UNIT -7:INTRODUCTION TO TURING MACHINES

7.1 problems that computers cannot solve
7.2 The turing machine
7.3programming techniques for turing machines
7.4 extensions to the basic turing machines
7.5 turing machines and computers

7.1 :Problems that computers cannot solve

7.2 The Turing machine

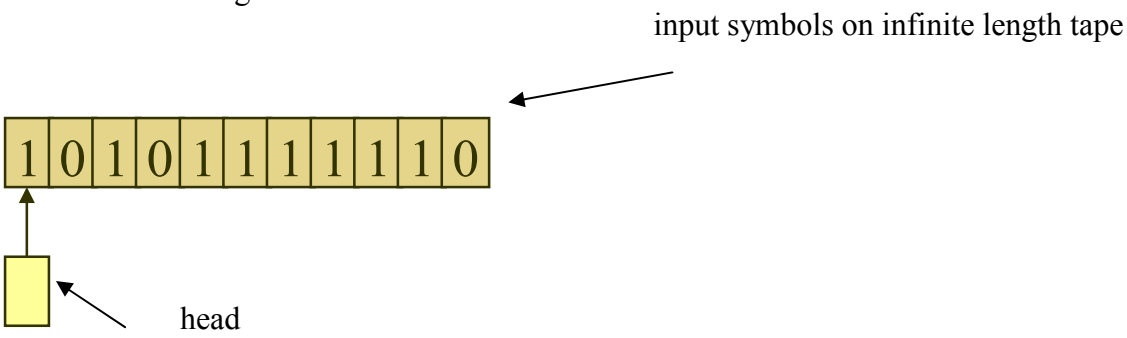
Definition:
A Turing Machine (TM) is an abstract, mathematical model that describes what can and cannot be computed. A Turing Machine consists of a tape of infinite length, on which input is provided as a finite sequence of symbols. A *head* reads the input tape. The Turing Machine starts at "start state" S_0. On reading an input symbol it optionally replaces it with another symbol, changes its internal state and moves one cell to the right or left.

Notation for the Turing Machine :

TM = <S, T, S_0, □, H> where,

S	is a set of TM states
T	is a set of tape symbols
S_0	is the start state
H □ S	is a set of halting states
□ : S x T □S x T x {L,R}	is the transition function
{L,R}	is direction in which the head moves

L : Left R: Right



The Turing machine model uses an infinite tape as its unlimited memory. (This is important because it helps to show that there are tasks that these machines cannot perform, even though unlimited memory and unlimited time is given.) The input symbols occupy some of the tape's cells, and other cells contain blank symbols.

Some of the characteristics of a Turing machine are:

1. The symbols can be both read from the tape and written on it.
2. The TM head can move in either directions – Left or Right.
3. The tape is of infinite length
4. The special states, Halting states and Accepting states, take immediate effect.

Solved examples:

TM Example 1:

Turing Machine U+1:

Given a string of 1s on a tape (followed by an infinite number of 0s), add one more 1 at the end of the string.

Input : #111100000000…….

Output : #1111100000000…….

Initially the TM is in Start state S_0. Move right as long as the input symbol is 1. When a 0 is encountered, replace it with 1 and halt.
Transitions:
$(S_0, 1) \longrightarrow (S_0, 1, R)$
$(S_0, 0) \longrightarrow (h, 1, STOP)$

TM Example 2 :

TM: X-Y
Given two unary numbers x and y, compute |x-y| using a TM. For purposes of simplicity we shall be using multiple tape symbols.

Ex: 5 (11111) – 3 (111) = 2 (11)
#11111b1110000…..
#___11b___000…

a) Stamp out the first 1 of x and seek the first 1 of y.

$(S_0, 1) \longrightarrow (S_1, \_, R)$
$(S_0, b) \longrightarrow (h, b, STOP)$
$(S_1, 1) \longrightarrow (S_1, 1, R)$
$(S_1, b) \longrightarrow (S_2, b, R)$

b) Once the first 1 of y is reached, stamp it out. If instead the input ends, then y has finished. But in x, we have stamped out one extra 1, which we should replace. So, go to some state s5 which can handle this.

$(S_2, 1) \longrightarrow (S_3, \_, L)$
$(S_2, \_) \longrightarrow (S_2, \_, R)$
$(S_2, 0) \longrightarrow (S_5, 0, L)$

c) State s3 is when corresponding 1s from both x and y have been stamped out. Now go back to x to find the next 1 to stamp. While searching for the next 1 from x, if we reach the head of tape, then stop.

$(S_3, \_) \longrightarrow (S_3, \_, L)$
$(S_3, b) \longrightarrow (S_4, b, L)$

$(S_4, 1) \longrightarrow (S_4, 1, L)$
$(S_4, \_) \longrightarrow (S_0, \_, R)$
$(S_4, \#) \longrightarrow (h, \#, STOP)$

d) State s5 is when y ended while we were looking for a 1 to stamp. This means we have stamped out one extra 1 in x. So, go back to x, and replace the blank character with 1 and stop the process.

$(S_5, \_) \longrightarrow (S_5, \_, L)$
$(S_5, b) \longrightarrow (S_6, b, L)$
$(S_6, 1) \longrightarrow (S_6, 1, L)$
$(S_6, \_) \longrightarrow (h, 1, STOP)$

Solved examples:

TM Example 1: Design a Turing Machine to recognize $0^n1^n2^n$
ex: #000111222_ _ _ _ _…….

Step 1: Stamp the first 0 with X, then seek the first 1 and stamp it with Y, and then seek the first 2 and stamp it with Z and then move left.

$(q_0, 0) \rightarrow (q_1, X, R)$
$(q_1, 0) \rightarrow (q_1, 0, R)$
$(q_1, 1) \rightarrow (q_2, Y, R)$
$(q_2, 1) \rightarrow (q_2, 1, R)$
$(q_2, 2) \rightarrow (q_3, Z, L)$

S_0 = Start State, seeking 0, stamp it with X
S1 = Seeking 1, stamp it with Y
S2 = Seeking 2, stamp it with Z

Step 2: Move left until an X is reached, then move one step right.

$(q_3, 1) \rightarrow (q_3, 1, L)$
$(q_3, Y) \rightarrow (q_3, Y, L)$
$(q_3, 0) \rightarrow (q_3, 0, L)$
$(q_3, X) \rightarrow (q_0, X, R)$

S3 = Seeking X, to repeat the process.

Step 3: Move right until the end of the input denoted by blank(_) is reached passing through X Y Z s only, then the accepting state S_A is reached.

$(S_4, X) \rightarrow (S_4, X, R)$
$(S_4, Y) \rightarrow (S_4, Y, R)$
$(S_4, Z) \rightarrow (S_4, Z, R)$
$(S_4, \_) \rightarrow (S_5, \_, R)$

S4 = Seeking blank

These are the transitions that result in halting states.

$(S_4, 1) \rightarrow (h, 1, R)$
$(S_4, 2) \rightarrow (h, 2, R)$
$(S_4, \_) \rightarrow (S_5, \_, R)$
$(S_0, 1) \rightarrow (h, 1, R)$
$(S_0, 2) \rightarrow (h, 2, R)$
$(S_1, 2) \rightarrow (h, 2, R)$
$(S_2, \_) \rightarrow (h, \_, R)$

TM Example 2 : Design a Turing machine to accept a Palindrome

ex: #1011101_ _ _ _ _…….
Step 1: Stamp the first character (0/1) with _, then seek the last character by moving till a _ is reached. If the last character is not 0/1 (as required) then halt the process immediately.

$(S_0, 0) \rightarrow (S_1, \_, R)$
$(S_0, 1) \rightarrow (S_2, \_, R)$
$(S_1, \_) \rightarrow (S_3, \_, L)$
$(S_3, 1) \rightarrow (h, 1, R)$
$(S_2, \_) \rightarrow (S_5, \_, L)$
$(S_5, 0) \rightarrow (h, 0, R)$

Step 2: If the last character is 0/1 accordingly, then move left until a blank is reached to start the process again

$$(q_3, 0) \rightarrow (q_4, \_)$$
$$(q_4, 1) \rightarrow (q_4, 1, R)$$
$$(q_4, 0) \rightarrow (q_4, 0, R)$$
$$(q_4, \_) \rightarrow (q_6, \_)$$
$$(q_5, 1) \rightarrow (q_6, \_)$$
$$(q_6, 1) \rightarrow (q_6, 1, L)$$
$$(q_6, 0) \rightarrow (q_6, 0, L)$$
$$(q_6, \_) \rightarrow (q_0, \_)$$

Step 3 : If a blank (_) is reached when seeking next pair of characters to match or when seeking a matching character, then accepting state is reached.

$$(q_3, \_) \rightarrow (q_A, \text{ACCEPT})$$
$$(q_5, \_) \rightarrow (q_A, \text{ACCEPT})$$
$$(q_6, \_) \rightarrow (q_A, \text{ACCEPT})$$

The sequence of events for the above given input are as follows:

$\#s_0 10101\_\_\_$

$\#\_s_2 0101\_\_\_$

$\#\_0s_2 101\_\_\_$

. . . .

$\#\_0101s_5\_\_\_$

$\#\_010s_6\_\_\_\_$

$\#\_s_6 0101\_\_\_$

$\#\_s_0 0101\_\_\_$

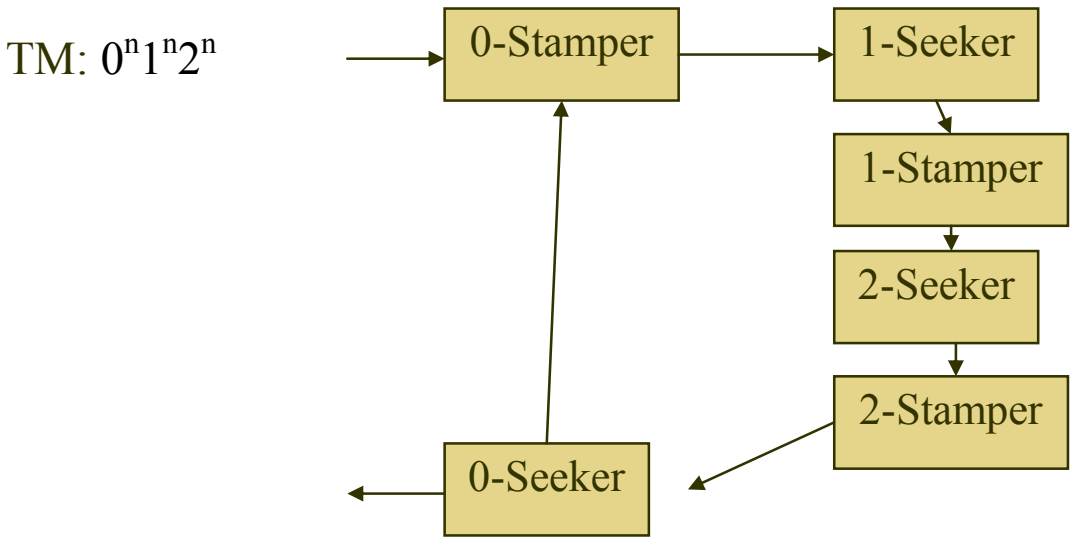
. . . .

$\#\_\_\_\_ s_5 \_\_\_\_\_$

$\#\_\_\_\_ s_A \_\_\_\_\_$

Modularization of TMs

Designing complex TM s can be done using modular approach. The main problem can be divided into sequence of modules. Inside each module, there could be several state transitions.
For example, the problem of designing Turing machine to recognize the language $0^n1^n2^n$ can be divided into modules such as 0-stamper, 1-stamper, 0-seeker, 1-seeker, 2-seeker and 2-stamper. The associations between the modules are shown in the following figure:



Load → Decode → Execute → Store

Universal Turing Machine

A Universal Turing Machine UTM takes an encoding of a TM and the input data as its input in its tape and behaves as that TM on the input data.

A TM spec could be as follows:

TM = (S,S0,H,T,d)
Suppose, S={a,b,c,d}, S0=a, H={b,d} T={0,1}
δ : (a,0) (b,1,R) , (a,1) (c,1,R) ,
(c,0) (d,0,R) and so on
then TM spec:
$abcd$abd01$a0b1Ra1c1Rc0d0R…….
where **$** is delimiter

This spec along with the actual input data would be the input to the UTM.
This can be encoded in binary by assigning numbers to each of the characters appearing in the TM spec.

The encoding can be as follows:

$: 0000	0 : 0101
a : 0001	1 : 0110
b : 0010	L : 0111
c : 0011	R : 1000
d : 0100	

So the TM spec given in previous slide can be encoded as:

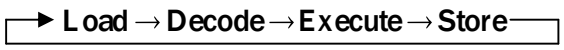
0000.0001.0010.0011.0100.0000.0001.0000.0010.0100 ……

Hence TM spec can be regarded just as a number.

Sequence of actions in UTM:
Initially UTM is in the start state S0.

- Load the input which is TM spec.
- Go back and find which transition to apply.
- Make changes, where necessary.
- Then store the changes.
- Then repeat the steps with next input.

Hence, the sequence goes through the cycle:



7.3:Extensions to Turing Machines

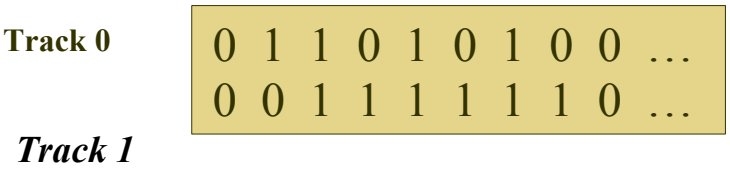
Proving Equivalence

For any two machines M_1 from class C_1 and M_2 from class C_2:

M_2 is said to be at least as expressive as M_1
if $L(M_2) = L(M_1)$ or if M_2 can *simulate* M_1.

M_1 is said to be at least as expressive as M_2
if $L(M_1) = L(M_2)$ or if M_1 can simulate M_2.

Composite Tape TMs



A composite tape consists of many *tracks* which can be read or written *simultaneously.*

A composite tape TM (CTM) contains more than one tracks in its tape.

Equivalence of CTMs and TMs

A CTM is simply a TM with a complex alphabet..

T = {a, b, c, d}
T' = {00, 01, 10, 11}

Turing Machines with Stay Option

Turing Machines with stay option has a third option for movement of the TM head:
left, right or *stay*.

STM = <S, T, □, s_0, H>

□: S x T à S x T x {L, R, S}

Equivalence of STMs and TMs

STM = TM:
Just don't use the S option…

TM = STM:

For L and R moves of a given STM build a TM that moves correspondingly L or R…

TM = STM:

For S moves of the STM, do the following:
1.Move right,
2.Move back left without changing the tape
3.STM: □(s,a) |-- (s',b,S)

TM: □(s,a) |-- (s'', b, R)
□(s'',*) |-- (s',*,L)

2-way Infinite Turing Machine

In a 2-way infinite TM (2TM), the tape is infinite on both sides.
There is no # that delimits the left end of the tape.

Equivalence of 2TMs and TMs

2TM = TM:
Just don't use the left part of the tape…
TM = 2TM:
Simulate a 2-way infinite tape on a one-way infinite tape…

Multi-tape Turing Machines

A multi-tape TM (MTM) utilizes many tapes.

Equivalence of MTMs and TMs

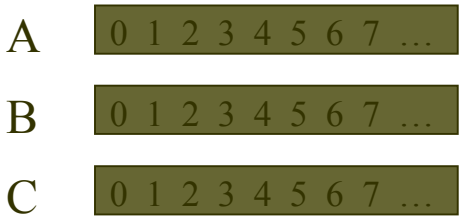
MTM = TM:

Use just the first tape…

TM = MTM:

Reduction of multiple tapes to a single tape.

Consider an MTM having *m* tapes. A single tape TM that is equivalent can be constructed by reducing *m* tapes to a single tape.



Non-deterministic TM

A non-deterministic TM (NTM) is defined as:

NTM = <S, T, s_0, □, H>

where □: S x T à $2^{SxTx\{L,R\}}$

Ex: (s_2,a) à {(s_3,b,L) (s_4,a,R)}

Equivalence of NTMs and TMs

A "concurrent" view of an NTM:

(s_2,a) à {(s_3,b,L) (s_4,a,R)}

è at (s_2,a), two TMs are spawned:

(s_2,a) à (s_3,b,L)

(s_2,a) à (s_4,a,R)