→ Non-Deterministic Finite Automata (NFA/NDFA)

NFA: The FA which has 0 or more transitions for any input symbol from any state is NFA.

→ NFA is a M(Mathematic model) is defined as a 5 tuple variables

$$M = \{Q, \Sigma, \delta, q_0, F\} \text{ where}$$

Q is finite set of states

Σ is finite set of input symbols / alphabets

q_0 is initial state

F is set of final states

δ is Mapping / Transition function

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

Note:

① The Accepting power of NFA and DFA is same

$$\therefore L(DFA) = L(NFA)$$

② NFA is more powerful than DFA and DFA is more efficient than NFA.

③ The construction of NFA is easier than DFA and understand the logic is easy in NFA

④ There is no concept of dead state in NFA

⑤ NFA takes care of valid transitions and No need to take care of Invalid transitions.

⑥ In NFA, the transition part is not unique corresponding to any string

⑦ NFA is a kind of parallel computing system where multiple threads can run concurrently.

⑧ NFA is an incomplete system. Since it is not requested to define the transition for each and every symbol at each and every state.

→ Acceptance by NFA :-

Let W be any string defined over alphabet corresponding to W, there may be multiple transitions starting from initial state.

⇒ out of all these transitions, if there exist atleast one transition path which starts at initial state, ends in any one of Final states, then the string W is accepted by NFA.

* → The set of all the strings which are accepted by NFA is known as language of NFA

$$L(NFA) = \{ x \text{ or } W \mid \delta(q_0, x) \text{ or } W = FS \}$$

$$= \{ W \mid \delta(q_0, W) \cap F \neq \emptyset \}$$

where F is set of final states

* → Every DFA is NFA

→ Construct NFA for following language

① $L = \{ W \in (a+b)^* / W$ starts with a $\}$

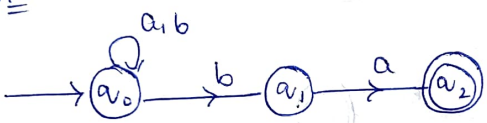
$RE = a(a+b)^*$

NFA

② $L = \{ W \in (a+b)^* / W$ ends with ba $\}$

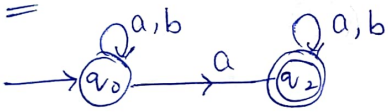
$R = (a+b)^* ba$

NFA



③ $L = \{ W \in (a+b)^* / W$ contains 'a' $\}$

NFA



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

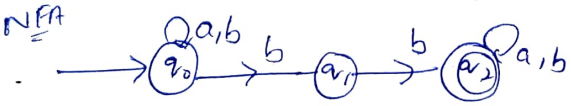
④ $L = \{ W \in (a+b)^* / W$ starts with ab $\}$

NFA

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

⑤ $L = \{ W \in (a+b)^* / W$ contains 'bb' $\}$

NFA



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

Assignment

① $L = \{ W \in (a+b)^* / W$ starts and ends with a $\}$

② $L = \{ W \in (a+b)^* / W$ starts and ends with same symbol $\}$

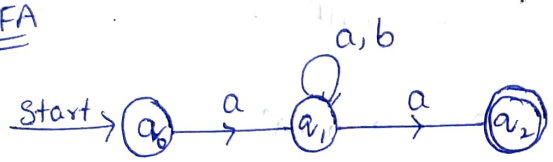
③ $L = \{ W \in (a+b)^* / W$ " " different symbol $\}$

④ $L = \{ u / W$ starts with aa or bb $\}$

⑤ $L = \{ w \in (a+b)^* / n^{th}$ symbol from LHS is always b $\}$

① A) RE = $a(a+b)^* a$

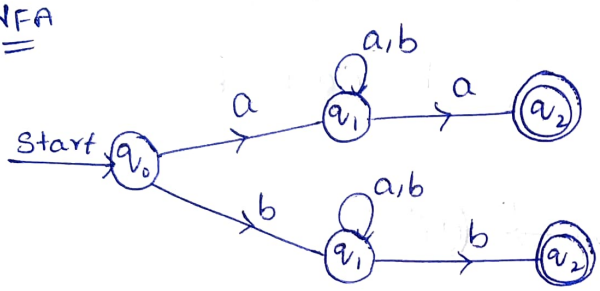
NFA



② A)

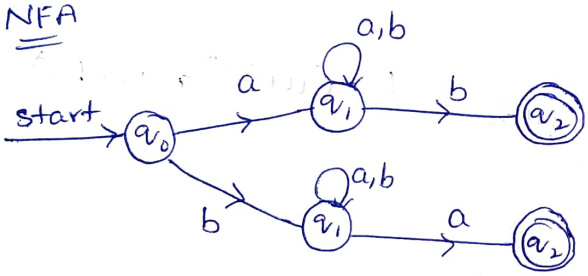
~~RE = a|b(a+b)* a|b~~ RE = $a + a(a+b)^* a + b + b(a+b)^* b$

NFA



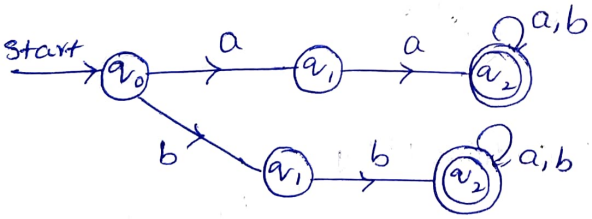
③ A)

NFA



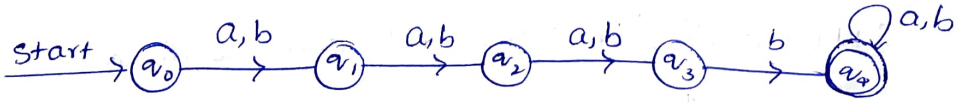
④ A)

NFA



⑤ A)

NFA



→ NFA to DFA conversion (Subset construction)

procedure for subset construction :-

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

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

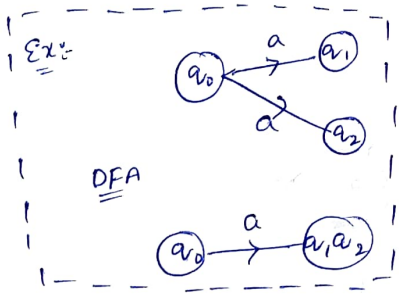
① Initial state

$q_0' = q_0$ (No change in initial state)

② Generation of Transition Function (δ' from δ)

$$\delta'(q, n) = \delta(q, n)$$

$$\delta'((q_0, q_1, q_3 \cdots q_n), k) = \bigcup_{i=1}^{n} \delta(q_i, k)$$



→ start the construction of δ' with the initial state and continue the process for every new state that comes under input column and terminate the process whenever no new state appears under input column.

③ Final states

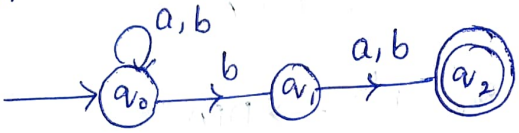
Every subset that contain the final state of NFA is a final state DFA

Ex: Construct DFA from NFA by subset constructor

$L = \{ w \in (a+b)^* / $ 3rd from R.H.S is b$\}$

Sol:

NFA



T.T for NFA

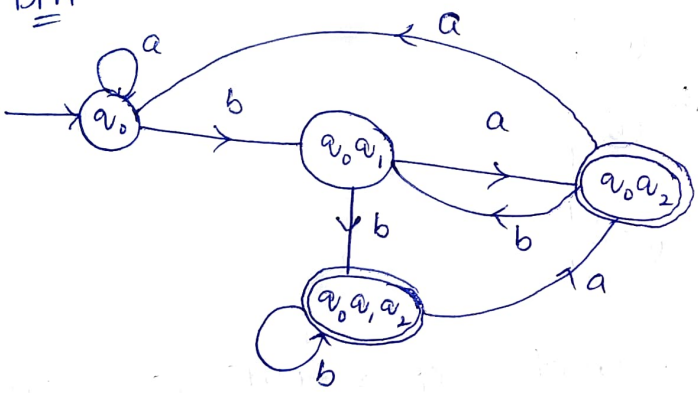
	a	b
→ q_0	q_0	$\{q_0, q_1\}$
q_1	q_2	q_2
* q_2	∅	∅

TT for DFA

	a	b
→ q_0	q_0	q_0q_1
q_0q_1	q_0q_2	$q_0q_1q_2$
* q_0q_2	q_0	q_0q_1
* $q_0q_1q_2$	q_0q_2	$q_0q_1q_2$

$\delta'((q_0,q_1),b) = \delta(q_0,b) \cup \delta(q_1,b)$

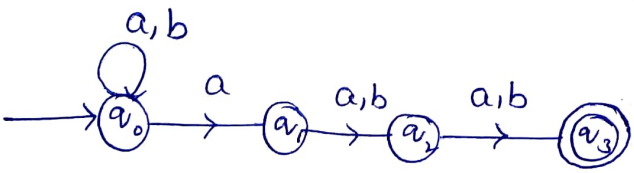
DFA



Ex: Construct DFA from NFA

$L = \{ w \in (a+b)^* / $ 3rd symbol from R.H.S is a$\}$

Sol: NFA



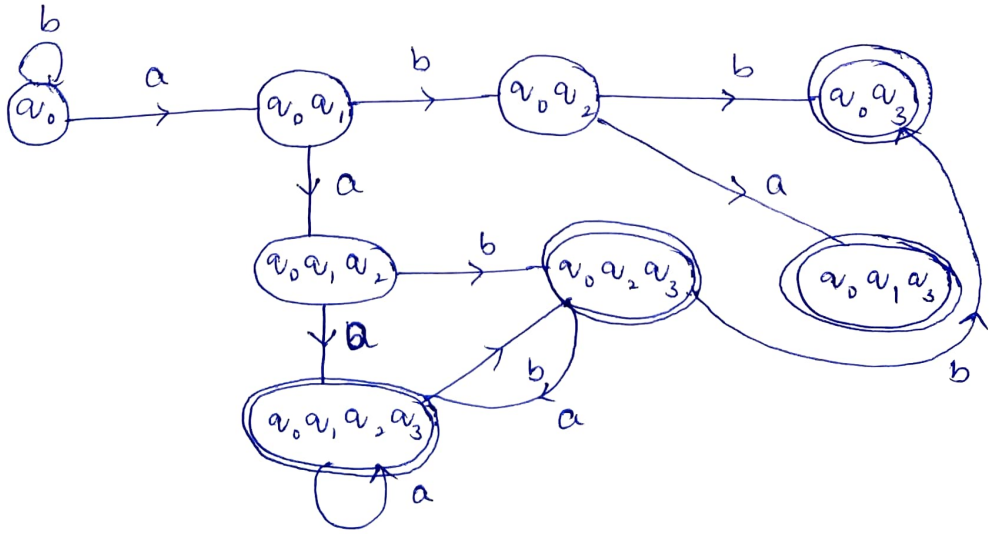
TT for NFA

	a	b
→ q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	q_2
q_2	q_3	q_3
* q_3	$\emptyset$	$\emptyset$

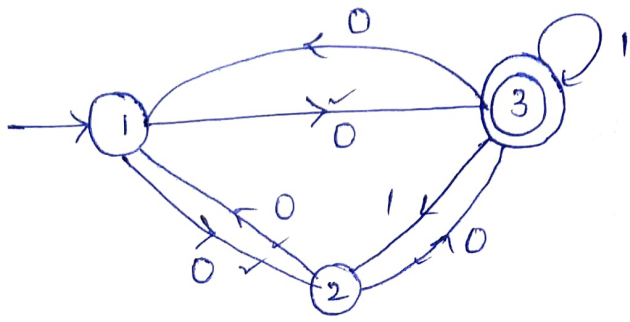
TT for DFA

	a	b
→ q_0	q_0q_1	q_0
q_0q_1	$q_0q_1q_2$	q_0q_2
$q_0q_1q_2$	$q_0q_1q_2q_3$	$q_0q_2q_3$
q_0q_2	$q_0q_1q_3$	q_0q_3
* $q_0q_1q_2q_3$	$q_0q_1q_2q_3$	$q_0q_2q_3$
* $q_0q_2q_3$	$q_0q_1q_2q_3$	q_0q_3
* $q_0q_1q_3$		
* q_0q_3	q_0q_1	q_0

DFA



① Construct DFA from NFA



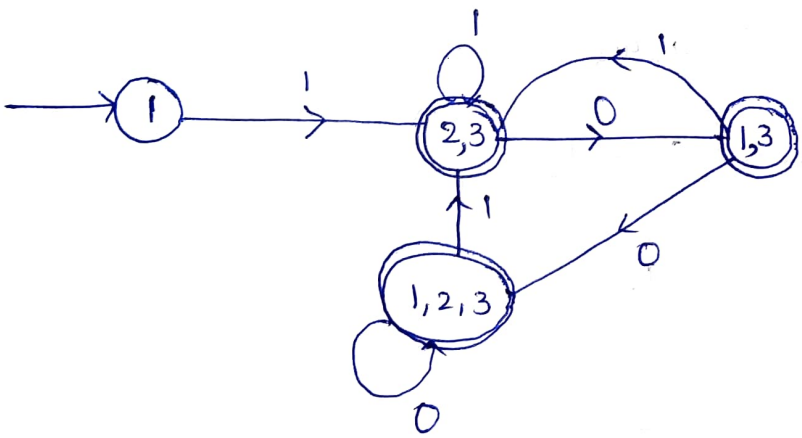
TT for NFA

		0	1
→	1	{2,3}	-
	2	{1,3}	-
*	3	1	{2,3}

TT for DFA

		0	1
→	1	[2,3]	-
*	[2,3]	[1,3]	[2,3]
*	[1,3]	[1,2,3]	[2,3]
*	[1,2,3]	[1,2,3]	[2,3]

DFA



NFA with Epsilon Moves / ε-NFA

Non Def :- NFA which has transitions even for epsilon (ε)

For Def :- The ε-NFA (M) is defined as a 5-tuple variable

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

Q = set of finite states

Σ = set of input alphabets

q_0 = initial state

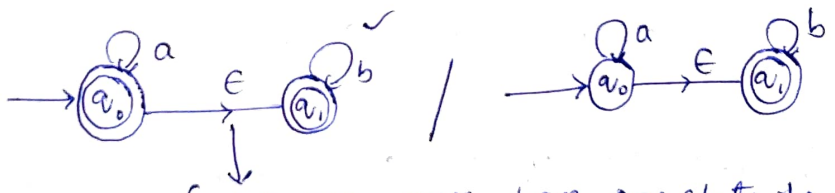
F = Finite set of Final states

δ : Transition function

$$Q \times \Sigma \cup \{\epsilon\} \longrightarrow 2^Q$$

Ex for ε-NFA :-

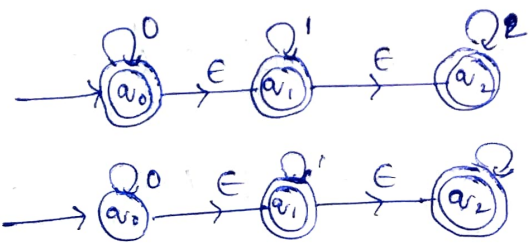
① $L = \{ a^n b^m / m, n \geq 0 \}$



(We can move from one state to another state without reading any ~~state~~ alphabet)

$a \cdot \epsilon \cdot b = ab$

② $L = \{ 0^m 1^n 2^p / m, n, p \geq 0 \}$



→ NFA with Epsilon Moves / ε-NFA

Non Def :- NFA which has transitions even for epsilon (ε)

For Def :- The ε-NFA (M) is defined as a 5-tuple variable

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

Q = set of finite states

Σ = set of input alphabets

q_0 = Initial state

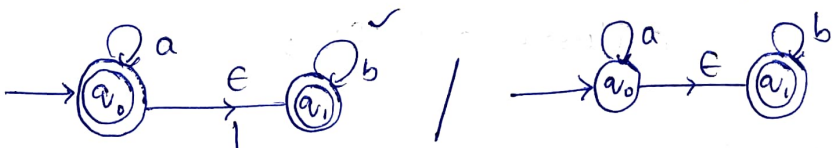
F = Finite * set of Final states

δ : Transition function

$$Q \times \Sigma \cup \{\epsilon\} \longrightarrow 2^Q$$

Ex for ε-NFA :-

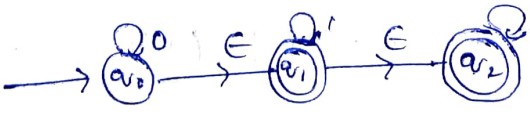
① $L = \{ a^n b^m / m, n \geq 0 \}$



(We can move from one state to another state without reading any ~~state~~ alphabet)

a . ε . b = ab

② $L = \{ 0^m 1^n 2^p / m, n, p \geq 0 \}$

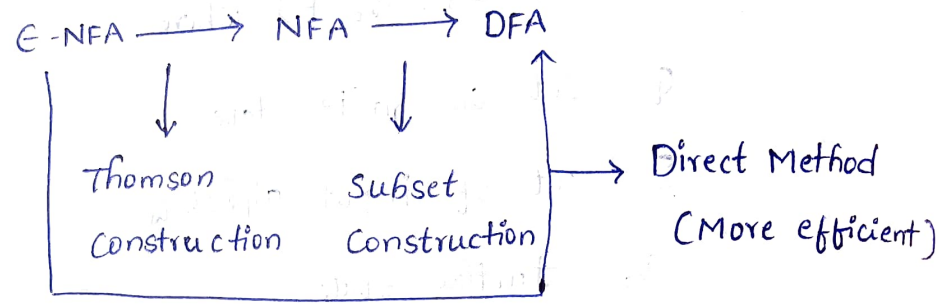


Note:-

① The accepting power of NFA, ϵ-NFA and DFA is same.

$$L(\epsilon\text{-NFA}) = L(\text{NFA}) = L(\text{DFA})$$

②



→ ϵ-closure(q) :-

Ex:-

ϵ-closure $(q_0) = \{q_0, q_1, q_2\}$

ϵ-closure $(q_1) = \{q_1, q_2\}$

ϵ-closure $(q_2) = \{q_2\}$

Def:- The set of all states which are at zero distance from the state q is known as ϵ-closure(q).

Note:- ① Every state is at zero distance from it-self.

② ϵ-closure $(\phi) = \phi$

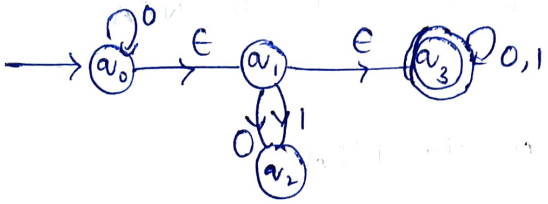
③ ϵ-closure (q) is must countable

③ ϵ-closure $(q_0 \cup q_1 \cup q_2 \cup \dots \cup q_n) = \bigcup_{i=1}^{n} \epsilon\text{-closure}(q_i)$

④ ϵ-closure (q) is a non-empty finite set of Q

⑤ ϵ-closure (q) is always closed set and countable

Ex: Construct ϵ-closure for the states defined in ϵ-NFA



ϵ-closure $(q_0) = \{q_0, q_1, q_3\}$

ϵ-closure $(q_1) = \{q_1, q_3\}$

ϵ-closure $(q_2) = \{q_2\}$

ϵ-closure $(q_3) = \{q_3\}$

ϵ-closure (q_0, q_2) = ϵ-closure $(q_0) \cup$ ϵ-closure (q_2)

$= \{q_0, q_1, q_3\} \cup \{q_2\}$

$= \{q_0, q_1, q_2, q_3\}$

→ Conversion from ϵ-NFA to NFA

Let $M = (Q, \Sigma, \delta, q_0, F) \longrightarrow \epsilon$-NFA

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

① Initial state

No change in initial state

$q_0' = q_0$

② Generation of Transition Function (δ')

$\delta'(q, x) = \epsilon$-closure $(\delta(\{\epsilon\text{-closure}(q)\}, x))$

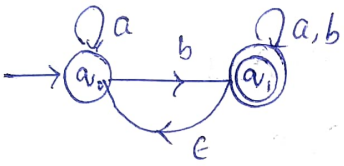
↓ NFA

↓ ϵ-NFA

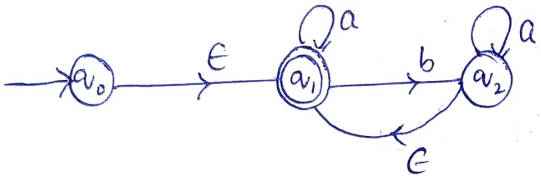
③ Final states:

Every state whose ϵ-closure contain the final state of ϵ-NFA is a final state in NFA

Ex:-1 Convert ϵ-NFA to NFA



Ex:-2



Ex:-1

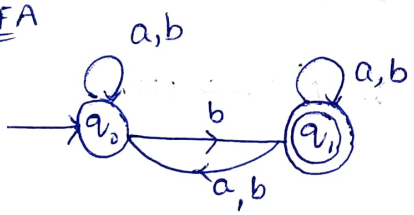
TT for NFA

	a	b
→ q_0	q_0	$\{q_0, q_1\}$
* q_1	$\{q_0, q_1\}$	$\{q_0, q_1\}$

	ϵ^*	a	ϵ^*
q_0	q_0	q_0	q_0
q_1	q_0	q_0	q_0
	q_1	q_1	q_0, q_1

	ϵ^*	b	ϵ^*
q_0	q_0	q_1	q_1
			q_0
q_1	q_0	q_1	q_0, q_1
	q_1	q_1	q_0, q_1

NFA



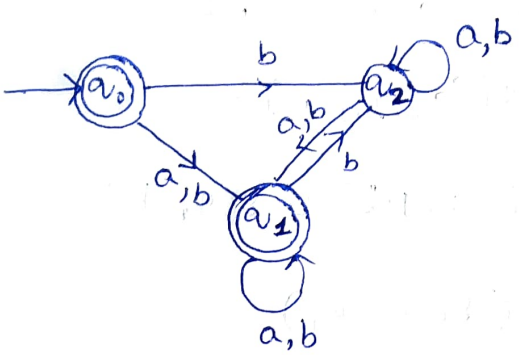
Ex:-2 TT for NFA

	a	b
→ * q_0	q_1	$\{q_1, q_2\}$
q_1	q_1	$\{q_1, q_2\}$
q_2	$\{q_1, q_2\}$	$\{q_1, q_2\}$

	ϵ^*	a	ϵ^*
q_0	q_0	$\emptyset$	$\emptyset$
	q_1	q_1	q_1
q_1	q_1	q_1	q_1
q_2	q_1	q_1	q_1
	q_2	q_2	q_1, q_2

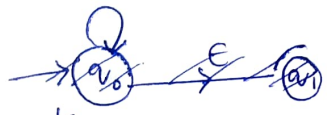
	ϵ^*	b	ϵ^*
q_0	q_0	$\emptyset$	$\emptyset$
	q_1	q_2	q_1, q_2
q_1	q_1	q_2	q_1, q_2
q_2	q_1	q_2	q_1, q_2
	q_2	$\emptyset$	$\emptyset$

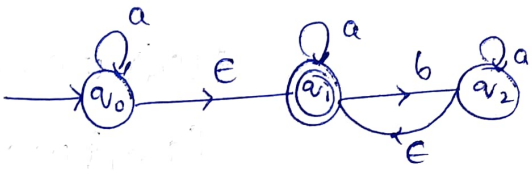
NFA



Ex3:

2nd example





ϵ-closure $(q_0) = \{q_0, q_1\}$

ϵ-closure $(q_1) = \{q_1\}$

ϵ-closure $(q_2) = \{q_2, q_1\}$

NFA

	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_1\}$
$* q_1$	q_1	$\{q_1, q_2\}$
$* q_2$	$\{q_1, q_2\}$	$\{q_1, q_2\}$

$\delta'(q_0, a) = \epsilon\text{-closure}(\delta(\epsilon\text{-closure}(q_0), a))$

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

$= \epsilon\text{-closure}(\delta(q_0, a) \cup \delta(q_1, a))$

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

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

$= \{q_0, q_1\} \cup \{q_1\}$

$= \{q_0, q_1\}$

$\delta(q_0, b) = \epsilon\text{-closure}(\delta\{\epsilon\text{-closure}(q_0), b\})$

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

$= \epsilon\text{-closure}(\delta(q_0, b) \cup \delta(q_1, b))$

$= \epsilon\text{-closure}(\delta\{q_0 \cup q_1\})$

$= \epsilon(q_0) \cup \epsilon(q_1)$

$= \{q_0, q_1\} \cup \{q_1\}$

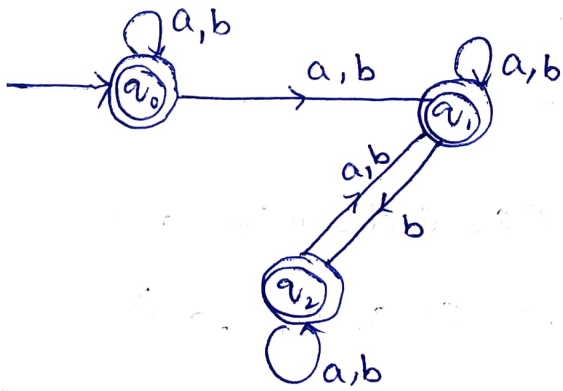
$= \{q_0, q_1\}$

$\delta(q_1, a) = \epsilon\text{-closure}(\delta(q_1, b))$

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

$= q_1$

NFA

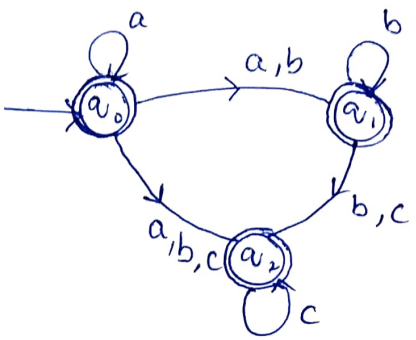


Ex 3:

TT for NFA

	a	b	b	c
* → q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$		$\{q_2\}$
* q_1	$\emptyset$	$\{q_1, q_2\}$		$\{q_2\}$
* q_2	$\emptyset$	$\emptyset$		q_2

NFA



→ Conversion from ϵ-NFA to DFA

procedure

$M = (Q, \Sigma, \delta, q_0, F) \rightarrow \epsilon\text{-NFA}$

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

① Initial state

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

② Construction of δ':

$$\delta'(q, x) = \epsilon\text{-closure}[\delta(q, x)]$$

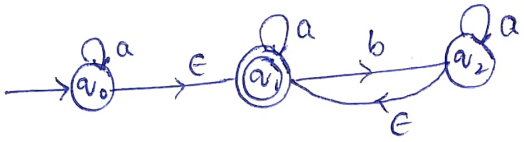
⇒ Start the construction of δ' with the initial state and continue the process for every state that comes under input column and terminate the process whenever no new states appear in the

③ Final state

Every subset which contain the final state of ∈-NFA is a final state in DFA.

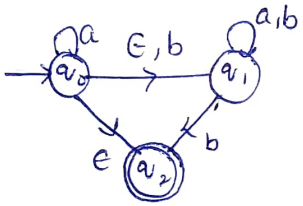
Ex:- ∈-NFA to DFA

①



②

③



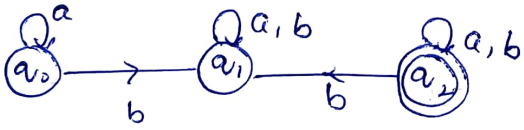
Decision properties of Finite Automata:

1. Emptiness / Non-Emptiness
2. Finiteness / Infiniteness
3. Member ship property
4. Equalness

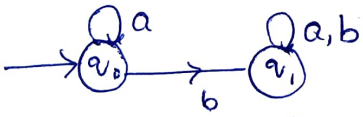
1. Emptyness / Non-Emptyness

1. Select states that cannot be reached from initial state (Unreachable states) and delete them
2. If the resulting machine contains atleast one final state then FA accepts Non-empty language (Non-Emptine
3. If the resulting machine is freefrom final state then it accepts Empty language

Ex:-1



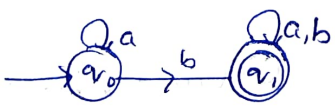
Step 1



∴ FA ~~accepts~~ is Emptyness

Ex:-2

Step 1



∴ FA ~~accepts~~ is NOn-emptinees

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

② Finiteness / Infiniteness

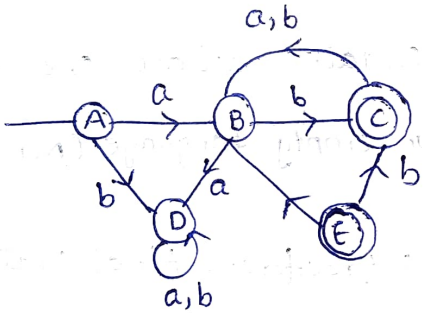
ⓐ Select the states that can't be reachable from Initial state (i.e, unreachable states) and delete them.

ⓑ Select the states from which we can't reach the final state (Dead state) and delete them.

ⓒ If the resulting machine

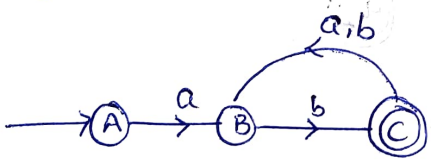
is free from cycles and loops then it accepts Finite language	contain cycles and loops then FA accepts Infinite Language.

Ex:-1



Step 1: E (Un reachable state), D (Dead state)

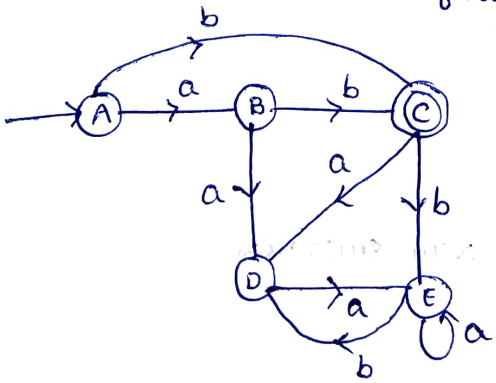
Resulting FA



∴ This FA accepts infinite no. of strings as it have cycle.

∴ Infinite language.

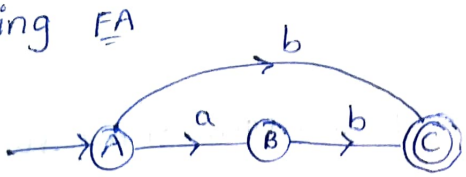
Ex:-2



① Unreachble states (No)

② Dead states (D, E)

Resulting FA



$L(FA) = \{b, ab\}$

∴ FA accepts finiteness

③ Membership property

If L is the language represented by FA ("M") then the string $W \in L(M)$ iff there exists a transition path which starts as intial state and ends in any one of the state.

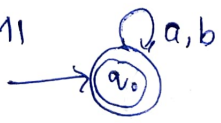
④ Equalness of FA:

Two finite state machines M_1 and M_2 is said to be equal if both of them accepts the some set of strings

$\therefore M_1 = M_2 \iff L(M_1) = L(M_2)$

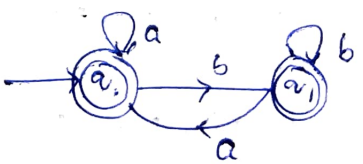
Ex:

M1



$L(M1) = (a+b)^*$

M2:



$L(M_2) = (a+b)^*$

$\therefore L(M_1) = L(M_2) \iff M1 = M2$

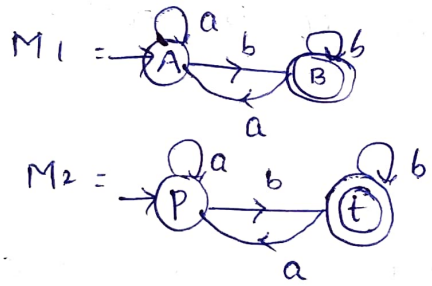
4] ⇒ Note :

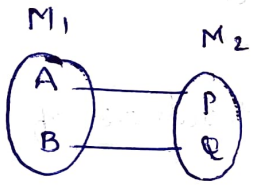
① Equal machines no need to contain same number of states.

② Two machines M1 and M2 are Isomorphic to each other if both machines should have same properties apart from accepting the same language

i.e; 2 machines M1 and M2 are Isomorphic to each other iff one can be obtained from other by replacing the states.

Ex :-





f1 : M1 → M2

f(A) = P

f(B) = Q

∴ M1 ≅ M2

③ Isomorphic machines are equal but equal machines no need to be Isomorphic

→ Comparision Algorithm :-

(To check equalness of FA)

step1 :- Let M1, M2 be two FSM

step2 :- Construct the transition table that contains the pair of states (x, y) where x is from M1 and y is from M2.

③ Start the construction of Transition Table with the pair of Initial states.

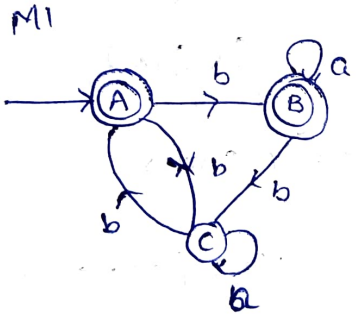
④ If the transition table contain any pair of the form (Final, Non Final) or (Non Final, Final) then stop the constructing table and declare that two machines are not equal.

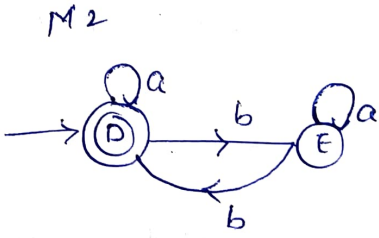
⑤ continue the process of constructing transition table for every new state that appears in the input column of the form (Final, Final) or (Non Final, Non Final) and terminate the process whenever no such new pair of occur under input column.

⑥ If the TT contains all pairs of the form (Final, Final) or (Non Final, Non Final) then two FSM are equal

Ex:-1

①

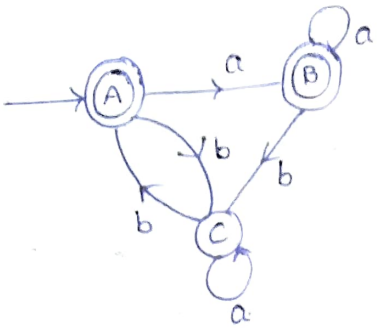




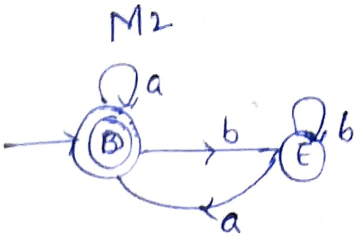
	a	b
→ (A,D)	(B,D) (F,F)	(C,E) (NF,NF)
(B,D)	(B,D)	(C,E)
(C,E)	(C,E)	(A,D) (F,F)

∴ These two are equal

Ex1: M1



M2



	a	b
(A,D)	(B,D)	(C,E)
	(F,F)	(NF,NF)
(B,D)	(B,D)	(C,E)
	(F,F)	(NF,NF)
(C,E)	(C,D)	
	(NF,F) X	

∴ M1 ≠ M2

⇒ Equal States:

Two states P,Q is said to be equal if both $\delta(P,x)$ and $\delta(Q,x)$ goes to Final states or Non-final states

∴ i.e $P = Q \Rightarrow \delta(P,x)$ Non-final states, $\delta(Q,x)$ Final states

→ Equal states also called as Non-distinguishable states.

⇒ Unequal states / Distinguishable states

Two states P, Q are said to be distinguishable if $\delta(P,x)$ goes to Final state and $\delta(Q,x)$ goes to Non-final (vice-versa) for atleast one input string x.

i.e. $P \neq Q$ $\Leftrightarrow$ $\delta(P,x)$ and $\delta(Q,x)$

$\delta(P,x)$: NFS ↑ ↓ FS

$\delta(Q,x)$: FS ↑ ↓ NFS

Note:

① Every state is equal to itself

② Final state can't be equal to Non-final state

③ Equality may exists between 2 final states or between 2 non-final states.

④ If P, Q are equal states then

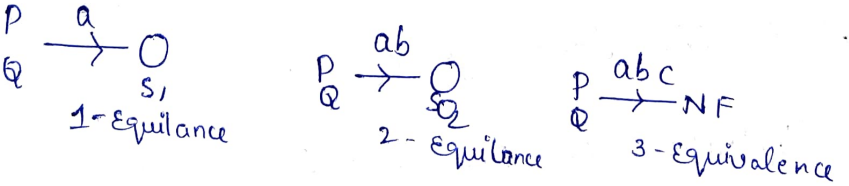
If P is final state then Q is also a final state

(or)

If P is non-final then Q is also non-final state

⇒ k-Equivalence class:-

Two states P, Q is said to be in k-Equivalence class if $\delta(p,x)$ and $\delta(q,x)$ goes to final state or non final states $\forall\ |x| \leq k$



⇒ The 2 states P, Q are equal iff

P, Q ← k-equivalence class (k-a) $\forall\ k \geq 0$

Note: ① Let $Q = \{q_0, q_1, q_2, \ldots q_n\}$ be the set of states in FA

Let P = Equality relation defined over the set Q

i.e $R = \{(q_i, q_j) / q_i = q_j\}$

$q_i R q_j \iff q_i = q_j$

between the Relation R is Equivalence Relation (Reflexive, Symmetric and Transitive)

② The No. of Equivalence sets = Number of states in Minimal FA

⇒ Minimization of Finite Automata

① The process of detection and elimination of states whose presence/Absence will not effect the Language of FA is minimization of FA

FA ⟶ Reduced FA

② Generally the presence/absence of Dead states, Unreachable states and Equal states will not effect the Language of FA

③ Simplification of FA includes elimination of dead states, equal state and unreachable states

④ But if DFA contain dead state, it is mandatory to maintain the dead state in minimal DFA also.

⑤ Since, if we remove the dead state then the system becomes inclomplete (i.e becomes NFA)

⑥ If any unreachable states are there, then remove that states directly.

⑦ There are 2 methods for finding Equal states

ⓐ State Equivalence method

ⓑ Table filling method.

Find the equal states using any one of thes 2 methods and merge them into a singe state.

⇒ Minimal FA :-

The FA after removal of unreachable states and Equal states.

→ For any language L, FA is not unique but minimal FA is unique

ⓐ State Equivalence method

Ex:-

T.T

	a	b
1	6	4
2	5	6
③	4	5
④	3	2
5	2	1
6	1	3

$Q = \{1, 2, 3, 4, 5, 6\}$

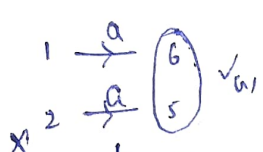
0-Equivalence set

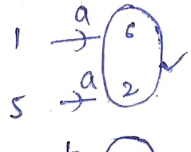
NFF ↑ Final state ↑

$\Pi_0 = \{1, 2, 5, 6\}\ \{3, 4\}$

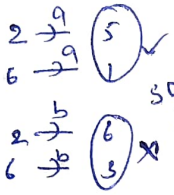
G1 G2

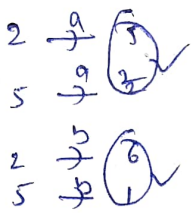
1 - Equivalence set $\pi_1 = \{\{1,6\}\ \{2,5\}\ \{3,4\}\}$
G11 G12 G13

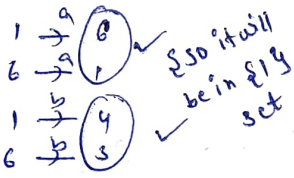




then check for (2,5) if it is in same class then include in {2} element set otherwise make it another set

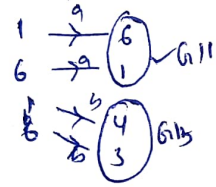


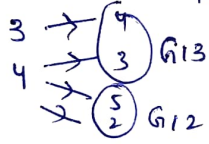




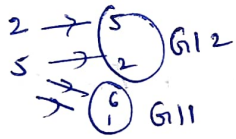
2 - Equivalence set $\pi_2 = \{(1,6)\ (2,5)\ (3,4)\}$
Equivalence

No-change = Equivalence Equivalence





Merge 1 and 6 - Name 1
" 2 and 5 - " 2
" 3 and 4 - " 3



TT

		a	b
→	1	1	3
	2	2	1
*	3	3	2

Ex 2: TT

		0	1
→	a	b	f
	b	g	c
	c	a	c
	d	c	a
	e	h	f
	f	c	g
	g	g	e
	h	g	c

0 - Equivalence set

$\pi_0 = \{\{a,b,d,e,f,g,h\}\ \{c\}\}$
G1 G2

1- Equivalence set $\pi_1 = \{\{a,e,g\}\ \{b,h\}\ \{d,f\}\ \{c\}\}$
G11 G12 G13 G21

a → b, g G11
b → g, c G12

a → b, f G11
b → c G12

a → b → G1 ✗
d → c → G2
a → b → G1 ✓
d → a → G1

b → g ✗
d → c
b → e ✗
d → a

→ So, d

	0	1
a	b	f
b	g	c
d	c	a
e	b	f

G1

	0	1
f	c	g
d	c	a

G2 G1

	0	1
e	h	f
g	g	e

G1 G1

	0	1
h	g	c
b	g	c

G1 G2 ✓

2 - equivalence set $\pi_2 = \{\{a,e\}\ \{g\}\ \{b,h\}\ \{d,f\}\ \{c\}\}$
G111 G112 G113 G13 G2

	0	1	
a	b	f	G12 ✓ G13 ✓
e	h	f	G12 ✓ G13 ✓
g	g	e	G11 G11

	0	1	
b	g	c	✓
h	g	c	✓

	0	1	
d	c	a	✓
f	c	g	✓

3 - Equivalence set $\pi_3 = \{\{a,e\}\ \{g\}\ \{b,h\}\ \{d,f\}\ \{c\}\} = 5$ states
a g b d c

	0	1	
a	b	f	✓
e	h	f	✓

	0	1	
b	g	c	✓
h	g	c	✓

TT =

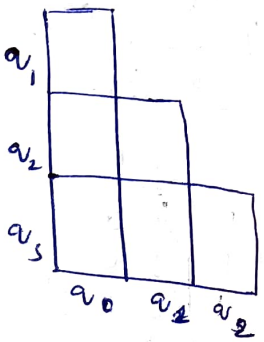
	0	1
→ a	b	d
b	g	c
* c	a	c
d	c	a
g	g	a

Table filling method:- (Myhill Nerode Theorem)

$Q = \{q_0, q_1, q_2, q_3\}$

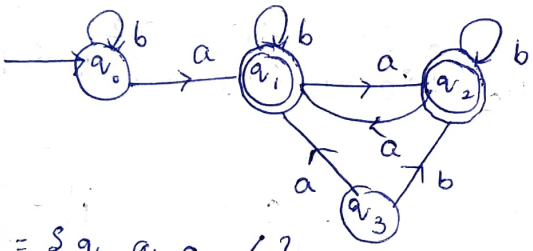
q_0	$q_0=q_0$	$q_0=q_1$	$q_0=q_2$	$q_0=q_3$
q_1	$q_1=q_0$	$q_1=q_1$	$q_1=q_2$	$q_1=q_3$
q_2	$q_2=q_0$	$q_2=q_1$	$q_2=q_2$	$q_2=q_3$
q_3	$q_3=q_0$	$q_3=q_1$	$q_3=q_2$	$q_3=q_3$
	q_0	q_1	q_2	q_3

Check



→ It's enough to check either upper triangular matrix or lower triangular matrix

Ex:-



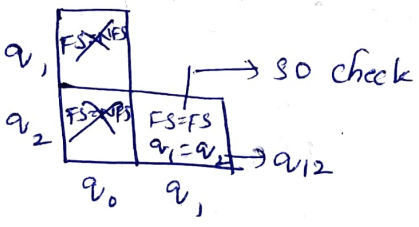
unreachable states - q_3

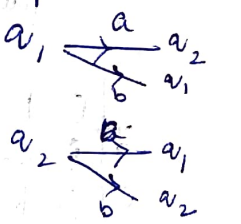
$Q = \{q_0, q_1, q_2\}$

NFS FS FS

	a	b
q_1	q_2	q_1, q_2
q_2	q_1, q_2	q_2

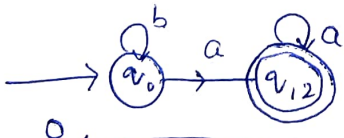
$q_1 = q_2$



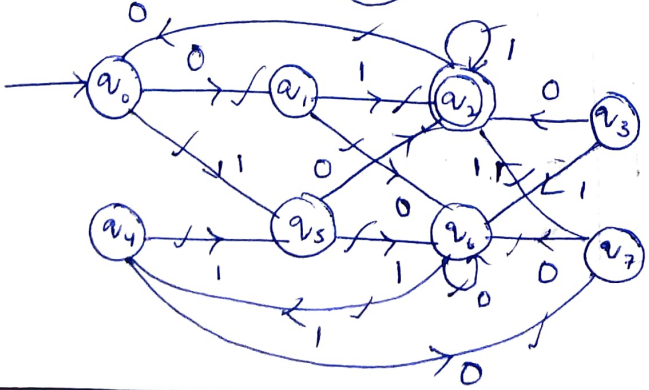


$\therefore$ FA = $\{q_0, q_{12}\}$

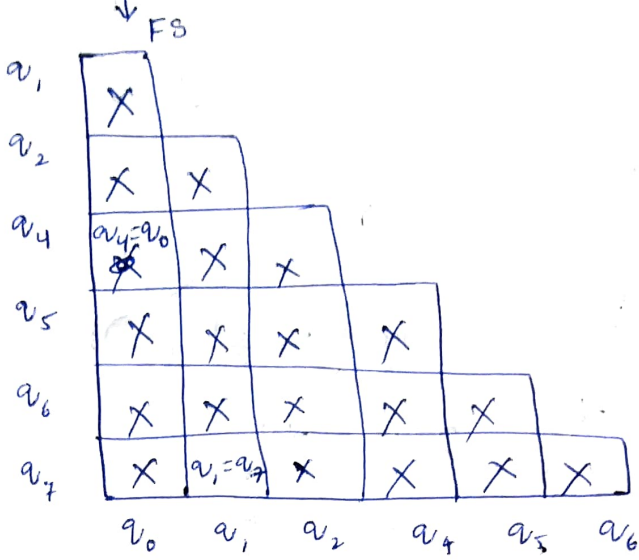
MFA



Ex:-



$Q = \{ q_0\ q_1\ q_2\ \cancel{q_3}\ q_4\ q_5\ q_6\ q_7 \}$



	0	1
q_0	q_1	q_5 → NFA
q_7	q_6	q_2 → FA

$\therefore q_0 \neq q_7$

	0	1
q_1	q_6 ✓	q_2 ✓ $q_1 = q_7$
q_7	q_6 ✓	q_2 ✓

	0	1
q_4	q_7	q_5 → NFA
q_7	q_6	q_2 → FA

$\therefore q_4 \neq q_7$

	0	1
q_6	q_6	q_4 → NFA
q_7	q_6	q_2 → FA

$q_6 \neq q_7$

	0	1
q_5	q_2	q_6 → FA
q_7	q_6	q_2

NFA $q_5 \neq q_7$

	0	1
q_6	q_6 ✓	q_4 ✗
q_0	q_1 ✓	q_5 ✗
q_1	q_6 ✓	q_2 ✗
q_4	q_7 ✓	q_5 ✓
q_5	q_2	q_6 ✗

$q_6 \neq q_1 \rightarrow q_6 \neq q_0$

$q_6 \neq q_7 \Rightarrow$ so, $q_6 \neq q_4$

$q_0 = q_4$

NFA → FA ✓

$q_6 \neq q_2$

	0	1
q_5	q_2	q_6
q_0	q_1	q_5
q_1	q_6	q_2
q_4	q_7	q_5

$q_2 \neq q_1$ so $q_5 \neq q_0$

$q_2 \neq q_6$ so, $q_5 \neq q_1$

$q_2 \neq q_7$ so $(q_5 \neq q_4)$

	0	1
q_4	q_7 ✗	q_5 ✓
q_0	q_1 ✓	q_5 ✓
q_1	q_6 ✗	q_2

$q_7 \neq q_1$ so, $q_4 \neq q_0$

$q_7 \neq q_6$ so, $q_4 \neq q_1$

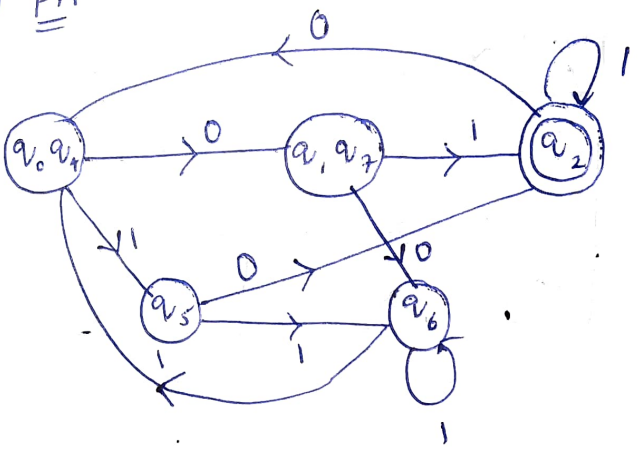
	0	1
q_0	q_1	q_5
q_1	q_6	q_2

so $q_0 \neq q_1$

Final $Q = \{ q_0q_4, q_1q_7, q_2, q_5, q_6 \}$

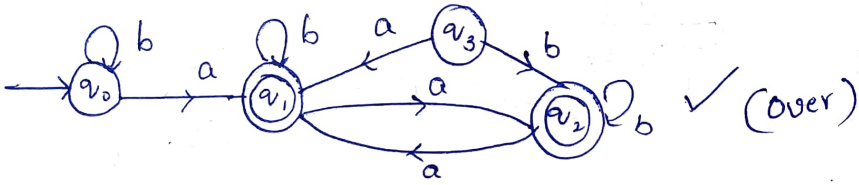
↳ Finally we get 5 states

Minimal FA



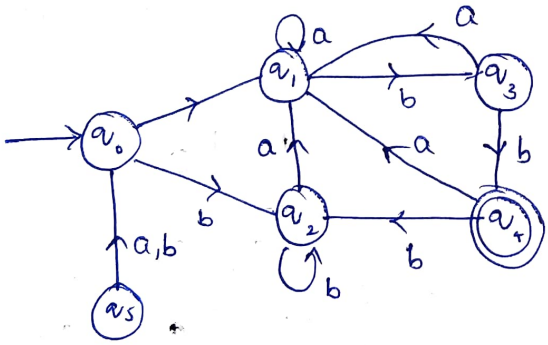
→ Construct Minimal Finite Automata by using SEM/TFM

(1)



✓ (over)

(2)



Finite Automata with Output

FA with output

- Moore Machine
- Melay Machine

→ Both Moore and Melay machines are special case of DFA

→ we need to define transitions for each and every state

→ Both machines acts like output generators rather than Language acceptors.

→ No need of final states $F = \emptyset$

→ No need of Dead states

⇒ Moore Machine Melay Machine

$$M = \{Q, \Sigma, \delta, q_0, \Delta, \lambda\}$$

where

Q is finite set of states

Σ is input alphabet

δ is a transition function

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

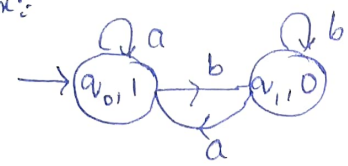
q_0 is Initial state

Δ : Output alphabet (Mostly 0, 1)

λ : Output function

→ state itself

Ex:



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

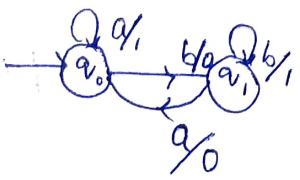
$\lambda : Q \rightarrow \Delta$

$q_0 \rightarrow 1$

$q_1 \rightarrow 0$

→ Transition

Ex:



$\lambda : Q \times \Sigma \rightarrow \Delta$

$\delta(q_0, a) \Rightarrow 1$

$\delta(q_0, b) = 0$

$\delta(q_1, b) = 1$

$\delta(q_1, a) = 0$

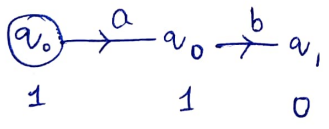
→ Moore Machine is going to associate output with state itself

→ Melay Machine is going to associate output with transition current state with input

Tracing

Moore

⇒ ab



$\lambda(ab) = 110$

⇒ aba = 1101

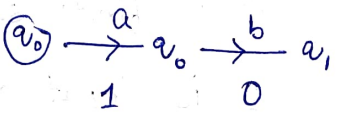
⇒ babb = 10100

⇒ ababba = 1101001

o/p string length - n+1

Melay

⇒ ab



$\lambda(ab) = 10$

⇒ aba = 100

⇒ babb = 0000

⇒ ababba = 100010

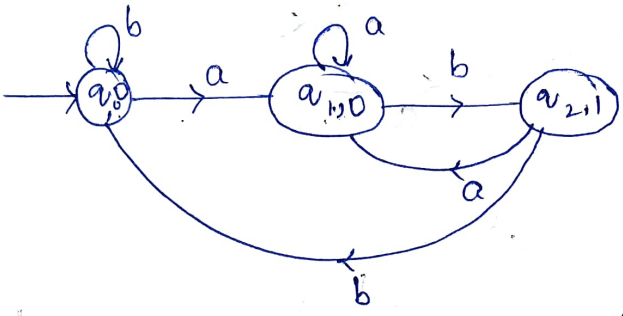
⇒ bab = 000

⇒ aabbab = 110100

⇒ abbabab = 1000100

o/p string length = n

① Construct a Moore Machine that takes set of all strings defined over {a,b} as input and prints '1' as output for every occurence of 'ab' as a string.



$\lambda(aba) = 0010$

$\lambda(baba) = 00010$

$\lambda(abbab) = 001001$

Means string has 2 (ab's)

② Construct a moore machine that takes set of all strings defined over {a,b} & count the occurence of substring "baa"

③ Construct a moore machine that takes set of strings over {0,1} and

produce 'A' as output if input ends with 10

" 'B' " " " " " " 11

Otherwise produce 'c' as output

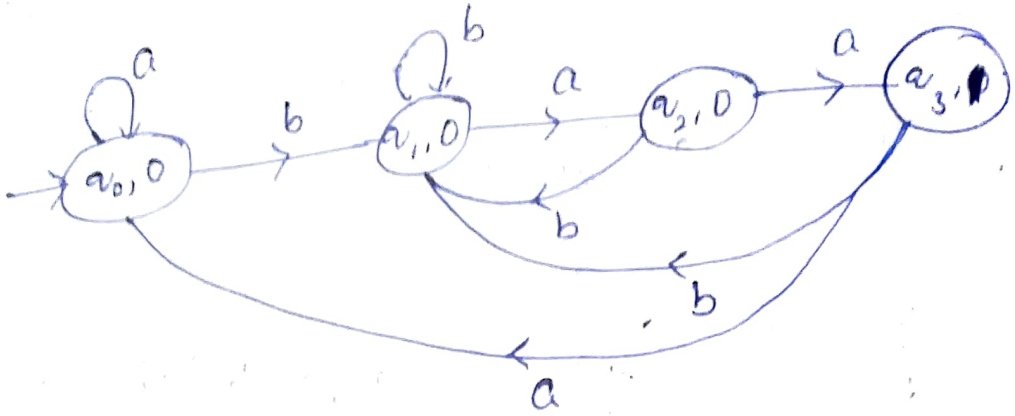
④ Construct Moore machine that takes all strings of 0's and 1's as input and give the 1's compliment of binary number output.

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

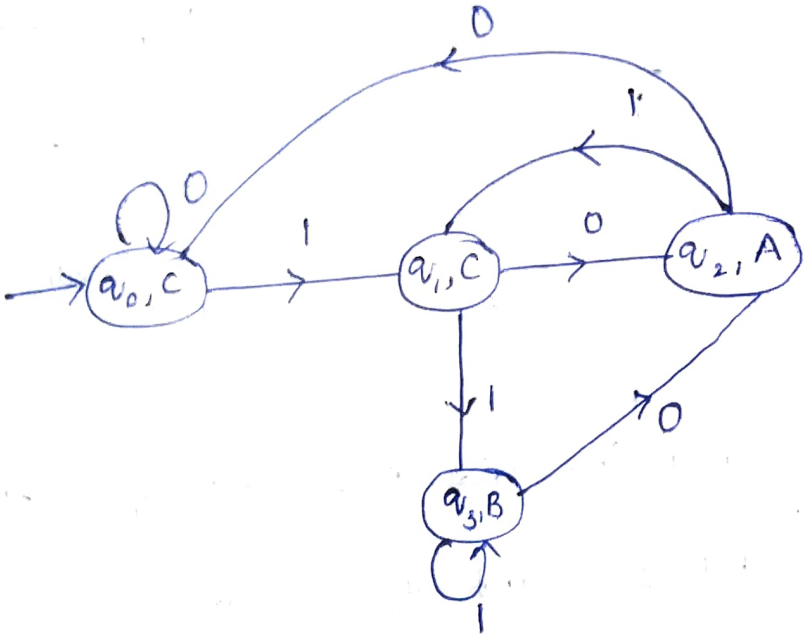
$\Delta = \{0,1\}$

$101 \rightarrow 010$

(2) A)



(3) A)

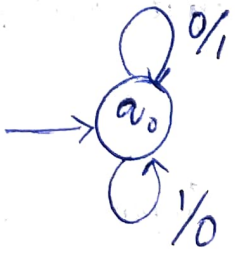


10 - A

11 - B

X - C

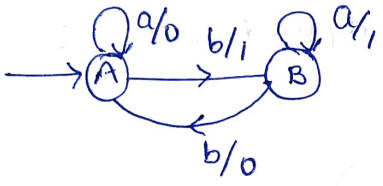
(4) A)



Melay Machine

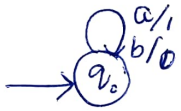
$M = (Q, \Sigma, \delta, \Delta, q_0, \lambda)$ ← O/P

O/P is with transition and the state



δ	a	Δ	b	Δ
→ A	A	0	B	1
B	B	1	A	0

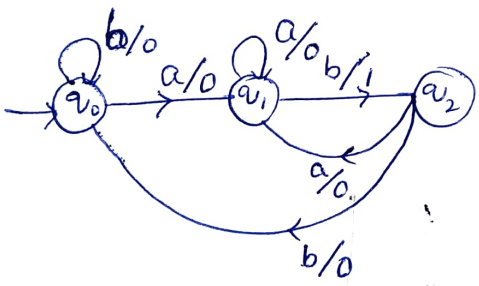
① Construct a Melay Machine which accepts strings of a's and b's and count no of a's in the i/p string



a-1, b-0

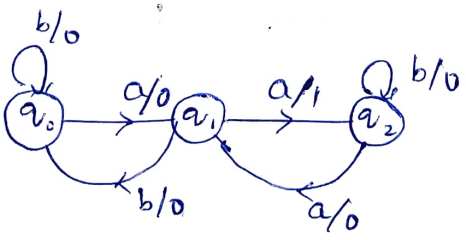
② Construct Melay Machine on $\Sigma = \{a, b\}$ and count number of occurences of substring ab

abab
1 1

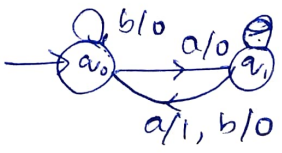


a b a a
0 1 0 0

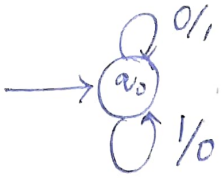
③ Construct Melay machine defined over $\Sigma = \{a, b\}$ and count no of occurence of 2a's consecution



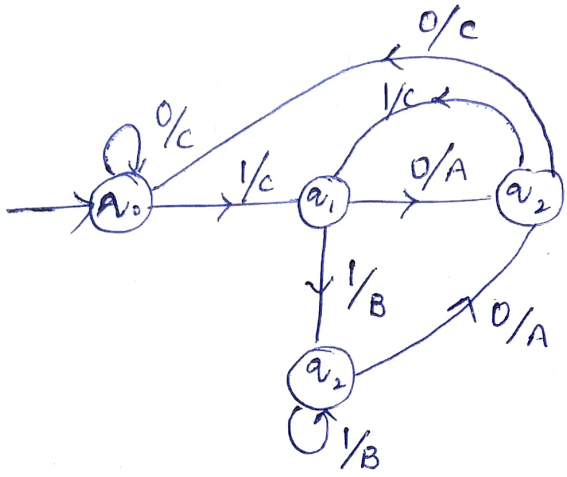
aa
1



④ Construct Melay Machine for 1's compliment



⑤ Construct Melay Machine defined over $\Sigma = \{0, 1\}$



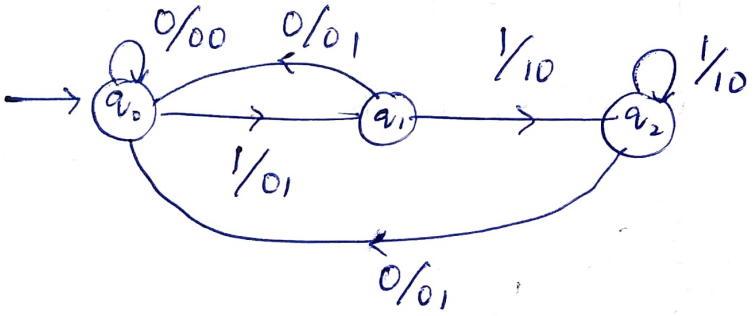
Output

X 10 - A

X 11 - B

Otherwise - C

⑥ What is the output produced by the given



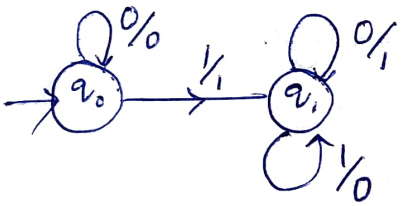
δ	0	Δ	1	Δ
→ q_0	q_0	00	q_1	01
q_1	q_0	01	q_2	10
q_2	q_0	01	q_2	10

(7) Construct Melay Machine defined over $\Sigma = \{0,1\}$ takes binary number as input and produce residue modulo 4 as o/p $\Sigma = \{0,1\}$ $\Delta = \{0,1,2,3\}$

(8) Construct Melay Machine that takes binary number of input and produce 2's compliment of that number as o/p

Hint:- Assume the string is read from LSB to MSB

(A)



(Input reads from LSB to MSB)

Ex:- 00100
↑↑↑↑↑
5 4 3 2 1

0 0 1 0 0

o/p - 0 0 1 1 1 (Reverse the string)

11100