Theory - Of - Computation

Unit - 01

Intro :- Alphabets, strings, and languages; Automata and Grammars, Deterministic finite Automata (DFA) - Formal Definition, Simplified notation : state transition graph, Transition table, Language of DFA, Nondeterministic finite Automata (NFA), NFA with epilson transition, language of NFA, Equivalence of NFA and DFA, Minimization of finite Automata, Distinguishing one string from other, Myhill - Nerode Theorem.

Alphabets, strings, and languages.

1. Alphabets :-

An Alphabet is a finite set of symbols.
Symbols are the basic building blocks of strings and languages
Denoted by the greek letter Σ (Sigma)

Ex :-

- $\Sigma = \{0, 1\}$ → Binary Alphabet.
- $\Sigma = \{a, b, c\}$ → Alphabet of English lowercase letters (subset)

2. strings :-

A string is a finite sequence of symbols taken from an alphabet
It can have zero or more symbols.

Important Terms :-

- length of string (|w|): Number of symbols in a string
  → Example : |101| = 3.

- Empty string (|ε|) : string of length zero
  → Example : ε, |ε| = 0

- Concatenation : Joining of two strings
  → Example : "ab" ∘ "ba" = "abba"

3. language (L)

A language is a set of strings formed from an alphabet.
It can be finite or infinite.

Examples :

$L = \{\varepsilon, 0, 1, 11, 00\}$

$L = \{w \in \{0,1\}^* \mid w \text{ contains even number of 0s}\}$

4. Operations on languages.

Let L_1 and L_2 be two languages over the same alphabet.

Operation	Definition	Example
Union ($L_1 \cup L_2$)	All strings in L_1 or L_2 or both	$L_1 = \{a, b\}, L_2 = \{b, c\} \rightarrow L_1 \cup L_2 = \{a, b, c\}$

Concatenation (L_1L_2) — All strings formed by combining strings from L_1 and L_2 — $L_1 = \{a\}, L_2 = \{b, c\} \rightarrow L_1L_2 = \{ab, ac\}$

Kleene Star (L^*) — All possible combinations of 0 or more strings from L — $L = \{a\} \rightarrow L^* = \{\epsilon, a, aa, aaa, \ldots\}$

5. String Notations

Σ^* (Kleene closure): Set of all possible strings, including ϵ
Σ^+ (Positive closure): All strings excluding ϵ

6 Important Notes

Σ^* is infinite even if Σ is finite.
Every language is a subset of Σ^*, i.e, $L \subseteq \Sigma^*$
languages are used to define grammars, automata, and computational models

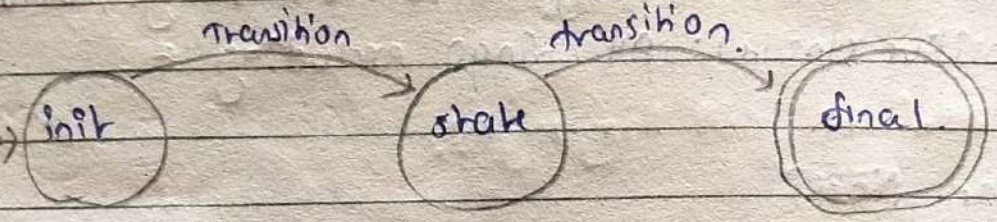
Automata and Grammars,

Concept of Automata

Automata (plural of Automaton) are mathematical models of computation that define abstract machines capable of reading input strings and determining their

acceptance or rejection based on a set of rules. These models are used to describe and analyze the behavior of hardware, software, and languages.

An automaton consist of a finite set of states, an input alphabet, a transition function, a ~~start~~ start state, and a set of accepting states. The study of automata forms the basis of theoretical computer science, compiler design, and formal language theory.



Concept of Grammars

A grammar is a set of formal rules that define the syntax of a language. It is used to generate strings that belong to a language. Each grammar consists of:-

- A set of variables (non-terminal symbols)
- A set of terminal symbols (alphabet)
- A set of production rules.
- A start symbol.

Grammars are used in language parsing, compiler construction, and natural language processing. They help define valid strings and their structure within a language.

The combination of automata and grammars provides a foundation ~~of~~ for understanding how machines process languages and forms the core of formal language theory.

Finite State Machines (FSM).

A Finite State Machine (FSM) is a theoretical model of computation used to design both computer programs and sequential logic circuits. It consists of a finite number of states, transitions between those states, and input symbols that trigger those transitions.

FSM accepts input strings and either accepts or rejects them based on the defined rules. It is a core model in automata theory, widely used in lexical analyzers, protocol design, control systems, and more.

FSM can be categorized into two types:-

- Deterministic Finite Automata (DFA)
- Non Deterministic Finite Automata (NFA)

Components of FSM.

An FSM is defined as a 5-tuple:

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

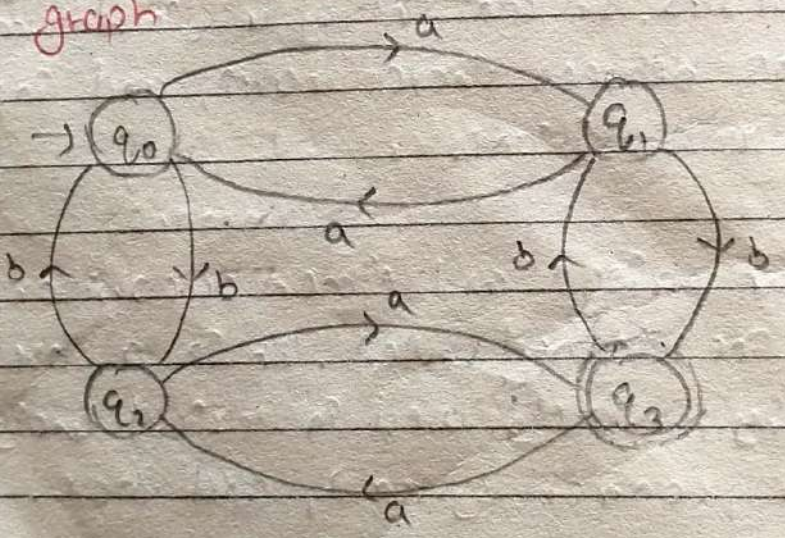
Where:

- Q = Finite set of states
- Σ = Input alphabet (symbols)
- δ = Transition function ($Q \times \Sigma \to Q$)
- q_0 = Initial state ($q_0 \in Q$)
- F = set of final / accepting states ($F \subseteq Q$)

FSM Diagram Example

Ex 1:

Transition graph



Ip Input string :- abbaba

Soln :- We know :-

$Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$

Initial state (q_0) = q_0, final state = q_3

$(q_0 \quad abbaba) \vdash q_1$

$(q_1 \quad bbaba) \vdash q_3$

$(q_3 \quad baba) \vdash q_1$

$(q_1 \quad aba) \vdash q_0$

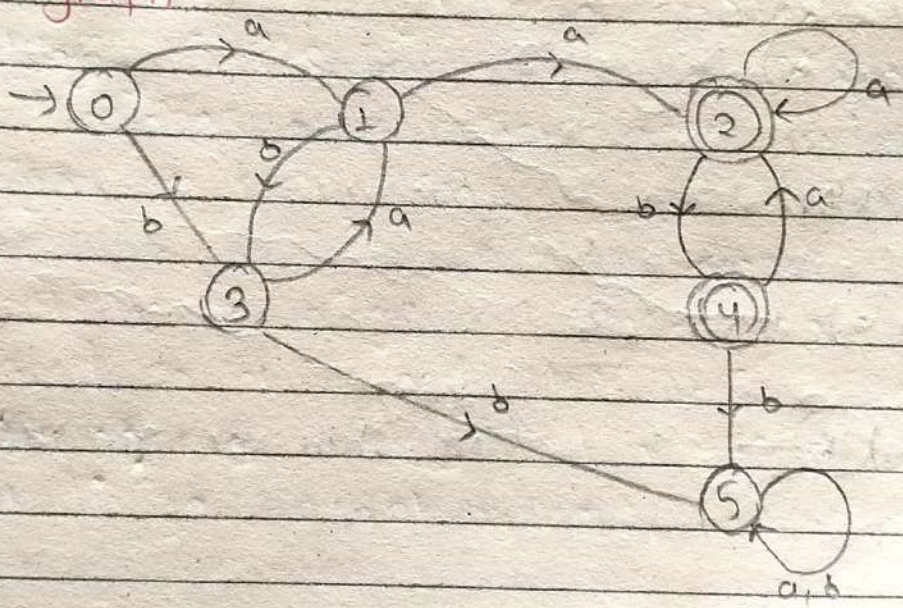
$(q_0 \quad ba) \vdash q_2$

$(q_2 \quad a) \vdash q_3$ (final state accepted).

δ	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

Ex2:

Transition graph:-



Input string :- babaabaab

Sol^n :-

Q = {0, 1, 2, 3, 4, 5}
Σ = {a, b}
Initial state = 0
final state = 5

(0 babababaab) ⊢ 3
(3 ababaaab) ⊢ 1
(1 babaaab) ⊢ 3
(3 abaaab) ⊢ 1
(1 baaab) ⊢ 3
(3 aaab) ⊢ 1
(1 ab) ⊢ 2
(2 b) ⊢ 4 final state not accepted

δ	a	b
→0	1	3
1	2	3
2	2	4
3	1	5
4	2	5
5	5	5

Deterministic Finite Automata

Formal Definition :-

A deterministic finite Automata (DFA) is a 5-tuple represented as :

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

Where :-

- Q = Finite set of states
- Σ = Finite input alphabet
- δ = Transition function ($\delta : Q \times \Sigma \to Q$)
- q_0 = start state ($q_0 \in Q$)
- F = Set of accepting / final states ($F \subseteq Q$)

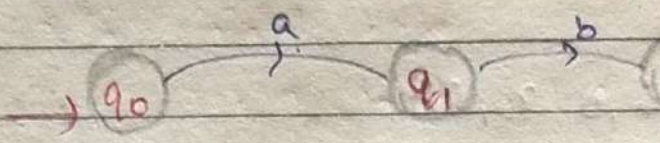
The DFA reads an input string symbol-by-symbol and changes its state according to the transition function. If after reading the entire string, the automation ends in an accepting state, the string is accepted.

Simplified Notation.

Instead of writing the full 5-tuple every time, we use the following simplified representation.

- States :- Represented by circles labeled with state names (eg. q_0, q_1).

- Start state :- Arrow pointing to the start state.
- Final state (s) :- Double-circled Nodes.
- Transitions :- Arrows between states labeled with input symbols.

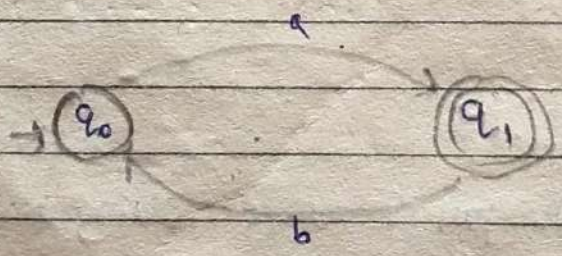


State transition graph.

A graphical representation of a finite automaton.

Components :-

(a) Nodes (circles) → Represent states.
(b) Directed edges → Represent transitions between states.
(c) Labels on edges → Indicate input signal symbols causing the transition.
(d) Start state → Usually marked with an incoming arrow.
(e) Final state → Indicated by a double circle.



State transition table

A tabular representation of state transitions for all possible inputs.

structure :-

Present state	Input(a)	Input(b)
q_0	q_1	q_0
q_1	q_1	q_0

Rows → states.
Columns → Input symbols
Entries → Next state after given input.

~~Language of DFA.~~

Non Deterministic Finite (NFA).

An NFA is similar to a DFA but allows :
- multiple transitions for the same input symbol from a state.
- ε-transitions (moves without consuming input).

An NFA accepts a string if at least one computation path ends on a final state.

Formally :-

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

- Q = set of states
- Σ = Input alphabet
- $\delta = Q \times (\Sigma \cup \varepsilon) \to P(Q)$ (transition function gives a set of states)
- q_0 = Initial state
- F = set of final states

Key Points.

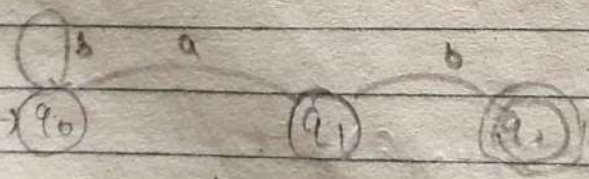
- Determinism vs Non-determinism :
  - DFA - Only one possible next state for each symbol.
  - NFA - Can have many or none possible next states.
- Power : NFA and DFA recognize the same class of languages (regular languages).
- Conversion :- Any NFA can be converted to an equivalent DFA

Diagram (Example)

Example NFA : Accepts all strings ending with 'ab'.

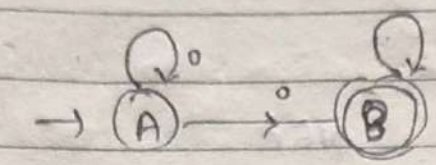
states : $Q = q_0, q_1, q_2$

- $q_0 \to$ start
- $q_2 \to$ final



Conversion from NFA to DFA

Ex 1.



$Q = [A, B]$

$\Sigma = [0, 1]$

Initial state = A

Final state = B

NFA

	0	1
→ A	A, B	ϕ
B	ϕ	B

DFA

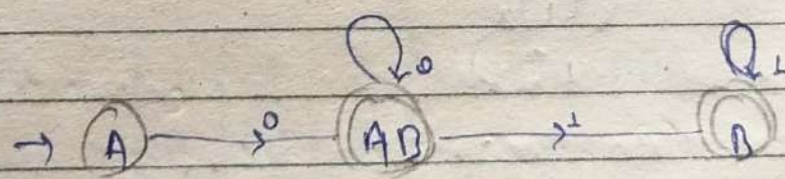
δ	0	1
→ A	[A, B]	–
[AB]	[AB]	B
B	–	B

A × 0 ⊢ A, B } union
B × 0 ⊢ ϕ.

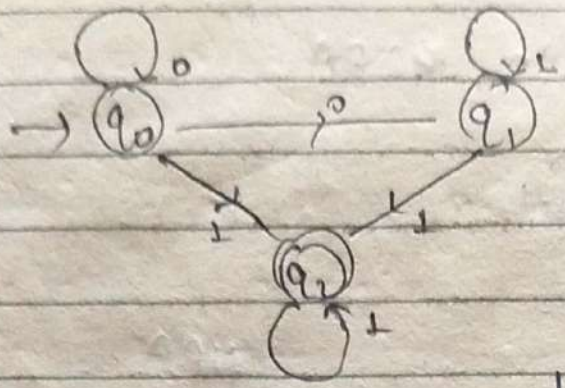
A × 1 ⊢ ϕ } union
B × 1 ⊢ B

Rules:-

1. Keep all states, start state, and final states same
2. Keep the same alphabet Σ as in DFA
3. Rewrite transition function to return a set of states instead of a single state.
   - for DFA: $\delta(q, a) =$
   - for NFA: $\delta'(q, a) = P$
4. No ε-transitions are added unless specifically required for a different design
5. The resulting NFA is identical in behavior to the original DFA.



Ex:



NFA

	0	1
→ q_0	q_0, q_1	q_2
q_1	ϕ	$q_1 q_2$
q_2	ϕ	q_2

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

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

I.S = q_0

F.S = q_2

DFA.

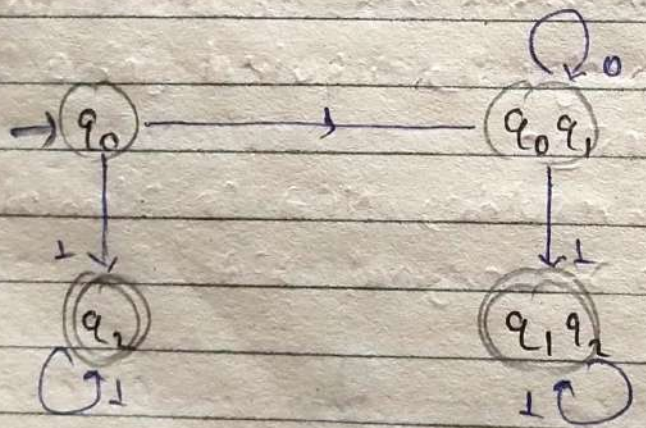
δ	0	1
→ q_0	$q_0 q_1$	q_2
$q_0 q_1$	$q_0 q_1$	$q_1 q_2$
$q_1 q_2$	ϕ	$q_1 q_2$
q_2	ϕ	q_2

$q_0 \times 0 \vdash q_0 q_1$ } $q_1 \times 0 \vdash \phi$ } ϕ

$q_1 \times 0 \vdash \phi$ $q_2 \times 0 \vdash \phi$

$q_0 \times 1 \vdash q_2$ } $q_1 \times 1 \vdash q_1 q_2$

$q_1 \times 1 \vdash q_1 q_2$ $q_2 \times 1 \vdash q_2$



NFA with epsilon transition.

An NFA with ε-transitions allows the automation to change its state without consuming any input symbol. ε (epsilon or lambda) means "empty string" or "no input".

Enables more flexible designs and simplification in automata construction.

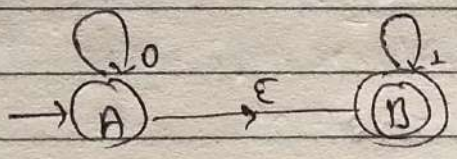
Formal Definition.

An ε-NFA is a 5-tuple:

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

- Q = Finite set of states
- Σ = Input alphabet (excluding ε)
- $\delta : Q \times (\Sigma \cup \varepsilon) \to 2^Q$ = Transition func
- q_0 = start state
- $F \subseteq Q$ = Final state

Conversion.



NFA :-

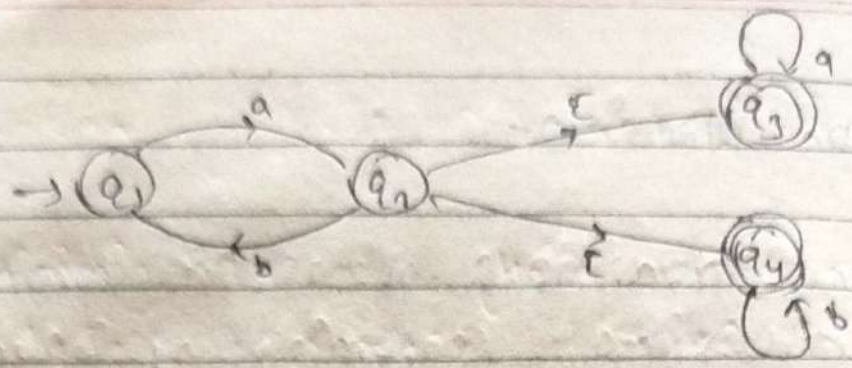
	0	1
→A	AB	B
B	ϕ	B

⇒

DFA →

	0	1
→A	AB	B
F.S AB	AB	B
F.S B	ϕ	B

Ex:-



NFA

	a	b
$\to q_1$	q_1, q_3, q_4	ϕ
q_2	q_3	$q_2 q_4$
F.S q_3	q_3	ϕ
F.S q_4	ϕ	q_4

DFA

	a	b
$\to q_1$	q_1, q_3, q_4	ϕ
F.S q_1, q_3, q_4	q_3	$q_2 q_4$
F.S $q_2 q_4$	q_1, q_3, q_4	q_4
F.S q_3	q_3	ϕ
F.S q_4	ϕ	q_4

Equivalance of NFA and DFA

Theorem:- For every NFA, there exists an equivalent DFA that recognizes the same language.
This means NFA and DFA have the same computational power (both accept exactly the class of regular languages).

Reason:-

An NFA may have multiple possible moves for the same input, or even ε-moves.

A. DFA is deterministic, meaning only one possible move for each state and input.
Using the subset construction (powerset) method, we can transform an NFA into a DFA.

Subset Construction Method.

Start state:
DFA's start state = ε-closure of NFA's start state.

States:
Each DFA state = a set of NFA states.

Transitions:
For each DFA state (set of NFA states) and input symbol, find all reachable NFA states (including ε-closure).

Final states:-
Any DFA state containing at least one NFA final state is a DFA final state.

P.T.O.

Minimization of finite Automata.

The minimization of a finite automation is the process of reducing the number of states in a DFA while preserving the same language it recognizes. The minimized DFA is unique (up to renaming of states).

Need for Minimization

1. To make the automaton more efficient in terms of memory and computation.
2. To eliminate redundant states that do not affect the accepted language.
3. To get a unique minimal representation of the language.

Steps for Minimization of DFA

Remove Unreachable States

- Identify states that cannot be reached from the initial state and eliminate them.

Partition states into Groups

- Initially divide states into two groups:
  - Final (accepting) states.
  - Non-final (non-accepting) states.

Rajshree
PAGE NO.:
DATE:

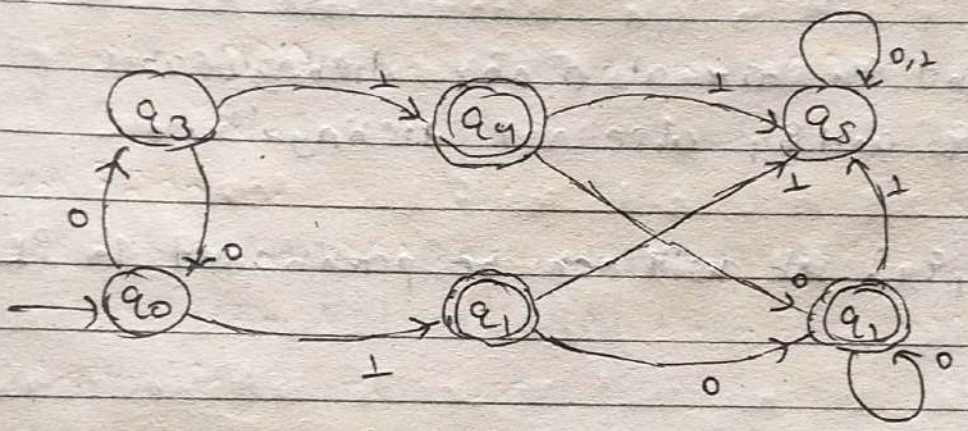
Refine Partitions:

- Split groups further based on transitions.
- Two states are equivalent if for every input symbol, their transitions go to states in the same group.

Repeat Until no further Division

- Keep refining until no new partition is possible.
- Each group of equivalent states represents one state in the minimized DFA.

~~Diagram~~ Example:-



Step1: P0 will contain two sets, One set have final states q_1, q_2, q_4 and another set have remaining states. ~~P0~~ = S0,

$$P_0 = \{[q_1, q_2, q_4], [q_0, q_3, q_5]\}.$$

Step2: To calculate P_1, we will check whether sets of partitions P0 can be partitioned or not:-

$P_0 = [A, B, C \ldots]$

(b) For set $[q_1, q_2, q_4]$: If A_0 and $A_1 \times 0 \to \in A$ = not distinguishable
else if A_0 and $A_1 \times 0$ or $1 \to \notin A$ = distinguishable

• Here

$q_1 \times 0 \vdash q_2$ and $q_1 \times 1 \vdash q_5$

$q_2 \times 0 \vdash q_2$ $q_2 \times 1 \vdash q_5$

Hence q_1 and q_2 are not distinguishable

Similarly :-

$q_1 \times 0 \vdash q_2$ and $q_1 \times 1 \vdash q_5$

$q_4 \times 0 \vdash q_2$ $q_4 \times 1 \vdash q_5$

Hence q_1 and q_4 are also not distinguishable.
Therefore q_2 and q_4 also not distinguishable

So, $[q_1, q_2, q_4]$ set will not be partitioned in P_1

(b) For set $[q_0, q_3, q_5]$:-

Here

$q_0 \times 0 \vdash q_3$ and $q_0 \times 1 \vdash q_1$

$q_3 \times 0 \vdash q_0$ $q_3 \times 1 \vdash q_4$

moves of q_0 and q_3 with input 0 and are q_3 and q_0 which are in same set of in partition P_0

Similarly q_0 and q_3 with input 1 are q_1 and q_4 which are moves of

in same set in partition P0.
So, q0 and q5 are not distinguishable

Now,

$q_0 \times 0 \vdash q_3$ and $q_0 \times 1 \vdash q_1$

$q_5 \times 0 \vdash q_5$ $q_5 \times 1 \vdash q_5$

moves of q0 and q5 with input 0 are q3 and q5 which are in same set of partition P0 but the moves of q0 and q5 with input 1 are q1 and q5 which are not in same set of partition in P0.

Therefore

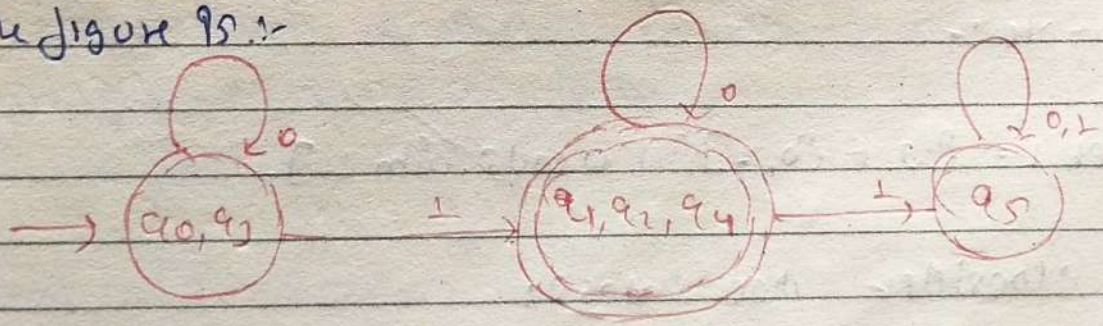
q0 and q5 are distinguisable

So,

set {q0, q3, q5} will be partitioned into {q0, q3} and {q5}. So,

$P_1 = [[q_1, q_2, q_4], [q_0, q_3], [q_5]]$

Hence Required DFA:-



Distinguishing one string from other

In the Theory of automata, two strings are said to be distinguishable with respect to a language L if there exists at least one extension (a string appended to them) such that one string belongs to the language after the extension, and the other does not.

Formal Definition

let x and y be two strings over alphabet Σ.

- strings x and y are distinguishable w.r.to L if there exists a string $z \in \Sigma^*$ such that:

$$xz \in L \text{ and } yz \notin L \quad \text{or}$$
$$x \notin L \text{ and } yz \in L$$

- otherwise, they are said to be the indistinguishable (equivalent).

Example :-

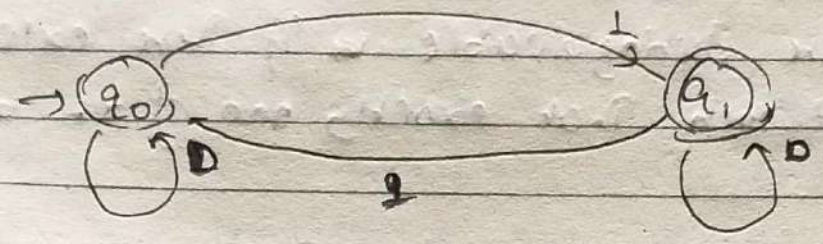
let $L = \{w \in \{0,1\}^* \mid w \text{ ends with } 1\}$.

- Consider two strings:
  - $x = 0, y = 1$

- Choose extension $z = \epsilon$ (empty string):
  - $xz = 0 \notin L$
  - $yz = 1 \in L$

Thus, x and y are distinguishable wrt L.

Diagram :-



Here

$q_0 \times 0 \vdash q_0$ and $q_0 \times 1 \vdash q_1$

$q_1 \times 0 \vdash q_1$ $q_1 \times 1 \vdash q_0$

Myhill - Nerode Theorem.

The Myhill - Nerode Theorem provides a characterization of regular languages using the idea of distinguishable strings.

It establishes a strong connection between languages, equivalence relations, and deterministic finite automata (DFA).

Statement of the Theorem

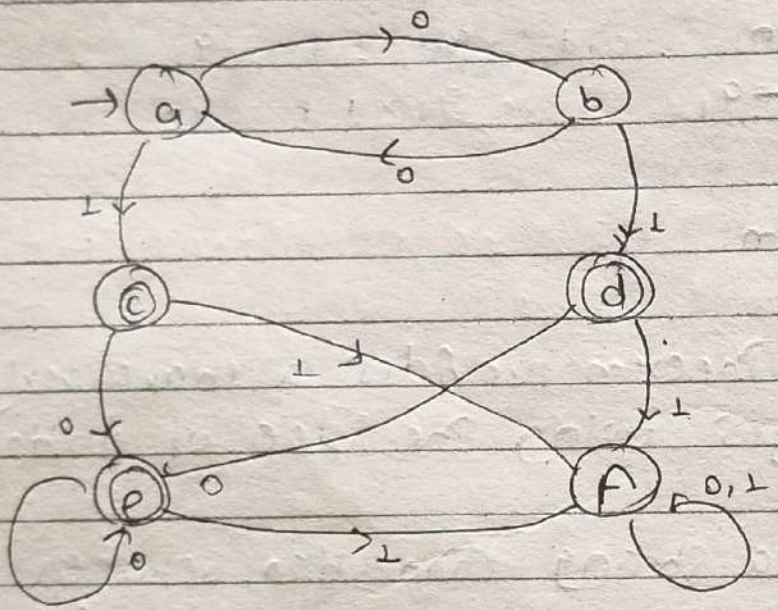
A language $L \subseteq \Sigma^*$ is regular if and only if the relation R_L defined as :-

$x R_L y \iff \forall z \in \Sigma^*, (xz \in L \iff yz \in L)$

has finite index

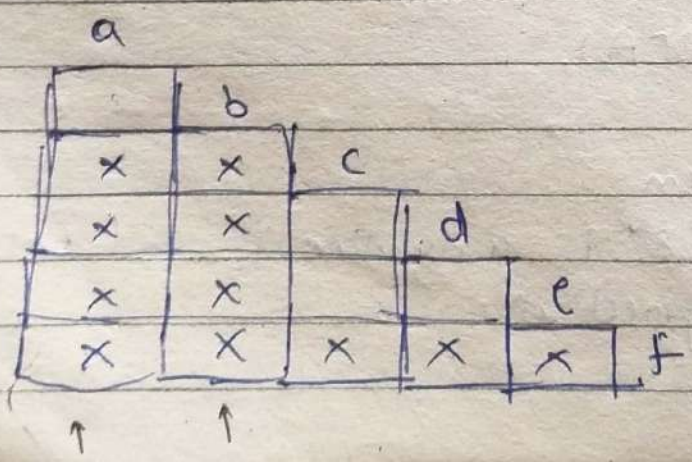
- x and y are indistinguishable wrt L if no continuation z can seperate them (both remain either inside or outside L).
- x and y are distinguishable if there exists some extension z such that exactly one of $xz, yz \in L$

Example:-



Non final
{a, b, f}

final
{c, d, e}.



(a, b)	(c, d)	(c, e)	(d, e)
$a \times 0 \vdash b$	$c \times 0 \vdash e$	$c \times 0 \vdash e$	$d \times 0 \vdash e$
$b \times 0 \vdash a$	$d \times 0 \vdash e$	$e \times 0 \vdash e$	$e \times 0 \vdash e$
$a \times 1 \vdash c$	$c \times 1 \vdash f$	$c \times 1 \vdash f$	$d \times 1 \vdash f$
$b \times 1 \vdash d$	$d \times 1 \vdash f$	$e \times 1 \vdash f$	$e \times 1 \vdash f$

(a, f)	(b, f)
$a \times 0 \vdash b$	$b \times 0 \vdash a$
$f \times 0 \vdash f$	$f \times 0 \vdash f$
$a \times 1 \vdash c$	$b \times 1 \vdash d$
$f \times 1 \vdash f$	$f \times 1 \vdash f$

Hence

$a = b$,

$c = d$, $c = e$, $\Big\}$ $c = d = e$

$d = e$,

f.