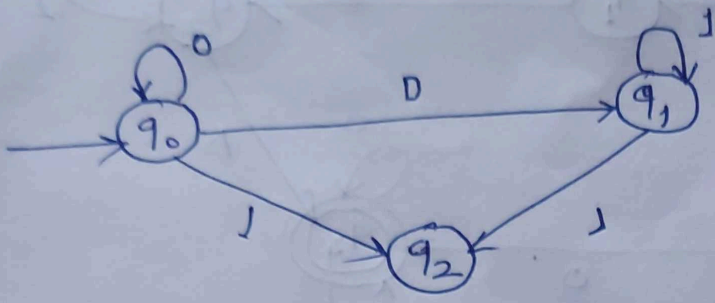
★ Nondeterministic FA (NFA)



$$NDFA = \{ Q, \Sigma, q_0, A, \delta \}$$

Q = set of state (Finite states)

Σ = set of Alphebet

q_0 = start / Initial state

$\forall \quad q_0 \in Q$

F/A = Accepting / final state

δ = Transaction Function

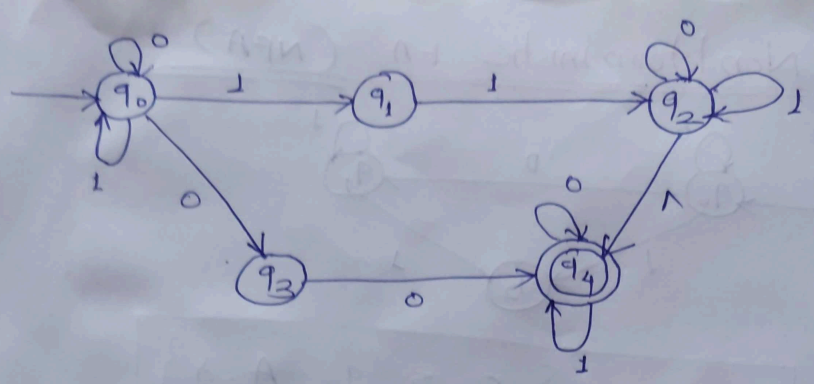
$\delta : 2^{Q \times \Sigma} \rightarrow Q$

eg i> $\delta(q_0, 0) = q_0$/

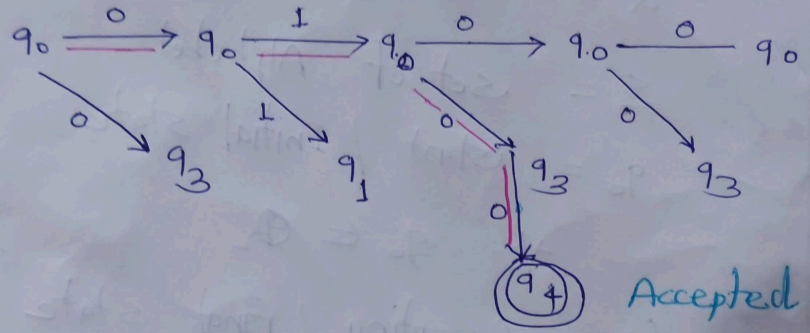
$\delta(q_0, 0) = q_1$

ii> $\delta(q_1, 1) = q_1$

$\delta(q_1, 1) = q_2$



(Ex) Processing of 0100



Containing 9 transactions in NFA due to this initially we can lose the time & space Complexity may reduce to

$9/4 = 225\%$

More time & space Complexity

☆ Computational Tree for NFA

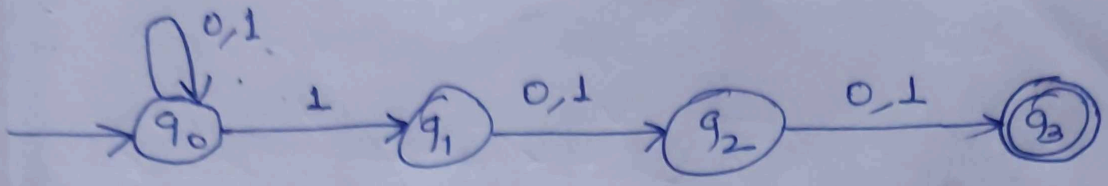
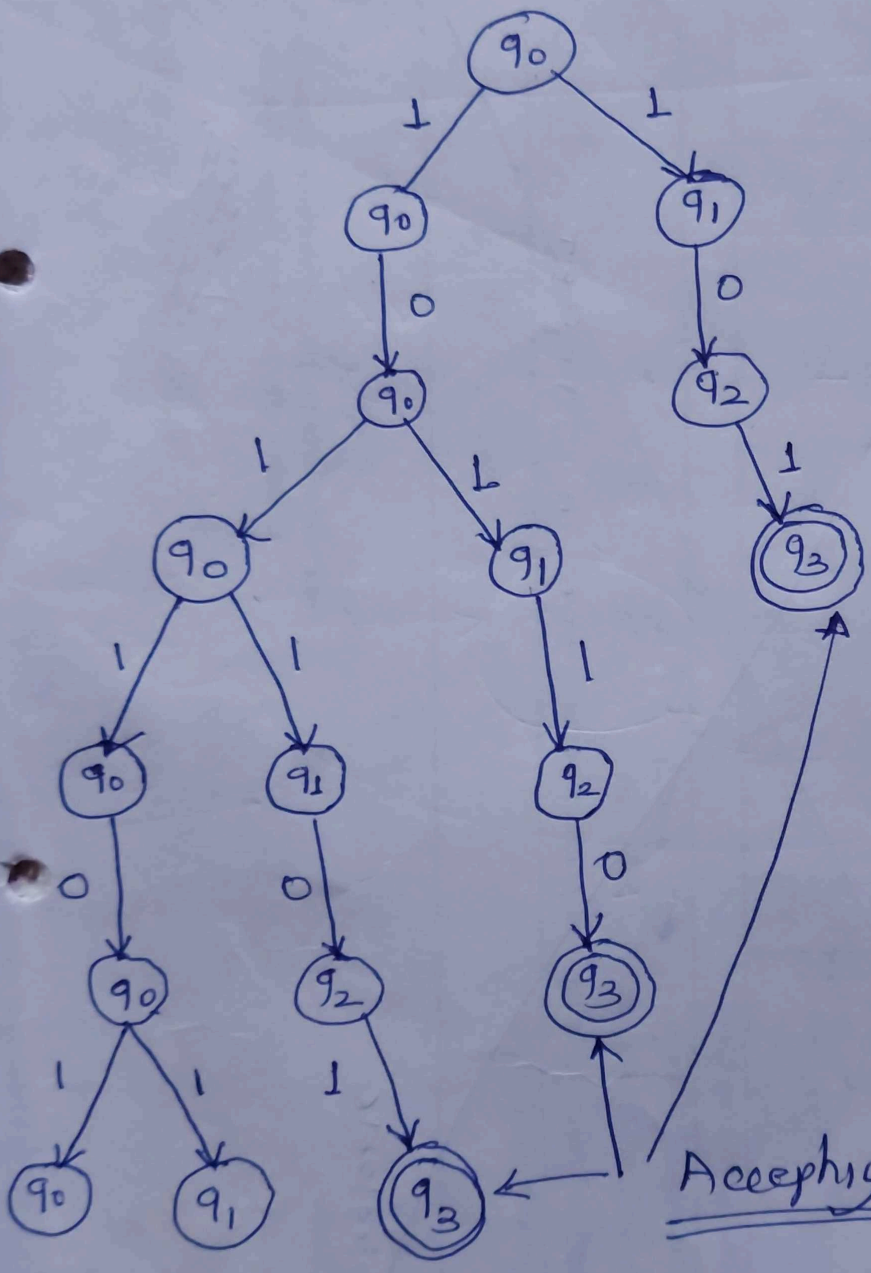


Fig :- NFA



Accepting states

chapter II

** NFA with Null Transitions

Defn - A nondeterministic finite automan with $\wedge$ transitions [abbreviated NFA-$\wedge$) is 5 tuple ($Q, \Sigma, q_0, A, \delta$) where Q & Σ are finite sets, $q_0 \in Q$, $A \subseteq Q$. &

$$\delta : Q \times (\Sigma \cup \{\wedge\}) \longrightarrow 2^Q$$

1) It contain emply inputs & they called as null transitions.

2) If the finite automata $M = \{Q, \Sigma, q_0, A, \delta\}$ is automata with $\wedge$ null transitions then automata without M' null transitions will be $M' = \{Q, \Sigma, q_0, A', \delta\}$

eg :-

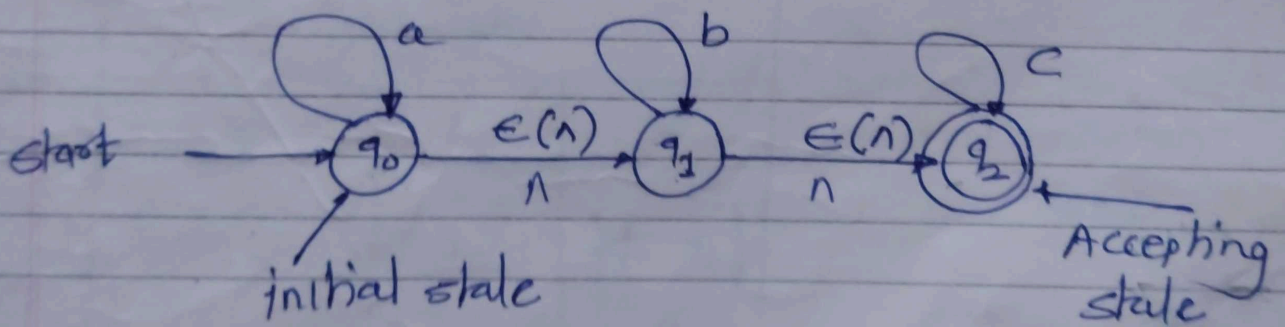


Fig:- A Automata with $\wedge$ (null) Transitions

3) In above figure it contain 4 inputs a, b, c & Λ so we construct transition table

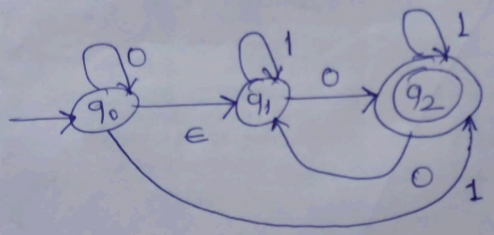
4) Transition Table

States	a	b	c	Λ
→ q_0	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_2\}$	ϕ	$\{q_2\}$
◎ q_2	ϕ	ϕ	$\{q_2\}$	ϕ

* NFA with ∈/Λ Epsilon Transaction

- It allows FA to jump from one state to another state without consuming any input symbol.
- Add one column for ∈ moves
- Easier for NFA Construction
- Epsilon Transactions are those can be reached from state q by repeatedly making ∈ transactions including itself.

eg



Transaction Diagram

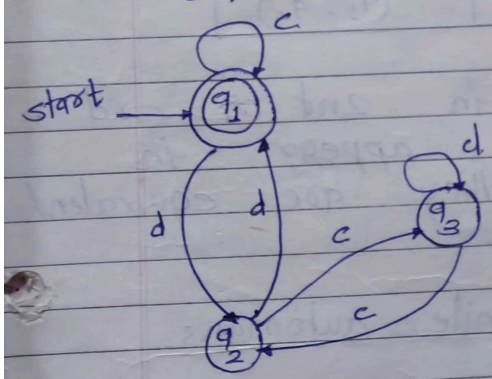
state	Alphebets		
	0	1	∈
q_0	q_0	q_2	$\{q_0, q_1\}$
q_1	q_2	q_1	ϕ
q_2	q_1	q_2	ϕ

Transaction Table

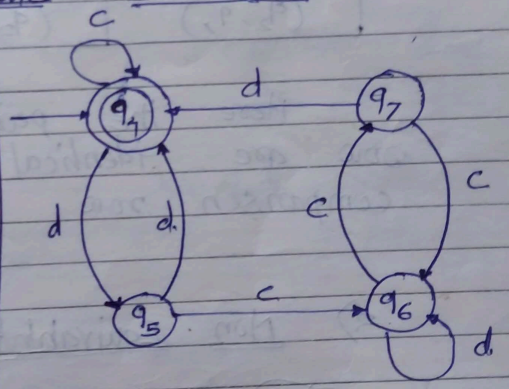
** Equivalence of FA's

- Two FA's over Σ are equivalent if they accept the same set of string over Σ
- They are called to be non equal if they contain one transaction at a transaction reaches at final position & other does not.
- The method called as comparision method

Eg 1> Equivanlent State Automan



Automan M_1



Automan M_2

In Above two automatans we have q_1 & q_4 as a accepting and initial state. & They contain the transactions c & d at each level

- Now in M_1 & M_2 we know that $\Sigma = \{c, d\}$.

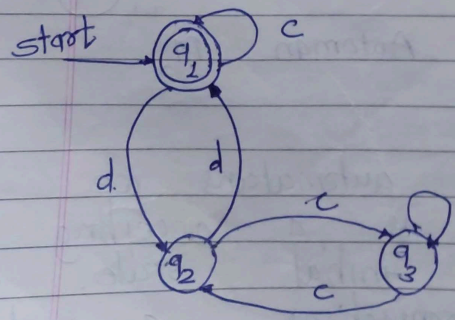
- M_1 having start state q_1 & M_2 having q_4

- Now we comparision table

(q, q')	$q_c^1 - q_c^2$	$q_d^1 \cdot q_d^2$
$(q_1 q_4)$	$(q_1 - q_4)$	$(q_2 - q_5)$
$(q_2 - q_5)$	$(q_3 - q_6)$	$(q_1 \cdot q_4)$
$(q_3 \cdot q_6)$	$(q_2 - q_7)$	$(q_3 \cdot q_6)$
$(q_2 - q_7)$	(q_3, q_6)	$(q_1 \cdot q_4)$

Here the pairs in 2nd & 3rd row are identical or appear in comparison row so they are equivalent.

2) Non Equivalent finite automans



1) Automan M_1

Chapter - 2

Kleenes Theorem

Part I Moore (Machine)

- Developed by Moore in 1955 by scientist moore.
- Moore machine is 6 tuple $(Q, \Sigma, q_0, \delta, \Delta, \lambda)$

where

(Q, Σ, q_0, δ) symbols from finite automata

Δ output alphebet / Tape symbol

λ mapping from Q to Δ. ie contain all outputs

- If Moore machine contain the symbols from $q_1, q_2 \ldots q_n \; \forall \; n \geq 0$ then

$\lambda(q_1), \lambda(q_2), \ldots \lambda(q_n)$ will contain all the strings.

Eg:- Consider moore machine

$M = (Q, \Sigma, \Delta, \delta, q_0, \lambda)$

where

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

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

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

Now from given data we can construct the Moore machine as follow.

State	δ		Output λ
	0	1	
q_0	q_3	q_1	a
q_1	q_1	q_2	b
q_2	q_2	q_3	a
q_3	q_3	q_0	a

fig:- A Moore Machine

Now we consider input string 0111
then transitions will be

$q_0 \xrightarrow{0} q_3 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2$

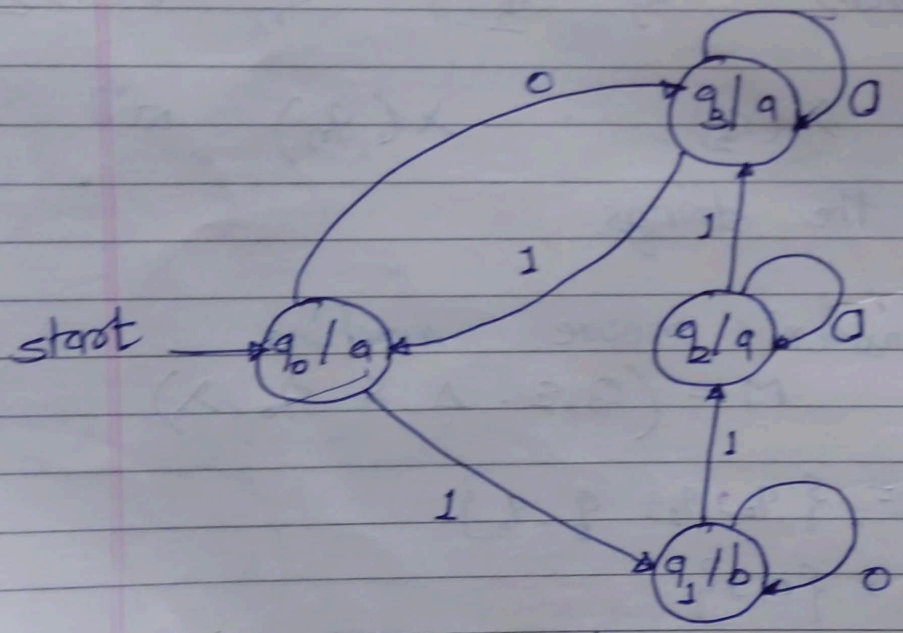


fig:- Transition diagram

from Transition diagram the output string will be

aaaba.

✱ Kleene's Theorem

Any regular language can be accepted by finite automata

Proof

If a language accepted by finite automata then it will accepted by also. Nondeterministic finite automata with null transitions

It will contain the operations like union, concatenation & Kleene *

If L_1 & L_2 are two languages then by induction hypothesis we get $L_1 \cup L_2$, $L_1 L_2$ & L_1^* as operations

Now consider if L_1 & L_2 are recognized NFA-Λ with M_1 & M_2 then

$$M_i = (Q_i, \Sigma, q_i, A_i, \delta_i)$$

Union :-

$$M_u = (Q_u, \Sigma, q_u, A_u, \delta_u)$$

$$\forall\ Q_u = Q_1 \cup Q_2 \cup \{q_u\}$$

$$A_u = A_1 \cup A_2$$

Here q_u is accepted by either Q_1 or Q_2

$A_u = A_1 \cup A_2$

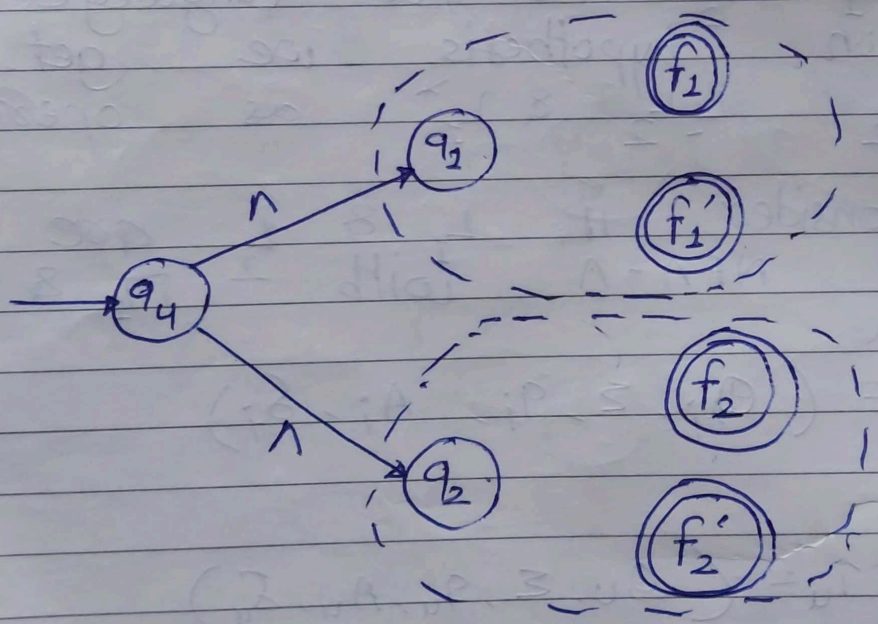
NFA- $\wedge$ for three basic regular languages

① Union

② Concatnenton

③ Kleene

① Union



② Concatenation

$M_c = (Q_c \cdot \Sigma \cdot q_c \cdot A_c \cdot \delta_c)$

here No need of new state init.

$q_c = q_1$ initial state

$A_2 = A_c$ Accepting state

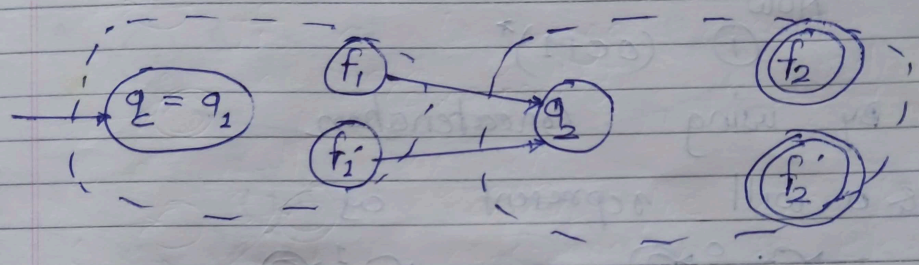


Fig:- Concatenation

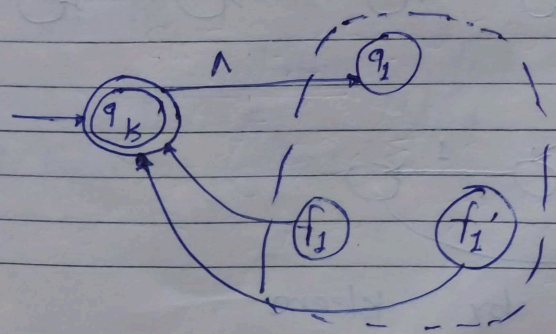
③ Kleene *

$M_K = (Q_K, \Sigma, q_K, A_K, \delta_K)$

q_K - new state not in Q_1

$Q_K = Q_1 \cup \{q_K\}$

$A_K = \{q_K\}$



Eg:- Construct the NFA-Λ for

1) $(00+1)^* (10)^*$

2) $(00+1)^*$

1) $(00+1)^* (10)^*$

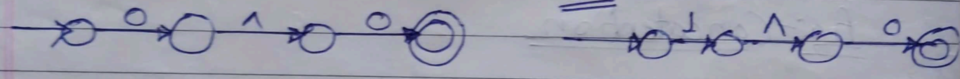
Now

① $(00+1)^*$

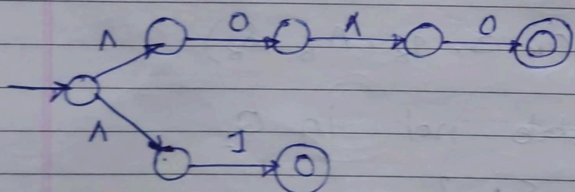
By using concatenation

1 & 0 will represent as

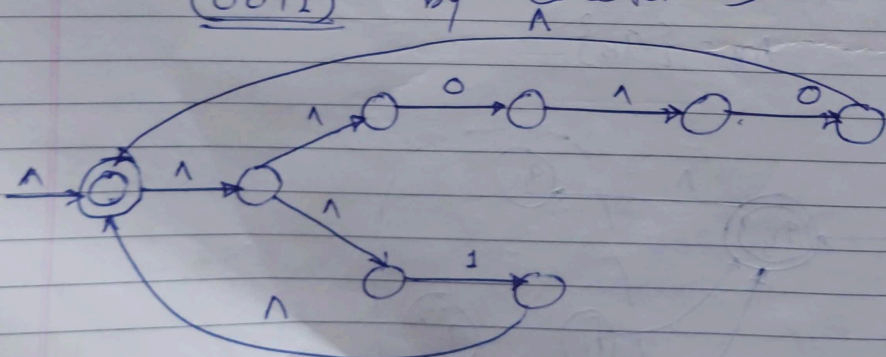
0 1



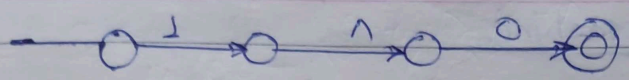
00 10



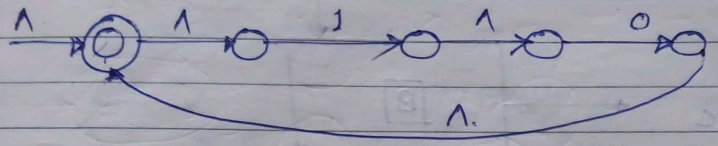
(00+1) by concatenation



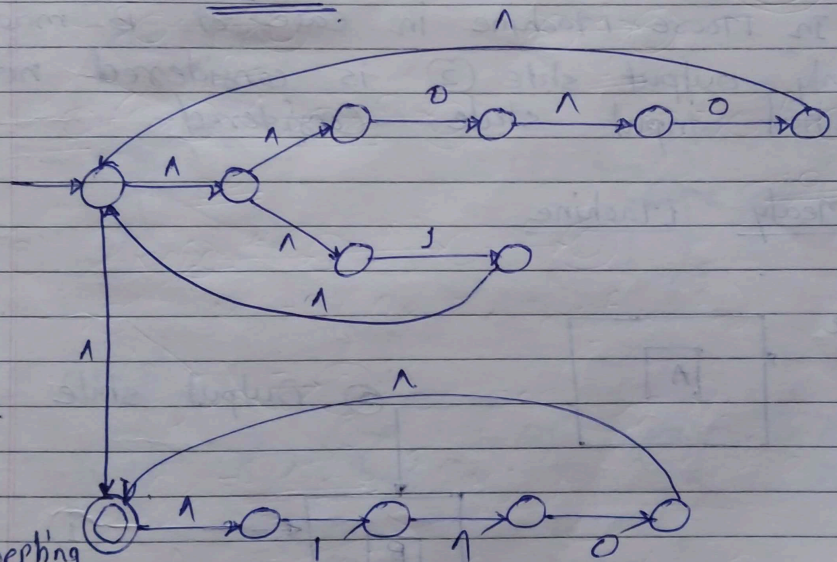
$(00+1)^*$ by kleene



10



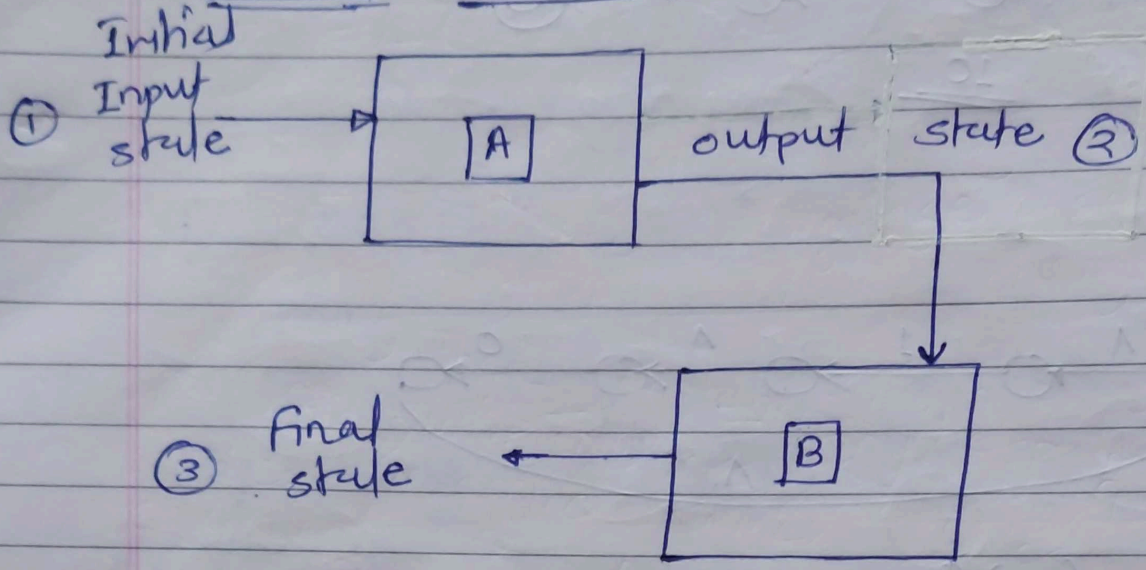
(10)*



Accepting or kleenee operation

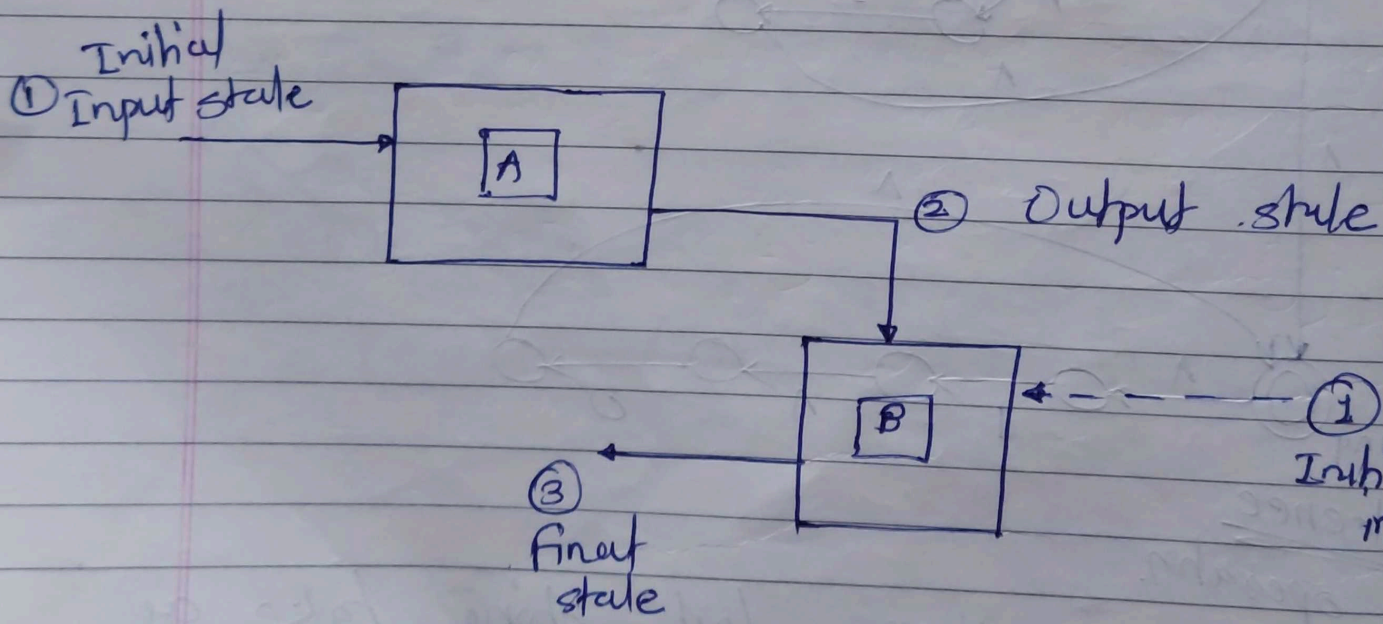
Here last string take as kleene string & denote it by * operation

Moore Machine



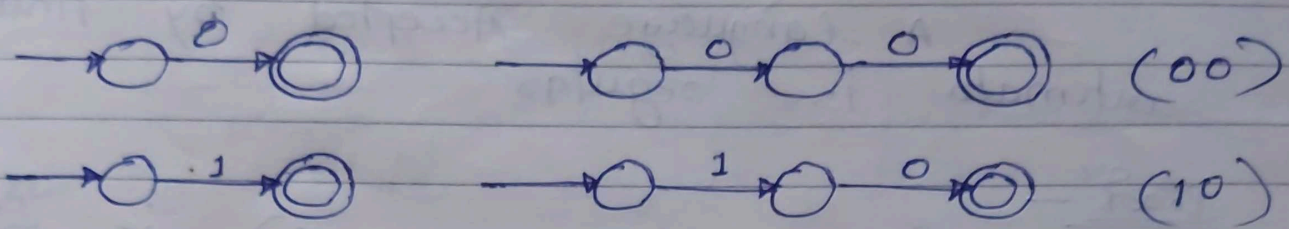
In Moore Machine in case of B machine only output state ② is considered not initial input state considered

Mealy Machine



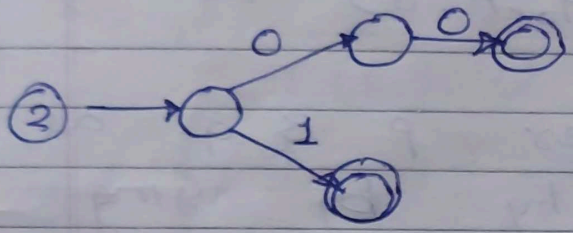
In case of Mealy M/c state B considered both initial input state & previous output state

A simplified NFA-Λ for $[00+1]^*[10]^*$

① 

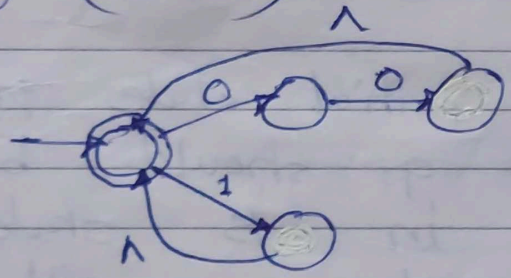
(00)

(10)

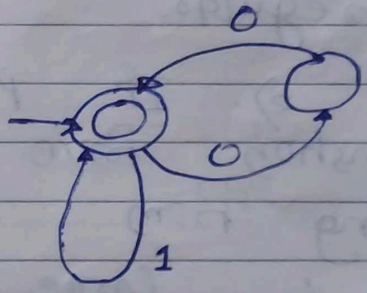
② 

(00+1)

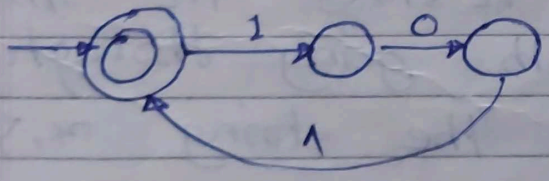
③ $(00+1)^*$



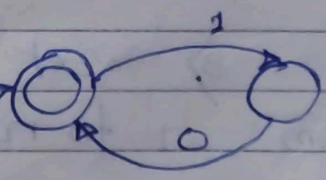
OR

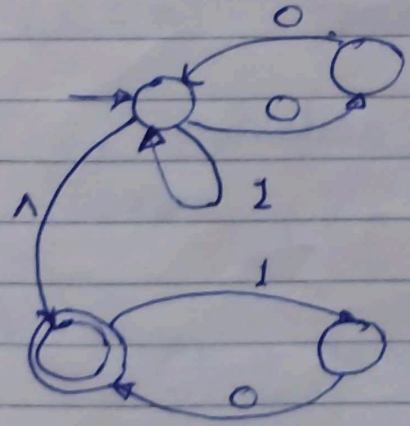


④ $(10)^*$



OR



⑤ 

★ Kleene's Theorem Part 2

A language Accepted by finite automata is regular

Proof —

If $M = (Q, \Sigma, q_0, A, \delta)$ is finite automata

1) If we consider p & q as two sets then by by using mathematical induction we can take union of it. which is regular.

2) If p is is initial state then transition from p to q should be going from p to q in ts state, if k is large enough then L will be L(P,q)

3) Let Consider n states in the from 1 to n & path going through s.

Now Consider the string x, y, z

& $x = yz$

$\delta^*(p, y) = s$

$\delta^*(s, z) = q$

Now

$p \xrightarrow{a} q \xrightarrow{b} r$

- If Language $L(P,q)$ for n states then

$$\boxed{L(P,q,n) = L(P,q)}$$

where n is not higher ie no of states higher than n

- if $L(P,q,n)$ is regular then $L(P,q,j)$ is also regular if.

$$\boxed{0 \leq j \leq n}$$

- If $j \leq n$ then

$$\boxed{L(P,q,n) = L(P,q,j)}$$

- while proving basis step ie $L(P,q,0)$ is regular.

ie the state does not contain more than 0 state ie it contain only one symbol. ie finite so $L(P,q,0)$ is regular.

- By induction hypothesis if $L(P,q,k)$ is regular then $L(P,q,k+1)$ is also regular.

$L(P, q, k+1)$

The path from p to q does contain path higher than $(k+1)$ state.

Let consider ywz. be the states in such a way

$y \Rightarrow$ from state p to $k+1$ to q

$w \Rightarrow$ from $(k+1)$ to q

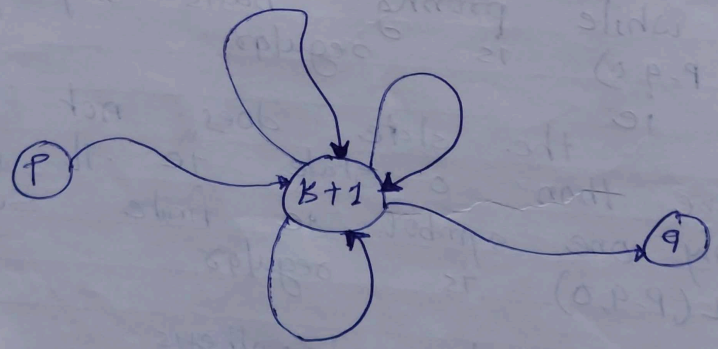
$z \Rightarrow$ from $(k+1)$ to back itself

ie

$y \in L(P, k+1, k)$

$w \in L(k+1, q, k)$

$z \in L(k+1, k+1, k)^*$



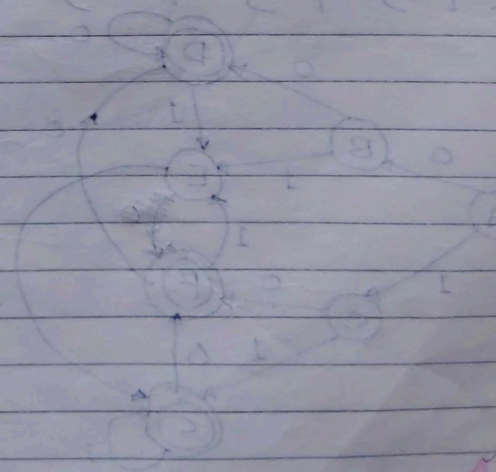
it follows either of following two cases

$$x \in L(P,q,k) \cup L(P,k+1,k)\, L(k+1,q,k)\, L(k+1,k+1,k)^* \quad \text{—①}$$

on the other hand by using ①

$$x \in L(P,q,k+1) = L(P,q,k) \cup L(P,k+1,k)\, L(k+1,q,k)\, L(k+1,k+1,k)^*$$

from the above formula we have proved it is regular because it satisfy the union, concatnehon & kleene* operation



* Minimum state FA for Regular Language

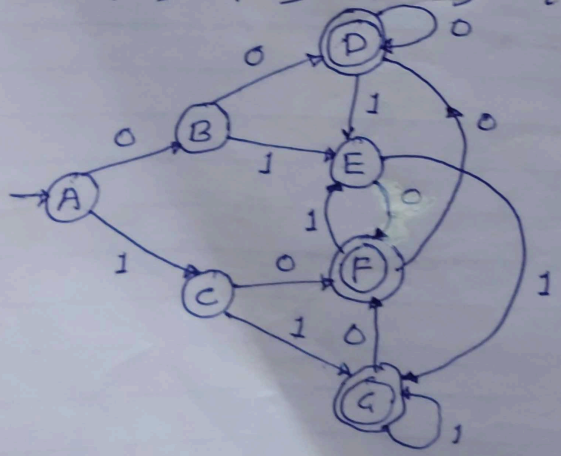
- Removing / minimizing the original states in the language so original meaning should not change.

Algorithm :-

1) Eliminate any state that can't be reached from initial state.

2) Then partation the remaining states into blocks so that all states in the same blocks are equivalent & no pair of states from different blocks are equivalent.

Eg:- From given table & diagram minimize FA. The partation of states into equivalent blocks is
({A}, {B}, {C, E}, {D, F}, {G})



on the given transaction diagram by applying Table filling Algorithm we get. following

X distinguishable states.

blank squares are equivalent.

	A	B	C	D	E	F
B	X					
C	X	X				
D	X	X	X			
E	X	X		X		
F	X	X	X		X	
G	X	X	X	X	X	X

fig:- Table of state Equivalence.

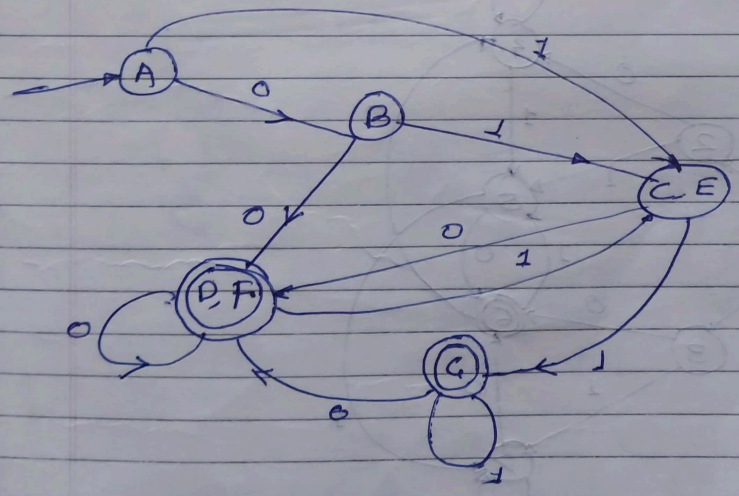


fig: minimus state in FA

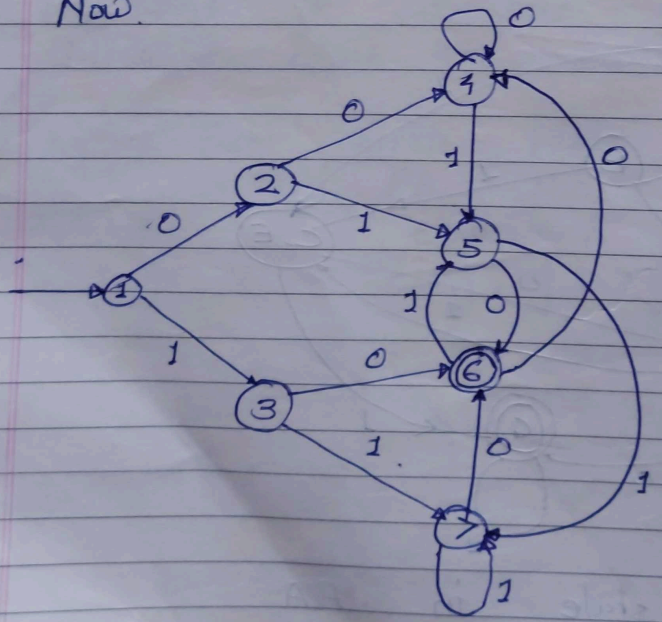
* Minimizing no of states in FA

Let $M = (Q, \Sigma, q_0, A, \delta)$ be the finite automata then while minimizing no of states.

1) Consider a single state for equal states

2) Remove unreachable states from initial state without affecting meaning of original state.

Consider A finite automata with $M = (Q, \Sigma, q_0, A, \delta)$ with 7 state Now.



(a) Original finite Automata [Transition Diagram]

state		input 0	1
A	1	0	1
0	2	00	01
1	3	10	11
00	4	00	01
01	5	10	11
10	(6)	00	01 ← Accepting state
11	7	10	11

Fig:- Transition Table

In Given table state 6 is accepting state so considered as (C)

A ⇒ state 1, 2 or 4

B ⇒ state 3, 5 or 7

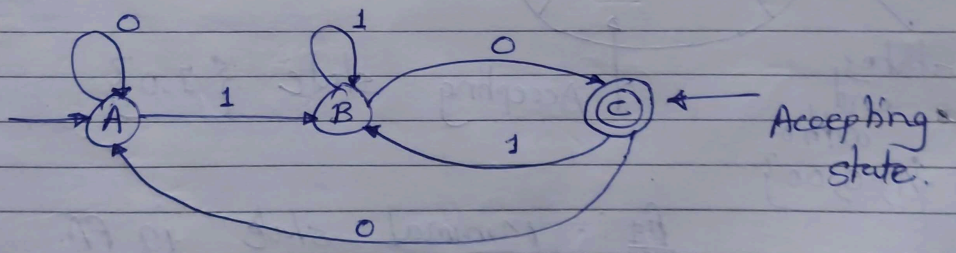


Fig:- Minimum state FA.

In following diagram it shows partation with original FA ie.

$L_1, L_2, L_3, L_4, L_5, L_6, L_7$

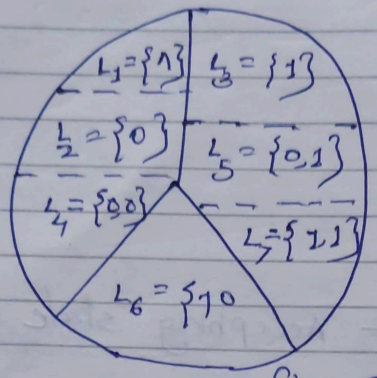
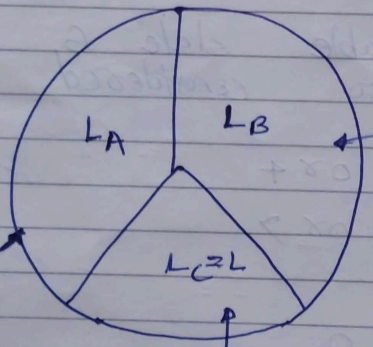


Fig:- Original finite Automata



state end with 1
{1, 01, 11}

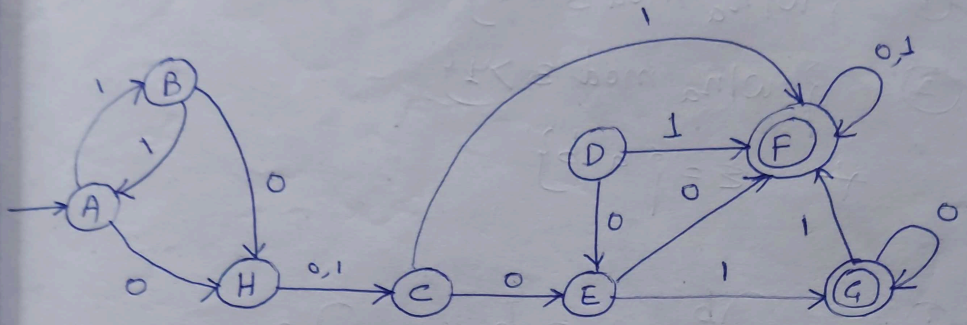
states end with 0
{1, 0, 00}

Accepting state {1, 0}

Fig :- Minimal state in FA.

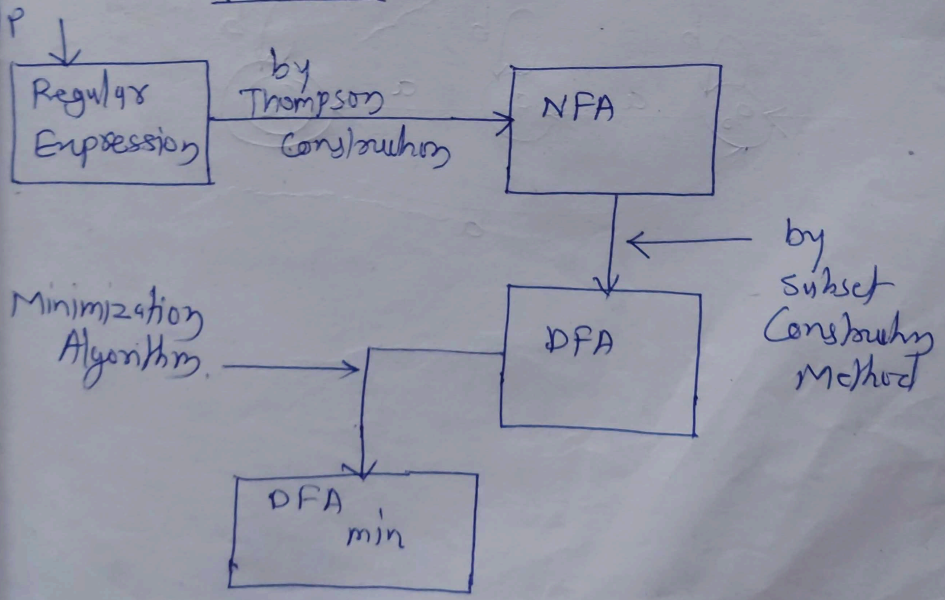
Hence it is minimal FA of given FA.

DFA Minimization



- Text Processing - larger data set
- Compiler
- Smaller DFA require less space & time
- Save resources

Compiler Conversion from Regular Expression to DFA min



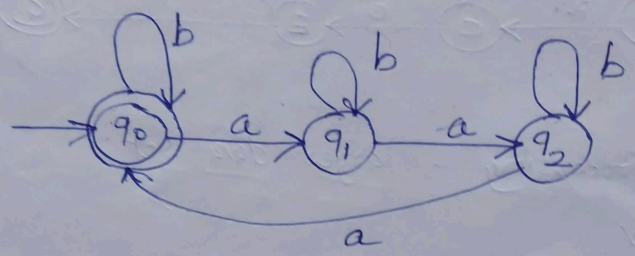
* Draw DFA for following

① $\{w | n_a \bmod 3 = 0\}$

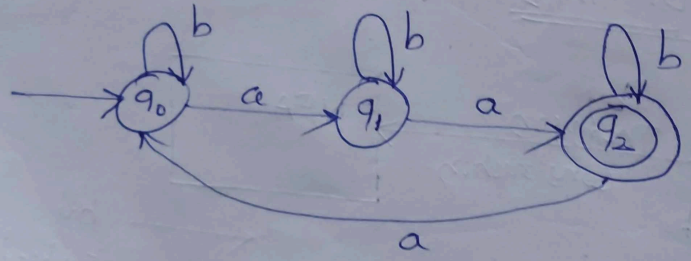
② $\{w | n_a \bmod 3 > 1\}$

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

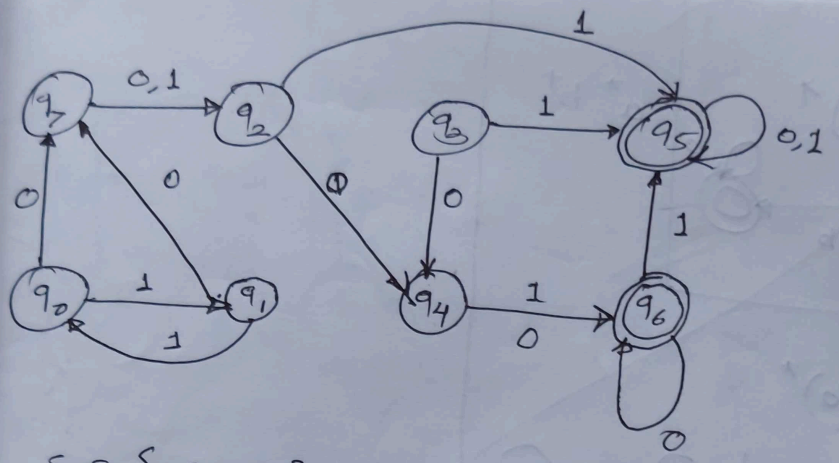
①



②



★ Reduce the DFA



$S_1 = \{ q_5, q_6 \}$

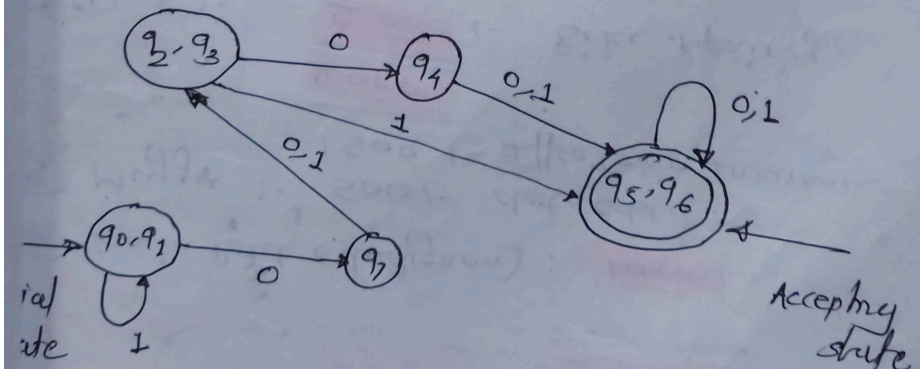
$S_2 = \{ q_0, q_1, q_2, q_3, q_4, q_7 \}$

While Taking Transitions we get

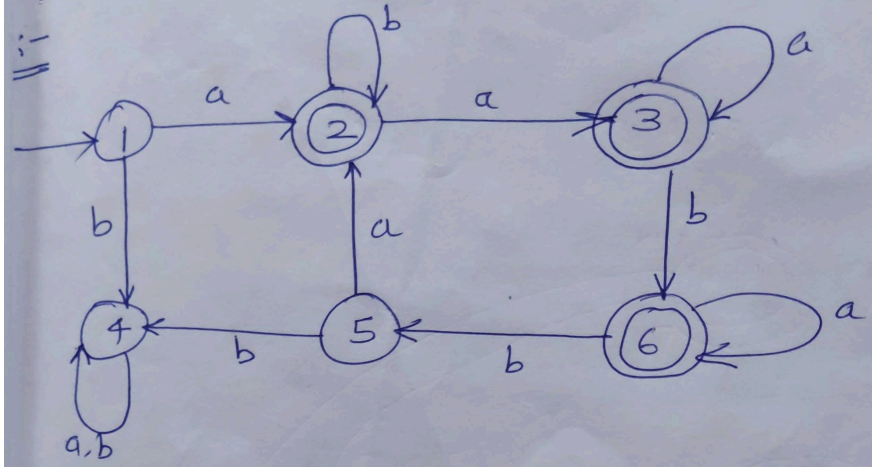
$S_1 = \{q_5, q_6\}$, $S_2 = \{ q_0, q_1 \}$, $S_3 = \{q_2, q_3\}$, $S_5 = \{q_4\}$

$S_5 = \{q_7\}$

Minimize DFA states in given DFA



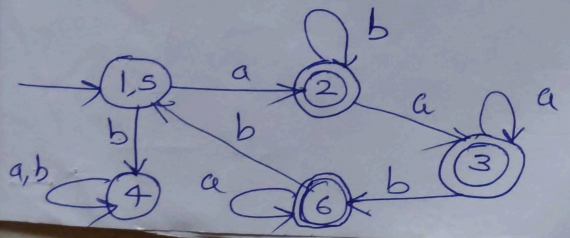
* Reduce the DFA :-



Transaction Diagram

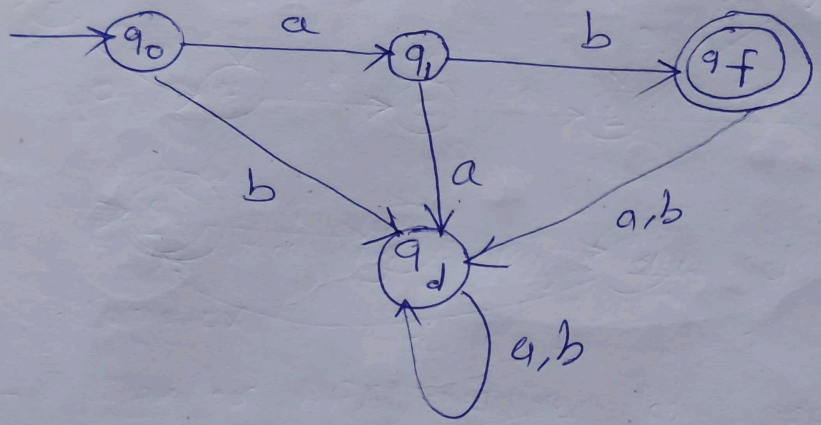
Now from Above transaction table will be

State	Alphebets / Input symbol	
	a	b
1	2	4
2	3	2
3	3	6
4	4	4
5	2	4
6	6	5

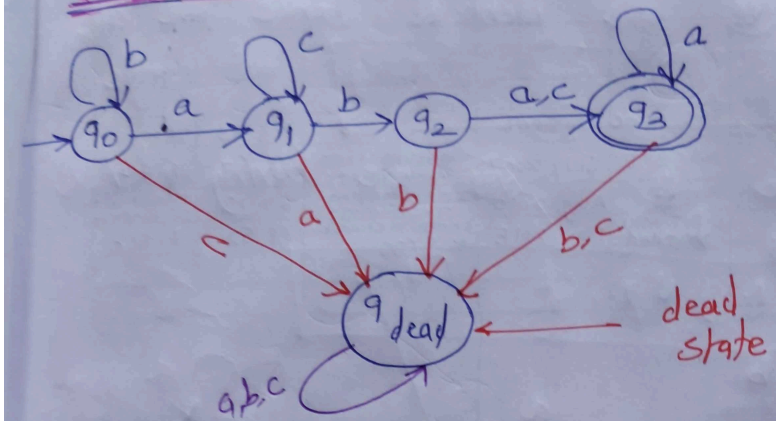


eg Exactly 1 as & 1 bs in string

string



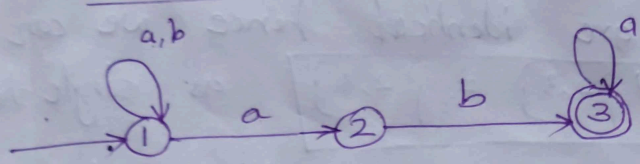
Dead states



	a	b	c
→ q_0	q_1	q_0	q_{dead}
q_1	q_{dead}	q_2	q_1
q_2	q_3	q_{dead}	q_3
(q_3)	q_3	q_{dead}	q_{dead}
q_{dead}	q_{dead}	q_{dead}	q_{dead}

The above Transaction Diagram Containing q_0 as initial state, q_3 Accepting state The transactions which are not going to other state with alphebets will goes to dead state

★ Convert NFA to DFA



NFA Containing 3 states hence possible No of states in NFA may be $2^n = 2^3 = 8$ states

$2^Q = \{\{\phi\}, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$.

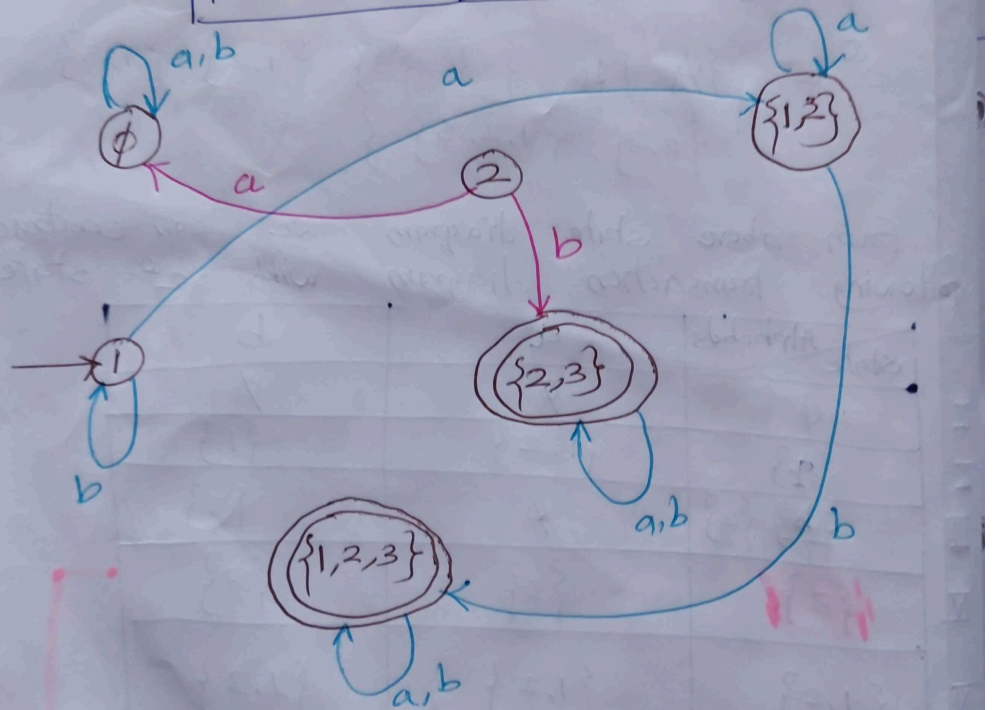
From above state diagram we can construct following transaction diagram with 2^Q states

	State \ Alphabets	a	b
I	ϕ	ϕ	ϕ
II	$\{1\}$	$\{1,2\}$	$\{1\}$
III	$\{2\}$	ϕ	$\{3\}$
IV	$\{3\}$	$\{3\}$	$\{3\}$
V	$\{1,2\}$	$\{1,2\}$	$\{1,3\}$
VI	$\{1,3\}$	$\{1,2,3\}$	$\{1,3\}$
VII	$\{2,3\}$	$\{3\}$	$\{3\}$
VIII	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,3\}$

From above table we seen that row (IV) & (VII) are identical hence we can take $\{3\} \cup \{2,3\} = \{2,3\}$ as single state.

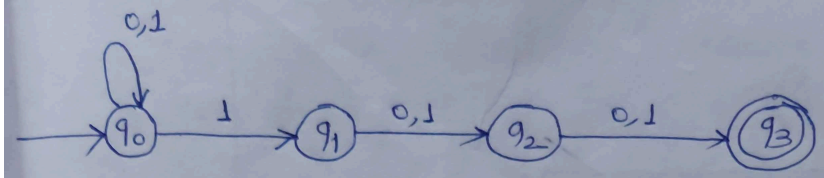
Likewise (VI) & (VIII) are having identical transactions hence

$$\{1,3\} \cup \{2,3\} = \{1,2,3\}$$



Hence Above is required DFA from given NFA

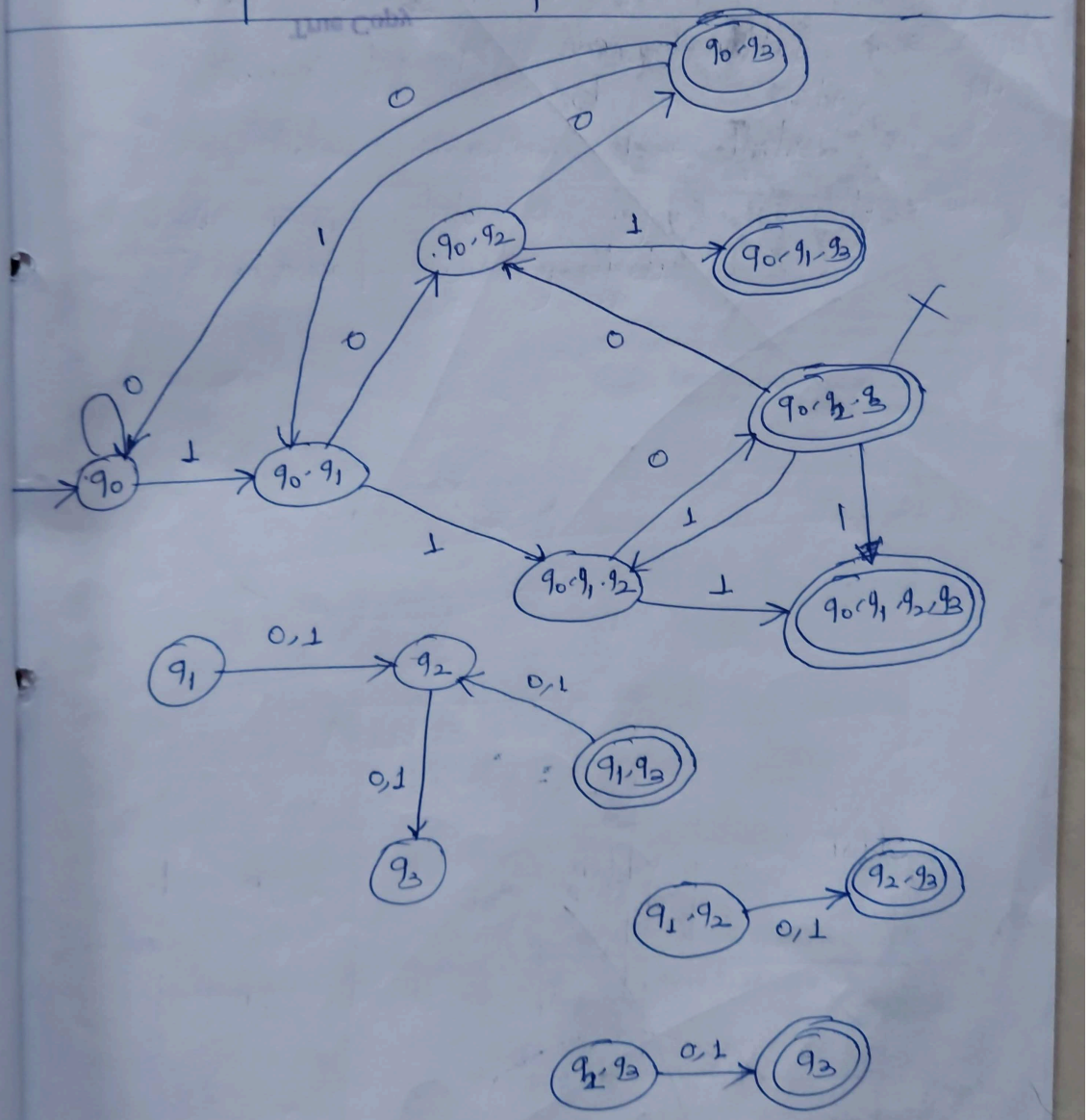
NFA to DFA by subset Construction Method.



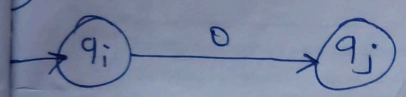
Now by subset Construction Method we can find all possible subset of states & find transition table ie $2^4 = 16$ states

State	Alphabet 0	Alphabet 1	
ϕ	ϕ	ϕ	← Empty
q_0	q_0	$\{q_0, q_1\}$	
q_1	q_2	q_2	
q_2	q_3	q_3	
q_3	ϕ	ϕ	← Empty
q_0, q_1	$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$	
q_0, q_2	$\{q_0, q_3\}$	$\{q_0, q_1, q_3\}$	
q_0, q_3	q_0	$\{q_0, q_1\}$	
q_1, q_2	$\{q_2, q_3\}$	$\{q_2, q_3\}$	
q_1, q_3	$\{q_2\}$	$\{q_2\}$	
q_2, q_3	q_3	q_3	
q_0, q_1, q_2	$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$	
q_0, q_1, q_3	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$	
q_0, q_2, q_3	$\{q_0, q_3\}$	$\{q_0, q_1, q_3\}$	

	0	1
$q_1\ q_2\ q_3$	$\{q_2, q_3\}$	$\{q_2, q_3\}$
q_0, q_1, q_2, q_3	$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$

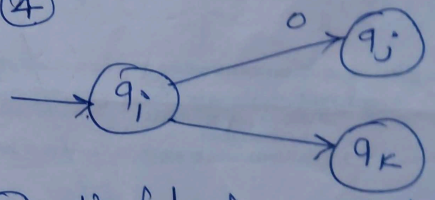


DFA

1. Having one state for given alphebet
2. Transaction function
   $\delta : Q \times \Sigma$
3. Language recognized by DFA called RL
4. 

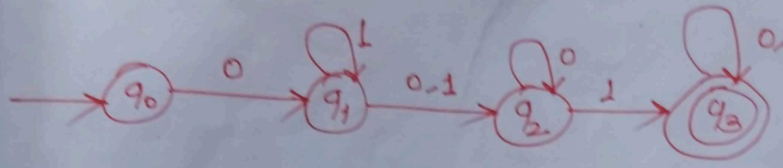
5. Harder to Construct due to limited states
6. Practical Implementation is feasible

NFA

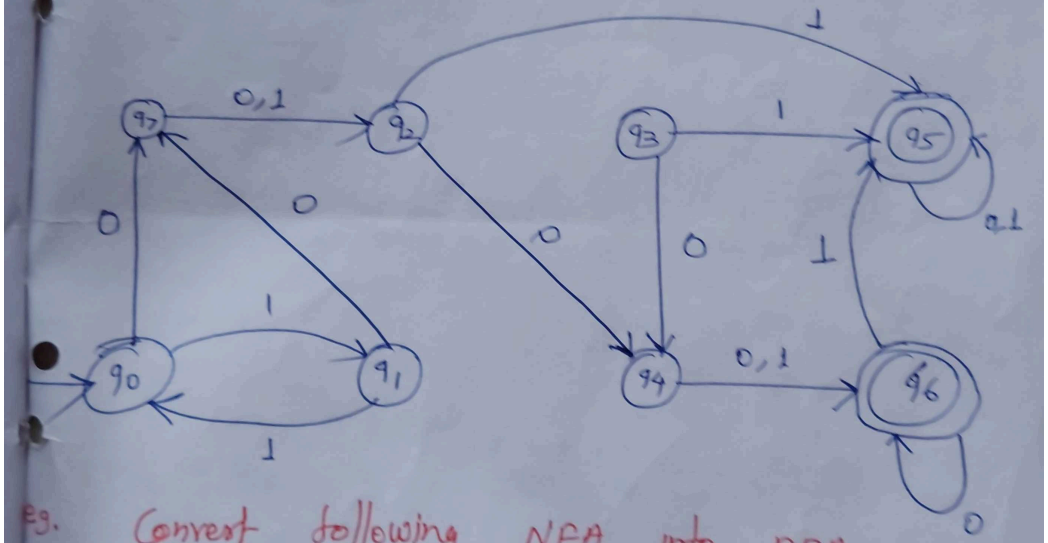
1. More than One state for single Alphebet
2. $\delta : Q \times \Sigma \rightarrow 2^Q$ (Subset of Q)
3. Outgoing transactions are non-deterministic
4. 

5. Useful for modelling of DFA & RE
6. Used in string Processing eg. grep Lexical Analyser
7. toss of coin, roll of dice
8. Easier to Construct
9. Practical Implementation is difficult need to convert into DFA.

* Convert following NDA to DFA



eg * Minimize following DFA



eg. Convert following NFA into DFA