LOVELY
PROFESSIONAL
UNIVERSITY

UNIT VI
TURING MACHINES AND COMPLEXITY

SYLLABUS

- TURING MACHINES AND COMPLEXITY : Turing Machine Model, Representation of Turing Machines, Design of Turing Machines, The Model of Linear Bounded Automaton, Power of LBA, Variations of TM, Non-Deterministic Turing Machines, Halting Problem of Turing Machine, Post Correspondence Problem, Basic Concepts of Computability, Decidable and Undecidable languages, RECURSIVELY ENUMERABLE LANGUAGE, Computational Complexity: Measuring Time & Space Complexity, Power of Linear Bounded Automaton, Variations of Turing Machine, Cellular automaton

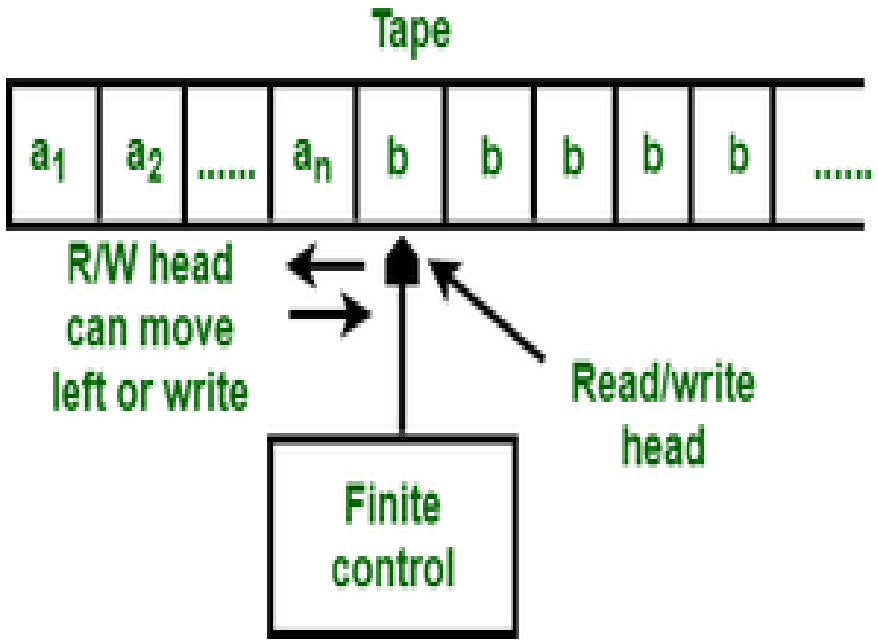
Introduction

- A Turing Machine is an accepting device which accepts the languages (recursively enumerable set) generated by type 0 grammars.

- It was invented in 1936 by Alan Turing.

Design of Turing Machine

- A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given.
- It consists of a head which reads the input tape. A state register stores the state of the Turing machine.
- After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left.
- If the TM reaches the final state, the input string is accepted, otherwise rejected.



block diagram of the basic model for the Turing machine is given in F

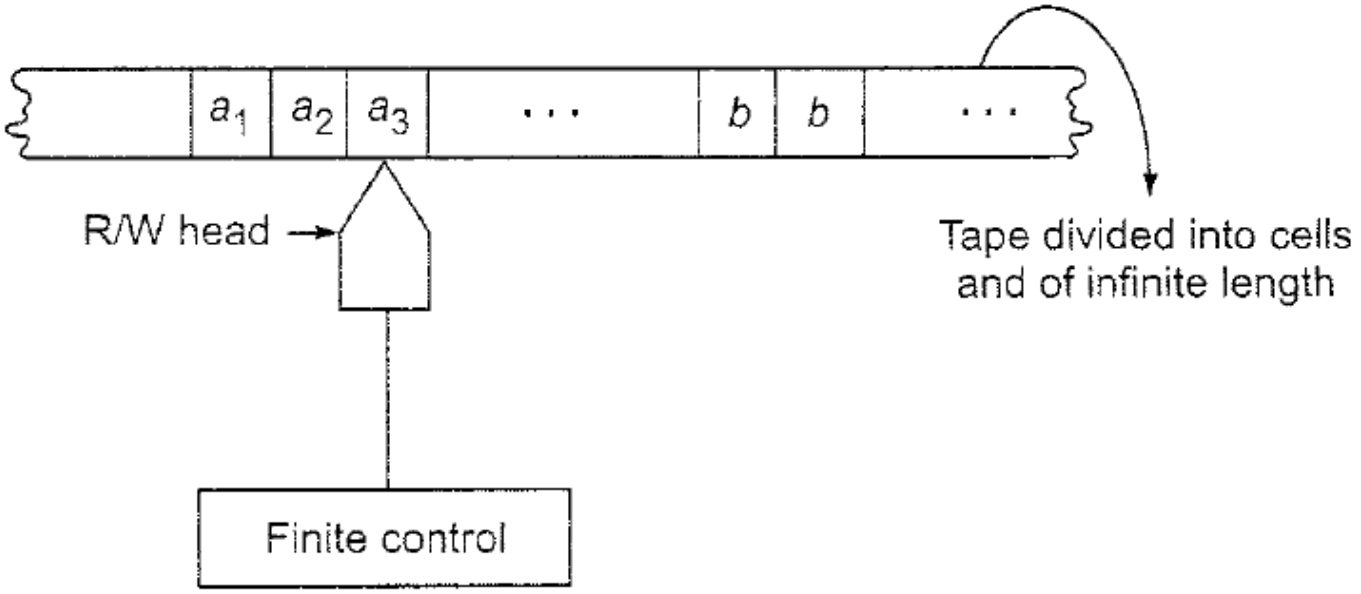


Fig. 9.1 Turing machine model.

Activate

Definition

- A TM can be formally described as a 7-tuple (Q, X, ∑, δ, q_0, B, F) where –
- **Q** is a finite set of states
- **X** is the tape alphabet
- **∑** is the input alphabet
- **δ** is a transition function; δ : Q × X → Q × X × {Left_shift, Right_shift}.
- **q_0** is the initial state
- **B** is the blank symbol
- **F** is the set of final states

Comparison with the previous automaton

Machine	Stack Data Structure	Deterministic?
Finite Automaton	N.A	Yes
Pushdown Automaton	Last In First Out(LIFO)	No
Turing Machine	Infinite tape	Yes

LOVELY PROFESSIONAL UNIVERSITY
PUNJAB (INDIA)
LOVELY PROFESSIONAL UNIVERSITY

Example of Turing machine

- Turing machine M = (Q, X, ∑, δ, q_0, B, F) with
- Q = {q_0, q_1, q_2, q_f}
- X = {a, b}
- ∑ = {1}
- q_0 = {q_0}
- B = blank symbol
- F = {q_f }

δ is given by –

Tape alphabet symbol	Present State 'q_0'	Present State 'q_1'	Present State 'q_2'
a	$1Rq_1$	$1Lq_0$	$1Lq_f$
b	$1Lq_2$	$1Rq_1$	$1Rq_f$

- Here the transition $1Rq_1$ implies that the write symbol is 1, the tape moves right, and the next state is q_1.
- Similarly, the transition $1Lq_2$ implies that the write symbol is 1, the tape moves left, and the next state is q_2.

Types of Turing Machine(Variation)

- MultiTape
- MultiHead
- 2way infinite TM
- Non Deterministic
- Deterministic
- Multi Dimensional
- Universal

Linear Bounded Automata

- A linear bounded automaton is a multi-track non-deterministic Turing machine with a tape of some bounded finite length.
- **Length = function (Length of the initial input string, constant c)**

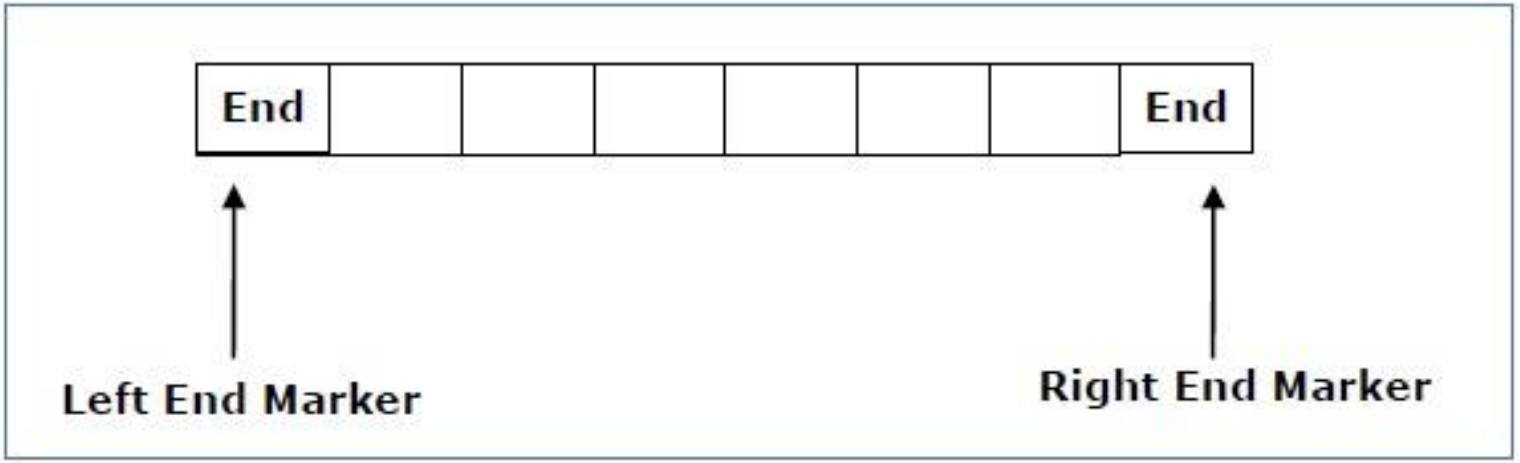
Here,

- **Memory information ≤ c × Input information**

- The computation is restricted to the constant bounded area.

- The input alphabet contains two special symbols which serve as left end markers and right end markers which mean the transitions neither move to the left of the left end marker nor to the right of the right end marker of the tape.

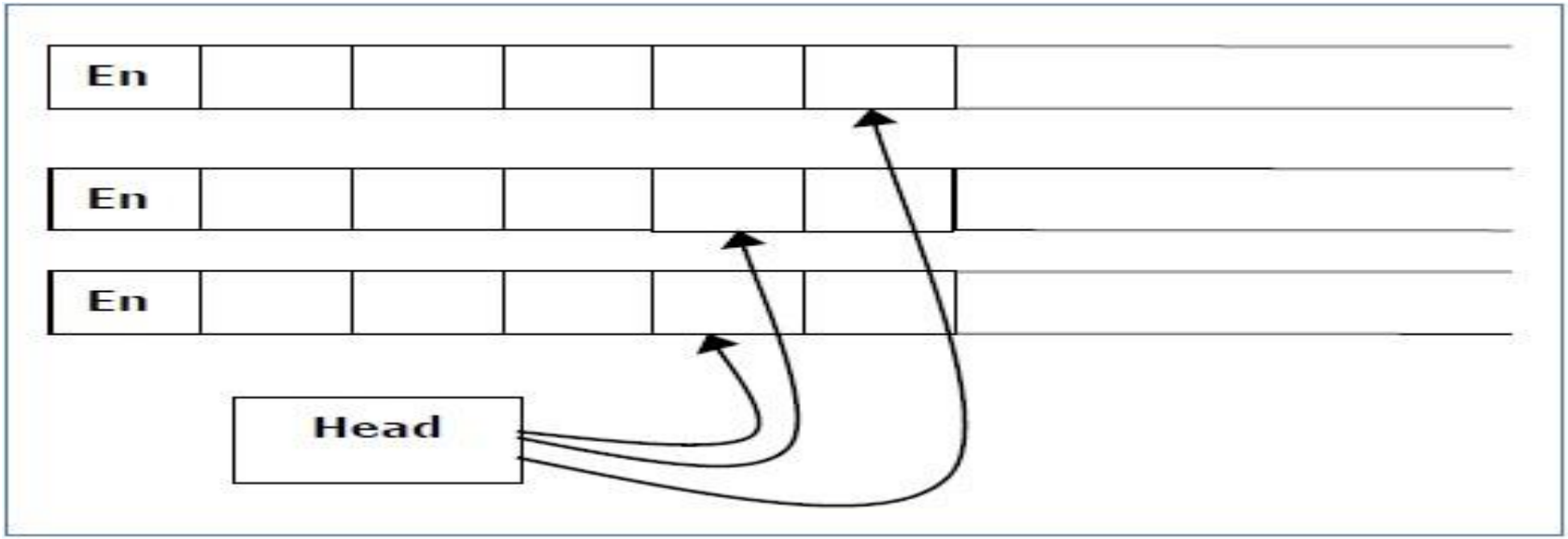
- A linear bounded automaton can be defined as an 8-tuple (Q, X, ∑, q_0, ML, MR, δ, F) where –
- **Q** is a finite set of states
- **X** is the tape alphabet
- **∑** is the input alphabet
- **q_0** is the initial state
- **M_L** is the left end marker
- **M_R** is the right end marker where $M_R \neq M_L$
- **δ** is a transition function which maps each pair (state, tape symbol) to (state, tape symbol, Constant 'c') where c can be 0 or +1 or -1
- **F** is the set of final states



- A deterministic linear bounded automaton is always **context-sensitive** and the linear bounded automaton with empty language is **undecidable.**.

Multi-tape Turing Machine

- Multi-tape Turing Machines have multiple tapes where each tape is accessed with a separate head.
- Each head can move independently of the other heads. Initially the input is on tape 1 and others are blank.
- At first, the first tape is occupied by the input and the other tapes are kept blank.
- Next, the machine reads consecutive symbols under its heads and the TM prints a symbol on each tape and moves its heads.



- A Multi-tape Turing machine can be formally described as a **6-tuple (Q, X, B, δ, q_0, F)** where –
- **Q** is a finite set of states
- **X** is the tape alphabet
- **B** is the blank symbol
- **δ** is a relation on states and symbols where
- $\delta: Q \times X^k \rightarrow Q \times (X \times \{Left\_shift, Right\_shift, No\_shift\})^k$
- where there is **k** number of tapes
- $\mathbf{q_0}$ is the initial state
- **F** is the set of final states
- **Note** – Every Multi-tape Turing machine has an equivalent single-tape Turing machine.

Multi-track Turing Machine

- Multi-track Turing machines, a specific type of Multi-tape Turing machine, contain multiple tracks but just one tape head reads and writes on all tracks.

- Here, a single tape head reads n symbols from **n** tracks at one step.

- It accepts recursively enumerable languages like a normal single-track single-tape Turing Machine accepts.

- A Multi-track Turing machine can be formally described as a 6-tuple $(Q, X, \sum, \delta, q_0, F)$ where –
- **Q** is a finite set of states
- **X** is the tape alphabet
- **∑** is the input alphabet
- **δ** is a relation on states and symbols where
- $\delta(Q_i, [a_1, a_2, a_3,....]) = (Q_j, [b_1, b_2, b_3,....]$, Left_shift or Right_shift)
- $\mathbf{q_0}$ is the initial state
- **F** is the set of final states
- **Note** – For every single-track Turing Machine **S**, there is an equivalent multi-track Turing Machine **M** such that **L(S) = L(M)**.

Non-Deterministic Turing Machine

- In a Non-Deterministic Turing Machine, for every state and symbol, there are a group of actions the TM can have.

- So, here the transitions are not deterministic. The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.

- An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted.

- If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a **Decider** and if for some input, all branches are rejected, the input is also rejected.

- A non-deterministic Turing machine can be formally defined as a 6-tuple $(Q, X, \sum, \delta, q_0, B, F)$ where –
- **Q** is a finite set of states
- **X** is the tape alphabet
- **∑** is the input alphabet
- **δ** is a transition function;
- $\delta : Q \times X \rightarrow P(Q \times X \times \{Left\_shift, Right\_shift\})$.
- $\mathbf{q_0}$ is the initial state
- **B** is the blank symbol
- **F** is the set of final states

POLLING QUESTIONS

1. Which of the functions can a turing machine not perform?

   a) Copying a string
   b) Deleting a symbol
   c) Accepting a pal
   d) Inserting a symbol

2. A Language L may not be accepted by a Turing Machine if:

a) It it is recursively ennumerable
b) It is recursive
c) L can be ennumerated by some turing machine
d) None of the mentioned

3. Turing machine (TM) is more powerful than FMS (Finite State Machine) because

- **A.** Tape movement is confined to one direction
- **B. I**t has no finite state
- **C.** It has the capability to remember arbitrarily long sequences of input symbols
- **D.**none of these

- Which of the following automata takes queue as an auxiliary storage?

a) Finite automata
b) Push down automata
c) Turing machine
d) All of the mentioned

- A context free grammar can be recognized by

a) Push down automata
b) 2 way linearly bounded automata
c) both a and b
d) None of the mentioned

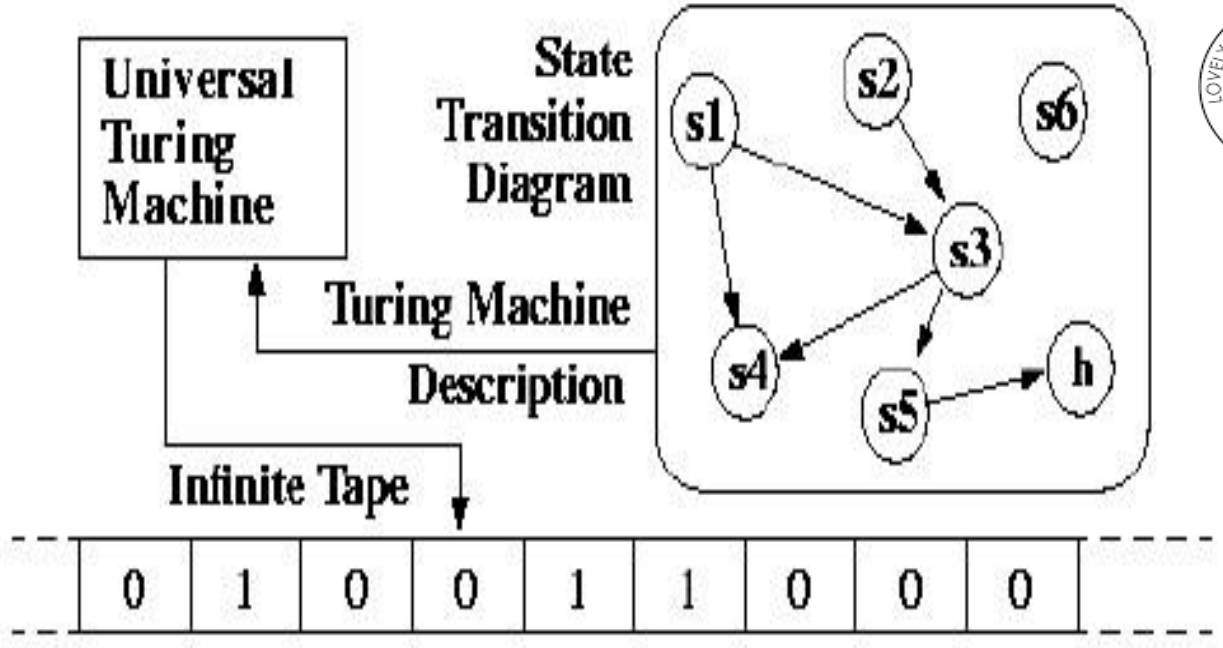
- A linear bounded automaton(LBA) is represented as an 8-tuple (Q, X, ∑, q_0, ML, MR, δ, F). What does X denote?

a) Initial Symbol
b) Tape symbol
c) Blank Symbol
d) None of these

Universal TM

- In computer science, a **universal Turing machine**(UTM) is a **Turing machine** that can simulate an arbitrary **Turing machine** on arbitrary input.

- The **universal machine** essentially achieves this by reading both the description of the **machine** to be simulated as well as the input thereof from its own tape.



UTM along with the input on the tape, takes in the description of a machine M.

The UTM can go on then to simulate M on the rest of the contents of the input tape.

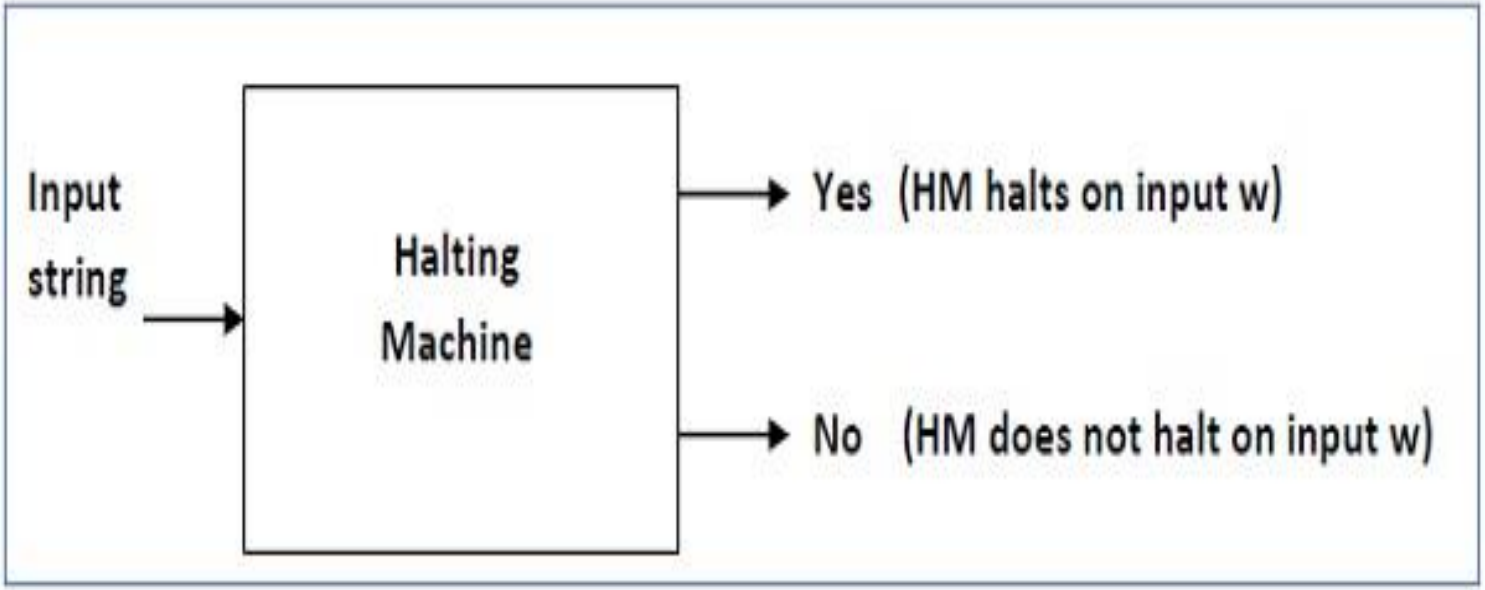
A universal Turing machine can thus simulate any other machine.

LOVELY PROFESSIONAL UNIVERSITY
PUNJAB (INDIA)
LOVELY PROFESSIONAL UNIVERSITY

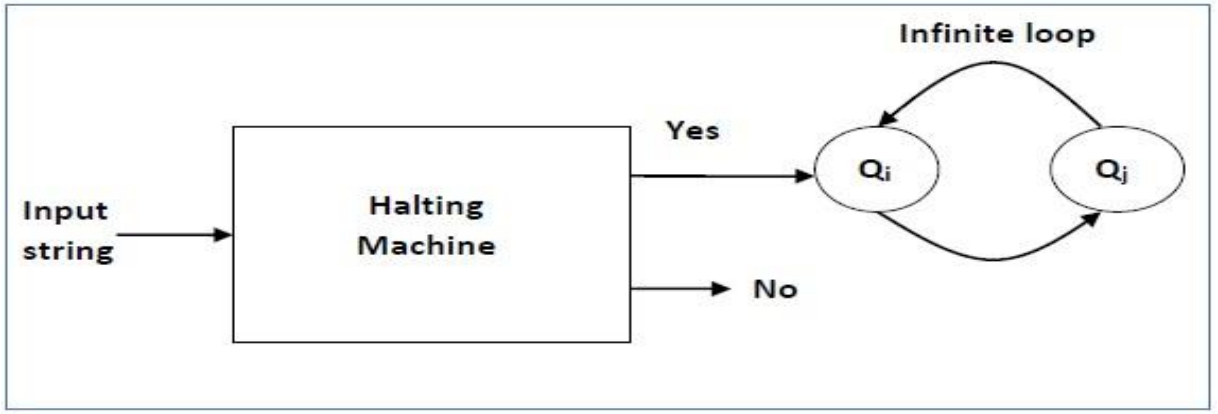
Turing Machine Halting Problem

- **Input** – A Turing machine and an input string **w**.
- **Problem** – Does the Turing machine finish computing of the string **w** in a finite number of steps? The answer must be either yes or no.
- **Proof** – At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a **Halting machine** that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'.

- The following is the block diagram of a Halting machine –



- Now we will design an **inverted halting machine (HM)'** as –
- If **H** returns YES, then loop forever.
- If **H** returns NO, then halt.
- The following is the block diagram of an 'Inverted halting machine' –



- Further, a machine **$(HM)_2$** which input itself is constructed as follows –
- If $(HM)_2$ halts on input, loop forever.
- Else, halt.
- Here, we have got a contradiction. Hence, the halting problem is **undecidable**.

Halting Problem

- In computability theory, the **halting problem** is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running, or continue to run forever.

- The problem is to determine, given a program and an input to the program, whether the program will eventually halt when run with that input.
- There are no resource limitations on the amount of memory or time required for the program's execution;
- It can take arbitrarily long and use an arbitrary amount of storage space before halting.
- The question is simply whether the given program will ever halt on a particular input.

- For example, in pseudocode, the program

```
while (true) continue
```

- does not halt; rather, it goes on forever in an infinite loop. On the other hand, the program

```
print "Hello, world!“
```

- does halt.

- While deciding whether these programs halt is simple, more complex programs prove problematic.

- Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist.

POLLING QUESTIONS

1. A turing machine that is able to simulate other turing machines:

   a) Nested Turing machines
   b) Universal Turing machine
   c) Counter machine
   d) None of the mentioned

LOVELY PROFESSIONAL UNIVERSITY

2. If d is not defined on the current state and the current tape symbol, then the machine

_____

a) does not halts
b) halts
c) goes into loop forever
d) none of the mentioned

3. Which of the following are the models equivalent to Turing machine?

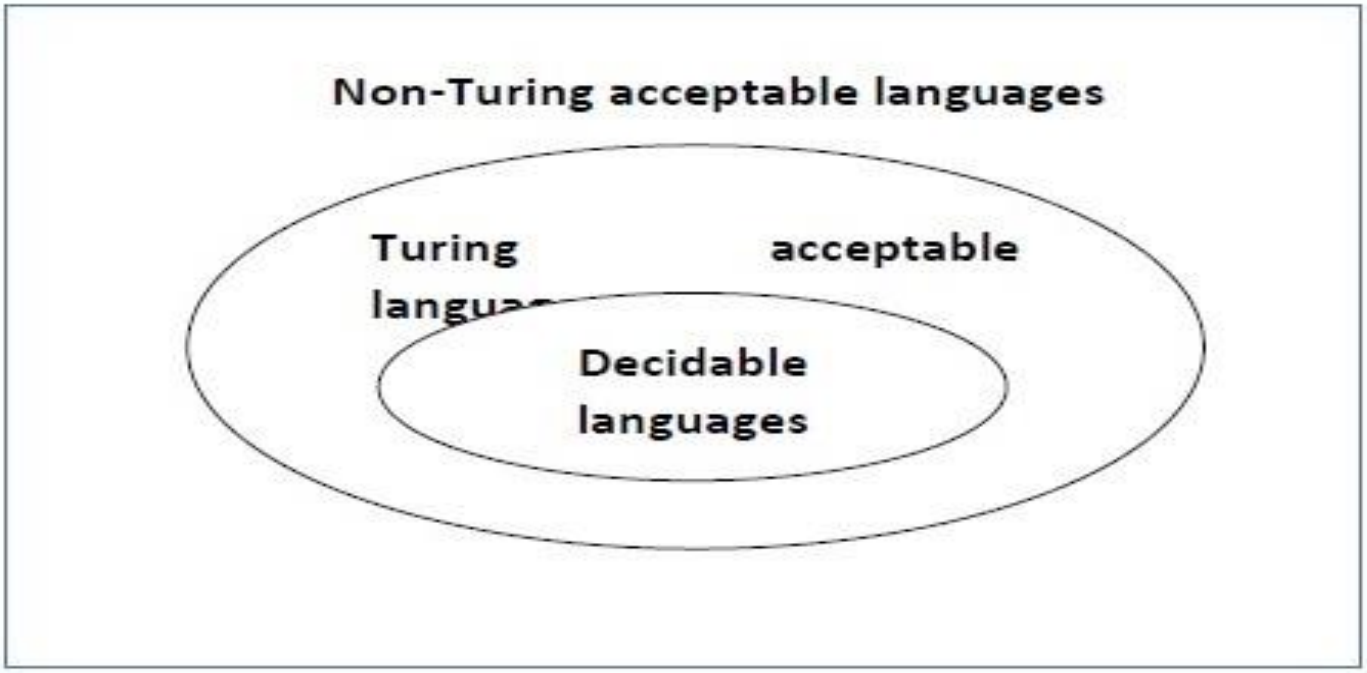
a) Multi tape turing machine
b) Multi track turing machine
c) Register machine
d) All of the mentioned

4. Which of the following is true for The Halting problem?

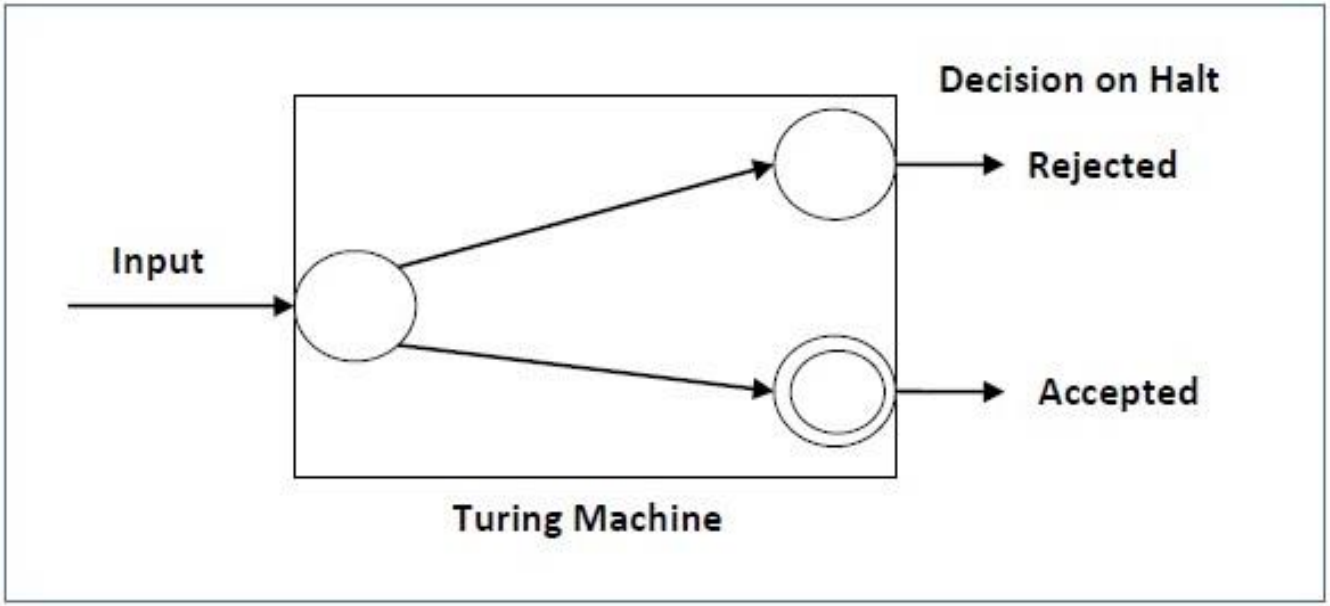
- **a.** It is recursively ennumerable
- **b.** It is undecidable
- **c.** both a and b
- **d.** None of the mentioned

Language Decidability

- A language is called **Decidable** or **Recursive** if there is a Turing machine which accepts and halts on every input string **w**. Every decidable language is Turing-Acceptable.



- A decision problem **P** is decidable if the language **L** of all yes instances to **P** is decidable.
- For a decidable language, for each input string, the TM halts either at the accept or the reject state as depicted in the following diagram –



Example 1

LOVELY PROFESSIONAL UNIVERSITY

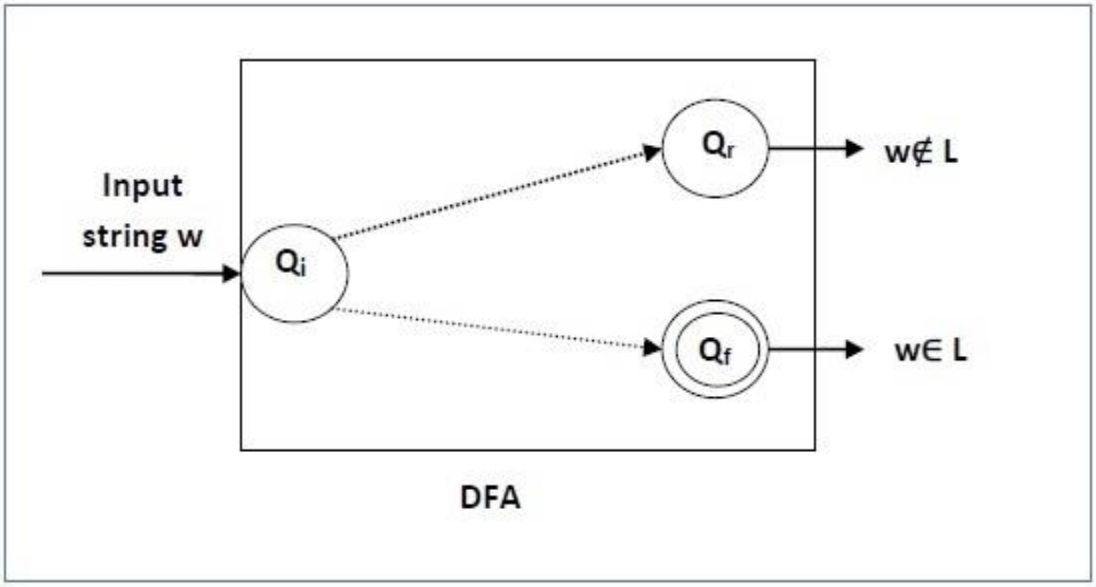
- Problem: Find out whether the following problem is decidable or not – Is a number 'm' prime?

- Solution: Prime numbers = {2, 3, 5, 7, 11, 13, ……}
- Divide the number **'m'** by all the numbers between '2' and '√m' starting from '2'.
- If any of these numbers produce a remainder zero, then it goes to the "Rejected state", otherwise it goes to the "Accepted state". So, here the answer could be made by 'Yes' or 'No'.
- **Hence, it is a decidable problem.**

Example 2

- Problem: Given a regular language **L** and string **w**, how can we check if **w ∈ L**?

- Solution: Take the DFA that accepts **L** and check if **w** is accepted



- Some more decidable problems are –
- Does DFA accept the empty language?
- Is $L_1 \cap L_2 = \emptyset$ for regular sets?

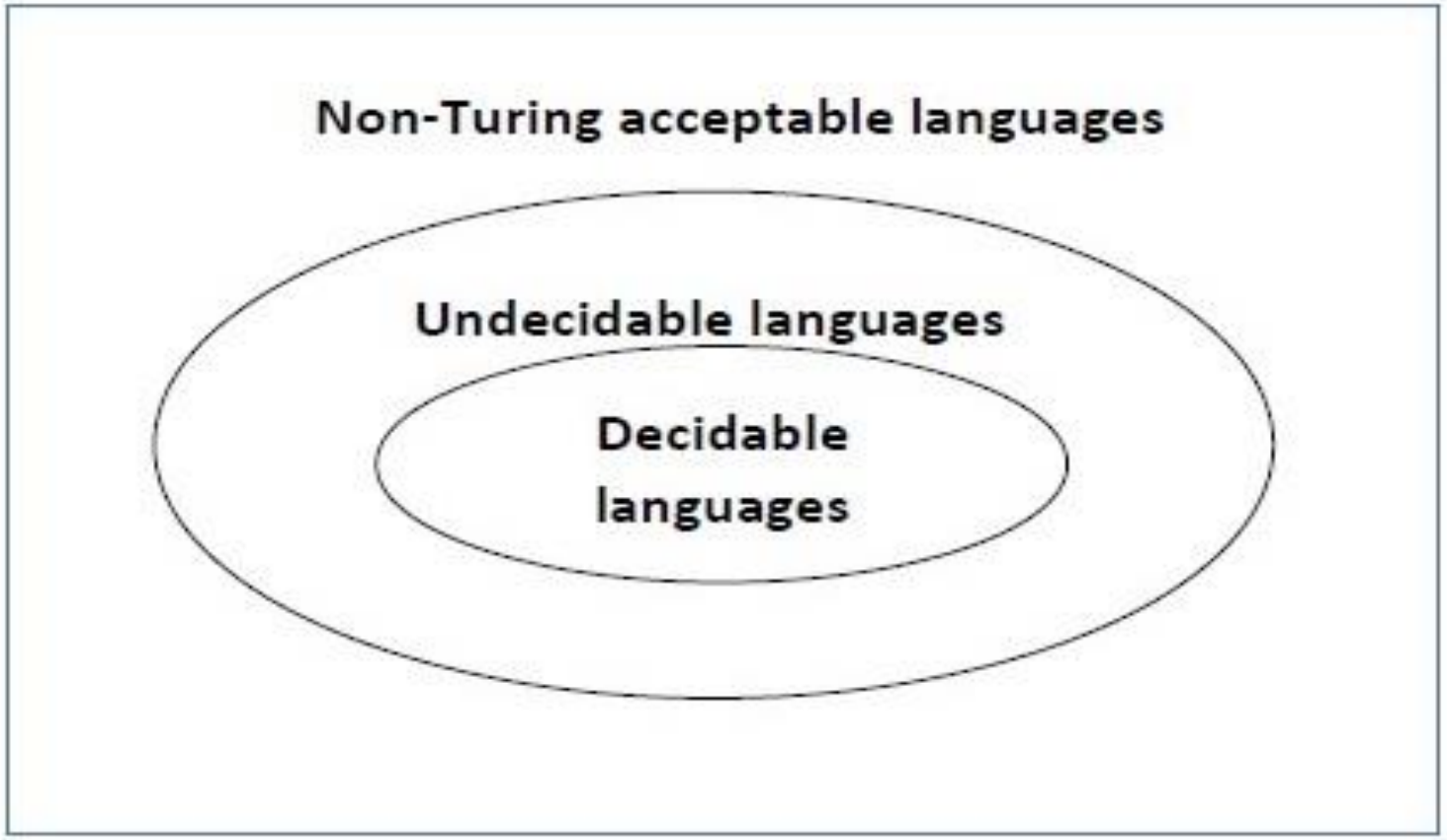
- **Note –**
- If a language **L** is decidable, then its complement **L'** is also decidable
- If a language is decidable, then there is an enumerator for it.

Undecidable Languages

- For an undecidable language, there is no Turing Machine which accepts the language and makes a decision for every input string **w** (TM can make decision for some input string though).

- A decision problem **P** is called "undecidable" if the language **L** of all yes instances to **P** is not decidable.

- Undecidable languages are not recursive languages, but sometimes, they may be recursively enumerable languages.


Non-Turing acceptable languages
Undecidable languages
Decidable languages

- **Example**

- The halting problem of Turing machine
- The mortality problem
- The mortal matrix problem
- The Post correspondence problem, etc.

Conclusion

- A problem is said to be **Decidable** if we can always construct a corresponding **algorithm** that can answer the problem correctly.

- A problem is said to be a Decidable problem if there exists a corresponding Turing machine which **halts** on every input with an answer- **yes or no**.

- The problems for which we can't construct an algorithm that can answer the problem correctly in finite time are termed as Undecidable Problems.

POLLING QUESTIONS

LOVELY PROFESSIONAL UNIVERSITY

1. A language L is said to be _____ if there is a turing machine M such that L(M)=L and M halts at every point.

   a) Turing acceptable
   b) decidable
   c) undecidable
   d) none of the mentioned

2. The problems which have no algorithm, regardless of whether or not they are accepted by a turing machine that fails to halts on some input are referred as:

a) Decidable
b) Undecidable
c) Computable
d) None of the mentioned

3.Recursive languages are also known as:

a) decidable
b) undecidable
c) sometimes decidable
d) none of the mentioned

4. Which of the following was not a part of Chomsky hierarchy ?

a) Context sensitive grammar
b) Unrestricted grammar
c) Recursive grammar
d) None of the mentioned

5. Decidable can be taken as a synonym to:

a) recursive
b) non recursive
c) recognizable
d) none of the mentioned

Computational Complexity

- In computer science, the **computational complexity** or simply **complexity** of an algorithm is the amount of resources required to run it. Particular focus is given to time and memory requirements.

LOVELY PROFESSIONAL UNIVERSITY

Measuring Time Complexity

- Time complexity of an algorithm signifies the total time required by the program to run till its completion.
- The time complexity of algorithms is most commonly expressed using the **big O notation**. It's an asymptotic notation to represent the time complexity.
- Time Complexity is most commonly estimated by counting the number of elementary steps performed by any algorithm to finish execution.

LOVELY PROFESSIONAL UNIVERSITY
PUNJAB (INDIA)
LOVELY PROFESSIONAL UNIVERSITY

Measuring Space Complexity

- Space complexity of an algorithm represents the amount of memory space needed the algorithm in its life cycle.
- Space needed by an algorithm is equal to the sum of the following two components
- A fixed part that is a space required to store certain data and variables (i.e. simple variables and constants, program size etc.), that are not dependent of the size of the problem.
- A variable part is a space required by variables, whose size is totally dependent on the size of the problem. For example, recursion stack space, dynamic memory allocation etc.

Space Time Tradeoff

- In computer science, a **space-time** or **time-memory tradeoff** is a way of solving a problem or calculation in less time by using more storage space (or memory), or by solving a problem in very little space by spending a long time.

- Most computers have a large amount of space, but not infinite space.

- A space-time tradeoff can be used with the problem of data storage.

- If data is stored uncompressed, it takes more space but less time than if the data were stored compressed.

POLLING QUESTIONS

- The class of recursively enumerable language is known as:
  a) Turing Class
  b) Recursive Languages
  c) Universal Languages
  d) RE

- Which of the following statements are false?
  a) Every recursive language is recursively enumerable
  b) Recursively enumerable language may not be recursive
  c) Recursive languages may not be recursively enumerable
  d) None of the mentioned

- A Turing machine operates over:
  a) finite memory tape
  b) infinite memory tape
  c) depends on the algorithm
  d) none of the mentioned

- Turing machine can be represented using the following tools:
  a) Transition graph
  b) Transition table
  c) Queue and Input tape
  d) All of the mentioned

- X is a simple mathematical model of a computer. X has unrestricted and unlimited memory. X is a FA with R/W head. X can have an infinite tape divided into cells, each cell holding one symbol. Name X?

  a) Push Down Automata
  b) Non deterministic Finite Automata
  c) Turing machines
  d) None of the mentioned

- Which of the following statements is/are true?
  a) Every multitape turing machine has its equivalent single tape turing machine
  b) Every multitape turing machine is an abstract machine
  c) Both (a) and (b)
  d) None of the mentioned

- Which of the following is/are a basic TM equivalent to?
  a) Multitrack TM
  b) Multitape TM
  c) Non-deterministic TM
  d) All of the mentioned

Cellular Automata

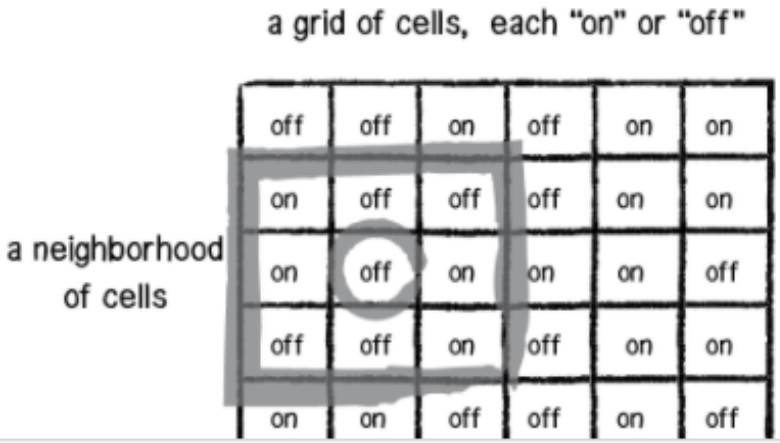
- A **cellular automaton** (pl. **cellular automata**, abbrev. **CA**) is a discrete model of computation studied in automata theory.

- A cellular automaton consists of a regular grid of *cells*, each in one of a finite number of *states*, such as *on* and *off*.

- The grid can be in any finite number of dimensions. For each cell, a set of cells called its *neighborhood* is defined relative to the specified cell.

A cellular automaton is a model of a system of "cell" objects with the following characteristics.

- The cells live on a ***grid***. (We'll see examples in both one and two dimensions in this chapter, though a cellular automaton can exist in any finite number of dimensions.)
- Each cell has a ***state***. The number of state possibilities is typically finite. The simplest example has the two possibilities of 1 and 0 (otherwise referred to as "on" and "off" or "alive" and "dead").
- Each cell has a ***neighborhood***. This can be defined in any number of ways, but it is typically a list of adjacent cells.



POLLING QUESTIONS

- Cellular automata produce

  a) exhaustive patterns
  b) exhaustive pseudo random patterns
  c) random patterns
  d) pseudo random patterns

- A **space-time** or **time-memory tradeoff** is a way of solving a problem or calculation in:

a) Less time
b) More time
c) Both a or b
d) None of these

- In cellular automata, the grid can have _____no of dimensions

a) Infinite
b) Finite
c) Both a and b
d) None of these

- Space needed by an algorithm is equal to____

a)Fixed part

b) Variable part

c) Both a and b

d) None of these