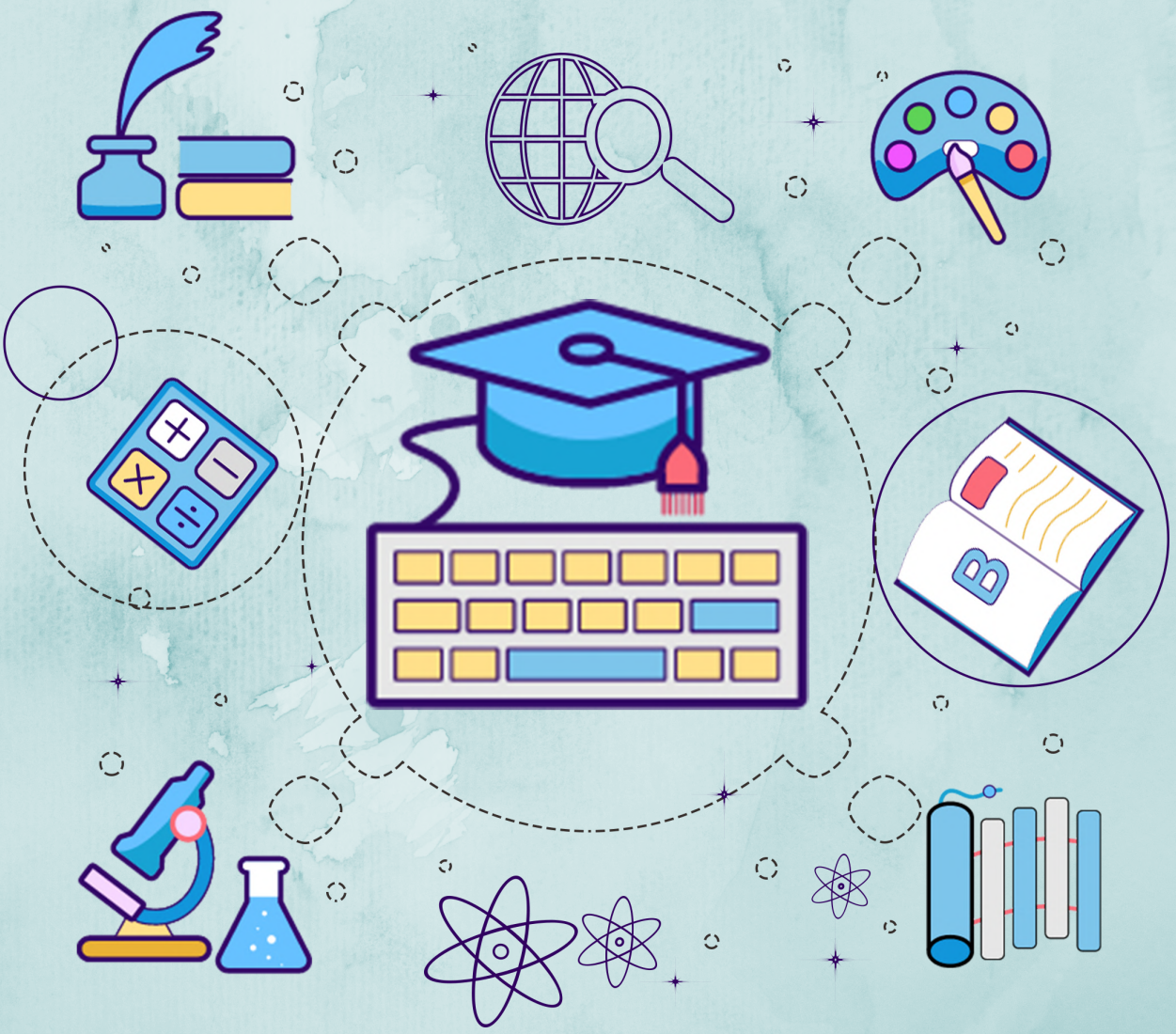
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KeralaNotes



SYLLABUS | STUDY MATERIALS | TEXTBOOK

PDF | SOLVED QUESTION PAPERS

www.keralanotes.com

KeralaNotes

KTU STUDY MATERIALS

FORMAL LANGUAGES AND AUTOMATA THEORY

CST 301

Module 5

Related Link :

- KTU S5 STUDY MATERIALS
- KTU S5 NOTES
- KTU S5 SYLLABUS
- KTU S5 TEXTBOOK PDF
- KTU S5 PREVIOUS YEAR SOLVED QUESTION PAPER

FORMAL LANGUAGES AND AUTOMATA THEORY

Module 5

Context-Sensitive Languages, Turing Machines:

Context-Sensitive Languages - Context Sensitive Grammar (CSG), Linear Bounded Automata.

Turing Machines - Standard Turing Machine, Robustness of Turing Machine, Universal Turing Machine, Halting Problem, Recursive and Recursively Enumerable Languages. Chomsky classification of formal languages.

Prepared by:

Karishma PK

Assistant professor

Computer Science Department EKCTC

MODULE - V

CONTEXT SENSITIVE GRAMMAR (CSG)

- A context sensitive Grammar is a collection of 4 tuples.

  $G = (V, T, P, S)$

  V → Set of variables / non Terminals

  T → Set of Terminals / input alphabets

  S → Starting symbol

  P → Production rule. $(\alpha \rightarrow \beta)$ where $\alpha, \beta \in (V \cup T)^+$

- A grammar $G = (V, T, P, S)$ is said to be context sensitive, if all production are of the form $\alpha \rightarrow \beta$ where $|\alpha| \leq |\beta|$ and β is not suppose to be empty.

- In which P are in the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$

- In which α_1 & α_2 are may empty, but β is non empty.

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

- The languages generated by these grammars are recognized by a Linear Bounded Automata

Context-sensitive Language (CSL)

The language that can be defined by context-sensitive Grammar is called CSL.

Properties of CSL

Union

Let L_1 & L_2 are two CSL. Then $L_1 \cup L_2$ is also context sensitive

Eg:- $L_1 = \{a^n b^n, n>0\}$

$L_2 = \{c^k d^k, k>0\}$

union of L_1 & L_2

$\Rightarrow L_1 \cup L_2$

$\Rightarrow \{a^n b^n\} \cup \{c^k d^k\}$ is also union

Concatenation

If L_1 & L_2 are CSL. Then $L_1 \cdot L_2$ is also context sensitive

Eg:- $L_1 = \{a^n b^n, n>0\}$

$L_2 = \{c^k d^k, k>0\}$

$L_1 \cdot L_2 = \{a^n b^n \cdot c^k d^k\}$

Intersection

If L_1 and L_2 are two CSL, then $L_1 \cap L_2$ are also context sensitive.

complement

If L_1 is a context sensitive Language (CSL), the complement of L_1 is also context-sensitive.

- Eg for context sensitive Grammar is

$AB \rightarrow AbBc$

$A \rightarrow bcA$

$B \rightarrow b$

7) Consider the following CFG what is the language generated by this grammar:

$S \to abc \mid aAbc$

$Ab \to bA$

$Ac \to Bbcc$

$bB \to Bb$

$aB \to aa \mid aaA$

Soln:-

$S \to aAbc$

$\to abAc$

$\to abBbcc$

$\to aBbbcc$

$\to aaAbbcc$

$\to aabAbcc$

$\to aabbAcc$

$\to aabbBbccc$

$\to aabBbbccc$

$\to aaBbbbccc$

$\to aaabbbccc$

∴ The language generated by this grammar is $\{a^n b^n c^n \mid n \geq 1\}$

Q) Consider the following CSG. What is the language generated by this grammar?

$S \rightarrow aTbb|bb$

$aT \rightarrow aaTb|ac$

~~soln:-~~ ~~$S \rightarrow aTb$~~

~~$\Rightarrow aaTbb$~~

soln:-

$S \rightarrow \underline{aTb}$

$\rightarrow aa\underline{T}bb$

$\rightarrow aaa\underline{T}bbb$

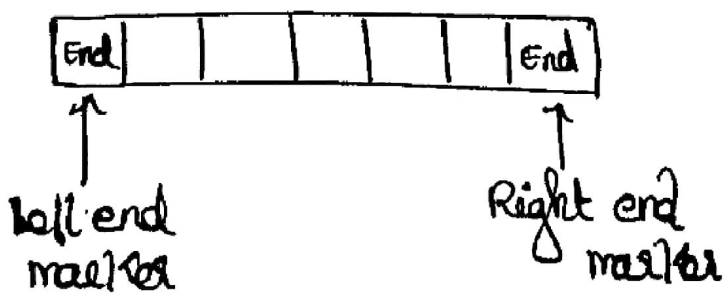
$\rightarrow aaacbbb$

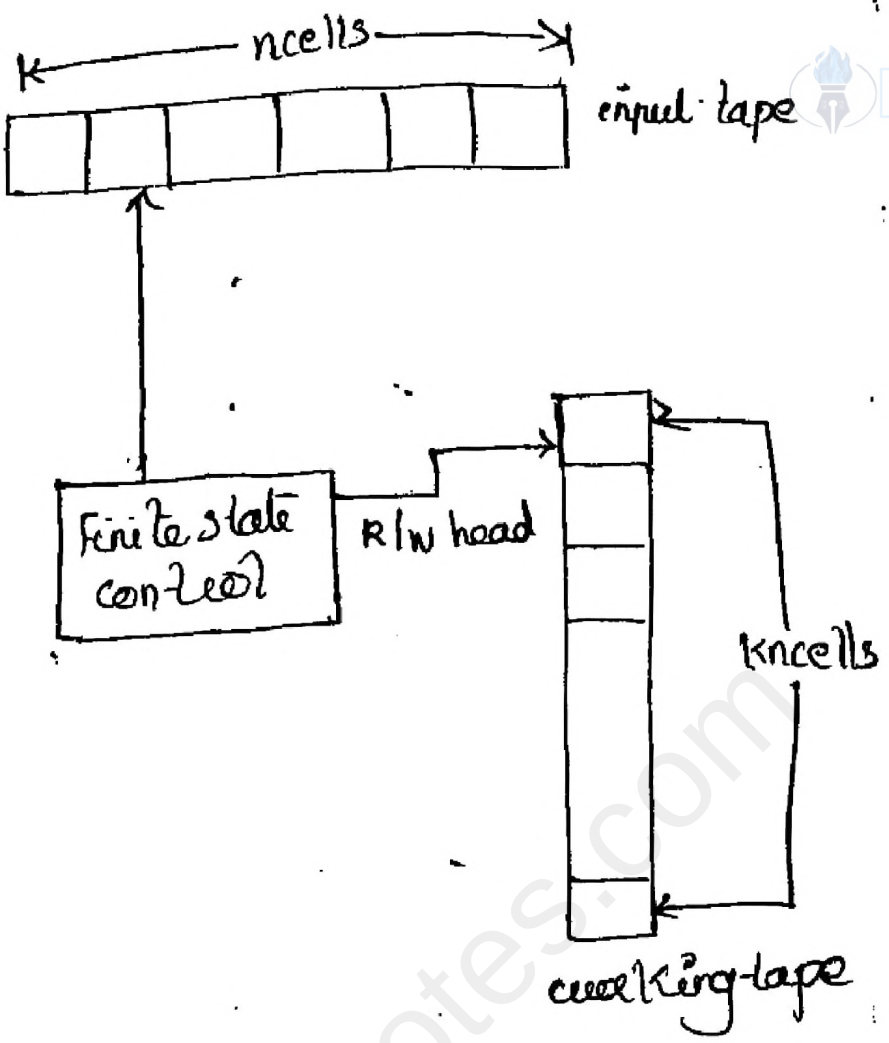
∴ The language generated by this grammar is

$\{a^n c b^n \mid n \geq 0\}$

Linear Bounded Automata (LBA)

- Linear Bounded Automata (LBA) is more powerful than PushDown Automata (PDA)
- Linear Bounded Automata consist of a tape, but the storage space of tape is restricted only to the length of the input string.
- The tape is divided into 'n' no: of cells.
- The computation is restricted to the constant bounded area.
- The input alphabets contain 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 deterministic linear bounded automaton is always context-sensitive and the linear bounded automaton with empty language is undecidable.





Formal definition

A Linear Bounded Automata can be defined as a tuple.

$$A = (Q, T, \Sigma, q_0, \overset{L}{\vdash}, \overset{R}{\dashv}, \delta, F)$$, where

Q => Finite set of states

Γ / T => Tape alphabet

Σ => set of input alphabet.

q_0 → Initial state

⊢/[| M_L → Left end marker

⊣/] | M_R → Right end marker (where $M_L \neq M_R$)

δ → Transition function which maps each pair (state, tape symbol) to (state, tape symbol, constant c)

where c can be 0, +1, -1, $(Q - F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

F → Set of final states.

(5) Design a LBA which accept $a^n b^n$, $n > 0$!

Soln:-

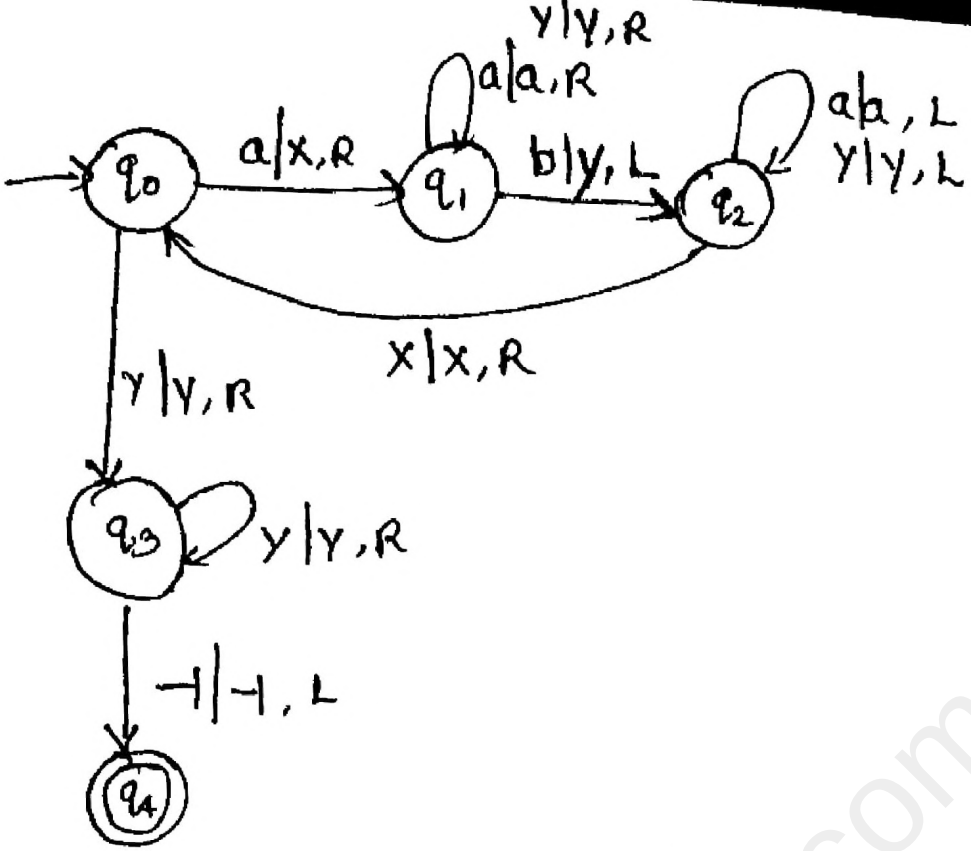
Let take a n = 2

$\Rightarrow a^2 b^2$

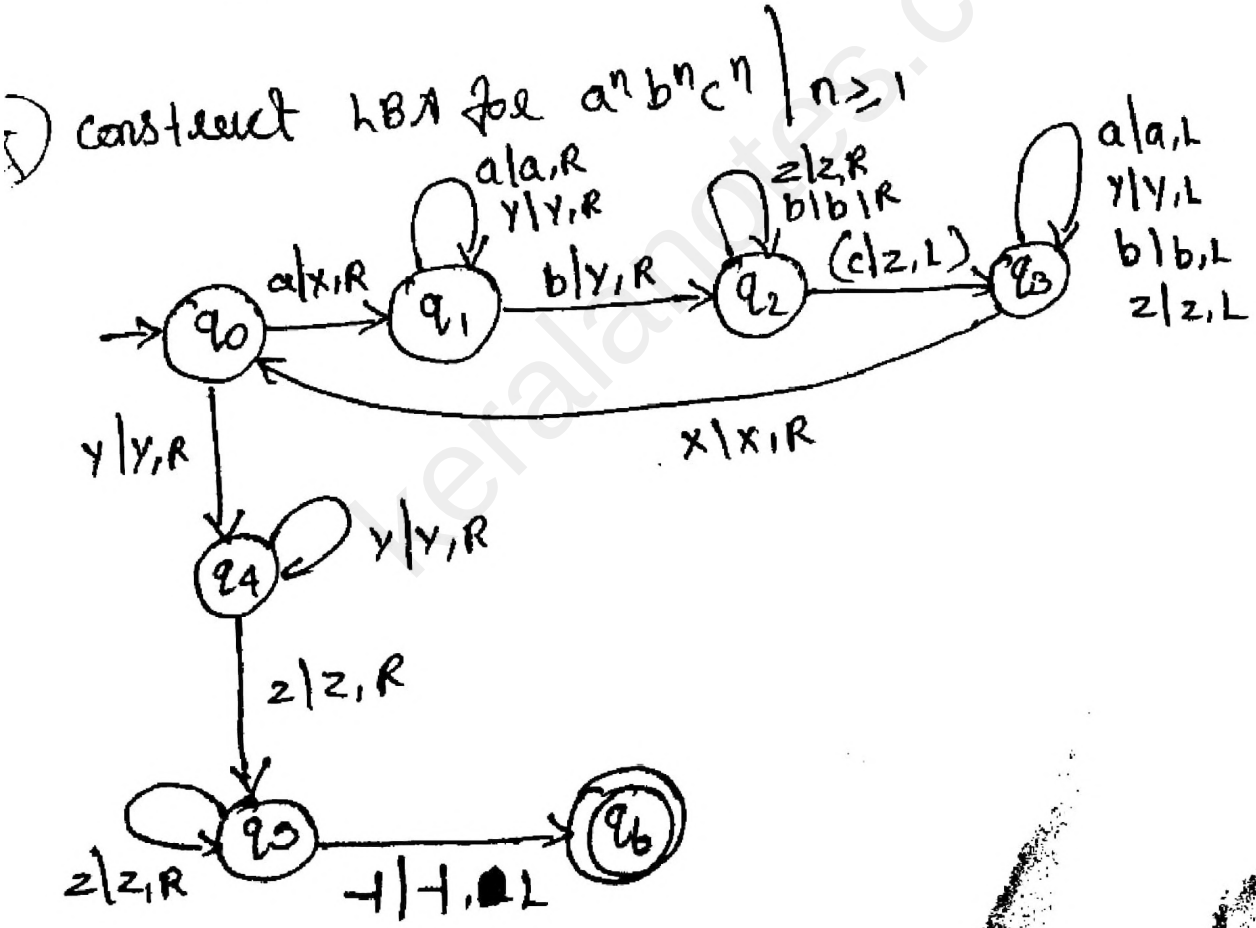
⊢	a	a	b	b	⊣

a → X

b → Y

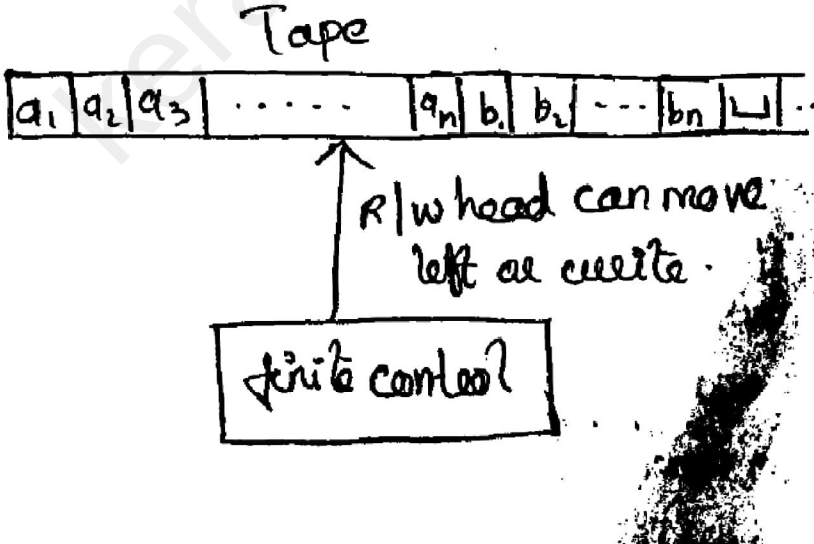


5) Construct LBA for $a^n b^n c^n \mid n \geq 1$



TURING MACHINE (TM)

- Turing machine was invented in 1936 by Alan Turing.
- It is an accepting device which accepts Recursive Enumerable Language generated by type 0 grammar
- A Turing Machine consists of a tape of infinite length on which read and write operation can be performed
- The tape consists of infinite cells on which each cell either contains input symbol or a special symbol called blank.
- It also consists of a head pointer which points to cell currently being read and it can move in both directions.



Operations on the Tape

- Read / scan the symbol which is pointed by the tape head
- update / write a symbol which is pointed by the tape head.
- Move the tape head one step left
- Move the tape head one step right.

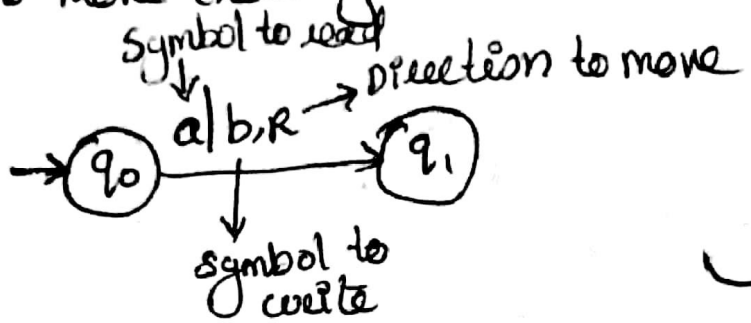
Features of the Turing Machine

- It has an external memory which remember arbitrary long sequence of input. (unlimited storage memory (tape))
- It has unlimited memory capability.
- The model has a facility by which the input at left or right on the tape can be read easily.
- 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 between input and output has been removed. There a common set of alphabets can be used for the Turing machine.

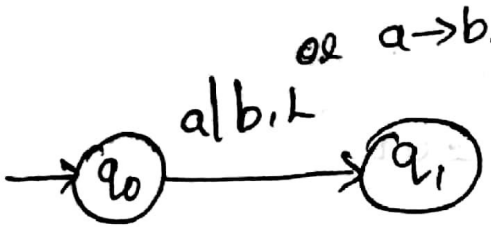
Operation Rule-1

At each step of computation.

- Read the current symbol
- write (update) the same cell
- Move exactly one cell either LEFT or RIGHT



Transition diagram



Transition diagram

operation Rule-2

- Turing Machine has a control portion, which controls the Turing Machine and is similar to FSM or PDA. It is deterministic also.
- TM has an initial state
- TM has 2 final states.
  - Accept states
  - Reject states

When the computation Halt (Stop)

- HALT & Accept
- HALT & Reject
- Loop (Keep on computation and fail to stop)

Formal definition of TM

A TM can be defined as a collection of 7 tuples.

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$$

$Q \Rightarrow$ Finite set of states

$\Sigma \Rightarrow$ Finite set of input symbols

$\Gamma / T \Rightarrow$ Tape symbol $(\Sigma \cup \sqcup)$

$q_0 \Rightarrow$ Initial state

$F \Rightarrow$ Final state.

$B / \sqcup \Rightarrow$ a blank symbol used as a end marker for input. $(\sqcup \notin \Sigma)$

$\delta \Rightarrow$ a transition function.

$$Q \times T \rightarrow Q \times T \times \{L, R\}$$

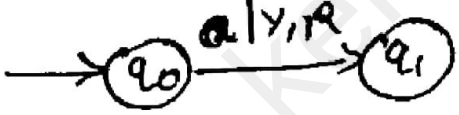
Eg:- for δ

$\delta(q_0, a) \rightarrow (q_1, Y, R)$

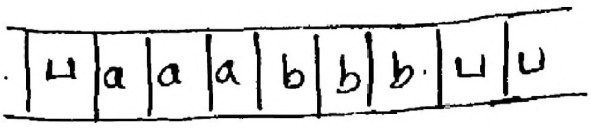
q_0 : state, a : i/P, q_1 : state, Y : write symbol (replaced symbol), R : move to right

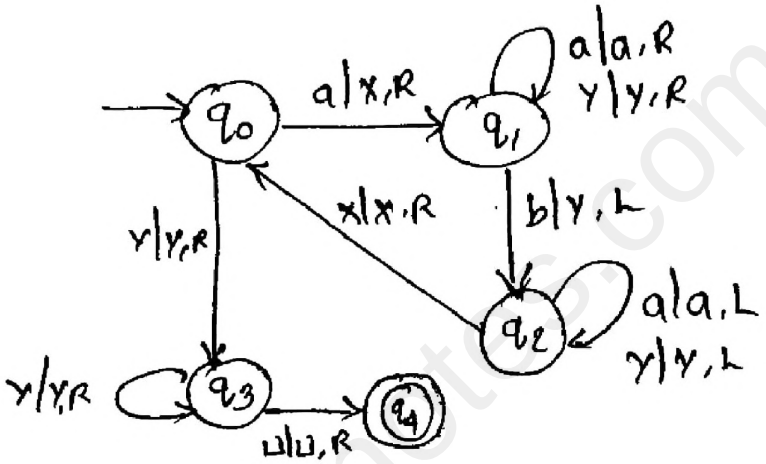
- Any computations that can be done on existing digital computer, can also be done by TM.
- If we are having an algorithm of a pblm, then that can't also be solved by TM.
- The language that acceptable by the TM is called Recursively Enumerable Language.

Transition diagram



6) Design a TM that accepts the language $L = \{ 0^i 1^i \mid i \geq 1 \}$



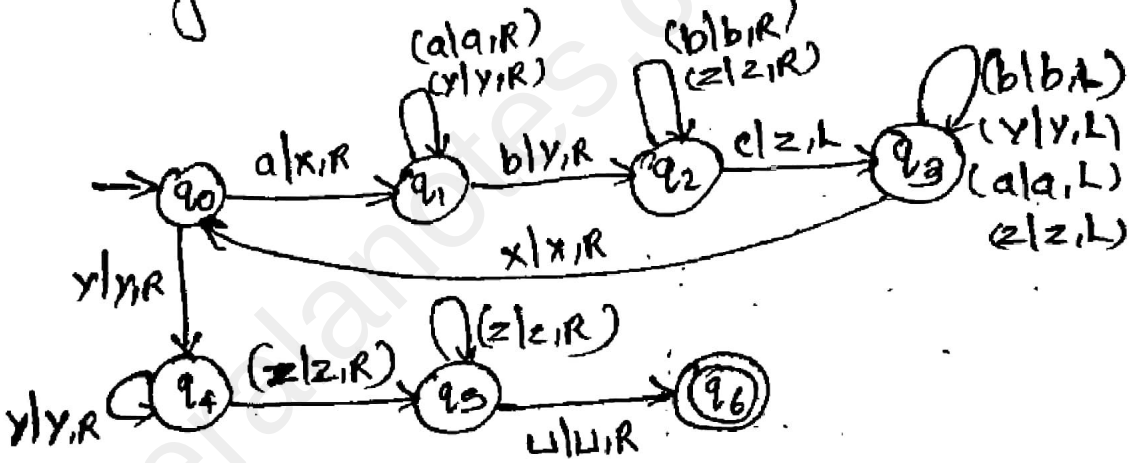


$\delta(q_0, a) \rightarrow (q_1, X, R)$
$\delta(q_1, a) \rightarrow (q_1, a, R)$
$\delta(q_1, Y) \rightarrow (q_1, Y, R)$
$\delta(q_1, b) \rightarrow (q_2, Y, L)$
$\delta(q_2, a) \rightarrow (q_2, a, L)$
$\delta(q_2, Y) \rightarrow (q_2, Y, L)$
$\delta(q_2, X) \rightarrow (q_0, X, R)$
$\delta(q_0, Y) \rightarrow (q_3, Y, R)$
$\delta(q_3, Y) \rightarrow (q_3, Y, R)$
$\delta(q_3, \sqcup) \rightarrow (q_4, \sqcup, R)$

Transition table

δ	a	b	x	y	⊔
q_0	(q_1, x, R)	-	-	(q_3, y, R)	
q_1	(q_1, a, R)	(q_2, y, L)	-	(q_1, y, R)	
q_2	(q_2, a, L)	-	(q_0, x, R)	(q_2, y, L)	
q_3	-	-		(q_3, y, R)	$(q_4, ⊔, R)$

1) design a TM $L = \{a^n b^n c^n : n \geq 1\}$? aaabbbccc



TM for $a^n b^n$

$\delta(q_0, a) = (q_1, x, R)$

$\delta(q_0, y) = (q_4, y, R)$

$\delta(q_1, a) = (q_1, a, R)$

$\delta(q_1, b) = (q_2, y, R)$

$\delta(q_1, Y) = (q_1, Y, R)$

$\delta(q_2, b) = (q_2, b, R)$

$\delta(q_2, c) = (q_3, z, L)$

$\delta(q_2, z) = (q_2, z, R)$

$\delta(q_3, a) = (q_3, a, L)$

$\delta(q_3, b) = (q_3, b, L)$

$\delta(q_3, Y) = (q_3, Y, L)$

$\delta(q_3, z) = (q_3, z, L)$

$\delta(q_3, X) = (q_0, X, R)$

$\delta(q_4, Y) = (q_4, Y, R)$

$\delta(q_4, z) = (q_3, z, R)$

$\delta(q_5, z) = (q_5, z, R)$

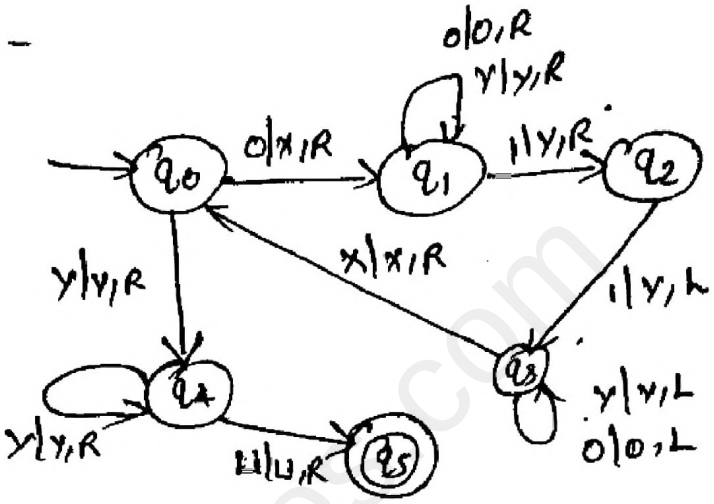
$\delta(q_5, \sqcup) = (q_6, \sqcup, R)$

Transition Table:

δ	a	b	c	X	Y	Z	$\sqcup$
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_5, z, R)	
q_5						(q_5, z, R)	$(q_6, \sqcup, R)$

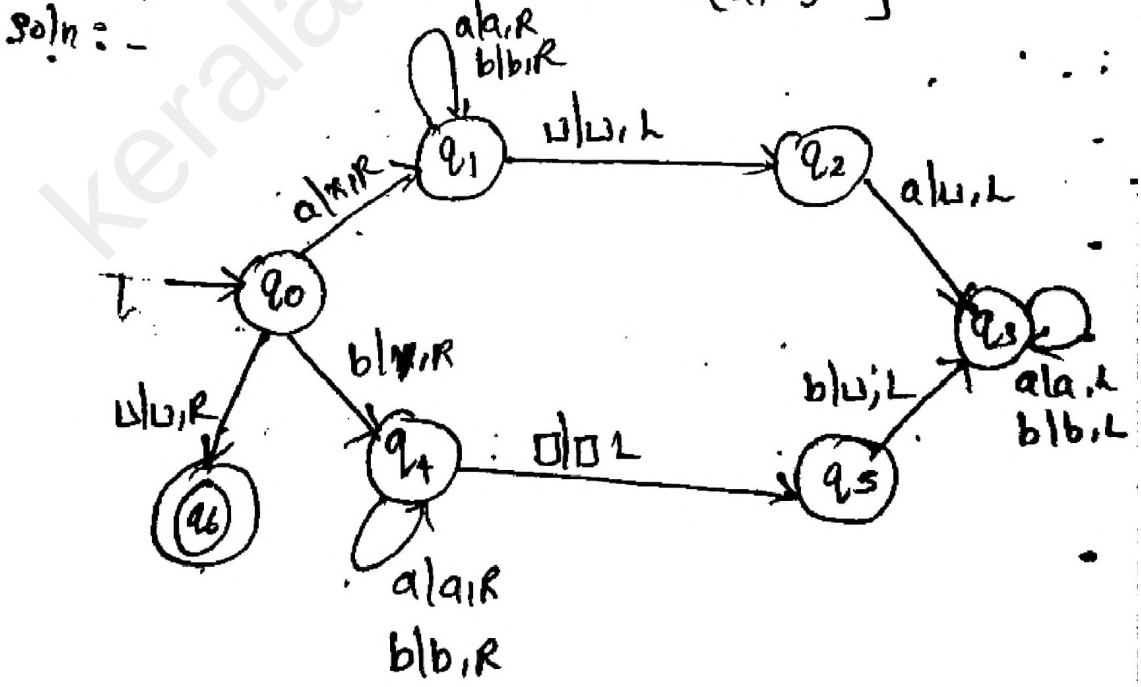
4) Design a TM which accepts $L = \{0^n 1^{2n} ; n \geq 1\}$

Soln :-



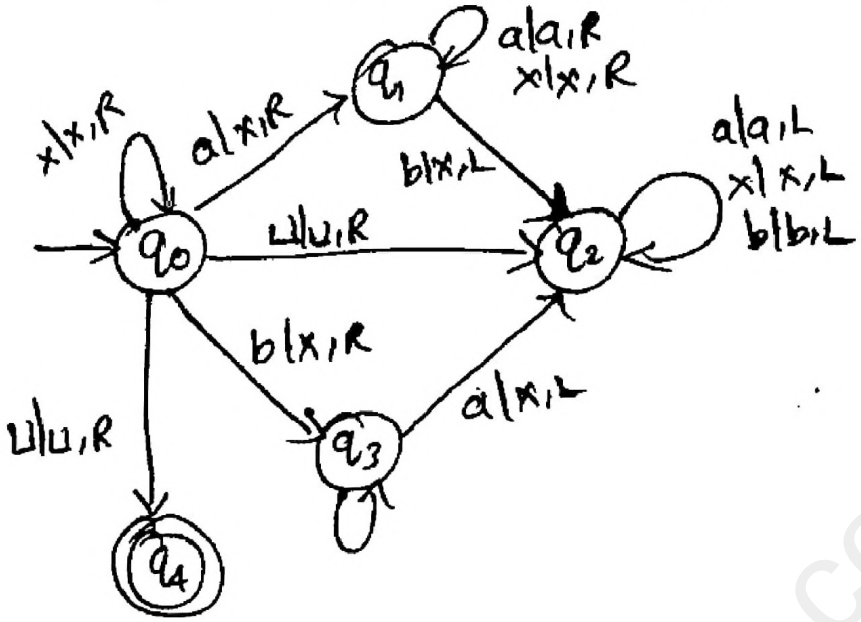
5) Design a TM which accepts $L = \{ww^R \mid w = \{a,b\}^*\}$

Soln :-



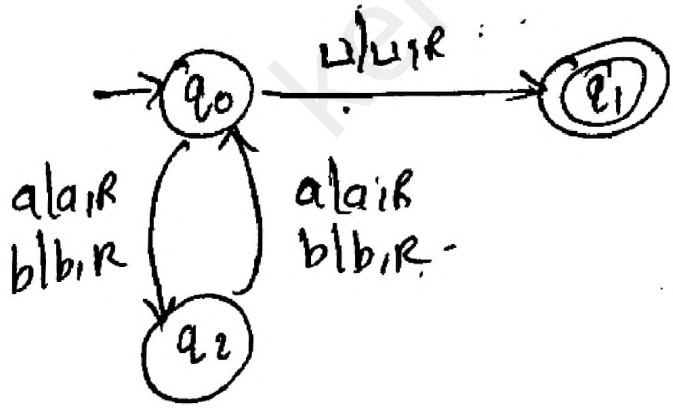
9) Design a TM which accepts :

$L = \{ w \mid \#a(w) = \#b(w) \}$

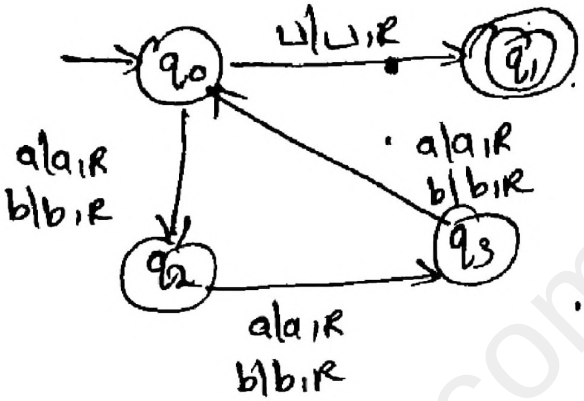


10) Design a TM which accepts

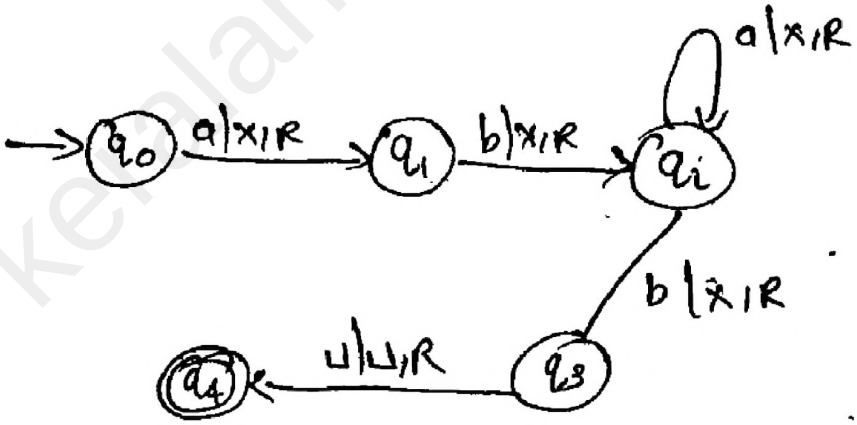
$L = \{ w \mid |w| = \text{even} \}$, $w \in \{a,b\}^*$.



(8) Design a TM which accepts the language $L = \{w$ such that $|w| =$ multiple of $3\}$, $w \in \{a,b\}^*$?



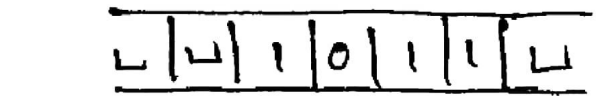
(9) Design a TM which accepts the language $L = \{aba^*b\}$.

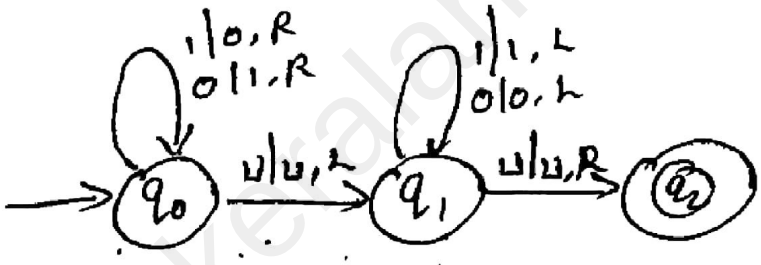


Turing Machine as a Transducer

Primary purpose of a computer is to transform input to output, it acts as transducer. The input of the computation will be all non blank symbols on the tape at the initial stage. At the conclusion of the computation, the output will be whatever is then on the tape.

) Design a TM to compute 1's complement.





4) Design a TM that computes $x+y$.

note:

use use unary notation (language have a single symbol in which any +ve integer x is represented by $w(x) = (1)^+$ ie,

$|w(x)| = x$

Let $x, y \in N$, we want to design a TM, T that performs the $f^n, f(x,y) = x+y$

The numbers x and y are encoded using the unary encoding scheme in such a way that x is encoded as 1^x & y is encoded as 1^y & $f(x,y)$ is encoded as $1^x 0 1^y$

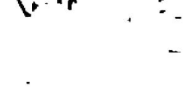
Sol:

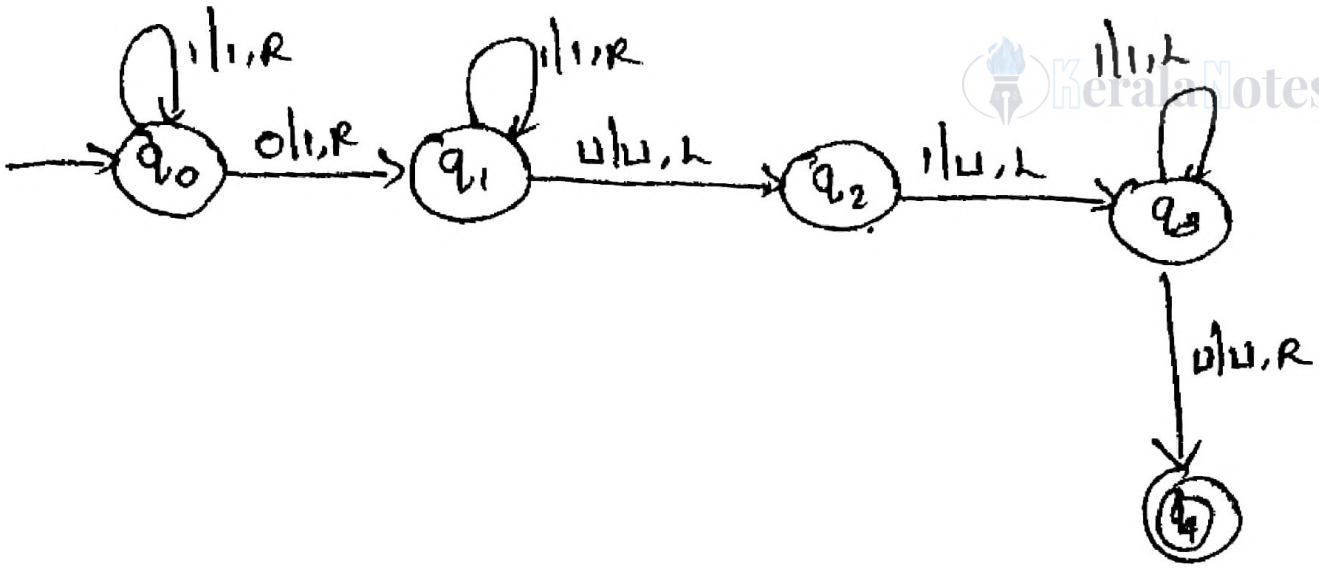
Assume $x=5$, & $y=2$

$\Rightarrow x+y$ separated with zero

$\Rightarrow 111110111$

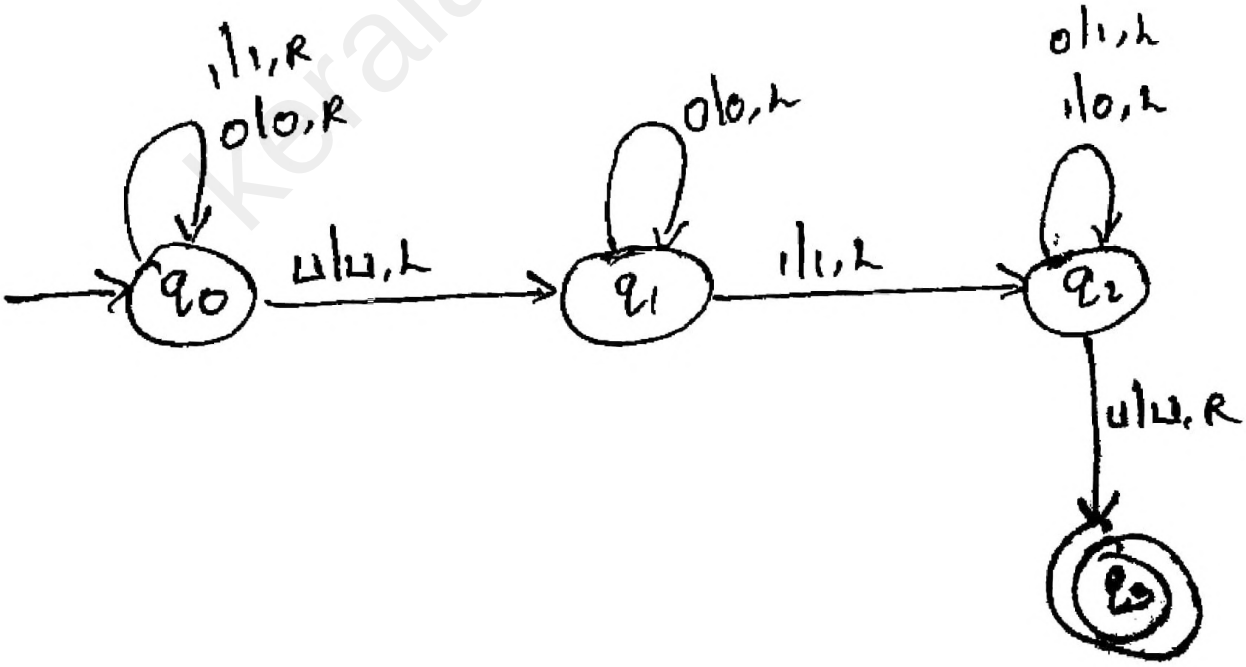
$\overline{5}\ \overline{2}$





(5) Design a TM to compute 2's complement of a number.

```
 1 1 0 1 0 1 0
 0 0 1 0 1 0 +
       1
-------------
 0 0 1 0 1 1
```



Nondeterministic Turing Machine

A Nondeterministic Turing Machine has 7 tuples.

$M = (Q, \Sigma, T, \delta, q_0, \sqcup, F)$

where

$Q \rightarrow$ Finite set of states

$\Sigma \rightarrow$ Finite set of input alphabets

$T \rightarrow$ Tape alphabet $[\Sigma \cup T - \{\sqcup\}]$

$\delta \rightarrow Q \times T \rightarrow 2^{Q \times T \times \{L, R\}}$

Transition function

$B / \sqcup / \square \rightarrow$ Blank symbol. $(B \in T)$

$q_0 \rightarrow$ Initial state

$F \rightarrow$ Set of final states.

Recursively Enumerable

A language L is said to be recursively enumerable if there exists a TM that accepts it.

Multitape Turing Machine

- Multitape 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.

Formal definition

A multi-tape TM can be formally described as a 6-tuple.

$M = (Q, T, B, \delta, q_0, F)$, where

$Q \rightarrow$ Finite set of states

$T \rightarrow$ Tape alphabet

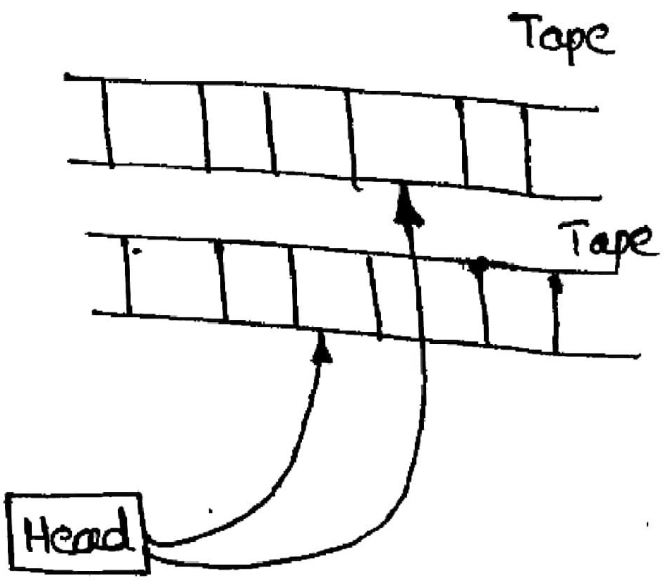
$B \rightarrow$ Blank symbol

$\delta \rightarrow$ is a relation on states & symbols.

$Q \times T^n \rightarrow Q \times T^n \times \{L, R\}^n$, i is the n^{th} tape.

q_0 - initial state

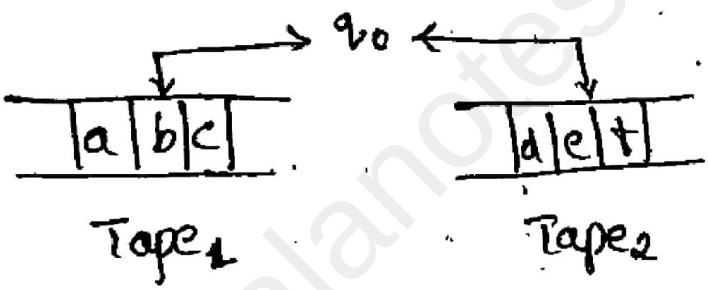
$F \rightarrow$ set of final states.

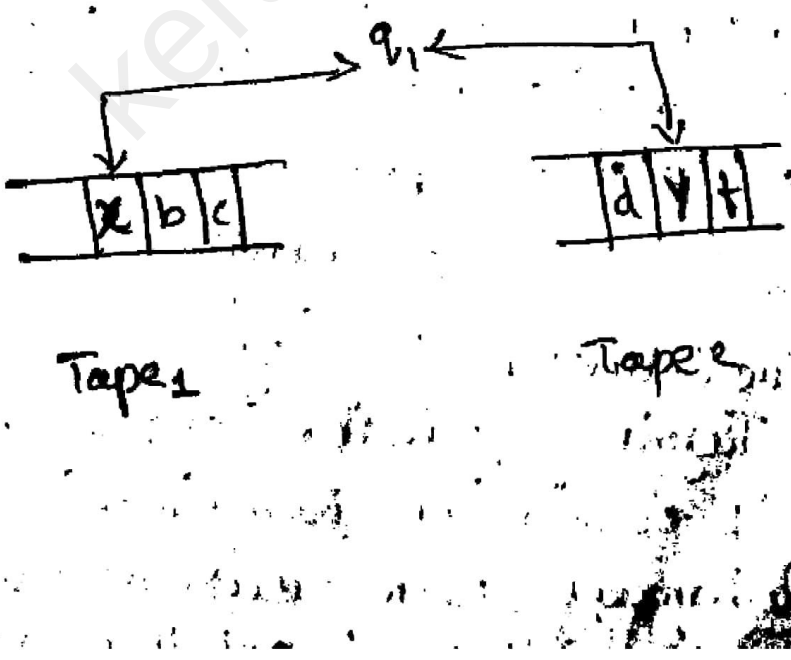


Eg:-

$n = 2$

$\delta(q_0, a, e) = (q_1, X, Y, L, R)$





Recursive & Recursive Enumerable languages

The Turing Machine may
- Halt and accept the input
- Halt and Reject the input
- never Halt

Recursive language

- Recursive Languages are also known as Turing decidable languages.
- A language 'L' is said to be recursive, if there exist a Turing Machine which will accept all the strings in 'L' and reject all the strings not in 'L'
- The Turing Machine will halt every time and give the answer (accepted or rejected) for each and every string input.

Recursive Enumerable language

- Recursive Enumerable languages are also known as Turing recognizable languages.
- A language 'L' is said to be Recursively Enumerable language, if there exist a Turing Machine which will accept and therefore

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'.
- Recursive language is a subset of Recursive Enumerable language.

Properties of Recursive language

Union

If L_1 and L_2 are two recursive languages, their union $L_1 \cup L_2$ will also be recursive because if TM halts for L_1 and halts for L_2, it will also halt for $L_1 \cup L_2$

Concatenation

If L_1 and L_2 are two recursive languages, their concatenation $L_1 \cdot L_2$ will also be recursive.

Kleene closure

If L_1 is recursive, its Kleene closure L_1^* will also be recursive.

Intersection

If L_1 and L_2 are 2 recursive languages, their intersection $L_1 \cap L_2$ will also be recursive.

complement

Complement of recursive language L_1 which is L_1' will also be recursive.

Properties of Recursive Enumerable language

Union

If L_1 & L_2 are 2 recursive enumerable languages, their union $L_1 \cup L_2$ will also be recursive enumerable.

Concatenation

If L_1 and L_2 are 2 recursive enumerable languages, their concatenation $L_1 \cdot L_2$ will also be recursive enumerable.

Kleen closure

If L_1 is recursive enumerable, its Kleene closure L_1^* will also be recursive enumerable.

intersection

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

complement

Recursive Enumerable languages are not closed under complement, which means complement of Recursive Enumerable language need not be recursive enumerable.

Decidability & Undecidability

Decidable language : yes/no

- A language L is decidable if it is a recursive language.
- All decidable languages are recursive languages and vice-versa

Partially decidable languages

- A language L is partially decidable, if L is a recursively enumerable language.

undecidable language

- A language is undecidable if it is not decidable.
- An undecidable language may sometimes be partially decidable but not decidable.
- If a language is not even partially decidable, then there exists no Turing Machine for that language.

Recursive language	TM will always Halt
Recursive Enumerable Language.	TM will halt some times and may not halt some times.
Partially Decidable Language	Recursively Enumerable language.
undecidable	no Turing Machine for that language.

Universal Turing Machine

- A TM is said to be universal Turing Machine if it can accept:
  - The input data
  - An algorithm (description) for computing.
- This is precisely what a general purpose digital computer does. A digital computer accepts a program written in high level languages.

- There a general purpose Turing Machine will be called a universal Turing Machine if it is powerful enough to simulate the behaviour of any digital computer, including any TuringMachine itself.
- A universal Turing Machine is just a Turing Machine whose programming simulates other Turing Machines.
- That is, the input to the universal Turing Machine is a description of a TM ; T and an input for T, and the universal Turing Machine simulates T on that input.
- It's universal in the sense that, for any problem that can be solved by TM, you could either use a TM that directly solves that problem, or you could use a universal Turing Machine and give it the description of a TM that directly solves the problem.

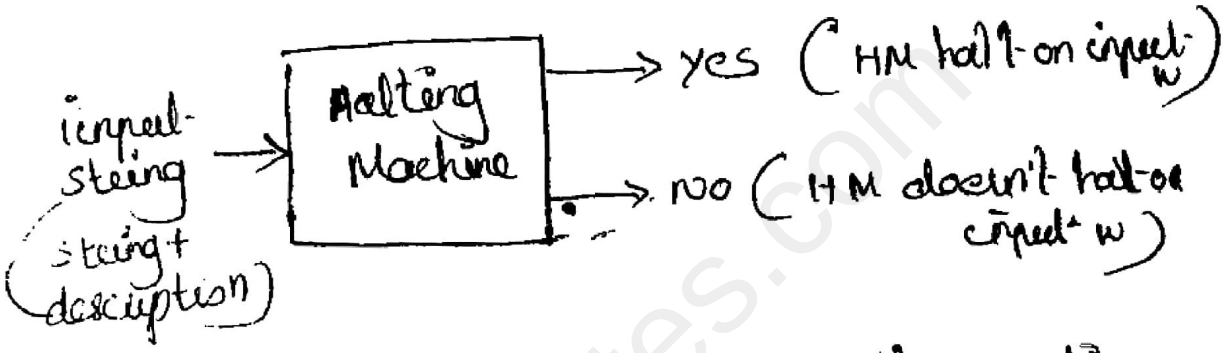
Halting Problem of TM

- 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.
- Alan Turing proved in 1936 that a general algorithm to solve the halting problem for all possible program-input pairs cannot exist.
- Halting problem is undecidable.
- Given the description of a Turing Machine M and an input w, does M, when started in the initial configuration q0w, perform a computation that eventually halts?
- Input - A TM and an input string w
- Problem — does the TM finish computing of the string w in a finite number of steps? The answer must be either yes or no.

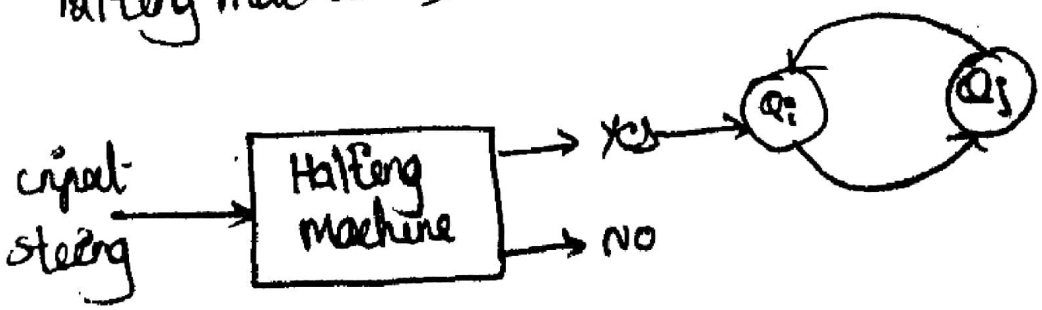
Proof

At first, we will assume that such a TM exists to solve this problem and then we will show it is contradicting itself.

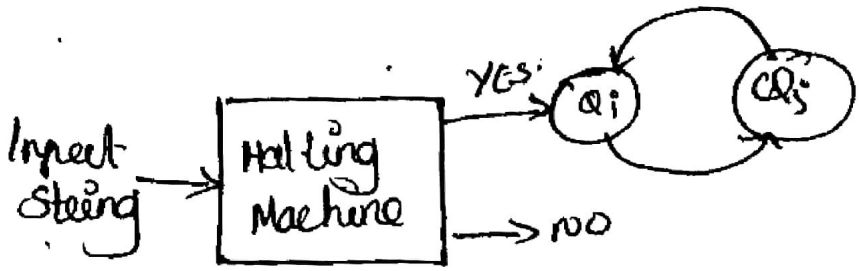
- we will call this Turing Machine as a Halting Machine that produces a 'yes' or 'no' in a finite amount of time.
- If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'.
- The following is the block diagram of a Halting machine.



- 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.

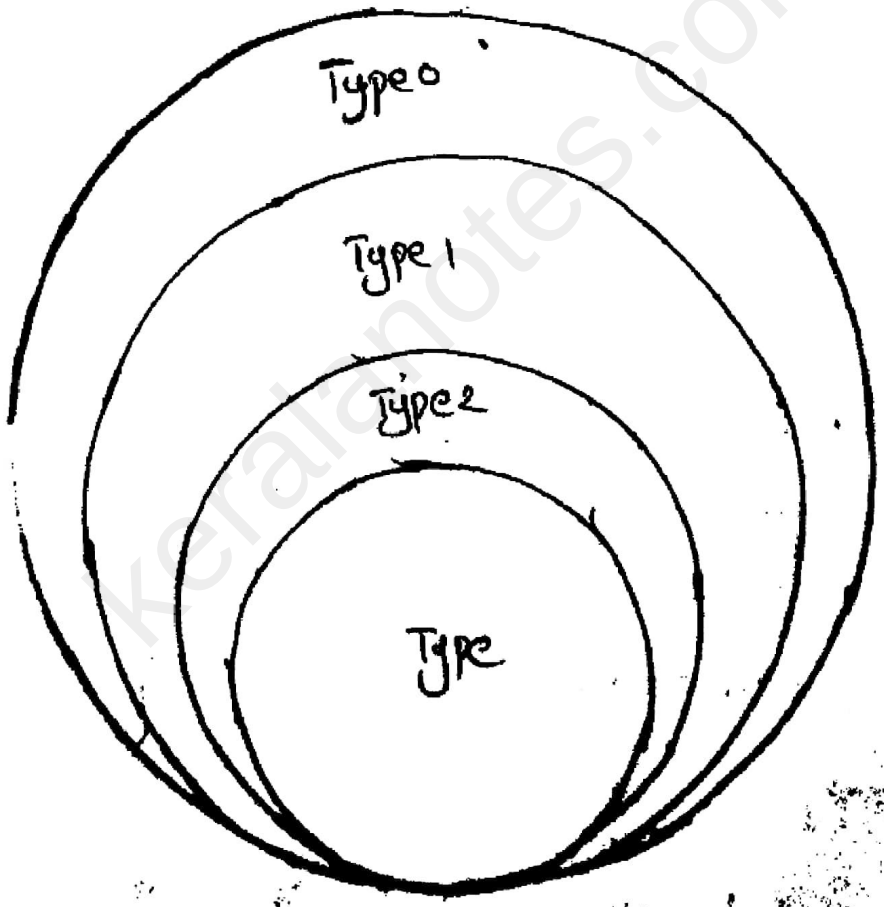


- Here we have got a contradiction. Hence the halting problem is undecidable.

Chomsky classification of languages

- Noam chomsky classified grammars and the generated languages into different types known as chomsky Hierarchy.
- 4 types.

- Type 0 languages – Recursively Enumerable languages.
- Type 1 languages – context sensitive languages
- Type 2 languages – Context free languages.
- Type 3 languages – Regular languages.



Type 0 Languages

- Also known as Recursively Enumerable languages.
- These languages are generated by type 0 grammars or unrestricted grammars
- Their production rule is of the form

$$\alpha \longrightarrow \beta$$

- where α & β are strings of terminals or nonterminals
- These languages are recognised using TM

Type 1 Language

- Also known as context Sensitive languages.
- These languages are generated by type 1 grammars or context-sensitive grammars.
- Their production rule is of the form.

$$\alpha \longrightarrow \beta \quad , \quad |\beta| \geq |\alpha|$$

- where α & β are strings of terminals or nonterminals.

- This language includes any strings of terminals and non terminals, where length of strings on the right must be greater than or equal to length of strings on the left side
- These languages are recognised using Linear Bounded Automata.

Type 2 languages

- Also known as context free languages
- These languages are generated by type 2 grammar or CFG
- Production in the form of

$$A \longrightarrow \alpha$$

- where α is arbitrary strings of grammar symbols & left side is a single non-terminal.
- Languages are recognised by using PDA.

Type 3 languages

- Also known as Regular languages.
- Generated by type 3 grammar or RG & described using RE

- These are the most specific languages and is the smallest subset of all the languages.
- Production rules in the form of

$A \rightarrow a$

$A \rightarrow a\beta$

- where β is strings of terminals & non terminals & $A \rightarrow$ non terminal, $a \rightarrow$ terminal.
- These languages are recognised by every FSA.