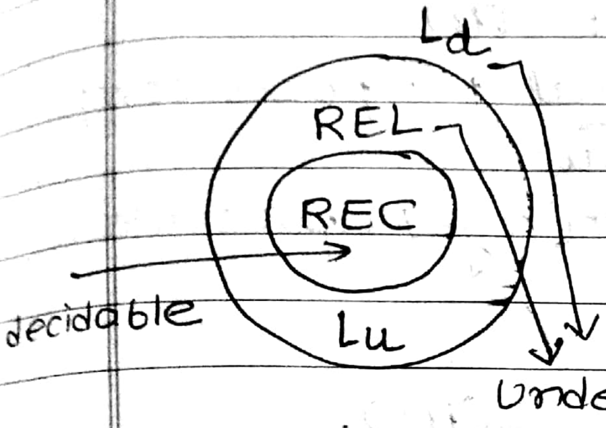
Undecidability and computational classes



REC : Recursive language
REL : Recursive Enumerable Language
Ld : Diagonlization Language

- **REC (Recursive Language)**

↳ ∃ TM which always halts
↳ such TM is called TTM (Total Turing m/c)
↳ It is called decider.
↳ eg. $\{a^n b^n c^n \mid n \geq 1\}$

- **REL (Recursive Enumerable language)**

↳ $w \in$ REL — TM always halts and accepts
↳ $w \notin$ REL — TM halts and Rejects
or
— TM goes to infinite loop.
↳ such language is also called universal language (Lu) which is a language accepted by universal Turing machine (UTM).

* Universal Turing machine (UTM)

- It is a special kind of TM.
- behaves like a general purpose computer
- UTM takes two input
  - (1) TM code M (binary Representation of TM)
  - (2) w as Input
- It accepts w iff $w \in L(M)$.

Input to UTM

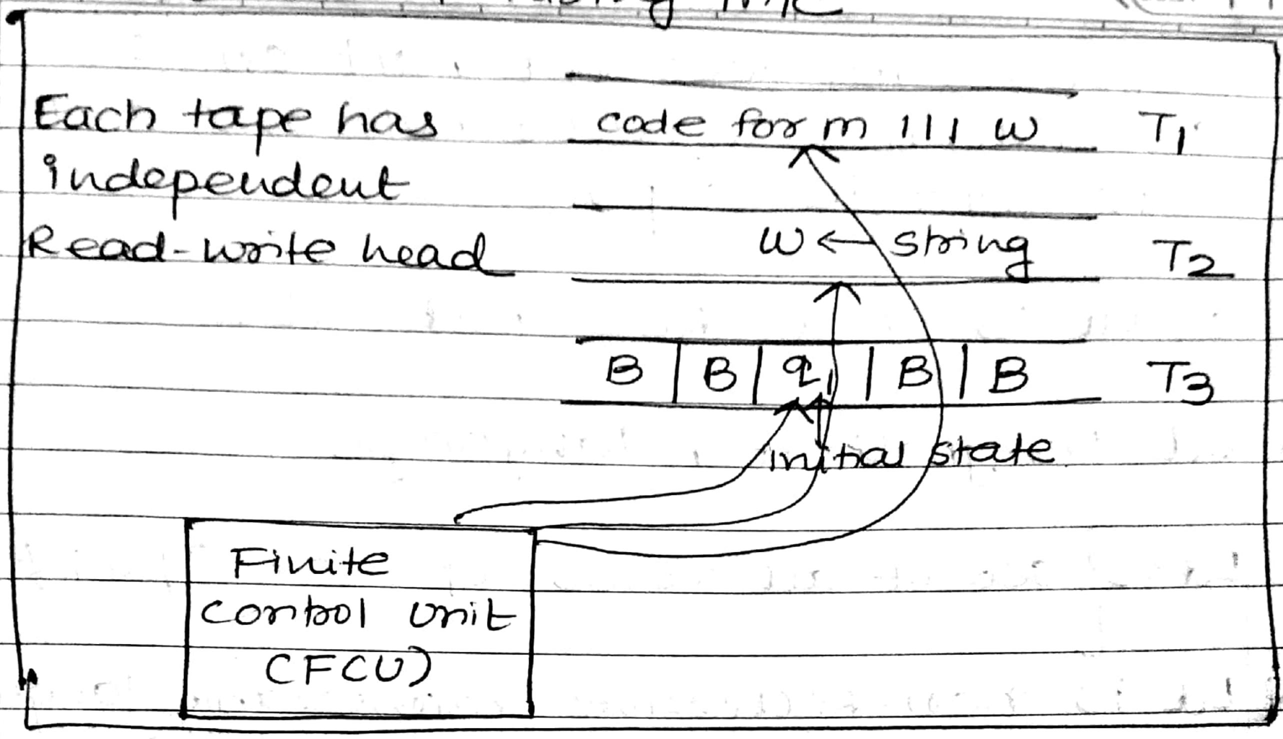
$\underbrace{\text{code for M}}_{\text{binary Representation of M}} \ \underbrace{111}_{\text{separator}} \ \underbrace{w}_{w \in (0+1)^*}$

$L_u = L(U)$ U : UTM

L_u: Universal language → which is a language accepted by UTM

- UTM has 3 tapes.
  - T_1 : consists of two part
    - (1) code for M
    - (2) w
  - T_2 : used to simulate the tape of M
  - T_3 : current state of M during simulation

Universal Turing m/c



Working of UTM

1. check if m is valid or Not.
 (validity means when TM is mapped to binary representation, it starts with 0).
 (so if m starts with 0, it is valid otherwise it is invalid).
 If m is Invalid,
 $L(m) = \phi$
 $\Rightarrow w \notin L(m)$
 $\Rightarrow$ UTM halts and Rejects
 else go to step 2.
2. If m is valid
 copy w from T_1 to T_2.
 (Keeps scanning T_1 up to 111 - After that copy w from T_1 to T_2).
3. Initialize T_3 with an initial state of M
 suppose q_1 is initial state.
4. simulate TM using T_1, T_2 and T_3.
5. If m accepts w, UTM accepts input.

suppose, m does not halt on ω.
(simulation may never halt and stuck into infinite loop)

∴ Lu is Not REC but REL.

- Ld : Diagnolization language

$L_d = \{$ set of all words $w_i \in (0+1)^* \mid w_i \notin L(M_i) \}$

↳ Ld is non-Recursive Enumerable language.
↳ we can't write computer prog. for such language.
↳ Ld is the first language which is non REL.

Problem

Decidable
↳ prob. is said to be decidable if TM exists to solve that prob. which always halts.

Undecidable
↳ prob is said to be undecidable if
(1) TM does not exist to solve the problem
(2) if TM exists, It halts or goes to infinite loop.

Page No.
Date: / /

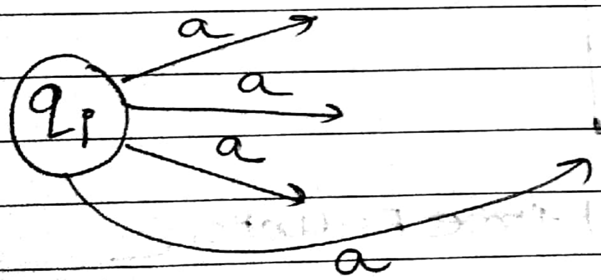
- **Class P :-** P stands for polynomial

↳ ∃DTM which always halts } TOC
DTM: Deterministic TM } context
⇔ ∃DTTM: Deterministic Total TM
⇔ ∃ Deterministic Algo. } Algo context

I/p of n length ⟶ $\boxed{\text{DTTM}}$ ⟶ o/p is generated in $O(T(n))$ steps/moves
where $T(n)$ = polynomial in n.

- **Class NP :** Non-Deterministic Polynomial

↳ ∃NTM which always halts
NTM: Non-Deterministic TM
⇔ ∃NTTM – Non-Deterministic Total TM
⇔ ∃ Non-Deterministic Algo.



↳ Multiple paths are available.
↳ longest path length: $O(T(n))$
where $T(n)$ is polynomial.

↳ In NTTM, all paths are executing parallely.

DTM: single threaded computer prog. which always halts. single thread means there is single path.

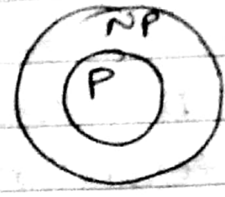
NTM: Multiple threaded prog. which always halts.

Scanned by CamScanner

open challenge.

$P = NP$ $\qquad\qquad$ $P \neq NP$

$P=NP$



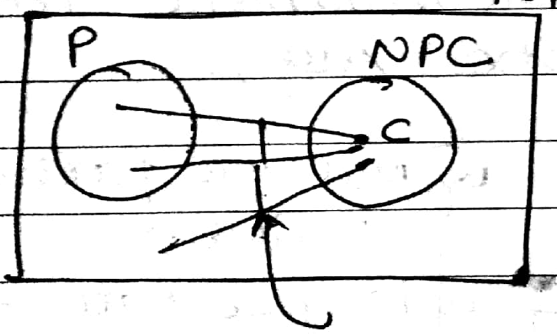
↳ Thousands of prob in NP for which no one could come up with the Deterministic polynomial time Algo.

↳ For that prob (NP), Not able to construct single thread algo.

- NPC (NP- Complete problem)

NP

$P \neq NP$



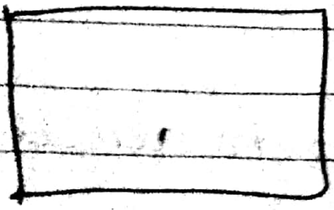
Polynomial time Reduction

A problem C is in NPC if

(1) $C \in NP$

(2) Every prob in NP is Reducible to C in polynomial time (including P)

$P=NP$ $\qquad$ $P=NP=NPC$



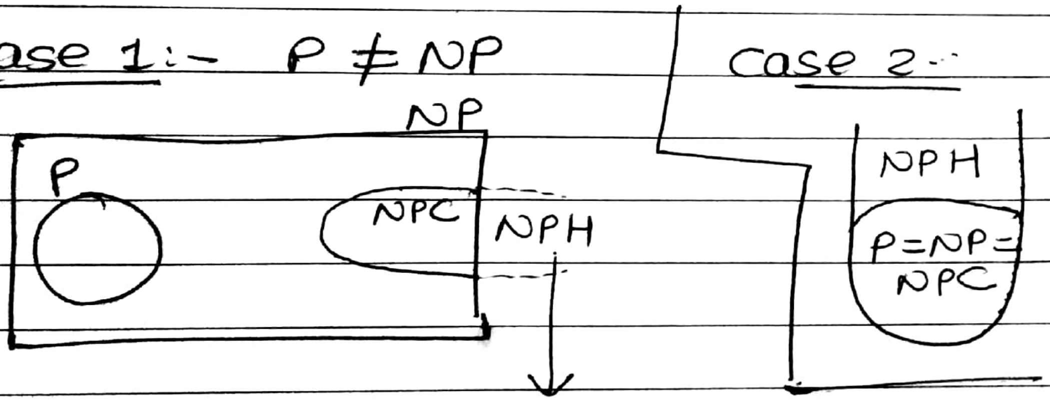
- In NPC, we don't have a polynomial time Deterministic Algo available.
- We do have exponential time algo $O(2^n)$, $O(n^n)$, $O(n!)$ ...

- **NP- Hard**

- X is NPH if
  $\exists$ a polynomial time Deterministic Reduction from every prob. in NP to X.

Case 1:- $P \neq NP$

Case 2:- $P = NP$



- May or May not lie to NP.
- If it lies in NP,

$NPC \subset NPH$

$$\boxed{NPH + NP = NPC}$$

Example of NPC
- clique problem
- Graph coloring
- vertex cover
- Longest common subsequence

Example of NPH
- subset problem
- TSP
- Halting prob.
- Boolean satisfiable