Chapter 2: Finite Automata

* Finite Automata :

* BASIC MACHINE :

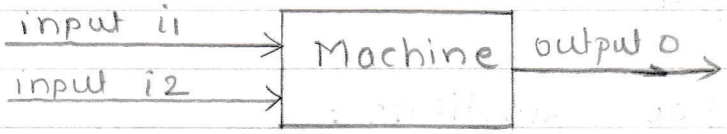
→ It is the machine, which recognizes as input set I and produces an output set O, where I and O are finite.

→ There is a one to one mapping from input to output Such a function relating input to output is known as MAchine Function or MAF.

→ The input and the output alphabet could be different.

* Features of Basic Machine :

1) It only interprets a set of input data and produces output set, which involves a combination of inputs. Hence also called as combinational Machine. E.g. AND gate.



2) It performs only a table look-up procedure from finit size table i.e. Machine function (MAF) table.

3) It has niether memory nor internal states.

4) The basic machine has a limited capacity. It will only react on the input.

To improve the capacity of the machine it is very necessary that we should consider the state of mind

during the design of the machine. There are the internal states of the machine.

Suppose you want the basic machine to check whether the given word is from the given language, which is infinite, and produce the output as yes or no. Practically it is impossible to create such a table, which can store an infinite word set. Therefore, we have to improvise the working by considering the machine with intermediate stages, which on receival of input chooses a particular path from initial stage to reach the final stage to produce valid output. Such a machine is having finite number of internal states, called as Finite State Machine (FSM).

* FINITE STATE MACHINE:

In FSM, the internal state of the machine alters when the machine receives an input and generates the required output.

It consists of two functions:

i) Machine function (MAF): $S \times I \rightarrow O$ (state) (input) (O/P)

ii) State function (STF): $S \times I \rightarrow S$ (next state)

MAF: At a particular state, for a given input, what is the output.

STF: At a particular state on receiving the input which is the next state we reach.

Both of them are functions of two arguments, current state and current input but their results are different.

during the design of the machine. There are the internal states of the machine.

Suppose you want the basic machine to check whether the given word is from the given language, which is infinite, and produce the output as yes or no. Practically it is impossible to create such a table, which can store an infinite word set. Therefore, we have to improvise the working by considering the machine with intermediate stages, which on receival of input chooses a particular path from initial stage to reach the final stage to produce valid output. Such a machine is having finite number of internal states, called as Finite State Machine (FSM).

* FINITE STATE MACHINE:

In FSM, the internal state of the machine alters when the machine receives an input and generates the required output.

It consists of two functions:

i) Machine function (MAF): $S \times I \rightarrow O$ (state) (input) (O/P)

ii) State function (STF): $S \times I \rightarrow S$ (next state)

MAF: At a particular state, for a given input, what is the output.

STF: At a particular state on receiving the input which is the next state we reach.

Both of them are functions of two arguments, current state and current input but their results are different.

* FA without output will be defined as a collection of 5 things or 5 tuples.

$FA = (Q, \Sigma, \delta, Q_0, F)$

where,

Q → set of states
Σ → Input alphabets
δ → transitions $\delta(q_1, i) = q_2$ (STF)
Q_0 → start state
F → set of final states

* FSM Design :

step 1 : Theory
step 2 : Logic
step 3 : Implementation
step 4 : Example

Q.1. Design a FSM to check whether the given decimal no. is divisible by 3.

→

~~step 1~~ : ~~Theory~~

To check divisibility by 3, sum of digits divisibility will not work in this case.
So, we can check divisibility of each no. based on remainders.

step 2 : Logic

Given input is decimal number

$\therefore I = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

* FSM properties/limitations:

1) Periodicity:

The limitation of FSM is that it does not have the capacity to remember arbitrarily large amount of information, because it has only a fixed number of states and this sets a limit to the length of the sequence it can remember. Also we have seen a finite control representation of FSM, where read head moves always one position to the right after reading an input symbol. Head can never move in reverse direction, therefore FSM cannot retrieve what it read previously, before coming to current position of tape. As it cannot retrieve we cannot say, it can remember something. This also means that FSM eventually will always repeat a state or produce a periodic sequence of states.

2) State Determination:

Since, the initial states of an FSM and the input sequence given to it, determines the output sequence, it is always possible to discover the unknown state, in which the FSM resides at a particular instance.

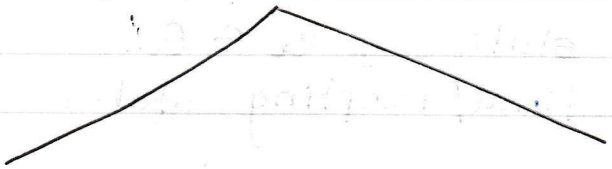
3) Impossibility of Multiplication:

An FSM cannot remember arbitrarily long sequences. Hence for multiplication operation it is required to remember two full sequences corresponding to multiplier and multiplicand, while multiplying, it is also required to store the partial sums that we obtain normally at intermediate stages of multiplication. Therefore, no FSM can multiply two given arbitrarily large numbers.

4) Impossibility of palindrome recognition:
FSM cannot recognizing a palindrome, because it doe not have that capability to remember all the symbols it reads until half the way point of input sequence, in order to match them in reverse order, with the symbols in second half of the sequence.

5) Impossibility to check well-formed ness of parenthes
As FSM has no capability to remembers all the earlier inputs to it, cannot compare with the remaining to check well formed ness. It is an impossible task for any FSM.

Finite Automata → (Mathematical model of FSM)



FA with final state (no o/p)	FA with o/p (no final state)
1) NFA	1) Moore m/c
2) DFA	2) Mealy m/c
eg string matching	is complement

* DFA (Deterministic FA)

- It consists of finite set of states, one state is called start/initial state and there are one or mo final/accepting states
- i.e. ~~A~~ FA is called deterministic if, it consists of a ~~From~~ finite set of states and set of transitions from state to state that occur on input symbol chosen from an alphabet Σ.
- from every vertex of its transition graph, there i an unique input symbol which takes the vertex state to the required state.
- i.e. from each state, on each i/p symbol, there is exactly one transition
- DFA can be represented mathematically by 5-tuples

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

where,

$Q \rightarrow$ Finite set of states

$\Sigma \rightarrow$ finite input alphabet

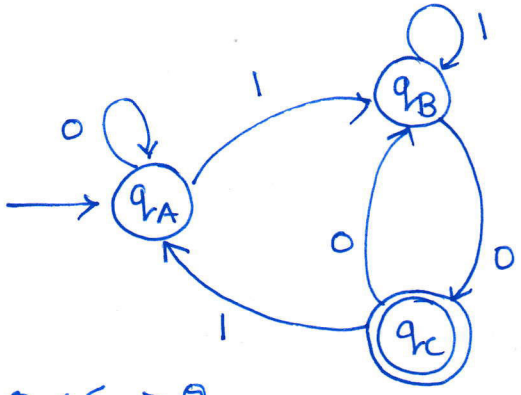
$\delta \rightarrow$ transition function

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

$q_0 \rightarrow$ initial state, $q_0 \in Q$

$F \rightarrow$ set of final/accepting states $(F \subseteq Q)$

Transition Graph:



$Q = \{q_A, q_B, q_C\}$

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

$q_0 = q_A$

$F = \{q_C\}$

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

$Q \backslash \Sigma$	0	1
q_A	q_A	q_B
q_B	q_C	q_B
q_C*	q_B	q_A

Acceptance by DFA:

From start state, scanning each input symbol left to right, the string is recognized by DFA if we are able to reach one of the final states of DFA when input is over.

$$(q_0, w) \vdash^* (q_f, \epsilon) \qquad (q_f \in F)$$

$*$ → sequence of moves

e.g: $\vdash (q_A, 110)$

$\vdash (q_B, 10)$

$\vdash (q_B, 0)$

$\vdash (q_C, \epsilon)$

* NFA (Non-deterministic FA)

- It consist of finite set of states, one state is called start/initial state and there are one or more final/accepting states.
- i.e. A FA is called non-deterministic if, it consist of a finite set of states and a set of transition from state to state that occur on input symbol chosen from an alphabet Σ.
- In NFA, from every vertex of its transition graph, there is more than one possible transition on the same input symbol from some state.
- i.e. from each state, on each i/p symbol, there can be 0, 1 or more transitions.

- NFA can be represented mathematically by 5-tuple:

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

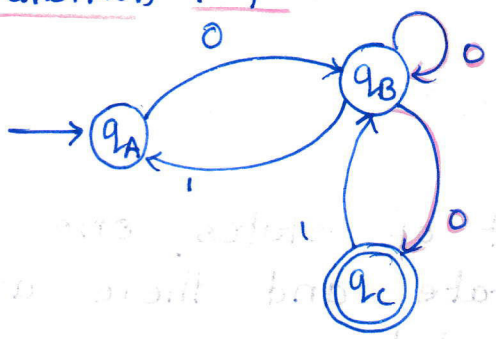
where,

$Q \rightarrow$ finite set of states
$\Sigma \rightarrow$ finite input alphabet
$\delta \rightarrow$ transition ~~state~~ function
$\delta : Q \times \Sigma = 2^Q$
$q_0 \rightarrow$ initial state, $q_0 \in Q$
$F \rightarrow$ set of accepting/final states ($F \subseteq Q$)

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

$2^A = \{\phi, \{1\}, \{2\}, \{1, 2\}\}$

transition Graph:



$Q = \{q_A, q_B, q_C\}$

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

$q_0 = q_A$

$F = \{q_C\}$

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

$Q \backslash \Sigma$	0	1
q_A	q_B	$\{\}$
q_B	$\{q_B, q_C\}$	q_A
q_C^*	$\{\}$	q_B

* In, NFA when we draw transition diagram then at some state we will have 2 outgoing arrows with the input symbols. Where as in DFA with every state there is exactly one outgoing arrow for each input symbol.

Acceptance by NFA:

$(q_0, w) \vdash^* (q_f, \epsilon)$

e.g. $(q_A, 00)$

$\vdash (q_B, 0)$

$\vdash (q_B, \epsilon)$ $\quad$ $\vdash (q_C, \epsilon)$

* It is known as Non-deterministic, as any state either q_i or q_k can be travelled on a and some times we require backtracking to properly check the validity of the string.

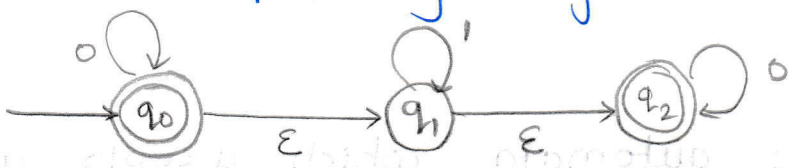
Here, if any one state is final the string is valid.

* NFA will be easier to conceptualize. With some modification it can be converted to DFA.

* Finite Automata with ε moves :

The FA with ε moves is the machine in which the state can be changed without receiving any input symbol. ε move indicate that it is a NULL movement from one state to another. or we can say the two states are equivalent.

Consider the following diagram :



In above diagram the state can be changed without any input

Strings accepted by above state diagram are :

- only 0's	0000
- Only 1's	1111
- 0's followed by 1's	00001111
- 1's followed by 0's	11000
- 0's followed by 1's followed by 0's	000111000

It is easier way to express the language but difficult to actually implement as the state can be changed at any moment. So it would require a proper backtracking to check the validity.

e.g. for input 000

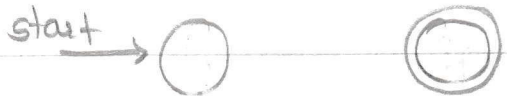
$q_0000 \rightarrow 0q_000$
$\rightarrow 00q_00$
$\rightarrow 000q_0$
with ε move $\rightarrow 000q_1$ which is not final state so not accepted.

* Regular Expression to NFA (Thompson's Construction me

→ Divide the regular expression into its constituent subexpressions and create the NFA for each expression using Rule 1, 2 and 3.

→ Combine the NFAs created by the above step using Rule 4.

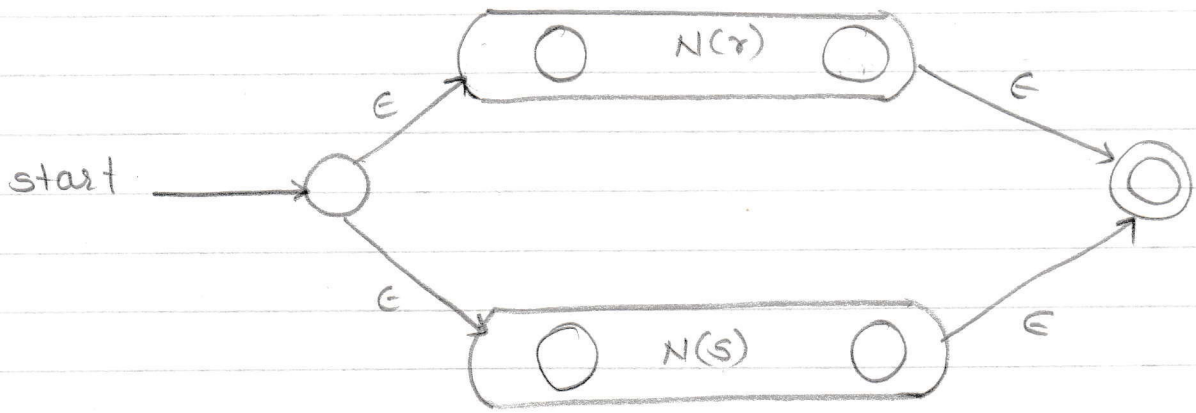
* Rule 1: NFA for RE $r = \phi$ i.e $\{\}$



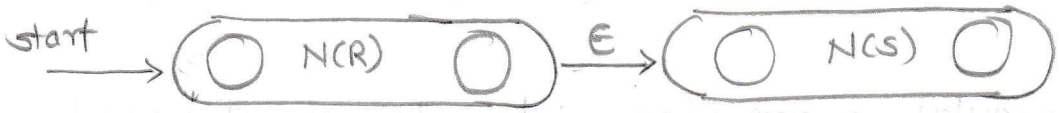
* Rule 2: NFA for RE $r = \epsilon$ i.e. $\{\epsilon\}$

* Rule 3: NFA for RE $r = a$ i.e. $\{a\}$

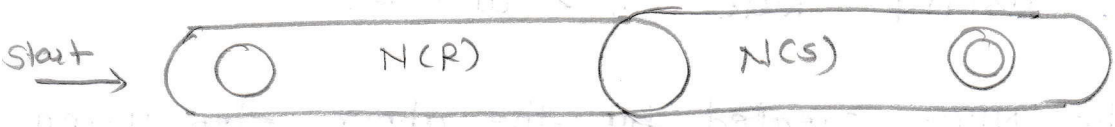
* Rule 4:
i) NFA for RE $r = (R) / (S)$ i.e. $L(R) \cup L(S)$



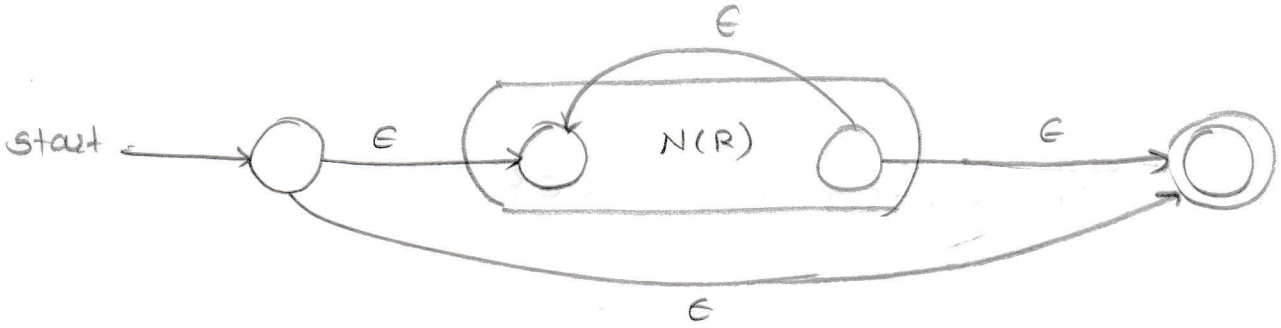
ii) NFA for RE $r = (R) \cdot (S)$ i.e. $L(R) \cdot L(S)$



OR



iii) NFA for RE $r = (R)^*$ i.e. $L(R)^*$



* Conversion of NFA to DFA

* ε closure of a state:

It is defined as the set of states that are reachable from that state by walking on ε-transitions alone by including that state too.

e.g: Consider previous NFA of $r = (a+b)^* abb$

ϵ - closure (0) = {0, 1, 2, 4, 7}

ϵ - closure (3) = {1, 2, 3, 4, 6, 7}

ϵ - closure (8) = {8}

* ε- closure of set of states:

It is defined as the union of ε-closure of each state of the set

e.g. ϵ - closure ({3, 8})

= ϵ - closure (3) $\cup$ ϵ - closure (8)

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

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

* Important Note: (Refer while solving next example)

If the final state is the part of y, then we make that y's state as final state as shown in δ (transition table).

In this case, 10 is the final state and it is part of y of state E. Therefore in the transition table we have shown state E as final.

* DFA minimization : (Classical Method)

* Important states :

state is important if there is a transition over the given Σ.

* Minimization Rule :

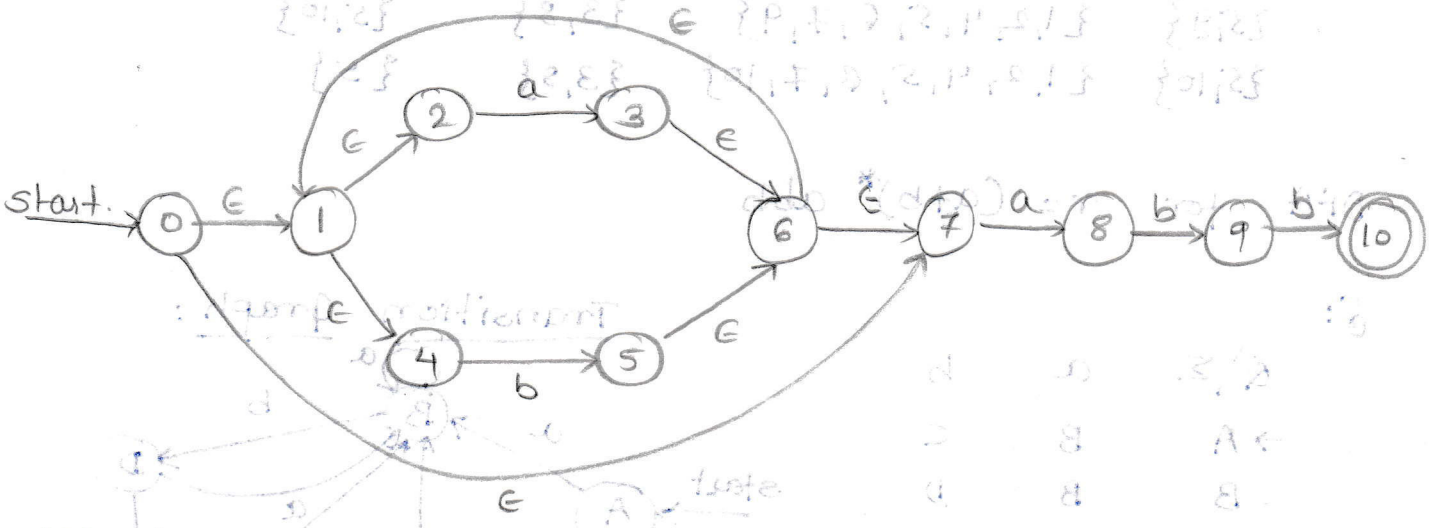
states can be merged if

(all states have same important states) & (all states include the final state or all states exclude the final states)

Q. Construct NFA for RE. $(a+b)^*$ abb and convert into min DFA.

→

NFA for $(a+b)^*$ abb is as follows:



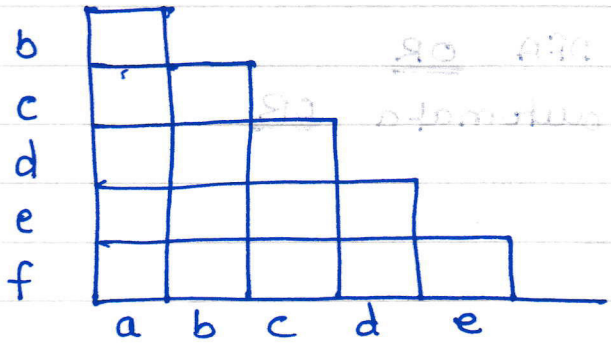
NFA to DFA :

	x	y = ε-closure(x)	δ(y,a)	δ(y,b)
A	{0}	{0,1,2,4,7}	{3,8}	{5}
B	{3,8}	{1,2,3,4,6,7,8}	{3,8}	{5,9}
C	{5}	{1,2,4,5,6,7}	{3,8}	{5}
D	{5,9}	{1,2,4,5,6,7,9}	{3,8}	{5,10}
E	{5,10}	{1,2,4,5,6,7,10}	{3,8}	{5}

* DFA Minimization (2nd Method - Box method)

Step 1: Remove all those states that are not rechable from start state.

Step 2: Create a table such that the rows and columns would be identified by state numbers.
e.g. if a, b, c, d, e, and f are the states, table is,



Step 3: Mark all those cells in which one state is final and other is non-final with "1". (Since these states are distinguishable states)

Step 4: If there are some unmarked cells, then we can check whether they can be marked as follows:

Let (P, q) be an unmarked cell and let 'a' be some input symbol.

i) If $\delta(P, a) = r$ and $\delta(q, a) = s$ then mark (P, q) with "2", if (r, s) is marked else put (P, q) in list (r, s).

ii) If $\delta(P, a) = r$ and $\delta(q, a) = r$ then we cannot take any decision w.r.t. such input

If the cell (P, Q) is still unmarked then repeat the step 4 with next i/p (if available)

step 5: All those cells that are unmarked represent the equivalent states which can be merged into a single state respectively.

Note: If a DFA is given and we are asked to minimize it then always do it by using box method else do it, using classical method.

Q: Find the minimized DFA **OR**
Find the minimized automata **OR**
Find minimized FA.

Q/Σ	a	b
→ A	B	C
B	B	D
C	B	C
D	B	E
E*	B	C

	A	B	C	D
→ B	2			
C		2		
D	2	2	2	
final → E	1	1	1	1

non final

Marked (A,E), (B,E), (C,E) & (D,E) as 1 as E is final state.

(A;B)

$\delta(A,a) = B$ $\quad \delta(B,a) = B$ ×

$\delta(A,b) = C$ $\quad \delta(B,b) = D$

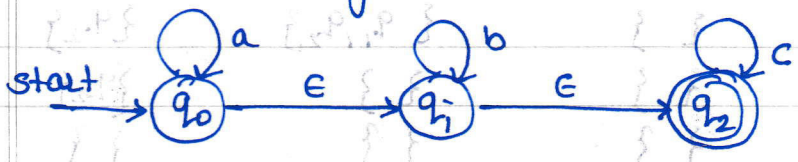
List (C, D) = (A, B)

* NFA with ϵ to NFA without ϵ :

* **Note:**

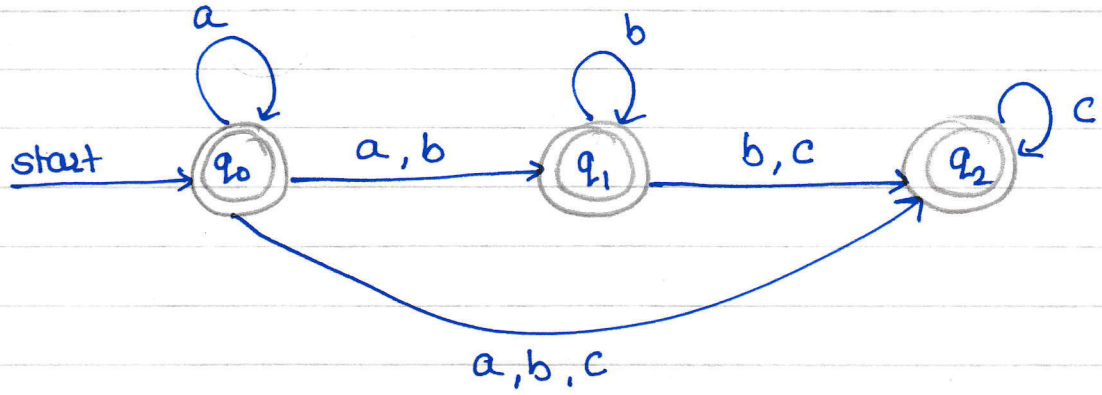
i) The number of states remain the same.
ii) Start state remains the same.
iii) Final state remains the same.
iv) If ϵ-closure of any state contains a final state then make that state also a final state.

Q. Convert the given NFA with ϵ to NFA without ϵ.



← NFA with ϵ-transit

→	x	y = ϵ-closure(x)	δ(y,a)	ϵ-closure (δ(y,a))	δ(y,b)	ϵ-closure (δ(y,b))	δ(y,c)	ϵ-closure (δ(y,c))
q_2 present so all states are final states	q_0*	$\{q_0, q_1, q_2\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$	$\{q_1\}$	$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
	q_1*	$\{q_1, q_2\}$	$\{\}$	$\{\}$	$\{q_1\}$	$\{q_1, q_2\}$	$\{q_2\}$	$\{q_2\}$
	q_2*	$\{q_2\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$	$\{q_2\}$	$\{q_2\}$



NFA - without ϵ - transition

TD to RE

Arden's Theorem | state elimination method

J.C. Martin

* Arden's Theorem : (TD to RE)

Let P and Q be two regular expressions over Σ. If P does not contain ϵ, then the following equation in R

$$R = Q + RP$$

has unique solution given by

$$R = QP^*$$

* The following assumptions are made about the NFA:

1) The NFA does not have ϵ-transitions
2) It has only one initial state, say v_1
3) The vertices are $v_1, v_2, \ldots, v_n$.
4) v_i is the R.E. representing the set of strings accepted by the system even if v_i is the final state
5) α_{ij} denotes the R.E. representing the set of labels of edges from v_i to v_j, when there is no edge, $\alpha_{ij} = \phi$.

Consequently we can get the following set of equations for $v_1 \ldots v_n$.

$$v_1 = v_1\alpha_{11} + v_2\alpha_{21} + \cdots + v_n\alpha_{n1} + \epsilon$$
$$v_2 = v_1\alpha_{12} + v_2\alpha_{22} + \cdots + v_n\alpha_{n2}$$
$$\vdots$$
$$v_n = v_n\alpha_{1n} + v_2\alpha_{2n} + \cdots + v_n\alpha_{nn}$$

By applying substitution, we can express v_i in terms of α_{ij}'s.

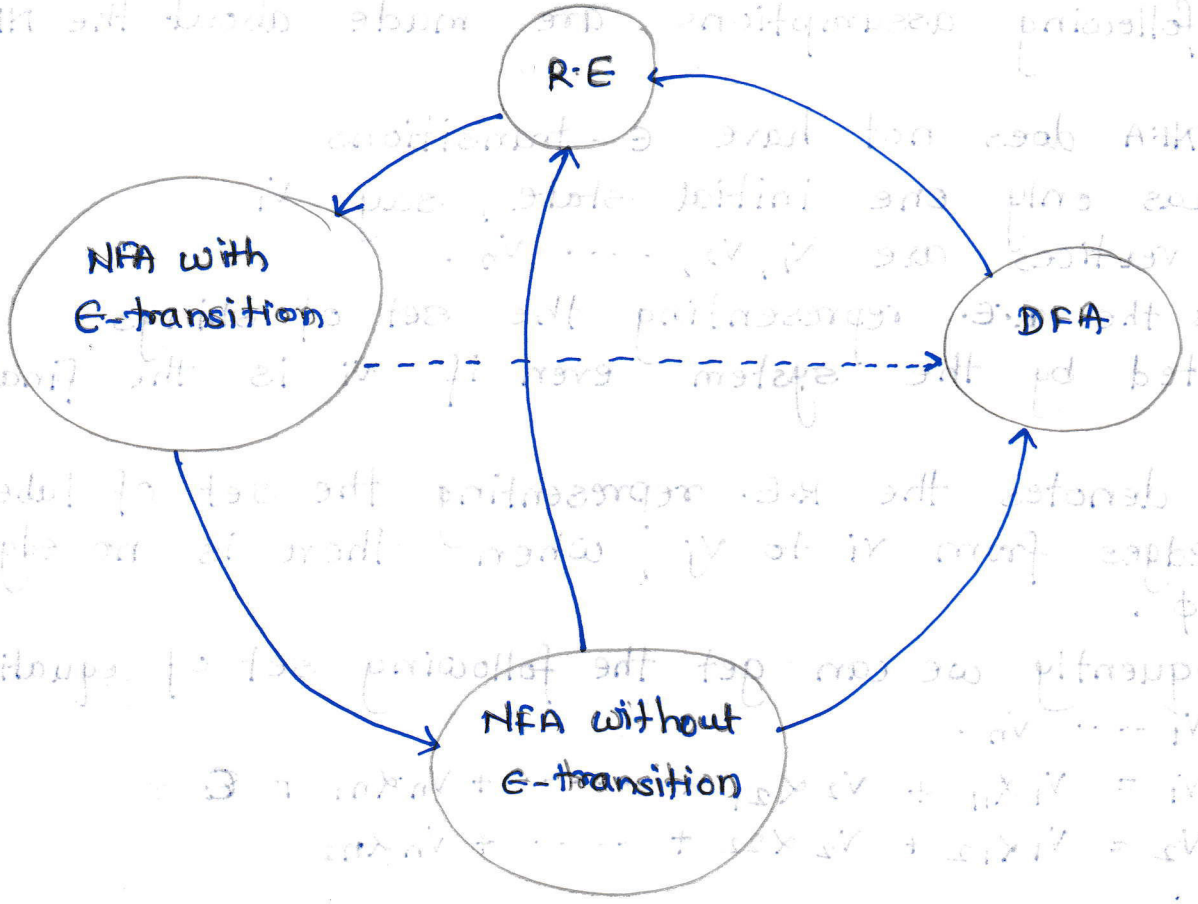
For getting the set of strings recognized by the NFA, we have to take the 'union' of all v_i's corresponding to final states.

Simple steps:

Step 1: Write the state equation by looking at the incoming transitions. (Add ϵ to start state ~~transi~~ equation)

step 2: Perform any combination of
- substitution
- rearrangement
- Arden's formula

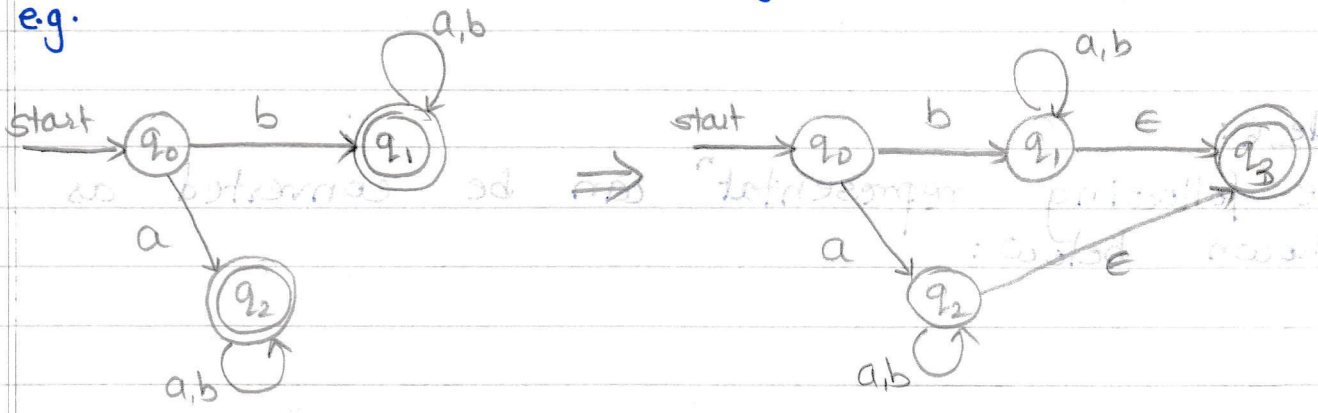
until we are able to get final state's equation in terms of Σ (RE)



TD to RE using state elimination method

Rule 1: TD must have one initial state and one final state. If there are more than one final states then convert it into a single final state.

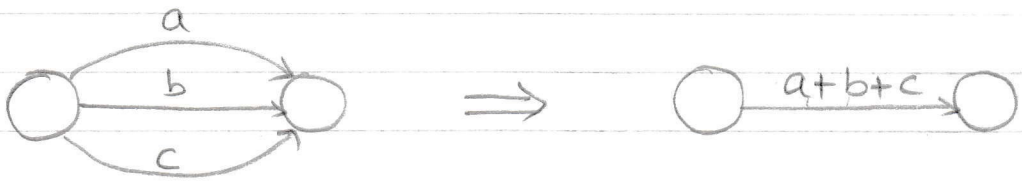
e.g.



* **Rule 2**:

If there is more than one transition from one state to another state they can be represented together as follows:

e.g.



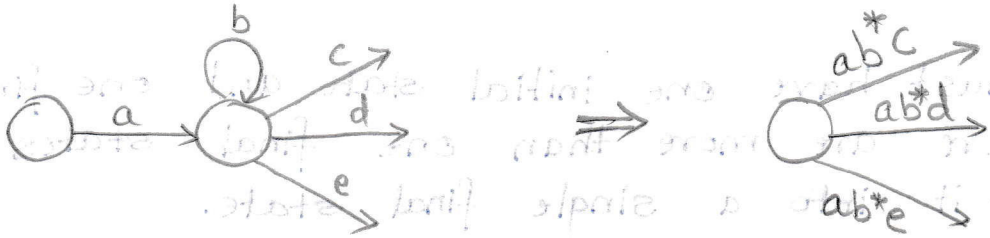
* **Rule 3**:

If there is following representation then combine them as follows:

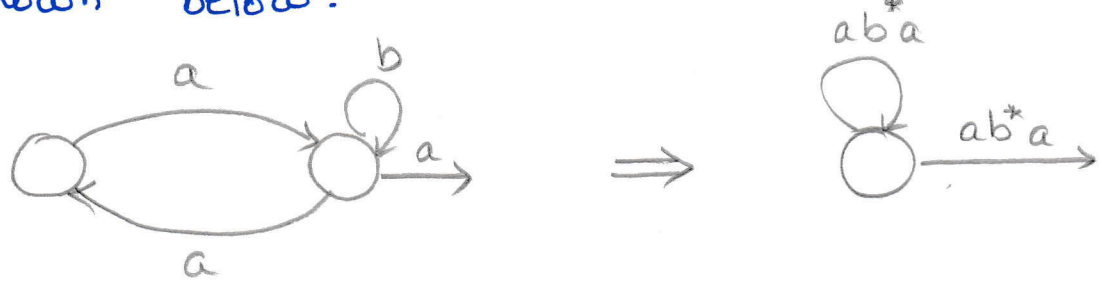
* **Rule 4**:

If there are more than one self loops on a state then it can be represented as follows:

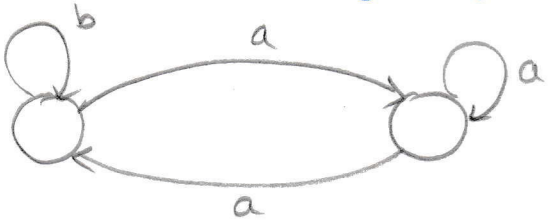
* **Rule 5:**
If we have following representation, then it can be represented as shown below:



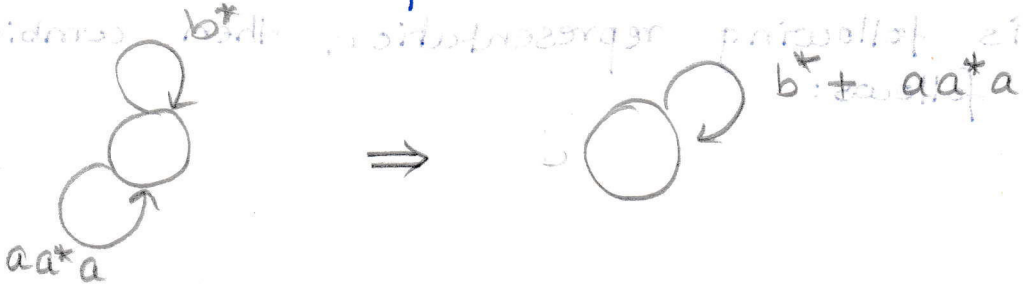
Rule 6:
The following representat^{n} can be converted as shown below:



Rule 7:
If we have following representation



then we can represent it as follows:



Finite Automata with output

* Moore Machine :

- It is FA with no final state and it produces the o/p sequence for the given i/p sequence.
- In Moore machine a symbol that is associated with each state is called o/p symbol.

* Representation :

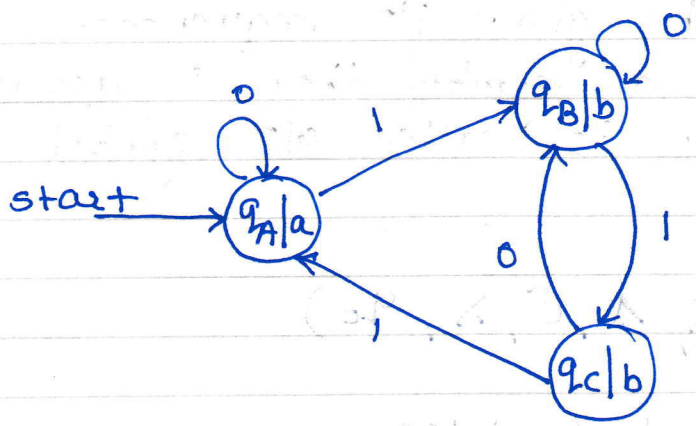
– A Moore machine is represented using 6 tuple as shown below :

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

where,

Q = finite set of states
Σ = i/p alphabet (finite)
Δ = o/p alphabet (finite)
δ = transition function , $\delta : Q \times \Sigma \to Q$
λ = o/p mapping fnc^n , $\lambda : \lambda(Q) \to \Delta$
q_0 = initial/start state , $q_0 \in Q$

e.g :



Here, $Q = \{q_A, q_B, q_C\}$
$\Sigma = \{0, 1\}$
$\Delta = \{a, b\}$
$q_0 = q_A$

δ:

$Q \backslash \Sigma$	0	1
q_A	q_A	q_B
q_B	q_B	q_C
q_C	q_B	q_A

λ:

$\lambda(q_A) = a$

$\lambda(q_B) = b$

$\lambda(q_C) = b$

Note:

In Moore machine if i/p is of length n then o/p is of length n+1.

e.g: Let i/p be 010

i/p	o/p
$\delta(q_A, 010)$	a
$\delta(\delta(q_A,0), 10)$	
$\delta(q_A, 10)$	a
$\delta(\delta(q_A,1),0)$	
$\delta(q_B, 0)$	b
q_B	b

i/p → 010 → n

o/p → aabb → n+1

* Mealy Machine :

- It is FA with no final state and it produces the o/p sequence for the given i/p sequences.
- In Mealy machine a symbol that is associated with each transition is called o/p symbol.

* Representation: (By 6 tuples)

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

where,

Q = finite set of states

Σ = i/p symbol (finite)

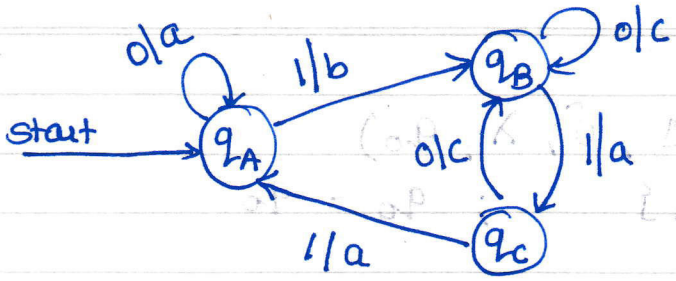
Δ = o/p symbol (finite)

δ = state transition function, $\delta : Q \times \Sigma \to Q$

λ = machine function, $\lambda : Q \times \Sigma \to \Delta$

q_0 = initial / start state

e.g.



Here, $Q = \{q_A, q_B, q_C\}$

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

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

$q_0 = q_A$

δ:

$Q \backslash \Sigma$	0	1
q_A	q_A	q_B
q_B	q_B	q_C
A_C	q_B	q_A

λ:

$Q \backslash \Sigma$	0	1
q_A	a	b
q_B	c	a
q_C	c	a

Note:

In Mealy machine if i/p is of length n, then o/p is also of length n.

e.g: Let i/p string be 010

i/p	o/p
$\delta(q_A, 010)$	
$\delta(\delta(q_A, 0), 10)$	a
$\delta(q_A, 10)$	
$\delta(\delta(q_A, 1), 0)$	b
$\delta(q_B, 0)$	c
q_B	

Note:

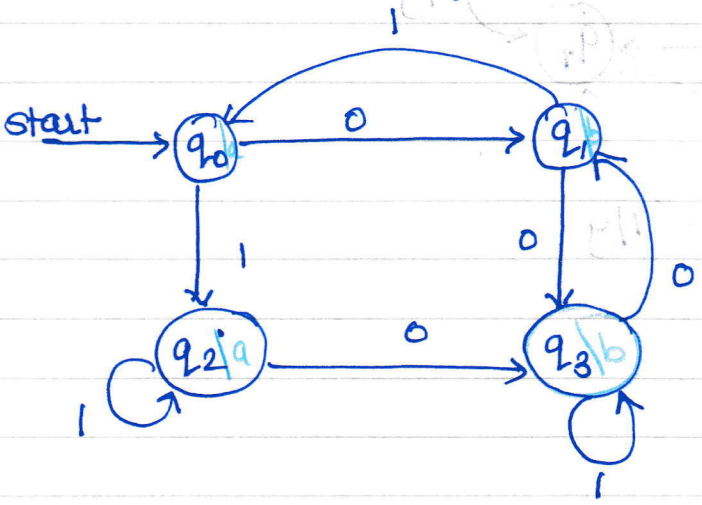
Thus in Moore machine, output depends on current state of the machine, irrespective of what the i/p on which the transition is made while in Mealy m/c o/p depends on both current state and the current i/p symbol.

* Moore Machine to Mealy Machine :

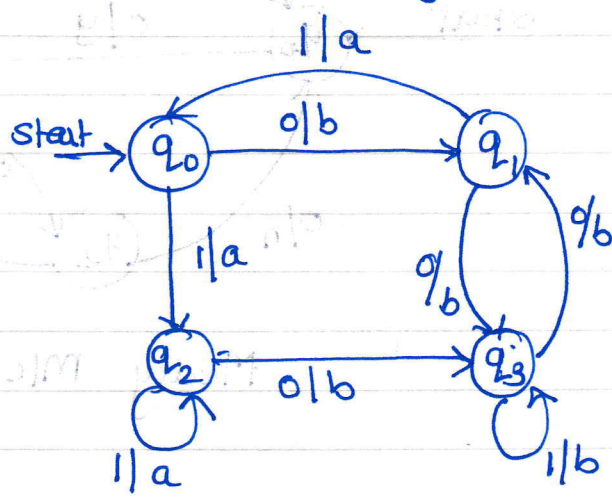
- If $M1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ is Moore machine, then equivalent Mealy Machine is $M2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$ where, $\lambda'(q, a) = \lambda(\delta(q, a))$ for all states q and input symbols a.

Step : Assign the o/p symbol associated with the state to all of its incoming transitions.

e.g: Convert the given Moore m/c to Mealy m/c.



Moore m/c.



Mealy m/c.

* Mealy m/c to Moore m/c

- If given Mealy Machine is, $M1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ then, an equivalent Moore Machine is

$$M2 = ([Q \times \Delta], \Sigma, \Delta, \delta', \lambda', [q_0, b_0])),$$

where, b_0 is an arbitrarily selected member of Δ and $\delta'([q, b], a) = [\delta(q, a), \lambda(q, a)]$ and $([q, b]) = b$

Steps:

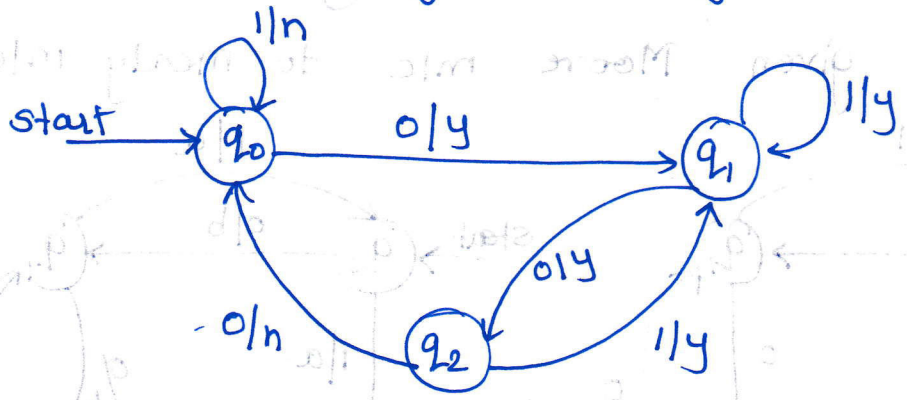
1) If o/p symbol with all incoming transitions to the

state are same, then assign the o/p symbol to that state.

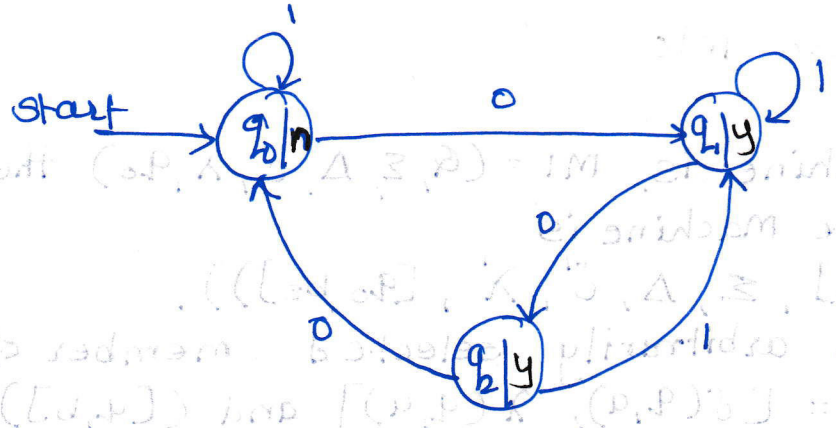
2) If the output symbols associated with all incoming transitions to a state are not same, then split the state as many times the o/p symbol, with each state producing a different o/p.

3) If there are no incoming transitions to a state, then assign any o/p symbol.

e.g: Convert the given Mealy m/c to Moore m/c.

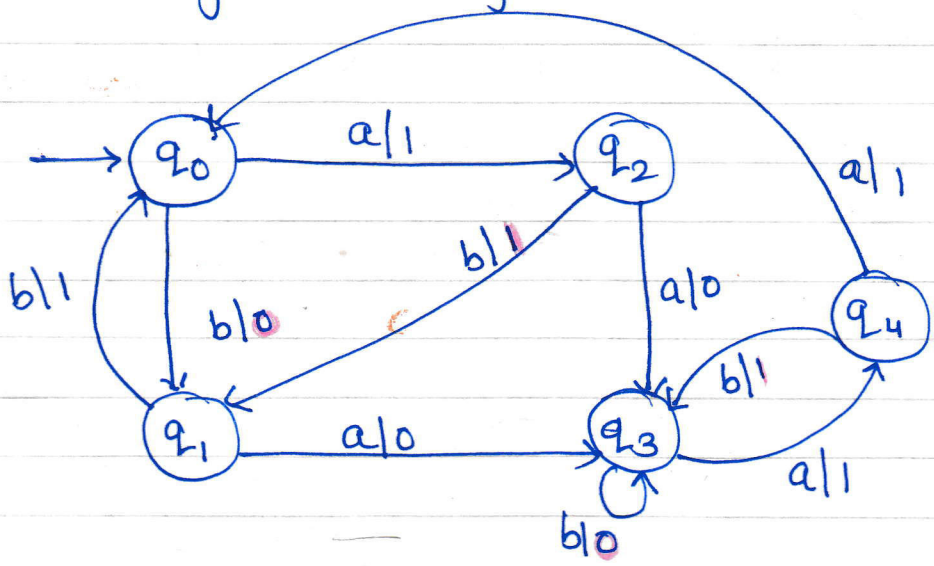


Mealy m/c.



Moore m/c

Q. Convert given Mealy m/c to Moore m/c.



Mealy m/c.

Here, q_1 has two output state '0' & '1', then split q_1 into two states q_{10} & q_{11}.
Now, if there is a transition from q_1 to q_0 with i/p b, then both the states q_{10} & q_{11} will send i/p transition on b to state q_0.
Similarly, for q_3 we have to split.