UNIT-5

UNDECIDABILITY AND ADVANCED TURING MACHINES

Types of Turing machine

Various types of TM's are

1. Turing machine with two dimensional tapes
2. Turing machine with multiple tapes
3. Turing machine with multiple heads
4. Turing machine with finite tape
5. Non deterministic turing machine.

It is observed that computationally all thes TM are equally powerful. That means one type can compute the same that other can. However the efficiency of computation may vary.

1) TM with two-dimensional tapes

This type of TM has two finite controls

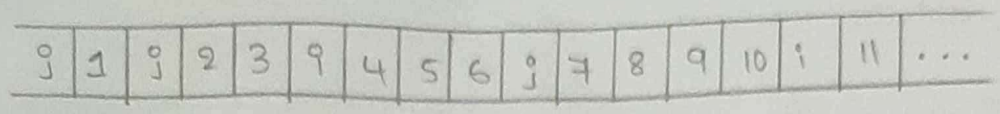
1) Read / Write head
2) Two dimensional tape

This tape has infinite extension to right and down. It is divided into small squares formed due to Corresponding rows and columns.

→ TM with one dimensional tape is equally powerful to that of two dimensional tape.

i ↓ / j →						
1	2	6	7	15	16	-
3	5	8	14	17	26	-
4	9	13	18	25	-	-
10	12	19	24	-	-	-
11	20	23	-	-	-	-
21	22	-	-	-	-	-
-	-	-	-	-	-	-

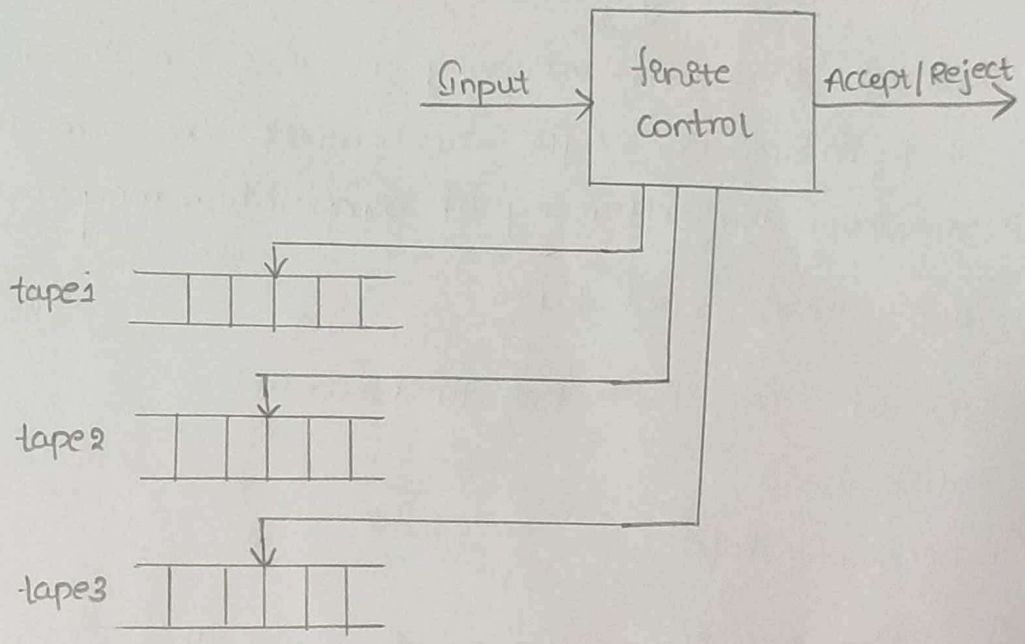
Two dimensional Tape



The head of Two dimensional tape moves one square up, down, left or right.

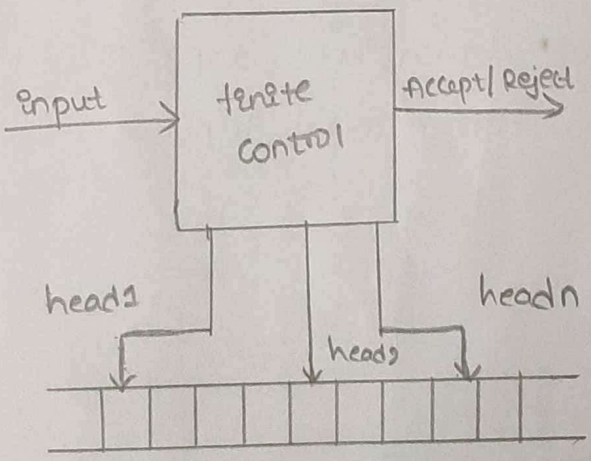
2) TM With multiple tapes

This is a kind of TM with one finite control and with more than one tape having its own read / write heads.



3) TM With multiple heads

The Tm with multiple heads can be shown below

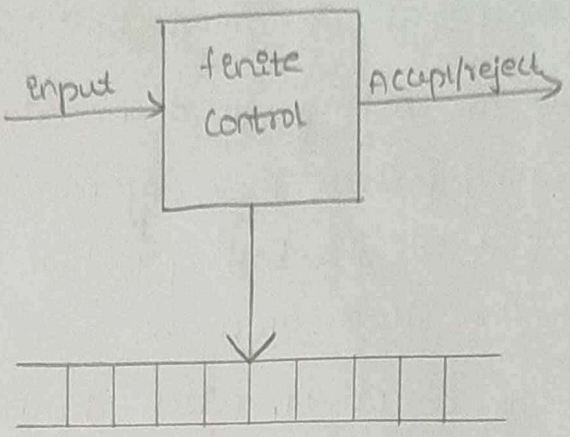


TM with multiple heads

There are n heads, but in any state, only one head can move. This type of TM are as powerful as one tape TM

4) Turing machine with infinite tape.

This is a TM that have one finite control and one tape which extends infinitely in both directions.



This type of TM's are as powerful as one tape TM's whose tape has a left end.

5) Non-determenistic turing machine

The concept of non-deterministic TM is similar to the NFA.

→ for any state and any i/p symbol it can take any action from a set rather than a definite predetermined action.

for example, the language like $L = \{ww \mid w \in (a+b)^*\}$ can be shown by non deterministic TM.

→ The non determenistic TM is as powerful as determinstic TM.

HALTING PROBLEM

→ This is a famous undecidable problem of TM. To state halting problem we will consider the given configuration of a TM.

→ The O/P of TM can be

(i) Halt: The machine starting at this configuration will halt after a finite number of states.

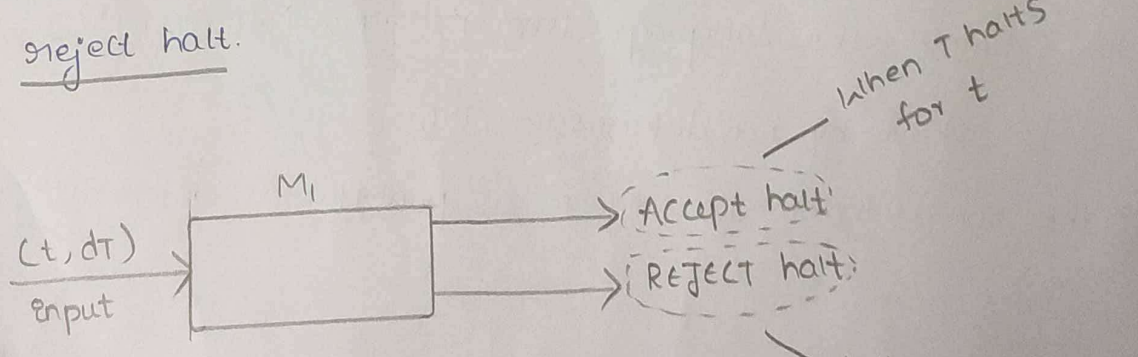
(ii) No Halt:- The machine starting at this configuration never

reaches a halt state, no matter how long it runs.

Now the question arises based on these two observations, given any functional matrix, i/p data tape and initial configuration, then is it possible to determine whether the process will ever halt? This is called halting problem.

That means we are asking for a procedure which enable us to solve the halting problem for every pair (machine, tape). The answer is "no". That is the halting problem is unsolvable. Now we will prove how it is unsolvable.

→ Let, there exists a TM M_1 which decides whether or not any computation by a TM T will ever halt when a description d_T of T and tape t of T is given. [That means input to machine M_1 will be (machine, tape)]. Then for every input (t, d_T) to M_1 if T halt for input t, M_1 also halts which is called accept halt. Similarly if T does not halt for i/p t then M_1 will halt which is called reject halt.

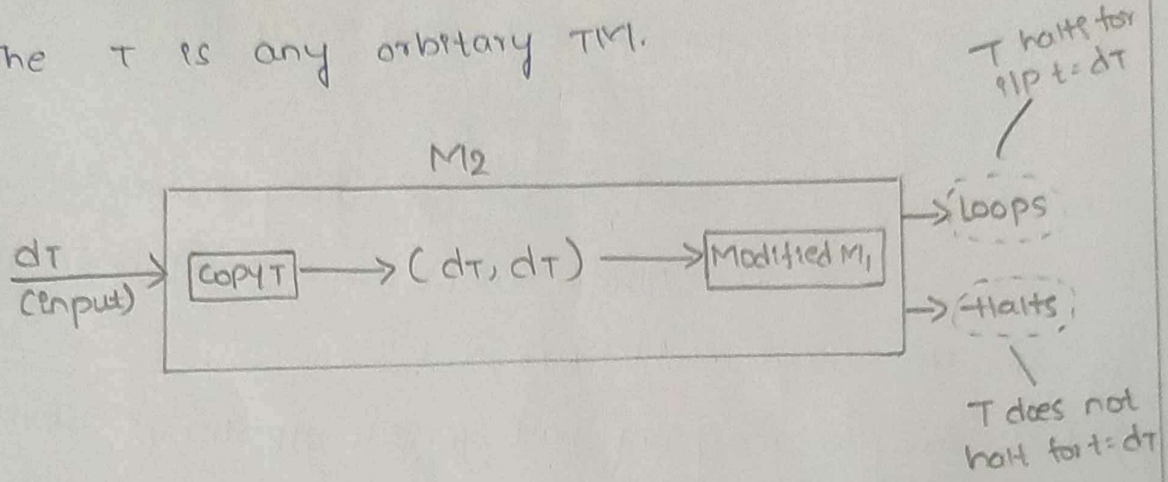


→ Now, we will consider another TM M_2 which takes an i/p d_T.

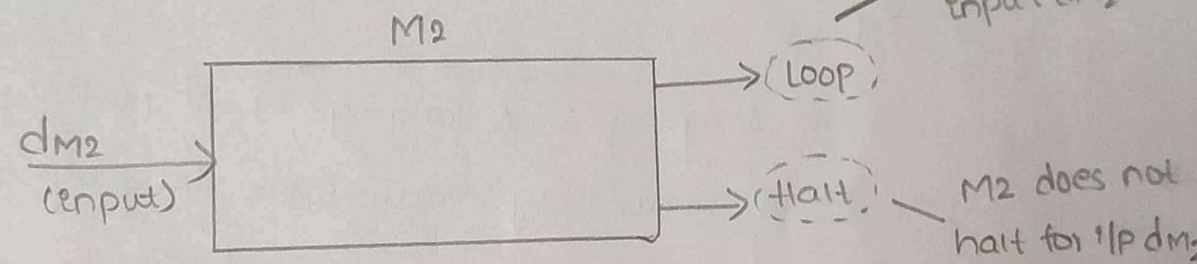
It first occupies d_T and duplicate d_T on its tape and then this duplicated tape information is given as i/p to Machine M_1.

But machine M_1 is a modified machine with the modification that whenever M_1 is supposed to reach an accept halt, M_2 loops forever. Hence behaviour of M_2 is as given. It loops if T halts for i/p $t = d_T$ and halts if T does not halt for $T = d_T$.

→ The T is any arbitary TM.



As M_2 itself is one TM we will take $M_2 = T$. That means we will replace T by M_2 from above given machine



Thus machine M_2 halts for i/p d_{M_2} if M_2 does not halt for i/p d_{M_2}. This is a contradiction. That means a machine M_1 which can tell whether any other TM will halt on particular i/p does not exist. Hence halting problem is unsolvable.

Recursive language: A language 'L' is said to be recursive if there exists a TM which accept all the strings in 'L' and reject all the strings not in 'L'.

→ The TM will halt every time and give an answer (accept or reject) for each and every string i/p.

Recursively Enumerable language:

→ A language 'L' is said to be recursively Enumerable language if there exists a TM which accepts (and therefore halt) for all the i/p strings which are in 'L'

→ But may or may not halt for all i/p strings which are not in 'L'.

Decidable language: 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 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 TM for that language.

Undecidable problem about Turing Machine

1) Reduction
2) Empty and non-Empty language
3) Rice's Theorem.

Reduction: It is a technique in which if a problem P1 is reduced to a problem P2 Then any solution of P2 solves P1.

→ In general we have an algorithm to convert an instance of a problem P1 to instance of a problem P2 that have the same answer then it is called P1 reduces P2.

→ Hence if P1 is not Recursive, then P2 is also not recursive

→ if P1 is not recursively enumarable, then P2 is also not recursively enumerable.

Theorem: if P1 is reduced to P2 then,

i) if P1 is undecidable then P2 is also undecidable

ii) if P1 is non-RE then P2 is also non-RE

Proof: (i) consider an instance W of P1. Then construct an algorithm such that the algorithm takes instance W as i/p and convert it into another instance x of P2.

→ Then apply that algorithm to check whether x is in P2, if algorithm answers "yes" then that means x is inP2.

→ Hy we can also say that W is in P1.

→ Since we have obtained P2 after reduction of P1. Similarly if algorithem answers "no" then x is not in P2, that also means w is not in P1.

This proves that if P1 is undecidable then P2 is also undecidable

(ii) We assume that P1 is non-RE but P2 is RE. Now construct an algorithm to reduce P1 to P2, but by this algorithm P2 will be recognized. That means there will be a TM says that "yes" if the i/p is P2 but may or may not halt for the i/p which is not in P2.

→ Apply a TM to check whether x is in P2. If x is accepted that also means w is accepted.

→ This procedure describes a TM whose language is P1 if w is in P1 then x is also in P2 and if w is not in P1 then x is also not in P2.

This proves that if P1 is non-RE then P2 is also non-RE.

(iii) Empty and non empty language:

There are two types of languages empty and non-empty Let Le denotes an empty language and Lne denotes non-empty language.

Let w be a binary string and M_i be a TM.

If $L(M_j) \neq \phi$ then M_i does not accept any i/p then w is in Le.

Similarly if $L(M_i)$ is not the empty language then w is in Lne. Thus we can say that,

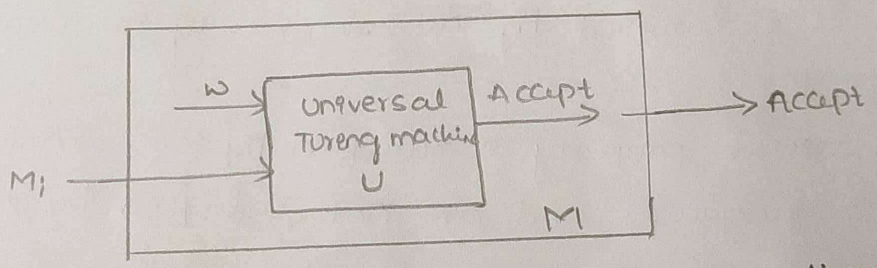
$Le = \{M_i \mid L(M) = \phi\}$ $Lne = \{M \mid L(M) \neq \phi\}$

Both Le and Lne are complument of one another

Theorem: Lne is recursively enumerable.

proof:- To prove this, we simply need to prove that there exists some TM which accepts Lne. So do this we need to construct non deterministic TM M that can be converted to determenistic TM.

The TM M accepts another TM M_i as i/p. With the non-determenistic capability M can guess the i/p w that can be accepted by M_i. If M_i accepts w then M also accepts the input M_i



Thus if at all M_i accepts even one i/p, the M will guess that streng and will accept M_i. But if $L(M_i) = \phi$ that means M can make no guess about the i/p streng and therefore can not accept M_i.

This proves that there should be such a TM whose language $L(M) \neq \phi$, thus $L(M) = Lne$.

Rice's Theorem:-

Theorem: Every non trivial property of Rt language is undecidable

proof: Rice theorem states that any non-trival Semantic property of a language which is recognized by a TM

is undecidable. A property, P is the language of all TM that satisfy that property.

formal definition: If P is a non-trivial property, and the language holding the language of all TMs that satisfy the property. property L_P, is recognized by TM M, then $L_P = \{\langle M \rangle | L(M) \in P\}$ is undecidable.

→ property of language P is simply a set of languages. If any language belongs to P ($L \in P$), it is said that L satisfies the property P.

→ A property is called to be trivial if either it is not satisfied by all recursively Enumerable languages.

→ (Or) if it is satisfied by all recursively Enumerable language.

→ A non-trivial property is satisfied by some recursively Enumerable languages and are not satisfied by others. formally speaking, in a non-trivial property, where $L \in P$, both the following properties hold:

property 1 - There exists TMs M_1 and M_2 that recognize the same language, i.e., either $(\langle M_1 \rangle, \langle M_2 \rangle \in L)$ or $(\langle M_1 \rangle, \langle M_2 \rangle \notin L)$

property 2 - There exists Turing machines M_1 and M_2 where M_1 recognizes the language while M_2 does not, i.e, $\langle M_1 \rangle \in L$ and $\langle M_2 \rangle \notin L$.

Proof : Suppose a property P is non-trivial and $\varphi \in P$.

Since, P is non trivial, at least one language satisfies P,

i.e., $L(M_0) \in P$, $\exists$ Turing machine M_0.

Let, w be the input in a particular instant and N is a Turing machine which follows -

on input x

→ Run M on w

→ if M does not accept (or doesn't halt), then do not accept x (or do not halt)

→ if M accepts w then run M_0 on x. If M_0 accepts x, then accept x.

A function that Maps an instance ~~ATM = <w, m>~~

$A_{TM} = \{<M, w> \mid M \text{ accepts input } w\}$ to a N

Such that

→ if M accepts w and N acapts the same language as M_0, then $L(M) = L(M_0) \in P$

→ if M does not accept w and N accepts φ, Then $L(N) = \varphi \notin P$

Since A_{TM} is undecidable and it can be reduced to L_P, L_P is also undecidable.

Decidable properties of Formal Language

Decidable properties	RL	CFL	CSL	RCL	REL
1 Membership $w \in L(m)$	✓	✓	✓	✓	×
2. Emptyness $L = \phi$?	✓	✓	×	×	×
3. finiteness $\|L\| \leq n$	✓	✓	×	×	×
4. Completness $L = \Sigma^*$	✓	✓	×	×	×
5. Equalness $L_1 \stackrel{?}{=} L_2$	✓	×	×	×	×
6. Complement $L^c = \Sigma^* - L$	✓	×	×	×	×

✓ — decidable

× — undecidable.

POST'S correspondence problem (PCP)

The undecidability of strings is determined with the help of post's correspondence problem (PCP).

→ "The post's correspondence problem consists of two lists of strings that are equal length over the input Σ.

The two lists are $A = w_1, w_2, w_3, \ldots w_n$

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

then there exists a non-empty set of strings

$i_1, i_2, i_3, \ldots i_n$ such that

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

To solve the PCP we try all the combination of $i_1, i_2, i_3 \ldots i_n$ to find the $w_i = x_i$ then we say that PCP has a solution.

Ex: Consider the correspondence system as given below

$A = (1, 0, 010, 11)$ and $B = (10, 10, 01, 1)$. The i/p set is $\Sigma = \{0, 1\}$. Find the solution.

Sol: A solution is 1 2 1 3 3 4, that means

$w_1, w_2 w_1, w_3 w_3 w_4 = x_1 x_2 x_1 x_3 x_3 x_4$

$10101001011 = 10101001011$

$|A| = 4 \cong |B| = 4$

Ex②: Obtain the solution for the following system of post's correspondence problem

$A = \{100, 0, 1\}$ $B = \{1, 100, 00\}$

The input symbols of A and B are same

$\Sigma = \{0, 1\}$ = $\Sigma = \{0, 1\}$

$|A| = 3 \cong |B| = 3$

The solution is 1 3 1 1 3 2 2.

w_1	w_3	w_1	w_1	w_3	w_2	w_2
100	1	100	100	1	0	0

x_1	x_3	x_1	x_1	x_3	x_2	x_2
1	00	1	1	00	100	100

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

$100\,1\,100\,100\,1\,00\,0\,0 = 1001100100100$

ex③: Obtain the solution for the following system of post's correspondence problem $A = \{ba, abb, bab\}$

$B = \{bab, bb, abb\}$

Now to consider 1, 3, 2 the streng bababbb from set A and bababbb from set B thus the two strengs obtained are not equal. As we can try various combinations from both the sets to find the unique sequence but we could not get such a sequence. Hence there is no soln for this system.

ex④: Obtain soln for the following correspondence system

$A = \{ba, ab, a, baa, b\}$, $B = \{bab, baa, ba, a, aba\}$

The i/p set is $\{a, b\}$

$w_1 w_5 w_2 w_3 w_4 w_4 w_3 w_4 = x_1 x_5 x_2 x_3 x_4 x_4 x_3 x_4$

The soln give a unique string

bababaabaaabaa.

ex⑤: Does pcp with two lists $x = (b, bb^3, ba)$ and $y = (b^3, ba, a)$ have a soln?

Now we have to find out such a sequence that strings formed by and y are identical. Such a sequence is 2, 1, 1, 3. Hence from x and y list

$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3$

$bab^3 b\ b\ ba \qquad ba\ b^3 b^3 a$

ex⑥: find whether the pcp $P = \{(10, 101), (011, 11), (101, 011)\}$ has a match. Give the soln.

let $P = \{(10, 101), (011, 11), (101, 011)\}$

$w_1 = \{10, 011, 101\}$

$w_2 = \{101, 11, 011\}$

The pcp has a solution if $w_{1i} = w_{2j}$,

$w_{12} = w_{23}$, $w_{13} = w_{21}$

but $w_{11} \neq w_{22}$

Hence we can not find any string $w_{1i} = w_{2j}$.

Hence this pcp has no solution.

Modified post Correspondence problem (MPCP):

The Modified post Correspondence problem (MPCP) is just like PCP except that we specify both the set of tiles and also a special tile. Matches for MPCP have to start with the special tile.

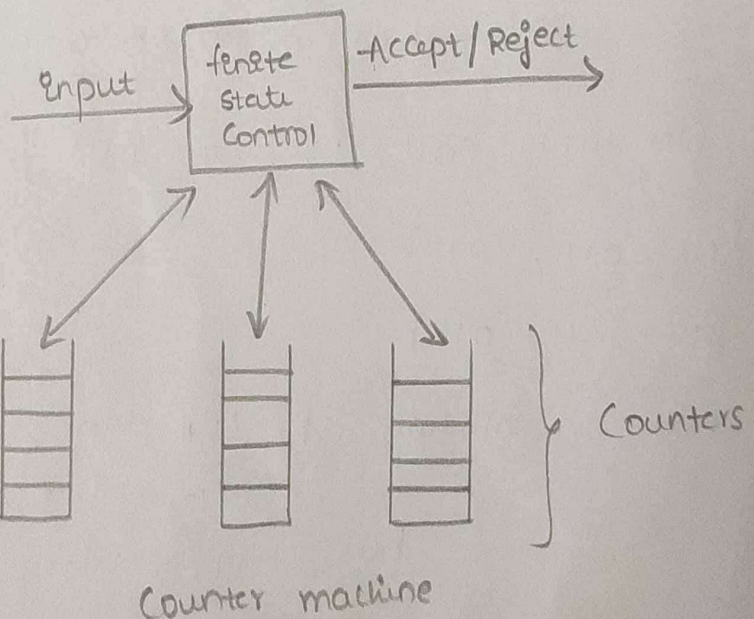
(or)

In MPCP the solution must starts from 1.

Ex:- 123, 113 .. etc.

Counter Machine

A Counter machine can be represented by following model-



Counter machine

There are two ways of representing Counter machine.

1) The Counter machine is similar to a multistack TM, But the only difference blw them is that in place of each stack there is a Counter. The Counter obtain non negative integers. Each move of Counter machine

depends on its state, input symbol. In one move counter machine can:

i) change state

ii) Add or subtract 1 from any of its counters.

The negative counters are not allowed at all.

2) The counter machine is similar to restricted multi stack machine. These restrictions are -

i) There are only two stack symbols: Z_0 and X

ii) The Z_0 is the bottom of stack marker. It is initially on each stack.

iii) Replace Z_0 only by string of the form $X^i Z_0$ where $i \geq 0$.

iv) Replace X only by X^i for $i \geq 0$. i.e. The Z_0 appears on the bottom of each ~~stack~~ and all other stack symbols are X.

There are two important observations about the counter machine.

1) Every language acapted by a counter machine is recursively enumerable.

2) Every language accepted by one counter machine is a context free language

Non-Recursive enumerable language

Universal Turing machine:

→ A universal Turing machine, M_u is an automata that, given as input the discription of any TM M and a string w can simulate the computation of M for i/p w.

→ TO construct such M_u, we consider a turing machine without loss of generality assume that

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, q_1, B, q_2)$$

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

q_1 = initial state

q_2 = final state

$\{0,1,B\} \in \Gamma$ are represented as $\{a_1, a_2, a_3\}$

→ Directions left or right are represented as

D_1 and D_2 respectively.

→ The transmitions of TM are encoded in special binary representation where each symbol is represented by 1

for ex: if there is a transition

$$\delta(q_i, a_j) = (q_k, a_l, D_m)$$

then the binary representation for the transition is $0^i10^j10^k10^l10^m$

transitions $t_1, t_2 \ldots t_n$

$$111\ t_1\ 11\ t_2\ 11\ t_3\ 11 \ldots 11\ t_n\ 111$$

Note: transitions need not be in any particular order.

→ If a string has to be verified then the problem is represented as a tuple $\langle M, w \rangle$ where M is definition of TM and w is i/p string.

Conside an example:

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

have moves defined as

$$\delta(q_1, 1) = (q_3, 0, R)$$
$$\delta(q_3, 0) = (q_1, 1, R)$$
$$\delta(q_3, 1) = (q_2, 0, R)$$
$$\delta(q_3, B) = \{q_3, 1, L)$$

Give the problem representation for the string w=1011

Sol: let the binary representation for

State : $\{q_1, q_2, q_3\}$ be $\{0, 00, 000\}$

alphabet : $\{0, 1, B\}$ be $\{0, 00, 000\}$

directions : $\{L, R\}$ be $\{0, 00\}$

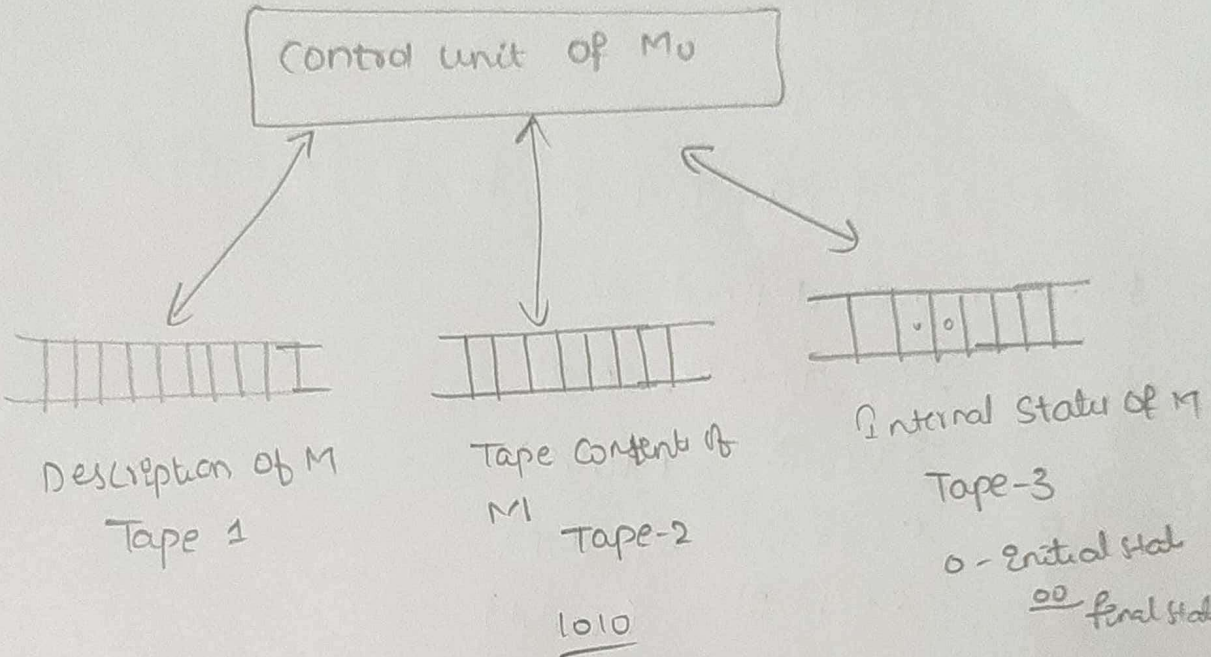
Transistions are represented as follows.

	Binary representation
$\delta(q_1, 1) = (q_3, 0, R)$	0100100010100
$\delta(q_3, 0) = (q_1, 1, R)$	0001010100100
$\delta(q_3, 1) = (q_2, 0, R)$	0001001001010 0
$\delta(q_3, B) = (q_3, 1, L)$	0001000100010010

∴ The problem representation $\langle M, 1011 \rangle$ is

(111) 0100100010100111000101010010011000100 1001010011100010001000100 10 (111) 1011

The following diagram shows organizations of U_{TM} which has a control unit and three tapes



non Recursively Enumerable language:-

If a language is not represented by Turing machine with / without halting then it is called non-REL

Ex:- Diagonalization

→ The language that is not accepted by any TM this proves the diagonalization. is not Recursively Enumerable

Σ^* is countable 2^{Σ^+} is uncountable.

	∈	a	b	aa	ab	ba	bb	aaa	aab	....
1)	1	1	0	0	0	0	0	0	0	
2)	0	1	1	1	0	0	0	0	0	
3)	1	1	0	1	0	0	0	0	0	

11 0

001 → This language is should be Countable as we assumed but it will not present in any of machine so it is Contraction. so it can be said to uncountable

$L_1 = \{\epsilon, a\}$

$L_2 = \{a, b\}, L_3 = \{\epsilon, a, aa\}$

undecidable of problems

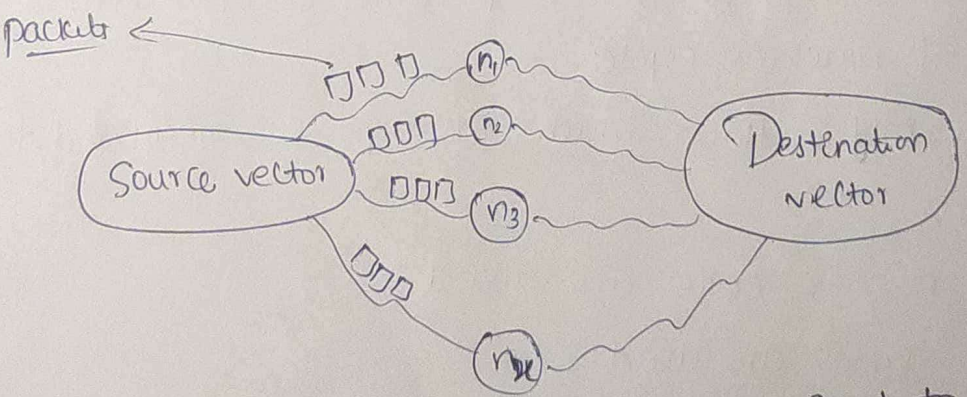
A problem p is said to be decidable if there exists a tureng machine to represent the problem or if there exists an algorethm to find the solution to the problem.

1) ambiguity in CFG
2) post correspondence problem (PCP)
3) Vertex vesting in a path
4) Code optimezation

Code optimization :- It is an undecidable problem we cannot say that a partaular piece of code Optimized code (in terms of cost space time)

Vertex vesting problem :-

In a NIw of Computer's a packet travelleng from source to Destenation may or may not visit a particular node in a path



ambiguity in CFG :- A Grammer G is said to be ambigeous Grammer if it generates more than one different parse for any string.

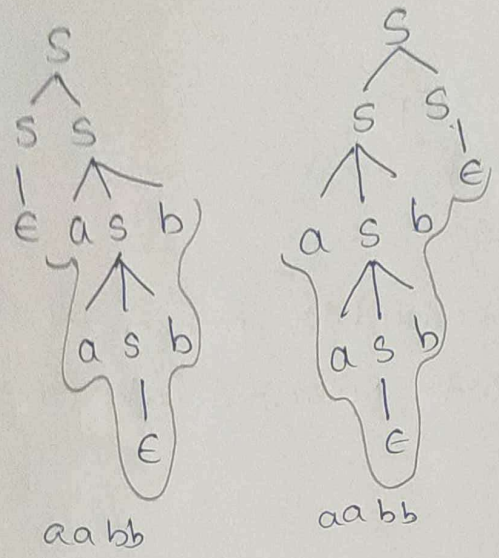
→ It is a trial and error method.
→ There exists no particular algorithm or tureng machine to prove the ambiguity in a grammer (undecidable problem)

Ex: $S \rightarrow asb \mid ss$

$S \rightarrow \epsilon$

for string "aabb" the above grammer generates two

Parse tree



Complexity classes

→ Time complexity :- How long computation takes to Execute. In T.M this could be measured as no. of moves which are required to perform computation.

→ Number of machine cycle.

Space Complexity :- How much storage is required for computation.

→ In TM, no. of cells are used.

→ no. of bytes are used.

Types of Complexity classes

1) P-class: Set of decision problem is solvable in polynomial time or in the class P.

If there exists an algorithm A such that

→ A. takes instance of D as i/p

→ A always o/p's the correct answer "yes" or "No".

→ There exists a polynomial p such that the execution of A on I/P of size n always terminate on p(n) steps

Eg:- The minimum spanning tree problem is in class p
Kruskal's algorithm

The class p is often considered as synonymous with the class of computationally feasible problem although in practice this is somewhat unrealistic.

NP class :-

A decision problem is non determenstically polynomial time solvable or in the class NP if there exists an Algorithm A such that -

→ There exists a polynomial p such that for each potential witness of each instance of size n of D. there exists an algorithm A takes at most p(n) steps

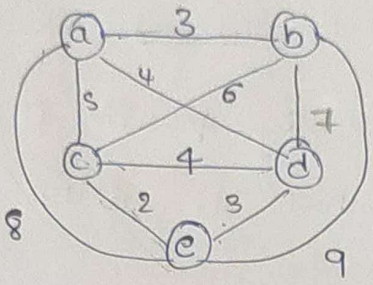
→ Think of a non-deterministic computes as a computer that magically guesses a solution, then has to verify that it is correct.

- If soln exist, Compute always guesses it
- One way to imagine it: a parallel computer that can freely spawn an infinite no. of processes
  - → One process work on each possible solution
  - → All processes attempt to verify that their solution works
  - → If a process finds it has a working solution ☺

So NP = problem verifable in polynomial time

→ Every problem in this class can be solvable in exponential time using exhaustive search.

Eg:- Travelling sales man problem.

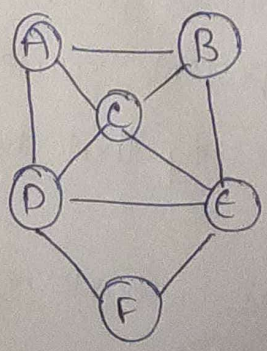


Routes	distance
a b c d e a	24
a b c e d a	19
a b d c e a	24
a b d e c a	16

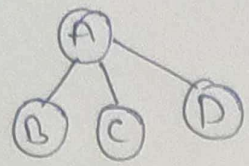
NP Complete : NP Complete are the subset of the class NP

It is the set of all decision problem whose solution can be solved & verified in polynomial time on a non deterministic TM is called NP Complete.

Ex:- hamelton cycle problem (starting from starting point and writing adjacent vertices as branches.

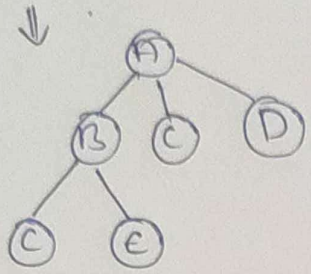


A ⇒

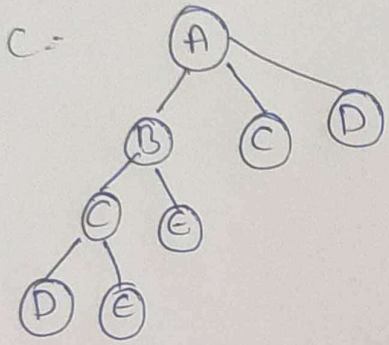


(∵ for A adjacent vertices)

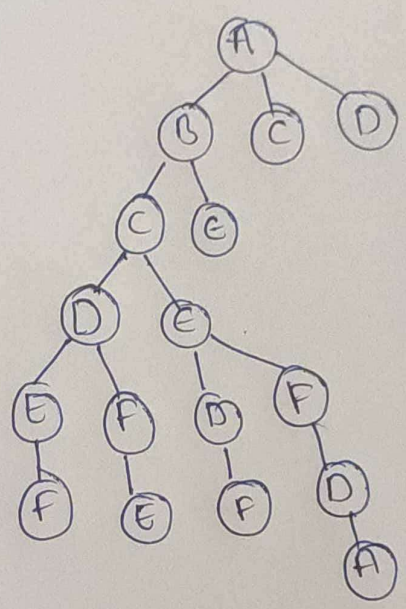
B ⇒



C:-

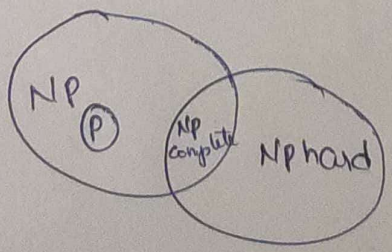


Similarly for D & E



← HC

NP hard:- The concept of NP hardness plays an vital role on the discussion about the relation between P & NP



Eg:- Travelling sales man problem