Dr. Vishwanath Karad
MIT WORLD PEACE UNIVERSITY | PUNE
TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

Theory of Computation

TY Btech CSE

Unit V

Course Objective & Course Outcomes

- **Course Objectives:**
  1. To Study computing machines by describing, classifying and comparing different types of computational models.
  2. Encourage students to study Theory of Computability and Complexity.
- **Course Outcomes:**

  After successful completion of this course students will be able to:

1. Construct finite state machines to solve problems in computing
2. Write mathematical expressions for the formal languages
3. Apply well defined rules for syntax verification
4. Construct and analyze Push Down, Post and Turing Machine for formal languages
5. Express the understanding of the decidability and Undecidability problems
6. Express the understanding of computational complexity.

Text Books & Reference Books

- **Text Books**

1. Michael Sipser "Introduction to the Theory of Computation" CENGAGE Learning, 3rd Edition ISBN-13:978-81-315-2529-6
2. Vivek Kulkarni, "Theory of Computation", Oxford University Press, ISBN-13: 978-0-19-808458-7

- **Reference Books**

1. Hopcroft Ulman, "Introduction To Automata Theory, Languages And Computations", Pearson Education Asia, 2nd Edition
2. Daniel. A. Cohen, "Introduction to Computer Theory" Wiley-India, ISBN:978-81-265-1334-5
3. K.L.P Mishra ,N. Chandrasekaran ,"Theory Of Computer Science (Automata, Languages and Computation)", Prentice Hall India,2nd Edition
4. John C. Martin, "Introduction to Language and Theory of Computation", TMH, 3rd Edition ISBN: 978-0-07-066048-9
5. Kavi Mahesh, "Theory of Computation: A Problem Solving Approach", Wiley-India, ISBN: 978-81-265-3311-4

Basic introduction to Complexity

- Concept of Decidability,
- un-decidability,
- Un-decidability of halting problem,
- Examples of undecidable problem:
- Post correspondence problem,
- Introductory ideas on Time complexity of deterministic and nondeterministic TM,
- P and NP ,
- Example of NP-Complete and NP hard Problem.

Church-Turing's Thesis

Everything that is algorithmically computable is computable by a Turing machine

Complexity of Turing Machine

- The complexity of a TM is directly proportional to the size of the functional matrix. In other words, we can say that the complexity of a TM depends on the number of symbols that are being used and the number of states of the TM. Hence:
- Complexity of a TM = $| \Gamma | \times | Q |$ (or $| I | \times$ $| S |$),

  where,

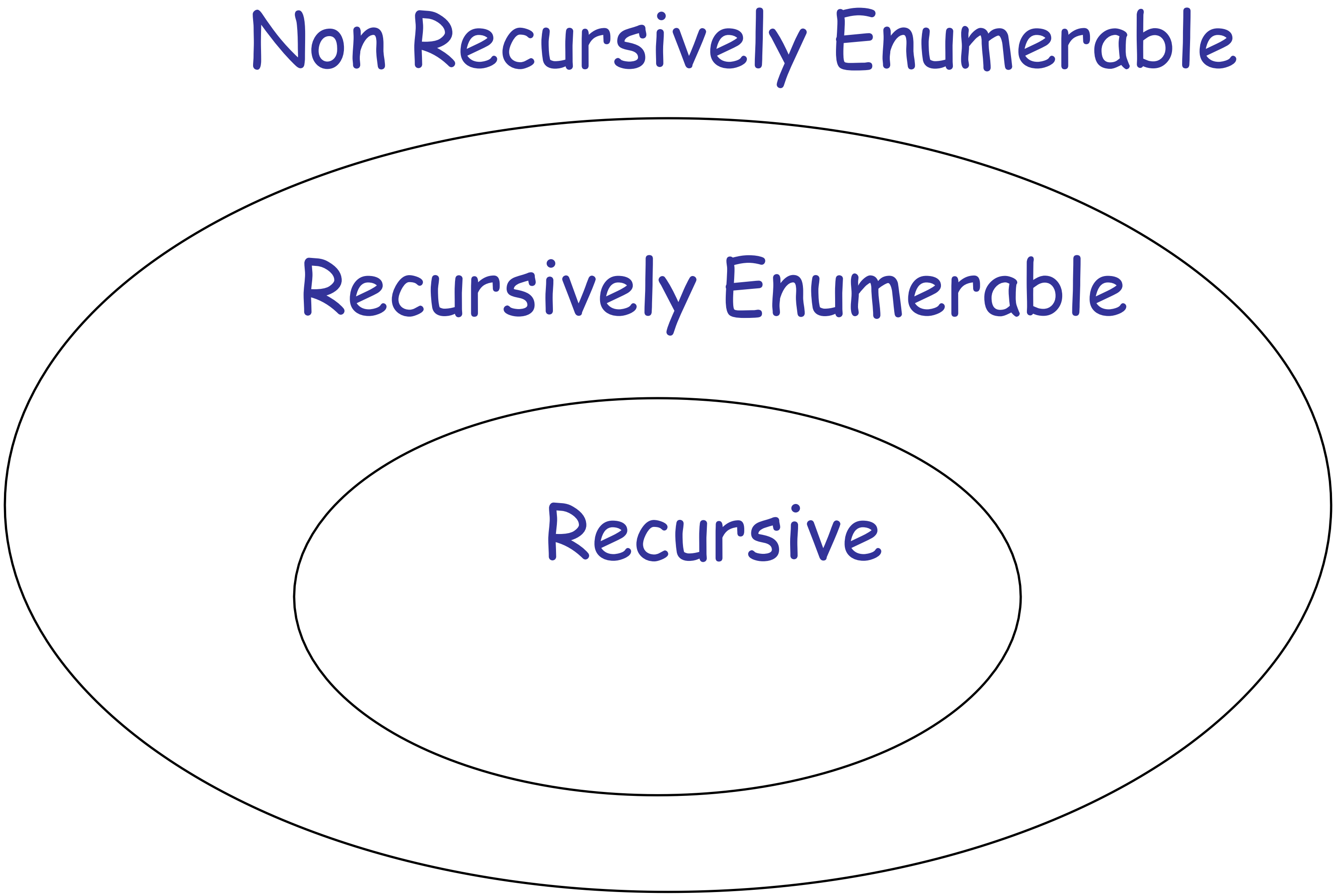
  $| \Gamma |$ = Cardinality of tape alphabet (i.e., number of tape symbols), and

  $| Q |$ = Number of states of the TM.
- For example,

  If $\Gamma = \{1, 0, a, c, ; , \}$ and $Q = \{q_0, q_1, q_2, q_3, q_4 = \text{halt}\}$

  Then, the complexity of the TM = $| \Gamma | \times | Q | = 6 \times 5 = 30$

Recursively Enumerable and Recursive Languages



Recursively Enumerable and Recursive Languages

A TM *recognizes* a language iff it accepts all and only those strings in the language.

A language L is called Turing-recognizable or recursively enumerable iff some TM recognizes L.

A TM *decides* a language L iff it accepts all strings in L and rejects all strings not in L.

A language L is called decidable or recursive iff some TM decides L.

Definition:

A language is **recursive** if some Turing machine accepts it and halts on any input string

In other words:

A language is recursive if there is a membership algorithm for it

Recursive and Recursively Enumerable Languages

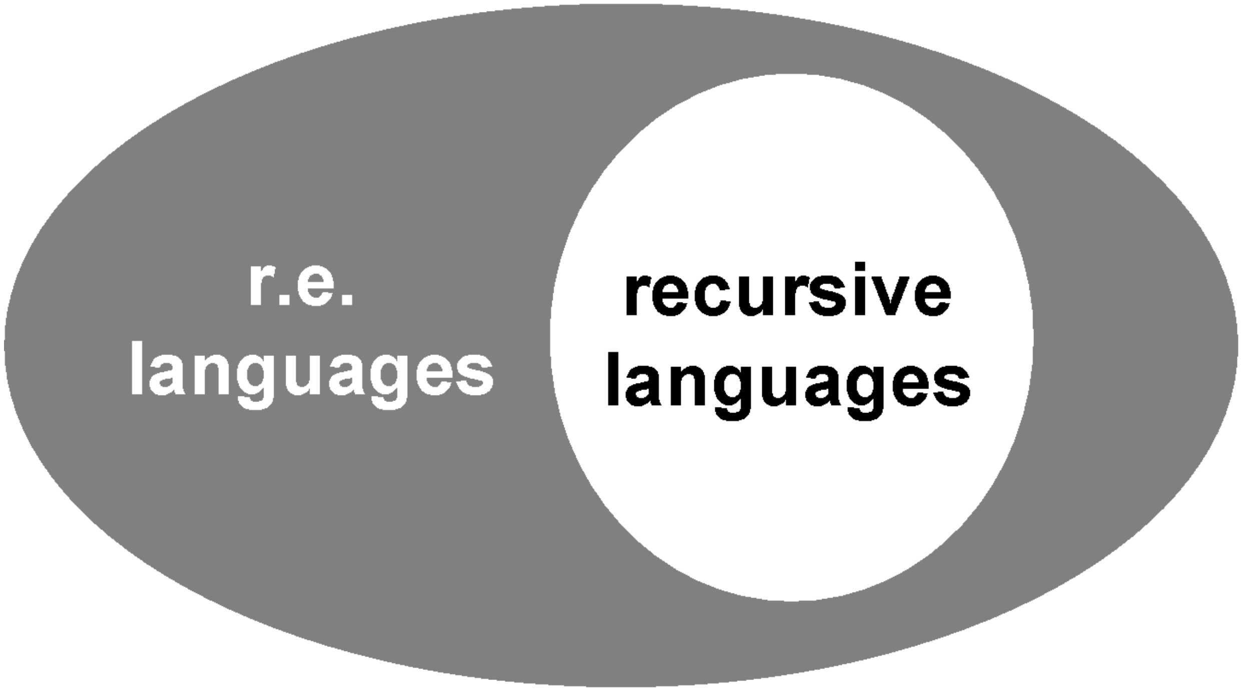
To summarize we can say that,

- **Recursively Enumerable Set**
  - A set S of words over Σ is said to be recursively enumerable, if there is a TM over Σ, which accepts every word in S and either rejects or loops for every word in $\sim S$ ($\sim S = \Sigma^* - S$). This can be represented as:
    - Accept TM = S
    - Reject (TM) $\cup$ loop (TM) = $\Sigma^* - S$
- **Recursive Set**
  - A set S of words over Σ is said to be recursive, if there is a TM over Σ, which accepts every word in S and rejects every word in $\sim S$ ($\sim S$
    $= \Sigma^* - S$). This can be represented as:
    - Accept (TM) = S
    - Reject (TM) = $\Sigma^* - S$
    - Loop (TM) = φ

Recursively Enumerable and Recursive Languages

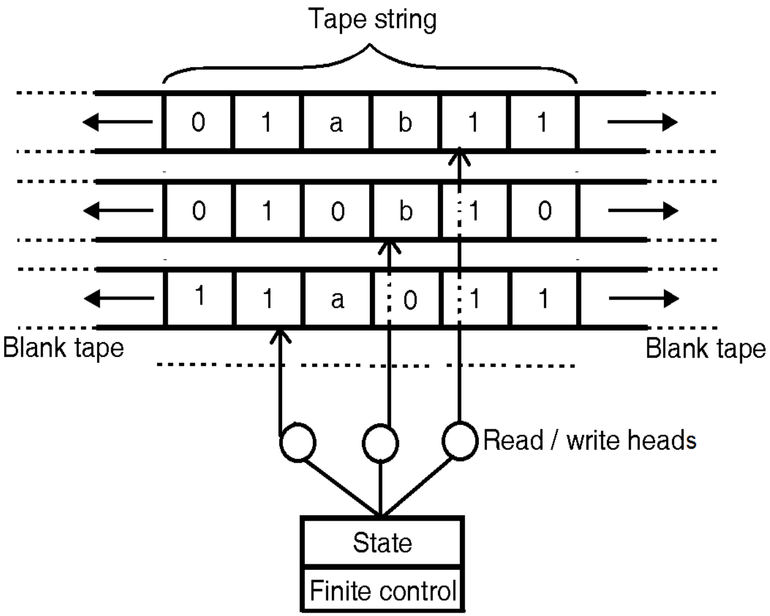
A language is called Turing-recognizable or recursively enumerable (r.e.) if some TM recognizes it

A language is called decidable or recursive if some TM decides it



Multi Tape Turing Machine

- Multi-tape Turing machines have k number of independent tapes, having their own read/write heads. These machines have independent control over all the heads—any of these can move and read/write their own tapes. All these tapes are unbounded at both the ends just as in the single-tape TM.
- Multi-tape TM and single-tape TM are equivalent in power (except for some difference in execution time



Non-Deterministic Turing Machine

A nondeterministic Turing Machine (NTM) differs from the deterministic variety by having a transition function δ such that for each state q and tape symbol X, δ(q, X) is a set of triples

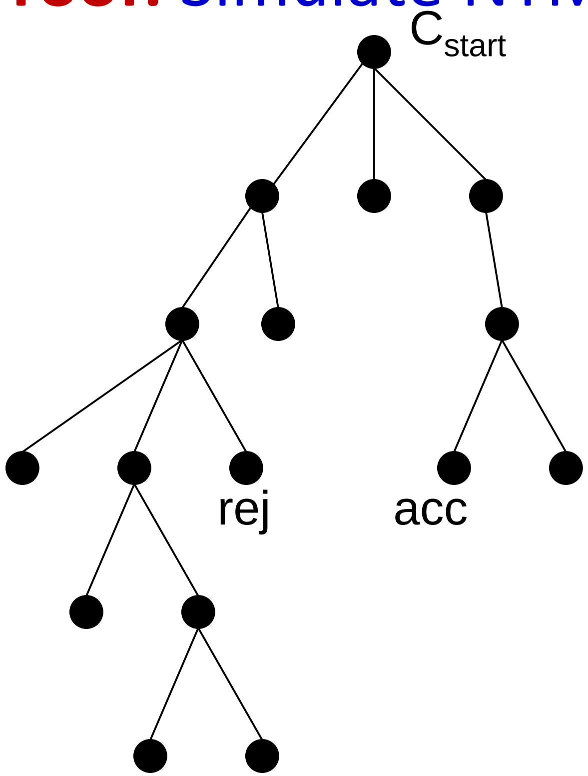
$$\{(q_1,Y_1,D_1), (q_2,Y_2,D_2), \ldots\ldots\ldots\ldots (q_k,Y_k,D_k)\}$$

Where k is any finite integer. The NTM can choose, at each step, any of the triples to be the next move. It cannot, however, pick a state from one, a tape symbol from another, a the direction from yet another.

NTM and DTM

Theorem: Every NTM has an equivalent (deterministic) TM

Proof: Simulate NTM with a deterministic TM



- computations of M are a tree
- nodes are configurations
- fanout is b = maximum number of choices in transition function
- leaves are accept/reject configurations

Simulating NTM M with a deterministic TM:

Breadth-first search of tree

- if M accepts: we will encounter accepting leaf and accept
- if M rejects: we will encounter all rejecting leaves, finish traversal of tree, and reject
- if M does not halt on some branch: we will not halt as that branch is infinite…

Simulating NTM M with a deterministic TM:

- use a 3 tape TM:
  - tape 1: input tape (read-only)
  - tape 2: simulation tape (copy of M's tape at point corresponding to some node in the tree)
  - tape 3: which node of the tree we are exploring (string in $\{1,2,\ldots b\}^*$)
- Initially, tape 1 has input, others blank

NTM and DTM

Here is the transition function of a nondeterministic TM $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_2\})$:

δ	0	1	B
q_0	$\{(q_0, 1, R)\}$	$\{(q_1, 0, R)\}$	$\emptyset$
q_1	$\{(q_1, 0, R),\ (q_0, 0, L)\}$	$\{(q_1, 1, R),\ (q_0, 1, L)\}$	$\{(q_2, B, R)\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$

Show the ID's reachable from the initial ID if the input is:

* a) 01.

b) 011.

Decidable language

- **Decidable language** -A decision problem P is said to be decidable (i.e., have an algorithm) if the language L of all yes instances to P is decidable.
- Example-
  (I) (Acceptance problem for DFA) Given a DFA does it accept a given word?
  (II) (Emptiness problem for DFA) Given a DFA does it accept any word?
  (III) (Equivalence problem for DFA) Given two DFAs, do they accept the same language?
-

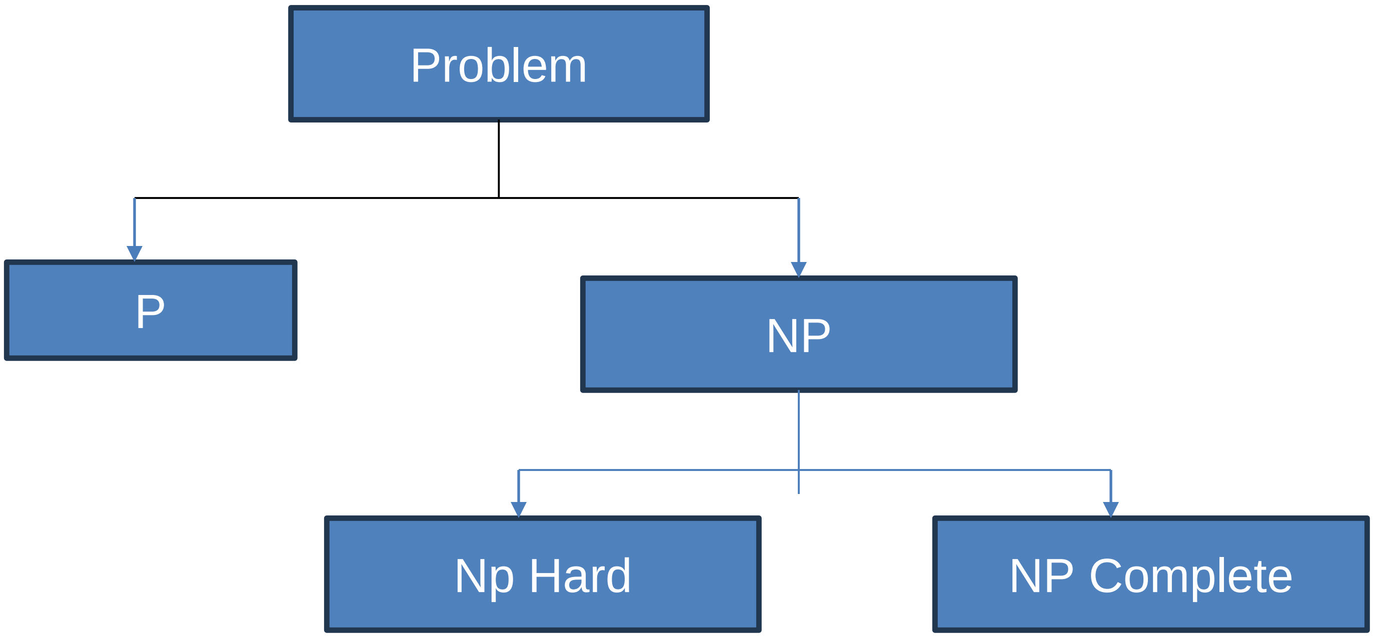
Undecidable language

- **Undecidable language** -– A decision problem P is said to be undecidable if the language L of all yes instances to P is not decidable or a language is undecidable if it is not decidable. An undecidable language maybe a partially decidable language or something else but not decidable.
- If a language is not even partially decidable , then there exists no Turing machine for that language.

Partially decidable or Semi-Decidable Language

- **Partially decidable or Semi-Decidable Language** -– A decision problem P is said to be semi-decidable (i.e., have a semi-algorithm) if the language L of all yes instances to P is RE. A language ‘L’ is partially decidable if ‘L’ is a RE but not REC language.

P and NP



NP Problem

NP Problem:

The NP problems set of problems whose solutions are hard to find but easy to verify and are solved by Non-Deterministic Machine in polynomial time.

NP-Hard Problem:

A Problem X is NP-Hard if there is an NP-Complete problem Y, such that Y is reducible to X in polynomial time. NP-Hard problems are as hard as NP-Complete problems. NP-Hard Problem need not be in NP class.

Examples:

Hamiltonian cycle ,Optimization Problem,Shortest Path

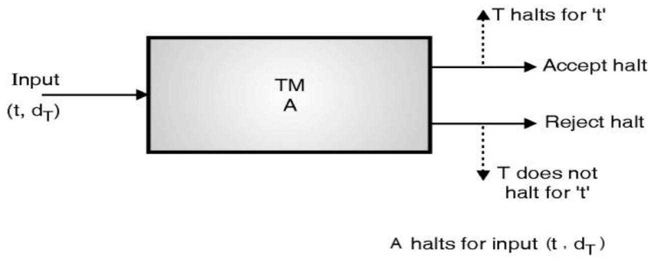
Solvable and Semi-Solvable Problems

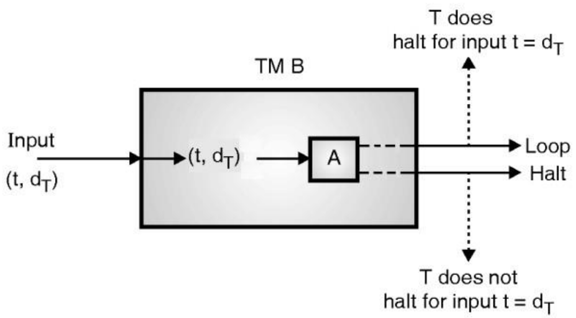
- **Solvable** problem: TM when applied to such a problem, always eventually terminates with the correct "yes" or "no" answer
  - A class of all such problems is called as **Recursive language**
  - The mathematical functions that denote these type of problems are called as **Total Recursive Functions**
  - Simple Examples - multiplication, addition, concatenation and many other

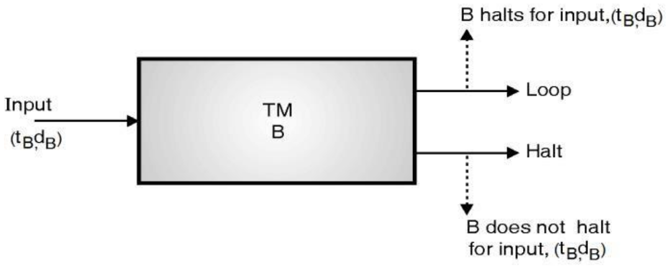
- **Semi-solvable** problem: TM when applied to such a problem, always eventually terminates with correct answer when answer is "yes" and may or may not terminate when the correct answer is "no"
  - A class of all such problems is called as **Recursively Enumerable language**
  - The mathematical functions that denote these type of problems are called as **Partial Recursive Functions**
  - Simple Examples - division, factorial and many other

Halting Problem and Unsolvablity

- For a given input for any general TM two cases arise,
  - The machine may halt after a finite number of steps
  - The machine may not ever halt no matter how long it runs
- Given any TM, problem of algorithmically determining whether it ever halts or not, is called as the **Halting Problem**
- The halting problem is **unsolvable**

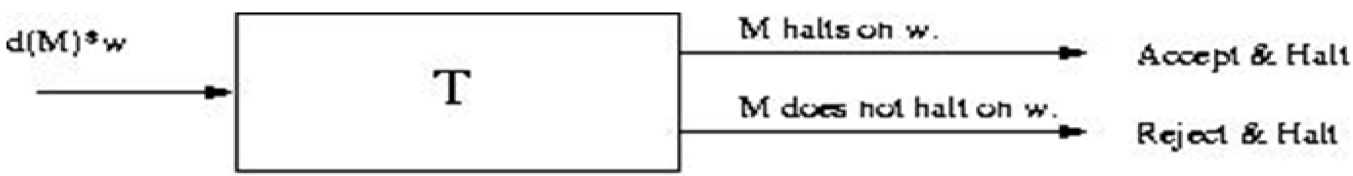






Proof :-Halting problem is undecidable

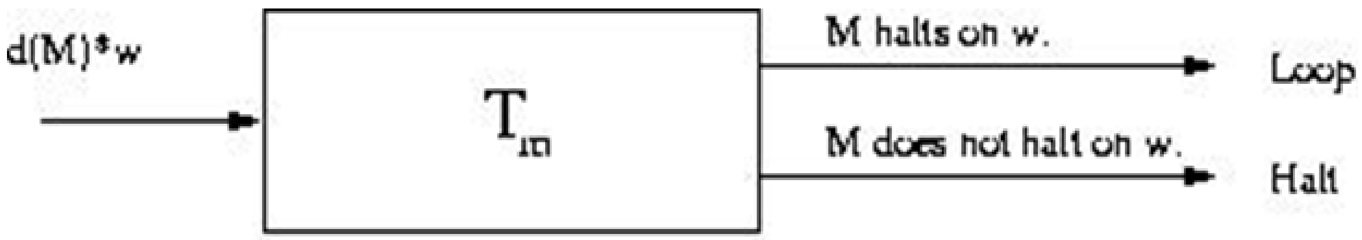
- Proof by contradiction
- There is a Turing machine T that will decide the halting problem.<M> this is the description of Turing machine M and string W . T write "accept" when M halts on w, and reject If M does not halts on W



Turing machine T

Proof :-Halting problem is undecidable

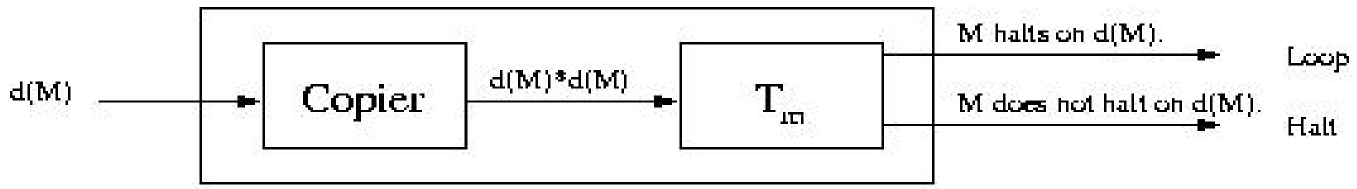
We build a Turing machine Tm and here we standardizing T so when T write yes and halt then Tm will goes into loop forever



Turing machine T_m

Proof :-Halting problem is undecidable

we build Tc with the help of TM. Here we take Tc as input which is the description of Turing machine M and we write it in this way d(M) now we copies it to obtain the string d(M)*d(M), where * is a symbol that break up the two copies of d(M) and then provide d(M)*d(M) to the Turing machine Tm .



Turing machine T_c

Proof :-Halting problem is undecidable

What Turing machine Tc does when a string given to it which describe Tc itself

d(Tc) is given as input to Tc it make copy of it and build the string d(Tc)*d(Tc) and allot to standardized T.so the altered T is specified a description of Tc and string d(Tc)

Turing machine T_c on input $d(T_c)$

END