TURING MACHINE

Turing machine was invented in 1936 by Alan Turing. It is an accepting device which accepts Recursive Enumerable Language generated by type 0 grammar.

There are various features of the T.M.-

1. It has an external memory which remembers arbitrary long sequence of input.
2. It has unlimited memory capability.
3. The tape head has the capability to move left and right.
4. The machine can produce a certain output based on its input. Sometimes it may be required that the same input has to be used to generate the output. So in this machine, the distinction b/w input and output has been removed. Thus a common set of alphabets can be used for the T.M.

⁂ Formal definition of a Turing Machine-

A Turing Machine $M=(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is a 7-tuple structure where-

Q - finite set of states.
Σ - is the character set of language.
Γ - set of tape symbol ; $\Gamma = \Sigma \cup \{B\}$ where B is blank symbol.
δ - set of transition function.
q_0 - initil state.
B - blank symbol.
F - set of final states $F \subseteq Q$.

"If a Turing Machine halts in final state, then the string is accepted."

✱ Basic Model of Turing Machine-

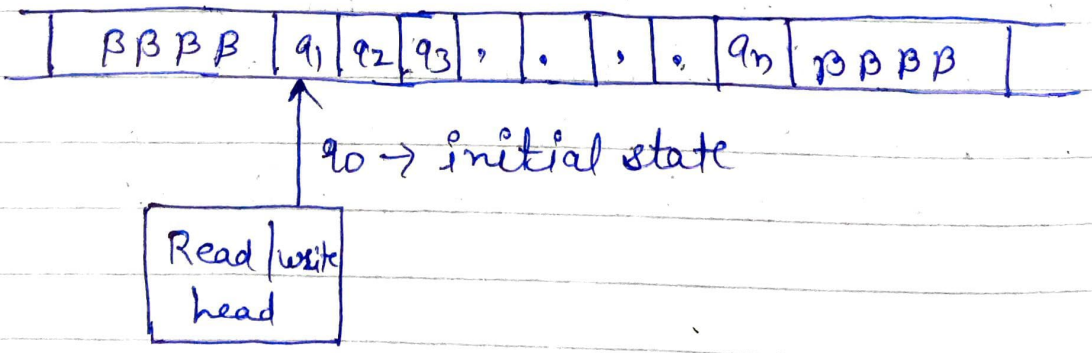


Fig- Basic model of Turing Machine

Turing Machine has an infinite tape divided into cells it can be infinite in both directions except for some finite portion $a_1, a_2, \ldots\ldots a_n$. The remaining will be blank. Blank is denoted by β.

The following notations are used to denote the direction of movement of the tape head.

L - left move
S - stationary
R - Right move

Transition fun - $\delta(q_i, a) \vdash (q_j, b, R)$

It indicates that if the symbol in current cell is a and the state under R/W head is q_i, then the new state under R/W head is q_j, and the current cell is changed to b by moving in right direction.

★ Turing Machine as Language Acceptors:-

A turing machine halts when it no longer has any available moves. If it halts in a final state, it accepts its input; otherwise, it rejects its input.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is a T.M, the language accepted by M. denoted by $L(M)$ is the set of those string in Σ^* that cause M to enter in final state.

A string w in the TM is said to be accepted if -

$$q_0 w \vdash^* \alpha_1 p \alpha_2 \text{ for some } p \in F$$

and $\alpha_1, \alpha_2 \in \Gamma^*$

The T.M does not accept string w if the machine halts in a non accepting state or does not halt.

Every TM divides the set of input string over the Σ in three classes:-

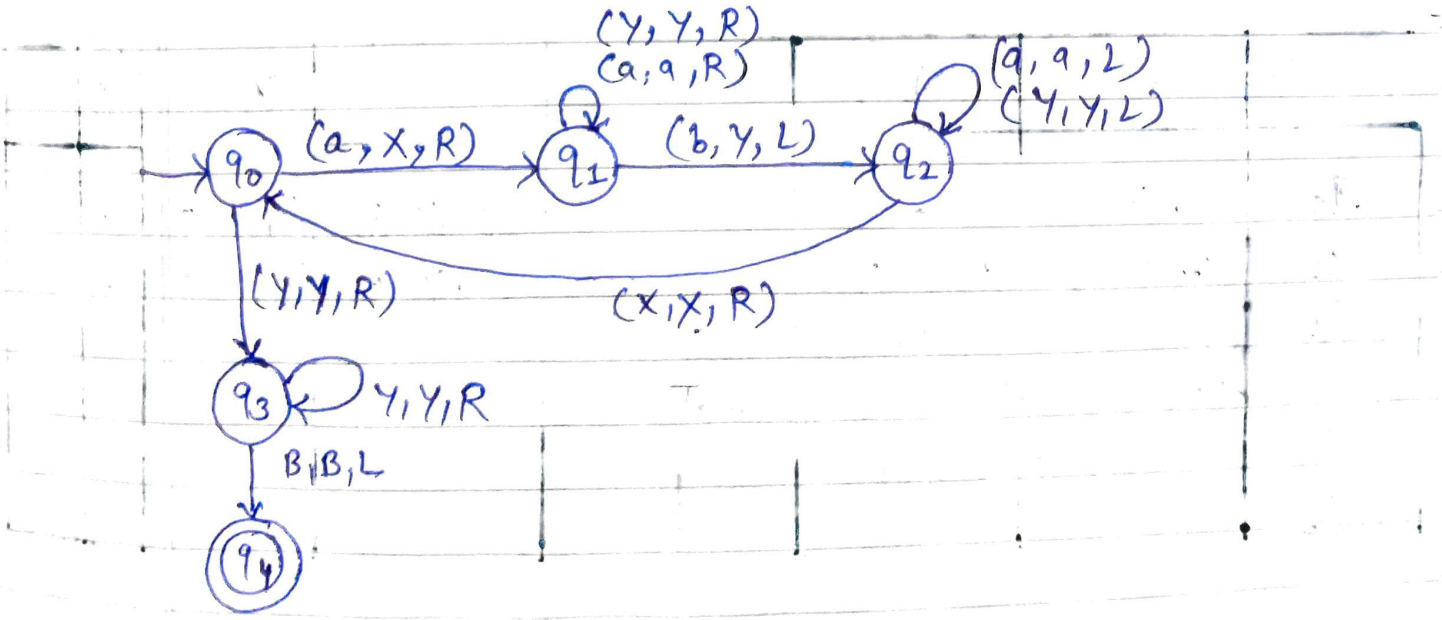
1. Accept :- This is the set of string which cause the TM to halt in some final state. (also called halting state).

DATE : / /
PAGE NO. :

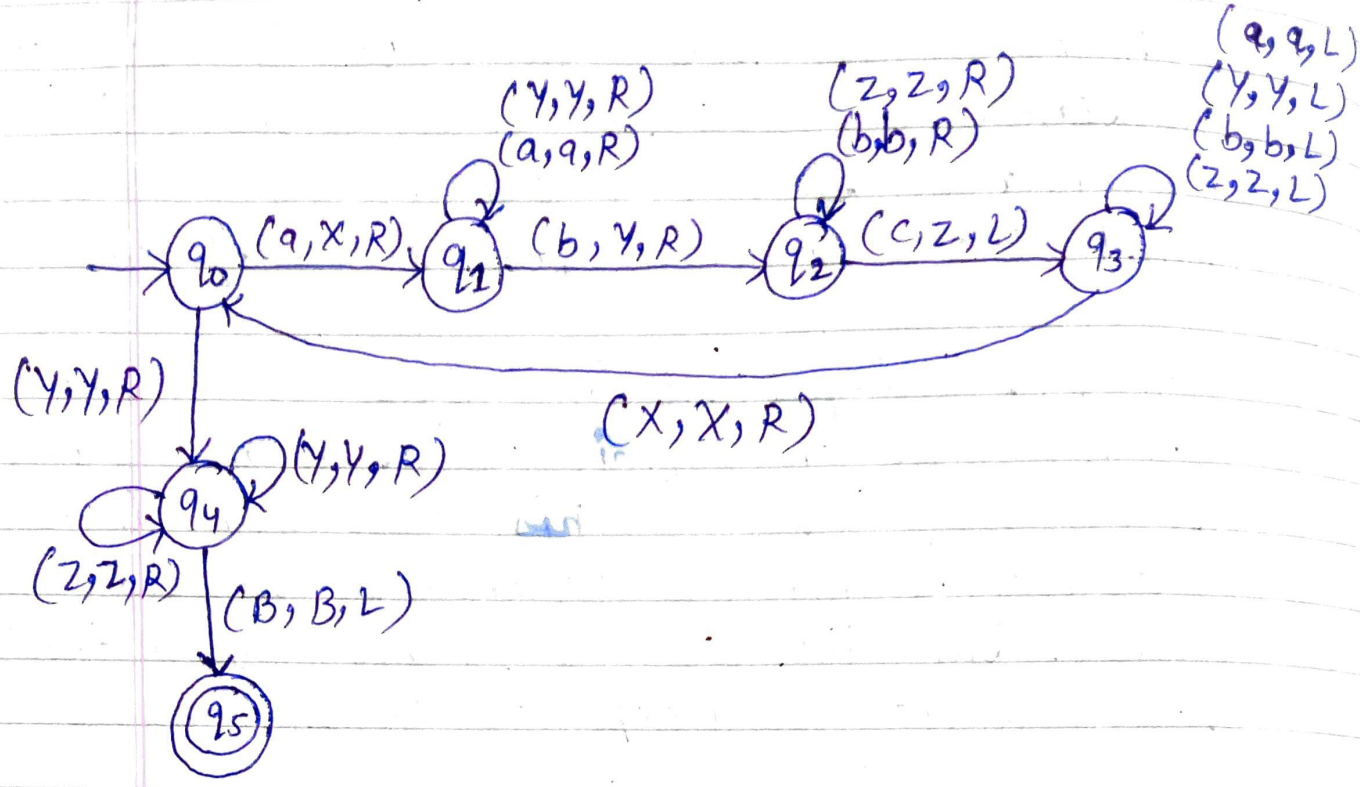
② **Reject:-** It may possible that the input string may lead to some configuration with non-final state from which the TM can not move i.e. to some combination of non final state and current symbol for which the transition fun. is undefined.

③ **Loop:-** An input string might cause the TM to enter an infinite loop, a never ending sequence of moves. In this case there is no outcome. The machine continues to make moves, but the observer is never sure that it is about to accept or crash.

Ex① Design a TM. for $\{a^n b^n \mid n >= 1\}$



Ex② Design a TM for $\{a^n b^n c^n \mid n >= 1\}$



Transition table for $\{a^n b^n c^n \mid n >= 1\}$

state	a	b	c	X	Y	Z	B
q_0	(q_1, X, R)	–	–	–	(q_4, Y, R)	–	–
q_1	(q_1, a, R)	(q_2, Y, R)	–	–	(q_1, Y, R)	–	–
q_2	–	(q_2, b, R)	(q_3, Z, L)	–	–	(q_2, Z, R)	–
q_3	(q_3, a, L)	(q_3, b, L)	–	(q_0, X, R)	(q_3, Y, L)	(q_3, Z, L)	–
q_4	–	–	–	–	(q_4, Y, R)	(q_4, Z, R)	(q_5, B, L)
q_5	–	–	–	–	–	–	–

✱ Turing Machine as Transducers:-

TM are more than just language acceptors. To use TM as transducer, treat the entire nonblank portion of the tape as input, and treat the entire nonblank portion of the tape as output when the machine halts.

A TM defines a fun. $y = f(x)$ for string $x, y \in \Sigma^*$ if

$$q_0 x \vdash^* q_f y$$

where q_f is a final state.

Turing Computable:- A function f is said to be Turing-computable, or just computable, if there exists some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ such that

$$q_0 w \vdash^* q_f f(w), \quad q_f \in F$$

for all $w \in D$

Ex① Design a TM. for 1's complement-

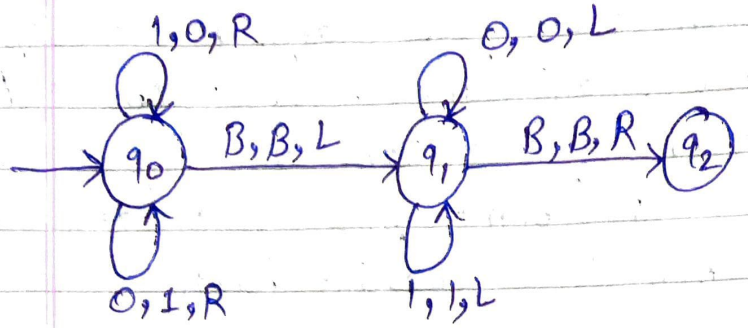
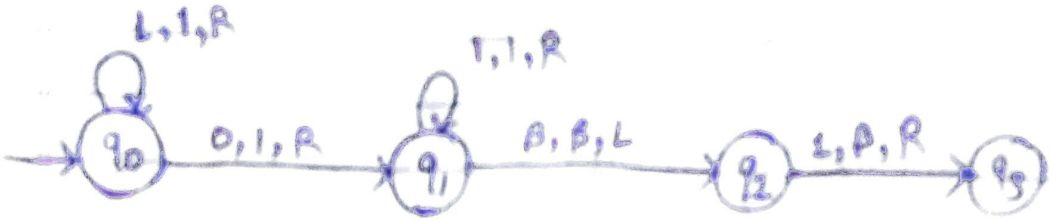


Fig- Transition diagram for 1's complement-

Transition table-

states	0	1	B
q_0	$1Rq_0$	$0Rq_0$	BLq_1
q_1	$0Lq_1$	$1Lq_1$	BRq_2
q_2	–	–	–

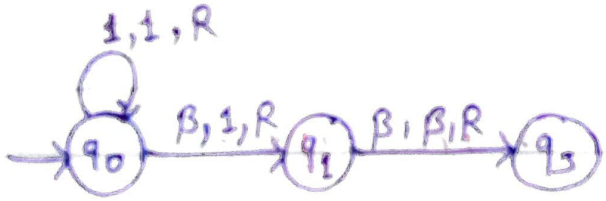
Ex② Design a TM. to find sum of 2 unary no.



state	0	1	B
q_0	$1Rq_1$	$1Rq_0$	–
q_1	–	$1Rq_1$	BLq_2
q_2	–	BRq_3	–
q_3	–	–	–

Ex③ Design a T.M. to increment a unary number. i.e.

$$f(n) = n+1$$



state	1	B
q_0	$1Rq_0$	$1Rq_1$
q_1	–	BRq_2
q_2	–	–

Modification to standard Turing Machine:-

Many modifications and enhancements have been proposed to improve TM. These enhancement are in terms of the no. of tapes, no. of tracks on the tape, allowable moves, attachment of memory etc.

① TM with stay option:- If instead of moving left or right on seeing an input, the head could also stay at one position without moving anywhere.

$$f: Q \times X \rightarrow Q \times X \times \{\text{left\_shift}, \text{Right\_shift}, \text{stay}\}$$

Still the no. of languages accepted by TM remains same.

② TM with semi-infinite tape - We know that TM has an infinite input tape with extends in both the directions (left & right) infinitely. So now if we restrict it to extend only in one direction & not in both the directions, i.e., we make the tape to be semi infinite, then also the no. of languages accepted by the TM remains same.

③ Offline T.M – In standard TM both the input and output are present on the tape, the head has the authority to move across the input and can change or modify the input, if we don't modify the input we can provide the input in separate file, which is read only then the head cannot make change to it.

If the TM wants to modify the input, the input need to be copied on the tape and the changes can be made by the head but still the input file remains unchanged as changes are made in the tape and not in the file. By doing such modification to TM still the no. of languages accepted by the TM remains the same.

④ Jumping TM – The standard TM's head can move only one step to the right or left, but in case of jumping TM the head can move not only just one step to the left or right but it can move more the one, i.e., 2, 3, 4, 5, 6, ... so on, or it can jump to any cell to the right or left of the input tape.

$f: Q \times X \rightarrow Q \times X \{ \text{Left\_shift}, \text{Right\_shift} \} \times \{n\}$

n, is the no of steps that we wish to move to the right or left. But still the languages accepted by TM. remains the same.

⑤ Non erasing TM – In standard TM the input symbol can be changed to blank, but if we remove this facility to changing the input symbol to blank then such type of TM is called as non-erasing T.M. We can replace the input with any other symbol except blank. By doing this modification still the no. of languages accepted by the TM. remains the same.

⑥ Always writing TM – Standard TM gives us the the freedom than on seeing an input we can leave as it is without doing any changes but in always writing TM it is compulsory to modify the input whenever we see it we cannot leave it as it is. This kind of modification didn't help in increasing the number of languages accepted by the TM.

⑦ Multitrack TM – A multitrack TM consists of multiple track on a single tape. In multitrack TM all the track reside on the same turing tape and move in same direction simultaneously i.e. – "the direction of motion of turing tape." The R/W head is capable of reading and writing from all the cells simultaneously which lie in the same column. By doing this modification to TM the no. of language accepted by TM remain

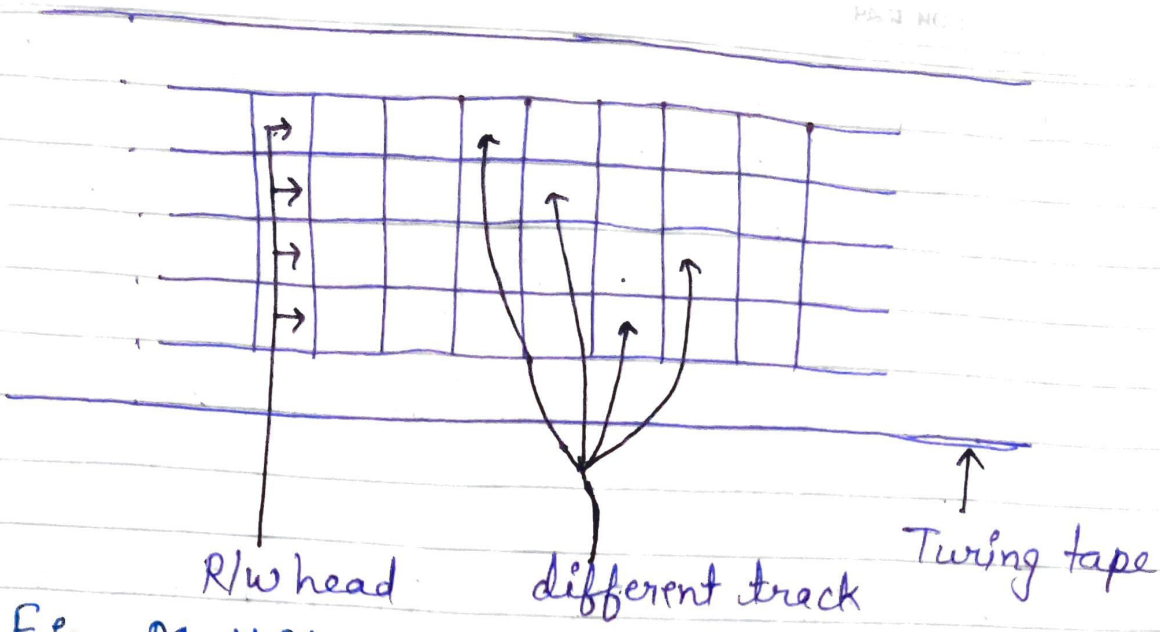


Fig- Multitrack Turing Machine.

⑧ Multitape TM- In multitape TM, each track reside on a different tape and can move in the desired direction independent of others. If the movement is not required then we make stationary moves

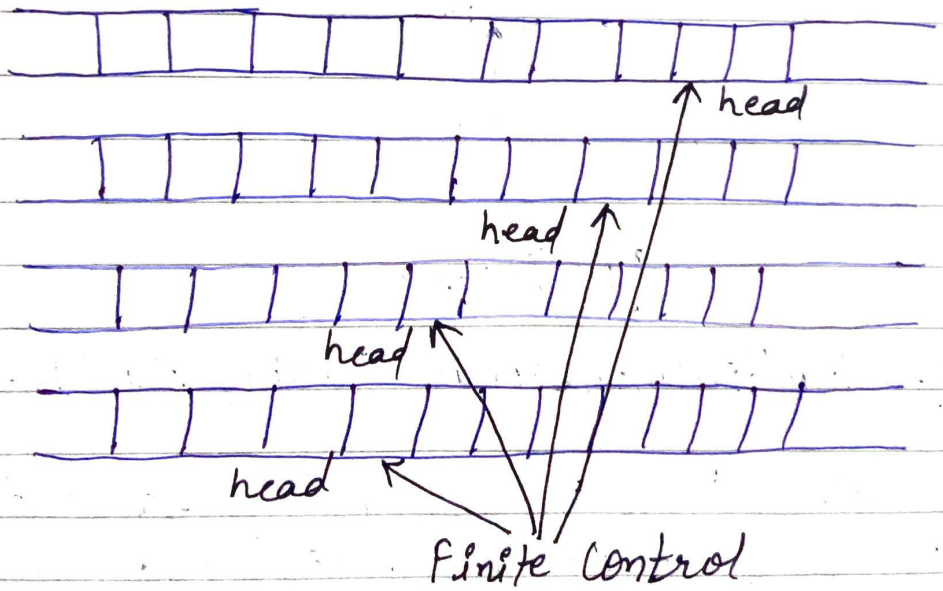


Fig- Multitape Turing Machine.

By doing this modification still the no. of languages accepted by TM remains same.

② Multidimensional TM :- In this TM the input tape can be viewed as extending infinitely in more than one direction (dimension - left / right / up / down)

The moves for multidimensional are formally defined as follows -

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D, stop\}$$

Here -

L = Left direction
R = Right direction
U = Upper direction
D = Down direction

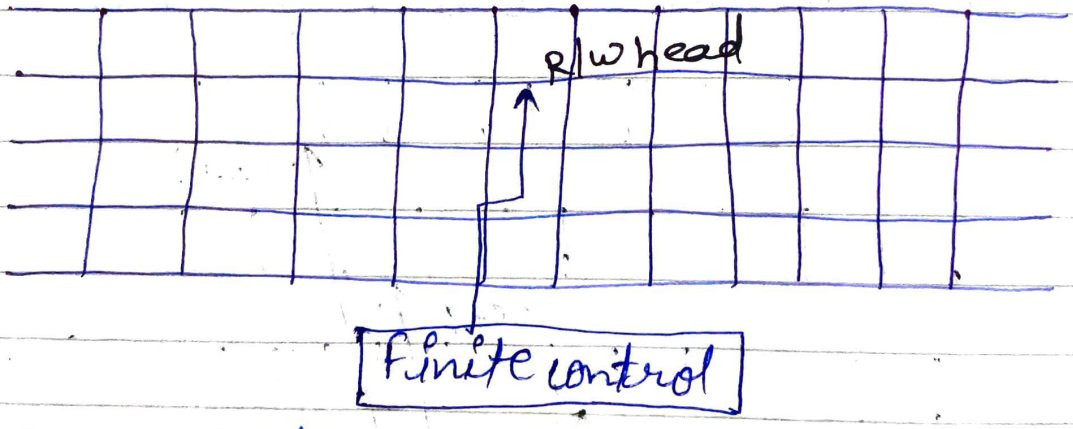


Fig- Multidimensional Turing Machine.

By doing this modification still the number of languages accepted by the TM remains the same.

(10) Nondeterministic TM – In a NDTM for every state and symbol, there are a group of actions the TM can have. So here the transitions are not deterministic. In DTM the next move of a TM is determined uniquely by δ. In NDTM we allow no move or multiple moves.

The computation of DTM can be viewed as a "tree" with only one branch, whereas the computation of an NDTM can be viewed as a "tree" having many possible branches.

The transition fun of an NDTM is as –

$$\delta(q_i, k) \vdash \{(q_1, a_1, D_1), (q_2, a_2, D_2) \ldots\ldots (q_n, a_n, D_n)\}$$

where Di is the direction of motion.

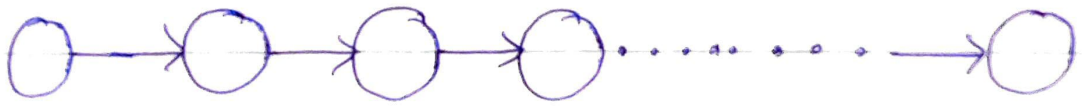


Fig - Deterministic Computation – "Only one next possible configuration at each step in computation".

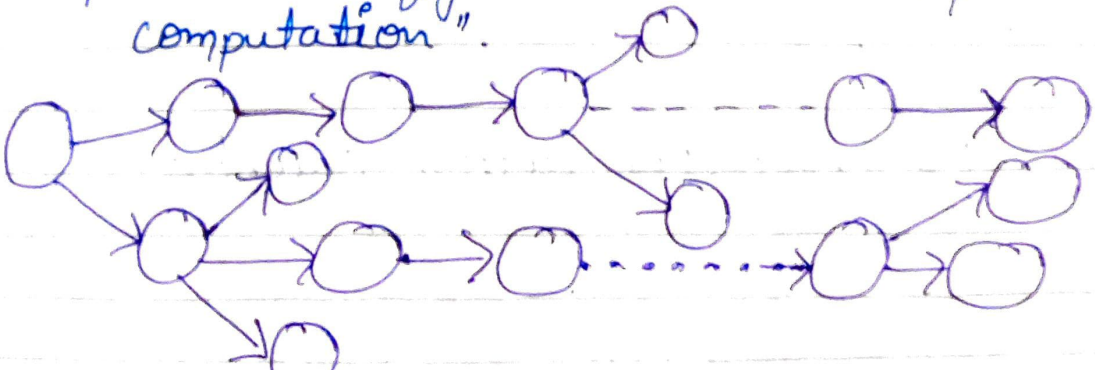


Fig - Nondeterministic Computation – At certain point computation may have one or more branches.

(11) Universal TM - Universal TM is a specified TM that can simulate the behaviour of any TM.

A UTM is a TM whose input consists of two parts -

(a) First part is the encoded TM followed by a marker.

(b) The second part is a string that is interpreted as the input of TM encoded in first part.

The UTM then simulates the processing of second part by the encoded TM in the first part of input.

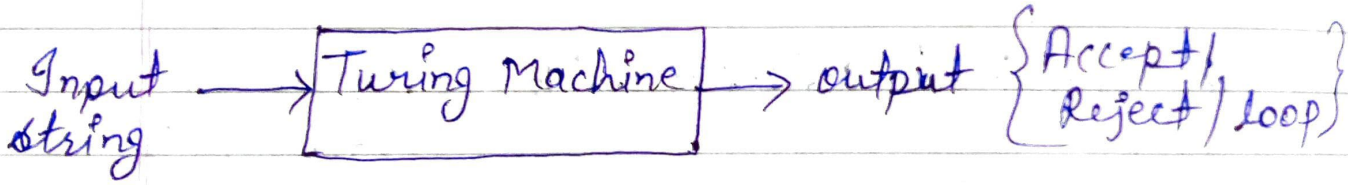


Fig- General Turing Machine

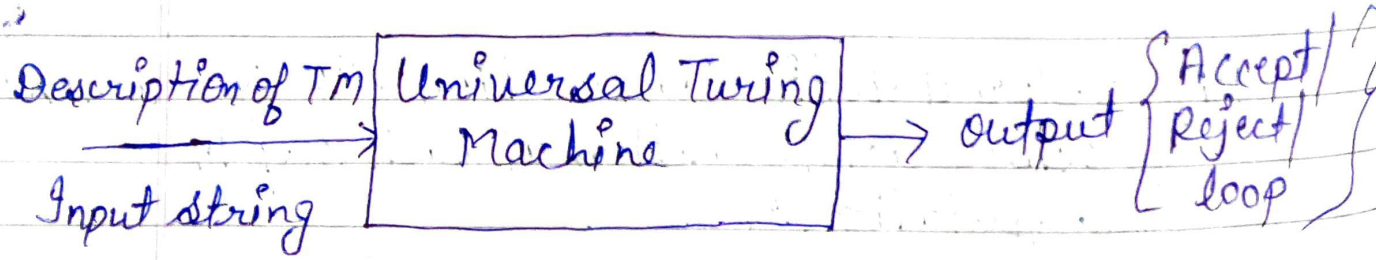
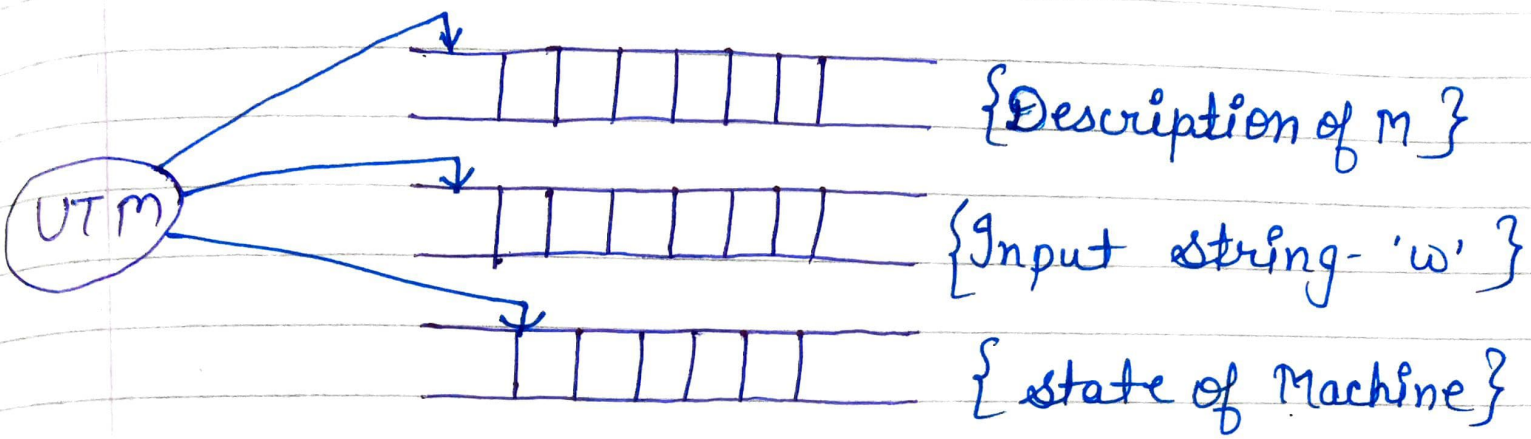


Fig- Universal Turing Machine.

To design UTM we use multitape TM because along with input string w we also give description of TM.



Let M_u be the UTM and input of M_u is the encoded Turing Machine M_1 followed by input w. M_u will simulate and Run TM on input.

(a) If M_1 would have crash (reject) on the input - M_u will crash (reject)
(b) If M_1 would accept the string w - M_u will accept
(c) If M_1 would loop on w - M_u will loop on w.

Recursive and Recursively Enumerable Language:-

There are three possible outcomes of executing a TM over a input. The TM may-
- → Halt and accept the input,
- → Halt and reject the input,
- → Never halt.

✱ Recursive Language:-

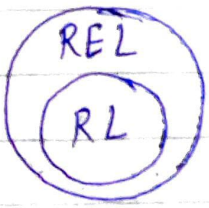
→ A language 'L' is said to be recursive if there exists a TM which will accept all the strings in L and reject all the strings not in L.

→ The TM will halt every time and give an answer (accepted or rejected) for each and every string input.

✱ Recursively Enumerable Language:-

→ A language L is said to be REL if there exists a TM which will accept (and halt) for all the input strings which are in L.

→ But may or may not halt for all input strings which are not in L.



$L_{Rec} \subseteq L_{REL}$

A Recursive language (RL) is also Recursively enumerable language (REL), but a REL is not necessarily recursive.

Deciding vs Recognizing :-

* Turing Machine M "recognizes" language L if
  ① the string L put M into the Accept state
  ② the string Not in L
    ⓐ either put M into Reject state
    ⓑ OR cause M to loop

* Turing Machine M "decides" language L if -
  ① the string L put M into accept state
  ② the string Not in L put M into the reject state.

★ Recognizer :-

A recognizer of a language is a machine that recognizes that language.

→ It halt in the Accept state on the string that are in the language.
→ It may or may not halt on string that are not in the language.

★ Decider :-

A decider of a language is a machine that decides that language.

→ It halt in the Accept state on the string that are in the language.
→ It also halts if the string is not in the language.

★ Turing Decidable language – A language is Turing-decidable (or decidable) if some TM decides it.
Also known as (Aka) – "Recursive language"

★ Turing Recognizable language – A language is Turing-recognizable if some Turing machine recognizes it.
Aka – "Recursively Enumerable language"

✳ Closure properties of Recursive language –

① Union – If L_1 and L_2 are two recursive languages, there union $L_1 \cup L_2$ will also be recursive because if TM halts for L_1 and L_2 it will also halts for $L_1 \cup L_2$

② Concatenation – If L_1 & L_2 are two RL, their concatenation $L_1 \cdot L_2$ will also be Rec.
For ex –

$L_1 = \{a^n b^n c^n \mid n >= 0\}$
$L_2 = \{d^m e^m f^m \mid m >= 0\}$
$L_3 = L_1 \cdot L_2$
$= \{a^n b^n c^n d^m e^m f^m \mid m >= 0 \text{ and } n >= 0\}$
is also recursive.

③ Kleene Closure – If L_1 is recursive, its Kleene closure L_1^* will also be recursive.
For ex – $L_1 = \{a^n b^n c^n \mid n >= 0\}$
$L_1^* = \{a^n b^n c^n \mid n >= 0\}^*$ is also Recursi

④ Intersection- If L_1 and L_2 are two recursive languages, their intersection $L_1 \cap L_2$ will also be recursive.

For ex-

$L_1 = \{a^n b^n c^n d^m \mid n>=0 \& m>=0\}$

$L_2 = \{a^n b^n c^n d^n \mid n>=0 \& m>=0\}$

$L_3 = L_1 \cap L_2$

$= \{a^n b^n c^n d^n \mid n>=0\}$ will be recursive

⑤ Complement - If L_1 is recursive language, the complement of L (which is $\Sigma^* - L_1$), will also be recursive.

Closure properties of REL :-

★ REL is closed under -
1. Union
2. Concatenation
3. Kleene Closure
4. Intersection

★ REL is not closed under -
1. set difference and
2. complement

Context-Sensitive Grammar :-

A grammar $G = (V_n, \Sigma, P, S)$ is said to be context sensitive, if all production rules in P have the form - $\alpha \rightarrow \beta$

where -

$$\alpha \in (\Sigma \cup V_n)^* \; V_n \; (\Sigma \cup V_n)^*$$

$$\beta \in (\Sigma \cup V_n)^+$$

and

$$|\alpha| <= |\beta|$$

OR -

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

where -

$$\alpha, \beta \in (\Sigma \cup V_n)^*$$

$$A \in V_n$$

$$\gamma \in (\Sigma \cup V_n)^+$$

Note- The production $S \rightarrow \epsilon$ is allowed if S is the start symbol and S does not appear on the right side of any production.

Context-sensitive grammars (CSG) are more powerful than context free grammar (CFG) because there are some languages that can be described by CSG but not by CFG.

✱ Context Sensitive Language –

The language generated by the CSG is called context sensitive language (CSL).

Ex –

① $S \rightarrow aTb \mid ab$
$aT \rightarrow aaTb \mid ac$

$L(G) = \{ab\} \cup \{a^n c b^n \mid n > 0\}$

② Consider the following CSG –

$S \rightarrow abc \mid aAbc$
$Ab \rightarrow bA$
$Ac \rightarrow Bbcc$
$bB \rightarrow Bb$
$aB \rightarrow aa \mid aaA$

The language generated by this grammar is

$S \rightarrow aAbc$
$S \rightarrow abAc$
$S \rightarrow abBbcc$
$S \rightarrow aBbbcc$
$S \rightarrow aaAbbcc$
$S \rightarrow aabAbcc$
$S \rightarrow aabbAcc$

$S \rightarrow aabbBbccc$
$S \rightarrow aabBbbccc$
$S \rightarrow aaBbbbccc$
$S \rightarrow aaabbbccc$

The language generated by this grammar is-

$L(G) = \{a^n b^n c^n \mid n >= 1\}$

* Closure Properties of CSL-

CSL are closed under -
→ Union
→ Intersection
→ Complement
→ Concatenation
→ kleene closure
→ Reversal