Turing Machines

A 'Turing Machine' consists of a finite control, a tape, and a head that can be used for reading or writing on that tape.

Def^n: A Turing machine is a quintuple $(K, \Sigma, \delta, s, H)$ where

K is a finite set of states;

Σ is an alphabet, containing the blank symbol $\sqcup$ and the left end symbol $\triangleright$, but not containing the symbols $\leftarrow$ and $\rightarrow$;

$s \in K$ is the initial state.

$H \subseteq K$ is the set of halting states.

δ, the transition function, is a f^n from $(K - H) \times \Sigma$ to $K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$ such that,

(a) for all $q \in K - H$, if $\delta(q, \triangleright) = (p, b)$, then $b = \rightarrow$

(b) for all $q \in K - H$ and $a \in \Sigma$, if $\delta(q, a) = (p, b)$ then $b \neq \triangleright$.

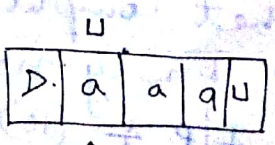
Example: Consider the Turing machine $M = (K, \Sigma, \delta, s, \{h\})$, which $K = \{q_0, q_1, h\}$,

$\Sigma = \{a, \sqcup, \triangleright\}$,

$s = q_0$.

and δ is given by the following table.

q	σ	$\delta(q, \sigma)$
q_0	a	$(q_1, \sqcup)$
q_0	$\sqcup$	$(h, \sqcup)$
q_0	$\triangleright$	$(q_0, \rightarrow)$
q_1	a	(q_0, a)
q_1	$\sqcup$	$(q_0, \rightarrow)$
q_1	$\triangleright$	$(q_1, \rightarrow)$



Machine change all a's to ⊔ until it finds a ⊔.

If T' is started in configuration $(q_0, \triangleright \underline{\sqcup} aaaa)$, its computation would be represented formally as follows :-

$$(q_0, \triangleright \underline{\sqcup} aaaa) \vdash_M (q_0, \triangleright \sqcup \underline{a} aaa)$$
$$\vdash_M (q_1, \triangleright \sqcup \underline{\sqcup} aaa)$$
$$\vdash_M (q_0, \triangleright \sqcup\sqcup \underline{a} aa)$$
$$\vdash_M (q_1, \triangleright \sqcup\sqcup\underline{\sqcup} aa)$$
$$\vdash_M (q_0, \triangleright \sqcup\sqcup\sqcup \underline{a} a)$$
$$\vdash_M (q_1, \triangleright \sqcup\sqcup\sqcup\underline{\sqcup} a)$$
$$\vdash_M (q_0, \triangleright \sqcup\sqcup\sqcup\sqcup \underline{a})$$
$$\vdash_M (q_1, \triangleright \sqcup\sqcup\sqcup\sqcup\underline{\sqcup})$$
$$\vdash_M (q_0, \triangleright \sqcup\sqcup\sqcup\sqcup\sqcup \underline{\sqcup})$$
$$\vdash_M^* (h, \triangleright \sqcup\sqcup\sqcup\sqcup\sqcup \underline{\sqcup})$$

The computation has ten steps.

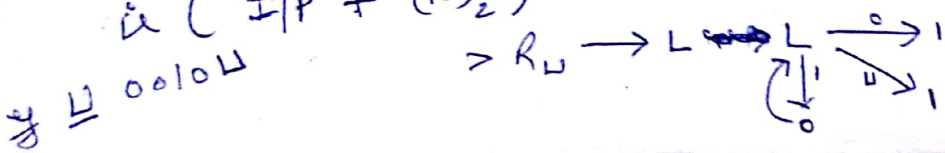
eg. 1) $R_\sqcup$ → go to right until you find a $\sqcup$.

2) $L_\sqcup$ → go to left until you find a $\sqcup$.

TM that computes the f^n $f(m) = m + 2$, where 'm' is a + ive binary no.

i.e. (I/P + $(10)_2$)

last word remains the same.

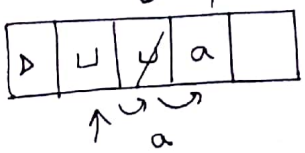


(3). $R_{\sqcup}$ → go to right if symbol scanned is $\sqcup$.

(4). $L_{\sqcup}$ → go to left if symbol scanned is $\sqcup$.

eg: A machine that scans to the right until it finds a non-blank square to the left of non-blank square & then copies the symbol ~~in that~~ square onto the square immediately to the left of where it was found.

$> R \overset{a \neq \sqcup}{\longrightarrow} La.$

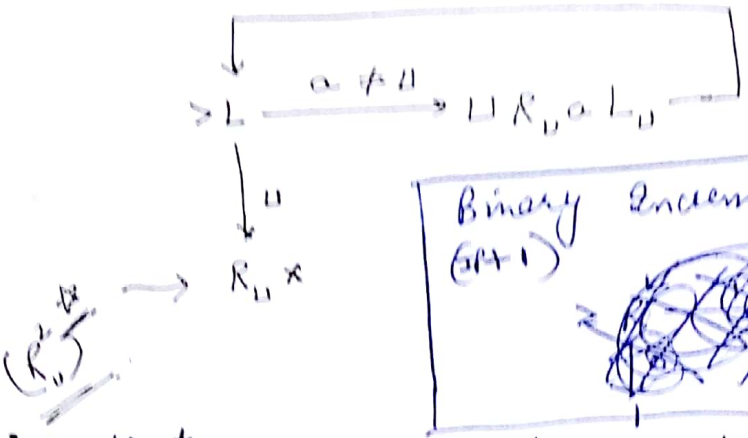


eg: copying machine C, starts with w, that is, if string w, containing only non-blank symbols but possibly empty, is put on an otherwise blank tape with one blank square to its left, and the head is put on the blank square to the left of w, then the machine will eventually stop with w ⊔ w on an otherwise blank tape. We can say that C transforms ⊔ w ⊔ into ⊔ w ⊔ w ⊔.

$> L_{\sqcup} \longrightarrow R \overset{a \neq \sqcup}{\longrightarrow} \sqcup R_{\sqcup}^{2}\, a\, L_{\sqcup}^{2}\, a$

$R \overset{\sqcup}{\longrightarrow} R_{\sqcup}$

eg: The right shifting machine S_R, transforms $\sqcup w \sqcup$, which w contains no blanks, it into $\sqcup\sqcup w \sqcup$.



eg: machine that erases the a's in it's tape.

Binary multiplier (I/P * 2)

⊔ 001010 ⊔ → ⊔ 0010100 ⊔ (simply add 0 in end)

$> R_\sqcup \xrightarrow{\sqcup} 0R$

Cohen (Turing Machine)

1). Σ is the set of alphabets (input letters)

2). Tape is divided into cells. Each cell containing 1 character of string.

3). Γ :- is an alphabet of characters that can be printed on the tape by the tape head. This can also include Δ (blank symbol), which we call erasing & do not include the blank as a letter in the alphabet Γ.

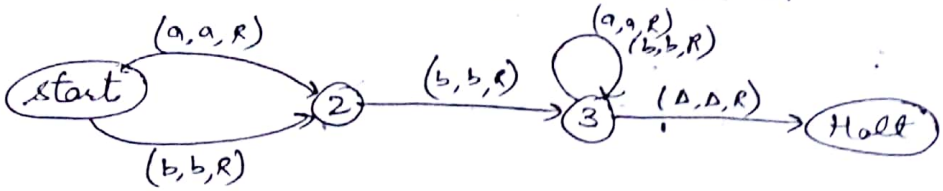
4). Each edge is labelled with a triplet of information:

(letter, letter, direction).

first letter: (either Δ or Σ or Γ) is the character tape head reads from the cell to which it is pointing.

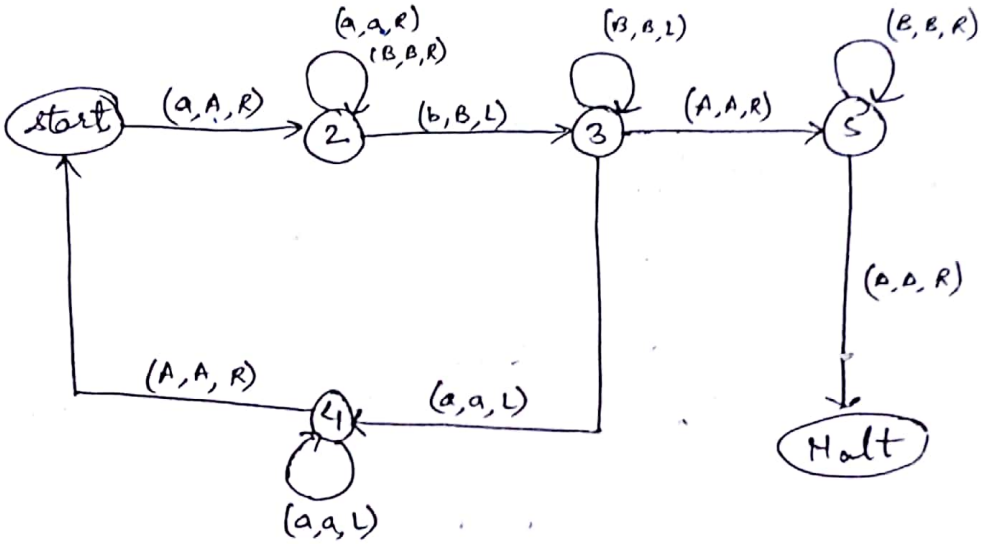
second letter: (either Δ or from Γ) is what tape head prints in the cell before it leaves.

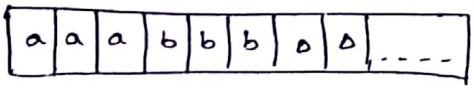
eg 1: TM that reads second character as 'b'

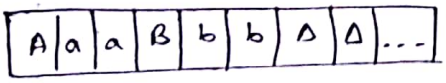


i.e. $(a+b)b(a+b)^*$

eg 2: $a^n b^n$





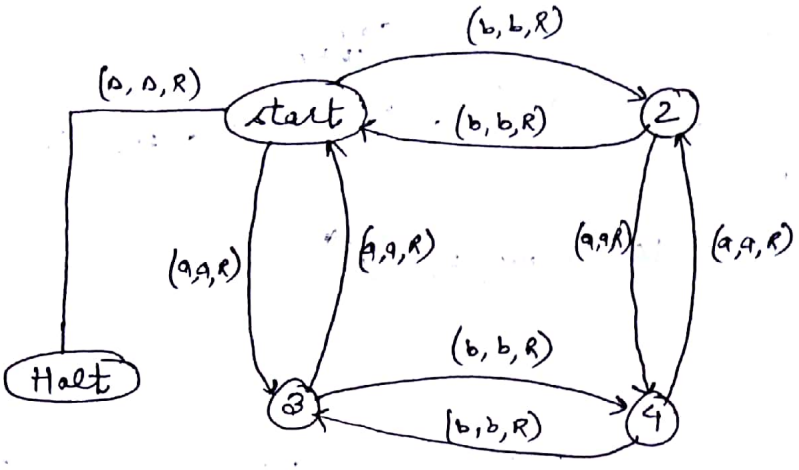


suppose string is aaa bbb, then the states are:-

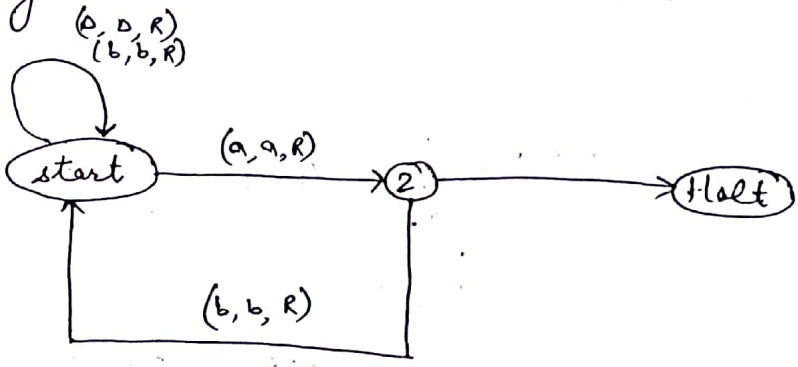
⊢ a̲aa bbb
A a̲a bbb
A aa̲ bbb
A aa B̲ bb
A a̲a Bbb
A̲ aaBbb
A a̲aBbb
AAa̲ Bbb
AAaB̲ bb
AAaBb̲ b
AAaB̲ Bb
AA a̲ BBb
AA̲ a BBb
AAa̲ BBb
AAAB̲ Bb
AAA B̲B b
AAA BB̲ b
AAA BB̲ B

AAA B̲ BB
AAA̲ BBB
AAA̲ BBB
AAA BB̲B
AAA BBB Δ̲
HALT.

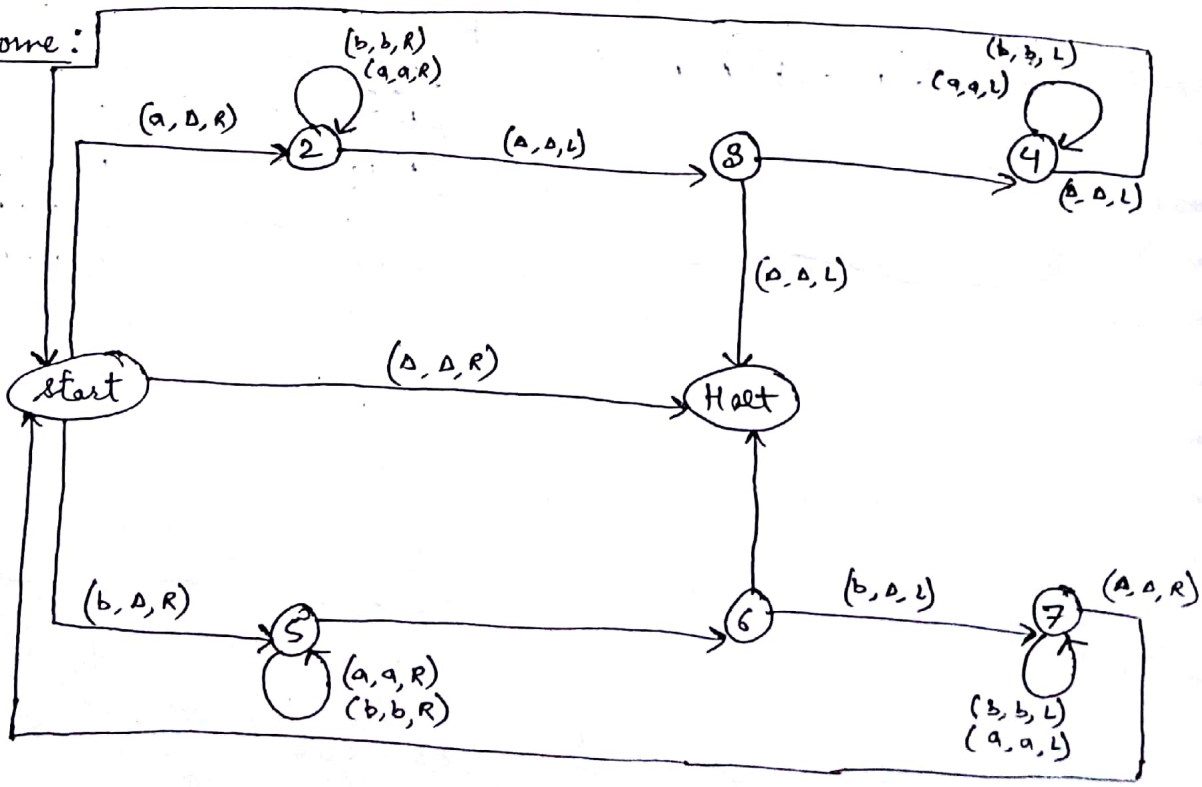
eg. EVEN-EVEN.



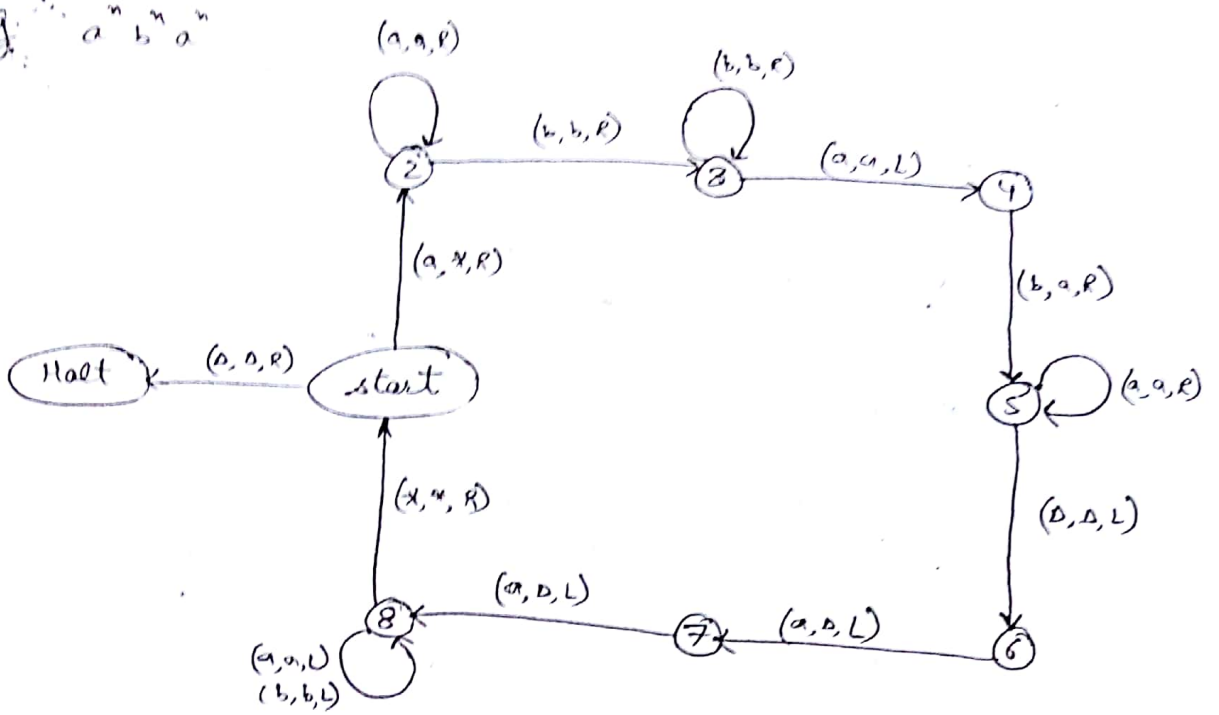
eg. strings having 'double a' in them somewhere.



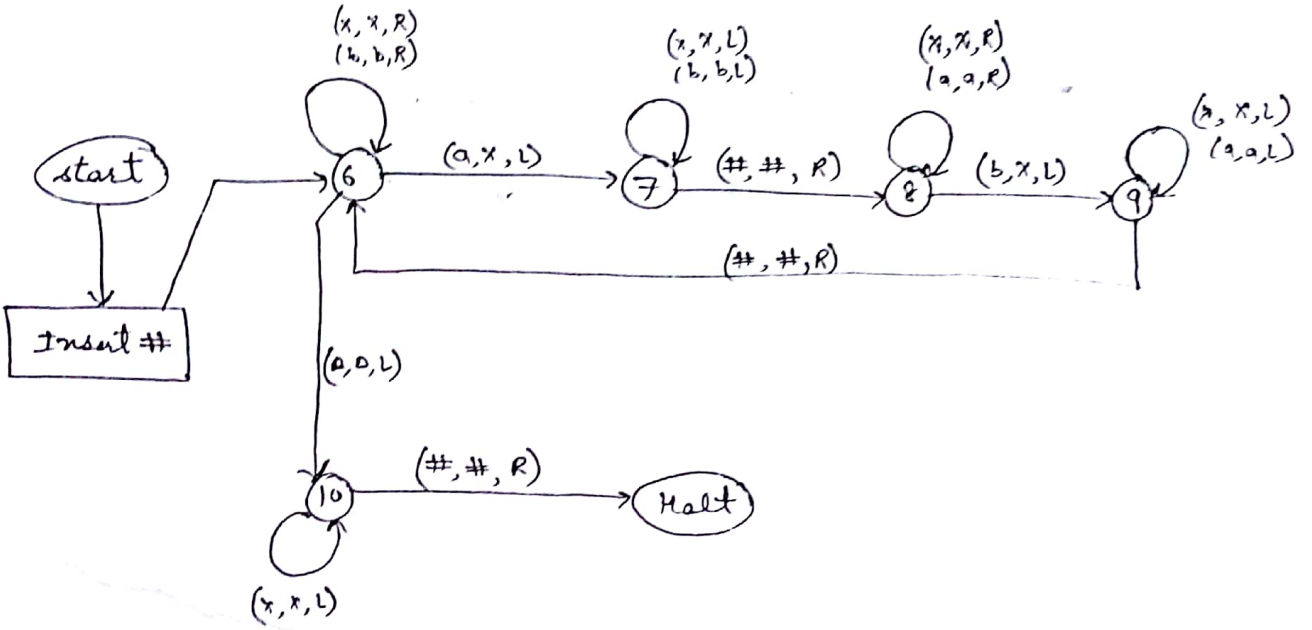
eg. Palindrome:



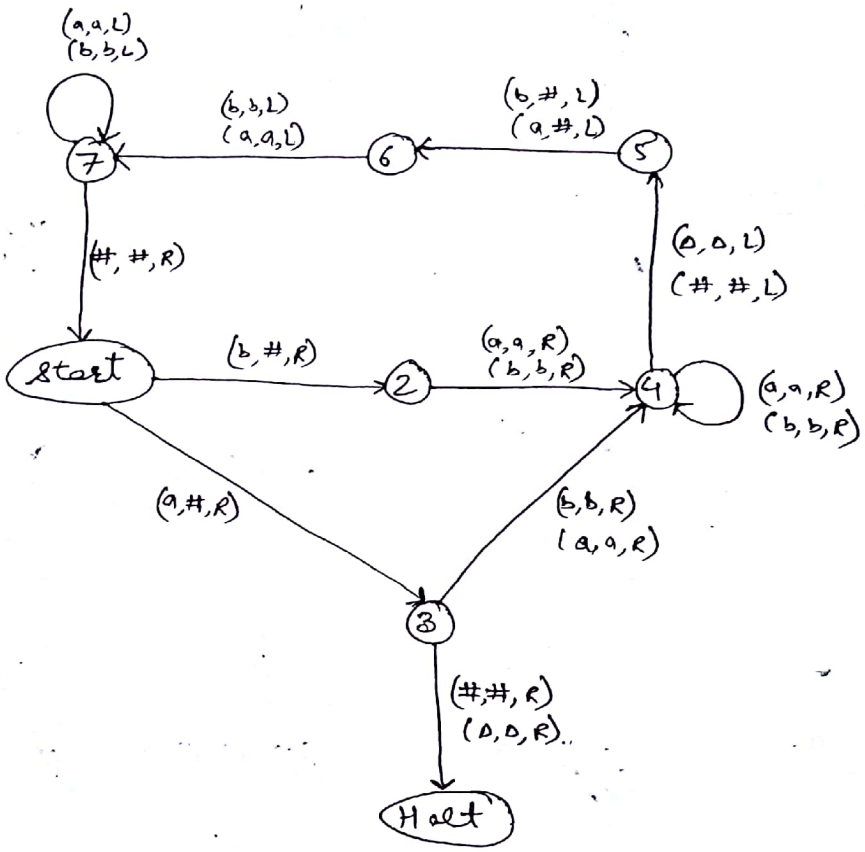
Eg: $a^n b^n a^n$



Eg:



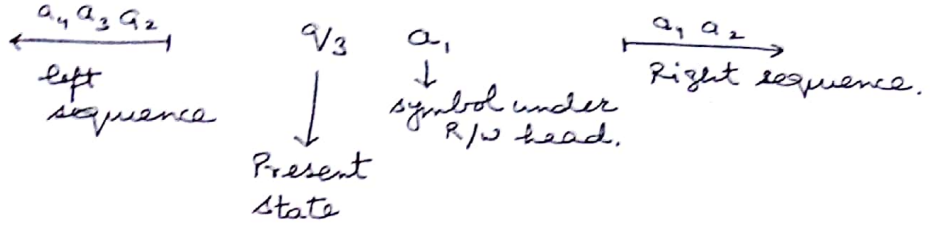
eg. TM that accepts all words with an odd number of letters that have 'a' as the middle letter.



Turing Machines

(K.L.P. Mishra)

Representation of ID



Transition Table of a Turing Machine

If we get $\alpha\beta\gamma$ in the table, it means that,

→ 'α' is to be written in the current cell.

→ 'β' gives the movement of the head.

→ 'γ' denotes the new state into which machine enters.

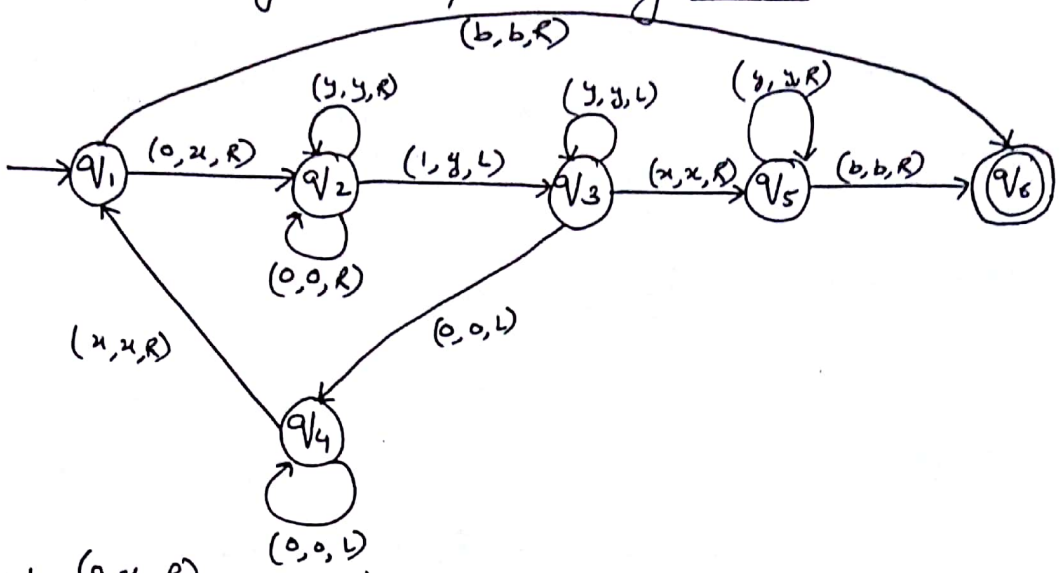
eg:- Transition Table of a TM.

state	Tape symbol. b	0	1
→ q_1	$1Lq_2$	$0Rq_1$	-
q_2	bRq_3	$0Lq_2$	$1Lq_2$
q_3		bRq_4	bRq_5
q_4	$0Rq_5$	$0Rq_4$	$1Rq_4$
(q_5)	$0Lq_2$		

check whether string '00b' is accepted or not.

$\vdash q_1 00b$
$\vdash 0q_1 0b$
$\vdash 00q_1 b$
$\vdash 0q_2 01$
$\vdash q_2 001$
$\vdash q_2 b001$
$\vdash bq_3 001$
$\vdash bbq_4 01$
$\vdash bb0q_4 1$
$\vdash bb01q_4 b$
$\vdash bb010q_5$
$\vdash bb01q_2 00$
$\vdash bb0q_2 100$
$\vdash bbq_1 0100$
$\vdash bq_2 b0100$
$\vdash bbq_3 0100$
$\vdash bbbq_4 100$
$\vdash bbb1q_4 00$
$\vdash bbb10q_4 0$
$\vdash bbb100q_4 b$
$\vdash bbb1000q_5 b$
$\vdash bbb10q_2 000$
$\vdash bbb1q_2 0000$
$\vdash bbbq_2 10000$
$\vdash bbq_2 b10000$
$\vdash bbbq_3 10000$
$\vdash bbbbq_5 0000.$

eg:- M is a Turing Machine. Obtain the computation sequence of M for processing the input string 0011.



Soln:

$\overset{\downarrow}{b}0011b \xrightarrow{(0,x,R)} b\overset{\downarrow}{x}011b$
$q_1 \qquad\qquad\qquad q_2$

	b	0	0	1	1	b

↑ R/W Head.
q_1

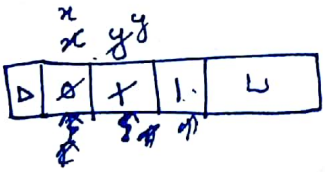
Computation sequence :-

$\overset{\downarrow}{b}0011b \xrightarrow[q_1]{(0,x,R)} b\overset{\downarrow}{x}011b \xrightarrow[q_2]{(0,0,R)} b\overset{\downarrow}{x}011b \xrightarrow[q_2]{(1,y,L)} b\overset{\downarrow}{x}0y1b \xrightarrow[q_3]{(0,0,L)} b\overset{\downarrow}{x}0y1b \xrightarrow[q_4]{(x,x,R)} b\overset{\downarrow}{x}0y1b \xrightarrow[q_1]{(0,x,R)} b\overset{\downarrow}{x}xy1b \xrightarrow[q_2]{(y,y,R)} bx\overset{\downarrow}{x}y1b \xrightarrow[q_2]{(1,y,L)} bx\overset{\downarrow}{x}yyb \xrightarrow[q_3]{(y,y,L)} b\overset{\downarrow}{x}xyyb \xrightarrow[q_3]{(x,x,R)} bx\overset{\downarrow}{x}yyb \xrightarrow[q_5]{(y,y,R)} bx\overset{\downarrow}{x}yyb \xrightarrow[q_5]{(y,y,R)} bxxy\overset{\downarrow}{y}b \xrightarrow[q_5]{(b,b,R)} bxxyyb\overset{\downarrow}{b} \xrightarrow[q_6]{}$

eg: Consider TM 'M' described by the transition table given below. Describe the processing of ⓐ 011 ⓑ 0011 ⓒ 001. using IDs. which of the above strings are accepted by M.

	0	1	x	y	b
→ q_1	xRq_2	-	-	-	bRq_5
q_2	$0Rq_2$	yLq_3	-	yRq_2	-
q_3	$0Lq_4$	-	xRq_5	yLq_3	-
q_4	$0Lq_4$	-	xRq_1	-	-
q_5	-	-	-	yRq_5	bRq_6
(q_6)	-	-	-	-	-

sol:



ⓐ $\vdash q_1 011$
$\vdash x q_2 11$
$\vdash q_3 x y 1$
$\vdash x q_5 y 1$
$\vdash x y q_5 1$ & $\delta(q_5, 1)$ is not defined, M halts & hence Not accepted.

ⓑ $q_1 0011$
$\vdash x q_2 011$
$\vdash x 0 q_2 11$
$\vdash x q_3 0 y 1$
$\vdash q_4 x 0 y 1$
$\vdash x q_1 0 y 1$
$\vdash x x q_2 y 1$
$\vdash x x y q_2 1$
$\vdash x x q_3 y y$
$\vdash x q_3 x y y$
$\vdash x x q_5 y y$
$\vdash x x y q_5 y$
$\vdash x x y y q_5 b$
$\vdash x x y y b q_6$
M halts & q_6 is in accepting state. hence string is accepted.

ⓒ $q_1 001$
$\vdash x q_2 01$
$\vdash x 0 q_2 1$
$\vdash x q_3 0 y$
$\vdash q_4 x 0 y$
$\vdash x q_1 0 y$
$\vdash x x q_2 y$
$\vdash x x y q_2$
M halts. q_2 is Not in accepting state & hence 001 is not accepted by Turing Machine M.

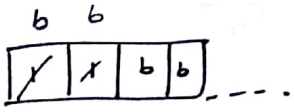
eg: Design a TM to recognize all strings consisting of an even no. of 1's.

Sol^n: steps to be followed:-

(a). q_1 is the initial state. M enters the state q_2 on scanning 1 and writes b.

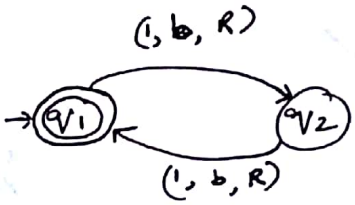
(b). If M is in state q_2 and scans 1, it enters q_1 & write b.

(c). q_1 is the only accepting state.



Transition Table:-

	1
q_1	$b\ q_2\ R$
q_2	$b\ q_1\ R$



computation sequence

(a) 11

$\vdash q_1\ 11$

$\vdash b\ q_2\ 1$

$\vdash bb\ q_1$

& q_1 is accepting state.

(b) 111

$q_1\ 111$

$\vdash b\ q_2\ 11$

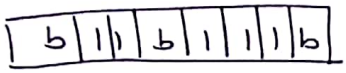
$\vdash bb\ q_1\ 1$

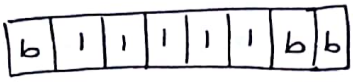
$\vdash bbb\ q_2$

M halts & q_2 is not in accepting state.

Q. Design a TM over $\{1, b\}$ which can compute concatenation function over $\Sigma = \{1\}$. If a pair of words (w_1, w_2) is the input, the o/p has to be $w_1 w_2$.
[w_1, w_2 consist of same characters].

Sol^n. Let us assume that the two words w_1 & w_2 are written initially on the input tape separated by the symbol b. for eg. if $w_1 = 11$ & $w_2 = 111$ then the input & output tapes are :-





respectively.

Steps :-
The main task is to remove the symbol b. This can be done in foll. manner :-

(a). The separating symbol b is found & replaced by 1
(b). The rightmost 1 is found & replaced by a blank b.
(c). The R/w head returns to the starting position.

computation for 11b111.

$q_0 11b111$
$\vdash 1q_0 1b111$
$\vdash 11q_0 b111$
$\vdash 111q_1 111$
$\vdash 1111q_1 11$
$\vdash 11111q_1 1$
$\vdash 111111q_1 b$
$\vdash 11111q_2 1b$
$\vdash 1111q_3 1bb$
$\vdash 111q_3 11bb$
$\vdash 11q_3 111bb$
$\vdash 1q_3 1111bb$
$\vdash q_3 11111bb$
$\vdash q_3 b11111bb$
$\vdash bq_f 11111bb$

Transition Table

States	1	b
$\rightarrow q_0$	$1Rq_0$	$1Rq_1$
q_1	$1Rq_1$	bLq_2
q_2	bLq_3	–
q_3	$1Lq_3$	bRq_f
(q_f)	–	–

computation for 1b1.

$q_0 1b1$
$\vdash 1q_0 b1$
$\vdash 11q_1 1$
$\vdash 111q_1 b$
$\vdash 11q_2 b$
$\vdash 1q_3 1bb$
$\vdash q_3 11bb$
$\vdash q_3 b11bb$
$\vdash bq_f 11bb$

TM that accepts $\{ 0^n 1^n \mid n \geq 1 \}$.

a) leftmost symbol in the given input string is 0, replace it by x & move right till we encounter a leftmost 1 in w. change it to y and move backwards.

b). Repeat @ with the leftmost 0. If we move back & forth & no 0 or 1 remains, move to a final state.

c). For strings not in the form $0^n 1^n$, the resulting state has to be non-final.

Keeping these ideas in mind we construct a TM M as follows:

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

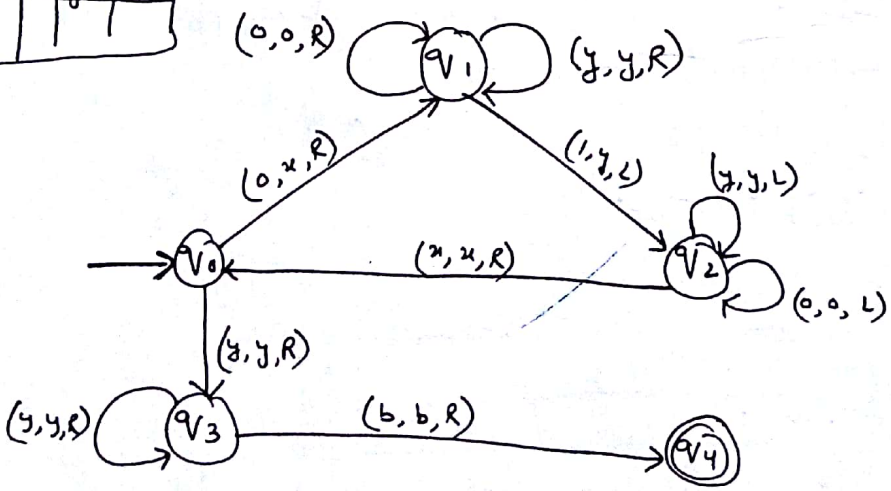
$Q = \{ q_0, q_1, q_2, q_3, q_f \}$

$F = \{ q_f \}$

$\Sigma = \{0, 1\}$

$T = \{0, 1, x, y, b\}$

Σ /	0	1	x	y	b
→ q_0	x			y	
q_1	0	y		y	
q_2	0		x	y	
q_3				y	b
q_4					



$q_0 0011 \vdash x q_1 011 \vdash x0 q_1 11 \vdash x q_2 0y1 \vdash q_2 x0y1 \vdash x q_0 0y1$
$\vdash xx q_1 y1 \vdash xx y q_1 1 \vdash xx q_2 yy \vdash x q_2 xyy \vdash xx q_0 yy \vdash xxy q_3 y$
$\vdash xxyy q_3 = xxyy q_3 b \vdash xxyyb q_4 b$ [accepted]

$q_0 010 \vdash x q_1 10 \vdash q_2 xy0 \vdash x q_0 y0 \vdash xy q_3 0$. [$\delta(q_3, 0)$ is not defined]

Q. Design a TM M to recognise the language

$$\{1^n 2^n 3^n \mid n \geq 1\}.$$

Sol^n. Let us evolve a procedure for the input string 112233. After processing, we require the ID to be of the form bbbbbbq_7.

step 1. q_1 is the initial state. The R/W head scans the leftmost 1, replaces 1 by b & moves to the right. M enters q_2.

step 2. on scanning the leftmost 2, the R/W head replaces 2 by b, & moves to the right. M enters q_3.

step 3. on scanning the leftmost 3, the R/W head replaces 3 by b & moves to the right. M enters q_4.

step 4. After scanning the rightmost 3, the R/W head moves to the left until it finds the leftmost 1. As a result, the leftmost 1, 2, and 3 are replaced by b.

step 5. steps 1-4 are repeated until all 1's, 2's & 3's are replaced by blanks. The change of ID's due to Procing is :- q_1 112233 ⊢ bq_2 12233 ⊢ b1q_2 2233 ⊢ b1bq_3 233 ⊢ b1b2q_3 33 ⊢ b1b2bq_4 3 ⊢ b1b2q_5b3 ⊢ b1b$q_5$2b3 ⊢ b1q_5b2b3 ⊢ b$q_5$1b2b3 ⊢ q_6b1b2b3 ⊢ b$q_1$1b2b3 ⊢ bbq_2b2b3 ⊢ bbb$q_2$2b3 ⊢ bbbbq_3b3 ⊢ bbbbb$q_3$3 ⊢ bbbbbbq_4b ⊢ bbbbb$q_7$bb. , Thus, q_1 112233 ⊢* q_7 bbbbbb. As q_7 is an accepting state, input string 112233 is accepted.

Transition Table.

states	1	2	3	b
→ q_1	b R q_2	–	–	b R q_1
q_2	1 R q_2	b R q_3	–	b R q_2
q_3	–	2 R q_3	b R q_4	b R q_3
q_4	–	–	3 L q_5	b L q_7
q_5	1 L q_6	2 L q_5	–	b L q_5
q_6	1 L q_6	–	–	b R q_1
(q_7)				

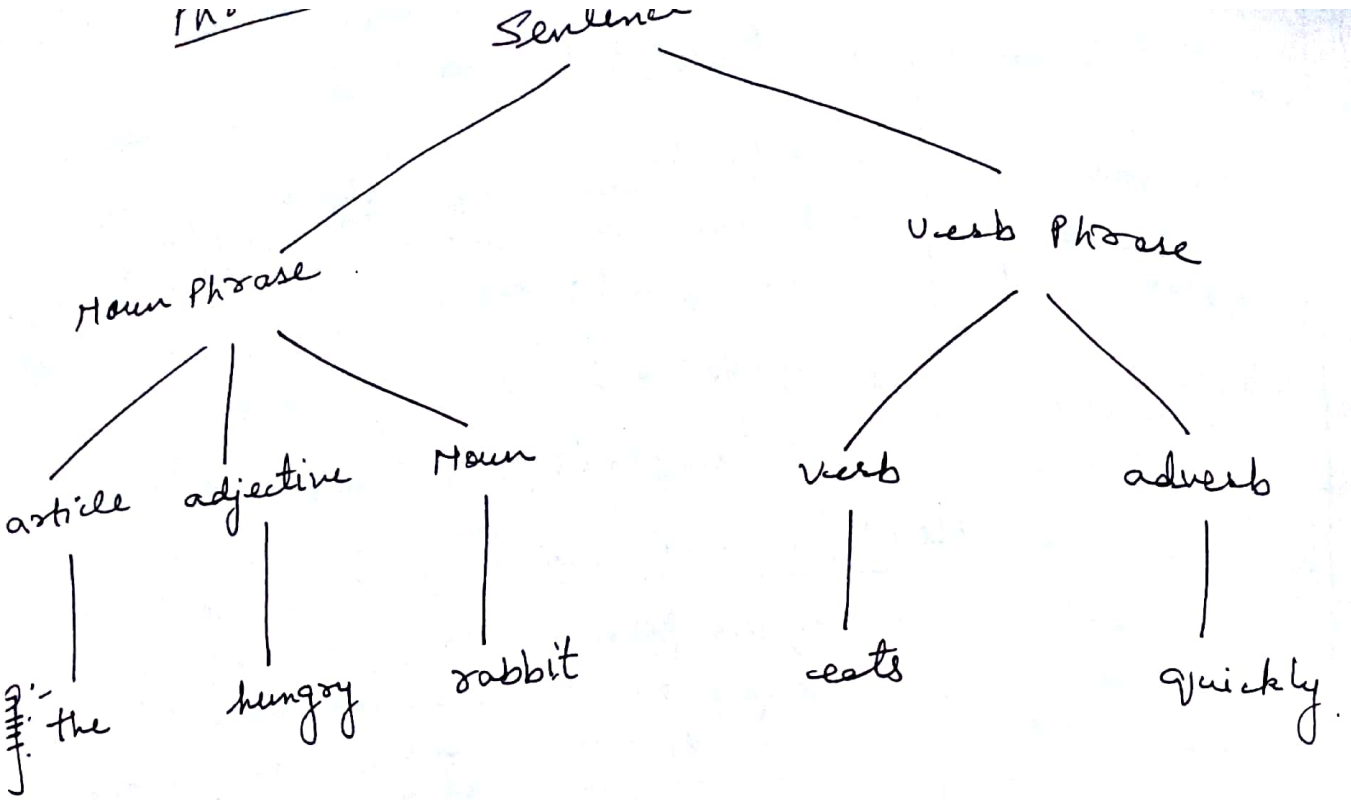
Defⁿ of a Turing Machine.

A TM M is a 7- tuple, namely $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$, where

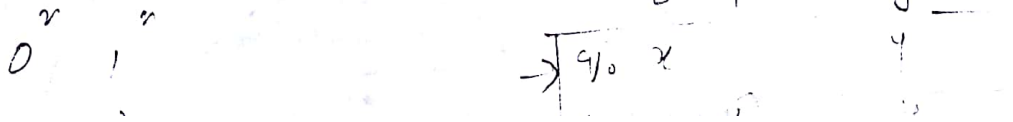
1. Q is a finite nonempty set of states.
2. Γ is a finite nonempty set of tape symbols.
3. $b \in \Gamma$ is the blank.
4. Σ is a non-empty set of input symbols & is a subset of Γ and $b \notin \Sigma$.
5. δ is a transition function mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head. D = L or R according as the movement is to the left or right.
6. $q_0 \in Q$ is the initial state and,
7. $F \subseteq Q$ is the set of Final states.

Notes :- 1) The acceptability of a string is decided by the reachability from the initial state to some final state. so the final states are also called the accepting states.

2). δ may not be defined for some elements of $Q \times \Gamma$.



Recursively enumerable :- A language L over the alphabet Σ is called recursively enumerable if there is a TM T that accepts every word in L and either rejects or loops forever for every word in the language L', the complement of L.



Computing with Turing Machines

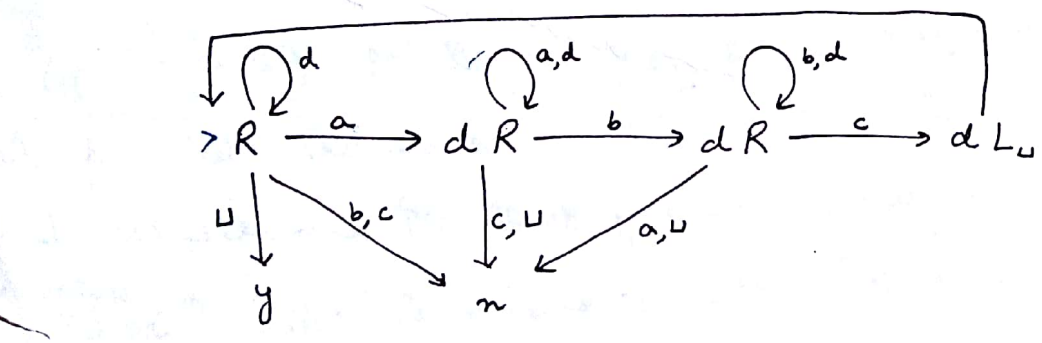
Initial configuration :- If $M = (K, \Sigma, \delta, s, H)$ is a TM and $w \in (\Sigma - \{\sqcup, \triangleright\})^*$, then initial configuration of M on input w is $(s, \triangleright \sqcup w)$.

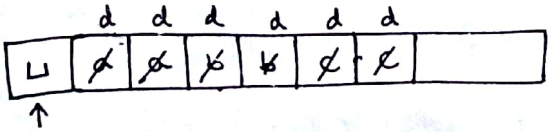
Accepting and Rejecting configuration :-

Let $M = (K, \Sigma, \delta, s, H)$ be a TM, such that $H = \{y, n\}$ consists of two distinguished halting states (y and n for "yes" and "no"). Any halting configuration whose state component is y is called an accepting configuration, while a halting configuration whose state component is 'n' is called a rejecting configuration.

eg: $L = \{a^n b^n c^n : n \geq 0\}$. Draw TM that decides L. Also

machine y makes the new state to be the accepting state y, while machine n moves the state to n.





Recursive functions :- Let $M = (K, \Sigma, \delta, s, \{h\})$ be a Turing Machine, Let $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ be an alphabet and let $w \in \Sigma_0^*$. Suppose that M halts on input w and that $(s, \triangleright \sqcup w) \vdash_M^* (h, \triangleright \sqcup y)$ for some $y \in \Sigma_0^*$. Then 'y' is called o/p of M on input w, and is denoted $M(w)$. $M(w)$ is defined only if M halts on input w, and in fact does so at a configuration of the form $(h, \triangleright \sqcup y)$ with $y \in \Sigma_0^*$.

Now, Let f be any function. we say that M computes function f if, for all $w \in \Sigma_0^*$, $M(w) = f(w)$. That is, for all $w \in \Sigma_0^*$, M eventually halts on input w, & when it does halt, its tape contains the string $\triangleright \sqcup f(w)$. A function f is called recursive, if there is a TM 'M' that computes f.

Recursively enumerable Languages :- Let $M = (K, \Sigma, \delta, s, H)$ be a Turing Machine, let $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$ be an alphabet, & let $L \subseteq \Sigma_0^*$ be a Language. we say that M semidecides L if for any string $w \in \Sigma_0^*$, the following is true: $w \in L$ if and only if M halts on input w. A language L is recursively enumerable if and only if there is a TM 'M' that semidecides L.

Note: If M fails to halt on input w, we write $M(w) = \nearrow$. In this ~~decision~~ notation, we can restate the def^n of semidecision of a language $L \subseteq \Sigma_0^*$ by TM M as follows: For all $w \in \Sigma_0^*$, $M(w) = \nearrow$ if and only if $w \notin L$.

eg: Let $L = \{w \in \{a,b\}^* : w$ contains atleast one $a\}$. Then L is semidecided by the TM shown

This machine, when started in configuration $(q_0, \triangleright \sqcup w)$ for some $w \in (a,b)^*$, simply scans right until an 'a' is encountered & then halts. If no 'a' is found, the machine goes on forever into the blanks that follow its input, never halting. So L is exactly the set of strings w in $\{a,b\}^*$ such that M halts on input w.

∴ M ~~is~~ semidecides L, & thus L is recursively enumerable.

Th: If L is a recursive language, then its complement $\bar{L}$ is also recursive.

Proof: If L is decided by TM $M = (K, \Sigma, \delta, s, \{y, n\})$, then $\bar{L}$ is decided by the TM $M' = (K, \Sigma, \delta', s, \{y, n\})$ which is identical to M except that it reverses the roles of the two special halting states y and n. That is, δ' is defined as follows:

$$\delta'(q,a) = \begin{cases} n & \text{if } \delta(q,a) = y, \\ y & \text{if } \delta(q,a) = n, \\ \delta(q,a) & \text{otherwise.} \end{cases}$$

It is clear that $M'(\omega) = y$ if and only if $M(\omega) = n$ & therefore M' decides $\overline{L}$.

Extensions of TM

(I). **Multiple Tapes:-** Let $K \geq 1$ be an integer. A K-tape TM is a quintuple $(K, \Sigma, \delta, s, H)$ where K, Σ, s and H are as in the definition of the ordinary TM, and δ, the transition function, is a f^n from $(K-H) \times \Sigma^K$ to $K \times (\Sigma \cup \{\leftarrow, \rightarrow\})^K$. That is, for each state q, and each K-tuple of tape symbols $(a_1, \ldots, a_K)$, $\delta(q, (a_1, \ldots, a_K)) = (p, (b_1, \ldots, b_K))$ where p is, as before, the new state and b_j is the action taken by M at tape j.

Also, we insist ~~on~~ that if $a_j = \triangleright$ for some $j \leq K$, then $b_j = \rightarrow$.

F. We adopt the convention that the input string is placed on the first tape, in the same way as it is in conventional standard TM. The other tapes are initially blank, with the head on the leftmost blank square of each. At the end of the computation, a K-tape TM is to leave its o/p on its first tape; the contents of other tapes are ignored.

eg:- Copying machine $\triangleright \sqcup w \sqcup$ into $\triangleright \sqcup w \sqcup w \sqcup$ where $w \in \{a, b\}^*$. (using 2- tape machine).

Sol^n. initially:- first tape : $\triangleright \sqcup w$

second tape : $\triangleright \sqcup$

Actions:-

1) Move the heads on both tapes to the right, copying each symbol on the first tape onto the second tape, until a blank is found on the first tape. The first square of second tape should be left blank.

After (1): first tape : $\triangleright \sqcup w \sqcup$

second tape : $\triangleright \sqcup w \sqcup$

2) Move the head on the second tape to the left until a blank is found.

After (2) first tape : $\triangleright \sqcup w \sqcup$

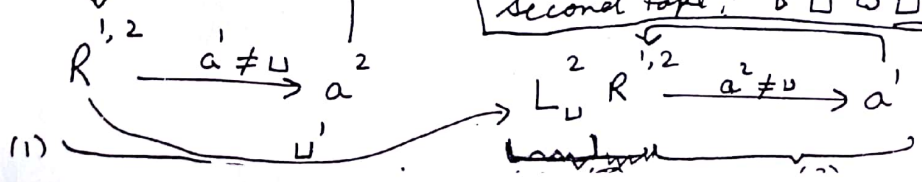
second tape : $\triangleright \sqcup w$

3) Again move the heads on both tapes to the right, this time copying symbols from the second tape onto the first tape. Halt when blank is found on the second tape.

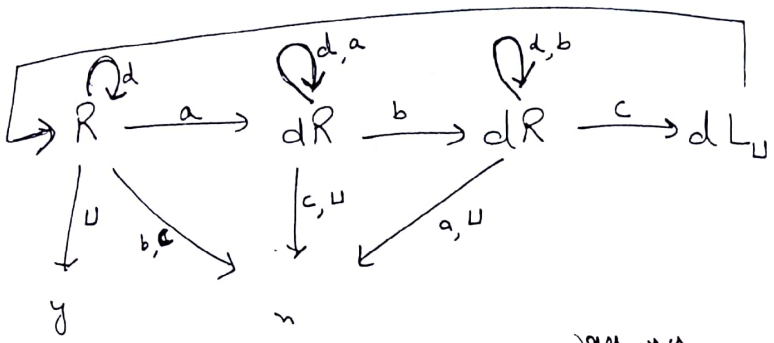
After (3) first tape : $\triangleright \sqcup w \sqcup w \sqcup$

second tape : $\triangleright \sqcup w \sqcup$

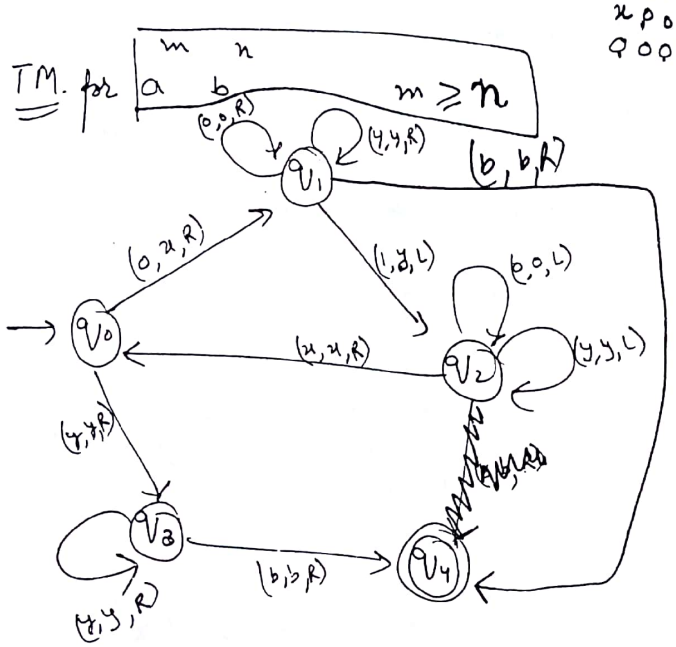
M:



1) TM. for $a^n b^n c^n$, $n \geq 1$.

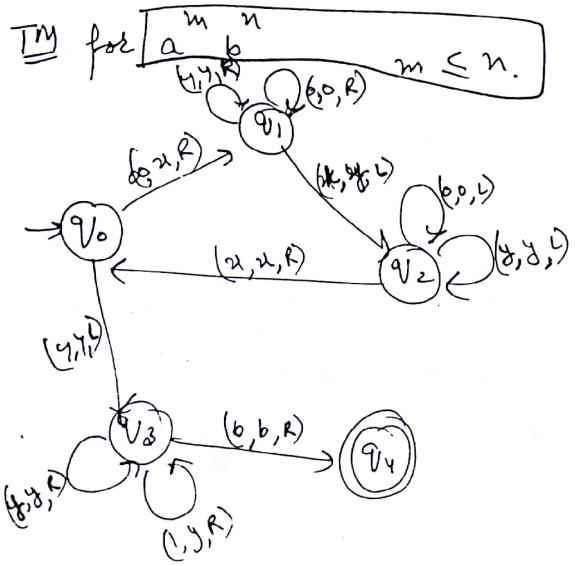


2) TM. for $a^m b^n$, $m \geq n$



	0	1	x	y	b
q_0	x			y	
q_1	0	y		y	
q_2	0		x	y	
q_3				y	b
q_4					

3) TM for $a^m b^n$, $m \leq n$.



	0	1	x	y	b
q_0	x			y	
q_1	0	y		y	
q_2	0		x	y	
q_3		y		y	b
q_4					

Extension of Turing Machines (contd...)

(1) **Multiple Tapes.** eg:- copying machine using 2-tapes.

(2) **Two-way infinite tape:-**

ie Tape ie used is infinite in both directions (ie ▷ has no meaning).

All cells are blank except input letters.

head is initially to the left of input.

(3) **Multiple Heads**

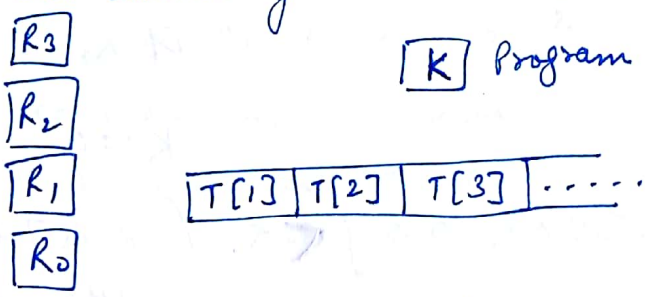
- Single tape, multiple Heads.
- All i/p symbols are scanned in one go. & moved or written independently.

(4) **Two-Dimensional Tape**

Tape is infinite in the form of infinite grid.

Random Access Turing Machines (4.4).

A Random Access TM has a fixed no. of registers & a one-way infinite tape. Each register & each tape square is capable of containing an arbitrary natural no.



Prog. of Random Access TM is a sequence of inst.^{n}

Initially reg. values are 0, prog. counter is 1 & tape contents encode the I/P string.

No.	Instructions	operand	Semantics
(i)	read	j	$R_0 := T[R_j]$
	write	j	$T[R_j] := R_0$
	store	j	$R_j := R_0$
	load	j	$R_0 := R_j$
	load	=c	$R_0 := c$
	add	j	$R_0 := R_0 + R_j$
	add	=c	$R_0 := R_0 + c$
	Sub	j	$R_0 := \max\{R_0 - R_j, 0\}$
	sub	=c	$R_0 := \max\{R_0 - c, 0\}$
	half		$R_0 := \lfloor \frac{R_0}{2} \rfloor$
	jump	s	$K := s$
	jpos	s	if $R_0 > 0$ then $K := s$
	j zero	s	if $R_0 = 0$ then $K := s$
(p)	halt		$K := 0$

j stands for Register No, $0 \leq j < k$. (k is Total no. of Reg. Here $k = 4$).

$T[i]$ denotes the current contents of tape square i.

R_j denotes the current contents of Register j.

$s \leq p$ denotes any instⁿ no. in the program. (ie $K = 1 \rightarrow$ read, $K = p \rightarrow$ halt)

c is any natural no.

All instⁿ change K to K+1 unless explicitly stated.

R_0 acts as accumulator where where all arithmetic & logical instⁿ takes place.

The K^{th} inst^{n} of Prog. will be executed next until a 'halt' inst^{n} is executed. At this point operation of Ran

Def^{n} - Random Access TM

A Random Access TM is a pair $M = (k, \Pi)$, when $k > 0$ is the no. of registers, & $\Pi = (\pi_1, \pi_2, \ldots, \pi_p)$, the program, is a finite seq. of inst^{n}, where each inst^{n} π_i is of one of the types (inst^{n} set.). we assume last inst π_p as halt inst^{n}.

Configuration of a random access TM (k, Π) is a $k+2$-tuple $(K, R_0, R_1, \ldots, R_{k-1}, T)$, where

$K \in N$ is the program counter, $0 \leq K \leq p$. The configuration is a halted configuration if $K = 0$.

for each j, $0 \leq j < k$, $R_j \in N$ is the current value of Reg j.

T, the tape contents, is a finite set of pairs of +tive integers.

i.e.

$\therefore$ configuration $C = (K, R_0, R_1, \ldots, R_{k-1}, T)$.

eg:- multiplication - TM. has no multiplication inst^{n} imply!

R_0 contains x & R_1 contains y. i.e $\begin{pmatrix} x \times y \\ 5 \times 3 \end{pmatrix}$. After Halt is executed R_0 contains the product. Multiplication is done by successive additions.

Initial config. $(1; 5, 3, 0, 0, 0; \phi)$.

↓

($k = 5$ Registers & TM does not interact with T i.e why ϕ).

1. store 2
   ↳ $(2; 5, 3, 5, 0, 0; \phi)$
2. load 1
   ↳ $(3; 3, 3, 5, 0, 0; \phi)$
3. jzero 19
   ↳ $(4; 3, 3, 5, 0, 0; \phi)$
4. Half
   ↳ $(5; 1, 3, 5, 0, 0; \phi)$
5. store 3
   ↳ $(6; 1, 3, 5, 1, 0; \phi)$
6. load 1
   ↳ $(7; 3, 3, 5, 1, 0; \phi)$
7. Sub 3
   ↳ $(8; 2, 3, 5, 1, 0; \phi)$
8. Sub 3
   ↳ $(9; 1, 3, 5, 1, 0; \phi)$
9. Jzero 13
   ↳ $(10; 1, 3, 5, 1, 0; \phi)$

10. load 4
↳ (11; 0,3,5,1,0; ∅)

11. add 2
↳ (12; 5,3,5,1,0; ∅)

12. store 4
↳ (13; 5,3,5,1,5; ∅)

13. load 2
↳ (14; 5,3,5,1,5; ∅)

14. add 2
↳ (15; 10,3,5,1,5; ∅)

15. store 2
↳ (16; 10,3,10,1,5; ∅)

16. load 3
↳ (17; 1,3,10,1,5; ∅)

17. store 1
↳ (18; 1,1,10,1,5; ∅)

19. load 4

18. jump 2
↳ (2; 1,1,10,1,5; ∅)

19. load 4 ⊢* (18; 0,0,20,0,15; ∅)
⊢ (2; 0,0,20,0,15; ∅)

20. halt. ⊢ (3; 0,0,20,0,15; ∅)
⊢ (19; 0,0,20,0,15; ∅)
⊢ (20; 15,0,20,0,15; ∅)

abbreviation of 'mpy' prog.
$(w = x * y)$

w := 0
while y > 0 do
begin
z := half(y)
if y - z - z ≠ 0 then
w := w + x
x := x + x
y := z
end
halt

Relation b/w these two programs :-

y stands for R_1,
x " " R_2,
z " " R_3,
w " " R_4

Defⁿ 2. Random Access TM.

The initial configuration of a random access TM $M = (k, \pi)$ with I/P $w = a_1 a_2 \ldots a_n \in (\Sigma - \{\sqcup\})^*$ is $(K, R_0, R_1, \ldots, R_{k-1}, T)$ where $K = 1$, $R_j = 0$ for all j, & $T = \{(1, E(a_1)), (2, E(a_2)), \ldots, (n, E(a_n))\}$ where E is a fixed bijection b/w Σ and $\{0, 1, \ldots, |\Sigma|-1\}$.

Also assume $E(\sqcup) = 0$.

we say that M accepts string $x \in \Sigma^*$ if the initial config. with I/P x yields a halted config with $R_0 = 1$.

we say that M rejects string $x \in \Sigma^*$ if the initial config. with I/P x yields a halted config. with $R_0 = 0$.

Let $\Sigma_0 \subseteq \Sigma - \{\sqcup\}$ be an alphabet & let $L \subseteq \Sigma_0^*$ be a lang. we say that M decides L if whenever $x \in L$, M accepts x & whenever $x \notin L$, M rejects x.

we say that M semidecides L if the foll. is true:

$x \in L$ iff M on I/P x yields some halted config.

Let $f: \Sigma_0^* \mapsto \Sigma_0^*$ be a function. we say that M computes f, if for all $x \in \Sigma_0^*$, the starting configuration of machine M with I/P x yields a halted configuration with these tape contents:

$\{(1, E(a_1)), (2, E(a_2)), \ldots, (n, E(a_n))\}$, where $f(x) = a_1 \ldots a_n$

eg:- Random access TM (in abbreviated form), deciding the language $\{a^n b^n c^n : n \geq 0\}$.

acount = bcount = ccount = 0, n = 1
while T[n] = 1 do: n := n+1, acount := acount + 1
while T[n] = 2 do: n := n+1, bcount := bcount + 1
while T[n] = 3 do: n := n+1, ccount := count + 1
if acount = bcount = count & T[n] = 0 then accept else reject.

Here, we assume, $E(a) = 1$, $E(b) = 2$, $E(c) = 3$ & variables acount, bcount, ccount to stand for nos of a's, b's & c's resp. Also, the abbreviation 'accept' for "load = 1, halt" & "reject" for "load = 0, halt".

4.5. Non-deterministic Turing machine

Let $M = (K, \Sigma, \Delta, s, H)$ be a non-deterministic TM where Δ is a subset of $((K-H) \times \Sigma) \times (K \times (\Sigma \cup \{\leftarrow, \rightarrow\}))$, rather than a f^n from $(K-H) \times \Sigma$ to $K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$.

we say that M accepts an input $w \in (\Sigma - (\triangleright, \sqcup))^*$ if $(s, \triangleright \underline{\sqcup} w) \vdash_M^* (h, u\underline{a}v)$ for some $h \in H$ & $a \in \Sigma$ & $u, v \in \Sigma^*$.

Note that a Nondeterministic machine accepts an I/P even though it may have many non halting computations on this I/P, as long as atleast one halting computation exist.

we say that M semidecides a lang. $L \subseteq (\Sigma - \{\triangleright, \sqcup\})^*$ if the foll. holds for all $w \in (\Sigma - \{\triangleright, \sqcup\})^*$: $w \in L$ iff M accepts w.

Not in course for (CBCS)

ch-5. undecidability

5.1. church Turing Thesis.

Turing machine that is guaranteed to halt on all inputs are called 'Algorithms'. This principle is known as 'church-Turing Thesis'. It is a thesis, not a theorem, because it is not a mathematical result. It simply asserts that a certain informal concept (algorithm) corresponds to a certain mathematical object (Turing machine).

5.2. universal Turing machines

Turing machines are considered as programming languages, in which other programs can be written. Programs written in this lang. can then be interpreted by a universal Turing machine.

Let $M = (K, \Sigma, \delta, s, H)$ be a TM. Let i & j be the smallest integers such that $2^i \geq |K|$ & $2^j \geq |\Sigma| + 2$.

each state in K will be represented as q followed by binary string of length i.

each symbol in Σ will be rep. as letter 'a' followed by string of j bits.

head directions ($\leftarrow, \rightarrow$) are treated as "honorary tape symbols" (is why "+2" in def" of j).

Rep. for special symbols $\sqcup, \triangleright, \leftarrow, \rightarrow$ are always fixed in lexicographical four smallest symbols resp. ie

$\sqcup$ rep. as $a\,0^j$

$\triangleright$ rep. as $a\,0^{j-1}\,1$

$\leftarrow$ rep. as $a\,0^{j-2}\,10$

$\rightarrow$ rep. as $a\,0^{j-2}\,11$

start state will always be represented as lexicographically first state q_0.

we denote the representation of whole TM M as "M".

"M" consists of Transition Table & δ. That is, it is a

seq. of strings of the form (q, a, p, b) with q & p representation of states & a, b of symbols.

Quadruples are listed in increasing lexicographic order, starting with $\delta(s, \sqcup)$.

The set of halting states H will be determined indirectly by the absence of its states as first components in any quadruple of "M".

If M decides a lang. & thus $H = \{y, n\}$, we will adopt the convention that y is the lexicographically smallest of the two halt states.

eg:- consider $M = (K, \Sigma, \delta, s, \{y, n\})$, where $K = \{s, q, h\}$, $\Sigma = \{\sqcup, \triangleright, a\}$ & δ is given by

state	symbol	δ
s	a	$(q, \sqcup)$
s	$\sqcup$	$(h, \sqcup)$
s	$\triangleright$	$(s, \rightarrow)$
q	a	(s, a)
q	$\sqcup$	$(s, \rightarrow)$
q	$\triangleright$	$(q, \rightarrow)$

Design "M".

sol^n: There are 3 states in K ie $|K| = 3$.
" " 3 symbols in Σ ie $|\Sigma| = 3$.

First find smallest 'i' & 'j'.

for i, we have
$2^i \geq |K|$
ie $2^i \geq |3|$
$\therefore \boxed{i = 2}$

for j, we have,
$2^j \geq |\Sigma| + 2$
or $2^j \geq 3 + 2$
or $2^j \geq 5$
$\therefore \boxed{j = 3}$

Thus, states & symbols are represented as follows:-

	state / symbol	representation.
K	s	$q00$ // start state rep. by $q0^i$
	q	$q01$
	h	~~q00~~ $q11$
Σ	$\sqcup$	$a000$ // $a0^i$
	$\triangleright$	$a001$ // $a0^{i-1}1$
	$\leftarrow$	$a010$ // $a0^{i-2}10$
	$\rightarrow$	$a011$ // $a0^{i-2}11$
	Σ [a] rest in Σ.	$a100$

lex. order

Thus, the representation of the string $\triangleright aa \sqcup a$ is

"$\triangleright aa \sqcup a$" = a001 a100 a100 a000 a100

The representation "M" of TM M is the following string:

"M" = $(q00, a100, q01, a000)$, $[\delta(s, a) = (q, \sqcup)]$
$(q00, a000, q11, a000)$, $[\delta(s, \sqcup) = (h, \sqcup)]$
$(q00, a001, q00, a011)$,
$(q01, a100, q00, a100)$,
$(q01, a000, q00, a011)$,
$(q01, a001, q01, a011)$ $[\delta(q, \triangleright) = (q, \rightarrow)]$

Universal Turing machine U :- U uses the encoding of other machines as programs to direct its operation. U takes two arguments, a description of machine M, "M" & description of an input string w, "w". U is to have the following property: U halts on input "M" "w" iff M halts on input w.

ie $U("M", "w") = "M(w)"$.

5.3. Halting Problem

Suppose program P is written in C++, & input X of that Prog P is also given. Then, we have a program

$$halts(P, X) = \begin{cases} \text{returns 'yes' if } P \text{ halt on input } X. \\ \text{returns 'no' if } P \text{ does not halt (run forever) on input } X. \end{cases}$$

Now consider another Program:-

```
diagonal(X)
a: if halts(X, X) then goto a else halt.
```

Here 'Halts' Program decides that Program X would halt if presented with itself as input, then diagonal(X) loops forever, otherwise it halts.

unanswerable Question:-

Does diagonal (diagonal) halt?

It halts iff the call halts (diagonal, diagonal) returns "no"

OR

It halts iff it does not halt.

This is a contradiction. Because we started assuming that halts (P,X) = no iff P does not halt on I/P X.

∴ the Program halts (P, X) does not exist. ie there is no program, no algorithm for solving the problem 'halts' would solve: to tell whether arbitrary program would halt or loop.