Unit-V

Undecidability:

⇒ A Language that is Not Recursively Enumerable

⇒ An undecidable problem that is RE

⇒ Undecidable problems about TM's

⇒ Post Correspondence problem

⇒ Other Undecidable problems

Intractable Problems

⇒ Polynomial time and space

⇒ Some NP- Complete problems

⇒ The classes P and NP

⇒ An NP Complete problem

recursively enumerable lang → TM

recursive language → TTM

TM accepts a language that lang is called recursive enumerable lang

⇒ Those languages are accepted by TM is REL

Recursive language:

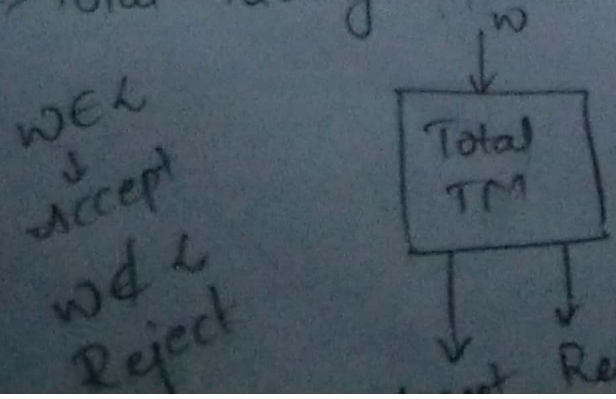
Total turing machine

TM have
- Accept
- Reject
- Loop

When we design a TM for language L then
- i.e $w \in L$ → Accept
- $w \notin L$ → Reject (or) Loop then

When we design a TM for a language L, it accepts all the words (or) strings belongs to the language.

⇒ Total turing machine have
- Accept
- Reject



$w \in L$ → Accept

$w \notin L$ → Reject

TTM are proper subset of TM

RE are " " of REL

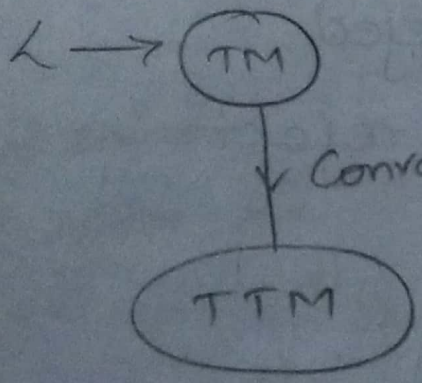
> L is called REL if there is a TM
> L is called Recursive if there is a TM

TTM:

In TTM: ~~wrong~~ right input → Accept
Wrong input → Reject

TM:

right i/p → Accept
Wrong i/p → Reject (or) Loop



Conversion from TM to TTM is not possible

(L) → No TM available

L → TTM also not available

Decidability and Undecidability:-

Decidability:-

A problem is decidable if we can construct a TM which will in finite amount of time for every i/p and give ans yes (or) no

Decidable

⇒ Does FA accept R<

⇒ L_1 and L_2 are 2 R< closed und

Union
Concatenation
Kleen-closure

⇒ L_1 & L_2 ⇒ CFL

$L_1 \cup L_2$

Undecidable

For given CFG is

$L(G_1)$ is ambiguous or not

L_1 & L_2 are CFL

$L_1 \cap L_2$ are CFL & not

Recursive Language:

⇒ A Language "L" is said to recursive if there exists a Turing Machine which will accept all the strings in "L" and reject all the strings not in "L".

⇒ The Turing machine will halt every time and give an answer (accepted or rejected) for each and every string input.

Recursively Enumerable Language:

⇒ A Language "L" is said to be a recursively Enumerable Language if there exists a Turing machine which will accept (and therfore halt) for all the input strings which are in "L".

⇒ But may or may not halt for all input string which are not in "L".

Decidable languages:

A Language "L" is decidable if it is a recursive language.

All decidable languages are recursive languages and vice-versa.

Partially decidable language:

⇒ A Language "L" is partially decidable if 'L' is a recursively enumerable language.

Undecidable Language:

⇒ A Language is undecidable if it is not decidable.

⇒ An undecidable language may sometimes be partially decidable but not decidable.

⇒ If a Language is not even partially decidable, then there exists no Turing machine for that language.

The Halting problem:-

08-07-19

Given a program, will it halt?

↓

Accept & Reject

⇒ Given a TM, will it halt when run on some particular given input string?

for this no algorithm is designed, to know whether the string will halts & not.

⇒ Ex: Given some program written in some language (Java/c etc..) will it ever get into a infinite loop (or) will it always terminate?

Answer:

⇒ In general we can't always know.

⇒ The best way we can do is run the program and see wheather it halts

⇒ for many programs we can see that it will always halt (or) sometimes loop

~~But for problems~~ in general

But for programs in general the Question is undecidable.

Undecidability of the Halting problem?

Given a program, will it halt?

=> Can we design a machine which if given a program can find out if that (or) decide if that program will always halt or not halt on a particular input?

We assume that, we are going to design a such machine

H(Program, Input)

- Halt
- Not halt

H(P, I)

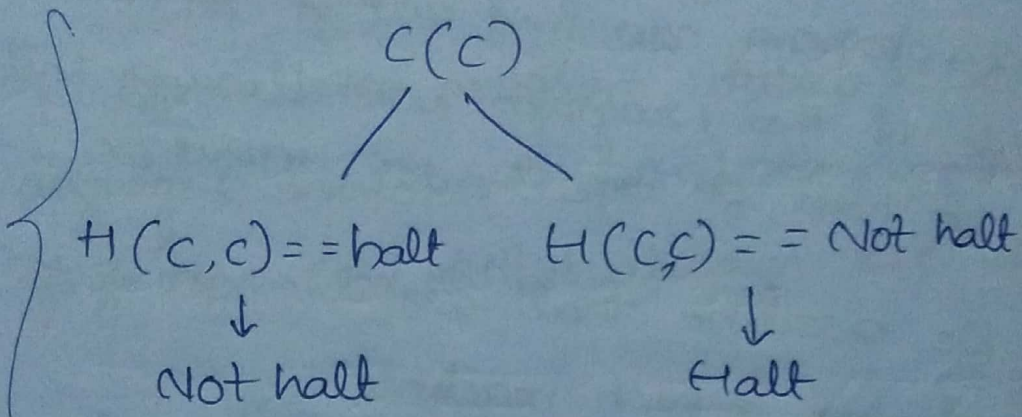
C(x) — any program

```
if { H(x x) == halt }
    loop forever;
else
    return;
```

⇒ If the program halts, it will goes to execute if case.

⇒ If the program does not halt, it execute else case.

⇒ What will happen, if we run 'C' on itself.



↳ This contradiction is not correct.

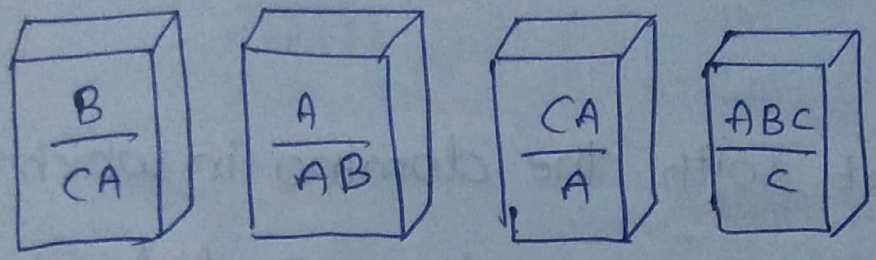
↳ Undecidable problem will not say the turing machine will halt or not

The Post Correspondence Problem:-

⇒ The post corresponce problem is an undecidable decision problem that was introduced by Emil Post in 1946.

Dominos:
(&) tiles

⇒ Dominos contains 2 parts
- Enumerator (top)
- denaminator (bottom)



⇒ We need to find a sequence of dominos such that the top and bottom strings are the same.

⇒ if we combine all the top symbols and then the symbols at the bottom

⇒ top symbols combination makes as string and same bottom symbols makes string

top string = Bottom string

⇒ They should form the same string

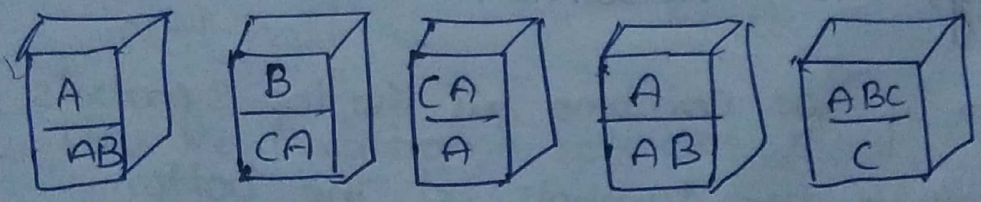
Rule:

⇒ We can arrange Dominos in which ever way you want. (finite sequence)

⇒ Dominos can be used any number of times.

Step 1:

Start with the domino in which the first symbol on the top and first symbol on the bottom are same. in order to form a string



A B CA A ABC

AB CA A AB C

→ ABCAAABC =ABCAAABC

another way of represting the PCP,

	A (top)	B (bottom)	
①	1	111	$\frac{1}{111}$
②	10111	10	$\frac{10111}{10}$
③	10	0	$\frac{10}{0}$

Step 1: top & bottom first string is same

2 cases are exist

① & ② dominos are same

	②	①	1	①
A:	10111	1	1	10
B:	10	111	111	0

⇒

	A	B	
①	10	101	10/101
②	011	11	011/11
③	101	011	101/011

① Possibility

Step 1: $\frac{10}{101} \Big| \frac{101}{011}$

$\frac{10|10\,1|10}{101|01|1\,101}$

2nd possib:

$\frac{10\,101}{10,1011}$;

3rd possib

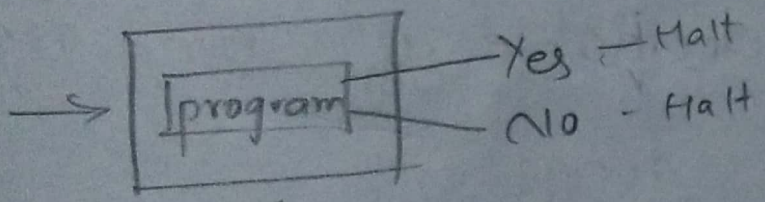
$\frac{10\,10}{101\,101}$ ✗

$\frac{10\,(0110110)1}{1010110110}$ ... ✗

Undecidable

Halting Problem:-

Question: Is it possible to tell whether a given machine will halt for some given input.



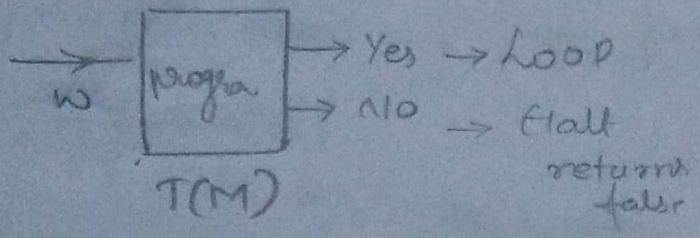
Before ~~running program~~

⇒ Have you ever asked a computer

⇒ Before running program this program will halt or not halt

Consider a machine, will halt at particular input 'w'

"will halt (M, w)" => exist.



Loop Halt (M, w)

{

if will halt (M, w) then

{

while (true);

}

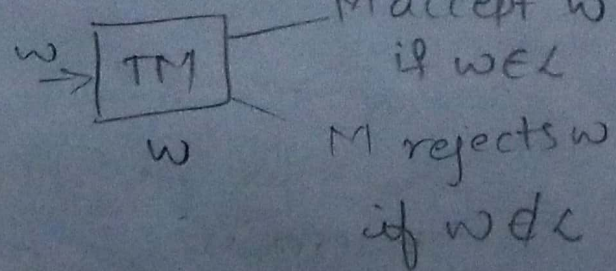
else

return false;

Contradiction

Decidable language (R.L)

A lang L is decidable if some TM decide



Tractiability and Intractability

Tractable problem

⇒ A problem is called tractable iff there is an effecient algorithm that solves it in polynomial time.

n^k → k is constant

i.e $O(n), O(n^2)$...

$O(n \times \log n)$

⇒ Solve in P Time (polyn

→ Big-O = $O(n)$

Intractable problem

⇒ There is no effici algorithm to solve it

(or)

A problem that can't be solved by polynomial time algorithm

i.e. $\{ 2^n \ldots n! \ldots \}$

↓

exponential ti

Tower of Hanoi

→ $2^n - 1 \rightarrow O(2^n)$

→ Solved in ∈-Time

(exponential time

P	NP
Deterministic polynomial time	Non-deterministic polynom time.
⇒ This class includes those languages which are recognized by	⇒ The class NP Consists all decision problems where 'Yes' can be ver

some Deterministic TM.

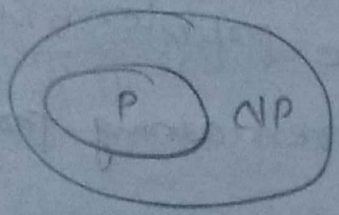
other define:

-> Solves in P-Time

-> odd & not

Ex: Sudoku, chess verified → certifies

⇒ This class includes those languages which are recognized by Non-Deterministic TM



⇒ ~~Any~~ Any problem can be solved in 'P' time using DTM, that problem also be solved in NP-time

NP-Hard and NP-Complete:

Polynomial time

Linear search - n

Binary search - $\log n$

Insertion sort - n^2

Merge sort - $n \log n$

Matrix multiplication - n^3

This alg will take polynomial time

Exponential time

0/1 knapsack - 2^n

Traveling SP - 2^n

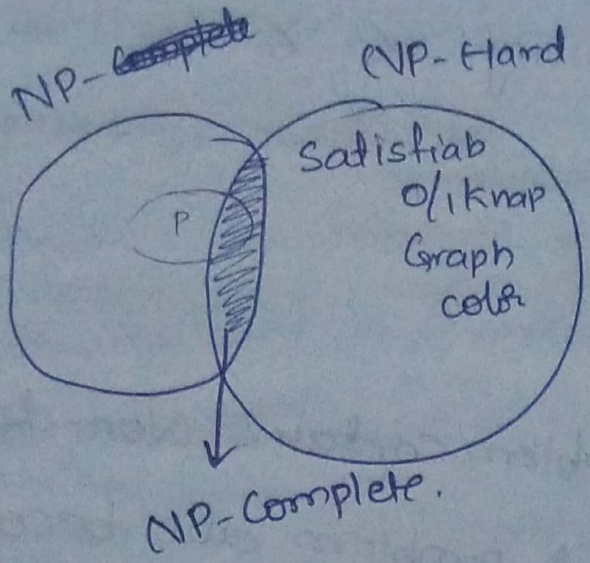
Sum of subsets - 2^n

Graph Coloring - 2^n

Hamiltonian cycle - 2^n

Kruskal's Algorithm

This alg will take exponential time



⇒ Intersection of NP-Hard & NP is NP-Complete.

⇒ If satisfiability in P, then P = NP by using Cook's theorem

⇒

Class of problem (algorithms)

P NP

⇒ Problems solved in polynomial time
ex: searching & sorting
O(n)
O(n²)
O(n log n)

problem cannot be solved can be verified in polynomial time
Ex: Suduko
pin matching
Here exponential time

⇒

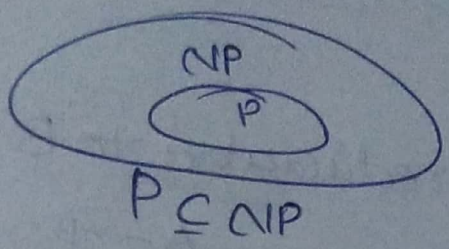
⇒

P- class

Easy to solve
&
Easy to verify
in
polynomial time

NP- class

Hard to solve
&
Easy to verify
in
Exponential time



is P = NP is a Hypothetical question.

⇒ if P = NP

⇒ all problems can be solved in less time

⇒ all security algorithm fails

⇒ Computation reaches to peaks

⇒ if P ≠ NP

⇒ there are some problems which will be never solved.

Every decision problem that is solvable by a deterministic polynomial time alg is also solvable by a polynomial time Non-Deter alg

Reduction:

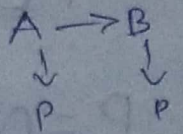
problem A reduces to Problem B iff there is a way to solve A by deterministic alg that solve B in Poly time

Properties:

1. if A is reducible to B &

   B is in P

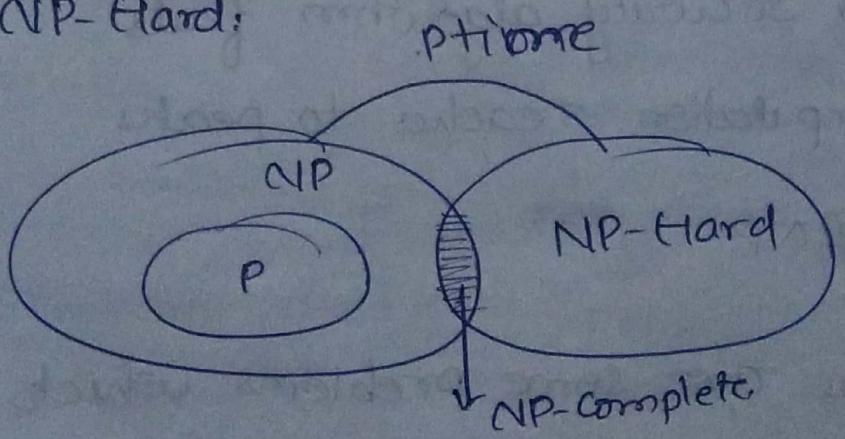
   then A is in P



2. if A is not in P ⇒ B is not in P

$\neg B \rightarrow \neg A$

⇒ NP-Hard:



⇒ i will reduce all NP-problems to another set in Polynomial time

⇒ The problem which is present in NP & NP-Hard is called NP-Comp

⇒ NP-Hard is called as Optimization Problem

⇒ NP-Complete is called as Decision Problem.

⇒ Type-1 Grammar:

Context-Sensitive Grammar
(CS)
length increasing Grammar
(CS)
Non-Contracting Grammar

⇒ It generates context sensitive lang which will be accepted by Linear bounded Automata (LBA).

$$\alpha \longrightarrow \beta$$

where $\alpha \beta \in (V \cup T)^*$

and $|\alpha| \leq |\beta|$

$\beta \in (V \cup T)^+$

Mean $A \rightarrow \epsilon$ is not allowed in Type 1 gram

⇒ where α and β are strings of Non-terminals & terminals.

⇒ Context sensitive grammars are more powerful than CFG because there are some languages are described by CSG but not by CFG.

⇒ CSL are less powerful than Unrestricted grammar.