UNIT V UNDECIDABILITY

Non Recursive Enumerable (RE) Language – Undecidable Problem with RE – Undecidable Problems about TM – Post's Correspondence Problem, The Class P and NP.

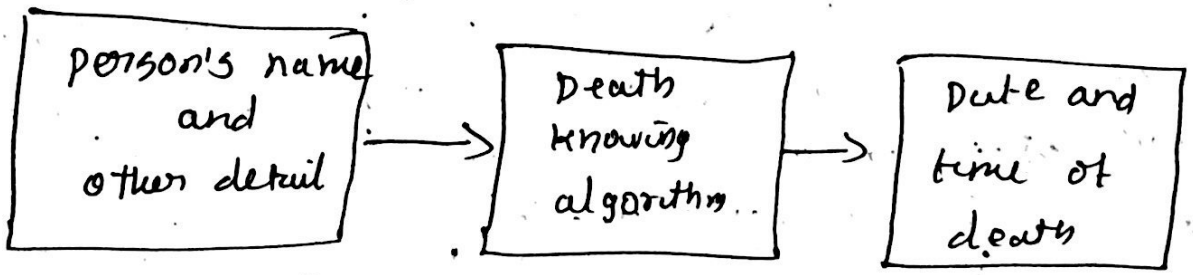
UNSOLVABLE PROBLEMS AND COMPUTABLE FUNCTIONS

UNSOLVABLE PROBLEMS

A problem whose language is recursive is said to be decidable otherwise, the problem is undecidable. That is, a problem is undecidable if there is no algorithm that takes as input an instance of the problem and determines whether the answer to that instance is yes or no.

Ex:

Can you develop an algorithm which will correctly tell us when a person will die? Whatever you want may be taken as input.



Undecidability of Death

For unsolvable problem, let us see the following problem.

1. Unsolvable problem of a non Recursive language
2. Unsolvable problem of Reduction
3. Unsolvable problem in Rice's Theorem
4. Post correspondence problem
5. Unsolvable Problems of Context Free Grammars

1) Unsolvable problem of a Non Recursive Language:

If a language is recursively enumerable then it is non Recursive. So now we have find a non recursive language that is unsolvable.

Let us see the following languages

1. Empty language (Le)
2. Non Empty language (Lne)
3. Non Self Accepting language (NSA)
4. Self Accepty language (SA)

1. Empty language "Le":

If $L(Mi) = \phi$, that is Mi does not accept any i/p, then w is in Le. Thus, Le is the language consisting of all TM's whose language is empty.

$$Le = \{ M \mid L(M) = \phi \}$$

2. Non Empty language (Lne):

If $L(Mi) \neq \phi$, then w is in Lne. Thus Lne is the language for TM that accept at least one input string

$$Lne = \{ M \mid L(M) \neq \phi \}$$

3. Non self ACCEPTING language (NSA):

The Non self Accepting language (NSA) is defined as

$$NSA = \{ w \in \{0,1\}^* \mid w = E(T) \}$$

for some Turing Machine T, and the i/p string $w \notin L(T)$

4. Self Accepting Language (SA):

The self Accepting language (SA) is defined as,

$$SA = \{ w \in \{0,1\}^* \mid w = E(T) \}$$

for some Turing Machine T, and the i/p string $w \in L(T)$

Theorem:

Non Empty language "Lne" is Recursively Enumerable.

Proof:

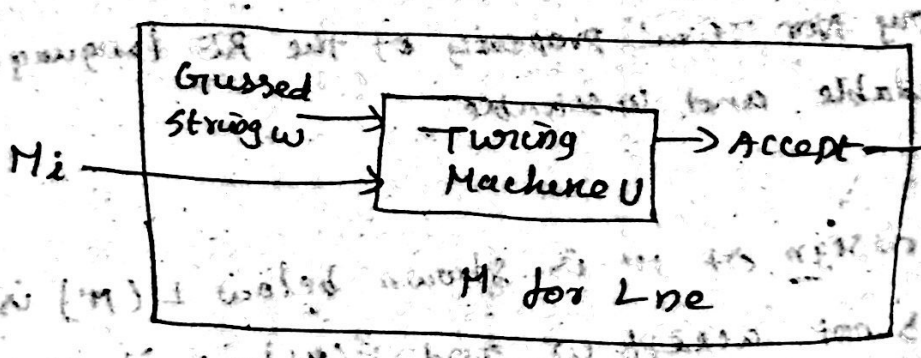
Now we have to construct a TM that accepts Lne.

M takes as inputs a Turing Machine code Mi

1) Using its Nondeterministic capability, M guesses an input w that Mi might accept.

2) M tests whether Mi accept 'w'. If Mi accepts 'w' then M also accepts its own input 'w'. This is done by M simulating the universal TM "U" that accept Lu.

3) If Mi accepts w, then M accepts its own input, which is Mi

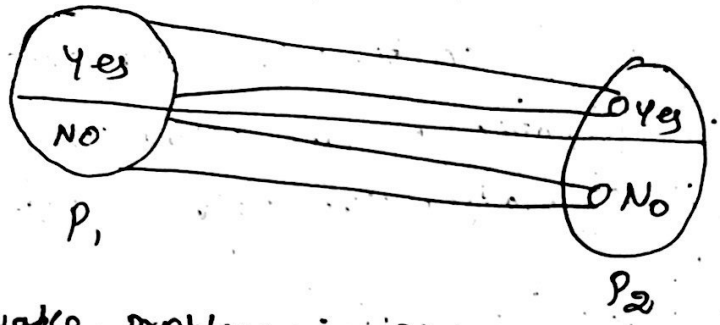


Thus if Turing Machine code Mi accepts even one string, M will guess that string and accept Mi. However, if $L[Mi] = \phi$, then no guess w leads to acceptance by Mi, so M does not accept Mi.

Thus $L(M) = Lne$ and Lne is recursively enumerable.

2) Unsolvable Problem of Reduction:

Formally, a reduction from P_1 to P_2 is a TM that takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape. So reduction takes an instance of P_1 as input and produces an instance of P_2 as output



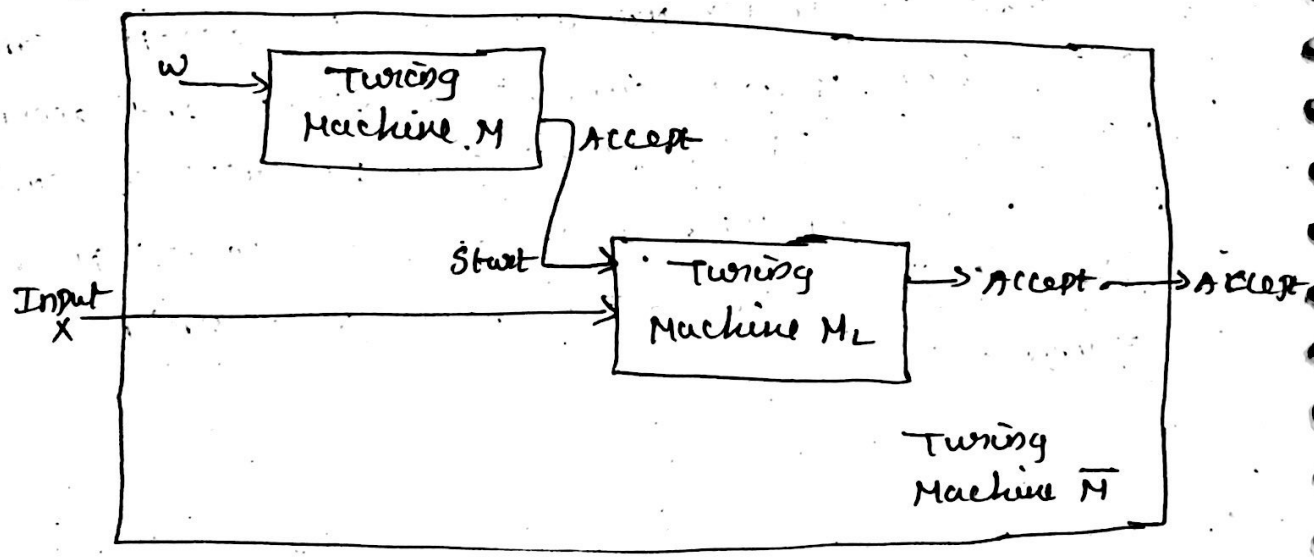
3) Unsolvable problem in Rice's Theorem:

Rice's Theorem:

Every Non Trival property of the RE language is Undecidable and Unsolvable.

Proof:

The design of M' is shown below L(M') is ϕ if M does not accept w, and L(M') = L if M accept w.



construction of M' for Rice Theorem

* $L(M') = \phi$ if M does not accept w
* $L(M') = L$ if M accept w

The Turing Machine M' is a two tape Turing Machine.

1) One Tape is used to simulates M on w
2) The other Tape of M' is used to simulates M_L on the input x to M'.

The Turing Machine M' is constructed to perform following,

→ Simulate M on input w. The string 'w' is not the input to M', rather M' writes M and w on to one of its tape and simulates the Universal Turing Machine 'U' on that pair.

→ If M does not accept w, then M' does not perform anything. M' never accept its own i/p x.

→ If M accept w, then M' accept its own i/p x.

This algorithm is a reduction of L_u to L_P, and proves that the property P is undecidable and unsolvable.

4) Post's correspondence Problem (PCP):

The undecidability of Strings is determined with help of Post's correspondence problem (PCP). Let us define the PCP.

The Post's correspondence problem consist of two lists of strings that are of equal length over the input Σ.

The two list are.

$A = w_1, w_2, w_3, \ldots, w_n$

and

$B = x_1, x_2, x_3, \ldots x_n$

then there exists a non empty set of integers $i_1, i_2, i_3, \ldots i_n$ such that-

$w_1, w_2, w_3, \ldots, w_n = x_1, x_2, x_3, \ldots, x_n$

to find $w_i = x_i$ then we say that PCP has a solution.

Example problem:

1) Let $\Sigma = \{0, 1\}$ and let A and B be two lists defined as follows,

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

Find the instance of PC

Solution:

$w_1 = 1$: $x_1 = 111$

$w_2 = 10111$ $x_2 = 10$

$w_3 = 10$ $x_3 = 0$

Now find instance of PCP

$w_1, w_2, w_3, \ldots w_n = x_1, x_2, x_3, \ldots x_n$

let us take $n = 4$ and take combination 2, 1, 1, 3

$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3$

$10111110 = 10111110$

Instance of PCP

2) Let $\Sigma = \{0,1\}$ and A & B be the list as,

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

Find the instance of PCP

Modified PCP:

An intermediate version of PCP is modified PCP (MPCP), there is the additional requirement on solution that the first pair on the A and B list must be the first pair in the solution.

An instance of MPCP is two lists,

$A = w_1 w_2, \ldots w_k$

$B = x_1 x_2, \ldots x_k$

And solution is a list of 'o' or more integer $i_1, i_2, \ldots i_m$ such that

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

Example problem:

Consider the following list A & B and find instance

	List A	List B
i	w_i	x_i
1	10	10
2	110	11
3	11	011

1	10
110	0
0	11

Soln:

1,3,2

Now the sequence taken is 2, 3

so

$w_1 w_2 w_3 = x_1 x_2 x_3$

$10110 11 = 10 11 011$

Instance of MPCP = 2,3

Completion of the Proof of PCP undecidability:

Rules:

1) The First Pair is

List A	List B
#	# $q_0 w$ #

2) Tape symbols and separator # can be appended

List A	List B
X	X
#	#

3) To simulate a move of M,

List A	List B	Transition
qX	Yp	if $\delta(q,X) = (p,Y,R)$
Zqx	pZY	if $\delta(q,X) = (p,Y,L)$
q#	Yp#	if $\delta(q,B) = (p,Y,R)$
Zq#	pZY#	if $\delta(q,B) = (p,Y,L)$

Z is tape symbol

4) For each q in F, then for all tape symbols X and Y:

List A	List B
XqY	q
Xq	q
qY	q

5) Finally, we use the final pair to complete the solution

List A	List B
q##	#

Example:

1). Let us convert the Turing machine, $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_3\})$ where δ is

q_i	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
q_1	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
q_2	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
q_3	–	–	–

And the input string $w = 01$. Find solution.

Soln:

Rule i:

List A	List B	Source
#	# q_1 01 #	$w = 01$, q_1 = Initial state
0	0	Tape symbols and the separator # is appended.
1	1	
#	#	
$q_1$0	1q_2	$\delta(q_1, 0) = (q_2, 1, R)$
0$q_1$1	$q_2$00	$\delta(q_1, 1) = (q_2, 0, L)$
1$q_1$1	$q_2$10	
0q_1#	$q_2$01#	$\delta(q_1, B) = (q_2, 1, L)$
1q_1#	$q_2$11#	
0$q_2$0	$q_3$00	$\delta(q_2, 0) = (q_3, 0, L)$
1$q_2$0	$q_3$10	
$q_2$1	0q_1	$\delta(q_2, 1) = (q_1, 0, R)$

List A	List B	source
$q_2\#$	$\circ q_2\#$	$\delta(q_2, B) = [q_2, 0, R)$
$\circ q_3 0$	q_3	
$0 q_3 1$	q_3	
$1 q_3 0$	q_3	
$1 q_3 1$	q_3	
$\circ q_3$	q_3	q_3 is an Accepting state.
$1 q_3$	q_3	
$q_3 0$	q_3	
$q_3 1$	q_3	
$q_3 \# \#$	$\#$	q_3 is an Accepting state.

$w = 01$

$q_1 01 \vdash 1 q_2 1 \vdash 10 q_1 \vdash 1 q_2 0 1 \vdash q_3 101$ // accepted

5. Unsolvable problems of CFGs:

Theorem:

If is undecidable whether CFGs is ambiguous.

Proof: we have to prove that " G_{AB} is ambiguous if and only if instance (A, B) of PCP has a solution.

If part:

there are two derivation G_{AB} as

$A \to w_1 A a_1 \mid w_2 A a_2 \mid \dots \mid w_k A a_k \mid w_1 a_1 \mid \dots \mid w_k a_k$

$B \to x_1 B a_1 \mid x_1 B a_2 \mid \dots \mid x_k B a_k \mid x_1 a_1 \mid \dots \mid x_k a_k$

where A and B are the list generated by CFGs.

Now derivations are,

$S \rightarrow A \rightarrow w_{i_1} w_{i_2} \cdots w_{i_m} a_{i_m} \cdots a_{i_1}$

$S \rightarrow B \rightarrow x_{i_1} x_{i_2} \cdots x_{i_m} a_{i_m} \cdots a_{i_1}$

The solution is

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

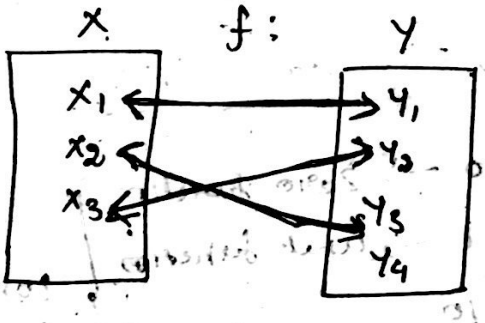
This implies that G_{A+B} is ambiguous.

COMPUTABLE FUNCTIONS:

Primitive Recursive Functions:

Basic Definitions

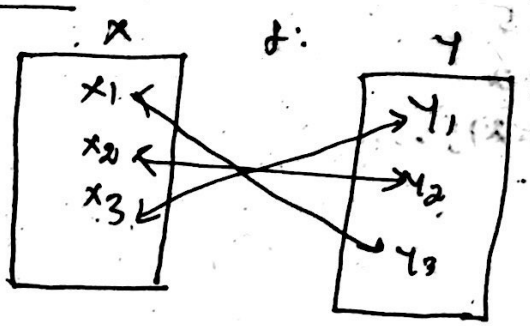
1. Partial Function



Partial function, f from X to Y is a function that assigns every elements of X to at most one element of Y.

Ex: $f(m,n) = m-n$ is a partial function [since $m-n$ generate +ve & -ve values]

2. Total Function:



Total function, f from X to Y is defined as the function that assigns every element of x to unique element of y

Ex: $f(m,n) = m+n$ [generates only five values]

3. Initial Function:

The initial function include

i) Constant Function
ii) Successor Function
iii) Projection Function

i) Constant function:

A function of the form,

$C_a^k : N^k \longrightarrow N$ for $k \geq 0$ & $a \geq 0$ is called Constant Function

General Form:

$$C_a^k(x) = a \text{ for } x \in N^k$$

Example:

$C(5) = 0$ → Zero function
$C(4) = 1$ → Unit function
$C(10) = 10$
⋮

} Constant Function

ii) Successor Function [S]:

A function, S: $N \rightarrow N$ defined by $S(x) = x+1$ is said be successor function.

Example:

$S(4) = 5$
$S(1) = 2$
$S(20) = 21, \ldots$

iii) Projection function [P_i^k]

A function of the form, $P_i^k : N^k \to N$ defined by $P_i^k(x_1, x_2, x_3 \ldots x_i, \ldots x_k) = x_i$ for $k \geq 1$ and $1 \leq i \leq k$

Example: $P_1^3(1, 3, 5) = 1$

$P_3^4(2, 4, 6, 8) = 6$

$P_2^2(0, 1) = 1$, etc

4. Complex Primitive Recursive Functions:

Complex functions are obtained by applying certain operations on the initial functions. They are,

i) Composition

ii) Primitive Recursion

i. Composition:

Let 'f' be a partial function defined as,

$N^k \to N$, for $1 \leq i \leq k$

and 'g' is a partial function from $N^m \to N$, then the composition of 'f' and 'g' is a partial function, defined by 'h' as

$$h(x) = f(g_1(x), g_2(x), \ldots g_k(x)) \quad [\because x \in N^m]$$

that is,

$$h(x) = f(g(x_1, x_2 \ldots x_n), g_2(x_1, x_2 \ldots x_n), \ldots g_k(x_1, x_2 \ldots x_n))$$

ii. Primitive Recursion:

A function 'f' over N is recursive if there exists a constant, 'k' and a function 'h' (x, y) such that

$f(0) = k$

$f(n+1) = h(n, f(n))$

Let g be function of 'n' variable and 'h' be another function with '(n+2)' variables.

The primitive Recursion function is obtained as,

$$f: N^{n+1} \to N,$$
$$f(x,0) = g(x)$$
$$f(x,k+1) = h(x, k, f(x,k)), \quad x \in N^n, k \geq 0$$

That is,

$$f(x_1, x_2, \cdots x_n, 0) = g(x_1, x_2, \cdots x_n) \text{ and}$$
$$f(x_1, x_2, \cdots x_n, k+1) = h(x_1, x_2, \cdots x_n, k, f(x_1, x_2, \cdots x_n, k))$$

Example problems:

1) Let $f_1(x,y) = x+y$, $f_2(x,y) = 2x$, $f_3(x,y) = xy$, $g(x,y,z) = x+y+z$. Find the composition of g with f_1, f_2, f_3.

Soln:

The composition function, $h(x,y)$ is given as,

$$h(x,y) = g[f_1(x,y), f_2(x,y), f_3(x,y)]$$
$$= g(x+y, 2x, xy)$$
$$= x+y+2x+xy$$
$$\boxed{h(x,y) = x+y+2x+xy}$$

Here, f_1, f_2, f_3, g are total function then h also total function

2) Given: $f_1(x,y) = x-y$, $f_2(x,y) = y-x$, $g(x,y) = x+y$ obtain $h(x,y)$ over g and f.

Soln:

$$h(x,y) = g[f_1(x,y), f_2(x,y)]$$
$$= g(x-y, y-x)$$
$$\boxed{h(x,y) = x-y+y-x}$$

3) Let $f_1(x_1,x_2) = x_1x_2$, $f_2(x_1,x_2) = \lambda$, $f_3(x_1,x_2) = x_1$, $g(x_1,x_2,x_3) = x_2x_3$. Generate composition function $h(x_1,x_2)$.

Soln:

$$h(x_1,x_2) = g[f_1(x_1,x_2), f_2(x_1,x_2), f_3(x_1,x_2)]$$
$$= g(x_1x_2, \lambda, x_1)$$
$$= \lambda x_1$$

$$\boxed{h(x_1,x_2) = \lambda x_1 = x_1}$$

4) Show that $f_1(x,y) = x+y$ is primitive recursive.

Soln:

To prove that f_1 is primitive recursive.

Let g be a function of single variable and h be a function of three variables.

When $y=0$:

$$f_1(x,y) = f_1(x,0)$$
$$f_1(x,0) = x+0 = x = g(x) = U_1^1(x)$$

When $y=y+1$

$$f_1(x,y) = f_1(x,y+1)$$
$$f(x,y+1) = h(x,y,f_1(x,y))$$
$$= U_3^3(x,y,z)$$

$\therefore$ g and h are primitive recursive

Hence $f_1(x,y)$ is primitive recursive.

—x—

2) Show that $f(x,y) = x-y$ is primitive Recursive.

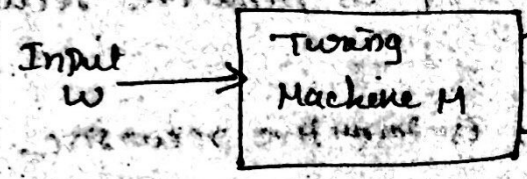
RECURSIVE AND RECURSIVELY ENUMERABLE LANGUAGES:

Recursive language:

A language 'L' is said to be Recursive language if $L = L(M)$ for some Turing Machine M such that,

1) If w is in L, then M accept
2) If w is not in L, then M reject, but it halt

So L is called decidable language if it is a Recursive language



Recursive Enumerable Language [RE]:

A language is Recursively Enumerable if there exists a Turing Machine that accepts every string present in the language and does not accept the string and it may cause the Turing Machine to enter into an infinite loop.

Properties of Recursive And RE languages:

1) The Union of two Recursively Enumerable [RE] is Recursively Enumerable.
2) The language L and its complement $\bar{L}$ are recursively enumerable, then L and $\bar{L}$ are recursive
3) The complement of a recursive language is also recursive.
4) The union of two recursive language is recursive
5) The intersection of two recursive language is recursive.
6) The union of two recursively enumerable languages is recursively enumerable.
7) The intersection of two recursively enumerable language is recursively enumerable.

Theorem:

If L is a recursive language, then L' is also a Recursive language.

(or)

If L is recursive language, so is L'.

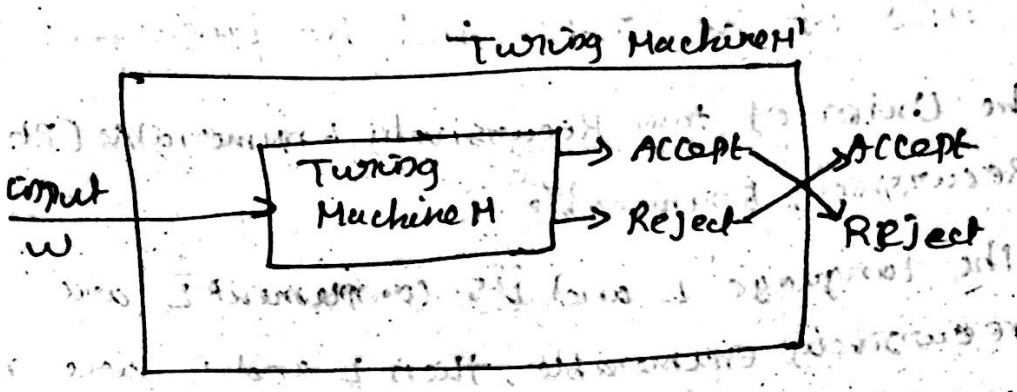
(or)

The complement of recursive language is also recursive language.

Proof:

To Prove L is recursive then L' is also recursive

let us construct the complement of the Turing Machine M as M' such that $L' = L[M']$ and its shown below



Construction of M' accepting the complement of M

M' just behaves like M. The Turing Machine M is constructed as follows:

1) The Accepting States of M are made as non accepting States of M' with no transitions.

2) M' has a new accepting State 'r'. and there are no transitions from 'r'.

Since M is guaranteed to halt, then M' is also guaranteed to halt. The Turing Machine M' exactly accepts those strings that are not accepted by TM

$$L' = \Sigma^* - L$$

. So the complement of recursive Language is also recursive.

Theorem:

If a language L and L' are Recursively Enumerable (RE) then L is Recursive.

(or)

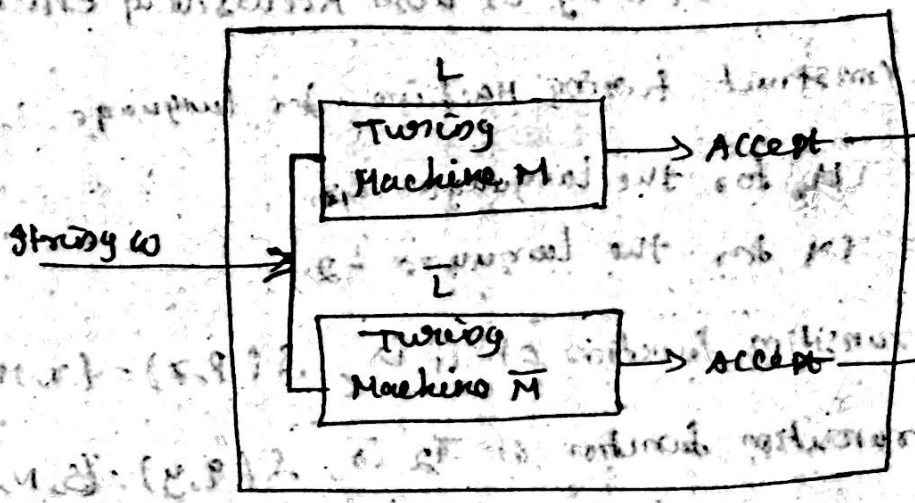
If both a language L and its complement L' are RE then L is recursive language.

Proof:

To prove L and L' is RE, then L is Recursive.

Let $L = L(M_1)$ and $L' = L(M_2)$ for some Turing Machine M_1 and M_2. Both M_1 and M_2 are simulated in

Parallel by a Turing Machine M. And it is shown below,



Simulation of Two TM accepting a lang & its complement

Here we are converting the two tape TM M into one tape TM 'M' to make simulation easy.

* One tape of M simulates the tape of M_1
* The other tape of M simulates the tape of M_2
* The states of M_1 & M_2 are each components of the state of M.
* If input w to M is in L, then M_1 will also accepts. then M accepts and halts.
* If input w is not in L, then M halts without accepting.

∴ we can conclude that L is Recursive language.

Theorem:

If L_1 and L_2 are recursively enumerable language over Σ, then $L_1 \cup L_2$ is also RE.

(or)

Union of two recursively enumerable language is RE.

Proof:

To prove $L_1 \cup L_2$ is also Recursively enumerable.

Let us construct turing Machine for language L_1 and L_2.

T_1 = TM for the language L_1

T_2 = TM for the language L_2

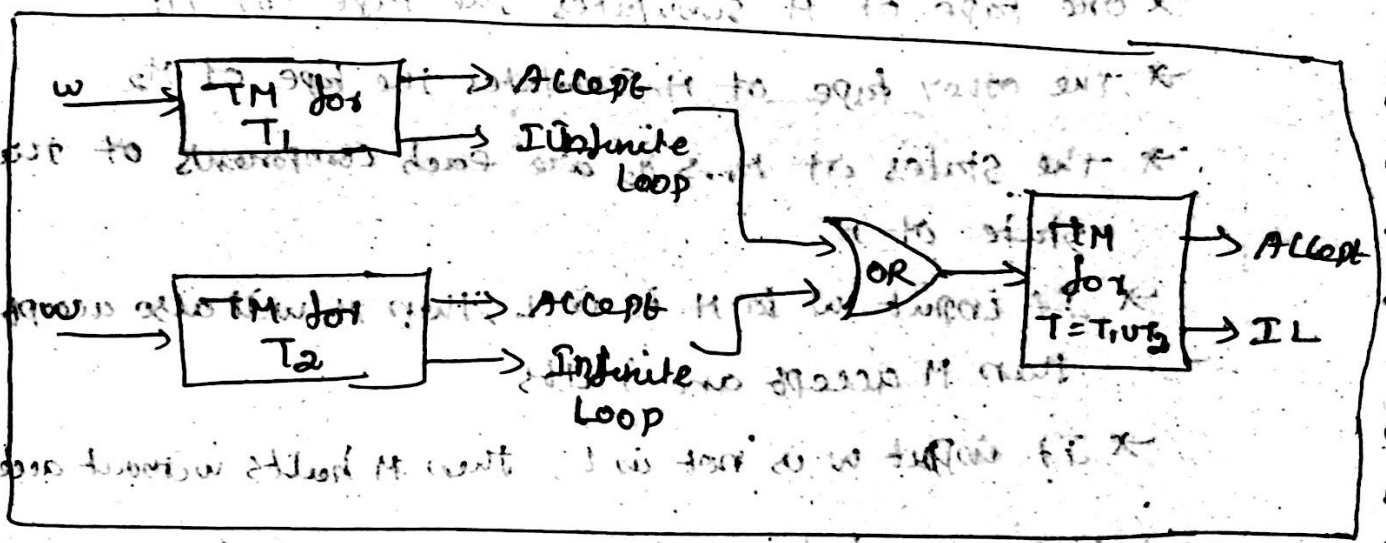
The transition function of T_1 is, $\delta(p, x) = (r, M, L)$

The transition function of T_2 is, $\delta(q, y) = (s, N, R)$

Then the transitions of $T_1 \cup T_2$ will be,

$$\delta[(p,q),(x,y)] = \{(r,s),(M,N),(L,R)\}$$

The Simulation of this machine is given below,



$T = T_1 \cup T_2$ is also Recursively Enumerable.

Here the turing machine T accept the string is accepted by using one of T_1 and T_2 and T enter in to infinite loop if both T_1 and T_2 reject the string.

Hence $T = T_1 \cup T_2$ and its language $L = L_1 \cup L_2$ is recursively enumerable.

Theorem:

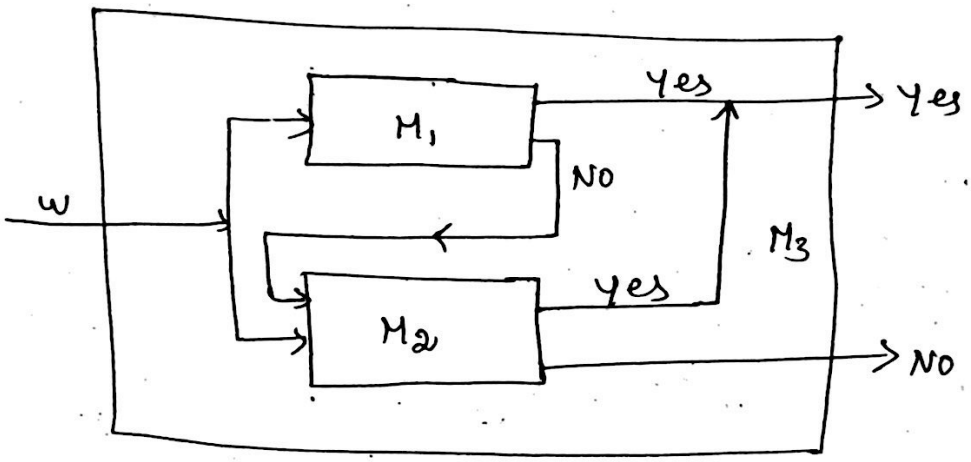
The union of two recursive language is recursive.

Proof:

Let L_1 and L_2 be two recursive language that are accepted by the Turing machine M_1 and M_2, given by

$$L(M_1) = L_1$$
$$L(M_2) = L_2$$

Let M_3 be the Turing Machine constructed by the union of M_1 and M_2. M_3 is constructed as,



* If $w \in L_1$, then M_1 accepts and thus M_3 also accepts since $L(M_3) = L(M_1) \cup L(M_2)$

* If M_1 reject, $[w \notin L_1]$, then M_3 simulates M_2. M_3 halt with "yes" if M_2 accepts "w" else returns "NO". Hence M_3, M_2, M_1 halt with either yes or NO.

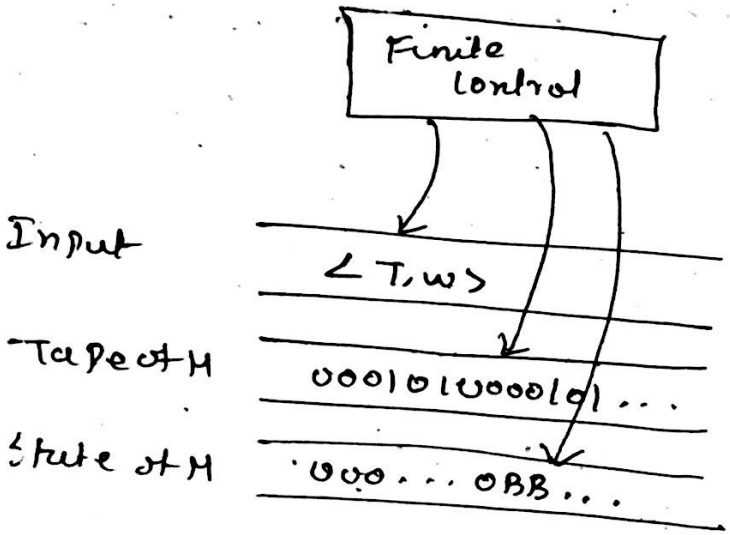
Thus the union of two recursive language is also recursive.

Universal Turing Machine:

→ The universal turing machine, Tu takes over the program and input set to process the program

→ The program and the inputs are encoded and stored on different tapes of multi-tape TM.

→ The Tu thus take up $\langle T, w \rangle$ where T is the special purpose TM that passes the program in the form of binary string, w is the data set that is to be processed by T.



The universal language L_u, is the set of all binary string (α), where α represents the ordered pairs $\langle T, w \rangle$ where

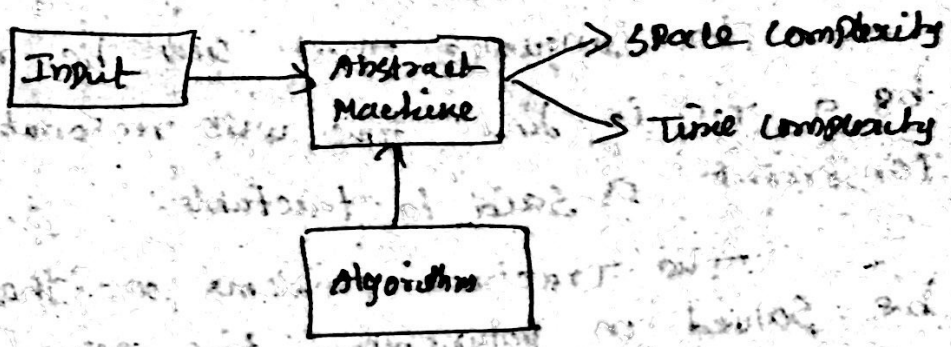
T → Turing Machine

W → any input string accepted by T

It can also be represented as $\alpha = \langle e(T) e(w) \rangle$

MEASURING AND CLASSIFYING COMPLEXITY:

Growth Rates of Functions:



Complexity problem is classified in two types

* Space complexity.
* Time complexity

→ The space complexity is the amount of memory space required by the algorithm/problem to complete its computation.

→ The time complexity is the amount of time required to run the program of the problem.

The growth rates of the functions can also be compared using the asymptotic notations like

* Big-oh [O]
* Big-Omega [Ω]
* Big-Theta [Θ]

TOP

TRACTABLE AND INTRACTABLE PROBLEMS:

Tractable Problems / Languages:

The language that can be recognized by a TM in finite time with reasonable space constraint is said to tractable.

The Tractable problems are those that can be solved in Polynominal time period.

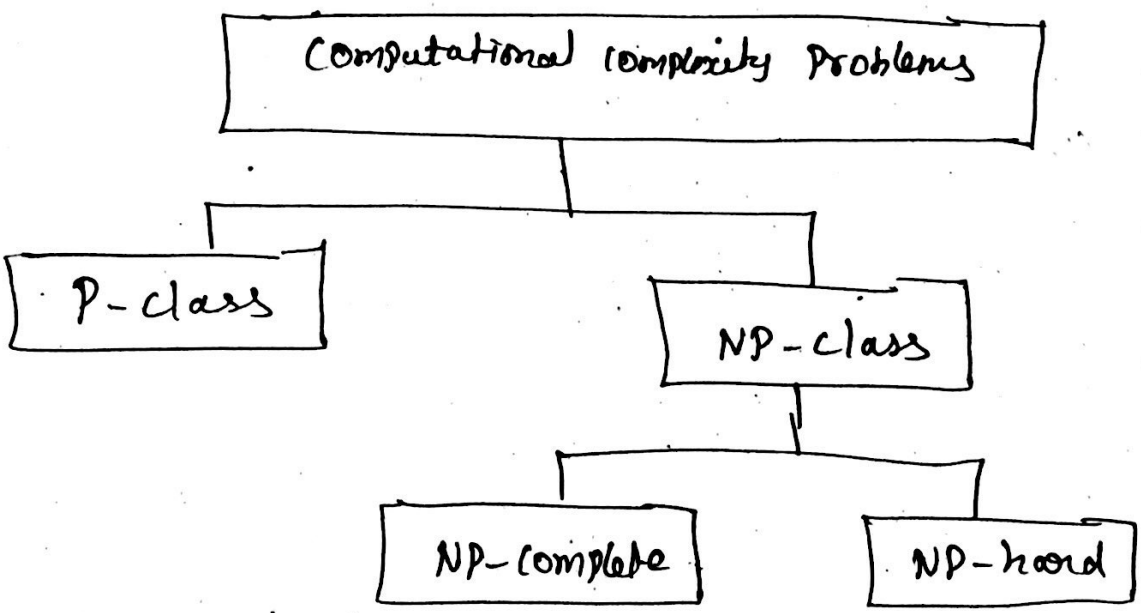
Intractable Problem:

The language that cannot be recognized by a TM with reasonable space and time constraint is called as Intractable problems.

These problems cannot be solved in Polynominal time period

Tractable and Possibly Intractable Problems: P and NP:

These are two groups in which a problem can be classified.



P-class Problem:

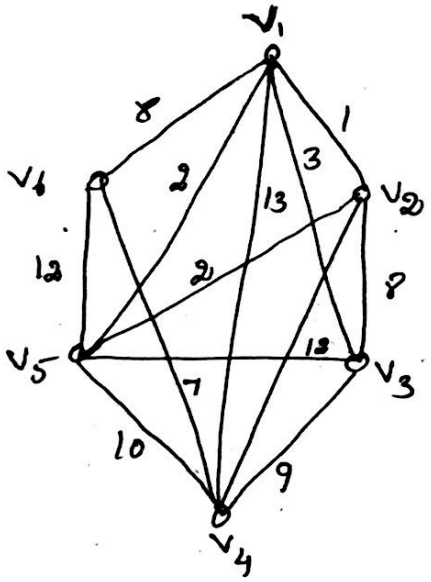
Problems that can be solved in Polynomial time.

Ex: Searching element, sorting element, multiplication of integers, finding all-pair-shortest path, finding minimum spanning tree, etc.

Example of P class problem:

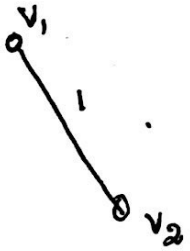
Kruskal's Algorithm: → In Kruskal's alg. the minimum weight is obtained.
→ In this alg also the circuit should not be formed.
→ Each time the edge of minimum weight is selected.

Example:

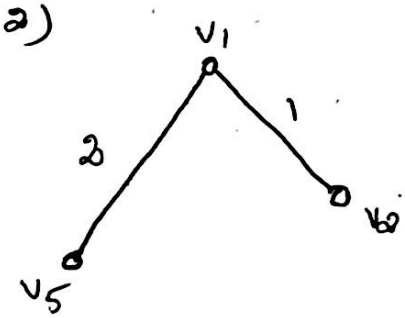


Soln:

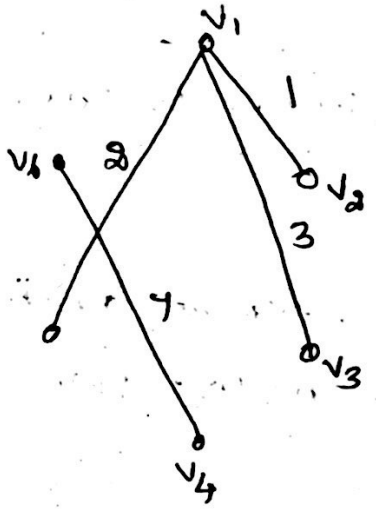
1)



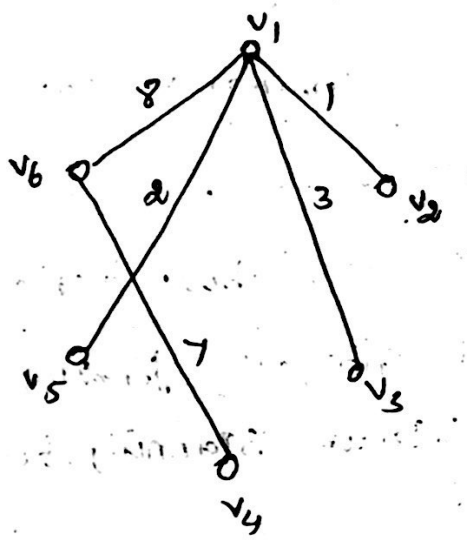
2)



3)

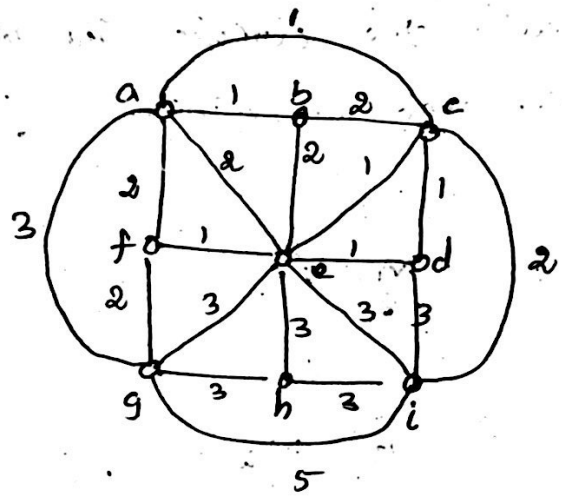


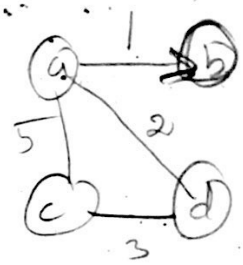
4)



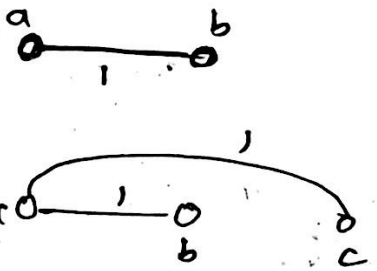
weight = 21

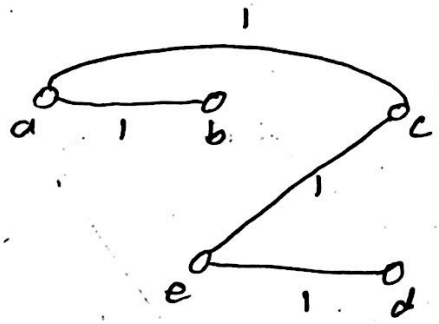
Example2: Find the minimum spanning tree using Kruskal's algorithm.



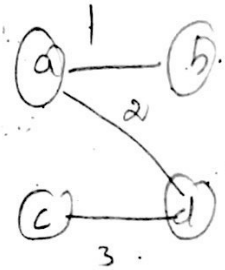


Soln:



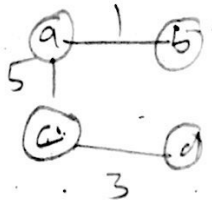


1)



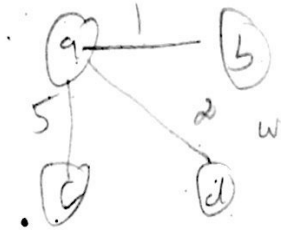
$w(T_1) = 6$

2)

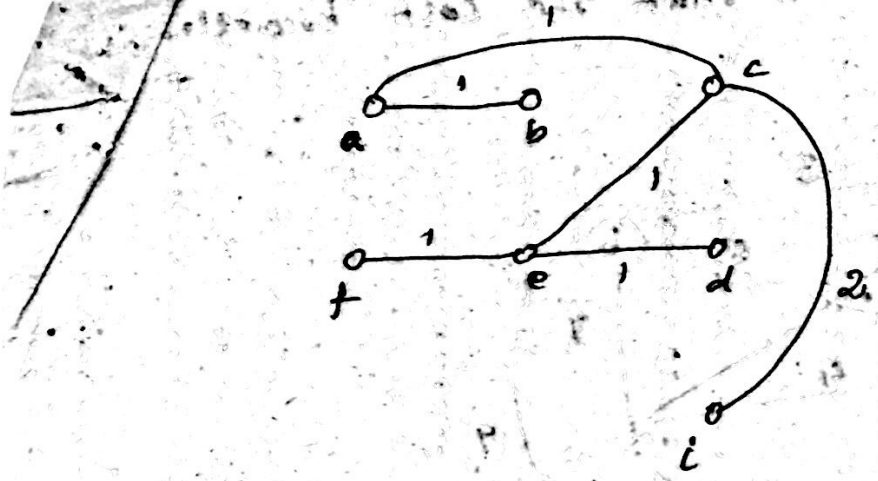


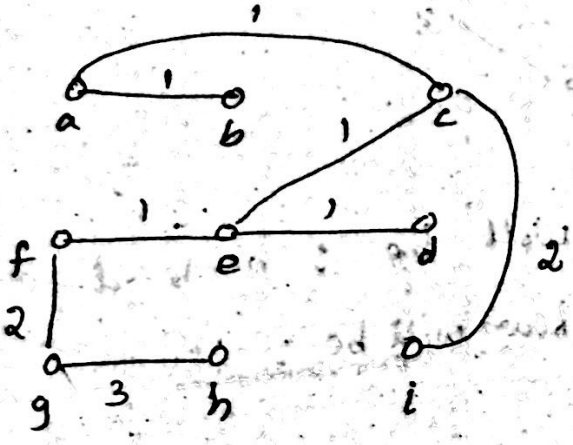
$w(T_2) = 9$

3)



$w(T_3) =$





NP-class problem:

Problems that can be sloved in non-deterministic polynominal time is called as NP-class problems

These types of NP problems are known as intractable problems

Example:

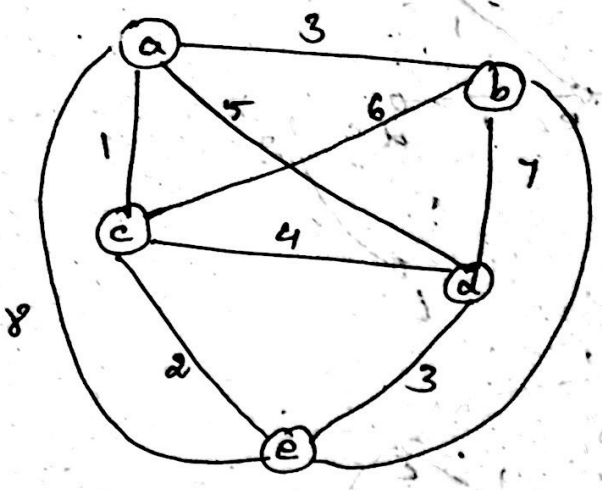
Towers of Hanoi, Traveling Salesman Problem, Graph colouring problem, Hamiltonian circuit problem, satisfiability problem, etc.

Example of NP class problem:

Travaling Salesman's problem (TSP)

-> Given a set of cities and cost to travel b/w each pair of cities, determine whether there is a path that visit every city once and returns to the

the first city. such that the cost travelled is less



The tour path will be : a - b - d - e - c - a

The total cost of tour will be : 16

NP - complete problem:

A problem is said to be NP - complete if it belongs to NP class problem and can be solved in polynomial time.

They are also called polynomial - time reducible problem.

NP - Hard problem:

→ A problem is said to be NP hard if there exists an algorithm for solving it and it can be translated into one for solving another NP problem.

→ A problem P is NP-Hard if

1) The problem is an NP class problem

2) For any other problem, P_2 in NP, there is a polynomial time reduction of L_2 to L_1.

→ Every NP complete problem must be NP - hard problem.

PROBLEMS ABOUT TURING MACHINE:

In Problems about Turing machine, we are going to see the following concepts.

1) Decidable Problems
2) Undecidable Problems
3) Codes for the Turing Machine
4) Enumerating the Binary string of Turing Machine
5) Undecidable problems about Turing machine

Problem types,

There are basically three types of Problems namely,

* Decidable | solvable | recursive
* Undecidable | unsolvable
* semi decidable | Partial solvable | recursively enumerable

Decidable | Solvable Problems:

A Problem, P is said to be decidable if there exists a turing machine, TM that decides P. Thus P is said to be recursive.

Consider a TM, M that halts with either 'yes' or 'no' after computing the input.

The machine is defined as,

$$F_P(w) = \begin{cases} 1 & \text{if } P(w) \\ 0 & \text{if } \neg P(w) \end{cases}$$

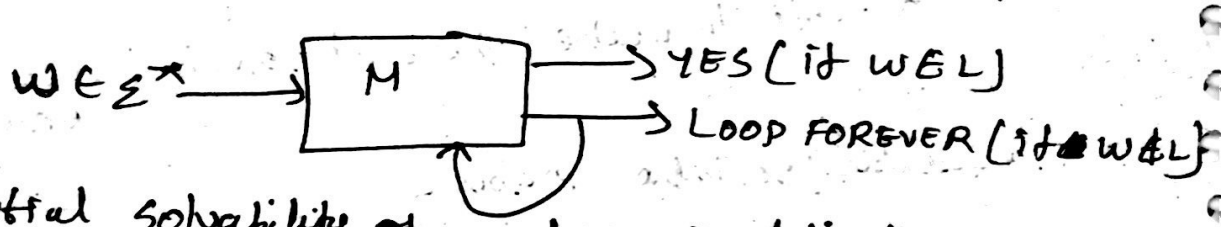
Undecidable Problem:

A Problem, P is said to be undecidable if there is a Turing machine, TM that doesn't decides P.

Semi decidable / Partial solvable / recursively enumerable

A problem, P is said to be semi decidable, if P is recursively enumerate.

A problem is RE if M terminates with 'YES' if it accepts $w \in L$; and doesn't halt if $w \notin L$



Partial solvability of machine is defined as,

$$F_P(w) = \begin{cases} 1 & \text{if } P(w) \\ \text{undefined} & \text{if } \neg P(w) \end{cases}$$

3. Code for Turing Machine

Now we are going to generate a binary code for the Turing machine so that each Turing machine with input alphabet {0,1} may be as a binary string.

Let us assume the states $q_1, q_2, q_3, \cdots q_n$ some value of K.

For Example

$q_1 \rightarrow 0$

$q_2 \rightarrow 00$

$q_3 \rightarrow 000$ etc

1) Assume that tape symbols as $x_1 x_2 x_3 \cdots x_m$ for some value 'm' such that

x_1 always will be symbol '0'

x_2 always will be symbol '1'

x_3 always will be 'B'

2) Assume the directions 'L' as D_1 and 'R' as D_2

Left $\rightarrow$ L $\rightarrow D_1 \rightarrow 0$

Right $\rightarrow$ R $\rightarrow D_2 \rightarrow 00$

We can encode the transition δ as follows

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

Code for the string $= 0^i 1 0^j 1 0^k 1 0^l 1 0^m$

The code for the entire Turing machine M consists of all the codes for the transitions in some order

Complete code $= C_1 11 C_2 11 C_3 11 \cdots 11 C_{N-1} 11 C_N$

where each C is the code for one transition of the Turing machine 'M'.

Example:

1) Find the code of the Turing Machine M be,

$M = [\{q_1, q_2, q_3\}, \{0,1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\}]$

where δ consist of the following rules,

$\delta[q_1, 1] = (q_3, 0, R)$

$\delta[q_3, 0] = (q_1, 1, R)$

$\delta[q_3, 1] = (q_2, 0, R)$

$\delta[q_3, B] = (q_3, 1, L)$

Soln:

According to the code for Turing Machine, the assumptions are;

1) The states are

$q_1 \rightarrow 0$

$q_2 \rightarrow 00$

$q_3 \rightarrow 000$

2) The Tape symbols are 0, 1, B. Assume

$0 \rightarrow x_1 \rightarrow 0$

$1 \rightarrow x_2 \rightarrow 00$

$B \rightarrow x_3 \rightarrow 000$

3) The Directions are, of the form of,

Left $\rightarrow L \rightarrow D_1 \rightarrow 0$

Right $\rightarrow R \rightarrow D_2 \rightarrow 00$

Now the transitions are,

1) $\delta[q_1, 1] = (q_3, 0, R)$

where $C_1 = 0^1 10^2 10^3 10^1 10^2$

$C_2 = 00100010100$

2) $\delta[q_3, 0] = (q_1, 1, R)$

$C_2 = 000101001000$

3) $\delta[q_3, 1) = (q_2, 0, R)$

$C_3 = 0001001001010 0$

4) $\delta(q_3, B) = (q_3, 1, L)$

$C_4 = 0001000100010010$

Complete code = $C_1 11 C_2 11 C_3 11 C_4$

Code = 0100100010100 ■ 11 0001010100100 11 0001001001010 0 11 0001001001010 0 11 0001000100010010

2) write the code for the following TM transitions,

$\delta(q_1, a) = (q_1, x, R)$, $\delta(q_1, 4) = (q_3, 4, R)$, $\delta(q_1, x) = (q_1, 0, R)$

$\delta(q_1, 1) = (q_2, 4, L)$, $\delta(q_1, 4) = (q_1, 4, R)$, $\delta(q_2, 0) = (q_2, 0, L)$

$\delta(q_2, x) = (q_1, x, R)$, $\delta(q_2, 4) = (q_2, 4, L)$, $\delta(q_3, 4) = (q_3, 4, R)$

$\delta[q_3, B) = (q_3, B, R)$.

---

4) Enumerating the Binary Strings:

In this binary string enumeration, we assign the integers 1 to all the binary strings, so that each string corresponds to one integer and each integer corresponds to one string.

Ex:

ε → First string

0 → second string

1 → Third string

00 → Fourth string

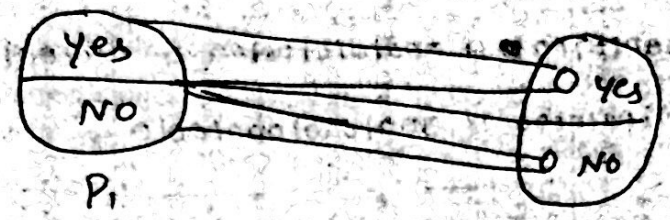
01 → Fifth string

10 → Sixth string and so on

CHOMSKIAN HIERARCHY OF LANGUAGES

5) Undecidable Problems about Turing Machine:

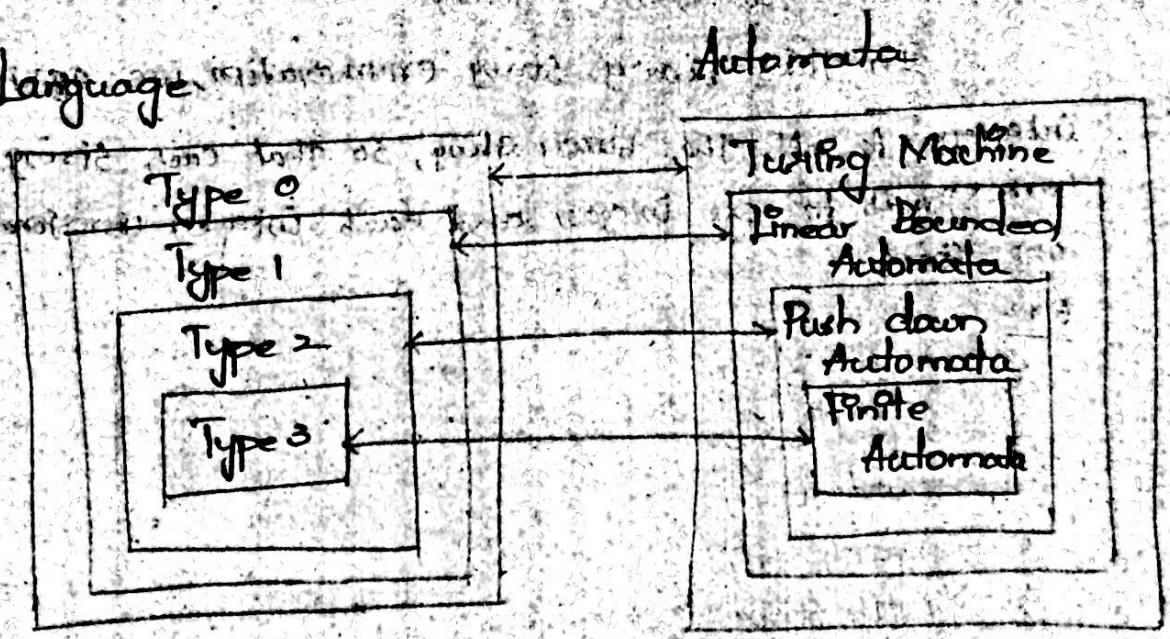
The proofs of undecidable problems about Turing machine says that any non trival property of Turing machine that depends only on the language the turing machine accepts must be undecidable.



Formally, a reduction from P_1 to P_2 is a TM that takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape. So reduction takes an instance of P_1 as input and produces an instance of P_2 as output

—x—

CHOMSKIAN HIERARCHY OF LANGUAGES:



The Halting Problem:

The halting problem is the problem of finding if the program/machine halts or loop forever.

The halting problem is undecidable over Turing Machines.

Ex:

```
while(1)
{
  printf("Halting problem");
}
```

The above code goes to infinite loop since the argument of while loop is true forever. Thus it doesn't halts.

Hence Turing problem is the example of undecidability.

Representation of the halting set

The halting set is represented as,

$$h(M, w) = \begin{cases} 1 & \text{if } M \text{ halts on input } w \\ 0 & \text{otherwise} \end{cases}$$

where,

M -> Turing Machine

w -> Input string

Theorem:

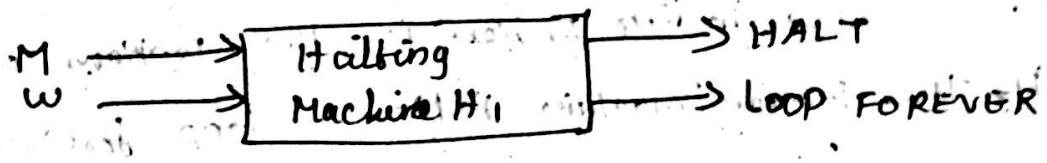
"Halting problem of Turing machine is unsolvable/undecidable.

Proof:

The theorem is proved by the method of proof by contradiction.

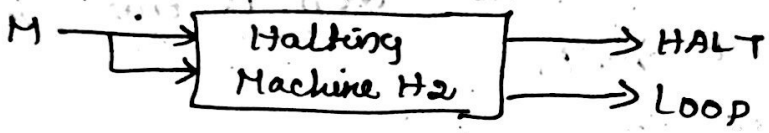
Let us assume that TM is solvable/decidable.

Construction of H_1:



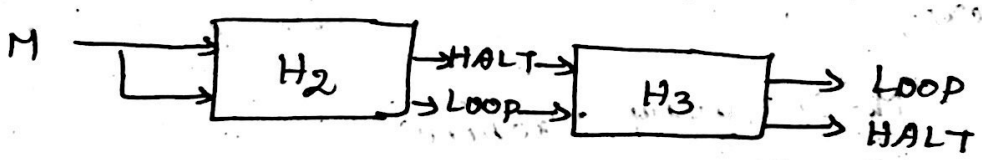
* Consider, a string describing M and I/p string, w for M.
* let H_1 generates "halt" if H_1 determines that the turing machine, M stops after accepting the I/p, w.
* otherwise H_1 loops forever when, M doesn't stop on processing w.

Construction of H_2:



* H_2 is constructed with both the inputs being M.
* H_2 determines M and halts if M halts otherwise loops forever.

Construction of H_3:



* let H_3 be constructed from the outputs of H_2
* If the o/p of H_2 are HALT then H_3 loops forever.
ELSE if the o/p of H_2 is loop forever, then H_3 halts.

Thus H_3 acts contractor to that of H_2.

Thus Halting problem is undecidable.