Unit-5:

Unsolvable Problems and Computable Functions.

→ The main objective of Unsolvable problems and computable functions is to learn about primitive recursive functions, construction of recursive & recursively enumerable Languages, enumerating the binary strings, generating codes for TM and construction of diagonalization language and Universal turing Machine.

Unsolvable problems & computable fns :-

— A pbm is computable if it can be solved in principle by a computing device.

Computable → solvable, decidable, recursive.

→ Computable functions → as a function that can be computed by any algorithm or any machine.

Primitive Recursive functions :-

→ Recursive functions are generated from the basic functions. Three basic functions using two operations are supported for primitive recursive functions.

3 basic functions are :

1. Zero function or constant function.
2. Successor function.
3. Projection function.

UNIT-5

Unsolvable Problems and Computational

tents:

1) unsolvability
2) Primitive Recursive functions
3) Recursive and Recursively Enumerable language
4) universal Turing Machine
5) Tractable & Intractable problems
6) Tractable & Possibly Intractable problem P & NP.
7) NP completeness
8) Polynomial Time Reductions
9) Proving NP-complete problem
10) Post's correspondence problem.
11) important Questions.

13 marks! (X) Question.

1) Describe in detail about recursively Enumerable language with example.
2) Explain in detail note on computable function with suitable example.

3) Explain in detail about primitive recursive function with examples?

4) Discuss detail notes on Enumerating a language with examples?

5) Universal Turing Machine with example,

6) Meaning of classifying complexity?

7) Tractable & Possibly intractable problems P & NP completeness?

8) Time & space computing of a Turing Machine?

Recursively Enumerable language:

Introductions:

- It has two categories. The first categories contains all language and the algorithms used for the language L in that turing Machine
- The second categories can't that the language can be modeled by TM but there no guarantee that the TM will eventually halt.

⇒ The first categories } → Recursive language (Predicted, it halts)

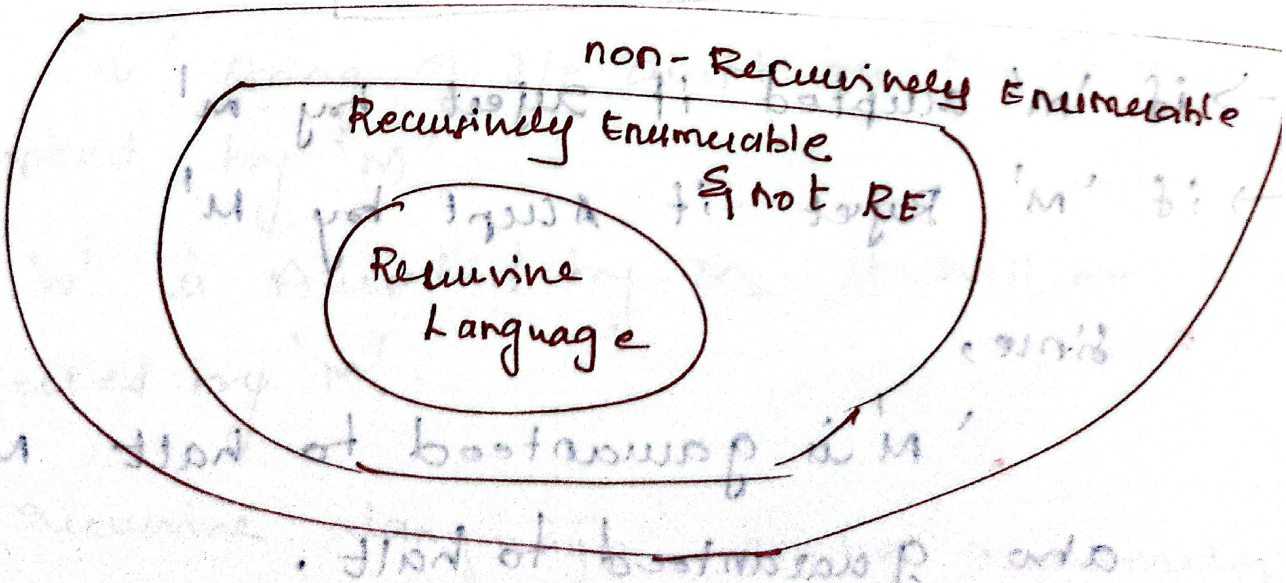
⇒ Second categories } → Recursively Enumerable language. (unpredictable, no gaurantee for halts).

categories of languages:

1) Recursive language for which the algorithm exists.

2) Recursively Enumerable language for which not sure that on which I/P the TM will ever halt.

3) Non recursively Enumerable language for which is no Turing Machine at all.



Property :

- decidable language (Recursive)
- undecidable language (Recursively Enumerable).

Theorem : (2)

If L is recursive language L' are also recursive language :

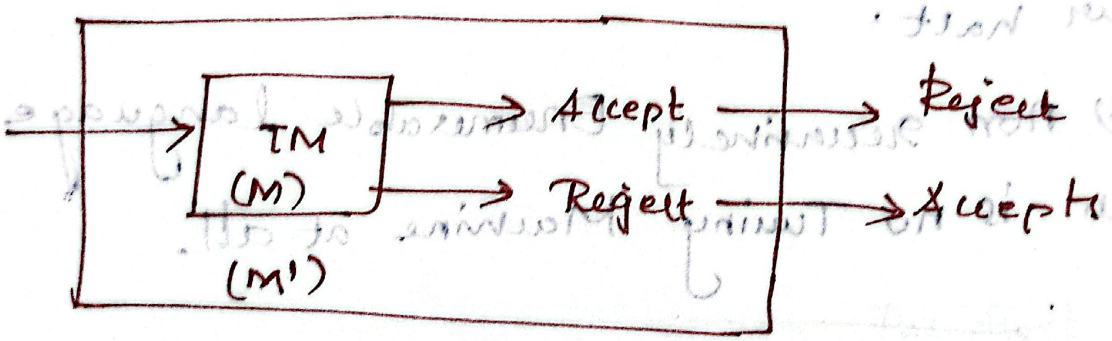
L - accepted by TM (M)

we denote, L(M)

'M' always halts.

Now,

M' such that L' = (M')



→ if 'm' accepted it reject by M'

→ if 'M' Reject it Accept by M'

- Since,
  - 'M' is guaranteed to halt M' is also guaranteed to halt.

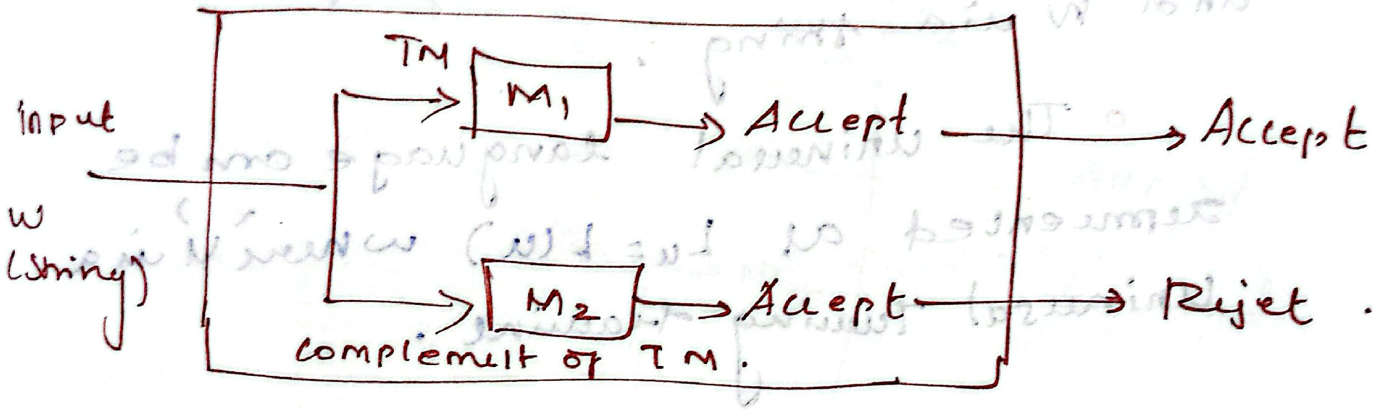
Theorm 2 :

If a language L and its complement L' both are RE then L is a recursive language.

Turing Machine 'M'

$\Rightarrow L(M) = L(M_1)L(M_2)$.

both M_1 and M_2 are parallel to each other.



- M_1 is Turing Machine
- M_2 is complement of TM.

→ if 'w' string of I/P accpt by M_1 it was accepted by 'M'

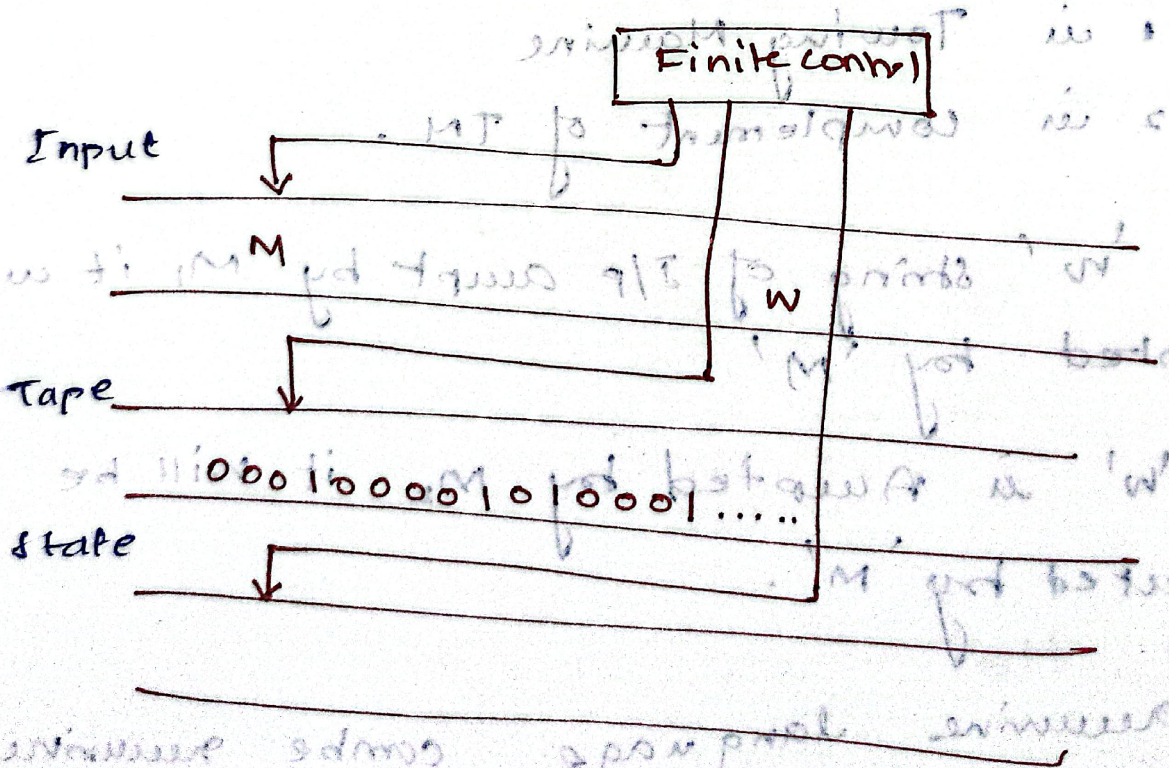
→ if 'w' is Accepted by M_2 it will be rejected by 'M'.

Thus a recursive language combe recursively enumerable but a RE language not necessarily be recursive.

UNIVERSAL TURING MACHINE

Introductions:

- The uninual language du ii a set of binary string wnich cans be modeled by a TM.
- The uninual language can be aepneented by a pair (M, w) wnene M ii TM that aupted this languages and w iia string.
- The uninual language can be aepneented as $L_u = L(U)$ wnenU' iia Uninusal Tuning Tnacnine.



- The universal Turing machine 'U' accepted by TM 'M'.

Undecidability of universal language: (∞)

where,

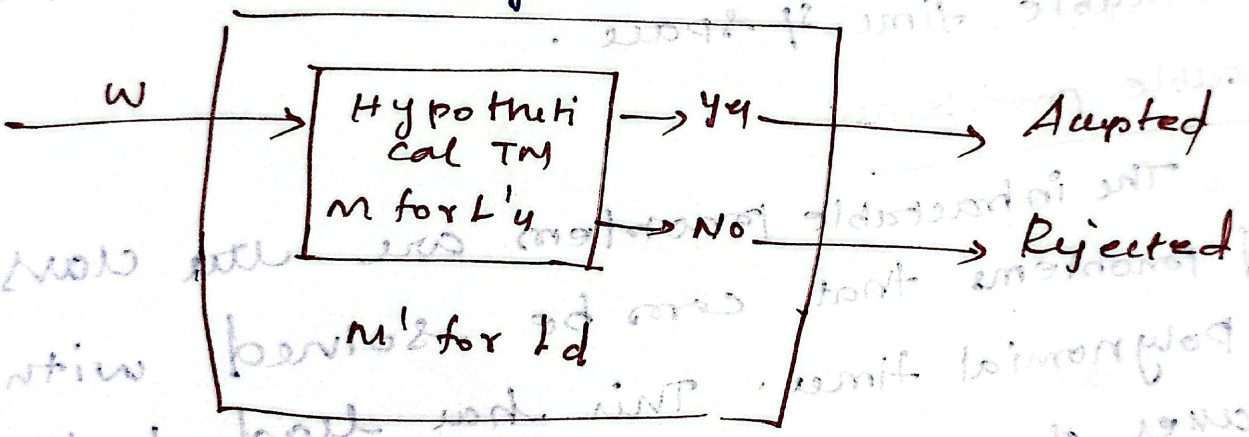
Theorm: → L_U is RE but not recursive.

Proof:

consider,

→ L_U is RE, assume L_U is recursive.

Then complement of L_U is L'_U.



→ L_d is diagonalized language it is not RE language

- Thus assume that L_U is Recursive is wrong, hence we proved that L_U is recursively Enumerable language.

Tractable and Intractable problems: (6m)

- Class of solvable problems is known as decidable problems. That means decidable problems can be solved in measurable amount of time or space.

Tractable problems:

- The tractable problems are the class of problems that can be solved within reasonable time of space.

Intractable problems:

- The intractable problem are the class of problems that can be solved within Polynomial times. This has lead to two classes of solving problems - P and NP class problems.

Computational complexity problems:

→ P - class.
→ NP - class.

P-class:

- Problem that can be solved in Polynomial time ("P" stands for polynomial).

example: Searching of key element, sorting of element, All pair shortest path.

NP- stand / problems:

- It stands for "non-deterministic polynomial time". Note that NP does not stand for "non-polynomial".

example:

- Traveling salesperson problem, graph coloring problem, knapsack problem Hamiltonian circuit problems.

NP-class

NP-hard NP-complete.

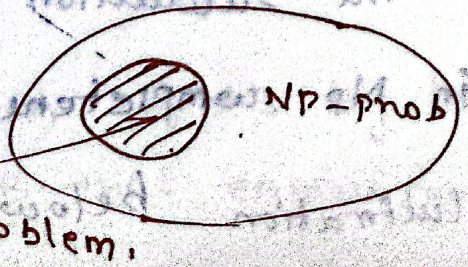
→ NP-complete problem are NP-hard but all NP-hard problems can not be NP-complete.

NP-Completeness:

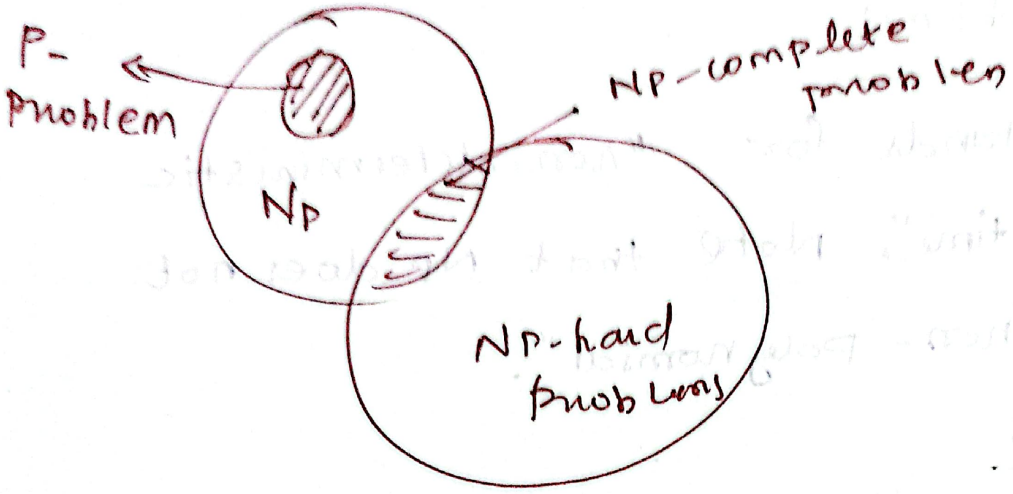
P ⊆ NP. (Non-deterministic polynomial language)

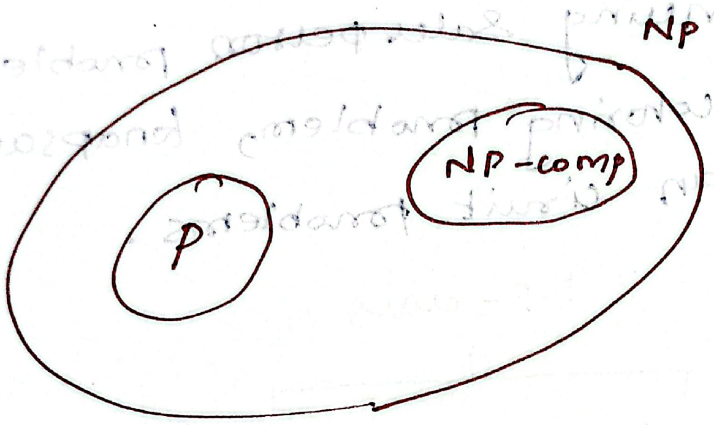
P = NP

P = NP (or) P ≠ NP.



Relation ship b/w P, NP, NP-complete & NP-hard problems,



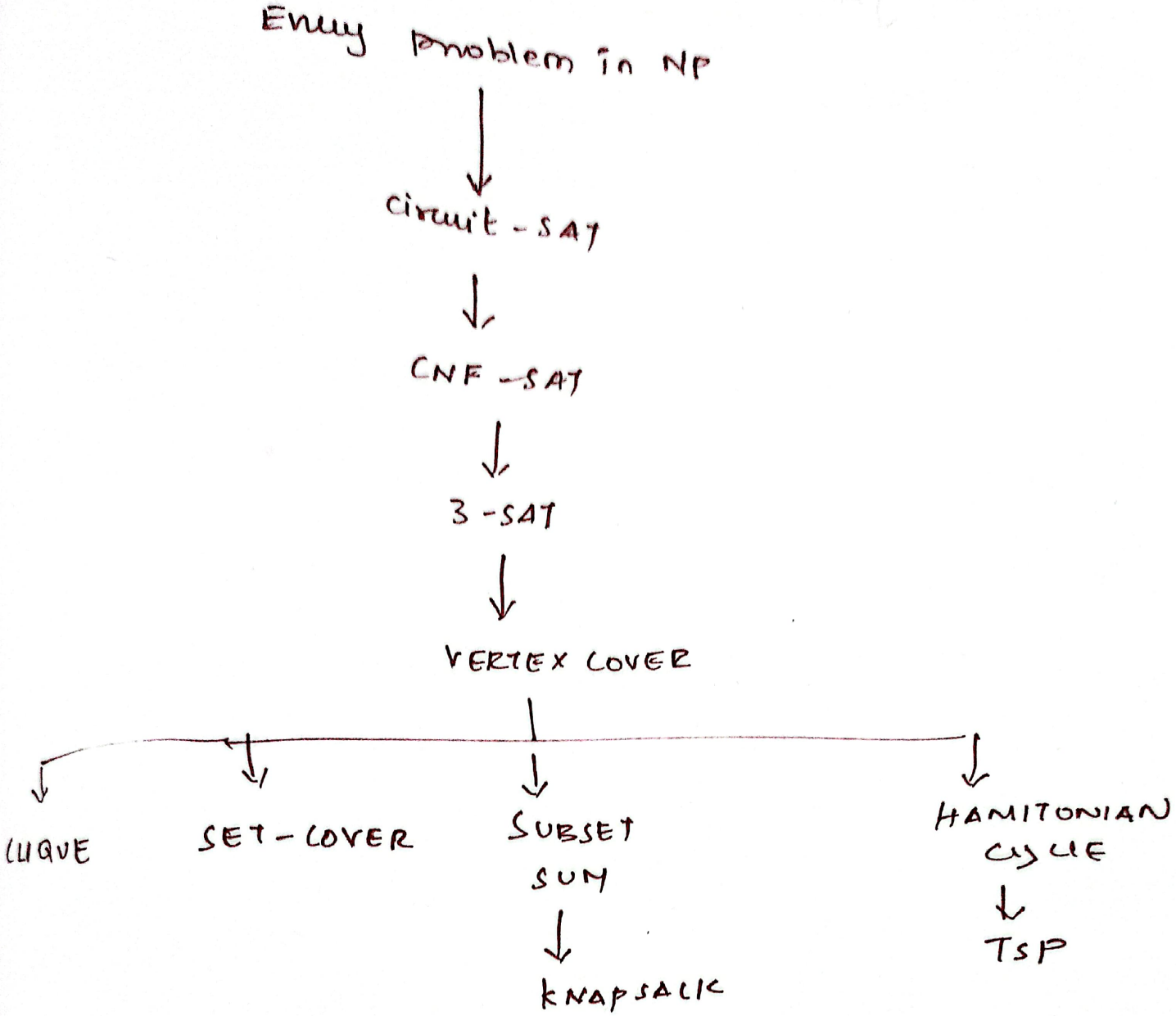


Polynomial time Reductions! (2) (8m.)

- To prove whether particular problem in NP complete or not we use polynomial time reducibility. That means,

$$A \xrightarrow{Poly} B \text{ & } B \xrightarrow{Poly} C \text{ (& then) } A \xrightarrow{Poly} C$$

- The reduction is an important task in NP-completeness proof. This can be illustration below,



Various types of reduction are;

→ Local replacement

→ component design

→ Local replacement:

- In the reduction A → B by dividing input to A in the form of components and then these components can be converted to components of B.

component design:

- In this reduction A → B by building special component for input B that enforce properties

FSA - outputs

→ FA may have o/p's corresponding to each transition. There are 2 types of Finite state machines that generates o/p

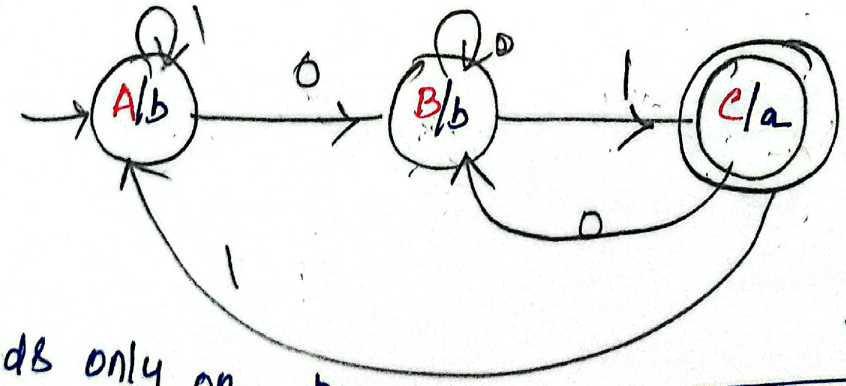
i) Moore Machine $(Q, \Sigma, O, \delta, \lambda, q_0)$ — O: o/p alphabet

ii) Mealy Machine $(Q, \Sigma, O, \delta, \lambda, q_0)$ — λ: o/p trans fn

(1) Construct a Moore M/c that prints 'a' whenever the sequence '01' is encountered in any i/p binary string & then construct to its equivalent Mealy Machine.

$\Sigma = \{0, 1\}$ $\quad \Delta = \{a, b\}$

Moore



State	0	1	o/p
→A	B	A	b
B	B	C	b
C	B	A	a

Finite write the o/p column

Moore

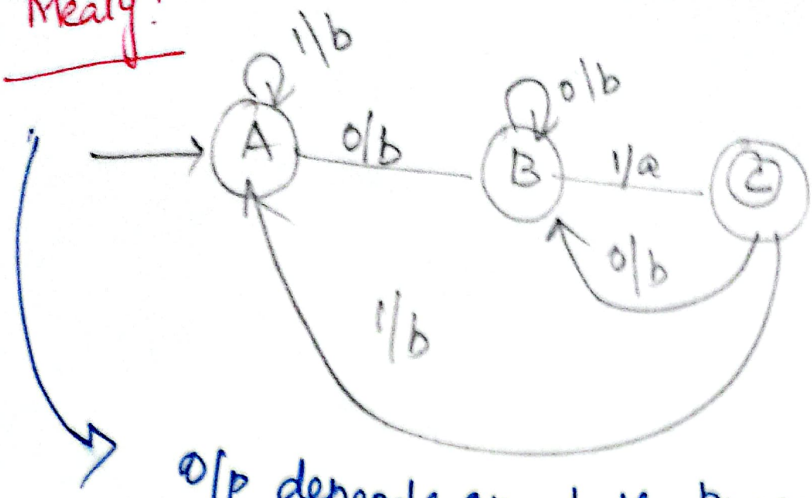
o/p depends only on present state.

Mealy M/c

	State	0	1
→	A	B, b	A, b
	B	B, b	C, a
*	C	B, b	A, b

Write o/p state then

Mealy:



State	0	1
→ A	B, b	A, b
B	B, b	C, a
C	B, b	A, b

o/p depends on both present State & present i/p.

---

2) Convert the gn Mealy M/c to its equivalent Moore m/c. [using Transition table].

State	a	b
→ q_0	q_3, 0	q_1, 1
q_1	q_0, 1	q_3, 0
q_2	q_2, 1	q_2, 0
q_3	q_1, 0	q_0, 1

Initially, start with q_0, chk with the table q_0 is available

$q_0 \rightarrow 1$

q_1 — $q_{10} \rightarrow 0$, $q_{11} \rightarrow 1$

q_2 — $q_{20} \rightarrow 0$, $q_{21} \rightarrow 1$

$q_3 \rightarrow 0$

	State	a	b	o/p
→	q_0	q_3	q_{11}	1
	q_{10}	q_0	q_3	0
	q_{11}	q_0	q_3	1
	q_{20}	q_{21}	q_{20}	0
	q_{21}	q_{21}	q_{20}	1
	q_3	q_{10}	q_{10}	0