UNIT-V UNDECIDABILITY

Recursive & Recursively Enumerable Languages:

- Algorithm: - After some finite no. of steps, always halts & gives you the answer.

Procedure: May or may not halt.

Ex: Given 'n', is n prime? → Algo
Given 'n', is n a perfect number? → Algo
Given 'n', is there a perfect number > n.? → Procedure (running 1/more times)

Recursive Language: - A language L is said to be recursive, if there is an algorithm to determine whether the given string w belongs to L or not.

Recursively Enumerable Language: - A language L is said to be recursively enumerable if there is a procedure to determine whether $w \in L$ or not.

→ Consider TM M & string w. There are 3 possible outcomes
1. TM halt & accept the string.
2. TM halt & reject the string.
3. TM never halts (runs forever)

Recursive Language: - A language L is said to be recursive if there exists a TM M such that ∀ strings w.
(i) $w \in L$, the TM halts & accepts the string.
(ii) $w \notin L$, the TM halts & rejects the string.

Recursive languages are called Turing decidable languages.

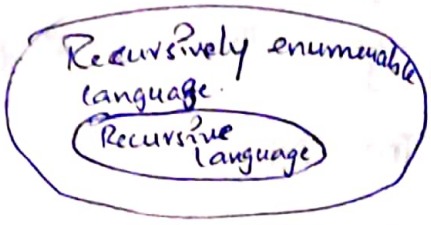
Recursively Enumerable Language: - A language L is said to be recursively enumerable if there exists a TM M such that ∀ strings w,
(i) If $w \in L$, the TM halts & accepts the string.
(ii) If $w \notin L$, the TM either halts & rejects the string

Scanned by CamScanner

or never halts.

→ Recursively enumerable languages are called Turing-Recognizable language.

→ Recursive lang ⊂ Recursively enumerable language

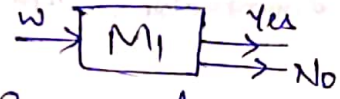


Properties of Recursive & Recursively Enumerable languages:

1. Union of 2 Recursive languages is recursive.

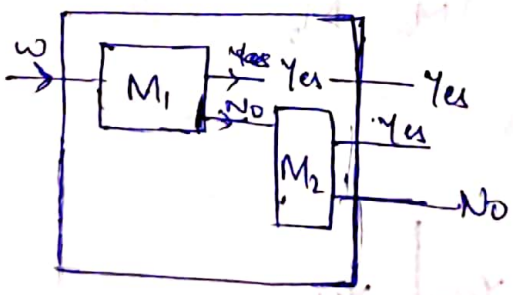
2. If L_1 & L_2 2 recursive languages, then, $L_1 \cup L_2$ is also recursive.

Proof: L_1 is accepted by TM M_1.



L_2 is accepted by a TM M_2:

M:

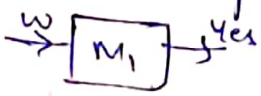


M accepts $L_1 \cup L_2$.

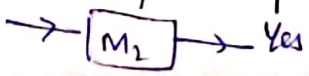
2. Union of 2 Recursively enumerable languages is recursively enumerable.

If L_1 & L_2 are 2 recursively enumerable languages, then $L_1 \cup L_2$ is also recursively enumerable.

Proof: L_1 is accepted by TM M_1

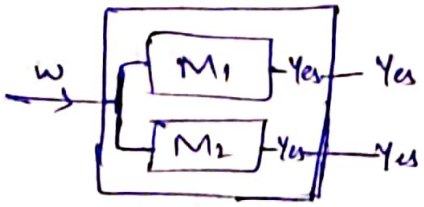


L_2 is accepted by TM M_2.



Scanned by CamScanner

M :

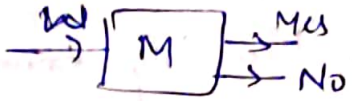


3. Complement of a Recursive language is Recursive.

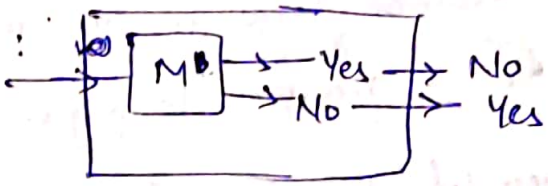
- If L is a recursive language, then so is $\bar{L}$.

Proof : L is accepted by a TM M.

M :

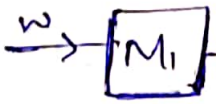


$\bar{L}$: $\bar{M}$:

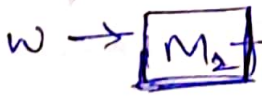


4. If L and $\bar{L}$ are recursively enumerable, then, L is recursive.

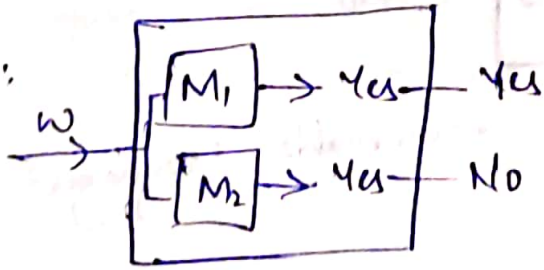
Proof : L is accepted by TM M_1.



$\bar{L}$ is accepted by TM M_2:



M :



Church - Turing Hypothesis :

- Any algorithm process can be simulated efficiently by a Turing Machine.

- Any process which could be called an effective procedure can be realised by TM.

Scanned by CamScanner

- We can model any mechanical computer with a TM.
- There is some TM corresponding to every computable problems.
- If there is no TM that decides problems P, then there is no algorithm that solves problems P.

Language that is not recursively Enumerable:
- Encoding of TM - Diagonalization language (L_d).

Encoding of TM:

- A TM can be encoded as a binary string.

To represent a TM $M=(Q, \Sigma, \Gamma, \delta, q_0, \text{b}, F)$ as a binary string, we assign integers to
1. States 2. Tape Symbols 3. Directions.

$Q = \{q_1, q_2, \ldots q_n\}$

Assume, Start state = q_1 & Final State = q_n.

$\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \text{b}\}$. We shall assume, the tape symbols as X_1, X_2, X_3.

$\therefore X_1 \leftrightarrow 0$, $X_2 \leftrightarrow 1$, $X_3 \leftrightarrow$ b

$D = \{L, R\}$ $\therefore$ For $L \leftrightarrow D_1$ & $R \leftrightarrow D_2$

$\delta(q_i, x_j) = (q_k, x_l, D_m)$

This rule can be encoded as binary string.

$0^i 1 0^j 1 0^k 1 0^l 1 0^m$; 1 → separator

Code for entire Turing Machine: is

$C_1 11 C_2 11 C_3 11 \ldots 11 C_{n-1} 11 C_n$; 11 → seperator. &

$C_1, C_2, C_3, \ldots C_n$ are binary code for transition rules. i.e C_i is code for transition rule.

Scanned by CamScanner

→ TM $M = (\{q_1, q_2, q_3, q_4, q_5, q_6\}, \{0,1\}, \{0,1,b\}, \delta, q, b, \{q_2\})$

$\delta(q_1, b) = (q_6, b, R)$: $0^1 1 0^3 1 0^6 1 0^3 1 0^2$

$\delta(q_6, 1) = (q_6, 1, R)$: $0^6 1 0^2 1 0^6 1 0^2 1 0^2$

$\delta(q_6, b) = (q_3, b, L)$: $0^6 1 0^3 1 0^3 1 0^3 1 0^1$

$\delta(q_3, 1) = (q_4, b, L)$: $0^3 1 0^2 1 0^4 1 0^3 1 0^1$

$\delta(q_4, 1) = (q_3, b, L)$: $0^4 1 0^2 1 0^3 1 0^3 1 0^1$

$\delta(q_4, b) = (q_5, b, R)$: $0^4 1 0^3 1 0^5 1 0^3 1 0^2$.

$\delta(q_5, b) = (q_2, 1, L)$: $0^5 1 0^3 1 0^2 1 0^2 1 0^1$

Left = 1
Right = 2
0 = 1
1 = 2
b = 3

∴ Code for entire TM:

$0^1 1 0^3 1 0^6 1 0^3 1 0^2 \, 11 \, 0^6 1 0^2 1 0^6 1 0^2 1 0^2 \, 11 \, 0^6 1 0^3 1 0^3 1 0^3 1 0^1 \, 11 \, 0^3 1 0^2 1 0^4 1 0^3 1 0^1 \, 11 \, 0^4 1 0^2 1 0^3 1 0^3 1 0^1 \, 11 \, 0^4 1 0^3 1 0^5 1 0^3 1 0^2 \, 11 \, 0^5 1 0^3 1 0^2 1 0^2 1 0^1$

(State, Tape symbol, State, Tape symbol, Direction — Transition Rule)

= 0100010000001000100 11 0000001001000000100100 11 000000
1000100010001011 000100100001000101 1 0000100100010001
011 0000100010000010001001 1 0000010001001 0010.

→ (A valid binary code in TM starts with 0).

→ Every TM can be encoded as binary string.

→ If M is a TM & w is an input string then we shall consider a code pair, ⟨M, w⟩

⟨M, w⟩ = Code for TM 111 w

Diagonalization Language (L_d):

M_i is the i^{th} T machine whose binary code is w_i

If w_i is not a valid code then we assume that M_i only has 1 state & no transitions.

$L_d(M_i) = \phi$

Consider an infinite Boolean Matrix:

D ↓ / s →	w_1	w_2	w_3	w_4	w_5 ...
M_1	1	0	1	1	0 ...
M_2	1	0	0	1	0 ...
M_3	0	1	1	0	1 ...
M_4	0	1	0	1	1 ...
M_5	0	0	0	0	1 ...
⋮					

$(i,j)^{th}$ entry will be '1', if i^{th} TM accepts the j^{th} string.
Otherwise, '0'.
From diagonal elements,
M_1 accepts w_1 i.e., M_2 doesn't accept w_2.
∴ Diagonalization Matrix is the strings which correspond to zero is diagonal. i.e

$$L_d = \{w_i \mid w_i \text{ is not accepted by } M_i\}$$

Taking all the strings in diagonal which correspond to 1 is.

$$\overline{L_d} = \{w_i \mid w_i \text{ is accepted by } M_i\}$$

Theorem: L_d is not recursively enumerable. That is, there's no TM that accepts L_d.
Proof: Suppose L_d is recursively enumerable, i.e;
L_d is accepted by M_i
Now, check whether w_i is in L_d.
Case-1: If w_i is in L_d.
∴ w_i accepted by M_i.
But by definition of L_d, w_i is not in L_d.
∴ By proof of Contradiction, we can say that L_d is not recursively enumerable by case-1.

Scanned by CamScanner

Case-2: If w_i is not in L_d. then w_i isn't accepted by M_i.

But by definit°, if w_i isn't accepted by M_i, it means $w_i \in L_d$.

But ∴ there is contradiction. in our assumption.

∴ There is no TM that accepts L_d.

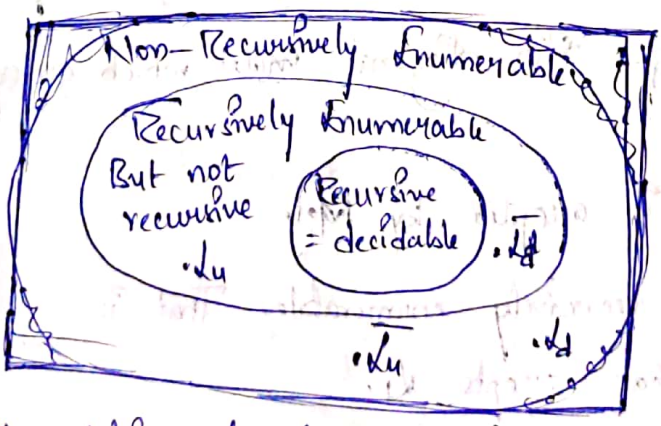
∴ L_d is not recursively enumerable.

An Undecidable Problem that is Recursively Enumerable:

A language 'L' is called decidable if it is recursive, L is called Undecidable if it isn't recursive.

We have 3 classes of languages.

1. Recursive Lang.
2. Languages that are Recursively enumerable but not recursive.
3. Non-recursively enumerable languages. (L_d)



There's no TM that accepts L_d & $\overline{L_u}$ (Complement of Universal language).

The Universal Language (L_u):

Contains a code pair such that M accepts w.

$L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\}$ where,

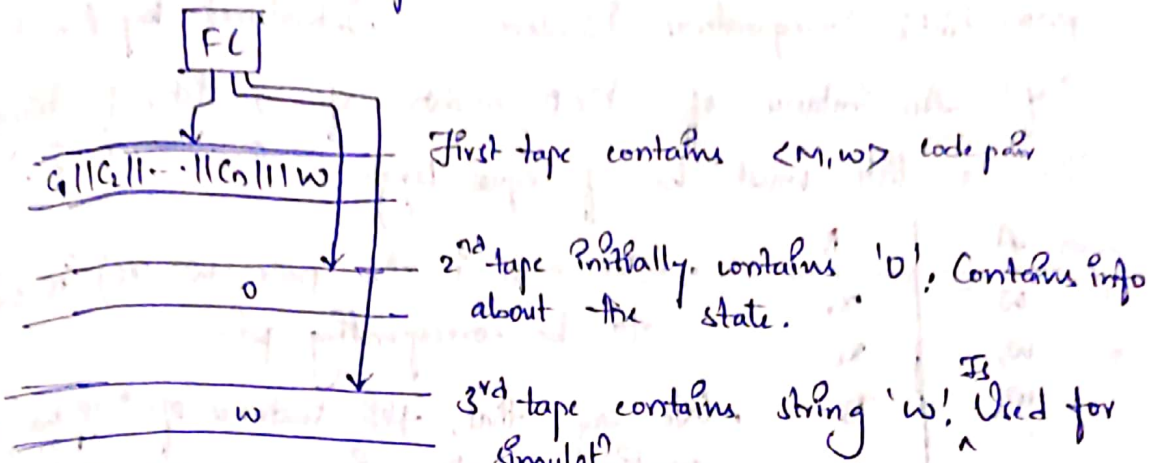
M is TM & w is a string & $\langle M, w\rangle$ is a binary code pair.

$\overline{L_u} = \{\langle M, w\rangle \mid M \text{ doesn't accept } w\}$

Scanned by CamScanner

L_u is accepted by a TM 'U' called the Universal TM.
Universal TM is a TM, that can simulate any other TM with including itself.

TM U has 3 tapes.



'0' represents the start state. (in 2nd tape).

Suppose after sometimes, let 2nd tape has 0000 & FC is pointing to '1' in 3rd tape.

∴ That means we're ~~reading~~ in q_4 & are reading 1.
$= (00)$

∴ It'll check whether the block starts with $0^4 1 0^2 \ldots$

Undecidability of the Universal Language:

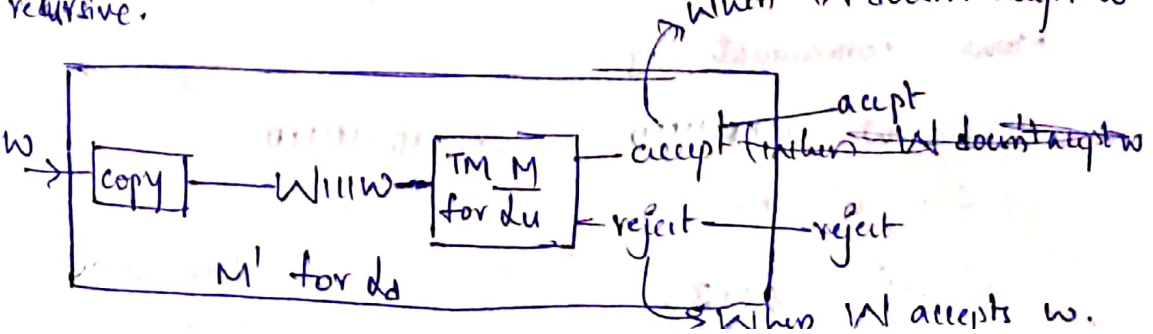
L_u is not recursive.

Suppose L_u is recursive, then $\overline{L_u}$ is also recursive.

If $\overline{L_u}$ is recursive, then L_d is also recursive.

But WKT, L_d is not recursively enumerable.

∴ There's a contradict of our assumption that L_u is recursive.



Scanned by CamScanner

Whether L_d depends on L_u

$\therefore$ as L_d is not accepted by TM, L_u also isn't accepted.

PCP & MPCP:

PCP: Post's Correspondence Problem. — Introduced by Emil Post.

Def^n: An instance of PCP consists of 2 lists of strings over some Σ.

The 2 lists must be of equal length.

Indx	A	B
1	w_1	x_1
2	w_2	x_2
3	w_3	x_3
⋮	⋮	⋮
k	w_k	x_k

- For each i, the pair (w_i, x_i) is said to be corresponding pair.
- We say that, this instance of PCP has a solution if there is a sequence of 1 or more integers, of the form $i_1, i_2, \ldots i_m$ $(m \geq 1)$ such that $w_{i_1} w_{i_2} w_{i_3} \cdots w_{i_m} = x_{i_1} x_{i_2} \cdots x_{i_m}$

The solution is $i_1 i_2 \cdots i_m$

i	A: w_i	B: x_i
1	1	111
2	10111	10
3	10	0

$i = 2$

w: 10111

x: 10

We can't concatenate with 2 as then, w = 1011110111 ; x = 1010 — can't be there.

$\therefore$ Can concatenate with 1.

then, w = 101111, x = 10111

$\therefore$ Sol^n = 21

We can't concatenate with 213. $\therefore$ $i = 1$.

211 w = 1011111 x = 10111111

Now concatenate 3.

$\therefore$ w = 10111111 0 x = 10 1111110

$\therefore$ $w = x$.

$\therefore$ Sol^n = 2113.

Scanned by CamScanner

∴ Solut^{n} = $(2113)^{*}(2113)^{*}$

→

i	A: w_i	B: x_i
1	10	101
2	011	11
3	101	011

i = 1

w = 10 , x = 101

i = 13

w = 10101 , x = 101011

~~i = 121~~

~~w = 1010110 , x = 101011~~

i = 133

w = 10101101 , x = 1010111011

i = 1333 - - - -

∴ This instance of PCP doesn't have a sol?

→

i	A: w_i	B: x_i
1	bbab	a
2	ab	abbb
3	baa	aa
4	b	bbb

ab

i = 2

w = ab , x = abbb

i = 21

w = abbbbab , x = abbbba

i = 214

w = abbbbabbb , x = abbbbabbbb

i = 2143

w = abbbbabbbbaa , x = abbbbabbbbaa

i = 4

w = b , x = bbb

i = 41

w = bbbab , x = bbbba

i = 414

w = bbbabbb , x = bbbabbbb

i = 4143

w = bbbabbbbaa , x = bbbabbbbaa

Scanned by CamScanner

Modified Post Correspondence Problem (MPCP):

Def^n: An instance of MPCP consists of lists of strings over some alphabet Σ.

A	B
w_1	x_1
w_2	x_2
⋮	⋮
w_k	x_k

For each i, the pair (w_i, x_i) is said to be a corresponding pair,

if the instance of MPCP has a sol^n if there is a sequence of zero or more integers $i_1 i_2 \cdots i_m$ $(m \geq 0)$ such that

$$w_1 w_{i_1} w_{i_2} \cdots w_{i_m} = x_1 x_{i_1} x_{i_2} \cdots x_{i_m}$$

The sol^n is $1 i_1 i_2 \cdots i_m$

→ An empty list can be a sol^n if $w_1 = x_1$

Reducing MPCP to PCP:

For every instance of MPCP, we can construct an instance of PCP.

Let us consider an instance of MPCP.

	A	B
i	w_i	x_i
1	w_1	x_1
2	w_2	x_2
⋮	⋮	⋮
k	w_k	x_k

Length = k

We can construct an instance of PCP as follows.

	C	D
i	y_i	z_i
0	y_0	z_0
1	y_1	z_1
2	y_2	z_2
⋮	⋮	⋮
k+1	y_{k+1}	z_{k+1}

Length = k+2.

~~For i=1 to k.~~

Alphabet is $\Sigma \cup \{*, \$\}$

For i=1 to k,

- Let y_i be w_i with a * after each symbol of w_i.

Scanned by CamScanner

- Let z_i be x_i with a '*' before each symbol of w_i.

$y_0 = *y_1$ & $z_0 = x_1$ &

$y_{k+1} = \$$ & $z_{k+1} = *\$$

→ MPCP

i	A: w_i	B: x_i
1	110	110110
2	0011	00
3	0110	110

So [1] = 2 3 1 (PCP)

00110110 00110

00110110110 0011011011O.

i	C: y_i	B: z_i	D: x_i
0	*1*1*0*	*1*1*0*1*1*0	
1	1*1*0*		*1*1*0*1*1*0
2	0*0*1*1*	*0*0	
3	0*1*1*0*	*1*1*0	
4			

i	C: y_i	D: x_i
0	*1*1*0*	*1*1*0*1*1*0
1	1*1*0*	*1*1*0*1*1*0
2	0*0*1*1*	*0*0
3	0*1*1*0*	*1*1*0
4	$	*$

Instance of PCP

* For PCP instance to have a sol^n, we must start with index 0 (i_0) & end with i_{k+1}.

If the constructed PCP has a sol^n then, the instance of MPCP also has a sol^n, i.e. $y_0 y_{i_1} \cdots y_{i_m} y_{k+1} = z_0 z_{i_1} \cdots z_{i_m} z_{k+1}$

i = 0

* w = *1*1*0* x = *1*1*0*1*1*0

i = 0 1

*1*1*0**1*1*0* *1*1*0*1*1*0*1*1*0*1*1*0

→ If we remove all '*'s and '$' from string, $z_0 z_{i_1} \cdots z_{i_m} z_{k+1}$ you'll get a string $x_1 x_{i_1} \cdots x_{i_m}$

→ If we remove all '*'s and '$' from string, $y_0 y_{i_1} \cdots y_{i_m} y_{k+1}$, then we'll get string $w_1 w_{i_1} \cdots w_{i_m}$

Scanned by CamScanner