UNIT-V
UNDECIDABALITY

Church- Turing Thesis

- In computability theory, the Church–Turing thesis is a thesis about the nature of computable functions.
- The Church-Turing thesis says that every solvable decision problem can be transformed into an equivalent Turing machine problem.
- To say that the Turing machine is a general model of computation means that any algorithmic procedure that can be carried out at all, by a human computer or a team of humans or an electronic computer, can be carried out by a TM.

- The theorem states that TM’s compute exactly same things that computers compute.
- In other words we can say, “no computational procedure will be considered as algorithmic procedure unless it can be represented using Turing Machine”.
- Unfortunately, this Thesis cannot be proven, but it is believed in the basis of amount of evidence that support it.
- Because, just it is an idea or statement that cannot be derived from logical statements.

Universal Turing Machine

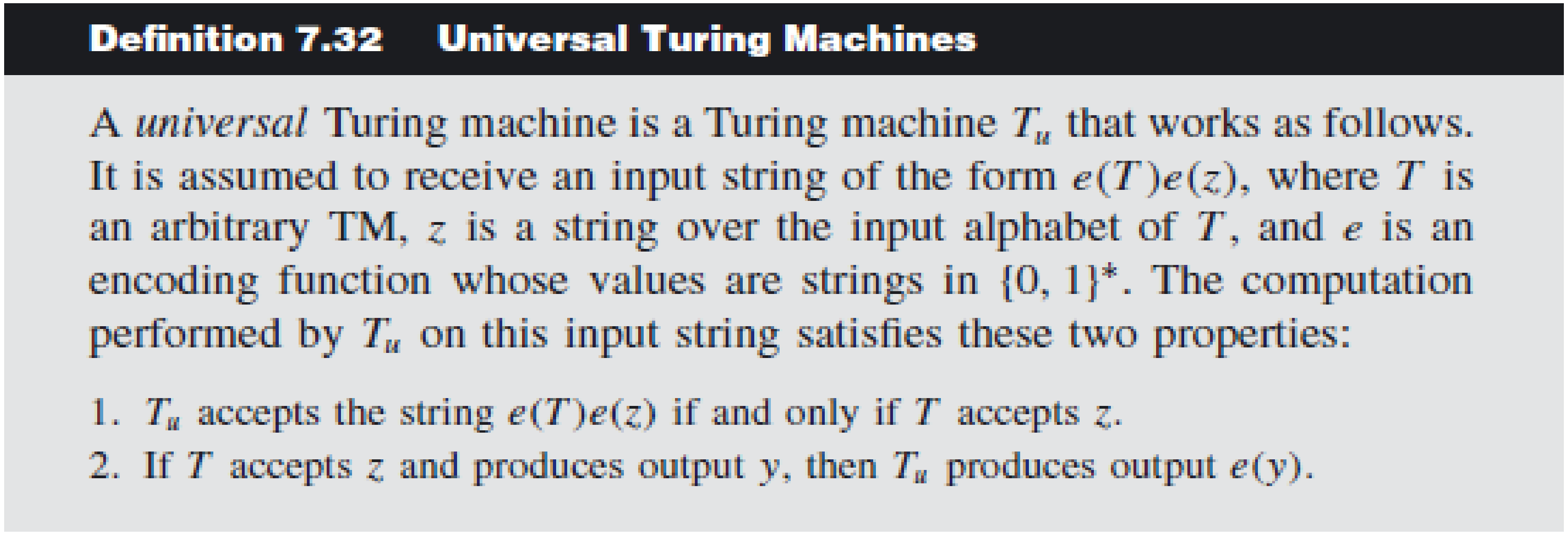
- TMs are mathematical models of general purpose computers and we are aware that computers are not designed to solve specific problems, they simply execute program instructions stored in their memory, based on the algorithm for any data.
- But, TMs so far we discussed are ‘unprogrammable’ devices solved to specific problem.
- But, is any possibility to convert a TM into programmable general purpose computer.

- Such general purpose TMs are called universal Turing Machines.

Definition: A Universal TM T<u>u</u> takes two arguments:

I. The description of computation i.e. TM

II. The Input String

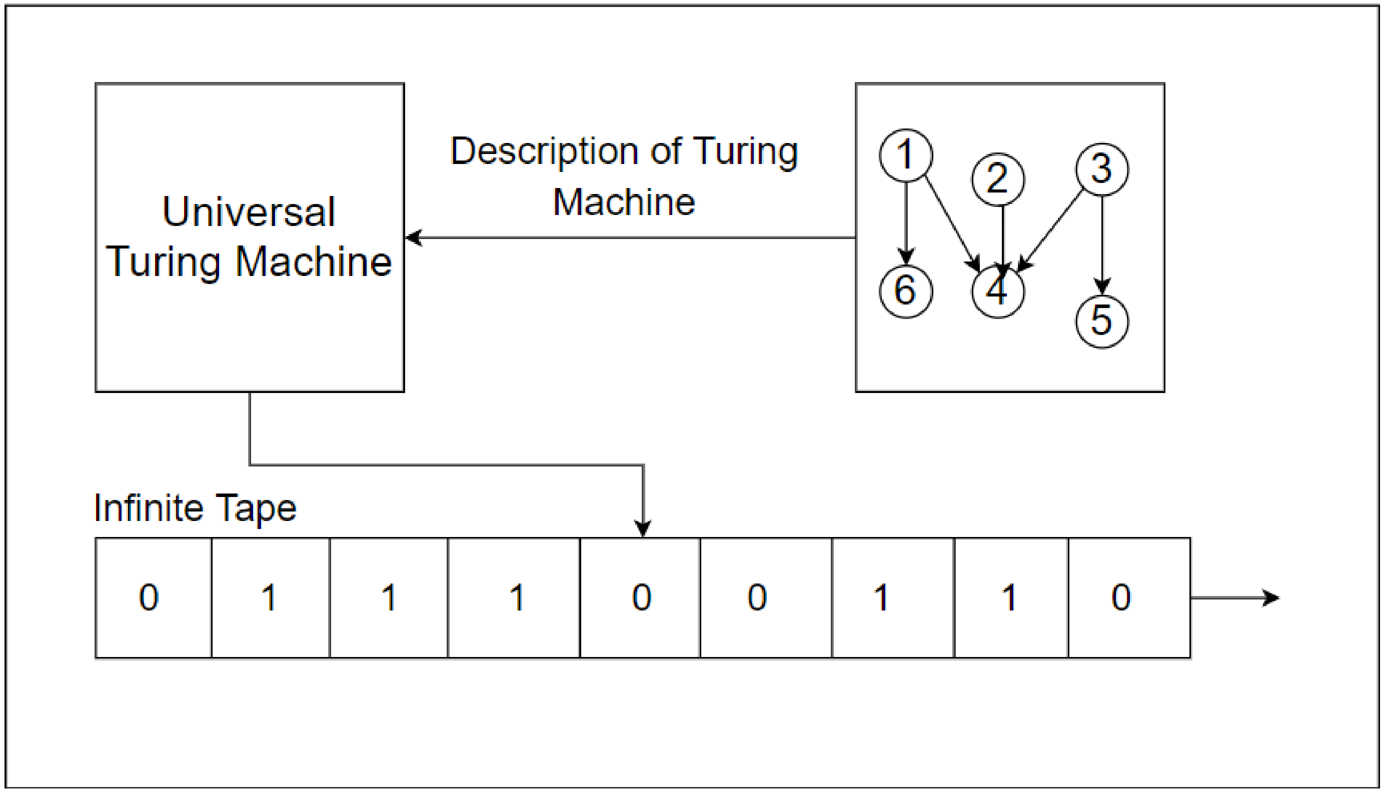


Definition 7.32 Universal Turing Machines

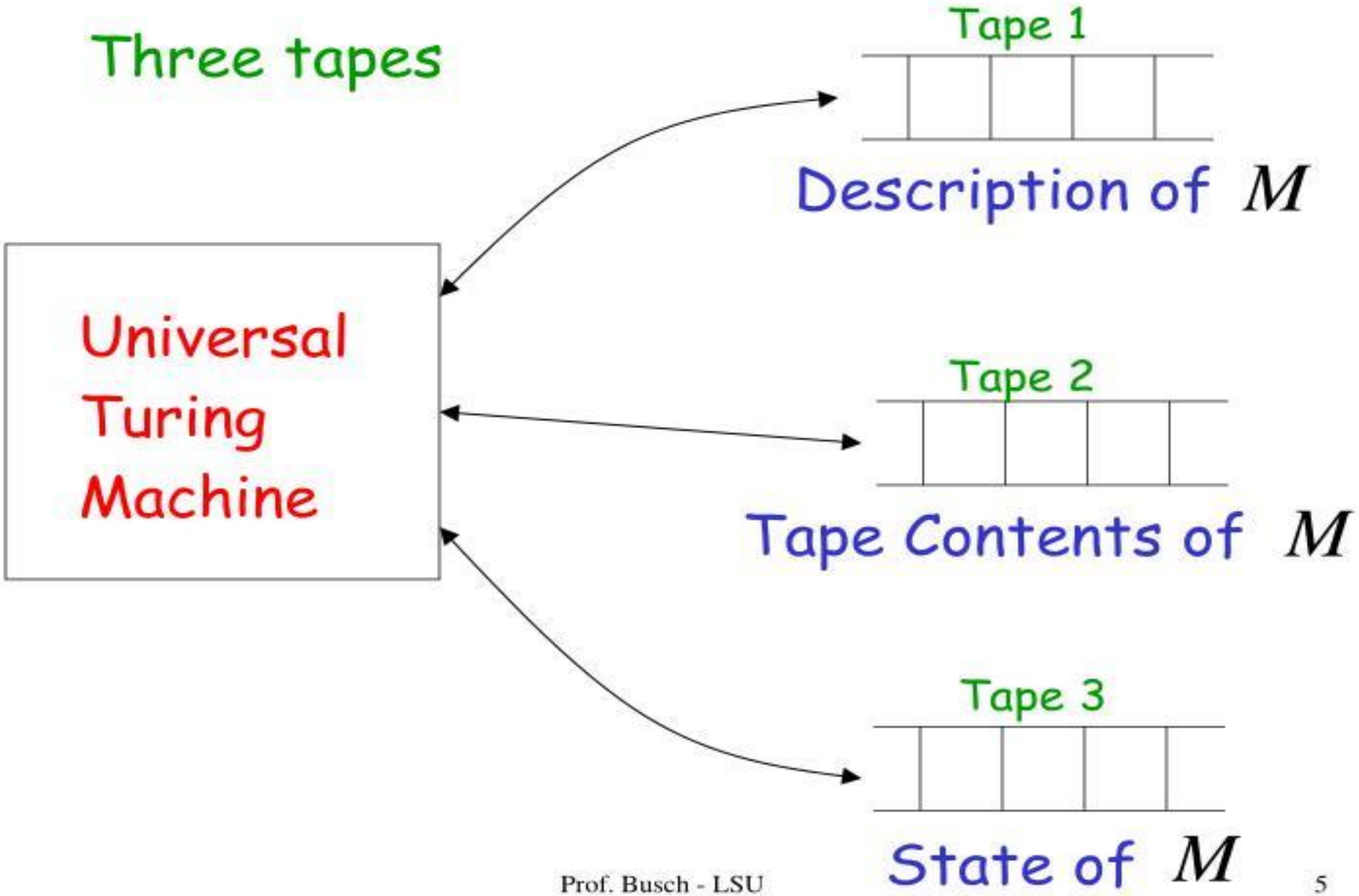
A *universal* Turing machine is a Turing machine T_u that works as follows. It is assumed to receive an input string of the form $e(T)e(z)$, where T is an arbitrary TM, z is a string over the input alphabet of T, and e is an encoding function whose values are strings in $\{0, 1\}^*$. The computation performed by T_u on this input string satisfies these two properties:

1. T_u accepts the string $e(T)e(z)$ if and only if T accepts z.
2. If T accepts z and produces output y, then T_u produces output $e(y)$.

Universal Turing Machine

- Universal TMs are extremely powerful and have enough power to simulate the behavior of any TM.
- 

Three tapes


Tape 1
Description of M
Universal Turing Machine
Tape 2
Tape Contents of M
Tape 3
State of M

Description of M

We describe Turing machine M
as a string of symbols:

We encode M as a string of symbols

Alphabet Encoding

Symbols:	a	b	c	d	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
Encoding:	1	11	111	1111	

State Encoding

States:	q_1	q_2	q_3	q_4	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
Encoding:	1	11	111	1111	

Head Move Encoding

Move:	L	R
	$\downarrow$	$\downarrow$
Encoding:	1	11

Transition Encoding

Transition: $\delta(q_1, a) = (q_2, b, L)$

Encoding: $1\,0\,1\,0\,11\,0\,1\,0\,1\,0\,1$

separator

Turing Machine Encoding

Transitions:

$\delta(q_1, a) = (q_2, b, L)$ $\qquad$ $\delta(q_2, b) = (q_3, c, R)$

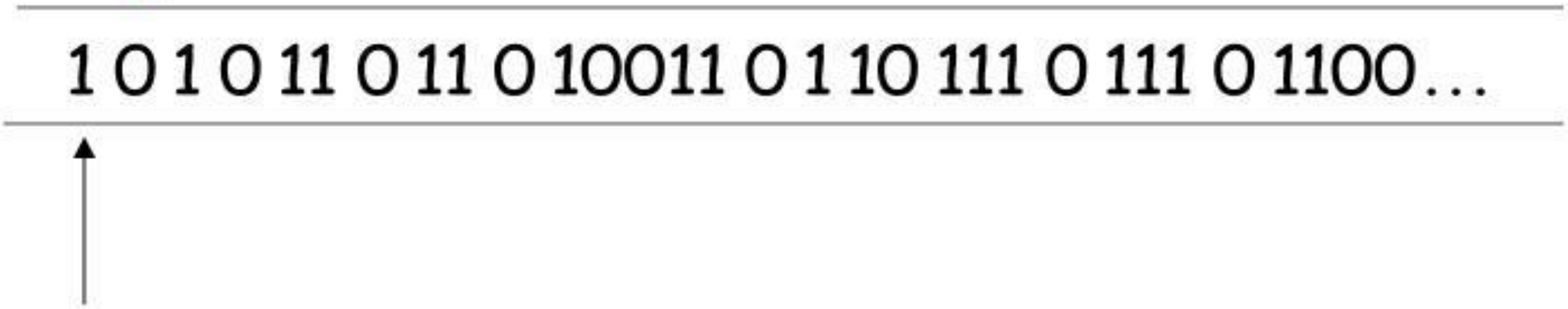
Encoding:

$$\underbrace{10101101101}\ \underset{\overset{\uparrow}{\text{separator}}}{00}\ \underbrace{1101101110111011}$$

Tape 1 contents of Universal Turing Machine:

binary encoding
of the simulated machine M

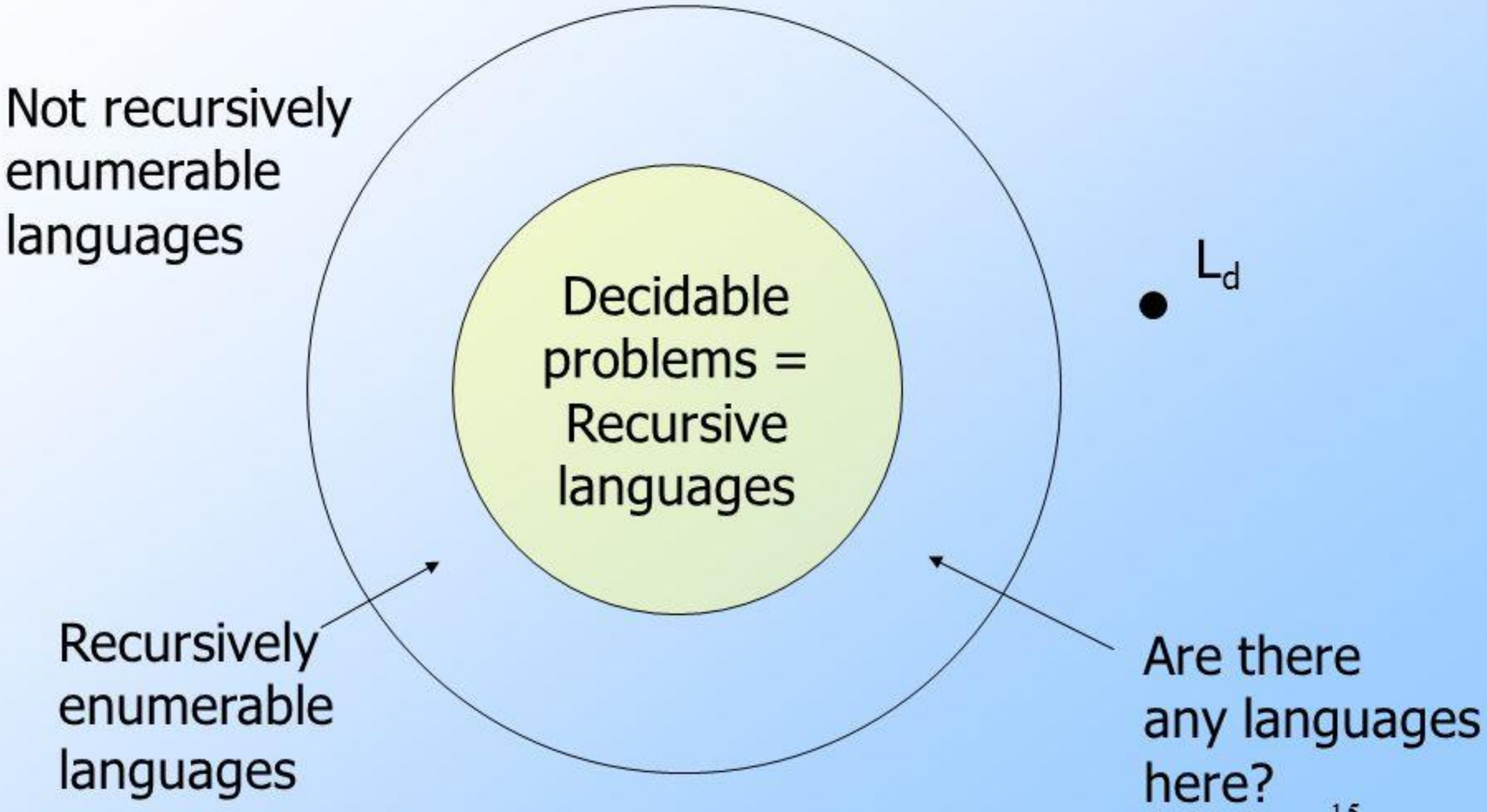
Tape 1



Diagonalization Language

- The language, that is not accepted by any Turing machine is called Diagonalization language.
- Means, Diagonalization languages are non-recursive enumerable languages.
- The diagonalization language Ld, is the set of strings of Wi, where Wi is not in L(Mi).

Relation between RE and Rec


Not recursively enumerable languages
Decidable problems = Recursive languages
L_d
Recursively enumerable languages
Are there any languages here?

Universal Language

- The Universal Language Lu is the set of binary strings that encode a pair <M,w> where w is accepted by M.
- Turing Machine M, is going to accept any type of languages over an specific input alphabets.
- To accept any type of languages, we require universal Turing machine.
- The universal language L_u is a recursively enumerable language.

Undecidable problems about Languages

- **Reducability or Reduction:**

  A reduction is a way of converting one problem to another problem in such a way that a solution to the second problem can be used to solve the first problem.

  For example, suppose that you want to find your way around a new city. You know that doing so would be easy if you had a map.

  Thus, you can reduce the problem of finding your way around the city to the problem of obtaining a map of the city.

- Reducibility always involves two problems, which we call A and B. If A reduces to B, we can use a solution to B to solve A.
- So in our example, A is the problem of finding your way around the city and B is the problem of obtaining a map.
- Reducibility also occurs in mathematical problems. For example, the problem of measuring the area of a rectangle reduces to the problem of measuring its length and width.

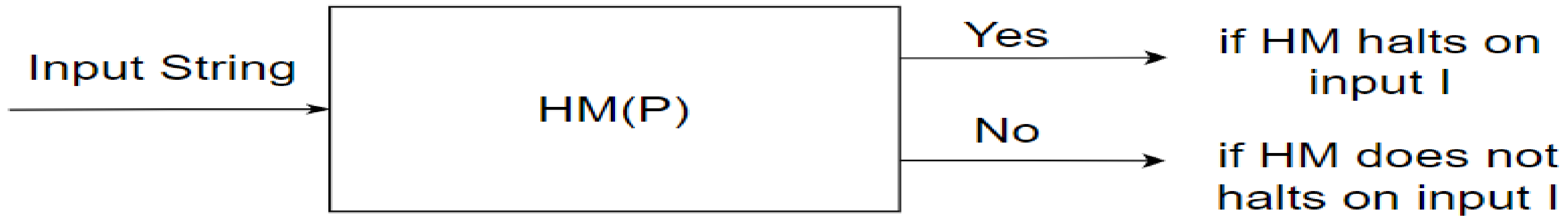
- In terms of computability theory, if A is reducible to B and B is decidable, A also is decidable.
- Equivalently, if A is undecidable and reducible to B, then B is undecidable.

UNDECIDABLE PROBLEMS FROM LANGUAGE THEORY:

The popular undecidable problem is Halting Problem.

Halting Problem

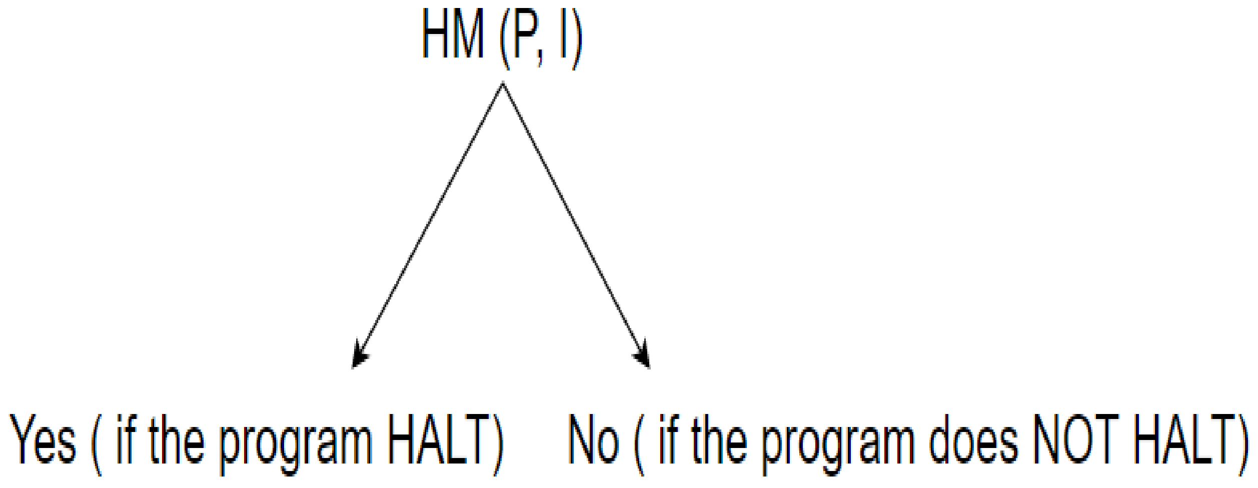
- **The Halting problem –** Given a program/algorithm will ever halt or not?
- Halting means that the program on certain input will **accept it and halt** or **reject it and halt** and it would **never go into an infinite loop**.
- Basically halting means terminating. So can we have an algorithm that will tell that the given program will halt or not.
- In terms of Turing machine, will it terminate when run on some machine with some particular given input string.



- The answer is **NO.** We cannot design a generalized algorithm that can accurately predict whether or not a given program will ever halt.
- The only way to find out is to run the algorithm and see if it halts.
- To prove it is undecidable we are going to use proof by contradiction.

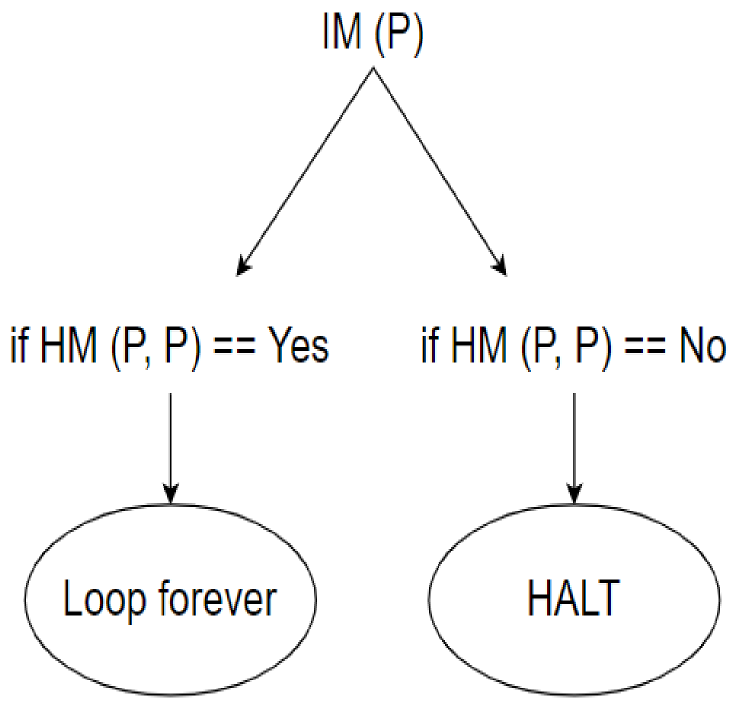
Proof By contradiction:

Step1: Assume we can create a machine called HM(P, I), where HM is the Halting machine, P is the program, and I is the input. After receiving both inputs, the machine HM will output whether or not the program P terminates.

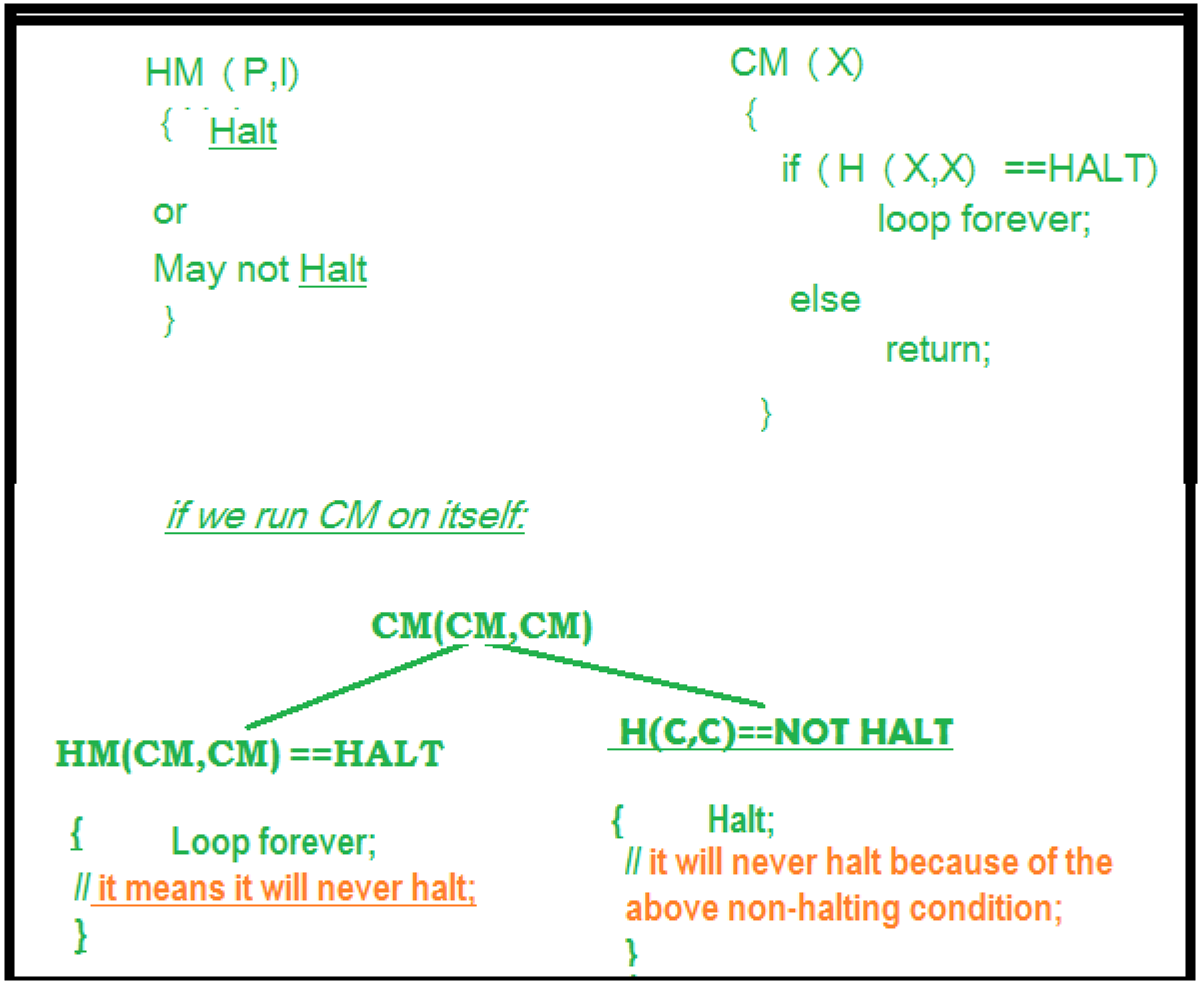

HM (P, I)
Yes (if the program HALT)
No (if the program does NOT HALT)

Step2: Now, create an inverted halting machine IM that takes a program P as input and,

- Loops forever If HM returns YES.
- Halts if HM returns NO.



Step3: Now, take a situation where the program IM is passed to the IM function as an input. Here, we got a contradiction. Let's understand how.



```
HM (P,I)
 {   Halt

 or

 May not Halt
 }
```

```
CM (X)
 {
    if (H (X,X)  ==HALT)
          loop forever;

    else
          return;

 }
```

if we run CM on itself:

```
{      Loop forever;
// it means it will never halt;
}
```

```
{      Halt;
 // it will never halt because of the
 above non-halting condition;
 }
```

- It is impossible for outer function to halt if its code (inner body) is in loop and also it is impossible for outer non halting function to halt even after its inner code is halting.
- So the both condition is non halting for CM machine/program even we had assumed in the beginning that it would halt.
- So this is the contradiction and we can say that our assumption was wrong and this problem, i.e., halting problem is undecidable.

Post Correspondence Problem

- The Post Correspondence Problem (PCP), is an undecidable decision problem. The PCP problem over an alphabet ∑ is stated as follows –
- Given the following two lists, **M** and **N** of non-empty strings over ∑ –

$$M = (x_1, x_2, x_3, \ldots\ldots\ldots, x_n)$$

$$N = (y_1, y_2, y_3, \ldots\ldots\ldots, y_n)$$

- We can say that there is a Post Correspondence Solution, if for some $i_1, i_2, \ldots\ldots\ldots\ldots i_k$, where $1 \leq i_j \leq n$, the condition $\mathbf{x_{i1} \ldots\ldots x_{ik} = y_{i1} \ldots\ldots y_{ik}}$ satisfies.

Example

- Find whether the lists

  M = (abb, aa, aaa) and N = (bba, aaa, aa)

  have a Post Correspondence Solution?

Solution:

Here,

$x_2x_1x_3$ = 'aaabbaaa' and **$y_2y_1y_3$ = 'aaabbaaa'**

We can see that

$x_2x_1x_3 = y_2y_1y_3$

Hence, the solution is **i = 2, j = 1, and k = 3.**

- Consider the correspondence system as given below

  A = (b, bab^3, ba) and B = (b^3, ba, a). The input set is ∑ = {0, 1}. Find the solution.

- **Solution:**

A solution is 2, 1, 1, 3.

That means $w_2w_1w_1w_3 = x_2x_1x_1x_3$

w_2	w_1	w_1	w_3
bab^3	b	b	ba
x_2	x_1	x_1	x_3
ba	b^3	b^3	a

Rice’s Theorem

Rice’s Theorem

Def: Let a “property” P be a set of recognizable languages.

Ex: P_1={L | L is a decidable language}

P_2={L | L is a context-free language}

$P_3=\{L \mid L = L^*\}$

$P_4=\{\{\varepsilon\}\}$

$P_5=\emptyset$

P_6={L | L is a recognizable language}

L is said to “have property P” iff $L \in P$

Ex: $(a+b)^*$ has property P_1, P_2, P_3 & P_6 but not P_4 or P_5

$\{ww^R\}$ has property P_1, P_2, & P_6 but not P_3, P_4 or P_5

Def: A property is “trivial” iff it is empty or

it contains all recognizable languages.

- A property is called to be trivial if either it is not satisfied by any recursively enumerable languages, or if it is satisfied by all recursively enumerable languages.
- The two Trivial Properties are Decidable i.e. either none or all.

- Rice theorem states that any non-trivial property of a language which is recognized by a Turing machine is undecidable.

Formal Definition

- If P is a non-trivial property, and the language holding the property, L_p , is recognized by Turing machine M, then $L_p = \{<M> \mid L(M) \in P\}$ is undecidable.

- A non-trivial property is satisfied by some recursively enumerable languages and are not satisfied by others.
- In a non-trivial property, where L ∈ P, both the following properties hold:
  - **Property 1** − There exists Turing Machines, M1 and M2 that recognize the same language, i.e. either

(<M1>, <M2> ∈ L) or (<M1>,<M2> ∉ L)

  - **Property 2** − There exists Turing Machines M1 and M2, where M1 recognizes the language while M2 does not, i.e. <M1> ∈ L and <M2> ∉ L