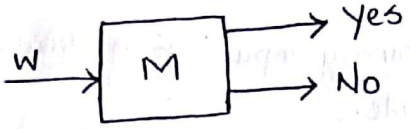
UNIT-V COMPUTATIONAL COMPLEXITY

Undecidability:

Basic definitions:

Recursive Language => A language is recursive if there exists a Turing machine that accepts every string of the language and rejects the string that is not in the language.



Decidable problems => A problem whose language is recursive is said to be decidable otherwise the problem which can be answered as "yes" are called solvable (or) decidable.

Example: Sorting, Seaching etc..

Undecidable problems => The problem which can be answered as "No" are called Undecidable problems.

A problem is said to be undecidable if there is no algorithm that takes as input, an instance of the problem and determines whether the output to that instance is 'yes' or 'No'.

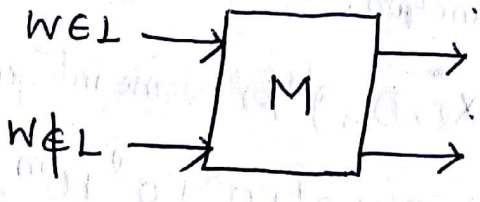
Example: post correspondence problems, Halting problem, etc,..

Recursively enumerable language =>

A language $L \subseteq \Sigma^*$ is recursively enumerable if there exists a TM, M that accepts every string, $w \in L$ and does not accept strings that are not in L.

If the input string is accepted, M halts with answer "yes".

If the string is not an element of L, then M may not halt and enters into infinite loop.



⇒ A language is RE if there exists a TM that accepts every string of the language and does not accept string that are not in the language.

⇒ The long range goal of proving the undecidability consisting of pairs such that,

(i) M is a Turing machine encoded in binary

(ii) W is a string of 0's and 1's.

(iii) M accepts input W.

⇒ If the problem with the binary input is undecidable, then TM with any other alphabet is undecidable.

⇒ The input given to the Turing machine should be in a well-formed representation using binary values.

Codes for Turing Machine

To represent a TM $M = (Q, \{0,1\}, \Gamma, \delta, q_1, B, F)$ as a binary string, first assign integer to states, type symbols and directions L and R.

(i) Assume the states are $q_1, q_2, \ldots q_k$ for some k. Using the integers available in the suffix of each state, the string can be represented as $q_1 \rightarrow 0$, $q_2 \rightarrow 00$, $q_3 \rightarrow 000$ etc...

(ii) Assume the tape symbols 0, 1, B are represented as, $X_1, X_2, X_3 \ldots X_m$

Where $X_1 \rightarrow 0$

$X_2 \rightarrow 1$

$X_3 \rightarrow B$

(iii) Assume the directions are represented as D_1 and D_2, where.

$L \rightarrow D_1 \rightarrow 0$

$R \rightarrow D_2 \rightarrow 00$

After representing each state, symbol and direction using integers, we can encode the transition function.

$$\delta(q_i, X_j) = (q_k, X_l, D_m) \text{ for some integers } i, j, k, l, m.$$

The encoded string is given by $0^i 1 0^j 1 0^k 1 0^l 1 0^m$.

The code for entire TM consists of all strings, separated by pair of 1's.

$$C_1 11 C_2 11 C_3 11 \cdots C_{n-1} 11 C_n.$$

problem: obtain the code for $\langle M, 1011\rangle$, where $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\})$

has moves $\delta(q_1,1) = (q_3,0,R)$, $\delta(q_3,0) = (q_1,1,R)$

$\delta(q_3,1) = (q_2,0,R)$, $\delta(q_3,B) = (q_3,1,L)$

Solution:

Assume the tape symbols be encoded as,

$0 - X_1 = 0$

$1 - X_2 = 00$

$B - X_3 = 000$

Directions can be encoded as, $L - D_1 = 0$, $R - D_2 = 00$

Transition are encoded as,

(i) $\delta(q_1,1) = (q_3,0,R)$ $\quad C_1 \Rightarrow 0^1 1 0^2 1 0^3 1 0^1 1 0^2$

(ii) $\delta(q_3,0) = (q_1,1,R)$ $\quad C_2 \Rightarrow 0^3 1 0^1 1 0^1 1 0^1 1 0^2$

(iii) $\delta(q_3,1) = (q_2,0,R)$ $\quad C_3 \Rightarrow 0^3 1 0^2 1 0^2 1 0^1 1 0^2$

(iv) $\delta(q_3,B) = (q_3,1,L)$ $\quad C_4 \Rightarrow 0^3 1 0^3 1 0^3 1 0^2 1 0^1$

The complete code is given by,

$C_1 11 C_2 11 C_3 11 C_4$

$M = 0100100010100\,11\,000101010100\,11\,000100100101001\,1$

0001000100010010

∴ Code for $\langle M, 1011\rangle$ is given by

$= \langle 0100100010100\,11\,000101010100\,11\,000100100101001\,1$

$0001000100010010, 1011\rangle$.

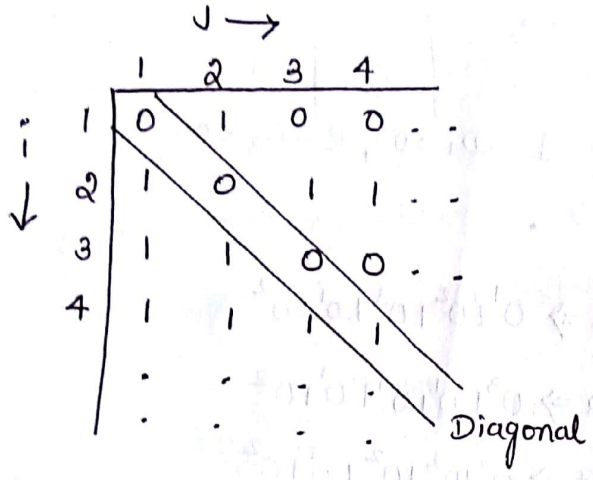
① Diagonalization Language (L_d).

It consist of all the strings w such that the TM represented by w does not accept the input w. L_d has no turing machine that accepts it.

⇒ Some integers do not belong to any TM. If W_i is not a valid TM code, then take the TM M_i to be the TM with one state and no transitions.

Definition ⇒ the diagonalization language L_d is the set of strings W_i, where W_i is not in $L(M_i)$.

L_d consists of all strings w such that the TM, M whose code is w does not accept the input w.



if (i,j) = 1, yes it is accepted

if (i,j) = 0, No it is rejected.

ith row ⇒ characteristic vector for the language $L(M_i)$

Where 1's in this row correspond to the member of this language.

Inorder to find L_d, Complement the diagonal.

Here diagonal value = 0001, After complementing value becomes 1110

Diagonalization ⇒ process of complementing the diagonal to construct the characteristic vector of a language that cannot be language that appears in any row.

Property: L_d is not recursively Enumerable.

Proof: Suppose L_d is accepted by some TM M defined by $L(M)$.

Since L_d is a language over alphabet $\{0,1\}$, there may be atleast one code for M, say i (ie) $M = M_i$.

Check whether W_i is in L_d,

By definition $L_d = \{ W_i \mid M_i \text{ does not accept } W_i \}$

Two possibilities,

(i) $W_i \in L_d$ ⇒ (i,j) entry is '0' and M_i does not accept W_i. But our assumption here it is TM M_i which accepts W_i. There is a contradiction.

(ii) $W_i \notin L_d$ ⇒ (i,j) entry is '1' and M_i accept W_i. But by definition of L_d, M_i does not accept W_i. So there is a contradiction.

Hence proved

Universal language (L_u)

Consider the binary string representation $\langle M, w \rangle$ such that M accepts w called as Universal language L_u.

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

L_u is the set of strings representing a TM and an input accepted by that TM. So there is a TM U called as Universal Turing machine.

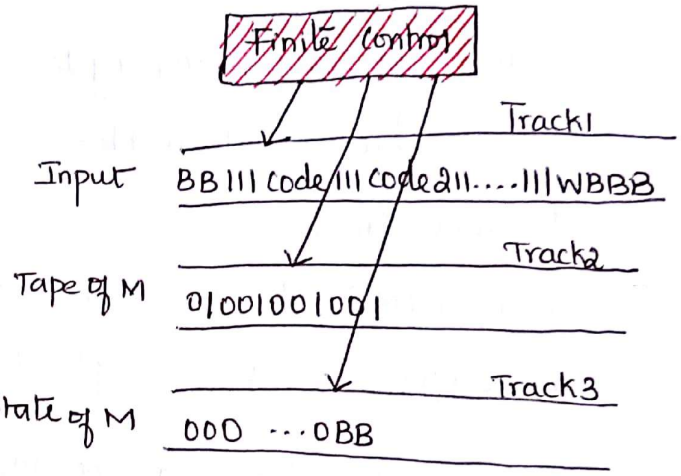
Theorem: L_u is recursively enumerable.

Proof: Inorder to prove this theorem, it is necessary to construct a TM U that accepts L_u. The turing machine U consists of a three track input tape.

1st track ⇒ Hold the input tape ($\langle M, w \rangle$)

2nd track ⇒ Tape symbols are written in unary form.

3rd track ⇒ represents states (Unary)



Operation:

1. First make sure that the code for M, is a legitimate code for some TM M. Otherwise it halts without accepting.
2. Initialize the 2nd tape with input w, inf its encoded form. Keep 0 the start state of M on the third tape and move the head of U's second tape to the first simulated cell.
3. If 0^i is the current state with 0^j the current i/p symbol appeared on track three and two respectively then U finds the corresponding transition of the form $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ on track 1 and replaces 0^i by 0^k and 0^j by 0^l
4. Move the head on tape two to the position corresponding to the value of M.
5. The Universal Turing Machine U accepts $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ if M has the transition of the form $\delta(0^i, 0^j) = (0^k, 0^l, 0^m)$. Otherwise M halts without accepting.

∴ Thus U simulates M and accepts w. Thus L_u is recursively

Rice Theorem :

All nontrivial properties of the RE languages are undecidable.
(ie) It is not possible to recognize the property by a TM.

A property is trivial if it is either empty which is satified by no language or is a all RE language, or else it is nontrivial.

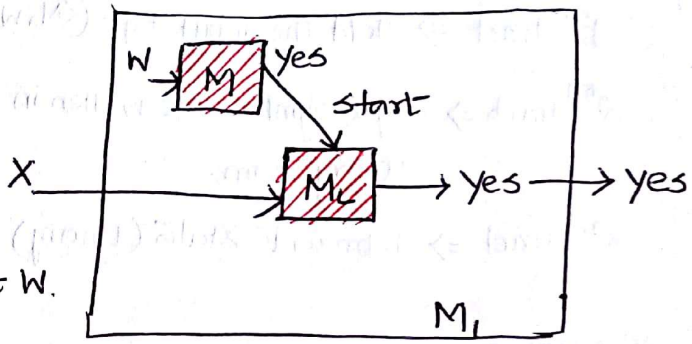
Theorem: Every non-trivial property of the RE language is undecidable.

Proof: Let $\mathcal{P}$ be a non-trivial property of the RE language.

Assume that ϕ is not in $\mathcal{P}$. Since $\mathcal{P}$ is nontrivial, There must be some non empty language L that is in $\mathcal{P}$.

Here M_1 => TM accepting L

L_u => Undecidable.



Rice Theorem.

Design of M_1 is

(i) if $L(M_1) \neq \phi$, if M does not accept W.

(ii) if $L(M_1) = L$, if M accepts W.

M_1 is a two-tape TM. one tape is used to simulate M on W and another tape is used to simulate M_L on input X to M_1.

TM, M_1 is constructed as follows,

=> M_1 ignores its input and simulates M on w, If M does not accept w, then M_1 does not accept X. If M accepts W, Then M_1 simulates M_L on x, accepting X if and only if M_L accepts X, thus M_1 either accepts ϕ or L depending on M.

=> It is possible to use the hypothetical $M_{\mathcal{P}}$ to determine if $L(M_1)$ is in $\mathcal{P}$. Since $L(M_1)$ is in $\mathcal{P}$ if and only if $\langle M, w \rangle$ is in L_u.
We have an algorithm for L_u, a contradiction.

Thus $\mathcal{P}$ must be undecidable.

Undecidable problems about Turing Machine

Reduction:

P_1 reduces to P_2 => If there is an algorithm used to convert instance of a problem P_1 to instance of a problem P_2 with the same answer, then it is said that P_1 reduces P_2.

Thus, P_1 is not recursive => P_2 cannot be recursive

P_2 is non-RE => P_1 cannot be RE.

So, any instance of P_1 that has "yes" answer can be turned into an instance of P_2 with a "yes" answer and every instance of P_1 with answer "No" answer can be turned into an instance of P_2 with "no" answer.

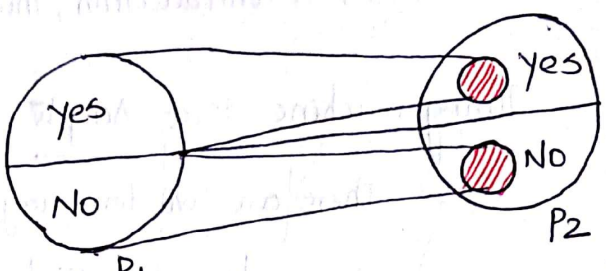


Fig: Reduction from P_1 to P_2.

A reduction can be defined as "A reduction from P_1 to P_2 is TM that accepts/takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape."

Theorem: If there is a reduction from P_1 to P_2 then.

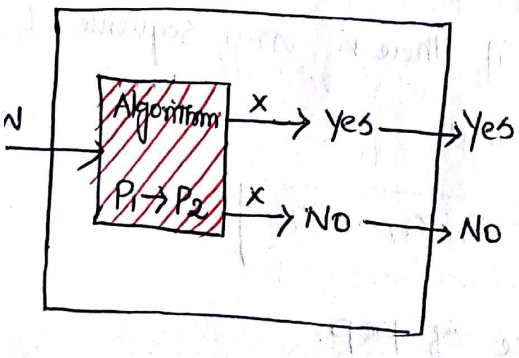
(i) If P_1 is undecidable, then so is P_2.

(ii) If P_1 is non RE, then so is P_2.

Proof:

(i) If P_1 is undecidable, then so is P_2.

Assume P_1 is undecidable. Let A be the algorithm which converts the instance of P_1 to instance of P_2.

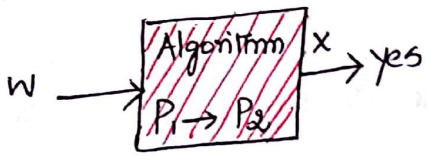


Suppose w be the instance of P_1, given to the algorithm A that converts w into an instance X of P_2.

o/p	Inference
Yes	X is in P_2
No	X is not in P_2

(ii) If P_1 is non-RE, then so is P_2.

Assume that P_1 is non-RE but P_2 is RE. since P_2 is RE, have an algorithm to reduce P_1 to P_2 that is there is a TM gives "yes" output if it is input is in P_2 but may not halt if it is input is not in P_2.



If W is in P_1, then X is in P_2, so the TM will accept W.

If W is in P_1, then X is not in P_2, so the TM may or may not halt and will ~~not~~ accept W.

This is a contradiction, thus if P_1 is non RE, then P_2 is non RE.

Turing machine that Accepts the Empty language.

There are two languages involving TM called L_e and L_{ne},

L_e – Empty language

L_{ne} – Non-empty language.

If $L(M_i) = \phi$, M_i does not accept any input, then W is in L_e.

If $L(M_i) \neq \phi$, language is called L_{ne}.

$$\therefore \boxed{L_e = \{M \mid L(M) = \phi\}}$$

$$\boxed{L_{ne} = \{M \mid L(M) \neq \phi\}}$$

Post- Correspondence problem

An instance of post - Correspondence problem (pcp) consists of two lists of strings over Σ.

$A_k = w_1, w_2, \ldots w_k$

$B = x_1, x_2, \ldots x_k$ for some integer k.

The instance of pcp has a solution if there is any sequence of integers $i_1, i_2, \ldots i_m$ with $m \geq 1$ such that

$$\boxed{w_{i1}, w_{i2}, w_{i3}, \ldots w_{im} = x_1, x_2, \ldots x_k}$$

is a solution to this instance of PCP

Pbm: let $\Sigma = \{0, 1\}$. let A and B be strings. Find the instance of PCP.

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

Solution:

Given $W = (1, 10111, 10) = (w_1, w_2, w_3)$

$X = (111, 10, 0) = (x_1, x_2, x_3)$

Take $M = 4$, Take this combination 2, 1, 1, 3

By concatenating string in this series.

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3$$

$$10111110 = 10111 \; 111 \; 0$$

∴ Instance of PCP = 2, 1, 1, 3.

Pbm: Does PCP with two lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$ have a solution?

Solution: Given $x = (b, bab^3, ba) = (x_1, x_2, x_3)$

$y = (b^3, ba, a) = (y_1, y_2, y_3)$

Take $M = 4$, Take the combination 2, 1, 1, 3

Check the string in this series.

$$x_2 x_1 x_1 x_3 = y_2 y_1 y_1 y_3$$

$$bab^3 \underbrace{bb\, ba} = bab^3 b^3 a$$

$$bab^3 b^3 a = bab^3 b^3 a$$

$$\text{List } x = \text{List } y$$

∴ Hence PCP has a solution for the given lists.

Modified PCP:

Given lists A and B of k strings each from Σ^*

$$A = w_1, w_2, \ldots w_k$$

$$B = x_1, x_2, \ldots x_k$$

It has the solution $i_1, i_2, \ldots i_r$ such that

$$w_1, w_{i_1}, w_{i_2}, \ldots w_{ir} = x_1, x_{i_1}, x_{i_2}, \ldots x_{ir}$$

Difference b/w the MPCP and PCP is that in the MPCP, a solution is required to be stored with the first string on each list.

Ex: Let $\Sigma = \{0, 1\}$. let A and B the list of string defined as.

i	ListA w_i	ListB x_i
1	1	10
2	110	0
3	0	11

Find the instance of MPCP.

Solution: MPCP is given by, $w_1, w_{i_1}, w_{i_2}, \ldots w_{ir} = x_1, x_{i_1}, x_{i_2}, \ldots x_{ir}$

$$110110 = 10110$$

Where the instance is given 1, 3, 2.

Problems: Construction of PCP from MPCP.

Consider the TM M and w = 01, where $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_3\})$ and δ is given by.

	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
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	-	-	-

Reduce the above problem to PCP and find the solution:

Solution: An instance of MPCP with two lists can be A and B

List A: #

List B: $\#q_101\#$

Solve

Rule	List A	List B	
1	#	$\#q_{1}01\#$	
2	0	0	
	1	1	
	#	#	
3.	$q_{1}0$	$1q_{2}$	from $\delta(q_{1},0) = (q_{2},1,R)$
	$0q_{1}1$	$q_{2}00$	from $\delta(q_{1},1) = (q_{2},0,L)$
	$1q_{1}1$	$q_{2}10$	
	$0q_{1}\#$	$q_{2}01\#$	from $\delta(q_{1},B) = (q_{2},1,L)$
	$1q_{1}\#$	$q_{2}11\#$	
	$0q_{2}0$	$q_{3}00\#$	from $\delta(q_{2},0) = (q_{3},0,L)$
	$1q_{2}0$	$q_{3}10\#$	
	$q_{2}1$	$0q_{1}$	from $\delta(q_{2},1) = (q_{1},0,R)$
	$q_{2}\#$	$0q_{2}\#$	from $\delta(q_{2},B) = (q_{2},0,R)$
4	$0q_{3}0$	q_{3}	
	$0q_{3}1$	q_{3}	
	$1q_{3}0$	q_{3}	
	$1q_{3}1$	q_{3}	
	$0q_{3}$	q_{3}	
	$1q_{3}$	q_{3}	
	$q_{3}0$	q_{3}	
	$q_{3}1$	q_{3}	
5.	$q_{3}\#\#$	#	

For the input $w = 01$, computation is

$\vdash q_{1}01 \vdash 1q_{2}1 \vdash 10q_{1}B \vdash 1q_{2}01 \vdash q_{3}101 \vdash q_{3}01 \vdash q_{3}1 \vdash q_{3}$.

Since q_3 is final state, a solution to the instance of MPCP is written from list A and list B as follows,

$\#q_{1}01\#1q_{2}1\#10q_{1}\#1q_{2}01\#q_{3}101\#q_{3}01\#q_{3}1\#q_{3}\#\#$.

Properties of Recursive and Recursively enumerable languages

Property 1: The union of two recursively enumerable language is recursively enumerable.

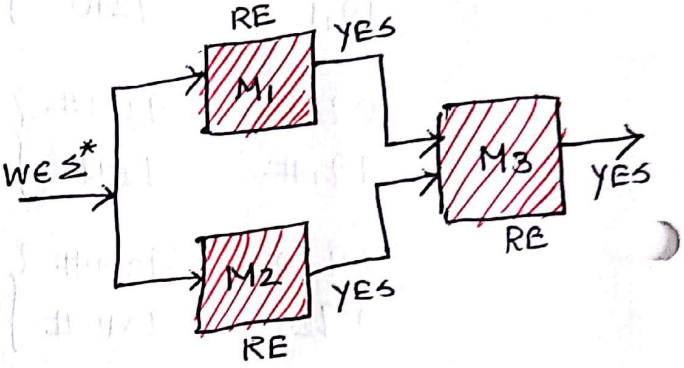
Proof: Let L_1 and L_2 be two recursively enumerable language accepted by the Turing machine M_1 and M_2.

⇒ If a string $w \in L_1$ then M_1 return "YES", accepting the input string, else loops forever. Similarly if a string $w \in L_2$ then M_2 return "YES", else loop forever.

⇒ Here, the output of M_1 and M_2 are written on the input tape of M_3.

⇒ The TM, M_3 returns "YES", if at least one of the outputs of M_1 and M_2 is "YES".

⇒ M_3 decides on $L_1 \cup L_2$ that halts with the answer "YES" if $w \in L_1$ or $w \in L_2$.



⇒ Else M_3 loops forever for both M_1 and M_2 loop forever.

Hence, the Union of two recursively enumerable languages is also recursively enumerable.

Property 2: Intersection of two recursively enumerable language is recursively enumerable.

Proof: Let L_1 and L_2 be two recursively enumerable language accepted by the TM M_1 and M_2.

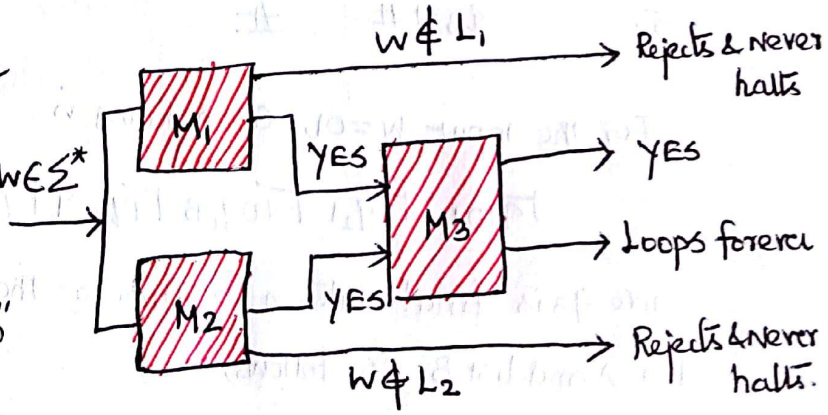
⇒ If a string $w \in L_1$ and then M_1 returns "YES" accepting the input. Else it will not halt after rejecting $w \notin L_1$.

⇒ Similarly if a string, $w \in L_2$ the M_2 returns "YES" else reject it and loop forever.

⇒ Here o/p of M_1 and M_2 are written the i/p tape M_3.

⇒ M_3 returns "YES" if both the outputs of M_1 and M_2 is "YES".

⇒ If at least one of M_1 or M_2 is "NO" it rejects 'w' and never halts.



⇒ Thus M_3 decides on $L_1 \cap L_2$ that halts if and only if $w \in L_1$ and $w \in L_2$. Else

Property3: A language is recursive if and only if both it and its complement are recursively enumerable.

Proof:

Let L and $\bar{L}$ be two recursively enumerable language accepted by TM's M_1 and M_2.

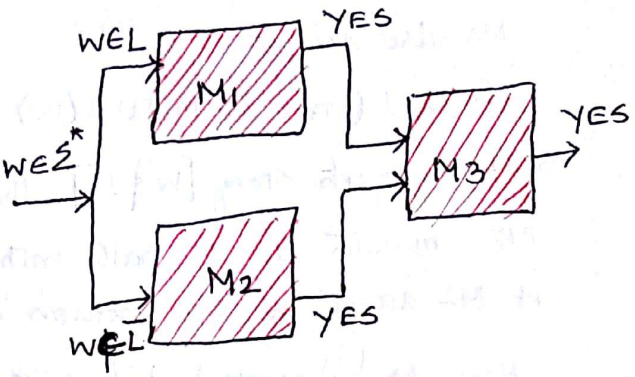
⇒ If a string $w \in L$, it is accepted by M_1 and M_2 halts with answer "YES". Else M_1 enters into infinite loop.

⇒ If a string $w \in \bar{L}$ [$w \notin L$], then it is accepted by M_2 and M_2 halts with answer "YES", otherwise M_2 loops forever.

⇒ From the diagram, if $w \in L$, then M_1 accepts w and halts with "YES".

⇒ If $w \notin L$, then M_2 accepts w and halts with "YES".

⇒ Since M_1 and M_2 are accepting the complements of each other, one of them is guaranteed to halt for every input, $w \in \Sigma^*$



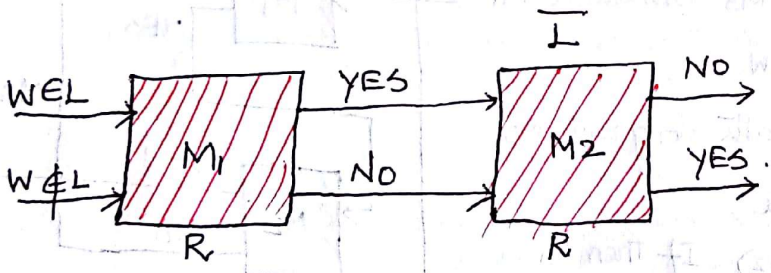
Hence M_3 is a Turing machine that halts for all strings.

Property4: The complement of a recursive language is recursive.

Proof:

Let L be a recursive language accepted by TM, M_1.

Let $\bar{L}$ be a recursive language accepted by TM, M_2.



Let $w \in L$, then M_1 accepts w and halts with "YES".

M_1 rejects w if $w \notin L$ and halts with "NO".

M_2 is activated once M_1 halts.

M_2 works on $\bar{L}$ and hence if M_1 returns "YES", M_2 halts with "NO".

If M_1 returns "NO", then M_2 halts with "YES".

Thus for all w, where $w \in L$ or $w \notin L$ halts with either "YES" or "NO".

Property 5: The union of two recursive language is recursive.

Proof:

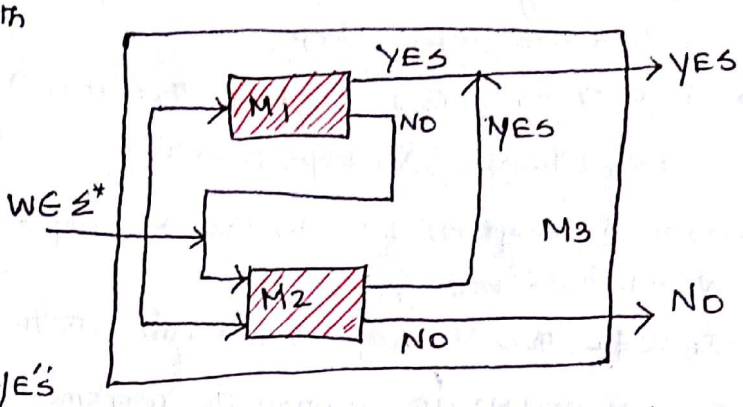
Let L_1 and L_2 be two recursive languages that are accepted by the TM's M_1 and M_2 given by $L(M_1) = L_1$, $L(M_2) = L_2$

⇒ The TM, M_3 first simulates M_1 with the input string w.

⇒ If $w \in L$, then M_1 accepts and thus M_3 also accepts.

Since $L(M_3) = L(M_1) \cup L(M_2)$.

⇒ If M_1 rejects string $[w \notin L_1]$, then M_3 simulates M_2. M_3 halts with "YES" if M_2 accepts 'w' else returns "NO".



Hence M_3, M_2, M_1 that halt with either YES or NO on all possible inputs.

Property 6: The intersection of two recursive language is recursive.

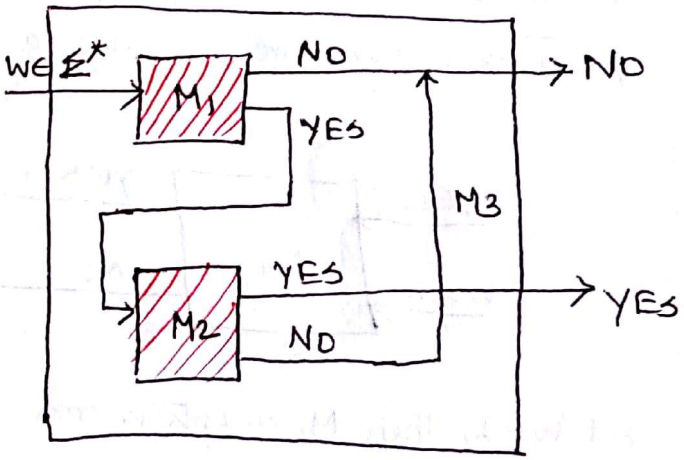
Proof: Let L_1 and L_2 be two recursive languages accepted by M_1 and M_2

Where $L(M_1) = L_1$

$L(M_2) = L_2$.

⇒ The turing machine M_3 simulates M_1 with the input string w.

⇒ If $w \notin L_1$, then M_1 halts along with M_3 with answer "No". Since, $L(M_3) = L(M_1) \cap L(M_2)$. If Then M_1 accepts with the answer "YES" and M_3 simulates M_2.



⇒ If M_2 accepts the string, Then the answer of M_2 and M_3 are "YES" and halts. Else M_2 and M_3 halt with answer "No".

Thus, the intersection of two recursive language is recursive.

Introduction to Computational Complexity

Definition:

Complexity ⇒ It defines whether the given problem is easy to solve or hard to solve.

Complexity of problems is classified into two types.

1. Space complexity ⇒ Amount of memory space required by the algorithm/problem to complete its computation.
2. Time complexity ⇒ Amount of time required to run the program of the problem.

Time and space complexity of Turing Machine

Time complexity (T_T)

⇒ The number of transitions/moves taken by the TM to compute the input is called the time complexity of the TM.

⇒ Let T be a Turing machine, then the time complexity is a function $T_T(n)$ is the maximum no. of transitions for computing the input, n using T.

T_T depends on the input size of the problem.

When 'n' is larger, T_T will be large; when 'n' is small T_T will be less.

Space Complexity (S_T)

⇒ Space complexity is the number of tape-cells required to process the turing machine, T for computing the input n.

⇒ $S_T(n)$ is the maximum number of tape-cells used by T to process an input of length 'n'.

$S_T(n)$ depends on the number of inputs stored on T.

If n is large, then more cells are required to store and process.

If n is small, then limited number of cells on the tape are required by the TM to store and process.

Complexity types / Efficiency types

(i) Best case ⇒ The minimum number of time / space required by the TM to compute the input.

(ii) Average case ⇒ the average number of time / space required by the TM to compute the input.

(iii) Worst case ⇒ The maximum number of time / space required by the TM to process the input.

Complexity of Non-deterministic TM.

Time Complexity ⇒ $T_T(n)$ is a function that refers to the maximum time taken by the NTM to compute the input X, where $|x| = n$.

Space Complexity ⇒ $S_T(n)$ is the maximum amount of tape-cells required by the TM to process X with $|x| = n$.

The time and space complexity is undefined if the NTM loop forever.

Complexity classes: class P and class NP.

Class P:

⇒ P refers to the class of problems that can be solved in polynomial time.

⇒ polynomial time algorithms are efficient ones that can be solved more rapidly.

⇒ these are also referred as tractable problems.

⇒ Example: Searching an element in an ordered list, sorting elements in a list, finding minimum spanning tree of a graph.

Class NP:

⇒ NP refers to the class of problems that are solved by non-deterministic polynomial time.

⇒ These types of NP problems are known as intractable problems.

⇒ Example: Towers of Hanoi, Travelling salesman problem, Graph coloring problem, Hamiltonian circuit problem, Satisfiability problem etc..

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 problems.

⇒ A NP-complete problem can be transformed into any other in polynomial time.

NP - Hard problem:

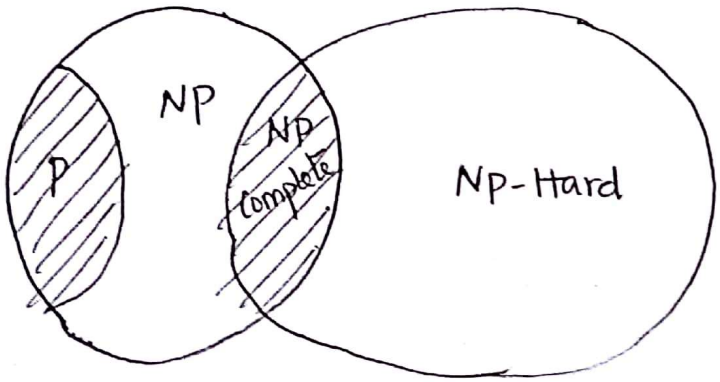
⇒ A problem is said to be NP hard if there exists an algorithm for solving it and it can be translated into one for solving another NP-problem.

⇒ A problem P_1 is NP-hard if

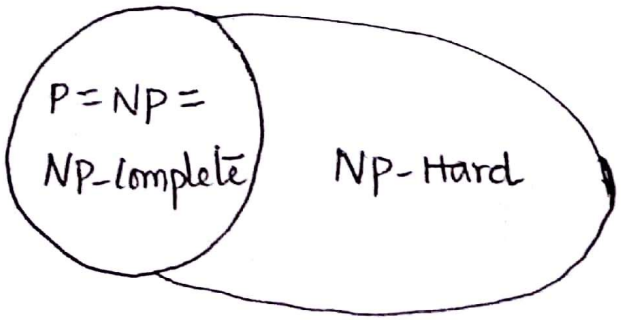
- the problem is an NP class problem.
- For any other problem P_2 in NP, there is a polynomial reduction of L_2 to L_1

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

Conclusion:



$P \neq NP$



$P = NP$