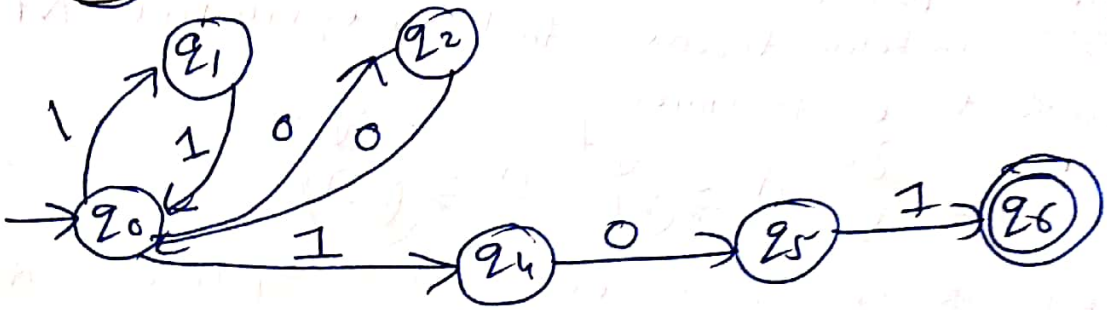
Step-2 Since q_1 is connected to q_2 with λ-transition, so eliminate q_2 and show all transition of q_2 on q_0. Since q_2 is a final state, so q_0 now becomes final state.

Example 2

Convert the following NFA with ϵ-transition to NFA without transition.

Ans Since q_0 is connected to q_3 through λ-transition, So Connect all q_3 transition to q_0.

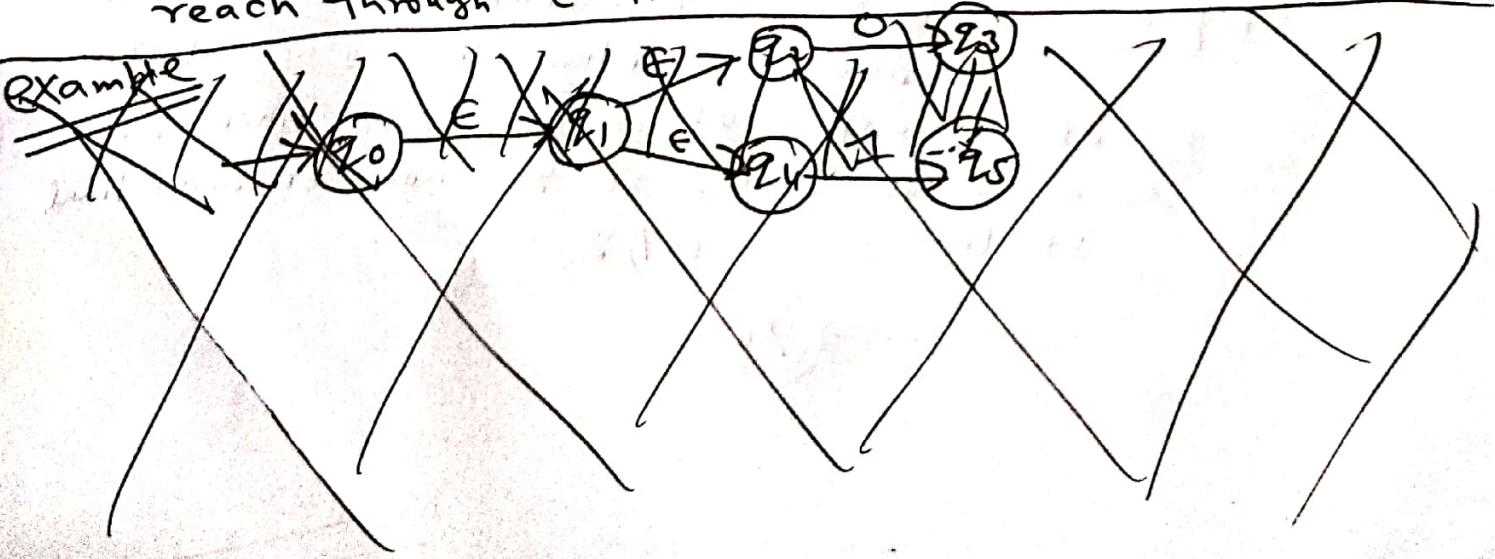




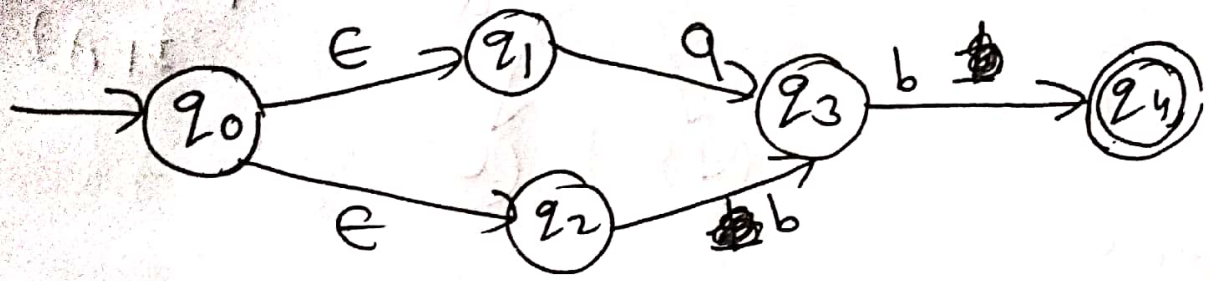
Conversion of NFA with ϵ-transition to DFA using ϵ-closure method

ϵ-closure: ϵ-closure of a state q_i is the set of states including q_i where q_i can reach through ϵ-transition.

Example



Find the DFA for the given NFA with ϵ-transition using ϵ- closure method.



Ans **ϵ-closure:** specify the total no. of states that are reachable from a given states on ϵ-transition.

$A = \epsilon\text{-closure}(q_0) = \epsilon\text{-closure}(0)$

for simplicity $\{q_0=0, q_1=1, q_2=2, q_3=3, q_4=4\}$

$A = \epsilon\text{-closure}(0)$

$A = \{0,1,2\}$

for A

$\delta(A, a) = \epsilon\text{-closure}(A, a)$

$= \epsilon\text{-closure}((0,1,2), a)$

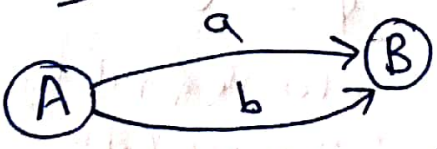
$= \epsilon\text{-closure}(3) = \{3\} = B$ (state B)

$\delta(A, b) = \epsilon\text{-closure}(A, b)$

$= \epsilon\text{-closure}((0,1,2), b)$

$= \epsilon\text{-closure}(3) = \{3\} = B$

i.e.



(Partial DFA)

for B $\delta(B, a) = \epsilon\text{-closure}(B, a)$

$= \epsilon\text{-closure}(\{3\}, a) = \epsilon\text{-closure}(\phi)$

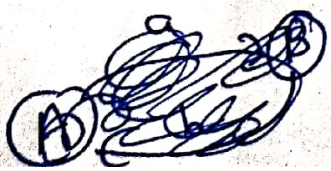
$= \{\phi\} = C$ (state C)

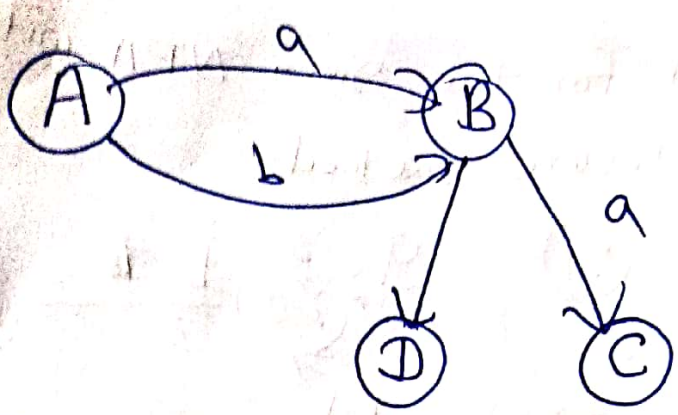
$\delta(B, b) = \epsilon\text{-closure}(B, b)$

$= \epsilon\text{-closure}(3, b)$

$= \epsilon\text{-closure}(4) = \{4\} = D$

= state D





(Partial DFA)

for C

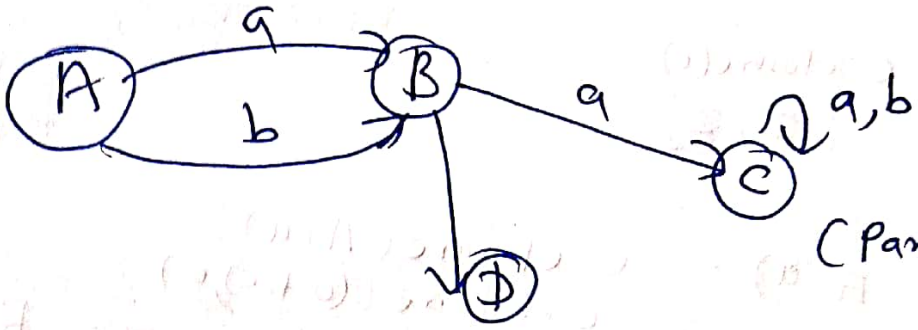
$\delta(C, a) = \epsilon\text{-closure}(\{\phi\}, a)$

$= \phi \; \epsilon\text{-closure}(\phi)$

$= \phi = C$

$\delta(C, b) = \epsilon\text{-closure}(\{\phi\}, b)$

$= \epsilon\text{-closure}(\phi) = \phi = C$



(Partial DFA)

for D

$\delta(D, a) = \epsilon\text{-closure}(\{\}, a)$

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

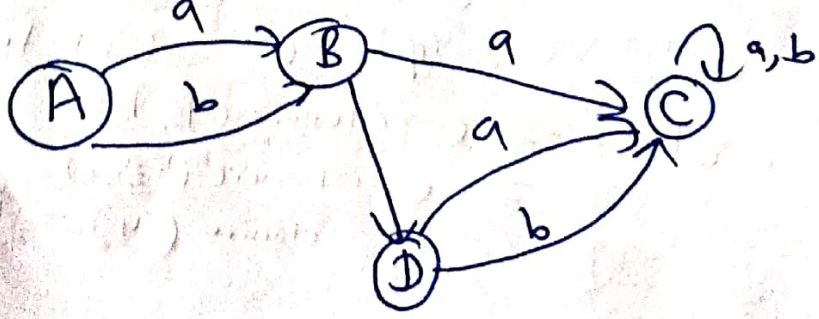
$= \phi = C$

$\delta(D, b) = \epsilon\text{-closure}(\{\}, b)$

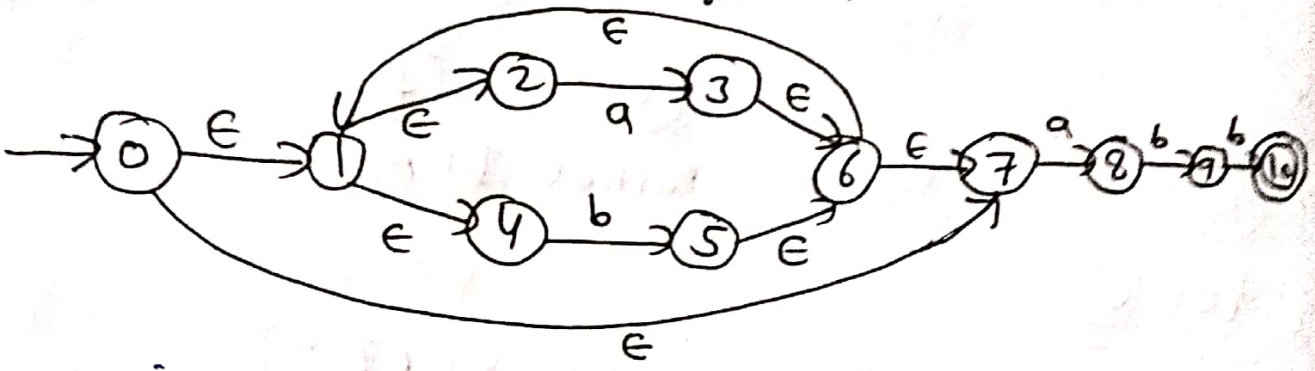
$= \epsilon\text{-closure}(\phi)$

$= \phi = C$

Final DFA



Find the DFA using ϵ-closure method for the given NFA with ϵ-transitions



Ans

A = initial state = ϵ-closure(0)

A = {0,1,2,4,7} — (1)

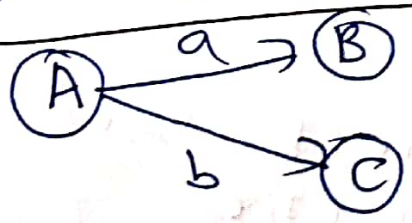
for A

δ(A,a) = ϵ-closure({0,1,2,4,7}, a)
= ϵ-closure(3,8)
= ϵ-closure(3) ∪ ϵ-closure(8)
= {1,2,3,4,6,7} ∪ {8}

δ(A,a) = {1,2,3,4,6,7,8} = B — (2)

δ(A,b) = ϵ-closure({0,1,2,4,7}, b)
= ϵ-closure(5)

δ(A,b) = {1,2,4,5,6,7} = C — (3)



(Partial DFA)

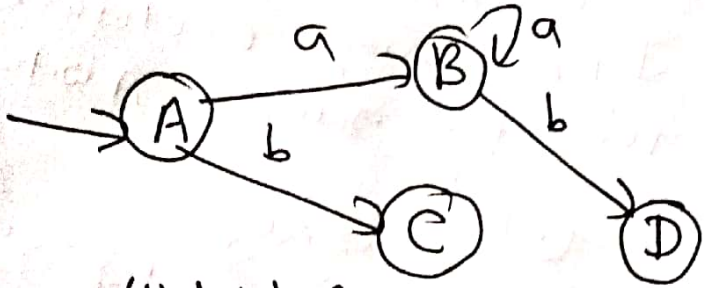
for B

δ(B,a) = ϵ-closure({1,2,3,4,6,7,8}, a)
= ϵ-closure(3,8)
= ϵ-closure(3) ∪ ϵ-closure(8)
= {1,2,3,4,6,7} ∪ {8}

δ(B,a) = {1,2,3,4,6,7,8} = B

δ(B,b) = ϵ-closure({1,2,3,4,6,7,8}, b)
= ϵ-closure(5,9) = ϵ-closure(5) ∪ ϵ-closure(9)
= {1,2,4,5,6,7} ∪ {9}

δ(B,b) = {1,2,4,5,6,7,9} = D — (4)



Updated (Partial DFA)

for C

$\delta(C, a) = \epsilon\text{-closure}(\{1,2,4,5,6,7\}, a)$

$= \epsilon\text{-closure}(3,8) = \{1,2,3,4,6,7,8\}$

$\boxed{\delta(C,a) = B}$

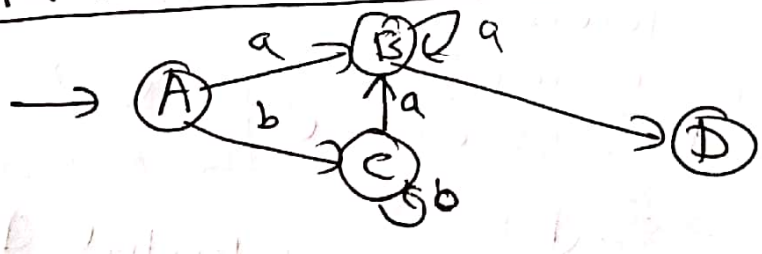
$\delta(C, b) = \epsilon\text{-closure}(\{1,2,4,5,6,7\}, b)$

$= \epsilon\text{-closure}(5)$

$= \{1,2,4,5,6,7\}$

$\boxed{\delta(C,b) = C}$

Updated Partial DFA



for D

$\delta(D, a) = \epsilon\text{-closure}(\{1,2,4,5,6,7,9\}, a)$

$= \epsilon\text{-closure}(3,8)$

$= \{1,2,3,4,6,7,8\}$

$\boxed{\delta(D,a) = B}$

$\delta(D, b) = \epsilon\text{-closure}(\{1,2,4,5,6,7,9\}, b)$

$= \epsilon\text{-closure}(5,10)$

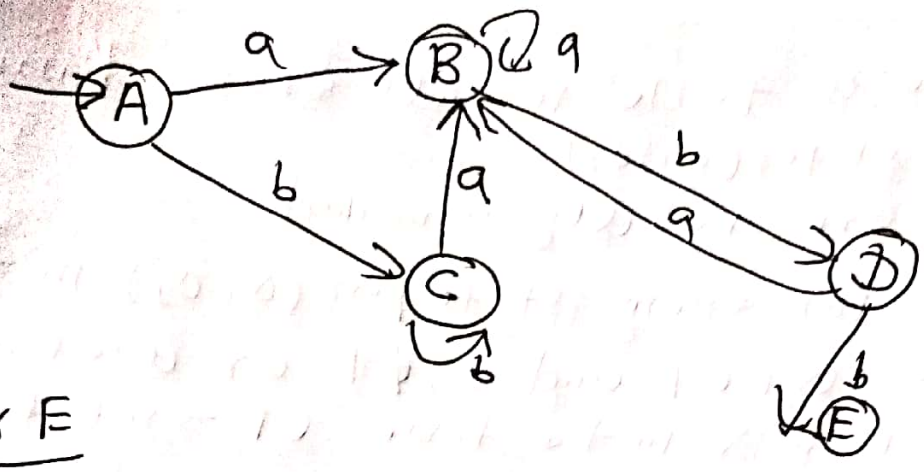
$= \epsilon\text{-closure}(5) \cup \epsilon\text{-closure}(10)$

$= \{1,2,4,5,6,7\} \cup \epsilon\text{-closure}(10)$

$= \{1,2,4,5,6,7\} \cup \{10\}$

$\boxed{\delta(D,b) = \{1,2,4,5,6,7,10\} = E}$ — (5)

Updated Partial DFA is



for E

$\delta(E, a) = \epsilon\text{-closure}(\{1,2,4,5,6,7,10\}, a)$

$= \epsilon\text{-closure}(3,8)$

$= \{1,2,3,4,6,7,8\}$

$\boxed{\delta(E,a) = B}$

$\delta(E, b) = \epsilon\text{-closure}(\{1,2,4,5,6,7,10\}, b)$

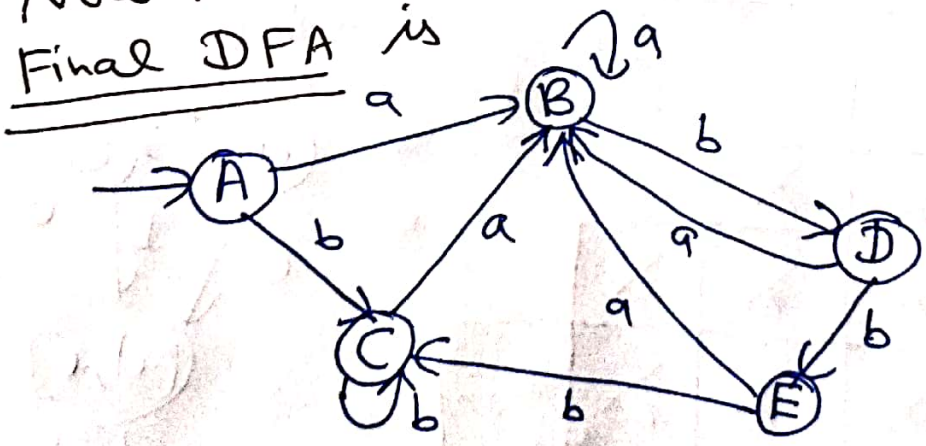
$= \epsilon\text{-closure}(5)$

$\boxed{\delta(E,b) = \{1,2,4,5,6,7\}}$

$\boxed{\delta(E,b) = C}$

Now No new state, So stop the process

Final DFA is



Final DFA

(42) DFA Minimization using Myhill Nyrode theorem

(1) Draw a table for all ~~states~~ pairs of states (Q_i, Q_j).

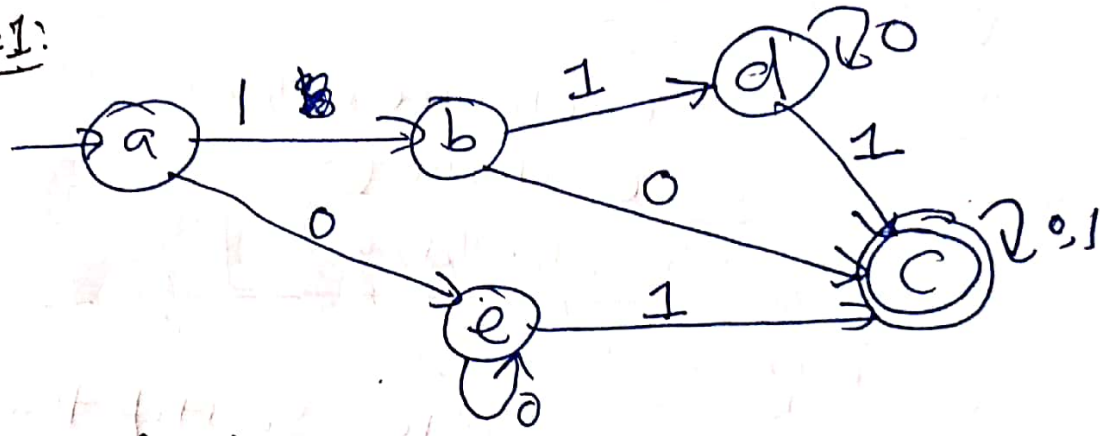
(2) All are initially unmarked.

(3) Consider every state pair (Q_i, Q_j) in the DFA where $Q_i \in F$ and $Q_j \notin F$ or (vice-versa and mark them 'X' in the table. (F → set of final states).

(4) Repeat this step until we can not mark anymore states.

(5) Combine all the unmarked pair (Q_i, Q_j) and mark them a single state in the reduced DFA.

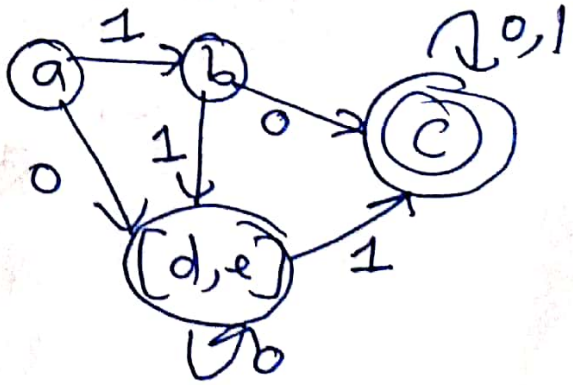
Example1:



Equivalent table

	a	b	c	d
b	X			
c	X	X		
d	X	X	X	
e	X	X	X	✓

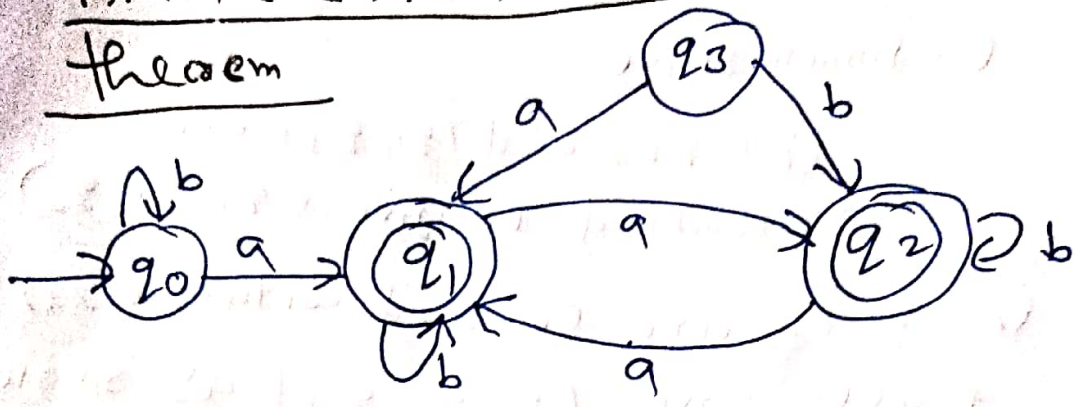
equivalent pair = [d, e]



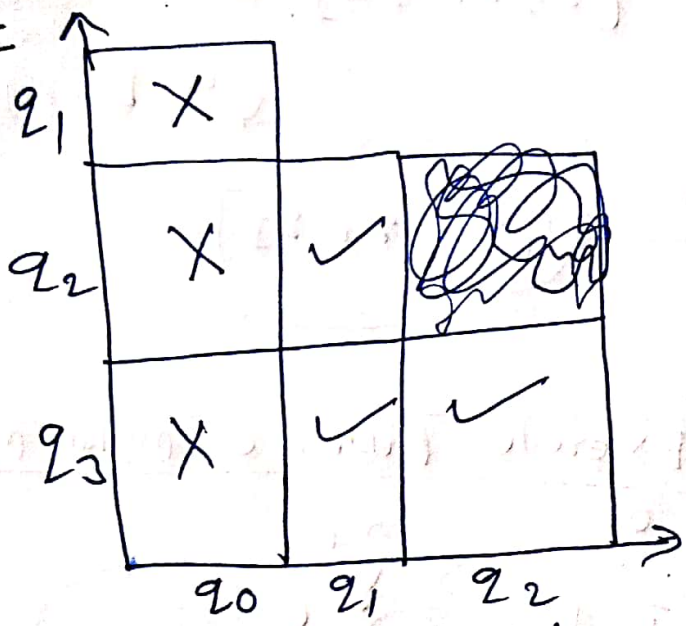
Reduced DFA

Example: 2

Minimize DFA ~~for~~ using Myhill Nyrode theorem



Ans



(q_0, q_1) $q_0 \xrightarrow{a} q_1 \in F$; $q_0 \xrightarrow{b} q_0 \notin F$ } so put 'X'
$q_1 \xrightarrow{a} q_2 \in F$, $q_1 \xrightarrow{b} q_1 \in F$

(q_0, q_2) $q_0 \xrightarrow{a} q_1 \in F$, $q_0 \xrightarrow{b} q_0 \notin F$ } so put 'X'
$q_2 \xrightarrow{a} q_1 \in F$, $q_2 \xrightarrow{b} q_2 \in F$

(q_0, q_3) $q_0 \xrightarrow{a} q_1 \in F$, $q_0 \xrightarrow{b} q_0 \notin F$ } so put 'X'
$q_3 \xrightarrow{a} q_1 \in F$, $q_3 \xrightarrow{b} q_2 \in F$

(q_1, q_2) $q_1 \xrightarrow{a} q_2 \in F$ $q_1 \xrightarrow{b} q_1 \in F$ } so put '✓'
$q_2 \xrightarrow{a} q_1 \in F$ $q_2 \xrightarrow{b} q_2 \in F$

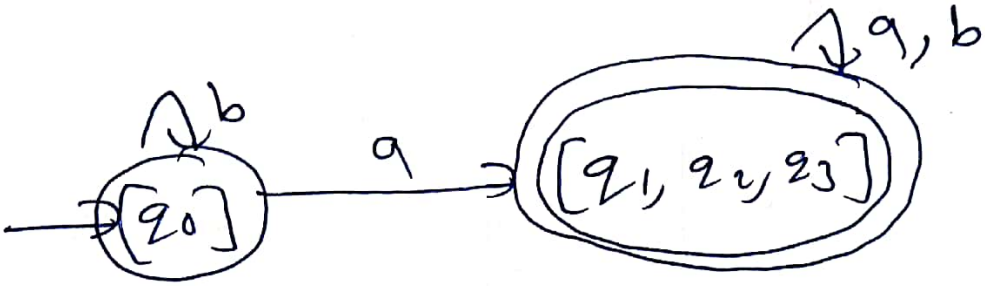
(q_1, q_3) $q_1 \xrightarrow{a} q_2 \in F$ $q_1 \xrightarrow{b} q_1 \in F$ } so put '✓'
$q_3 \xrightarrow{a} q_1 \in F$ $q_3 \xrightarrow{b} q_2 \in F$

(q_2, q_3) $q_2 \xrightarrow{a} q_1 \in F$ $q_2 \xrightarrow{b} q_2 \in F$ } so put ✓
$q_3 \xrightarrow{a} q_1 \in F$, $q_3 \xrightarrow{b} q_2 \in F$

(44) Since (q_1, q_2), (q_1, q_3) and (q_2, q_3) have ('—') so Recombine them if have Common state

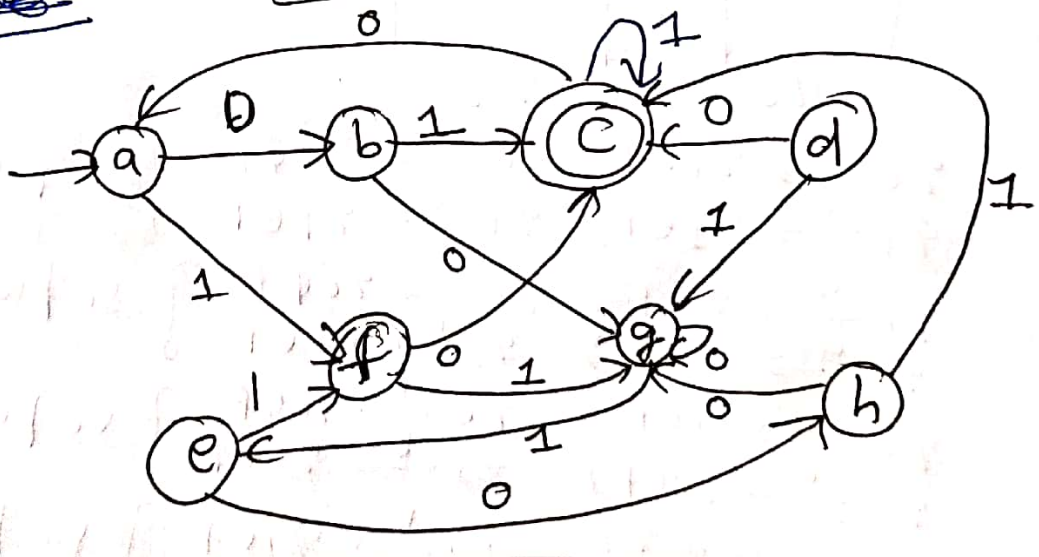
(q_1, q_2) and (q_1, q_3) are recombined to give (q_1, q_2, q_3)

(q_1, q_2, q_3) are combined with (q_2, q_3) to finally give $[q_1, q_2, q_3]$ as single state



Excercise

Excercive Problem on Mylill Minimizatn



Answer