Theory Of Computation

Unit – I

INTRODUCTION:

- **Automata** theory (also known as **Theory Of Computation**) is a theoretical branch of Computer Science and Mathematics, which mainly deals with how efficiently the problem can be solved on a model of computation using an appropriate algorithm.
- Automata enables the scientists to understand how machines compute the functions and solve problems. The main motivation behind developing Automata Theory was to develop methods to describe and analyze the dynamic behavior of discrete systems.

- Automata is originated from the word “Automaton” which is closely related to “Automation”.
- Now, let’s understand the basic terminologies, which are important and frequently used in Theory of Computation.
  - Symbol
  - Alphabet
  - String
    - Length of String
    - Empty String
  - Language

- **Symbol:** Symbol(often also called **character**) is the smallest building block, which can be any alphabet, letter or any picture.

a, b, c, 0, 1, ...

- **Alphabets (Σ):** Alphabets are set of symbols, which are always ***finite***.

Examples:

$\Sigma = \{0, 1\}$ is an alphabet of binary digits

$\Sigma = \{0, 1, \ldots\ldots, 9\}$ is an alphabet of decimal digits

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

$\Sigma = \{A, B, C, \ldots\ldots\ldots, Z\}$

- **String:** String is a ***finite*** sequence of symbols from some alphabet. String is generally denoted as ***w*.**
  - Length of a string is the number of occurrence of symbols that might appear in the string. Length of a string is denoted as |w|.
  - Empty string is a special string that's length is 0. Empty string is denoted by ε.

Number of Strings (of length 2) that can be generated over the alphabet {a, b} -

-	-
a	a
a	b
b	a
b	b

Length of String $|w| = 2$

Number of Strings = 4

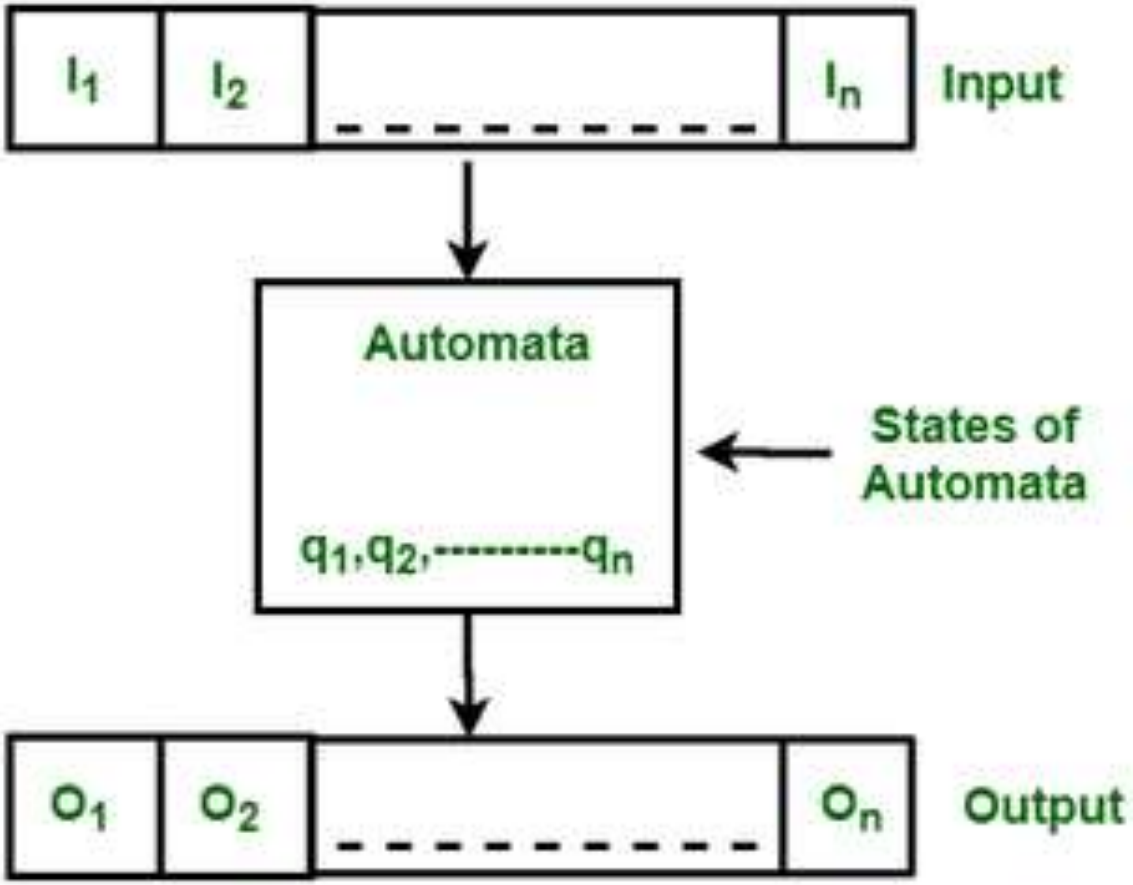
Conclusion:

For alphabet {a, b} with length **n**, number of strings can be generated = $\mathbf{2^n}$.

- **Language:**
- A language is a collection of words that arranged in some well defined manner.
- A language is a *set of strings*, chosen from some Σ* or we can say- ‘A language is a subset of Σ* ‘. A language which can be formed over ‘ Σ ‘ can be **Finite** or **Infinite**.

AUTOMATON:

- The word automaton is derived from the Greek word automata that means 'to automate' or 'self-acting'. An automaton can be defined as a system that perform certain functions without direct participation of man. It accepts energy, material and (or) information as an input and convert them into the final product under the guidance of control signal.
  - Ex. – automatic photo printing machine, automatic packing machine etc.



Basic structure of an Automaton

The figure shows following features of automata:

- Input
- Output
- States of automata
- State relation
- Output relation

- **Input:** $\sum$ is the input alphabet that contains all the input values I_0, I_1, I_2, I_3, I_4,..... I_n. At each discrete instance of time t_0, t_1, t_2,.....t_n these input values are applied to the input side of the model that is shown in the figure.
- **Output:** The output variable O contains all the output values that are O_0, O_1, O_2,..... O_n.

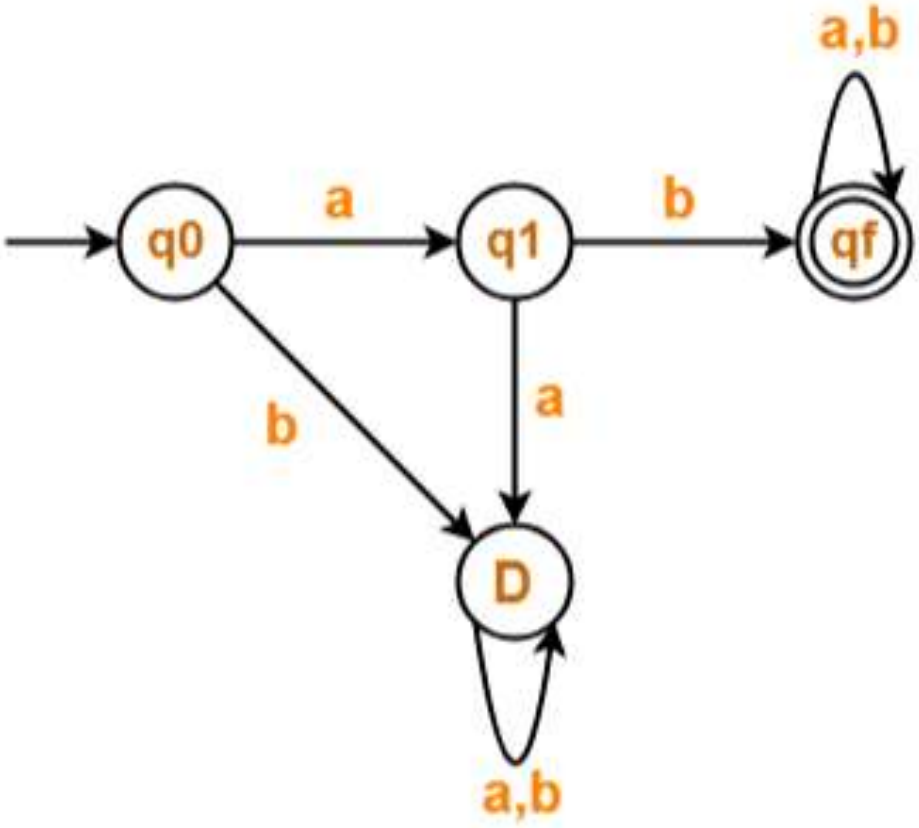
- **State:** At each instant of time the automaton can be in one of the state q_0, q_1, q_2,..... q_n.
- **State Relation:** The next state of an automaton at any instant of time is determined by the present state and the present input.
- **Output Relation:** The output is related to either the state only or both the state and the input.

- **There are 4 major families of automaton system that are:**
  - Finite Automata
  - Pushdown Automata
  - Linear-Bounded Automata
  - Turing Machine

Finite automata (FA):

- Finite Automata is a simplest kind of language recognition machine. The finite automata consists of number of states and some basic rules that is used for transition from one state to another.
- Finite Automata has 4 types of states:
  - Initial
  - Intermediate
  - Final
  - Dead

- **Initial:** It is the starting state of FA. Input enters from this state and starts transition. This is simply indicated by an arrow.
- **Intermediate:** All the states except the initial and final states where the transition occurs is called an intermediate state.
- **Final:** This state is also called accepting state because it determines whether the input string is accepted or not by the FA.
- **Dead:** This is an optional state.



Sample state diagram for finite automata with dead state

Definition of Finite Automata:

- A finite automaton is a collection of 5-tuple (Q, ∑, δ, q0, F), where:
- Q: finite set of states
- ∑: finite set of the input symbol
- q0: initial state
- F: **final** state
- δ: Transition function

Model of Finite automata:

- Finite automata can be represented by input tape and finite control.
- **Input tape:** It is a linear tape having some number of cells. Each input symbol is placed in each cell.
- **Finite control:** The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read.

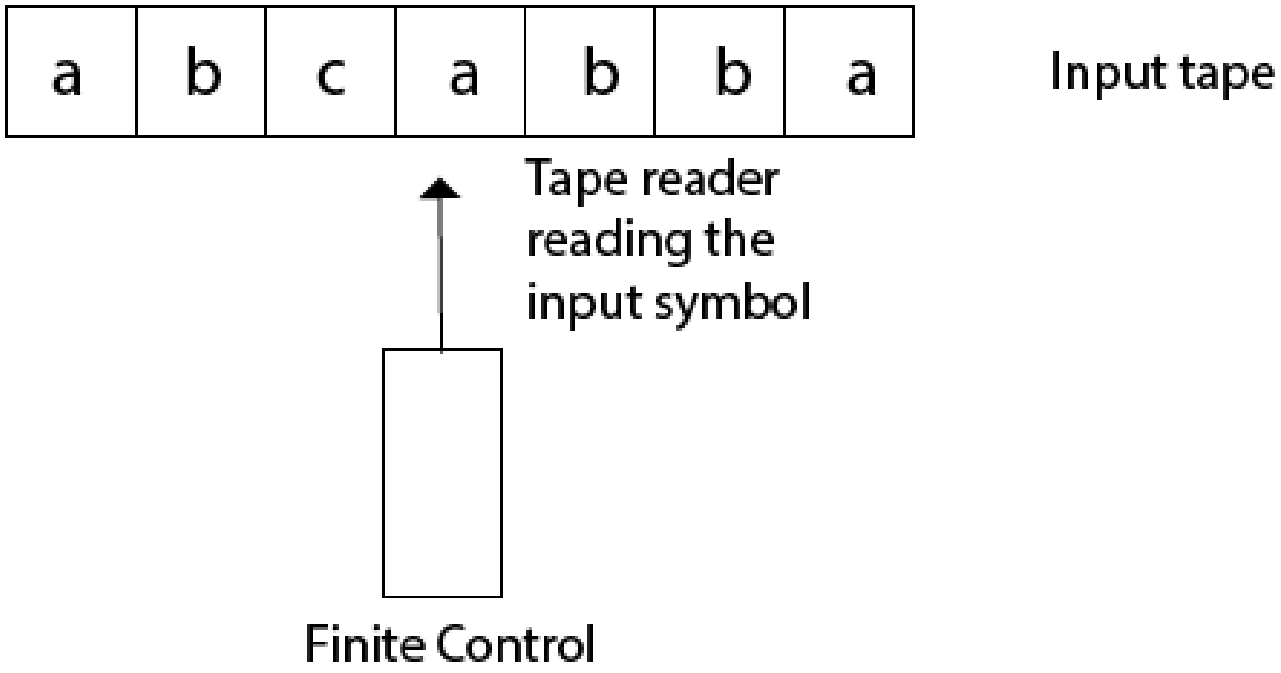
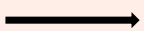
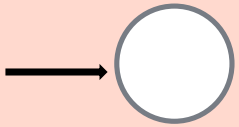


Fig :- Finite automata model

FINITE AUTOMATA NOTATIONS:

S. No	Notation	Meaning
1		Represent a state
2		Represent transition from one state to another
3		Starting state
4		Final state

Finite automata representation:

- FA can be represented by 2 ways:
  - Transition Diagram
  - Transition Table

Transition diagram:

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

- DFA with ∑ = {0, 1} accepts all strings starting with 1.

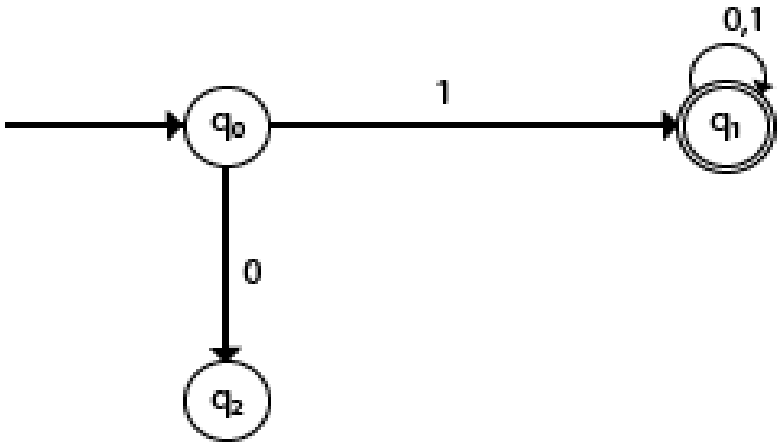


Fig: Transition diagram

Transition table:

- The transition table is basically a tabular representation of the transition function. It takes two arguments (a state and a symbol) and returns a state (the "next state").
- A transition table is represented by the following things:
  - Columns correspond to input symbols.
  - Rows correspond to states.
  - Entries correspond to the next state.
  - The start state is denoted by an arrow with no source.
  - The accept state is denoted by a star or enclosed in a double circle.

Acceptability of String by Finite Automata :-

A string w is accepted by the finite automata (Machine) $M = (Q, \Sigma, \delta, q_0, F)$ if it ends at final state q_f. The language accepted by the machine M is denoted by $L(M)$.

ex.(5) Construct a FA where $Q = \{q_0, q_1, q_2, q_3, q_f\}$, $\Sigma = \{a, b\}$ and $F = \{q_f\}$ The transition function δ of the above automata is :-

$\delta(q_0, a) = q_1$
$\delta(q_0, b) = q_2$
$\delta(q_1, a) = q_3$
$\delta(q_1, b) = q_0$
$\delta(q_2, a) = q_2$
$\delta(q_2, b) = q_3$
$\delta(q_3, a) = q_f$
$\delta(q_3, b) = q_2$
$\delta(q_f, a) = q_3$
$\delta(q_f, b) = q_f$

Name of Lecturer : Chary Shree

(a) Given the sequence of states for the acceptability of input string - abbaaba

(b) Check following strings are accepted by the FA or not
(i) abaababa (ii) bbbbb (iii) aaaaa

Solution:-

(a) Input string = abbaaba
starting state = q_0
Thus -

$\delta(q_0, a) = q_1$
$\delta(q_1, b) = q_0$
$\delta(q_0, b) = q_2$
$\delta(q_2, a) = q_2$
$\delta(q_2, a) = q_2$
$\delta(q_2, b) = q_3$
$\delta(q_3, a) = q_f$

The sequence of states for the acceptability of string abbaaba is:-

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_0 \xrightarrow{b} q_2 \xrightarrow{a} q_2 \xrightarrow{a} q_2 \xrightarrow{b} q_3 \xrightarrow{a} q_f$$

(b) ① starting state = q_0

Input string = abaababa

$= \delta(q_0, abaababa)$

$= \delta(q_1, baababa)$

$= \delta(q_0, ababa)$

$= \delta(q_1, baba)$

$= \delta(q_3, aba)$

$= \delta(q_2, ba)$

$= \delta(q_3, a)$

q_f

Finally we have reached at final state therefore string abaababa is accepted by finite automata.

Name of Lecturer : Charu Shree

(b) (ii) starting state = q_0

input string = b b b b b

$= \delta(q_0, bbbbb)$

$= \delta(q_2, bbbb)$

$= \delta(q_3, bbb)$

$= \delta(q_2, bb)$

$= \delta(q_3, b)$

q_2

At the end of the string we does not reach at final state. Hence string bbbbb is rejected.

(b) (iii) starting state = q_0

input string = a a a a a

$= \delta(q_0, aaaaa)$

$= \delta(q_1, aaaa)$

$= \delta(q_3, aaa)$

$= \delta(q_f, aa)$

$$= \delta(q_3, a)$$
$$\downarrow$$
$$q_f$$

At the end of the string we reach at final state. Hence string aaaaa is accepted by the finite automata.

★ Applications of Finite Automata -

ⓐ Lexical Analyzer - Lexical analysis is a part of the compilation process. In lexical analyzer finite automata is used to recognize the validity of the tokens because it checks whether the given token (smallest individual unit of cod is correct or not.

ⓑ Text Editors - Finite automata is also used for constructing the text editors. Suppose we want to write a statement Application of FA using small text editor. Then all the successive words in the statement are highlighted based on logical prediction for the help of user.

Name of Lecturer : Charu Shree

(c) Spell Checker – In our computer system, there in a dictionary for checking the correctness of a word. There is also a finite automata that is used to recognize the correctness of a word by using the dictionary & give a appropriate suggestions to a user.

(d) Sequential Circuit Design – FA is also used for designing the sequential circuits.

Types of Finite Automata –

There are two types of FA –

1. Deterministic Finite Automata (DFA)
2. Non-deterministic Finite Automa (NDFA/NFA)

1. Deterministic Finite Automata (DFA) –

(1) D (Deterministic) – There is exactly one transition for every input symbol from the current state. So, the machine is deterministic.

(2) F (Finite) – It has finite number of state. so it is called finite

③ A (Automata) - Automata is a machine which may accept or reject the string. Hence it is called DFA.

★ Definition of DFA -

Mathematically DFA is a five-tuple structure written as $M = (Q, \Sigma, \delta, q_0, F)$ where,

ⓐ M - is the name of machine. It can be called by any name.
ⓑ Q - is a finite non empty "set of states."
ⓒ Σ - is a set of "input symbols"
ⓓ δ - is a transition function. It takes 2 inputs - input symbol and current state & return the next state.
ⓔ q_0 - is the initial (starting) state i.e. $q_0 \in Q$.
ⓕ F - is the set of Final state i.e. $F \subseteq Q$.

ex- Design a DFA that accept even no. of a's and any no. of b's over input alphabet $\Sigma = \{a, b\}$.

Solution-

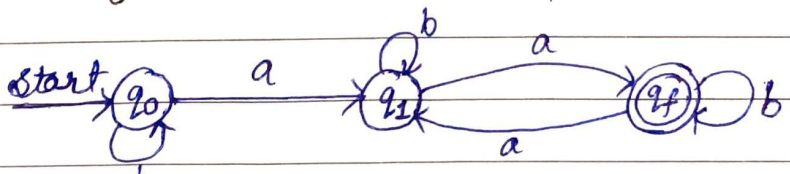
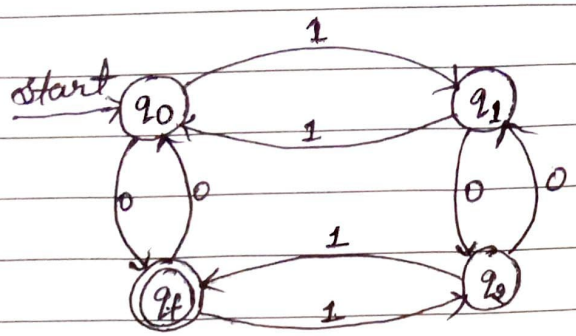


Fig:- Transition diagram

Transition Table :-

State	Input a	Input b
→ q_0	q_1	q_0
q_1	q_f	q_1
(*) q_f	q_1	q_f

ex Consider the DFA shown in the following figure & check acceptability of the string 10111100.



Solution: Starting state = q_0

Input string = 10111100

current state = $\delta(q_0, 10111100)$

$\downarrow$

$= \delta(q_1, 0111100)$

$\downarrow$

$= \delta(q_2, 111100)$

$\downarrow$

$= \delta(q_f, 11100)$

$\downarrow$

$= \delta(q_f, 1100)$

$\downarrow$

$= \delta(q_2, 100)$

$\downarrow$

$= \delta(q_f, 00)$

$\downarrow$

$= \delta(q_0, 0)$

Estd. Yr. 2000
ARYA GROUP OF COLLEGES
AIET ☐ ACERC ☐ AIETM ☐ ACP ☐ APGC ☐
LECTURE NOTES
Excellence in Education
Branch / Faculty: ..... Year / Sem.: ..... Subject: .....
Topic: ..... Unit: ..... Lecture No. .....

$= \downarrow$

$= q_f$

$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{1} q_f \xrightarrow{1} q_2 \xrightarrow{1} q_f \xrightarrow{1} q_2 \xrightarrow{1} q_f \xrightarrow{0} q_0 \xrightarrow{0} q_f$

After processing the string 10111100 we have reached the final state q_f. Hence the string 10111100 is accepted by given DFA.

Ex Design a DFA which accepts only those string that starts with 1 and always ends with 00 over alphabet $\Sigma = \{0, 1\}$

Solution-

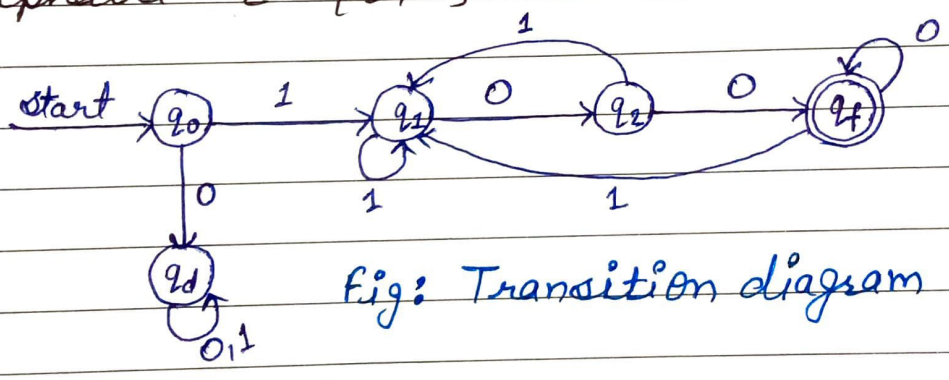


Fig: Transition diagram

Transition Table:-

state	Input	
	0	1
$\rightarrow q_0$	q_d	q_1
q_1	q_2	q_1
q_2	q_f	q_1
q_d	q_d	q_d
(q_f)	q_f	q_1

Name of Lecturer : Charu Shree

② Non deterministic Finite Automata -

The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.

★ **Definition of NFA-** Mathematically, NFA is 5 tuple structure written as: $M=(Q, \Sigma, \delta, q_0, F)$ where -

Q - finite set of states
Σ - input alphabet
δ - transition function where- $\delta: Q\Sigma \to 2^Q$
q_0 - initial (starting) state; $q_0 \in Q$
F - set of final state ; $F \subseteq Q$.

ex Draw a transition diagram for NDFA which accepts all the strings - 'ends with a'-

$L = \{a, aa, ba, aba \cdots\}$
$\Sigma = \{a, b\}$

Solution-

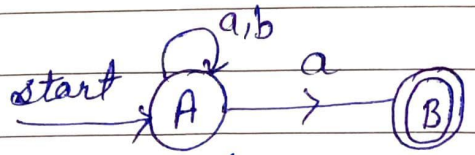


Fig:- Transition Diagram.

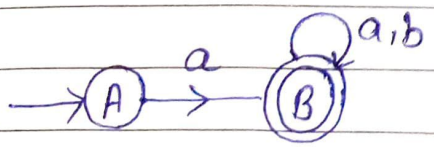
Transition table-

State	Input	
	a	b
→ A	{A,B}	A
(B)	ϕ	ϕ

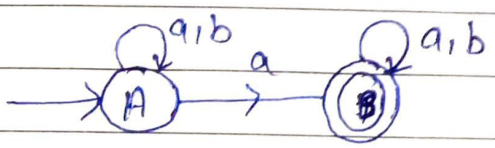
Name of Lecturer : .....

Examples of NDFA over $\Sigma = \{a, b\}$

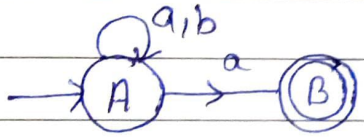
ex ① $L_1 = \{$ start with 'a' $\} = \{a, aa, aaa, ab, aab, abb \ldots\}$



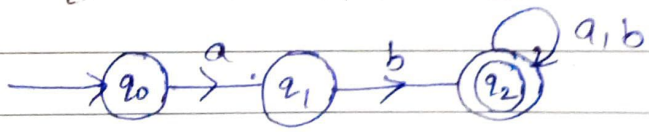
ex ② $L_2 = \{$ contain 'a' $\}$



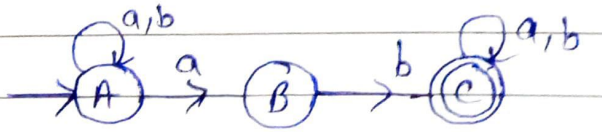
ex ③ $L_3 = \{$ ends with 'a' $\}$



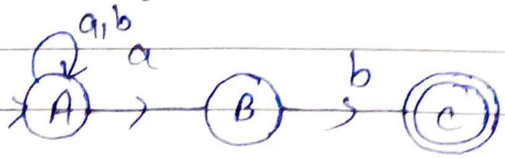
ex ④ $L_4 = \{$ start with 'ab' $\}$



ex ⑤ $L_5 = \{$ contains 'ab' $\}$



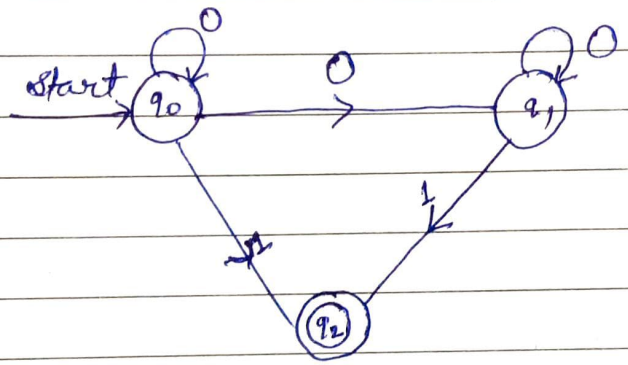
ex ⑥ $L_6 = \{$ ends with 'ab' $\}$



ex- Draw the transition diagram for NDFA and the transition table is given below-

State	Input	
	0	1
→ q_0	$\{q_0, q_1\}$	q_2
q_1	q_1	q_2
(q_2)	-	-

Solution-



Acceptability of String by NFA/NDFA :-

In DFA, a string is said to be accepted if the transition of entire string is stop in a final state because in DFA only one possible path for processing string that confirm whether the DFA accepts the string or not.

While in NFA there may be multiple simultaneous possibilities when it follows different alternative paths. In NFA among all the alternative paths if anyone path leads to a final state then the string is said to be accepted by NFA.

Name of Lecturer : .....

NFA with Λ-moves (null moves):-

When a machine move from one state to another but it moves without reading input symbol such move is called Λ-moves (ε).

★ Definition of NFA with Null moves (Λ-moves):-

NFA is a five-tuple structure

$$M = (Q, \Sigma, \delta, q_0, F)$$

where-

ⓐ M- is a machine
ⓑ Q- set of states
ⓒ Σ = set of input symbols
ⓓ δ - transition function ; $Q \times (\Sigma \cup \{\Lambda\}) \to 2^Q$
ⓔ q_0 - initial state ; $q_0 \in Q$
ⓕ F - set of final states ; $F \subseteq Q$.

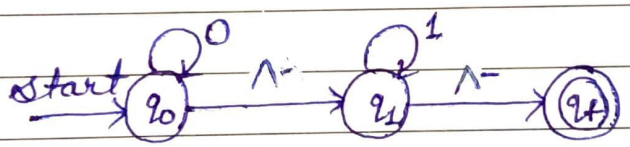
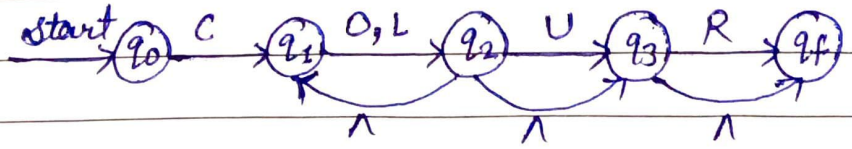


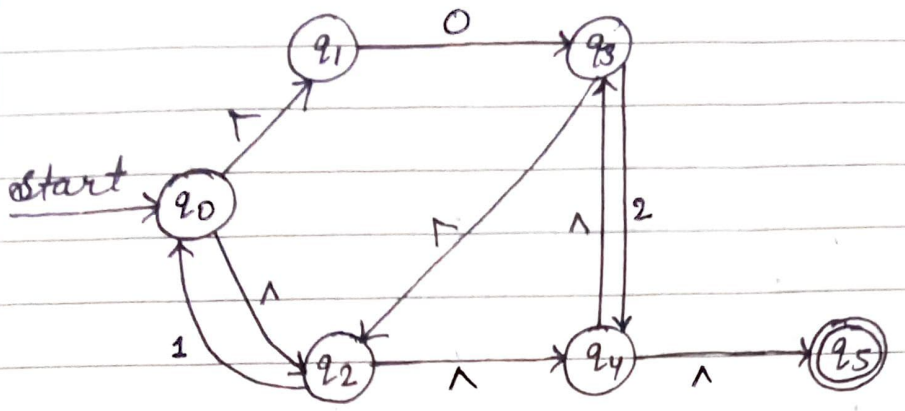
Fig: Transition diagram for NFA with Λ-moves

ex- Construct a NFA that accepts the string containing the letters 'COLOR' or 'COLOUR' over input alphabet $\Sigma = \{C, O, L, U, R\}$.

Solution-

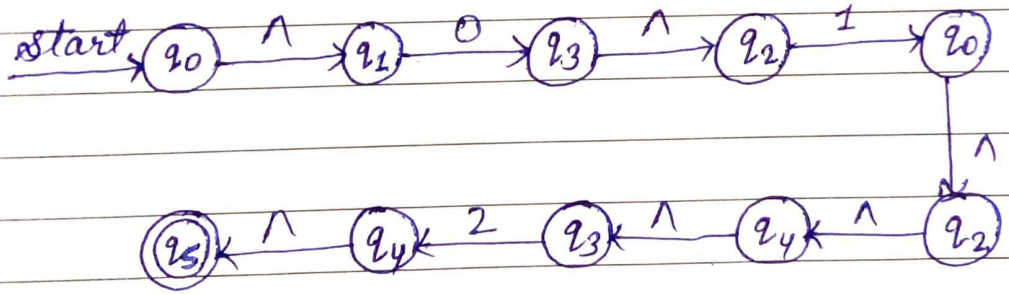


ex- Test whether the input string '012' is accepted or not by the given NFA



Solution:- Input string = 012
Current state (start state) = q_0
Final state = q_f

Transition sequence for string '012'-



After processing of the string '012' we have reached to the final state. Hence the string '012' is accepted by given NFA/NDFA.

★ Significance of NFA / NDFA with Λ- moves (Null moves):-

Construction of DFA is very difficult for certain input sets. Sometime to reach to final state we do not need any input symbol. In such a case we simply want to reach to final state. Such a transition to that specific state should be without any input symbol we requires Λ-moves by which a proper state can be obtained for reaching to final state. For these type of sets firstly we will construct NFA with Λ- moves. After that we construct NFA without Λ- moves and this NFA can be converted to DFA. Thus Λ- moves plays an important role in NFA.

★ Comparison between DFA and NFA:-

DFA	NFA/NDFA
① DFA stands for deterministic finite automata.	NDFA/NFA stands for Nondeterministic finite automata.
② In DFA for each pair of state & input symbol we have only one possible next state	In NFA for each pair of state & input symbol we have many possible next state.
③ DFA cannot use empty string transition.	NFA can use empty string transition.
④ It is more difficult to construct.	It is easier to construct in comparision of DFA.
⑤ Backtracking is allowed	Backtracking may or may not be allowed.
⑥ Requires more space	Requires less space.
⑦ DFA can be understood as one machine & a DFA machine can be constructed for every input & output	NFA can be understood as several little machines that compute together & there is no possibility to construct an NFA machine for every input & output.
⑧ DFA will reject the string if it ends other than accepting state	NFA dies or rejects the string.

Name of Lecturer : .....

★ Equivalence between two Finite Automata's :-

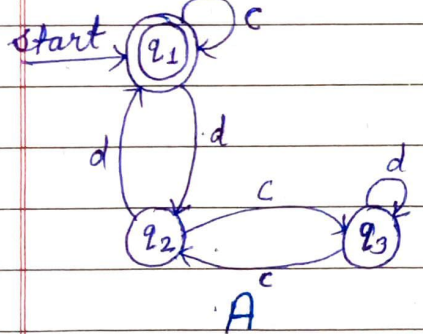
The term equivalence means equal in some respect.

★ Steps to Identify equivalence :-

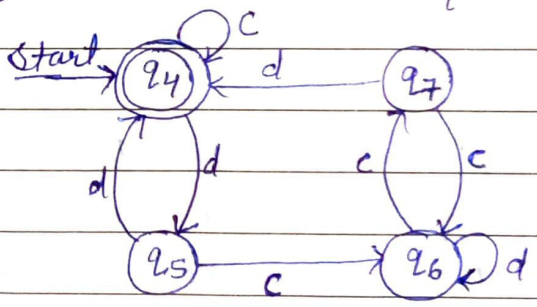
(1) For any pair of states $\{q_i, q_j\}$ the transition input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta\{q_i, a\} = q_a$ and $\delta\{q_j, a\} = q_b$

The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is Intermediate state and the other is Final state.

(2) If Initial state is Final state of one automaton, then in second automaton also Initial state must be Final state for them to be equivalent.



A



B

States	C	d
(q_1, q_4)	(q_1, q_4)	(q_2, q_5)
(q_2, q_5)	(q_3, q_6)	(q_1, q_4)
(q_3, q_6)	(q_2, q_7)	(q_3, q_6)
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

★ Converting NFA without $\wedge$-moves to its Equivalent DFA:-

For every NFA there exists an equivalent DFA. Thus, we can construct a DFA from a given NFA.

The transition of NFA maps the input $Q \times \Sigma \to 2^Q$. Thus for the output in NFA there are at most 2^Q possibilities.

Steps for converting NFA to DFA:-

Step 1 – Whenever we want to convert NFA to its equivalent DFA, we must have either state transition table (STT) or state transition diagram (STD) of that NFA.

→ If we have STT for that NFA then we have not to do anything at this stage & simply process it for the conversion.

→ If we have STD of any particular NFA then firstly convert it into the STT.

Step 2 – In this step we construct a subset table from the STT.

→ All input symbols of NFA makes the input symbol of the resultant DFA.

Name of Lecturer : .....

→ In NFA, there exists either one or more start states. We simply makes the set of all these start states & make it the start states of the resultant DFA.

→ Now we will find all the states of the resultant DFA. In NFA, there are finite no. of states. For this we make a set of all the states for the given NFA & find the subset of this set. All these subset are the resultant states of the DFA.

If NFA have n states then the resultant DFA have at most 2^n states.

Step 3- In this step we will find only accessable states for constructing the equivalent DFA.

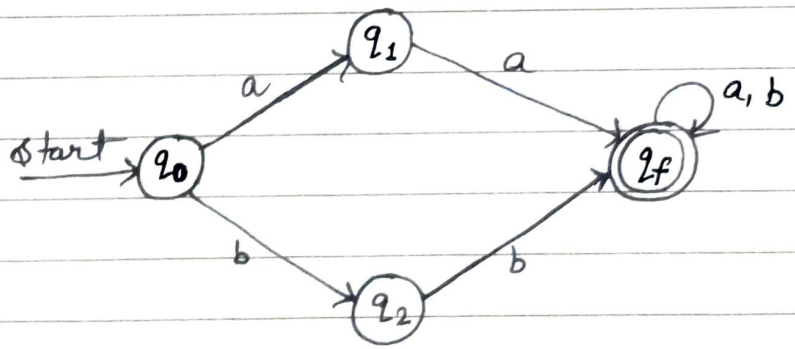
For finding the accessable states of the resultant DFA, we simply start with the start state & the input symbol of the resultant DFA. In transition fun. δ we take the start state of the resultant DFA & the input symbol of the resultant DFA. Then in the transition function δ we take start state of resultant DFA & any input symbol of the resultant DFA. Similarly, we take the start symbol with the other input symbol & so on.

Hence our subset table is constructed from step (2). This subset table is the STT of the resultant DFA.

Step.4- Now for finding the final states of the resultant DFA, we simply search it in the set of accessible states which contains at least one of the final state.

Step 5- In the last step we can construct the STD for the resultant DFA from the STT.

ex- Convert the given NDFA to its equivalent DFA.



Solution:-

Step ① Here STD of NFA is given. So firstly we will find STT for NFA.

state	INPUT	
	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	q_f	ϕ
q_2	ϕ	q_f
(q_f)	q_f	q_f

Step ② Input symbol of the resultant DFA = Input symbol of the given NDFA = (a, b)

Step ③ Start state of resultant DFA = start state of given NDFA = q_0

Step ④ Since there are 4 states in given NDFA so the number of at most state in the resultant DFA = $2^4 = 16$.
All the states of the resultant DFA = set of subset of the states of given NFA
= $\phi, q_0, q_1, q_2, q_f, [q_0, q_1], [q_0, q_2], [q_0, q_f], [q_1, q_2], [q_1, q_f], [q_2, q_f], [q_0, q_1, q_2], [q_0, q_1, q_f], [q_1, q_2, q_f], [q_0, q_2, q_f], [q_0, q_1, q_2, q_f]$

Step ⑤ Now we will find the accessible states from the states that was derived in the step (4). The start state of DFA is q_0.

state	INPUT	
	a	b
→ q_0	$[q_0, q_1]$	$[q_0, q_2]$

Now we have two states $[q_0, q_1]$ & $[q_0, q_2]$. we select both states because these are not present till now. Hence we will define the transition for it.

$$\delta([q_0, q_1], a) = \delta(q_0, a) \cup (q_1, a)$$
$$= [q_0, q_1] \cup [q_f]$$
$$= [q_0, q_1, q_f]$$

$$\delta([q_0, q_1], b) = \delta(q_0, b) \cup \delta(q_1, b)$$
$$= [q_0, q_2] \cup \phi$$
$$= [q_0, q_2]$$

$$\delta([q_0, q_2], a) = \delta(q_0, a) \cup \delta(q_2, a)$$
$$= [q_0, q_1] \cup \phi$$
$$= [q_0, q_1]$$

$$\delta([q_0, q_2], b) = \delta(q_0, b) \cup \delta(q_2, b)$$
$$= [q_0, q_2] \cup q_f$$
$$= [q_0, q_2, q_f]$$

states	Input	
	a	b
$\rightarrow q_0$	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_f]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0, q_1]$	$[q_0, q_2, q_f]$

Now we have two new states i.e. $\{q_0, q_1, q_f\}$ & $\{q_0, q_2, q_f\}$. So we process these states –

$$\delta([q_0, q_1, q_f], a) = \delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_f, a)$$
$$= [q_0, q_1] \cup q_f \cup q_f$$
$$= [q_0, q_1, q_f]$$

$$\delta([q_0, q_1, q_f], b) = \delta[q_0, b] \cup \delta(q_1, b) \cup \delta(q_f, b)$$
$$= [q_0, q_2] \cup \phi \cup q_f$$
$$= [q_0, q_2, q_f]$$

$$\delta([q_0, q_2, q_f], a) = \delta(q_0, a) \cup \delta(q_2, a) \cup \delta(q_f, a)$$
$$= [q_0, q_1] \cup \phi \cup q_f$$
$$= [q_0, q_1, q_f]$$

$$\delta([q_0, q_2, q_f], b) = \delta(q_0, b) \cup \delta(q_2, b) \cup \delta(q_f, b)$$
$$= [q_0, q_2] \cup q_f \cup q_f$$
$$= [q_0, q_2, q_f]$$

state	a	b
→ q_0	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_f]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0, q_1]$	$[q_0, q_2, q_f]$
$[q_0, q_1, q_f]$	$[q_0, q_1, q_f]$	$[q_0, q_2, q_f]$
$[q_0, q_2, q_f]$	$[q_0, q_1, q_f]$	$[q_0, q_2, q_f]$

Step ⑥ Now we have no new choices for the next state so we stop our processing. The final transition table for DFA is given below.

state	Input	
	a	b
→ q_0	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_f]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0, q_1]$	$[q_0, q_2, q_f]$
$[q_0, q_1, q_f]$	$[q_0, q_1, q_f]$	$[q_0, q_2, q_f]$
$[q_0, q_2, q_f]$	$[q_0, q_1, q_f]$	$[q_0, q_2, q_f]$

Step ④ Transition diagram for the equivalent DFA is:-

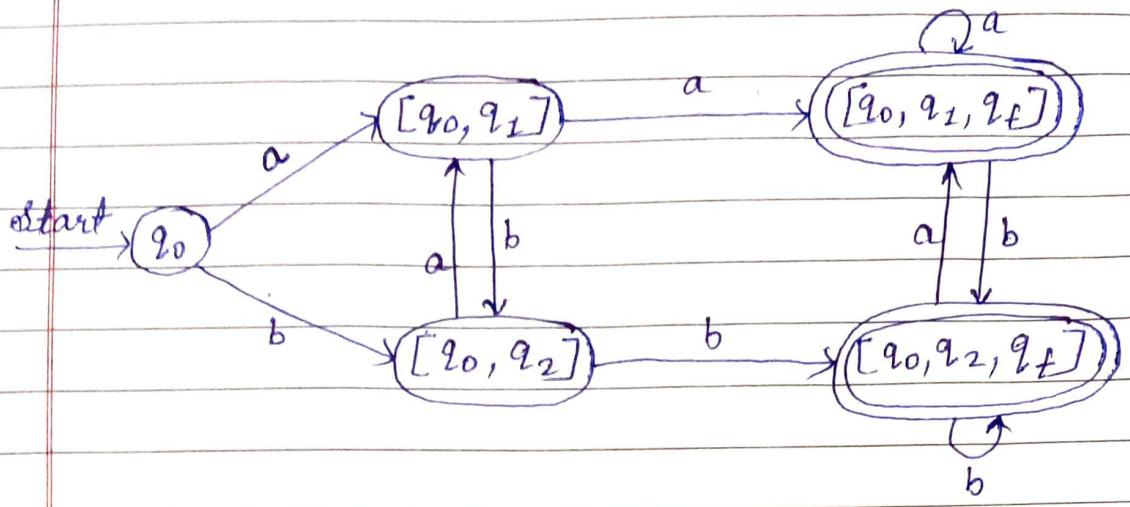


Fig- Transition diagram of DFA.

Conversion of NFA with $\wedge$-moves to its Equivalent NFA without $\wedge$-moves:-

Steps:-

① Firstly we will find all the pair of vertices which contain $\wedge$-moves. That is called $\wedge$-closure $\{q_i\}$ where $q_i \in Q$.

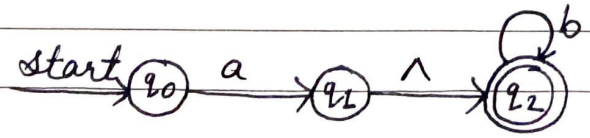
② Then δ' transition can be obtained. The δ' transition means a $\wedge$-closure on δ transition.

③ Step (2) is repeated for each input symbol and for each state of given NFA.

④ Using the resultant state the transition table for equivalent NFA without $\wedge$-moves can be built

⑤ In the last step we can construct the STD from the STT that was obtained in step 4.

ex- Convert the given NFA with $\wedge$-moves to its equivalent NFA without $\wedge$-moves.



Name of Lecturer : .....

Solution-

Step ① Firstly we make a STT for the STD.

State	Input		
	a	b	^
→ q_0	q_1	ϕ	ϕ
q_1	ϕ	ϕ	q_2
((q_2))	ϕ	q_2	ϕ

Step ② Now we will find the ^-closure

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

Step ③ In this step we will find the δ' transition

$$\delta'(q_0, a) = \wedge\text{-closure}\{\delta(\delta'(q_0, \wedge), a)\}$$
$$= \wedge\text{-closure}\{\delta(q_0, a)\}$$
$$= \wedge\text{-closure}\{q_1\}$$
$$= [q_1, q_2]$$

$$\delta'(q_0, b) = \wedge\text{-closure}\{\delta(\delta'(q_0, \wedge), b)\}$$
$$= \wedge\text{-closure}\{\delta(q_0, b)\}$$
$$= \wedge\text{-closure}\{\phi\}$$
$$= \phi$$

$\delta'(q_1, a) = \Lambda\text{-closure}\{\delta(\delta'(q_1, \Lambda), a)\}$
$= \Lambda\text{-closure}\{\delta((q_1, q_2), a)\}$
$= \Lambda\text{-closure}\{\delta(q_1, a) \cup \delta(q_2, a)\}$
$= \Lambda\text{-closure}\{\phi\}$
$= \phi$

$\delta'(q_1, b) = \Lambda\text{-closure}\{\delta\{\delta'(q_1, \Lambda), b)\}$
$= \Lambda\text{-closure}\{\delta((q_1, q_2), b)\}$
$= \Lambda\text{-closure}\{\delta(q_1, b) \cup \delta(q_2, b)\}$
$= \Lambda\text{-closure}\{\phi \cup q_2\}$
$= \Lambda\text{-closure}\{q_2\}$
$= q_2$

$\delta'(q_2, a) = \Lambda\text{-closure}\{\delta(\delta'(q_2, \Lambda), a)\}$
$= \Lambda\text{-closure}\{\delta(q_2, a)\}$
$= \phi$

$\delta'(q_2, b) = \Lambda\text{-closure}\{\delta(\delta'(q_2, \Lambda), b)\}$
$= \Lambda\text{-closure}\{\delta(q_2, b)\}$
$= q_2$

Step (4) The transition table for δ' function is -

state	Input	
	a	b
$\rightarrow q_0$	$\{q_1, q_2\}$	ϕ
q_1	ϕ	$\{q_2\}$
q_2	ϕ	$\{q_2\}$

Step⑤ The NFA can be drawn as -

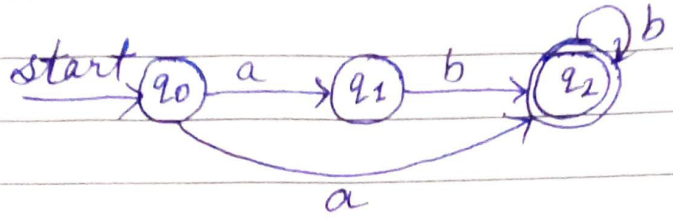
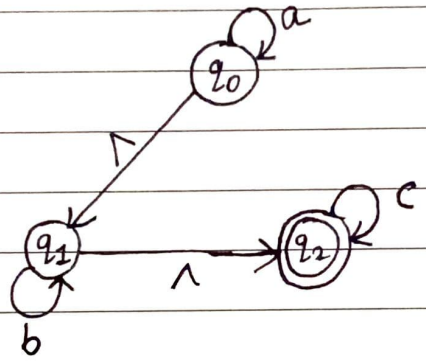


Fig: STD of NFA without Null moves.

Ex- Convert the given NFA with null moves to its equivalent NFA without null moves -



★ Solution:-

Step① First make the STT.

state	Input			
	a	b	^	c
→ q_0	q_1	ϕ	ϕ	ϕ
q_1	ϕ	ϕ	q_2	ϕ
(q_2)	ϕ	q_2	ϕ	q_2

Step ② Now we will find the Λ- closure

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

Λ- closure (q_1) = $\{q_1, q_2\}$

Λ- closure (q_2) = $\{q_2\}$

Step ③ In this step we will find δ' transition

$\delta'(q_0, a)$ = Λ- closure $\{\delta(\delta'(q_0, \Lambda), a)\}$

= Λ- closure $\{\delta(q_0, q_1, q_2), a\}$

= Λ- closure $\{\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)\}$

= Λ- closure $\{q_0 \cup \phi \cup \phi\}$

= Λ- closure $\{q_0\}$

= $[q_0, q_1, q_2]$

$\delta'(q_0, b)$ = Λ- closure $\{\delta(\delta'(q_0, \Lambda), b)\}$

= Λ- closure $\{\delta(q_0, q_1, q_2), b\}$

= Λ- closure $\{\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)\}$

= Λ- closure $\{\phi \cup q_1 \cup \phi\}$

= Λ- closure $\{q_1\}$

= $[q_1, q_2]$

$\delta'(q_0, c)$ = Λ- closure $\{\delta(\delta'(q_0, \Lambda), c)\}$

= Λ- closure $\{\delta(q_0, q_1, q_2), c\}$

= Λ- closure $\{\delta(q_0, c) \cup \delta(q_1, c) \cup \delta(q_2, c)\}$

= Λ- closure $\{\phi \cup \phi \cup q_2\}$

= Λ- closure $\{q_2\}$

= q_2

$\delta'(q_1, a) = \Lambda\text{-closure}\{\delta(\delta'(q_1, \Lambda), a)\}$
$= \Lambda\text{-closure}\{\delta(q_1, q_2), a\}$
$= \Lambda\text{-closure}\{\delta(q_1, a) \cup \delta(q_2, a)\}$
$= \Lambda\text{-closure}\{\phi \cup \phi\}$
$= \Lambda\text{-closure}\{\phi\}$
$= \phi$

$\delta'(q_1, b) = \Lambda\text{-closure}\{\delta(\delta'(q_1, \Lambda), b)\}$
$= \Lambda\text{-closure}\{\delta(q_1, q_2), b\}$
$= \Lambda\text{-closure}\{\delta(q_1, b) \cup \delta(q_2, b)\}$
$= \Lambda\text{-closure}\{q_1 \cup \phi\} = \Lambda\text{-closure}\{q_1\}$
$= [q_1, q_2]$

$\delta'(q_1, c) = \Lambda\text{-closure}\{\delta(\delta'(q_1, \Lambda), c)\}$
$= \Lambda\text{-closure}\{\delta(q_1, q_2), c\}$
$= \Lambda\text{-closure}\{\delta(q_1, c) \cup \delta(q_2, c)\}$
$= \Lambda\text{-closure}\{\phi \cup q_2\}$
$= \Lambda\text{-closure}\{q_2\}$
$= q_2$

$\delta'(q_2, a) = \Lambda\text{-closure}\{\delta(\delta'(q_2, \Lambda), a)\}$
$= \Lambda\text{-closure}\{\delta(q_2, a)\}$
$= \Lambda\text{-closure}\{\phi\}$
$= \phi$

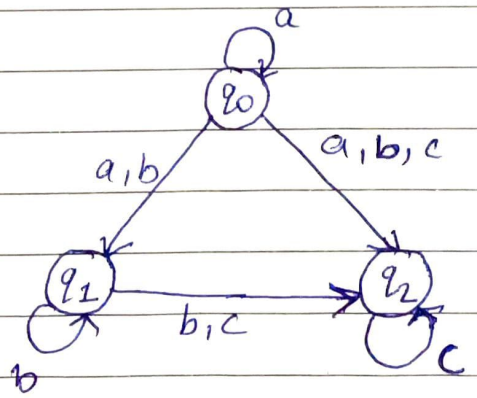
$\delta'(q_2, b) = \Lambda\text{-closure}\{\delta(\delta'(q_2, \Lambda), b)\}$
$= \Lambda\text{-closure}\{\delta(q_2, b)\}$
$= \Lambda\text{-closure}\{\phi\}$
$= \phi$

$\delta'(q_2, c) = \Lambda$- closure $\{\delta(\delta'(q_2, \Lambda), c)\}$
$= \Lambda$- closure $\{\delta(q_2, c)\}$
$= \Lambda$- closure $\{q_2\}$
$= q_2$

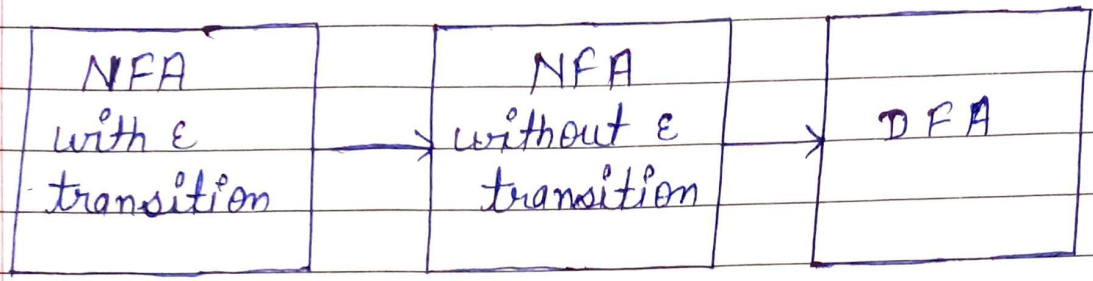
Step (4) Now we will design transition table for δ' function.

state	Input		
	a	b	c
→ q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	q_2
q_1	ϕ	$\{q_1, q_2\}$	q_2
((q_2))	ϕ	ϕ	q_2

Step (5) The NFA can be drawn as-



Conversion from NFA with ε (null transition) to DFA :-



Steps -

1. Firstly we will make all pair of vertices which contain ε moves. That is called ε closure.
2. δ' transition can be obtained. The δ' transition means ε closure on δ moves.
3. Step (2) is repeated for each input symbol & for each state of given NFA.
4. Using the resultant state the transition table NFA without ε closure can be constructed.
5. In this step we have STT of NFA. So we make the input symbol of NFA to the input symbol for the resultant DFA.
6. Now the start state of NFA = start state of DFA. Then find the transition from this start state

(7) To find the states of the DFA we will make set of all states of given NFA & from these states we can find the states of the DFA which are subset of given NFA.

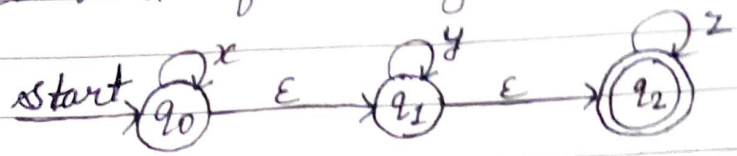
(8) Now from the state of the DFA we have to select only accessible states.

(9) For selecting the accessible states we have to apply the transition function δ on the start states of the DFA with the input symbol to find the new states.

(10) Repeat step (9) till no new states are found.

(11) For finding the final states of the resultant DFA we make all accessible states as the final state which contain atleast one final state of NFA.

Ex- Convert following NFA with ε moves to DFA.



Solution-

① First make transition table.

state	Input			
	x	y	z	ε
→ q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
(q_2)	ϕ	ϕ	q_2	ϕ

② Find ε closure

ε closure (q_0) = $\{q_0, q_1, q_2\}$

ε closure (q_1) = $\{q_1, q_2\}$

ε closure (q_2) = $\{q_2\}$

③ Find δ' transition.

$\delta'(q_0, x)$ = ε closure $\{\delta(\delta'(q_0, \varepsilon), x)\}$

= ε closure $\{\delta(q_0, q_1, q_2), x\}$

= ε closure $\{\delta(q_0, x) \cup \delta(q_1, x) \cup \delta(q_2, x)\}$

= ε closure $\{q_0 \cup \phi \cup \phi\}$

= ε closure $\{q_0\}$

= $[q_0, q_1, q_2]$

$\delta'(q_0, y) = \varepsilon$ closure $\{\delta(\delta'(q_0, \varepsilon), y)\}$
$= \varepsilon$ closure $\{\delta(q_0, q_1, q_2), y\}$
$= \varepsilon$ closure $\{\delta(q_0, y) \cup \delta(q_1, y) \cup \delta(q_2, y)\}$
$= \varepsilon$ closure $\{\phi \cup q_1 \cup \phi\}$
$= \varepsilon$ closure $\{q_1\}$
$= [q_1, q_2]$

$\delta'(q_0, z) = \varepsilon$ closure $\{\delta(\delta'(q_0, \varepsilon), z)\}$
$= \varepsilon$ closure $\{\delta(q_0, q_1, q_2), z\}$
$= \varepsilon$ closure $\{\delta(q_0, z) \cup \delta(q_1, z) \cup \delta(q_2, z)\}$
$= \varepsilon$ closure $\{\phi \cup \phi \cup q_2\}$
$= \varepsilon$ closure $\{q_2\}$
$= q_2$

$\delta'(q_1, x) = \varepsilon$ closure $\{\delta(\delta'(q_1, \varepsilon), x)\}$
$= \varepsilon$ closure $\{\delta(q_1, q_2), x\}$
$= \varepsilon$ closure $\{\delta(q_1, x) \cup \delta(q_2, x)\}$
$= \varepsilon$ closure $\{\phi \cup \phi\}$
$= \varepsilon$ closure $\{\phi\}$
$= \phi$

$\delta'(q_1, y) = \varepsilon$ closure $\{\delta(\delta'(q_1, \varepsilon), y)\}$
$= \varepsilon$ closure $\{\delta(q_1, q_2), y\}$
$= \varepsilon$ closure $\{\delta(q_1, y) \cup \delta(q_2, y)\}$
$= \varepsilon$ closure $\{q_1 \cup \phi\}$
$= \varepsilon$ closure $\{q_1\}$
$= [q_1, q_2]$

$$\delta'(q_1, z) = \varepsilon \text{ closure } \{\delta(\delta'(q_1, \varepsilon), z)\}$$
$$= \varepsilon \text{ closure } \{\delta(q_1, q_2), z\}$$
$$= \varepsilon \text{ closure } \{\delta(q_1, z) \cup \delta(q_2, z)\}$$
$$= \varepsilon \text{ closure } \{\phi \cup q_2\}$$
$$= \varepsilon \text{ closure } \{q_2\}$$
$$= q_2$$

$$\delta'(q_2, x) = \varepsilon \text{ closure } \{\delta(\delta'(q_2, \varepsilon), x)\}$$
$$= \varepsilon \text{ closure } \{\delta(q_2, x)\}$$
$$= \varepsilon \text{ closure } \{\phi\}$$
$$= \phi$$

$$\delta'(q_2, y) = \varepsilon \text{ closure } \{\delta(\delta'(q_2, \varepsilon), y)\}$$
$$= \varepsilon \text{ closure } \{\delta(q_2, y)\}$$
$$= \varepsilon \text{ closure } \{\phi\}$$
$$= \phi$$

$$\delta'(q_2, z) = \varepsilon \text{ closure } \{\delta(\delta'(q_2, \varepsilon), z)\}$$
$$= \varepsilon \text{ closure } \{\delta(q_2, z)\}$$
$$= \varepsilon \text{ closure } \{q_2\}$$
$$= q_2$$

④ Now design a transition table for δ' transition. This table is also called NDFA/NFA table.

state	Input		
	x	y	z
$\rightarrow q_0$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
(*) q_2	ϕ	ϕ	$\{q_2\}$

⑤ Find the input symbol of DFA.
Input symbol of DFA = input symbol of given NFA
= {x, y, z}

⑥ start state of DFA = start state of given NFA = q_0

⑦ All the states of the resultant DFA = set of subset of the set of states of given NFA = ϕ, $q_0, q_1, q_2, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}$

⑧ Find the accessible states from the state that is derived in step ⑦. The start state of DFA is q_0.

$\delta(q_0, x) = [q_0, q_1, q_2]$
$\delta(q_0, y) = [q_1, q_2]$
$\delta(q_0, z) = [q_2]$

state	Input		
	x	y	z
$\rightarrow q_0$	$[q_0, q_1, q_2]$	$[q_1, q_2]$	$[q_2]$

Now we have three new states $[q_0, q_1, q_2]$, $[q_1, q_2]$ & $[q_2]$. So we define the transition function for these.

$\delta([q_0, q_1, q_2], x) = \delta[q_0, x] \cup \delta(q_1, x) \cup \delta(q_2, x)$
$= [q_0, q_1, q_2] \cup \phi \cup \phi$
$= [q_0, q_1, q_2]$

$$\delta([q_0, q_1, q_2], y) = \delta[q_0, y] \cup \delta(q_1, y) \cup \delta(q_2, y)$$
$$= [q_1, q_2] \cup [q_1, q_2] \cup \phi$$
$$= [q_1, q_2]$$

$$\delta([q_0, q_1, q_2], z) = \delta(q_0, z) \cup \delta(q_1, z) \cup \delta(q_2, z)$$
$$= q_2 \cup q_2 \cup \phi$$
$$= q_2$$

$$\delta([q_1, q_2), x) = \delta(q_1, x) \cup \delta(q_1, x)$$
$$= \phi \cup \phi$$
$$= \phi$$

$$\delta([q_1, q_2], y) = \delta[q_1, y) \cup \delta(q_2, y)$$
$$= [q_1, q_2] \cup \phi$$
$$= [q_1, q_2]$$

$$\delta([q_1, q_2], z) = \delta[q_1, z] \cup \delta(q_2, z)$$
$$= [q_2] \cup [q_2]$$
$$= [q_2]$$

$$\delta(q_2, x) = \phi$$
$$\delta(q_2, y) = \phi$$
$$\delta(q_2, z) = q_2$$

⑨ Now we have no choice for the next state that is considered. so we stop our processing. Now we make the STT for DFA :-

State	Input		
	x	y	z
→ q_0	$[q_0, q_1, q_2]$	$[q_1, q_2]$	q_2
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_1, q_2]$	q_2
$[q_1, q_2]$	ϕ	$[q_1, q_2]$	q_2
q_2	ϕ	ϕ	q_2

(10) Transition diagram of equivalent DFA is:-

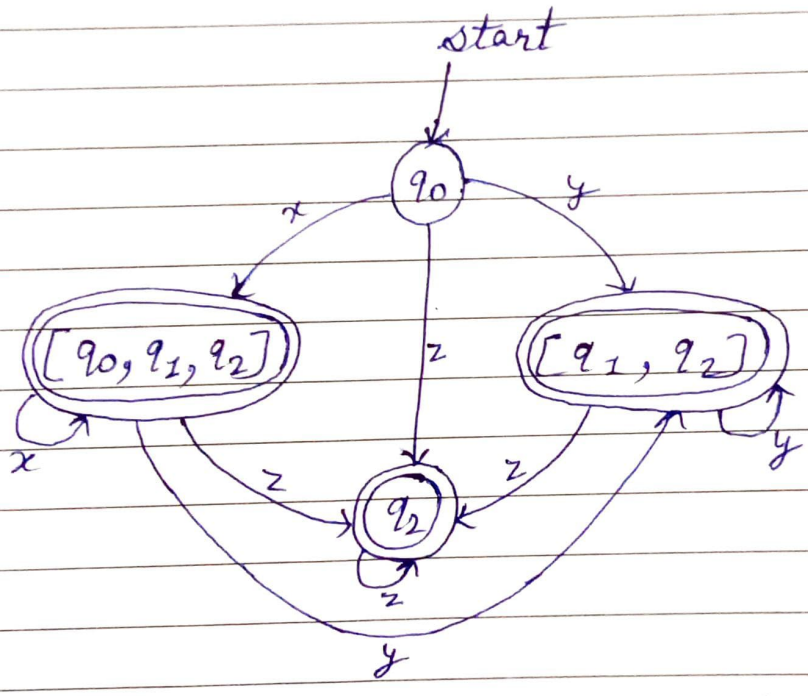


Fig:- Transition diagram for equivalent DFA.

Minimization of Finite Automata :-

The minimization of FA means reducing the no. of states from given FA. Thus we get the FA with redundant state after minimizing the automata.

Method for minimizing the FA :-

Step ① Remove all the states that are unreachable from the initial state via any set of the transition of DFA.

② Draw the transition table.

③ Split the transition table into two tables T_1 & T_2. T_1 contains all final states, & T_2 contains non-final states.

④ Find similar rows from T_1 such that :-

$$\delta(q, a) = p$$
$$\delta(r, a) = p.$$

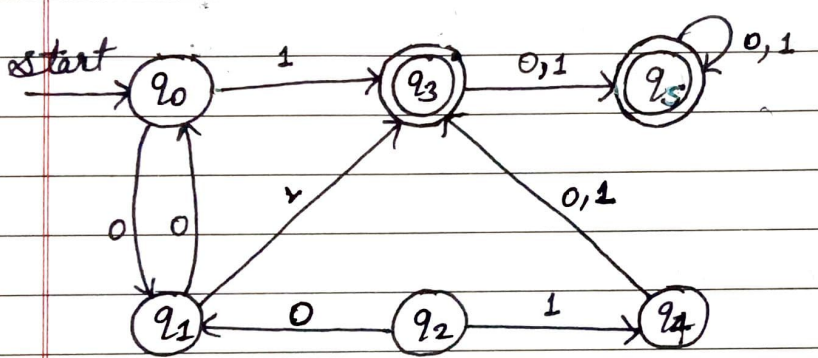
that means find the two states which have the same value of a & b and remove one of them.

⑤ Repeat step ③ until we find no similar rows available in the transition table T_1.

⑥ Repeat step ③ & ④ for table T_2 also.

Now combine the reduced T_2 & T_2 tables. The combined transition table is the transition table of minimized DFA.

ex- Construct a minimum finite state automata for the DFA that is shown in following figure -



Solution:-

step ① In the given DFA, q_2 & q_4 are the unreachable states so remove them.

step ② Draw the transition table for the rest of the states

state	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_0	q_3
(q_3)	q_5	q_5
(q_5)	q_5	q_5

step (3) Now divide rows of transition table into two sets as:

(a) One set contains those rows, which start from non-final states:

state	0	1
q_0	q_1	q_3
q_1	q_0	q_3

(b) Another set contains those rows, which starts from final states:

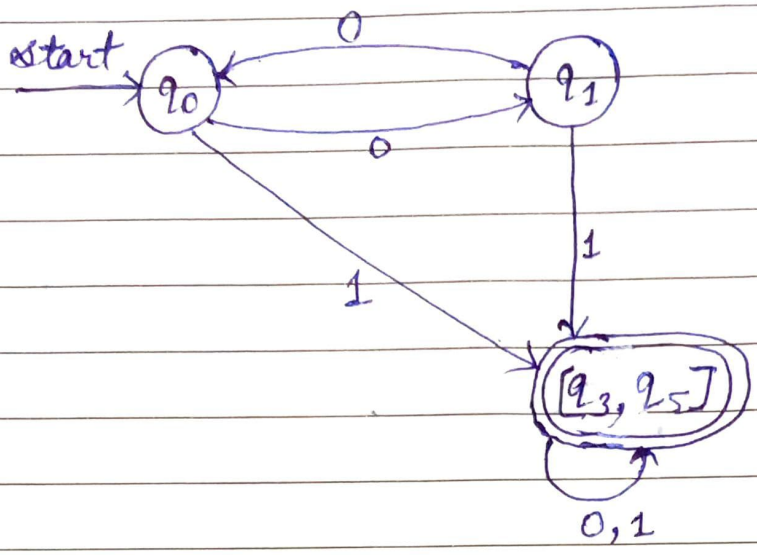
state	0	1
q_3	q_5	q_5
q_5	q_5	q_5

step (4) Set 1 has no similar rows so set 1 will be the same

step (5) In set 2 row 1 & row 2 are similar since q_3 and q_5 transit to the same state on 0 & 1. Hence q_3 and q_5 are equivalent.

step (6) Now combine set 1 and set 2.

state	0	1
$\rightarrow q_0$	q_1	q_3
q_1	q_0	q_3
$([q_3, q_5])$	$[q_3, q_5]$	$[q_3, q_5]$



Finite Automata with Outputs:-

The FA which we discussed in the earlier section is a collection of 5-tuples $M = (Q, \Sigma, \delta, q_0, F)$. It produce the output in the binary form. If it accepts the string it produce 'yes' otherwise it will produce 'NO'. This acceptability is decided on the basis of reachability of the final state but if we need to specify the output of the FA then we requires FA along with output.

Types of FA with Output:-

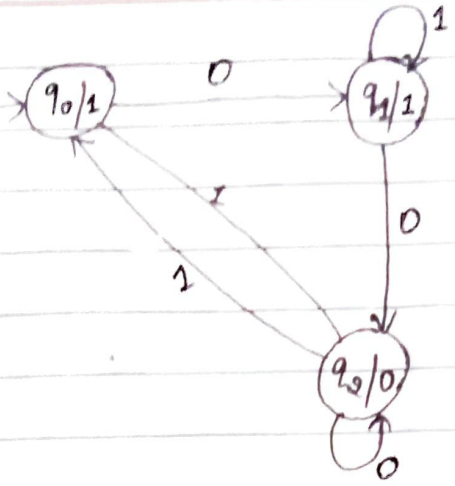
ⓐ Moore Machine
ⓑ Mealy Machine

ⓐ **Moore Machine:-** This machine was invented by 'E.F Moore'. So it is called Moore machine. Moore machine is a finite state machine in which the output depends on present state only. Moore machine can be described by 6 tuples $(Q, q_0, \Sigma, \Delta, \delta, \lambda)$ where -

Q :- finite set of states
q_0 :- initial state of machine
Σ :- finite set of input alphabet
Δ :- Output alphabet
δ :- transition function where $Q \times \Sigma \to Q$
λ :- Output function where $Q \to \Delta$

Name of Lecturer : .....

Ex (1) STD -



transition table for Moore machine -

Current state	Next state (δ)		Output (λ)
	0	1	
q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

In the above moore machine, the output is represented with each input state separated by /. The output length for a moore machine is greater than input by 1.

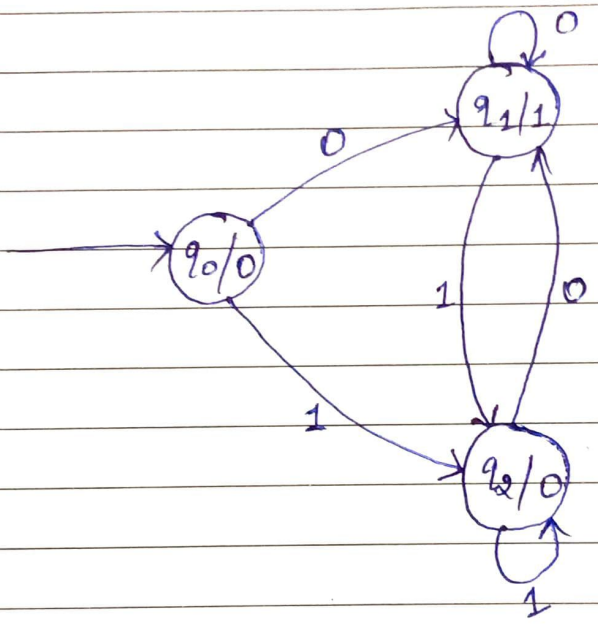
Input:- 010

Transition:- $\delta(q_0, 0) \Rightarrow \delta(q_1, 1) \Rightarrow \delta(q_1, 0) \Rightarrow q_2$

output:- 1110 (1 for q_0, 1 for q_1, again 1 for q_1, 0 for q_2).

Ex. (2) Design a Moore machine to generate 1's complement of a given binary number.

Solution- To generate 1's complement of a given binary no. the simple logic is that if the input is 0 then the output will be 1 and if the input is 1 then the output will be 0.



Current state	Next state		output (λ)
	0	1	
→ q_0	q_1	q_2	0
q_1	q_1	q_2	1
q_2	q_1	q_2	0

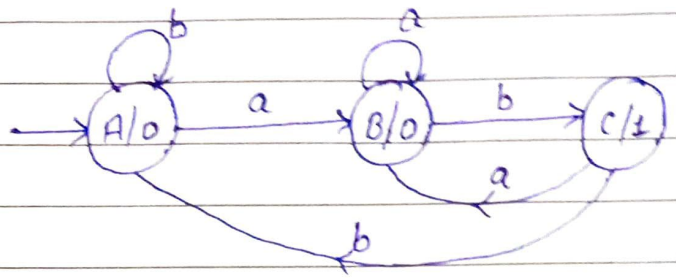
For instance, take one binary no 1011 then-

Input		1	0	1	1
state	q_0	q_2	q_1	q_2	q_2
output	0	0	1	0	0

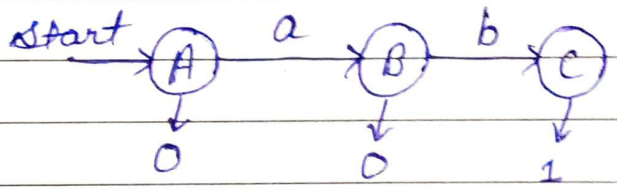
Ex ③ Construct a moore machine that takes set of all strings over {a, b} as i/p and prints '1' as o/p for every occurrance of 'ab' as a substring.

solution — $\Sigma = \{a, b\}$

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



(a) For input 'ab' –



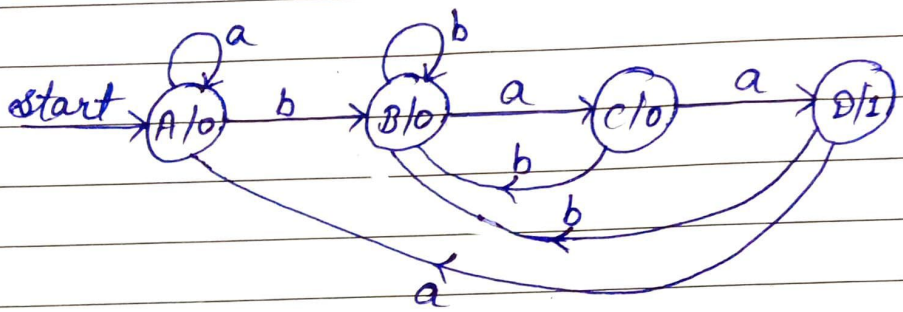
(b) For input 'abaab'

Input		a	b	a	a	b
state	A	B	C	B	B	C
output	0	0	1	0	0	1

Ex ④ Construct a moore m/c that takes set of all strings over {a, b} and counts no. of occurrences of substring 'baa'

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

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



ⓐ for input 'abaa'

Input		a	b	a	a
state	A	A	B	C	D
output	0	0	0	0	1

ⓑ for input 'baabbaa'

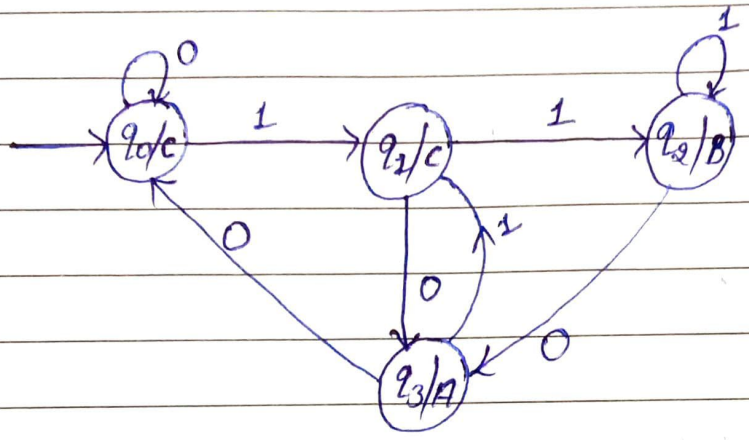
Input		b	a	a	b	b	a	a
state	A	B	C	D	B	B	C	D
output	0	0	0	1	0	0	0	1

Name of Lecturer : .....

Ex (5) Construct a more m/c that takes set of all strings over {0,1} & produces 'A' as o/p if i/p ends with '10' or produces 'B' as o/p if i/p ends with '11' otherwise produces 'c'.

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

$\Delta = \{A, B, C\}$



For input 111 -

Input		1	1	1
state	q_0	q_1	q_2	q_2
output	C	C	B	B

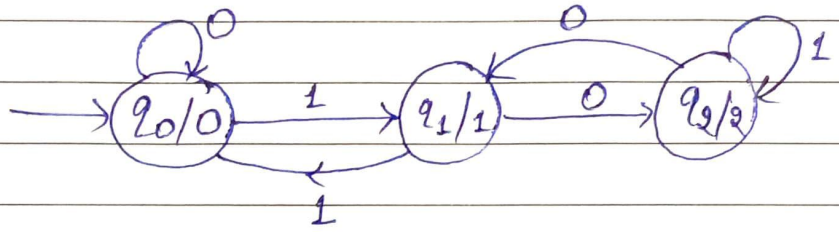
. . . . .

Input		1	0	1	1	0
state	q_0	q_1	q_3	q_1	q_2	q_3
output	C	C	A	C	B	A

Ex (6) Construct a moore m/c that takes binary no's as i/p & produces "residue modulo 3" as o/p.

Solution- $\Sigma = \{0, 1\}$ $\Delta = \{0, 1, 2\}$

	0	1	Δ
$\rightarrow q_0$	q_0	q_1	0
q_1	q_2	q_0	1
q_2	q_1	q_2	2

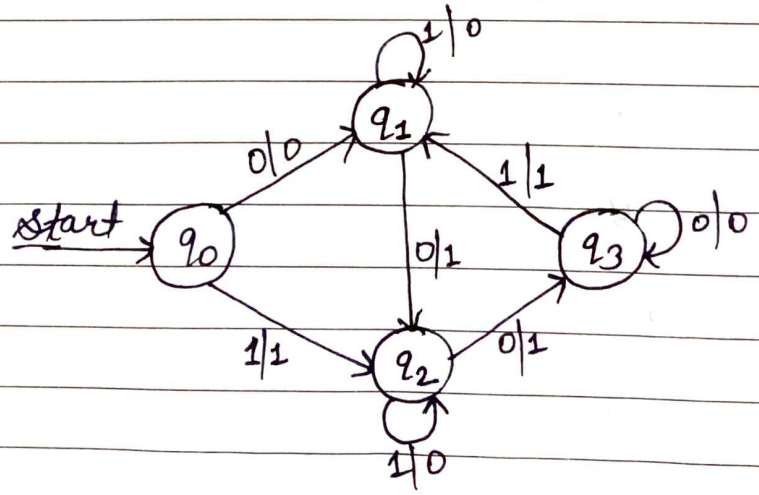


Name of Lecturer : .....

(b) Mealy Machine:- This machine was introduced by G.H. Mealy. Hence it is called mealy machine. Mealy machine a finite state machine in which the output depends on both present state and present input symbol.

Mealy machine consists of 6 tuples
$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where
Q: finite set of states
q_0: initial state
Δ: set of output alphabet
δ: transition function ; $Q \times \Sigma \to Q$
λ: output function ; $Q \times \Sigma \to \Delta$
Σ: input alphabet

Ex-① transition diagram for mealy machine-

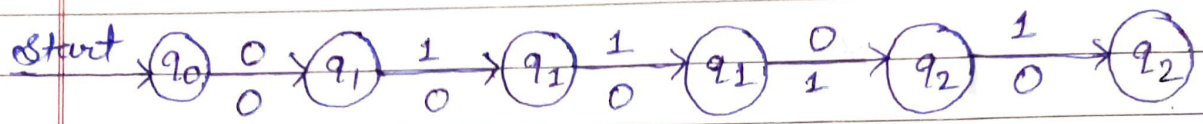


Name of Lecturer : .....

transition table -

state	Next state 0		Next state 1	
	state	output	state	output
→ q_0	q_1	0	q_2	1
q_1	q_2	1	q_1	0
q_2	q_3	1	q_2	0
q_3	q_3	0	q_1	1

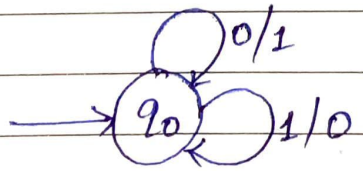
For input 01101 the transition sequence.



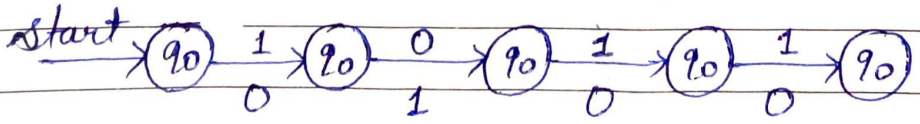
The output string is 00010 for the input string 01101.

Ex② Design a mealy m/c to generate 1's complement of a given no.

solution -

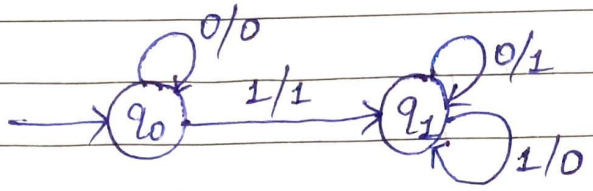


For input 1011 -



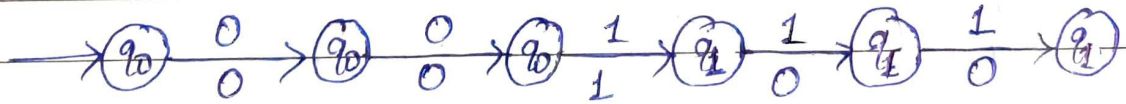
Ex ③ Construct a mealy m/c that takes binary no. as i/p & produces 2's complement of that no. as o/p. Assume the string is read LSB to MSB and end carray is discarded.

Solution –



For input 11100
1's comp - 00011
2's comp - 00100

transition –
starts from LSB to MSB



The output is - 00100

✳ Conversion from Moore machine to Mealy Machine -

In the Moore m/c the o/p is associated with every state, & in the mealy machine the output is given along the edge with input symbol.

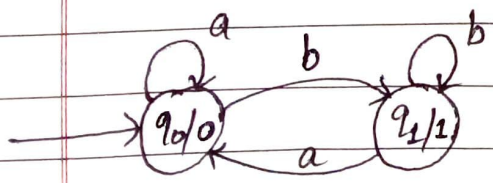
The equivalence of the Moore machine & mealy machine means both the machines generate the same output string for same input string.

Methods :-

Let $M = (Q, \Sigma, \delta, \lambda, q_0)$ be a Moore machine. The equivalent mealy machine can be represented by $M' = (Q, \Sigma, \delta, \lambda', q_0)$. The output function λ' can be obtained as:

$$\lambda'(q, a) = \lambda(\delta(q, a))$$

ex ① Convert the following Moore m/c to its equivalent mealy m/c.



Q	a	b	λ
q_0	q_0	q_1	0
q_1	q_0	q_1	1

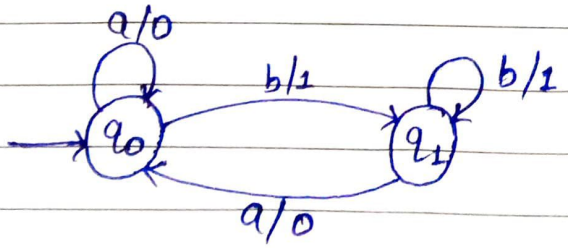
The equivalent Mealy machine can be obtained as follows -

$$\lambda'(q_0, a) = \lambda(\delta(q_0, a))$$
$$= \lambda(q_0)$$
$$= 0$$
$$\lambda'(q_0, b) = \lambda(\delta(q_0, b))$$
$$= \lambda(q_1)$$
$$= 1$$

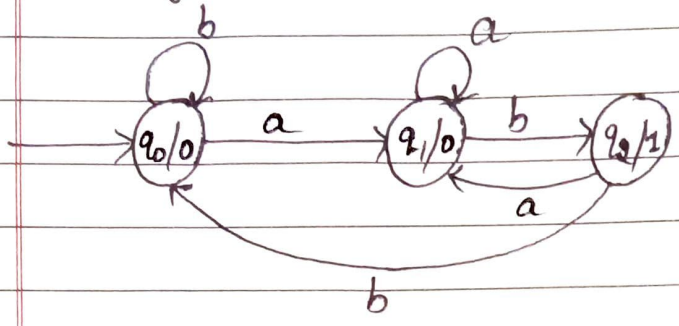
The λ for state q_1 is as follows -

$$\lambda'(q_1, a) = \lambda(\delta(q_1, a))$$
$$= \lambda(q_0)$$
$$= 0$$
$$\lambda'(q_1, b) = \lambda(\delta(q_1, b))$$
$$= \lambda(q_1)$$
$$= 1$$

Q \ Σ	Input 0		Input 1	
Q	state	o/p	state	o/p
q_0	q_0	0	q_1	1
q_1	q_0	0	q_1	1



Ex (2) Convert the given Moore m/c into its equivalent Mealy machine.



Solution-

Transition table of given moore m/c is as:-

δ	a	b	λ
q_0	q_1	q_0	0
q_1	q_1	q_2	0
q_2	q_1	q_0	1

The equivalent mealy m/c can be obtained as follows-

The λ for state q_0 -

$$\lambda'(q_0, a) = \lambda(\delta(q_0, a))$$
$$= \lambda(q_1)$$
$$= 0$$

$$\lambda'(q_0, b) = \lambda(\delta(q_0, b))$$
$$= \lambda(q_0)$$
$$= 0$$

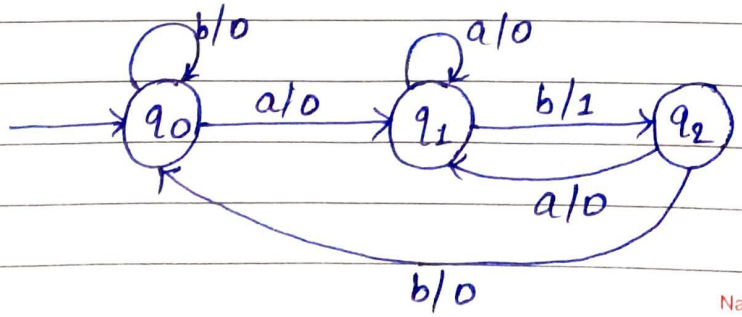
The λ for state q_1 is as-

$$\lambda'(q_1, a) = \lambda(\delta(q_1, a))$$
$$= \lambda(q_1)$$
$$= 0$$
$$\lambda'(q_1, b) = \lambda(\delta(q_1, b))$$
$$= \lambda(q_2)$$
$$= 1$$

The λ for state q_2 is as-

$$\lambda'(q_2, a) = \lambda(\delta(q_2, a))$$
$$= \lambda(q_1)$$
$$= 0$$
$$\lambda'(q_2, b) = \lambda(\delta(q_2, b))$$
$$= \lambda(q_0)$$
$$= 0$$

Q \ Σ	Input a		Input b	
	state	o/p	state	o/p
q_0	q_1	0	q_0	0
q_1	q_1	0	q_2	1
q_2	q_1	0	q_0	0



Conversion from Mealy machine to Moore m/c -

In moore m/c, the output is associated with every state, & in mealy m/c, the o/p is along the edge with input symbol. To convert moore machine to mealy machine, state output symbol are distributed to input symbol paths.

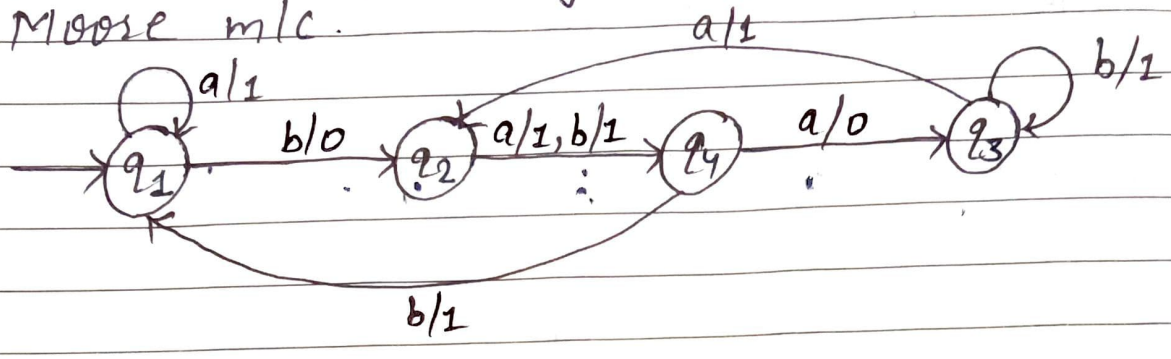
Steps:-

1. For each state (q_i), calculate the number of different output that are available in the transition table of the mealy m/c.

2. Copy state q_i, if all the outputs of q_i are the same. Break q_i into n states as q_{in}, if it has n distinct outputs where $n = 0, 1, 2, \ldots$

3. If the output of initial state is 0, insert a new initial state at the starting which gives 1 output.

Ex (1) Convert the following Mealy m/c into equivalent Moore m/c.



Solution -

state	Input a		Input b	
	state	o/p	state	o/p
q_1	q_1	1	q_2	0
q_2	q_4	1	q_4	1
q_3	q_2	1	q_3	1
q_4	q_3	0	q_1	1

Steps:-

(1) For state q_1, there is only one incident edge with output 0. So, we don't need to split this state in Moore m/c.

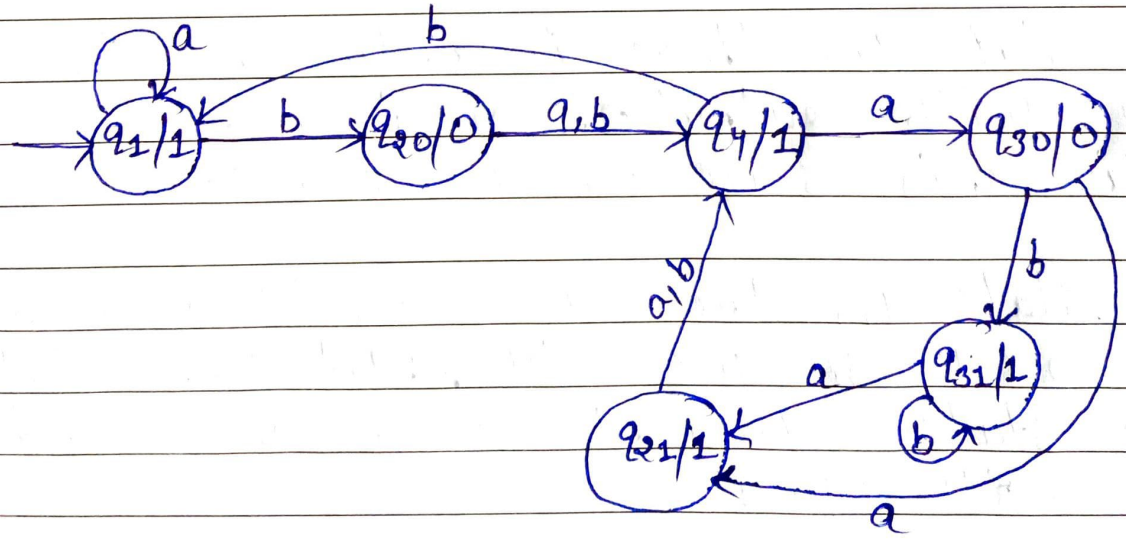
(2) For state q_2, there is 2 incident edge with o/p 0 & 1. so we will split this state into two states - q_{20} (state with o/p 0) and q_{21} (with o/p 1).

(3) For state q_3, there is 2 incident edge with output 0 & 1 so we will split this state into two states - q_{30} (with o/p 0) & q_{31} (with o/p 1)

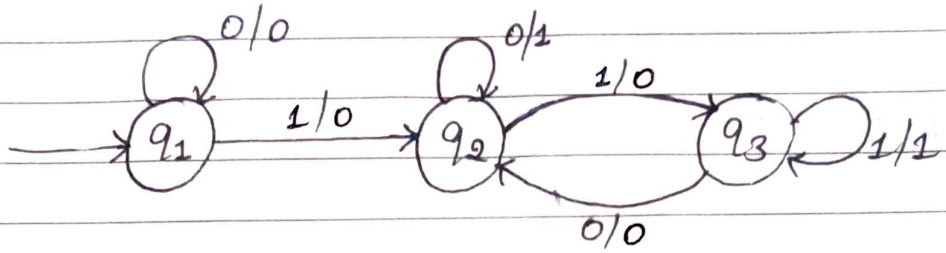
(4) For state q_4 there is only one incident edge with o/p 0. So we don't need to split this state in moore m/c.

State	a	b	output
q_1	q_1	q_2	1
q_{20}	q_4	q_4	0
q_{21}	q_4	q_4	1
q_{30}	q_{21}	q_{31}	0
q_{31}	q_{21}	q_{31}	1
q_4	q_{30}	q_1	1

The transition diagram:-



Ex(2) Convert the following Mealy m/c into equivalent moore m/c.



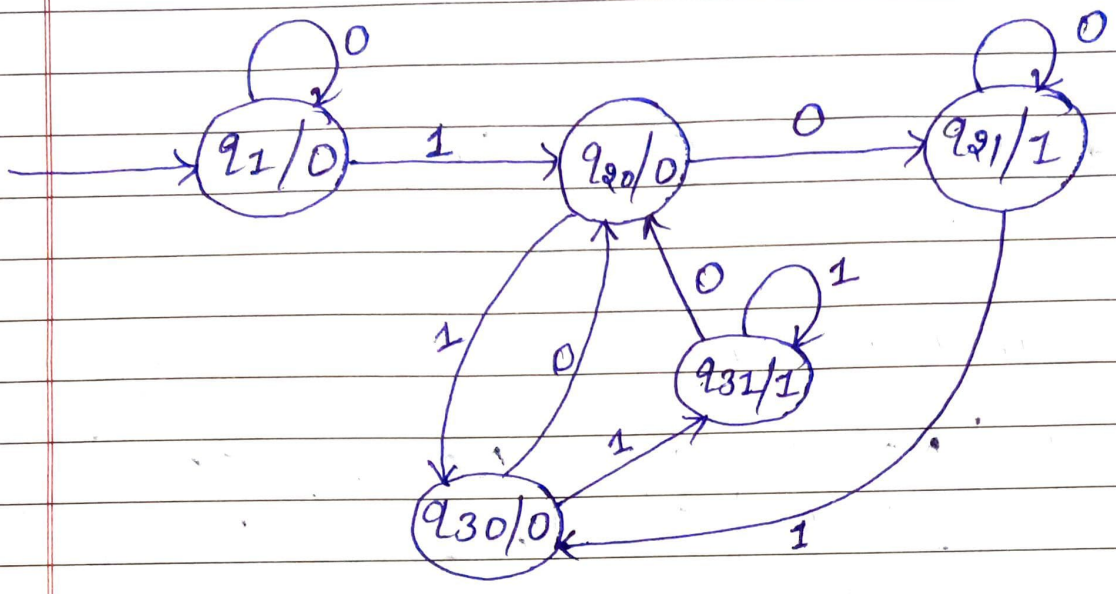
Solution- transition table for above mealy m/c.

Present state	Next state 0		Next state 1	
	state	O/P	state	O/P
q_1	q_1	0	q_2	0
q_2	q_2	1	q_3	0
q_3	q_2	0	q_3	1

state q_1 has only 1 o/p & state q_2 and q_3 have output 0&1. so we will split them For q_2, two states will be q_{20} and q_{21}. similarly for q_3 two states will be q_{30} & q_{31}.

state	0	1	O/P
q_1	q_1	q_{20}	0
q_{20}	q_{21}	q_{30}	0
q_{21}	q_{21}	q_{30}	1
q_{30}	q_{20}	q_{31}	0
q_{31}	q_{20}	q_{31}	1

The transition diagram for moore m/c will be :-



Regular Expression :-

The language accepted by finite automata can be easily described by simple expression called 'Regular Expression'.

→ The language accepted by some regular expression are referred to as Regular language.

→ A regular expression can also be described as a sequence of pattern that defines a string.

* Some rules for regular expression:-

1. ε, is a Regular expression denoting the set $\{\varepsilon\}$
2. ϕ, is Regular expression denoting the empty set $\{\}$
3. For each symbol $a \in \Sigma$, a is regular expression denoting set $\{a\}$
4. Union of two RE $(R_1 + R_2)$ is also Regular exp.
5. Concatenation of two RE $(R_1 \cdot R_2)$ is also Reg exp.
6. Kleene closure * of RE (R^*) is also Reg. exp.
7. If R is Reg-exp, (R) is also Reg-exp.
8. Any combination of the rules 1-7 is also a Regular expression.

★ Example of Reg. exp for finite language -

① No string $\{\} \rightarrow \phi$
② length 0 $\{\varepsilon\} \rightarrow \varepsilon, \lambda$
③ length 1 $\{a, b\} \rightarrow (a+b)$
④ length 2 $\{ab, ba, aa, bb\} \rightarrow aa+ab+ba+bb$
$= (a+b)(a+b)$
⑤ length 3 $\rightarrow (a+b)(a+b)(a+b)$
⑥ At most 1 0 or 1 $\{\varepsilon, a, b\}$ $(\varepsilon+a+b)$
⑦ At most 2 $(\varepsilon+a+b)(\varepsilon+a+b)$

★ Example of Reg exp for Infinite language -

Important note :-
① $a^* = \varepsilon, a, aa, aaa, aaaa, \ldots\ldots$
here * denotes to $0, 1, 2, 3, \ldots\ldots$
② $a+ = a, aa, aaa, aaaa, \ldots\ldots\ldots$
$+ = * - \varepsilon$
$aa^* = a+$

★ Regular Expression examples -
Consider Alphabet $\varepsilon = \{a, b\}$

① All strings having a single 'b'
$a^* b a^*$

② All strings having atleast one 'b'
$(a+b)^* b (a+b)^*$

③ All strings having 'bbbb' as substring
$(a+b)^* bbbb (a+b)^*$

④ All strings end with 'ab'
$(a+b)^* ab$

⑤ All strings starts with 'ba'
$ba(a+b)^*$

⑥ All strings beginning and end with 'a'
$a(a+b)^* a$

⑦ All strings containing 'a'
$(a+b)^* a (a+b)^*$

⑧ All strings starting & end with different symbol
$(a(a+b)^* b + b(a+b)^* a)$

⑨ All strings having two 'b'.
$a^* ba^* b a^*$

⑩ All string having even length
$((a+b)(a+b))^*$

⑪ All strings having odd length.
$((a+b)(a+b))^* (a+b)$

(12) Set of all strings 'divisible by 3'
⇒ So the language contain - 0, 3, 6, 9, 12 ---
length strings -

⇒ $((a+b)(a+b)(a+b))^*$

(13) Number of a's should be exactly 2.
$b^* a b^* a b^*$

(14) No. of a's atleast 2 ($a \geq 2$)
$b^* a b^* a (a+b)^*$

(15) No. of a's atmost 2. ('a' can be 0, 1, 2 only)
$b^* (\epsilon + a) b^* (\epsilon + a) b^*$

(16) No. of a's are even
$(b^* a b^* a b^*)^* + b^*$
or
$(b^* a b^* a)^* . b^*$

(17) starting and ending with same symbol
⇒ Here the language contains -
$L = \{\epsilon, a, b, aa, bb, \ldots\ldots\}$

⇒ $a(a+b)^* a + b(a+b)^* b + a + b + \epsilon$

Identity Rules For Regular Expression:-

If R, R_1, R_2 are regular expression then the identity rules of regular expression are:-

1. $\epsilon \cdot R = R \cdot \epsilon = R$
2. $\epsilon^* = \epsilon$ (ϵ is a null string)
3. $(\phi)^* = \epsilon$
4. $\phi R = R\phi = \phi$
5. $\phi + R = R + \phi = R$
6. $R + R = R$
7. $RR^* = R^*R = R^+$
8. $(R^*)^* = R^*$
9. $\epsilon + RR^* = R^* = \epsilon + R^*R$
10. $(R_1 + R_2)R = R_1R + R_2R$
11. $(R_1 + R_2)^* = R_1^* R_2^* = R_1^* + R_2^*$
12. $R(\epsilon + R) = (\epsilon + R)R = R$
13. $(R + \epsilon)^* = R^* = (\epsilon + R)^*$
14. $((R_1R_2)^* R_1) = (R_1 (R_2 R_1)^*)$
15. $R^* R + R = R^* R$
16. $R^* R^* = R^*$

Note-

+ ⇒ Union {a+b = a union b}
• ⇒ Concatenation {a·b = a concatenate with b}

Name of Lecturer : .....

Arden's Theorem :-

Arden's theorem was invented by the scientist whose name is Arden. This theorem is basically used for checking the equivalence of two regular expressions. It is also used for simplifying the regular expression i.e. we will replace a given regular expression by an equivalent simpler expression.

Statement :- If P and Q are two regular expressions over the input set Σ then regular expression R is given as,

$$R = Q + RP$$

which has a unique solution-

$$R = QP^*$$

Proof -

Let P & Q are two regexp over the input string Σ. If P does not contain ϵ then there exists R such that

$$R = Q + RP \qquad -(1)$$

By considering the R.H.S of equation ①

$= Q + RP$
$= Q + QP^*P$ [Replacing R by QP^*]
$= Q(\epsilon + P^*P)$
$= QP^*$ [using identity ⑨]

Thus $R = QP^*$

To prove that $R = QP^*$ is a unique solution we will replace LHS. of equation ① by $Q+RP$.

$R = Q + RP$
$= Q + [Q + RP)P$
$= Q + QP + RP^2$ — ②

Again R can be replaced by $Q+RP$ in equation ②

$= Q + QP + (Q+RP)P^2$
$= Q + QP + QP^2 + RP^2$

Thus if we continue this process to n times then we get-

$= Q + QP + QP^2 ..... + QP^n + RP^{n+1}$

[Replace $R = QP^*$]

$= Q + QP + QP^2 + ... QP^n + QP^*P^{n+1}$
$= Q[\epsilon + P + P^2 + ... P^n + P^*P^{n+1}]$

$\boxed{R = QP^*}$

Hence it is proved that-

$\boxed{R = Q + RP \text{ has a unique solution- } R = QP^*}$

ex ① Prove $(a^*b^*)^* = (a+b)^*$

Solution -

Given - $(a^*b^*)^* = (a+b)^*$

By considering LHS -

$(a^*b^*)^*$ = { Any combination of a's, any combination of b's, any combination of a & b and ε }

= {ε, a, aa, b, bb, ab, ba, aab, bba, ...}

RHS -

$(a+b)^*$ = { ε, any combination of a's, any combination of b's and any combination of a's & b's }.

= {ε, a, aa, b, bb, ab, ba, aba, bab, ...}

So

LHS = RHS.

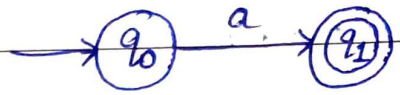
$(a^*b^*)^* = (a+b)^*$

Regular Expression and Automata-

✱ Conversion of Regular Expression to NFA/NDFA -

The rules that are used for converting a regular expression to NFA is given by Thompson which are -

Rule (1) For any $a \in \Sigma$, the reg.exp a denotes the language $\{a\}$.



The reg.exp ε denotes the language $\{\varepsilon\}$

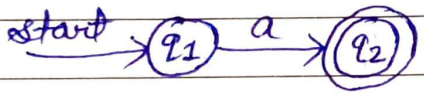
The reg. exp ϕ denotes the language $\{\}$. Whenever it is represented by NFA then the final state is unreachable.

It can also design as-

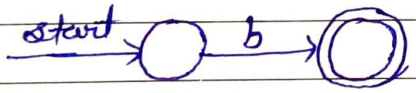
When we convert a regular expression into its equivalent NDFA then we can compose additional reg. exp by applying the operations on the reg. exp. These operations are given below -

① Grouping:- If $a \in \Sigma$ & it is represented by the language $\{a\}$ then the NDFA will be -

② Union - Let a & b are two regular exp. then the equivalent NDFA is -

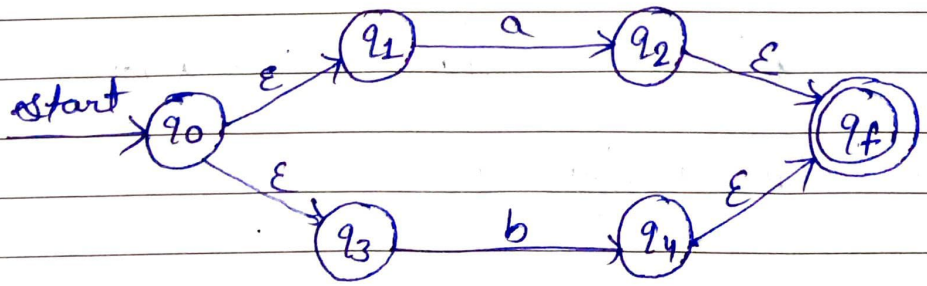


(a) NDFA for regexp a



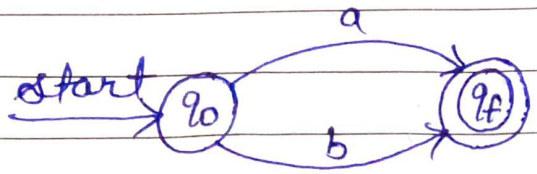
(b) NDFA for regexp b

The union of the two NDFA is -

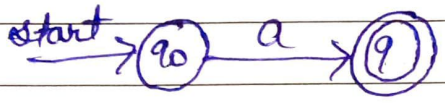


NDFA for regexp. a+b

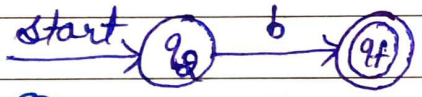
The previous NFA can be simplified as-



(3) Justaposition (concatenation):- Let a & b are two reg. exp.

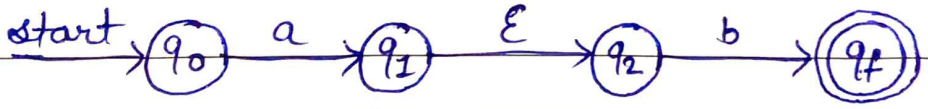


(a) NFA for reg. exp a



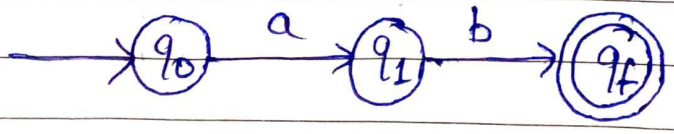
(b) NFA for reg. exp b

The concatenation of two NFA is -

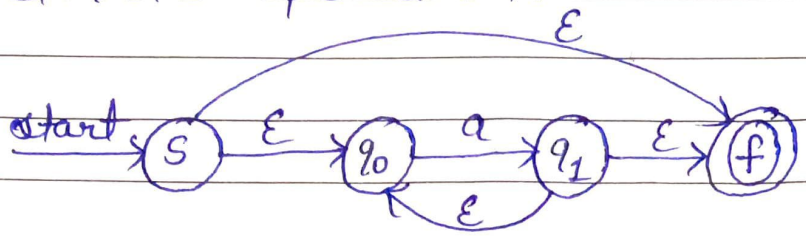


NDFA for reg. exp a.b

The above NFA can be simplified as-

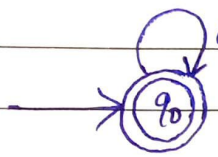


④ Kleene Star :- This operation is also called the closure operation.



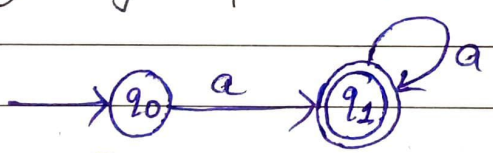
NDFA for reg. exp a^*

The above NDFA can be simplied as -

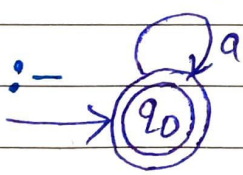


★ Examples of Reg. exp to NDFA conversion :-

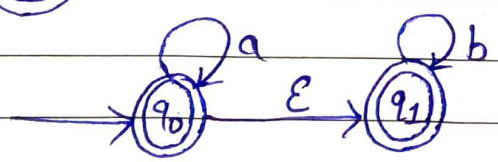
① a^+ :-



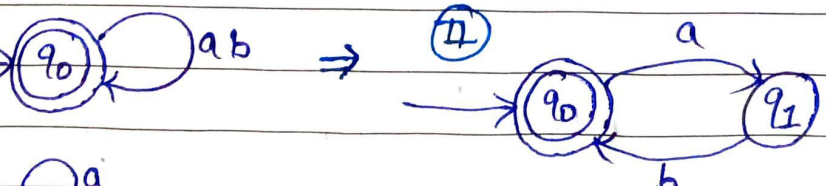
② $(a+b)^*$:-



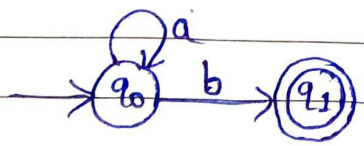
③ $a^* b^*$:-



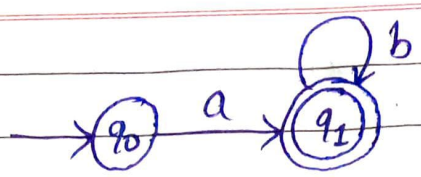
④ $(ab)^*$:- (I)



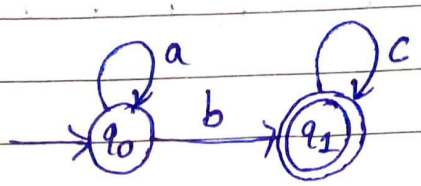
⑤ a^*b :-



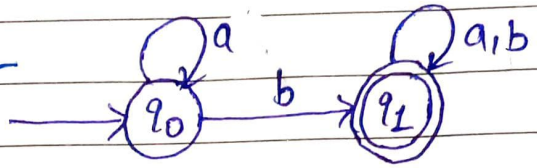
(6) ab^* :-



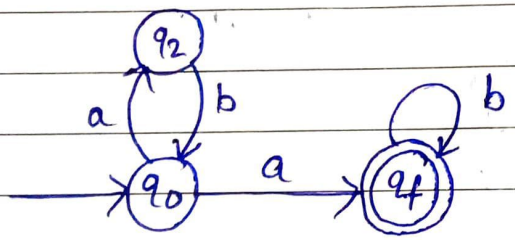
(7) a^*bc^* :-



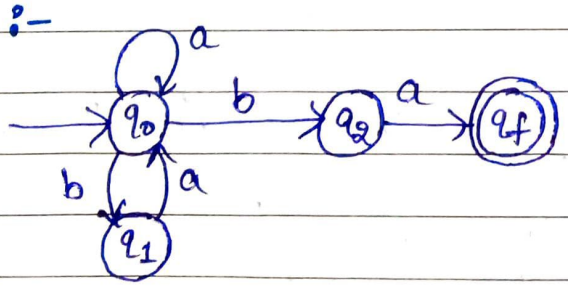
(8) $a^*b(a+b)^*$:-



(9) $(ab)^*a\,b^*$:-

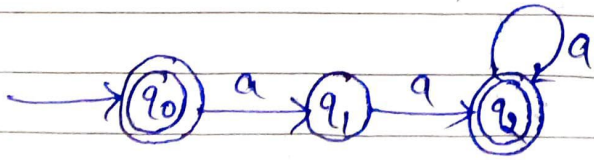


(10) $(a+ba)^*ba$:-



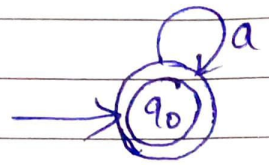
(11) $(aa+aaa)^*$:- $L = \{\epsilon, aa, aaa, aaaa, aaaaa, \ldots\}$

Only single a is not generated by this langu

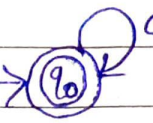


(12) $(a+aaaa)^*$:- The language generated by this regexp is -

$L = \{\varepsilon, a, aa, aaa, aaaa, aaaaa, \ldots\}$

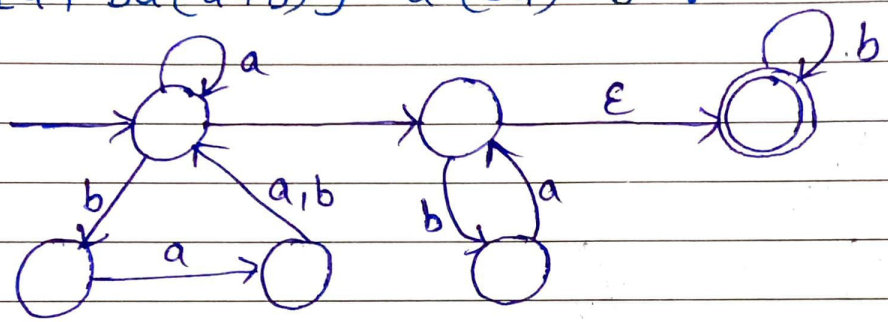


(13) $(ab)^* + (a+ab)^* b^* (a+b)^*$:-

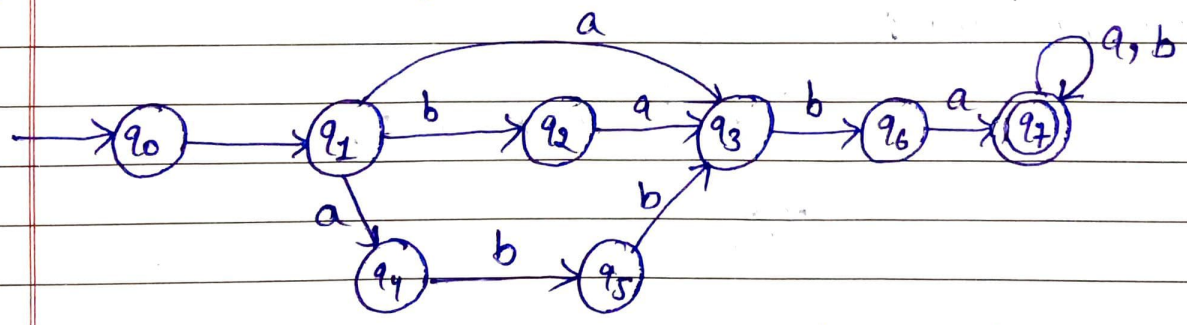


{for $(a+b)^*$}

(14) $[a + ba(a+b)]^* a (ba)^* b^*$:-

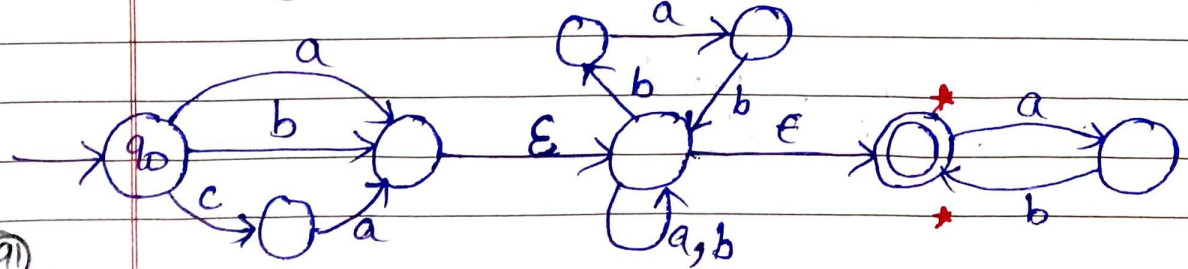


(15) $b(a+ba+abb)(ba(a+b)^*)$:-



(16) $(a+b+ca)((bab)^* + (a+b)^*)^* (ab)^*$:-

$= (a+b+ca)((bab)+(a+b))^* (ab)^*$



★ Conversion of Regular Expression to DFA :-

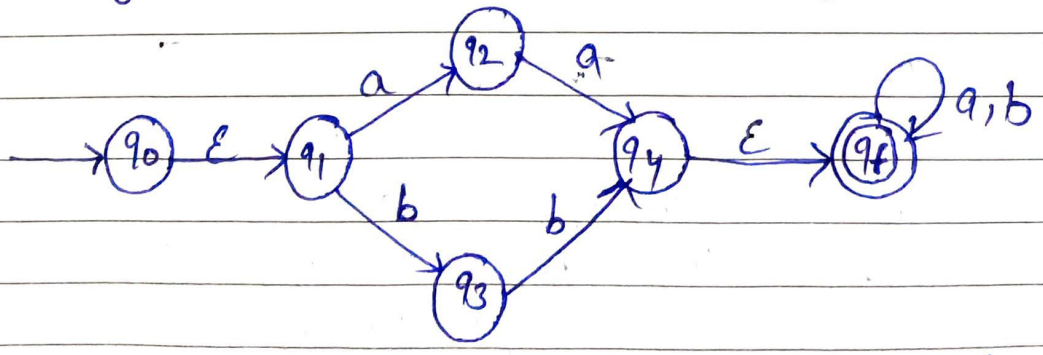
The conversion of Regexp to DFA involves following steps -

ⓐ Construct a transition diagram equivalent to the given regular expression with ε moves.
ⓑ Convert the NDFA with ε-moves to NDFA without ε-moves
ⓒ Construct STT from STD that is obtained in step 2.
ⓓ Convert the obtained NDFA to DFA.

Ex- ① Construct a finite automata (DFA) for the reg. exp-
$(a+b)^*(aa+bb)(a+b)^*$

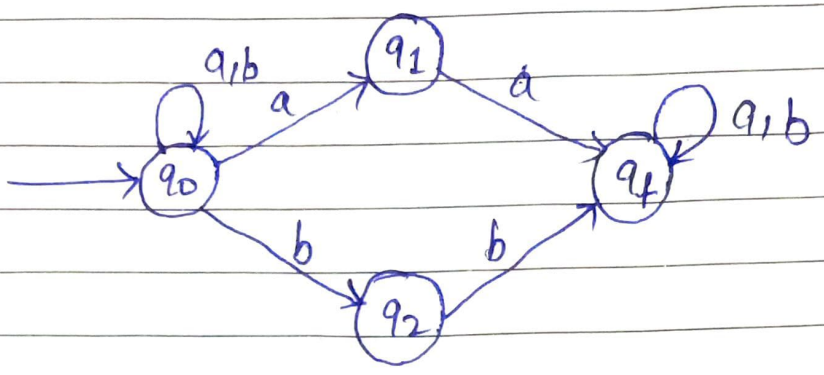
Solution- $R = (a+b)^*(aa+bb)(a+b)^*$

① Firstly we construct STD for the reg. exp-



NDFA for regexp $(a+b)^*(aa+bb)(a+b)^*$ with ε-moves-

② Remove all ε-moves -



NDFA without ε-moves for the reg. exp- $(a+b)^*(aa+bb)(a+b)^*$

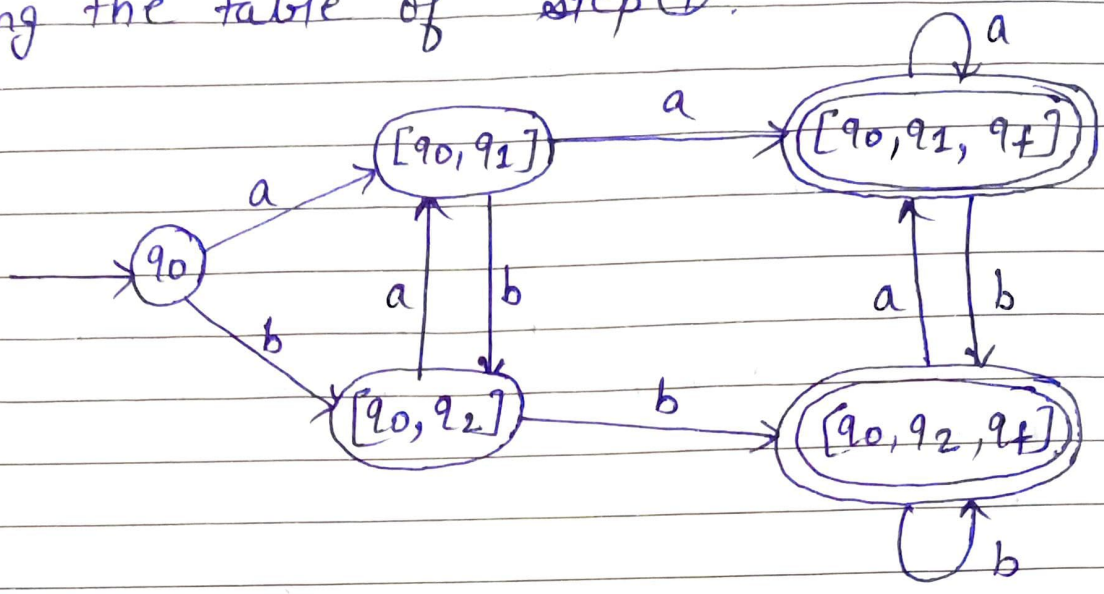
③ Construct the STT -

state	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	q_f	ϕ
q_2	ϕ	q_f
(q_f)	q_f	q_f

④ Now we construct the STT for DFA.

state	Input	
	a	b
→ q_0	$[q_0, q_1]$	$[q_0, q_2]$
$[q_0, q_1]$	$[q_0, q_2, q_f]$	$[q_0, q_2]$
$[q_0, q_2]$	$[q_0, q_1]$	$[q_0, q_2, q_f]$
($[q_0, q_1, q_f]$)	$[q_0, q_1, q_f]$	$[q_0, q_2, q_f]$
($[q_0, q_2, q_f]$)	$[q_0, q_1, q_f]$	$[q_0, q_2, q_f]$

(5) Now we construct the STD for the DFA by using the table of step (4).



(6) Now we need to reduce the no. of states from DFA.

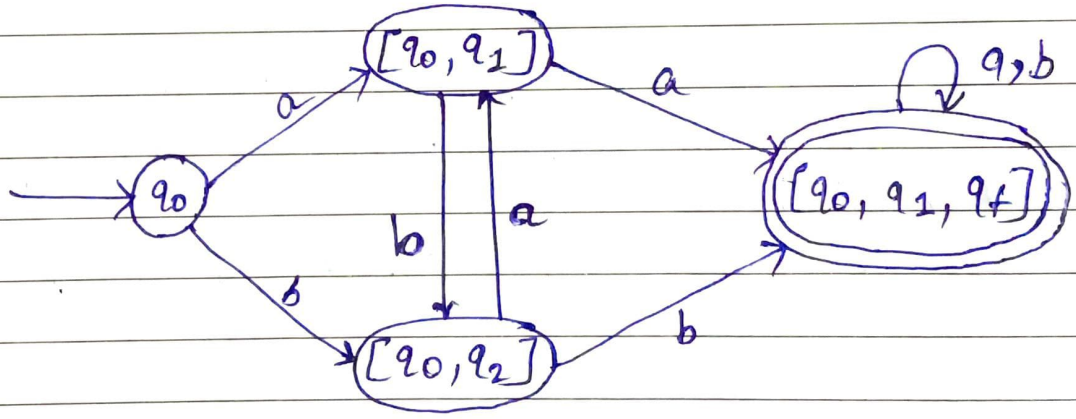


Fig- DFA for reg. exp $(a+b)^*(aa+bb)(a+b)^*$

☆ Converting DFA to Regular Expression :-

There are 2 methods to convert a finite automata into regular expression.

① Arden's Method
② State Elimination Method.

① Arden's Theorem - This theorem is popularly used to convert a given DFA to its regular expression. It states that -
Let P & Q be two reg.exp over Σ.
If P does not contain a null string ε, then -

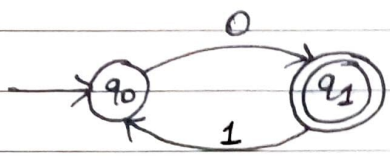
$R = Q + RP$ has a unique solution
i.e. - $R = QP^*$

Conditions - To use Arden's theorem -
① The transition diagram must not have any ε transition.
② There must be only a single initial state.

Steps -
① Find out the equations for each & every state of the given finite automata.
② Add 'ε' in the equation of initial state.
③ Bring final state in the form $R = Q + RP$ to get the required regular expression.

★ If there exists multiple final state, then:-
① write a regular expression for each final state separately.
② Add all the reg. exp to get the final reg. exp.

Ex① Find reg. exp. for the following finite automata using Arden's theorem.



Solution:-

step① Form a equation for each state

$$q_0 = \varepsilon + q_1 \cdot 1 \quad -①$$
$$q_1 = q_0 \cdot 0 \quad -②$$

step② Bring final state in the form $R = Q + RP$.
using ① & ② we get

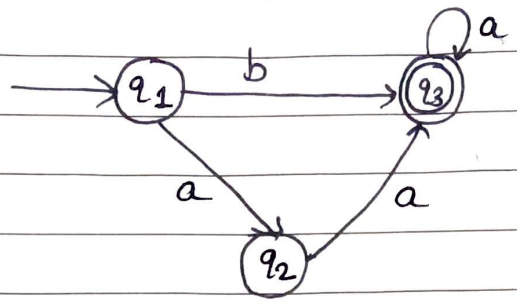
$$q_1 = (\varepsilon + q_1 \cdot 1) \cdot 0$$
$$= \varepsilon \cdot 0 + q_1 1 \cdot 0$$
$$= 0 + q_1(1 \cdot 0) \quad -③$$

using Arden's theorem in exp③, we get -

$$q_1 = 0 \cdot (1 \cdot 0)^*$$

Thus reg. exp for given finite automata is = $0(10)^*$

Ex ② Find reg. exp for following FA. using Arden's theorem.



Solution -

setp① Form an equation for each state -

$q_1 = \varepsilon$ — ①

$q_2 = q_1 \cdot a$ — ②

$q_3 = q_1 \cdot b + q_2 \cdot a + q_3 \cdot a$ — ③

setp② Bring final state in the form $R = Q + RP$

using ① in ② we get -

$q_2 = \varepsilon \cdot a$

$q_2 = a$ — ④

using ① & ④ in ③, we get -

$q_3 = q_1 \cdot b + q_2 \cdot a + q_3 \cdot a$

$q_3 = \varepsilon b + aa + q_3 \cdot a$

$= (b + a \cdot a) + q_3 \cdot a$ — ⑤

Using Arden's theorem in ⑤, we get -

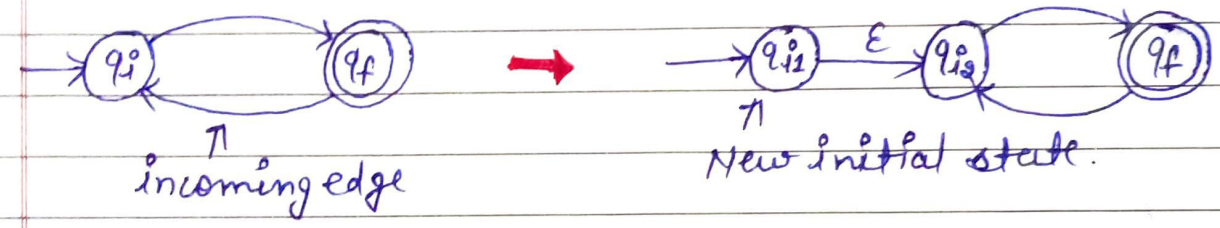
$q_3 = (b + a \cdot a)\, a^*$

Thus the reg. exp for given FA is = $\boxed{(b + aa)a^*}$

(2) State Elimination Method :- This method involves following steps -

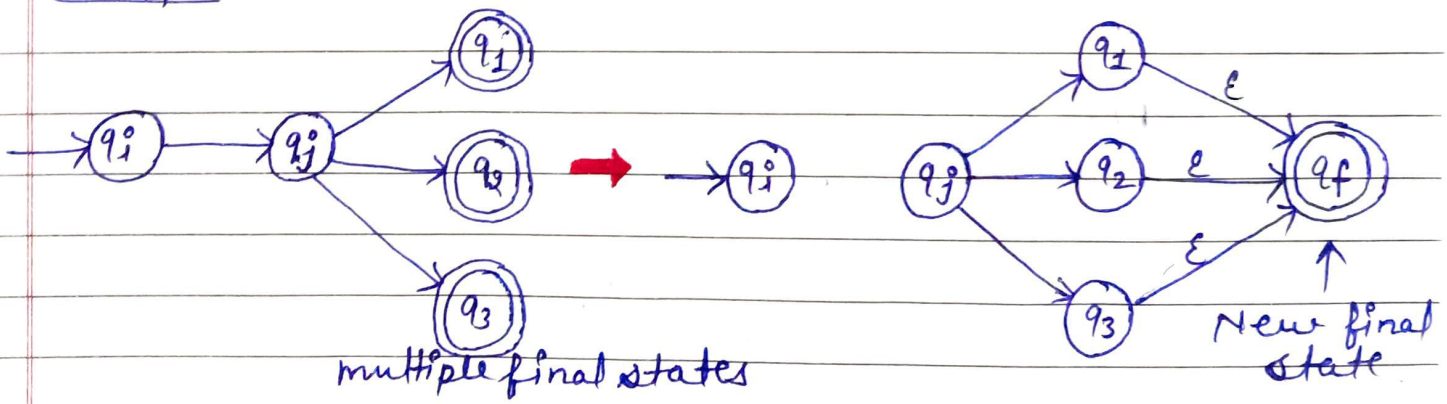
Step (1) If there exists any incoming edge to the initial state, then create a new initial state having no incoming edge to it.

Example -



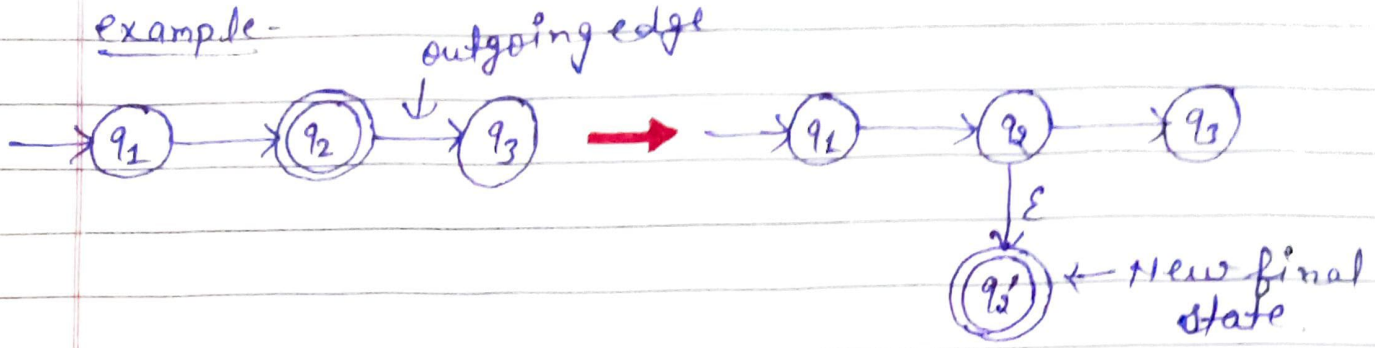
Step (2) If there exists multiple final state in the FA, then convert all the final states into non-final states & create a new single final state.

example -



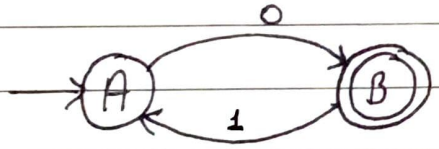
Step (3) If there exists any outgoing edge from the final state, then create a new final state having no outgoing edge from it.

example-



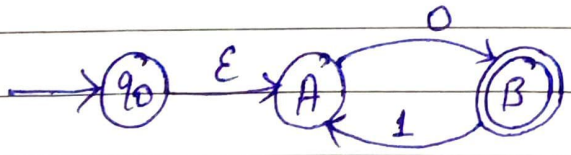
step④ Eliminate all the intermediate states one by one. Only an initial state going to final state will be left.

Ex① Find reg. exp. for the following F.A-

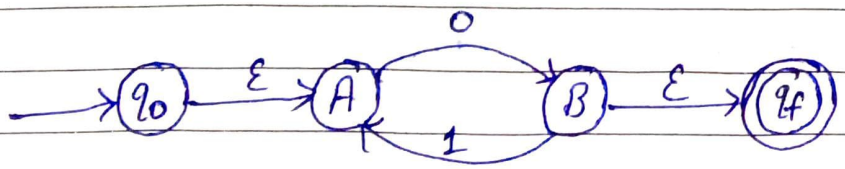


Solution-

step① Initial state A has incoming edge. so, we create a new initial state



step② Final state B has an outgoing edge. so we create a new final state



step③ Now we start eliminating the intermediate states.

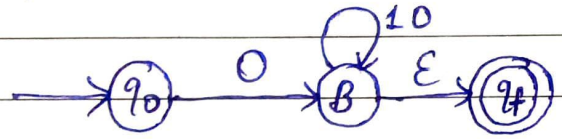
ⓐ First eliminate state A.

⇒ There is a path going from state q_0 to state B via state A.

⇒ So, after eliminating state A, we put a direct path from state q_0 to state B having cost $\varepsilon.0 = 0$.

⇒ There is a loop on state B using state A.

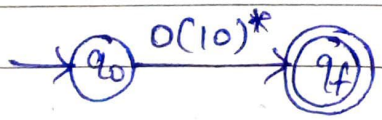
⇒ So after eliminating state A, we put a direct loop on state B having cost $1.0 = 10$.



ⓑ Now eliminate state B.

⇒ There is a path going from state q_0 to state q_f via state B.

⇒ So after eliminating state B, we put a direct path from q_0 to q_f having cost $0.(10)^*.\varepsilon = 0(10)^*$



So,

Regular Expression = $0(10)^*$

Equivalence of Two Regular Expression-

Two regular expressions R_1 and R_2 are equivalent if they represent the same set of language & also their corresponding FA are equivalent.

To prove regular expression R_1 & R_2 equivalent we can choose any one method of following-

Method(1) To prove reg. exp. R_1 & R_2 are equivalent, first we will find the string respective to reg exp. R_1 & R_2. If the string present in both the sets then R_1 & R_2 are equivalent.

Method(2) We can use the identities.

Method(3) We can construct the corresponding FA- M_1 & M_2 for Reg. exp R_1 & R_2. If both M_1 & M_2 are equal, then R_1 & R_2 are equivalent.

Ex- Show that $(a+b)^* = (a^*b^*)^*$

Solution- Consider L.H.S = $(a+b)^* = \{\varepsilon, a, aa, b, bb, ab, ba, \ldots\}$

$(a+b)^* = \{\varepsilon$, any combination of a's, any combination of b's, any combination of a & b$\}$

R.H.S = $(a^*b^*)^* = \{\varepsilon, a, aa, ab, ba, b, bb, \ldots\}$

$= \{\varepsilon$, any combination of a's, any combination of b's and any combination of a & b$\}$

Hence - LHS = RHS

Regular set –

A regular expression generates a set of strings known as a regular set. The regular set is also accepted by finite automata.

For ex–

if $L = \{\varepsilon, 00, 0000, 000000, 00000000, \ldots\ldots\}$

then it is called the set of even no. of 0's.

we can represent this set by FA as –

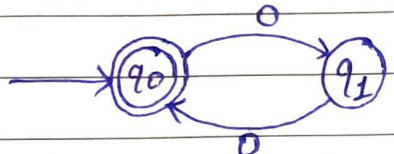


Fig- Finite automata for regular set of even number of 0's.

Closure Properties of Regular set –

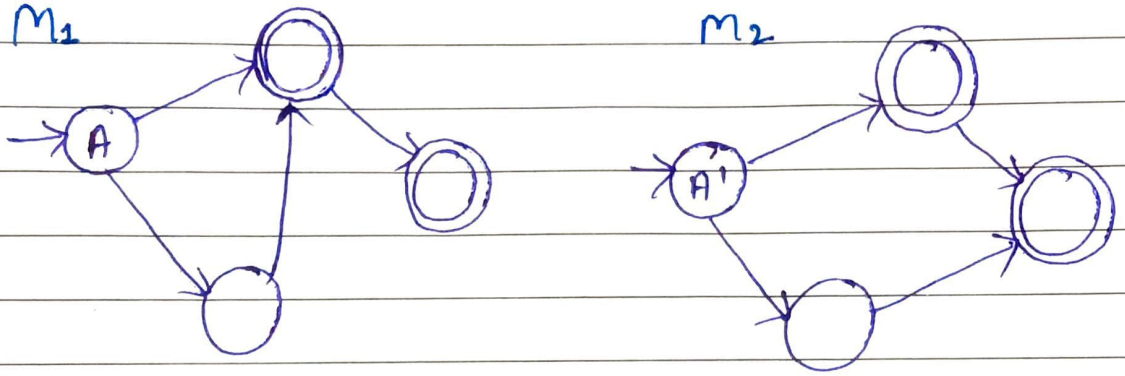
The term closure property relates to a mathematical system that contains the set and the operation performed on the element of the set. Every regular set can be expressed by its corresponding regular exp.

The regular set are closed over the operation of union, concatenation, transpose, complement, intersection, difference & Kleene's star.

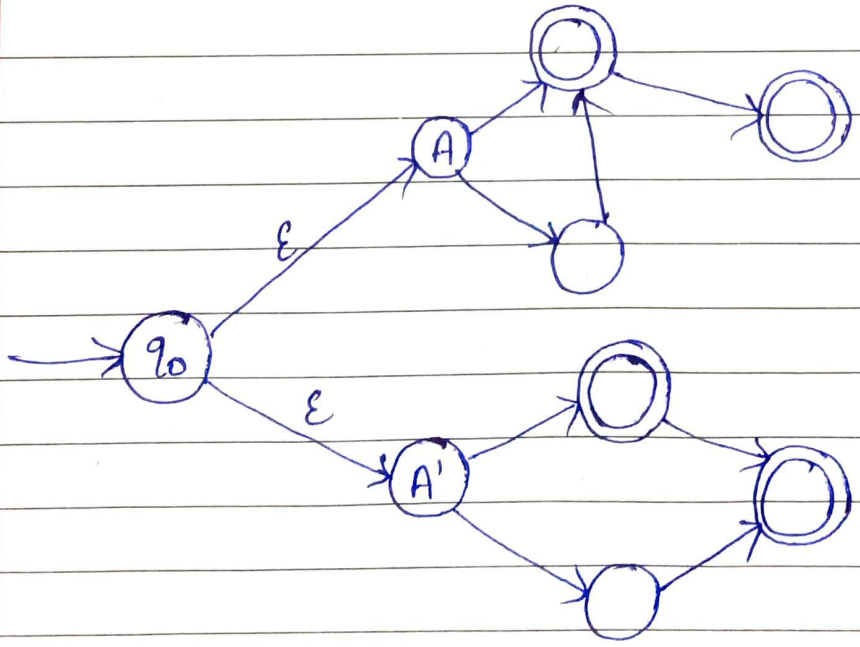
① Regular sets are closed under union operation-

★ **Theorem** - If R_1 & R_2 are two regular sets than $R_1 \cup R_2$ is always regular.

Definition - Since R_1 and R_2 are two regular sets then there exists finite automata M_1 & M_2 accepting R_1 and R_2 respectively.



$M_1 \cup M_2$ =



If L & M are Regular language, so the union is defined by -

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

Proof- Let us take two reg. exp.
$RE_1 = a(aa)^*$ and $RE_2 = (aa)^*$

So-
$L_1 = \{a, aaa, aaaaa, \ldots\}$ (string of even length including Null)

& $L_2 = \{\varepsilon, aa, aaaa, \ldots\}$ (even length strings including Null)

$L_1 \cup L_2 = \{\varepsilon, a, aa, aaa, aaaa, aaaaa, \ldots\}$
⇒ strings of all possible length including Null

$RE(L_1 \cup L_2) = a^*$ (which is a regular expression itself)

② Regular sets are closed under intersection operation-
If R_1 and R_2 are two regular sets then $R_1 \cap R_2$ is also a regular set.

Proof – Let us take two regular expressions
$RE_1 = a(a^*)$ and $RE_2 = (aa)^*$

So, $L_1 = \{a, aa, aaa, aaaa \ldots\}$ (strings of all possible lengths excluding Null)
$L_2 = \{\varepsilon, aa, aaaa, aaaaaa \ldots\}$ (strings of even length including Null)

$L_1 \cap L_2 = \{aa, aaaa, aaaaaa \ldots\}$ (strings of even length excluding Null)
$RE(L_1 \cap L_2) = aa(aa)^*$
which is a regular exp itself.

Name of Lecturer .....

③ Regular sets are closed under complement operation:-
If R is a regular set then $\bar{R}$ is also a regular set.

Proof-
Let us take a reg. exp- RE = $(aa)^*$
so L = {ε, aa, aaaa, aaaaaa.....}
= { even length strings including Null}

Complement of L is all the strings that is not in L.
so,
L' = {a, aaa, aaaaa,.....}
= { strings of odd length excluding Null}
RE(L') = $a(aa)^*$ which is a reg.exp itself.

④ Regular sets are closed under transpose (string reversal) operation:-
If R is regular set then its transpose R^T is also a regular set.

Proof-
let L = {01, 10, 11, 00}
RE(L) = 01+10+11+00

so,
L^R = {10, 01, 11, 00}
RE(L^R) = 10+01+11+00

⑤ Regular sets are closed under Concatenation operation :-

If R_1 and R_2 are two regular sets then $R_1 \cdot R_2$ is always regular.

Proof -

Let $RE_1 = (0+1)^*0$ and $RE_2 = 01(0+1)^*$

Here $L_1 = \{0, 00, 10, 000, 010, \ldots\}$
= (set of strings ending with 0}

and

$L_2 = \{01, 010, 011, \ldots\}$
= (set of strings beginning with '01')

Then $L_1 \cdot L_2 = \{001, 0010, 0011, 0001, 1001, \ldots\}$
= (set of strings containing 001 as a substring).

so -

$RE = (0+1)^*001(0+1)^*$

⑥ Regular sets are closed under kleen's operation :-

If R is a regular set then R^* is always regular.

Proof -

If $L = \{a, aaa, aaaaa, \ldots\}$
= {strings of odd length excluding Null}

i.e. $RE(L) = a(aa)^*$

$L^* = \{a, aa, aaa, aaaa, aaaaa, \ldots\}$
= {strings of all length excluding Null}

$RE(L^*) = a(a)^*$ is a regular exp.

Name of Lecturer : .....

⑦ Regular sets are closed under difference operation :-

If R_1 and R_2 are two regular sets then $(R_1 - R_2)$ is also a regular set.

Proof -

let us take two reg. exp -

$RE_1 = a(a^*)$ and $RE_2 = (aa)^*$

so,

$L_1 = \{a, aa, aaa, aaaa, \ldots\ldots\}$

= (strings of all possible lengths excluding Null)

$L_2 = \{\varepsilon, aa, aaaa, aaaaaa, \ldots\ldots\ldots\}$

= {strings of even length including Null}

$L_1 - L_2 = \{a, aaa, aaaaa, aaaaaaa, \ldots\ldots\}$

= (all odd length strings excluding Null)

$RE.(L_1 - L_2) = a(aa)^*$ which is a reg. exp.

Pumping Lemma for Regular Set:-

"Pumping lemma is a technique for proving a language L is not a regular language." In pumping lemma the lemma gives a necessary condition for an input string that belongs to a regular set. It also gives a method for generating (pumping) many input strings from a given string.

The pumping lemma is useful in the development of algorithms to answer certain questions concerning (regarding) finite automata such as whether the language accepted by given finite automata is finite or not etc.

Hence it is also called - "decision algorithm for regular set."

Note-

⇒ Pumping lemma is used to prove that a language is not Regular.

⇒ It can't be used to prove that a language is Regular.

If A is a Regular language, then A has a Pumping Length 'P' such that any string 's' where $|s| >= P$ may be divided into 3 parts $s = xyz$ such that the following conditions must be true.

① $xy^iz \in A$ for every $i >= 0$

② $|y| > 0$

③ $|xy| <= P$

To prove that a language is not Regular using Pumping Lemma, follow the below steps:
(We prove using contradiction)

→ Assume that A is Regular
→ It has to have a Pumping Length (say P)
→ All strings longer than P can be pumped $|s| >= P$
→ Now find a string 's' in A such that $|s| >= P$
→ Divide s into xyz.
→ show that $xy^iz \notin A$ for some i
→ Then consider all ways that s can be divided into xyz.
→ show that none of these can satisfy all the 3 pumping conditions at the same time
→ S cannot be Pumped == CONTRADICTION.

Ex- Using Pumping Lemma prove that the language $A = \{a^n b^n \mid n >= 0\}$ is not Regular.

Sol-

Assume that A is Regular

Pumping length = P

$S = a^P b^P \Rightarrow S = aaaaaaaabbbbbbbb$

$P = 7$

Case1- The y is in 'a' part -

aa aaaa bbbbbbbb

x y z

$xy^iz \Rightarrow$ for $i = 2$; xy^2z

aaaaaaaaaabbbbbbbb

here $a = 11$ & $b = 7$

so $a \neq b$ $\{11 \neq 7\}$

Case2- The y is in 'b' part -

aaaaaaa bbbbbbbb

x z

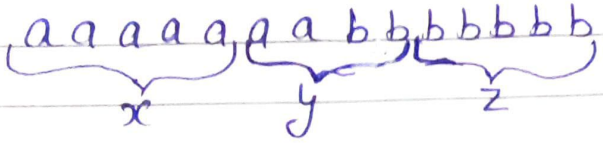
$xy^iz \Rightarrow$ for $i = 2$; xy^2z

aaaaaaabbbbbbbbbbbbbb

here $a = 7$ and $b = 7$

so $a \neq b$ $\{07 \neq 11$

Case3- The y is in 'a' & 'b' part -



$xy^iz \Rightarrow$ for $i = 2$; xy^2z

aaaaaaaabbaabbbbbbb

This string doesnot follow the condition a^nb^n

So it is proved that the language a^nb^n is not Regular.

✱ Applications of Pumping lemma -

Pumping lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is Regular.

⇒ If L is regular, it satisfies Pumping lemma

⇒ If L doesnot satisfy Pumping lemma, it is non-regular

Note:- If a language doesnot satisfy Pumping lemma, it is non-Regular language, but if language satisfy pumping lemma then it is un-decidable that whether the language is Regular or Non-Regular

Myhill-Nerode Theorem -

Myhill Nerode theorem was proposed by John Myhill and Anil Nerode.

If the language L can be implemented on on the finite state machine (FSM) and each state in the FSM corresponds to a particular class. Then the FA for the language L will contain a minimum of two states but if the states are more than two then we can divide into two categories. One where the string class belonging to class C1 reach the acceptor state and the other where the string belonging to class C2 reach the non-acceptor state. Hence language L can be implemented using finite automata.

"According to Myhill-Nerode theorem any language 'L' divides the set of all possible strings into separate (mutually exclusive) classes. The language L is regular if and only if the no. of classes is finite."

Ex- Let the language L of all strings, ending with b, defined over $\Sigma = \{a, b\}$ [check string is regular]

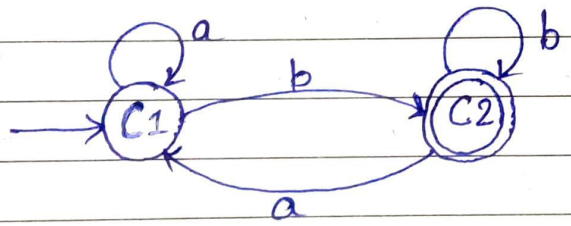
Solution-

It can be observed that L partitions Σ^* into the following classes-

C1 = set of all strings ending with a.
C2 = set of all strings ending with b.

Since these are finite classes. so L is regular language.

So following is the FA accepting L.



✯ Power of Language-

✯ Power of Alphabet (Σ) :-

If Σ is an alphabet then Σ^* is the set of all the strings from the alphabet Σ of length exactly k.

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

Σ^0 = set of all strings of length 0.
$= \{\epsilon\}$

Σ^1 = set of all strings of length 1.
$= \{a, b\}$

Σ^2 = set of all strings of length 2.
$= \{aa, ab, ba, bb\}$

Σ^3 = set of all strings of length 3.
$= \{aaa, bbb, aab, aba, bba, bab, abb, baa\}$

⋮

Σ^n = set of all strings of length n.
$= \{aaabaababababa \ldots\ldots, abab \ldots\ldots\}$

Positive closure of $\Sigma \Rightarrow$ '+'

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \Sigma^4 \ldots\ldots\ldots$

Kleene closure of $\Sigma \Rightarrow$ '*'

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \ldots\ldots\ldots$