Chapter-4 Turing Machine

→ Turing machine is a 5-tuple $(Q, \Sigma, \delta, q_0, H)$ where,

Q is finite set of states

Σ is alphabet set containing blank symbol, # and left ~~mos~~ end symbol, ▷

i.e. if Σ_0 is set of input alphabet,

$$\Sigma = \Sigma_0 \cup \{\#, \triangleright\}$$

q_0 is initial state

$H \subseteq Q$ is the halting state.

$\delta : (Q-H) \times \Sigma \to Q \times (\Sigma \cup \{\leftarrow, \rightarrow\})$ is transition function such that,

$\forall q \in (Q-H), \delta(q, \triangleright) = (p, b)$ then, $b = \rightarrow$

$\forall q \in (Q-H), \delta(q, a) = (p, b)$ then $b \neq \triangleright$

Thus, when turing machine is in state q & scans a, it enters the state p & eithes writes

b, if $b \in \Sigma$

or, moves left if $b = \leftarrow$

or, moves right if $b = \rightarrow$

→ Machine stops computation after reaching the halting state, $h \in H$.

→ It always moves right on left end symbol.

Configuration:-

Configuration of a T.M.

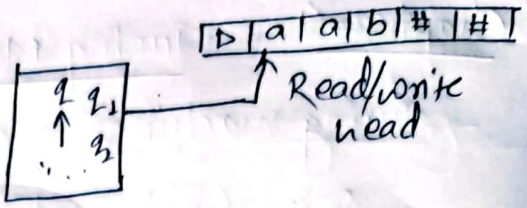
$T = (Q, \Sigma, \delta, q_0, H)$ is the member of

$$Q \times \triangleright\Sigma^* \times (\Sigma^*(\Sigma - \{\#\}) \cup \{e\})$$

For eg,

Any instance of Turing Machine, T can be given by its configuration $(q, \triangleright a, ab)$

This means, the machine is at state q and the scan head is at first input symbol 'a'.



This can also be represented by,
$(q, \triangleright \underline{a} abb)$ for simplicity where underscore gives the read/write head position.

Configuration ⊢ Configuration of Turing machine :-

For T.M, $T = (Q, \Sigma, \delta, q_0, H)$ and its configurations,

~~(q_1, w_1, a_1, u_1) and (q_2, w_2, a_2, u_2),~~

~~$(q_1, w_1, a_1, u_1) \vdash_T (q_2, w_2$~~

$(q_1, w_1 \underline{a_1} u_1)$ and $(q_2, w_2 \underline{a_2} u_2)$,

$(q_1, w_1 \underline{a_1} u_1) \vdash_T (q_2, w_2 \underline{a_2} u_2)$

if and only if, for some $b \in \Sigma \cup \{\leftarrow, \rightarrow\}$,

$\delta(q_1, a_1) = (q_2, b)$ and

either, $b \in \Sigma$, $w_1 = w_2$, $u_1 = u_2$, $a_2 = b$

or, $b = \leftarrow$, $w_1 = w_2 a_2$ and

either, $u_2 = a_1 u_1$ if $a_1 \neq \#$, $u_1 \neq e$

or, $u_2 = e$ if $a_1 = \#$, $u_1 = e$

or, $b = \rightarrow$, $w_2 = w_1 a_1$ and either $u_1 = a_1 u_2$

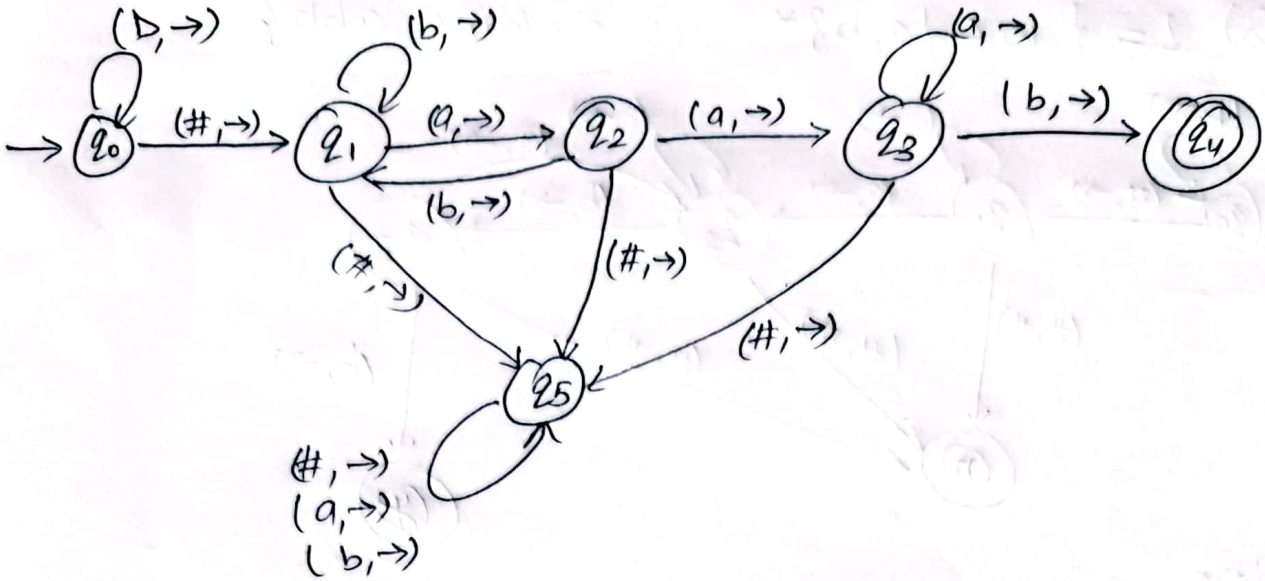
or, $u_1 = u_2 = e$, $a_2 = \#$

→ $\vdash_T^*$ is reflexive transitive closure of $\vdash_T$ so that configuration C_1 yields C_2 when $C_1 \vdash_T^* C_2$.

Example:

Construct a T.M that accepts the language $L = \{w \in \{a, b\}^* ;$ w contains substring $aab\}$.

→ Given the initial configuration $(q_0, \underline{\triangleright} \# w \#)$



Required Turing machine,

$T = (Q, \Sigma, \delta, q_0, H)$ where,

Set of states, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

Alphabet set, $\Sigma = \{a, b, \#, \triangleright\}$, $\Sigma_0 = \{a, b\}$, $\omega \in \Sigma_0^*$

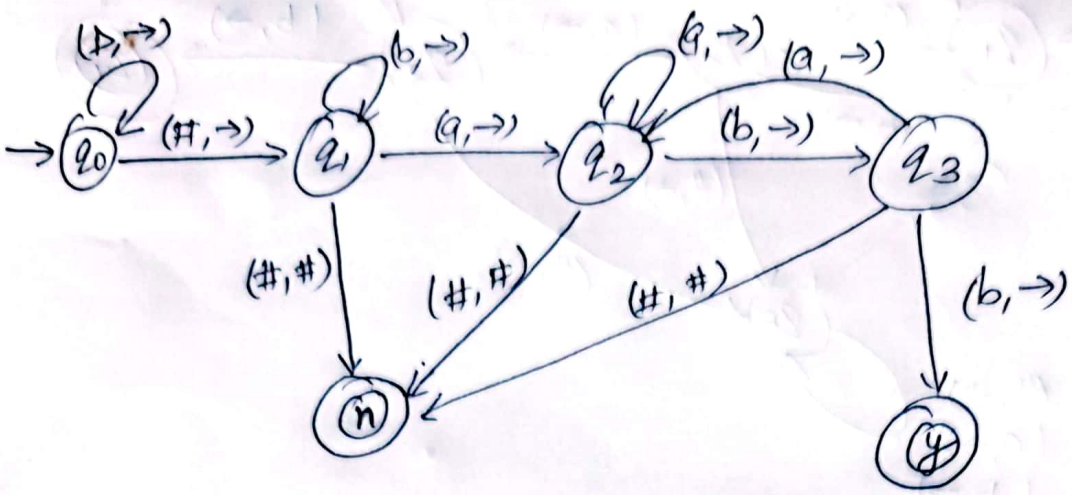
Transition function, $\delta = \{((q_0, \triangleright), (q_0, \rightarrow)), ((q_0, \#), (q_1, \rightarrow)),$
$((q_1, b), (q_1, \rightarrow)), ((q_1, a), (q_2, \rightarrow)),$
$((q_1, \#), (q_5, \rightarrow)), ((q_2, a), (q_3, \rightarrow)),$
$((q_2, b), (q_1, \rightarrow)), ((q_2, \#), (q_5, \rightarrow)),$
$((q_3, a), (q_3, \rightarrow)), ((q_3, b), (q_4, \rightarrow)),$
$((q_3, \#), (q_5, \rightarrow)), ((q_5, a), (q_5, \rightarrow)),$
$((q_5, b), (q_5, \rightarrow)), ((q_5, \#), (q_5, \rightarrow))\}$

q_0 is initial state.

Halting state is given as,

$H = \{q_4\}$

8) $L = \{ w \in \{a,b\}^* \ ; w \text{ contains 'abb'} \}$

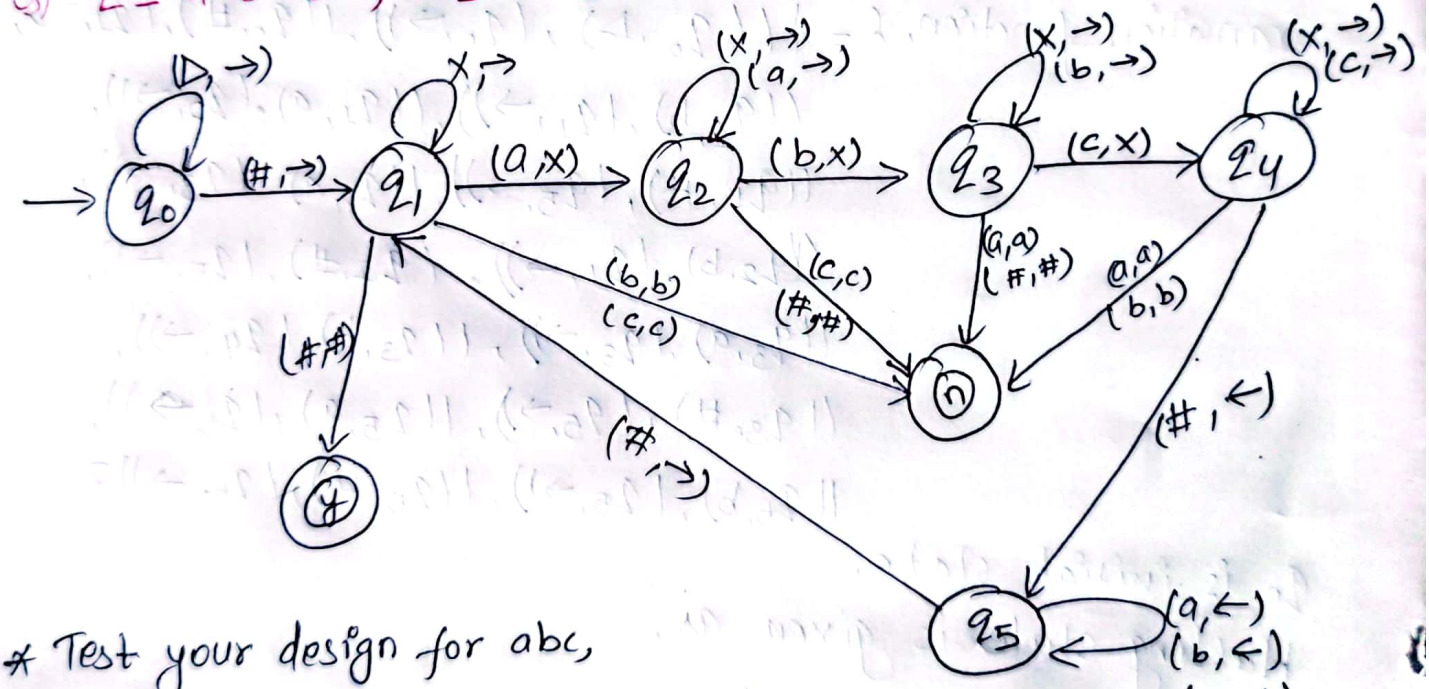


Here,

$H = \{y, n\}$ ← rejecting halting state

↑ accepting halting state

5) $L = a^n b^n c^n ; n \geq 0$ [Also called deciding T.M.]



* Test your design for abc,

$(q_0, \underline{\triangleright}\#abc\#) \vdash (q_0, \triangleright\underline{\#}abc\#) \vdash (q_1, \triangleright\#\underline{a}bc\#)$

$\vdash (q_2, \triangleright\#\underline{X}bc\#) \vdash (q_2, \triangleright\#X\underline{b}c\#)$

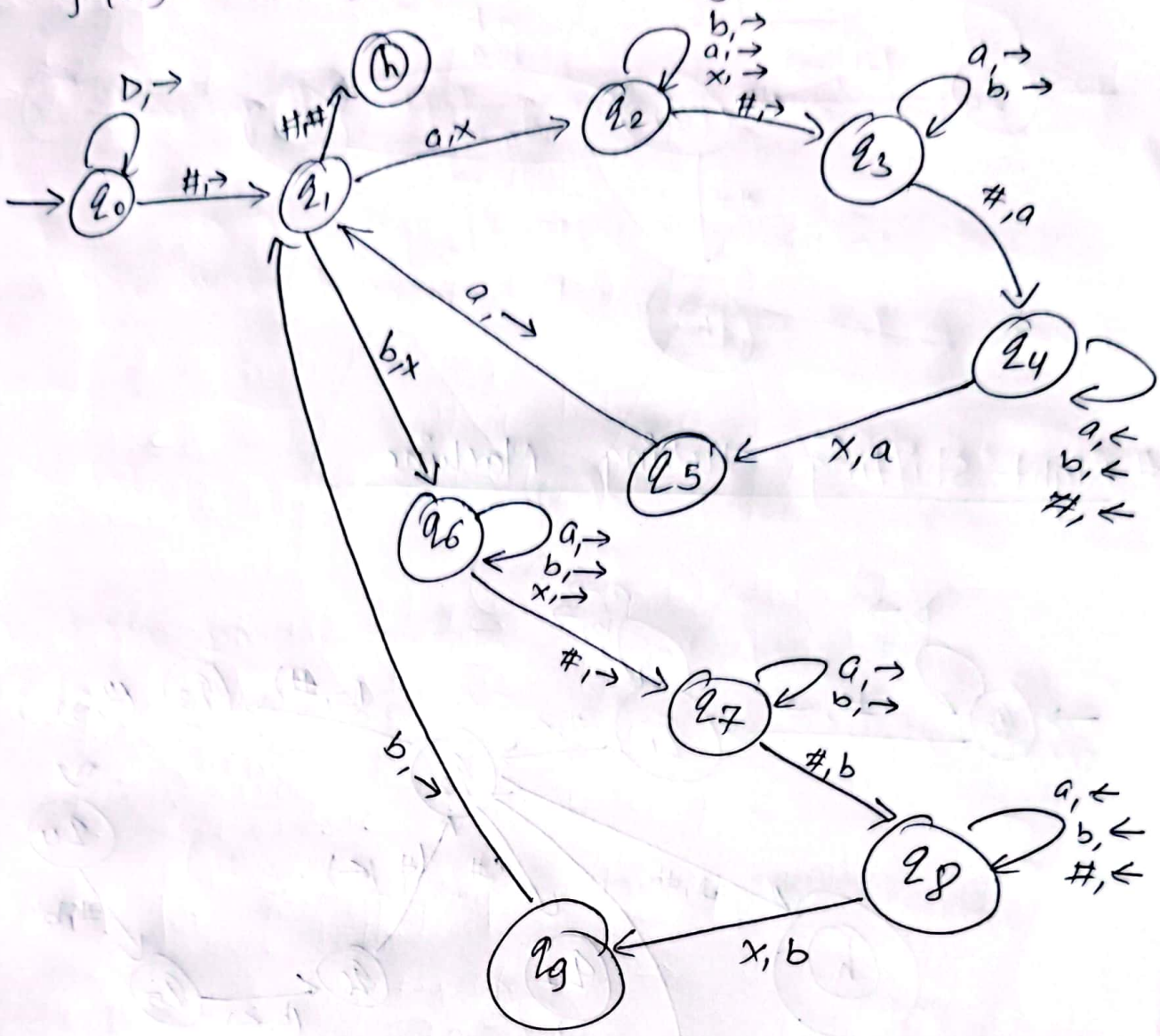
$\vdash (q_3, \triangleright\#X\underline{X}c\#) \vdash (q_3, \triangleright\#XX\underline{\#})$

$\vdash (q_4, \triangleright\#XX\underline{X}\#) \vdash (q_4, \triangleright\#XXX\underline{\#}) \vdash (q_5, \triangleright\#XX\underline{X}\#)$

$\ldots\ldots (y, \triangleright\underline{\#}XXX\#)$ //

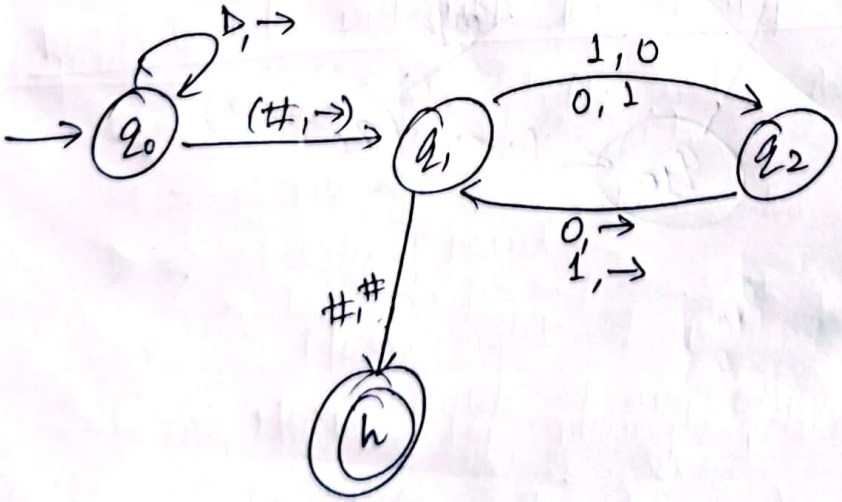
Example-2

$f(w) = ww$ [Also called copying T.M]

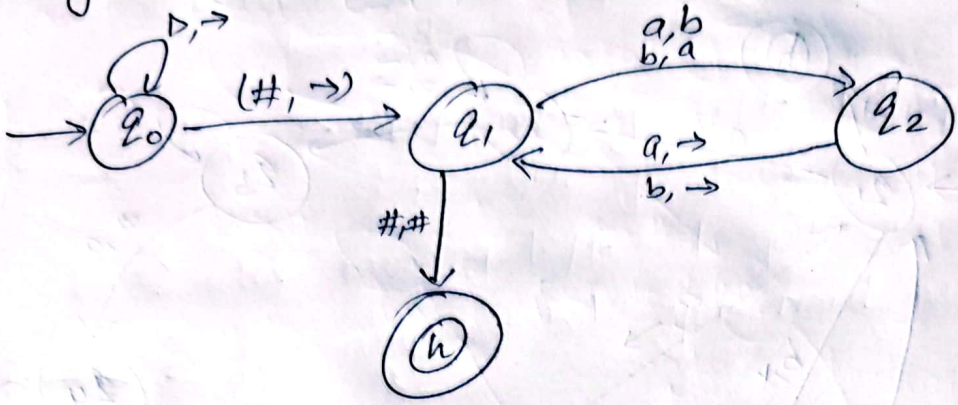


Example-3

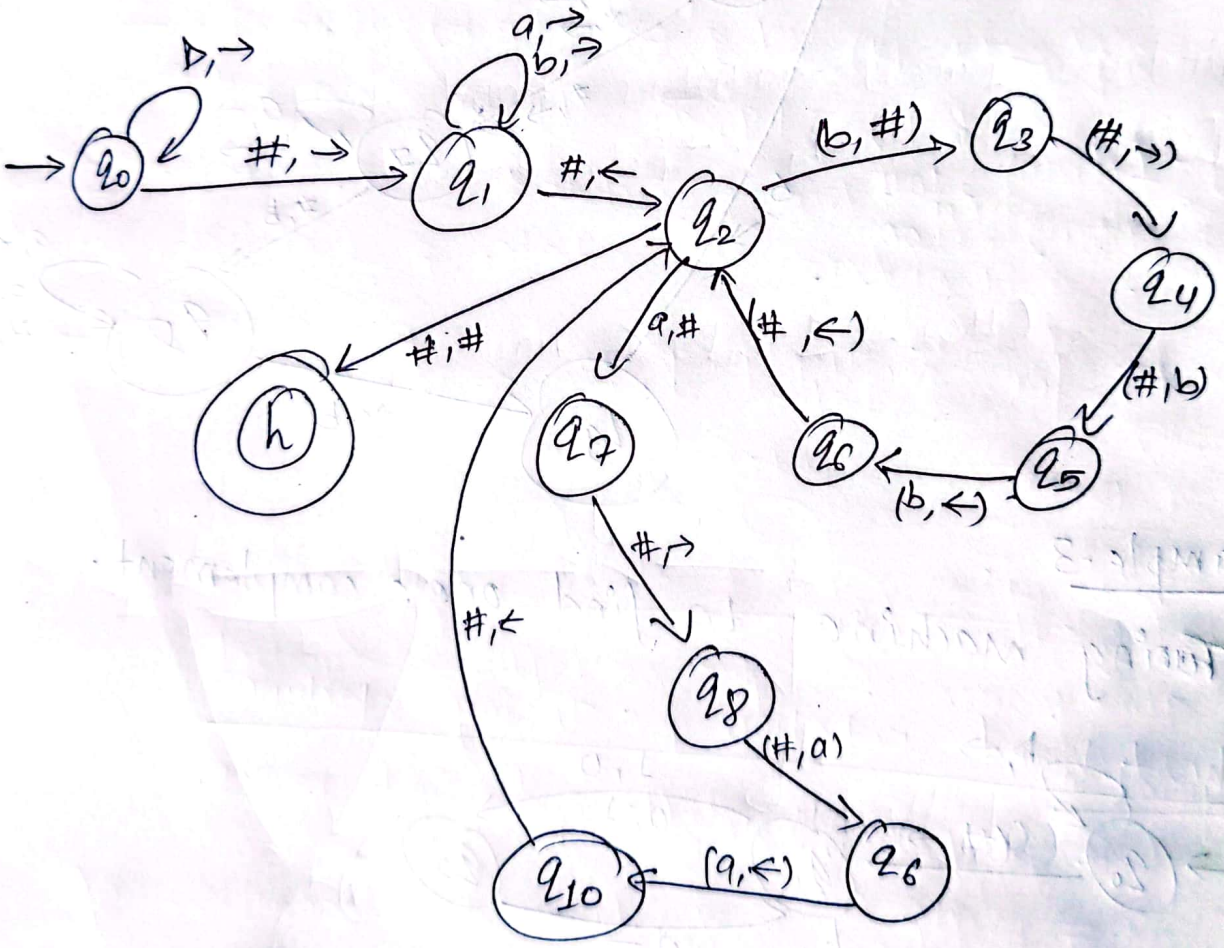
Turing machine to find one's complement.



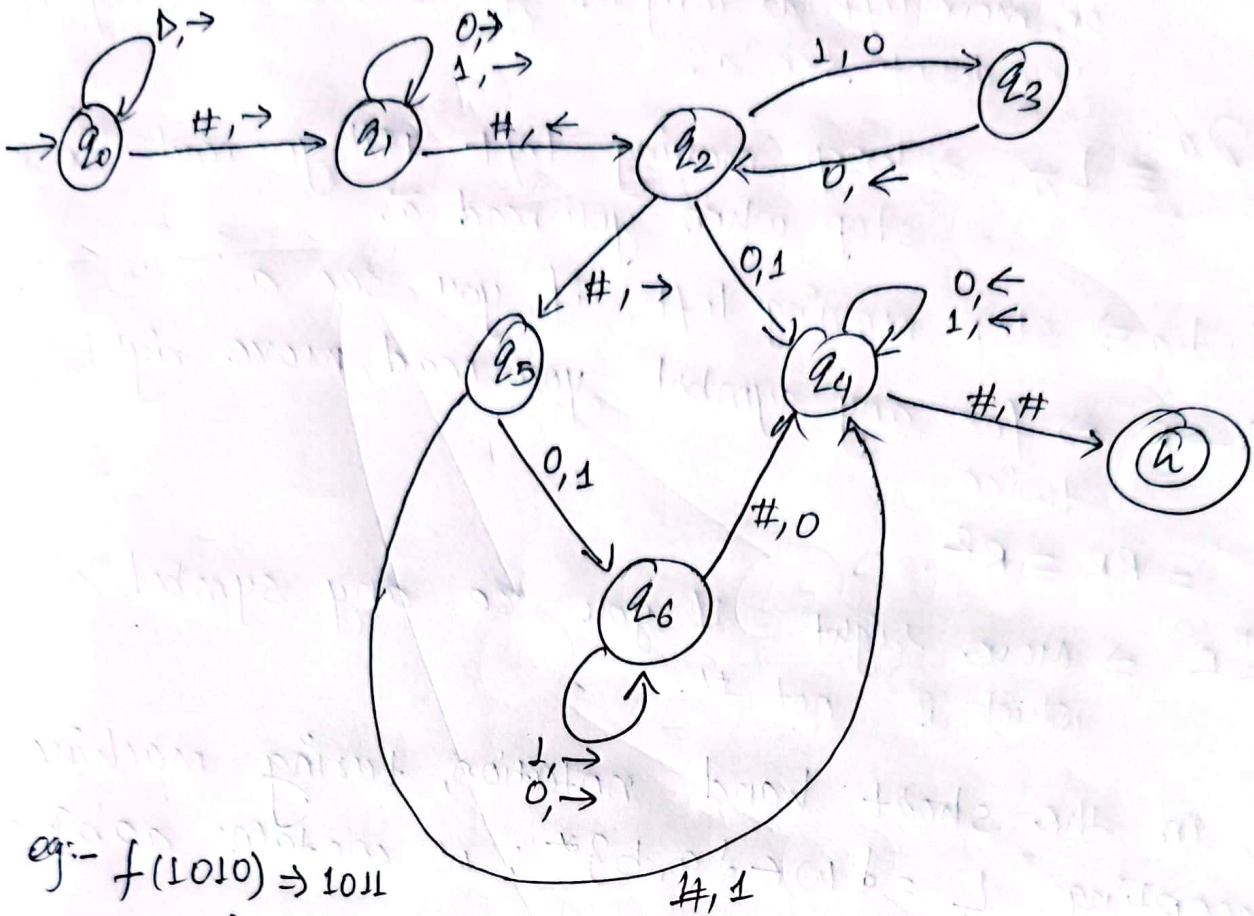
⑧ Construct a T.M that replace a by b & b by a.



Right shifting Turing Machine



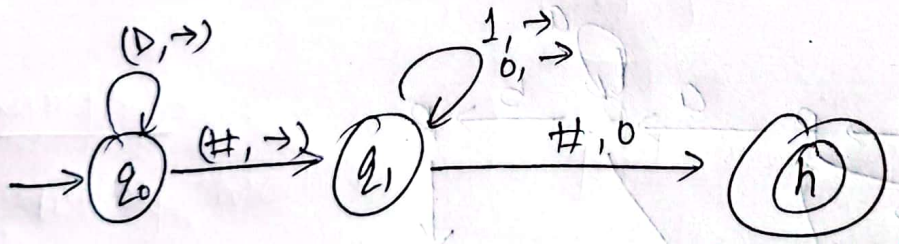
8) T.M for adding 1 to a binary number.



eg:- $f(1010) \Rightarrow 1011$

$f(1011) \Rightarrow 1100$

9) T.M. for multiplying a binary number by 2.



Short hand notation for Turing Machine:-

L, R ⇒ represents left and right moves.

a ⇒ write A

> ⇒ starting point

$\overline{a} \to R$ ⇒ Move right if you read 'a'

$\overline{\overline{a}} \to R$ ⇒ Move right if any other symbol other than 'a' is read.

L^{2^a} ⇒ keep on moving left as long as you see 'a'.
or, move left as long as you see 'a' & stop if you donot see 'a'.

$L^{2^a} \equiv L_{\bar{a}}$ ⇒ keep moving left till you read 'a' & stop when you read $\bar{a}$.

∴ L_a ⇒ stop moving left if you see a.

$R \to R$ ⇒ for any symbol you read, move right twice

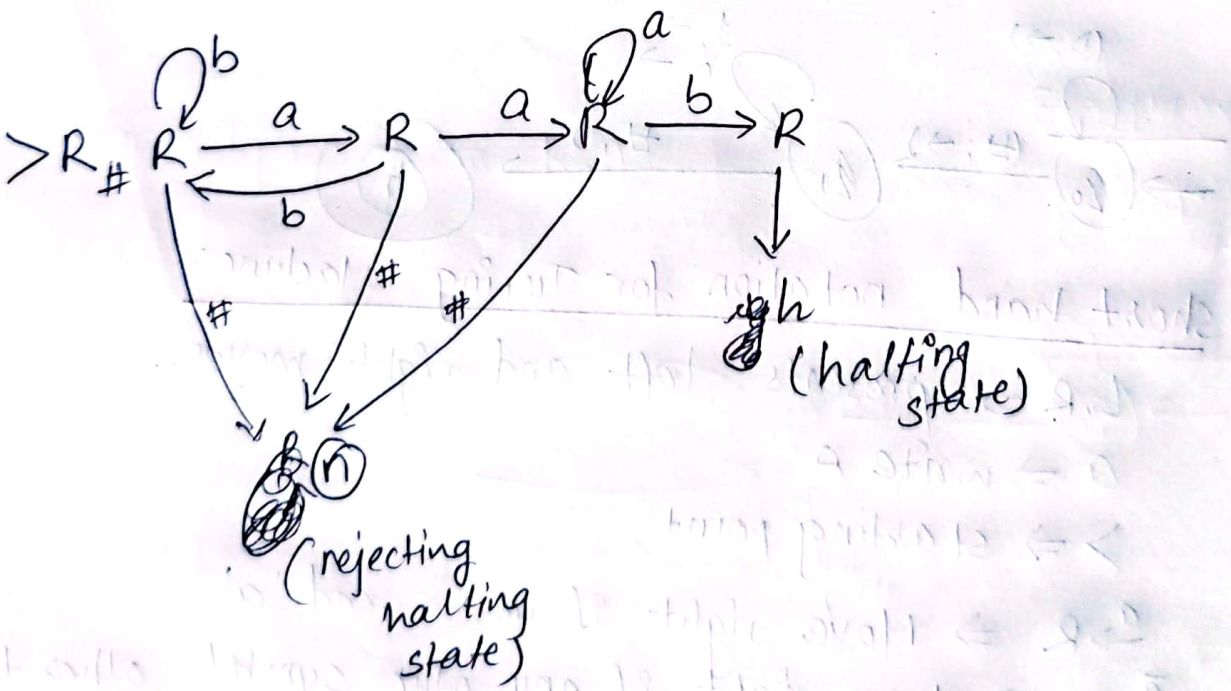
$\equiv RR \equiv R^2$.

$\xrightarrow{a\#\#} R$ ⇒ move right if you see any symbol 'a' which is not #.

So, in the short hand notation, turing machine accepting $L = \{ w \in \{a, b\}^* ;$ w contains aab$\}$ can be :-

where,

Initial configuration ⇒ $(q_0, \triangleright \# w \#)$



Recursive Language (Turing-Decidable) (Decidable) -(re)

Let, $T = (Q, \Sigma, \delta, q_0, H)$, $H = \{y, n\}$

$\Sigma_0 \subseteq \Sigma - \{\triangleright, \#\}$ be alphabet

$L \subseteq \Sigma_0^*$ be a language

Then, T semi-decides L if for any $w \in \Sigma_0^*$, $w \in L$ if and only if T halts on input w.

→ A language L is recursively enumerable if and only if there is Turing machine, T that semidecides it. (i.e, TM halt for every input, $w \in L$ but Not guaranteed to halt for $w \notin L$)

Example:-

A = CFL's that do not contain everything.

CFL, $L \neq \Sigma^*$

A = T.M's that halt's on specific input.

Not Recursively Enumerable (Not Partially decidable) (- not re)

⇒ NOT guaranteed to halt for any input.

Example:-

A = CFL's that contains everything.

∴ CFL, $L = \Sigma^*$ [$\bar{A}$ is re]

A = T.M that halt on every input [$\bar{A}$ is also not re]

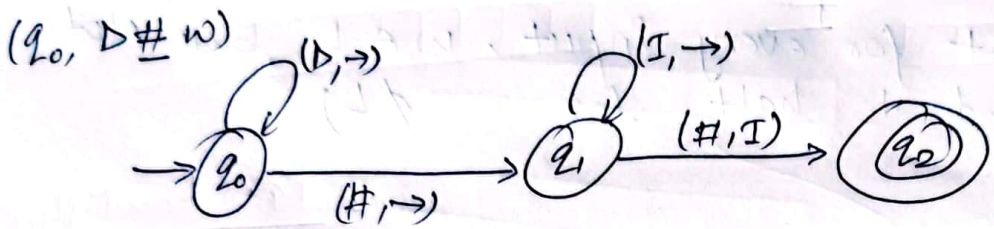
* If A is re then $\bar{A}$ is (not re)
* If A is recursive, it is also recursively enumeral.
* If A is recursive then its complement, $\bar{A}$ is also recursive.
* If A and $\bar{A}$ both are recursively enumerable, then A is recursive.

* A language is recursively enumerable if and only if it is turing enumerable (i.e, enumerator exist for that language).
* A language is recursive iff, it is lexicographically Turing enumerable.

Example:

Construct a T.M that computes $f(n) = n+1$

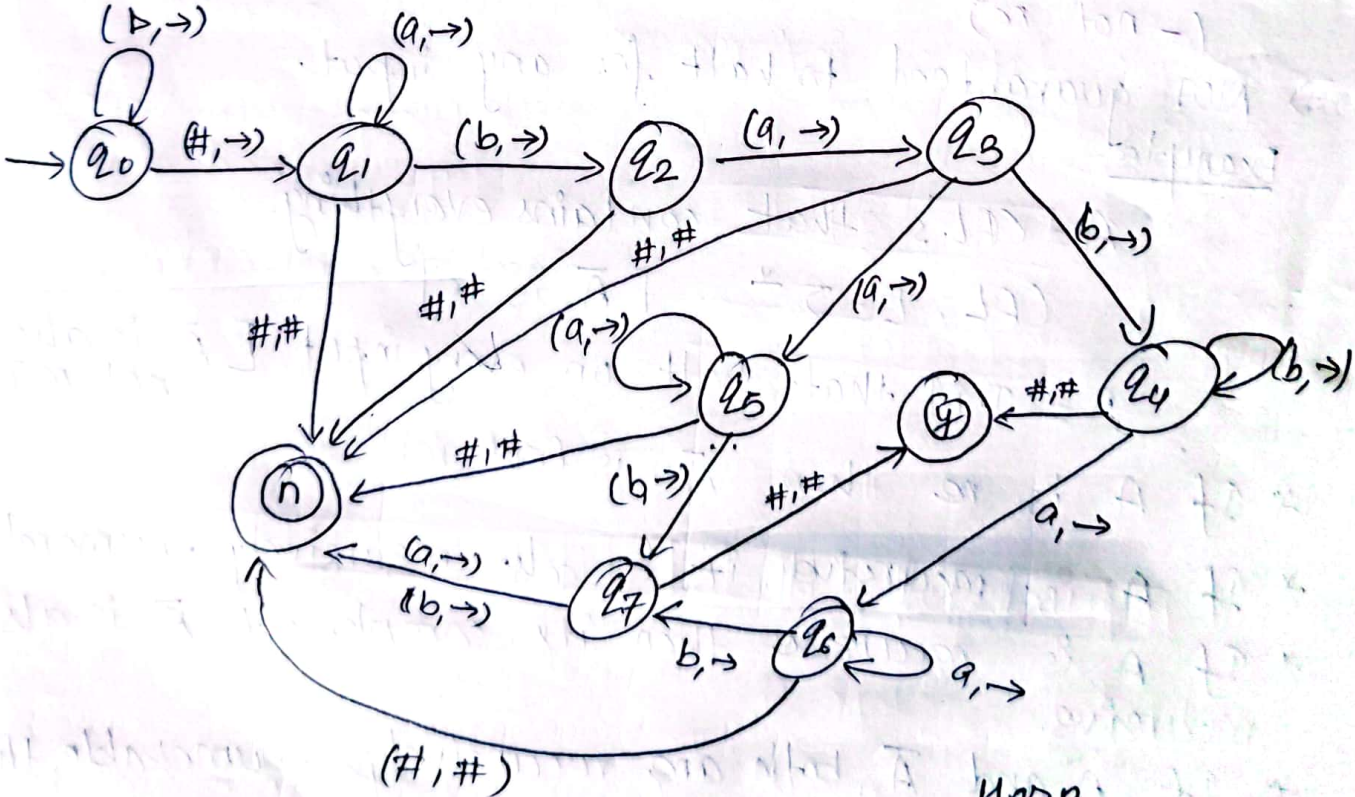
Let, $\Sigma = \{\triangleright, \#, I\}$, $\Sigma_0 = \{I\}$, $L \subseteq \Sigma_0^*$



→ Turing machine that decides, $a^*bab^*a^*b$

$\Sigma = \{\triangleright, a, b, \#\}$, $\Sigma_0 = \{a, b\}$, $w = \Sigma_0^*$, $w \in L$, $L \subseteq \Sigma_0^*$

Let starting configuration be $(q_0, \triangleright \# w)$



Here,

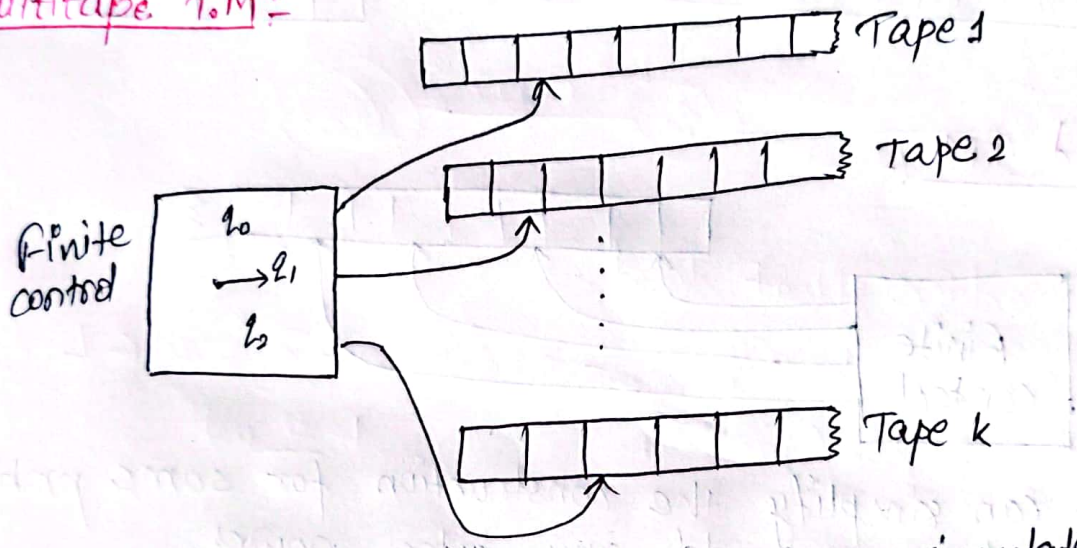
$H = \{y, n\}$

Extensions of Turing Machine:-

(i) Multitape T.M
(ii) Two-way infinite Tape T.M
(iii) Multihead T.M.
(iv) Two-dimensional Tape T.M
(v) Random Access T.M
(vi) Non-Deterministic T.M

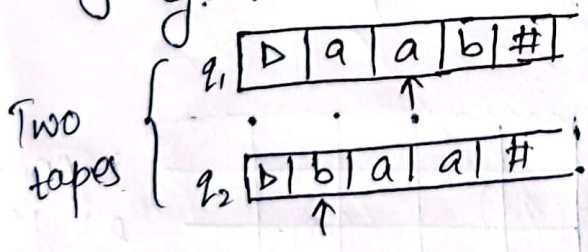
→ Two stack PDA has equal power for computation as that of one Turing Machine

Multitape T.M:-



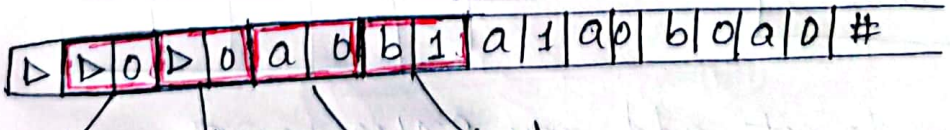
→ Multiple tape's information can be simulated in a single tape.

eg:-



* Using more than one tape instead of single tape
* Adds no extra power.

(is equivalent to)



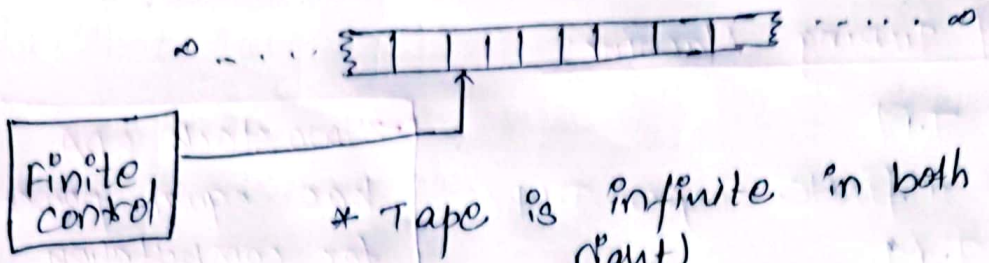
1st tape 1st symbol (0 represents scan head is not here)

2nd tape 1st symbol

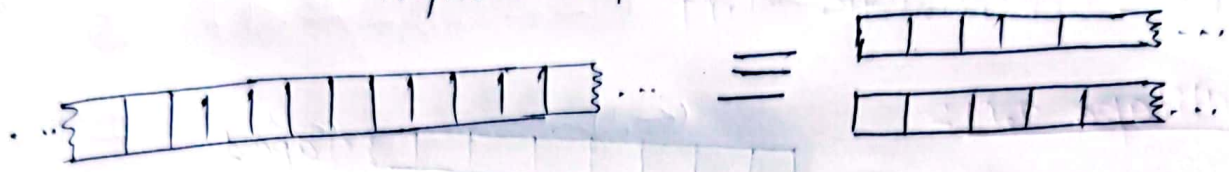
1st tape 2nd symbol

2nd tape 2nd symbol (1 represent scan head of T_2 is here)

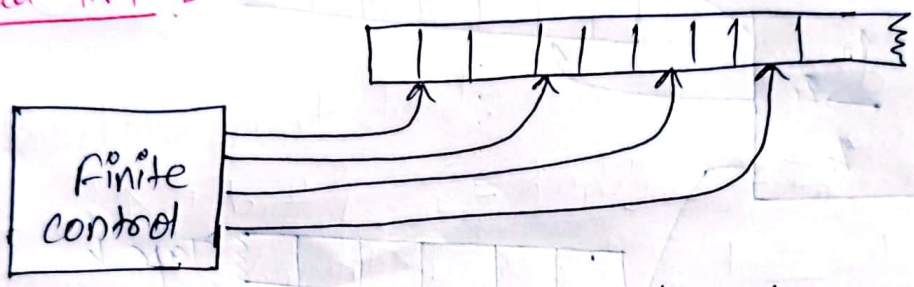
Two way infinite Tape T.M:



* Tape is infinite in both ways (left & right)
* Adds no power
* can be easily simulated by a one-way infinite Tape.



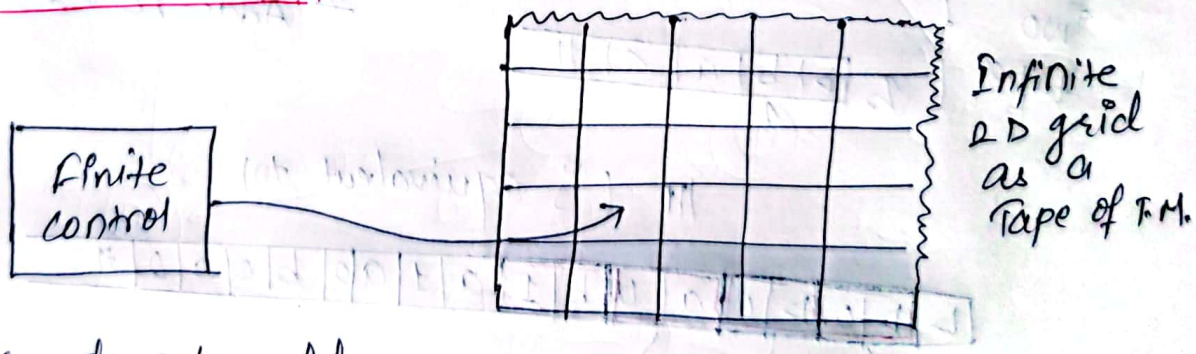
Multihead T.M :-



→ It can simplify the construction for some problems but does not add any extra power.

→ It can be simulated by a k-tape T.M. with one tape keeping track of head positions.

Two Dimensional T.M:



→ It also do not add any extra power

→ Even higher dimensional tape can be simulated by a single tape T.M.

Random Access Turing Machine:-

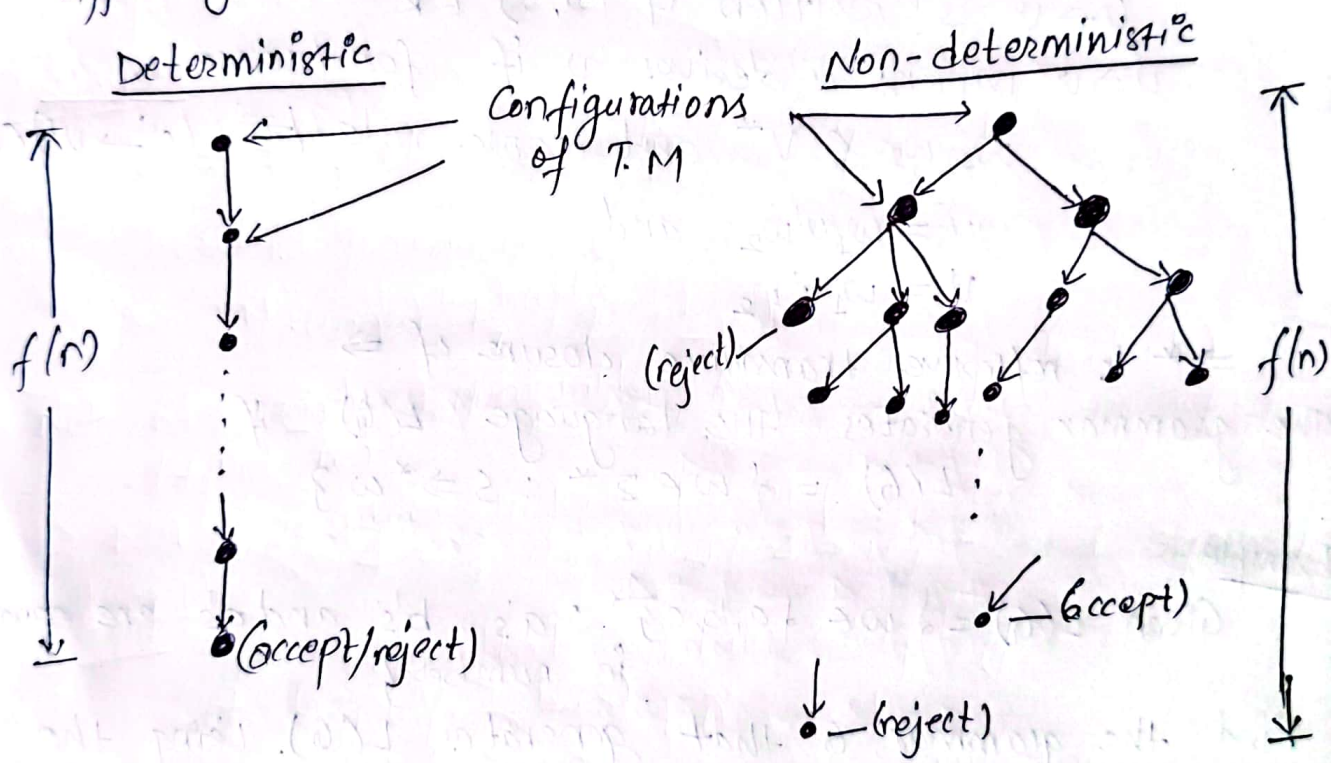
→ Taking sequential access tapes to random access memories.

→ These are also equivalent in power to the standard Turing machine, with only polynomial loss in efficiency.

Non-Deterministic Turing Machine:

→ Introducing non-determinism is to allow multiple choice on some state & scanned symbol.

→ Such machines can be ~~simultaneous~~ simulated by standard turing machines with exponential loss in efficiency.



Unrestricted Grammar:

→ It is a quad-tuple.

$$G = (V, \Sigma, R, S)$$

where,

V is set of variables.

Σ is set of terminals

$(V-\Sigma)$ is set of non-terminals

S is start variable, $S \in (V-\Sigma)$

R is set of rules

$$R \subseteq (V^* (V-\Sigma) V^*) \times V^*$$

The rule,

$u \to v$ is written if $(u, v) \in R$

$u \Rightarrow v$ means u derives v if for some, $w_1, w_2 \in V^*$ and some rule $(u' \to v') \in R$,

$u = w_1 u' w_2$ and

$v = w_1 v' w_2$

$\Rightarrow^*$ is reflexive transitive closure of $\Rightarrow$

The grammar generates the language $L(G)$ if

$$L(G) = \{ w \in \Sigma^* : S \Rightarrow^* w \}$$

Example:-

Given $L(G) = \{ w \in \{a, b, c\}^* : a's, b's \text{ and } c's \text{ are equal in number} \}$

Find the grammar, G that generates $L(G)$. Using the grammar, generate string "aabccb".

$$G = (V, \Sigma, R, S)$$

$$V = \{A, B, C, S, a, b, c\}$$

$$\Sigma = \{a, b, c\}$$

$$R = \{ \underset{①}{AB \to BA}, \underset{⑪}{AC \to CA}, \underset{③}{BC \to CB}, \underset{④}{BA \to AB}, \underset{⑤}{CA \to AC}, \underset{⑥}{CB \to BC}, \underset{⑦}{A \to a}, \underset{⑧}{B \to b}, \underset{⑨}{C \to c}, \underset{⑩}{S \to ABCS}, \underset{⑪}{S \to e} \}$$

For $w = aabccb \in L(G)$.

$S \Rightarrow ABCS \Rightarrow ABCABCS$ [using rule 10]
$\Rightarrow ABCABC$ [using rule 11]
$\Rightarrow ABCABC$ [using rule 5]
$\Rightarrow AABCBC$ [using rule 4]
$\Rightarrow AABCCB$ [using rule 3]
$\Rightarrow AABCCb$ [using rule 8]
$\Rightarrow AABCcb$ [" " 9]
$\Rightarrow AABccb$ [" " 9]
$\Rightarrow AAbccb$ [" " 8]
$\Rightarrow Aabccb$ [" " 7]
$\Rightarrow aabccb$ // [" " 7]

Theorem:-

If Turing Machine (TM) $T = (Q, \Sigma, \delta, q_0, H)$.

Then, there is grammar, G such that,

$(q, u\underline{a}v) \vdash_T (q', u\underline{b}'v')$

where,

$(q, u\underline{a}v)$ and $(q', u'\underline{b}'v')$ are configurations of T

iff

$u'q'a'v'\Delta \Rightarrow_G^* uquav\Delta$ ($\Delta \rightarrow$ n°ght end symbol)

Proof:-

Let $G = (V, \Sigma, R, S)$

$V = Q \cup \{S\} \cup \Sigma$

$R' = \{ pb \rightarrow qa$ if $\delta(q_0, a) = (p, b)$
$p, q \in Q$,
$a, b \in \Sigma$

$apb \rightarrow qab$
$ap\#\Delta \rightarrow qa\Delta$ } if $\delta(q, a) = (p, \rightarrow), b \in \Sigma$

$pba \rightarrow bqa$
$pa\#b \rightarrow aq\#b$
$pa\Delta \rightarrow aq\#\Delta$ } if $\delta(q, a) \rightarrow (p, \leftarrow), b \in \Sigma, a \neq \#$
if $\delta(q, \#) \rightarrow (p, \leftarrow), b \in \Sigma$.

$\} \subseteq R$

By construction, it can be said that

$(q, u\underline{a}v) \vdash_T (q', u'\underline{a}'v')$

iff

$u'q'a'v'\Delta \Rightarrow uqav\Delta$

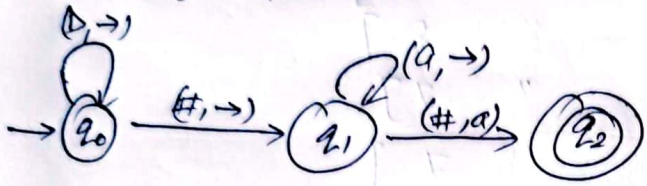
and the same will be true for $\vdash_T^*$

Example:-

$T = (Q, \Sigma, \delta, q_0, H)$

where,

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{\triangleright, \#, a\}$



$\delta = \{((q_0, \triangleright), (q_0, \rightarrow)),$
$((q_0, \#), (q_1, \rightarrow)),$
$((q_1, a), (q_1, \rightarrow)),$
$((q_1, \#), (q_2, a))\}$

Let G be the grammar,

$G = (V, \Sigma, R, S)$ such that,

$V = Q \cup \Sigma \cup \{S\}$

$R' = \{ q_2 a \rightarrow q_1 \#,$
$\triangleright q_0 a \rightarrow q_0 \triangleright a,\ \triangleright q_0 \# \rightarrow q_0 \triangleright \#,$
$\triangleright q_0 \# \Delta \rightarrow q_0 \triangleright \Delta,$
$\# q_1 a \rightarrow q_0 \# a,\ \# q_1 \# \rightarrow q_0 \# \#,$
$\# q_1 \# \Delta \rightarrow q_0 \# \Delta$
$a q_1 a \rightarrow q_1 aa,$
$a q_1 \# \Delta \rightarrow q_1 a \Delta$
$\} \subseteq R$

Configuration transition for in T for $\omega = aaa$

$(q_0, \underline{\triangleright} \# aaa) \vdash_T (q_0, \triangleright \underline{\#} aaa) \vdash_T (q_1, \triangleright \# \underline{a} aa)$
$\vdash_T (q_1, \triangleright \# a \underline{a} a) \vdash_T (q_1, \triangleright \# aa \underline{a}) \vdash_T (q_1, \triangleright \# aaa \underline{\#})$
$\vdash_T (q_2, \triangleright \# aaa a)$

So, in grammar G, $\triangleright \# aaa q_2 a \Delta \Rightarrow_G^* q_0 \triangleright \# aaa \Delta$

$\triangleright \# aaa \underline{q_2} a \Delta \Rightarrow \triangleright \# aaa \underline{q_1 \#} \Delta \Rightarrow \triangleright \# aa \underline{q_1} a \Delta \Rightarrow \triangleright \# a \underline{q_1} aa \Delta$
$\Rightarrow \triangleright \# \underline{q_1} aaa \Delta \Rightarrow \triangleright q_0 \# aaa \Delta \Rightarrow q_0 \triangleright \# aaa \Delta$ //.

Grammar that Computates:-

Definition:-

Let $G=(V,\Sigma,R,S)$ be a grammar, $\Sigma_0=\Sigma-\{b,\#\}$ and $f:\Sigma_0^*\to\Sigma_0^*$ be a function, then G computes f if and only if there are strings $x,y,x',y'\in V^*$ such that for any $u\in\Sigma_0^*$ and $v\in\Sigma_0^*$,

$$f(u)=v \text{ iff } xuy\Rightarrow x'vy'$$

Eg:- Let $f:\Sigma_0^*\to\Sigma_0^*$ such that

$$f(w)=w^R$$

$\Sigma_0=\{a,b\}$, $w\in\Sigma_0^*$

Let G be grammar that computes $f(w)$.

$G=(V,\Sigma,R,S)$, $\Sigma=\{a,b\}$

$V=\{A,B,X,Y,a,b\}$

$R'=\{Xa\to XA,\ Xb\to XB,\ Aa\to aA,\ Ba\to aB,$
$Ab\to bA,\ Bb\to bB,\ AY\to Ya,\ BY\to Yb,$
$XY\to e\}$

To compute f(ababb)

$$R=R'\cup\{S\to Xababb Y\}$$

So,

$S\Rightarrow \underline{Xa}babbY\Rightarrow X\underline{Ab}abbY\Rightarrow \underline{Xb}Aabb Y\Rightarrow XBAabbY$

$\Rightarrow XB\underline{aA}bbY\Rightarrow \underline{Xa}BAbbY\Rightarrow XA\underline{BA}bbY\Rightarrow XAB\underline{bA}bY$

$\Rightarrow XA\underline{bB}AbY\Rightarrow \underline{Xb}ABAbY\Rightarrow XBAB\underline{Ab}Y$

$\Rightarrow XBA\underline{Bb}AY\Rightarrow XB\underline{Ab}BAY\Rightarrow XB\underline{bA}BAY\Rightarrow$

$\underline{Xb}BABAY\Rightarrow XBBAB\underline{AY}\Rightarrow XBBAB\underline{Y}a$

$\Rightarrow XBBAYba\Rightarrow XBBYaba\Rightarrow XBYbaba\Rightarrow XYbbaba$

$\Rightarrow bbaba$ #.

Theorem:-

If a function from string to string or number to number is Turing computable, then it is grammatically computable.

Proof:-

Let $f: \Sigma_0^* \to \Sigma_0^*$ be Turing computable,

then,

$$T = (Q, \Sigma, \delta, q_0, \#)$$

such that for any $u \in \Sigma_0^*$,

$$f(u) = v \text{ iff } (q_0, \triangleright \# u) \vdash_T^* (q, \triangleright \# v) \; ; q \in H$$

Then, grammar G can be constructed such that,

$$G = (V, \Sigma, R, S)$$

and for, $x, x', y, y' \in V^*$

$$\underbrace{\triangleright q_0 \#}_{x} \underbrace{u}_{y} \Rightarrow_G^* \underbrace{\triangleright q \#}_{x'} \underbrace{v}_{y}$$

Here,

$$x = \triangleright q_0 \#, \; y = e$$

$$x' = \triangleright q \#, \; y' = e$$

Theorem:-

A language is Turing acceptable, if and only if some grammar can generate it.

Recursive Function Theory :-

Initial Functions for Natural Numbers (N)

i) Zero function (z), $z : N \to N$

$z(n) = 0 \quad \forall n \in N$

Result is always zero

k-ary zero function,

$z(n_1, n_2, \ldots, n_k) = 0 \; ; \; n_1, n_2 \ldots n_k \in N$

ii) k-ary projection function

$\pi_i^k (n_1, n_2, \ldots n_k) = n_i$

$1 \leq i \leq k, \; n_i \in N$

Result is one of the arguments $n_1, n_2 \ldots n_k$ $\quad \pi_1^k (n_1, n_2, \ldots n_k) = n_1$

iii) Successor Function :-

$S(n) = n + 1 \; ; \; n \in N$

→ Takes one argument (unary or 1-ary function) and returns succeeding number.

$S(5) = 5 + 1 = 6$

* Composition

for $k, l \geq 0$

$g : N^k \to N$ is k-ary function and $h_1, h_2, \ldots h_k$ are l-ary functions. Then, composition of g with $h_1, h_2, \ldots h_k$ is the l-ary function given as,

$f(n_1, n_2, \ldots n_l) = g(h_1(n_1, n_2 \ldots n_l), h_2(n_1, n_2 \ldots n_l), \ldots h_k(n_1, n_2, \ldots n_l))$

* Recursion :-

For $k \geq 0$, g is k-ary function, h is $(k+2)$-ary function.

Then, $(k+1)$-ary function 'f' defined recursively by g and h is,

$$f(n_1, n_2, \ldots n_k, 0) = g(n_1, n_2, \ldots, n_k)$$

$$f(n_1, n_2, \ldots n_k, m+1) = h(n_1, n_2, \ldots, n_k, m, f(n_1, n_2, \ldots n_k, m))$$

$\forall m, n_i \in N.$

* **Primitive Recursive function:-**

Primitive recursive function are obtained by application of composition and recursion on initial function.

* ~~Minim Minimalizable~~

* **Minimalizable function:-**

Let g be $(k+1)$-ary function, then for $k \geq 0$, g is minimalizable if there is a k-ary function f such that,

$$f(n_1, n_2, \ldots, n_k) = \begin{cases} \text{least } m \text{ such that, } g(n_1, \ldots n_k, m) = 1, \\ \quad \text{if such an } m \text{ exists;} \\ 0 \quad \text{otherwise.} \end{cases}$$

* **μ-Recursive function:-**

μ-recursive function can be formed from initial functions by operation of composition, recursion and minimalization of minimalizable ~~operatir~~ functions.

Theorem:-

Every μ-recursive function from string to string or number to numbers is Turing computable.

Examples:-

* $z(3) = 0$, $z(1,2,3,4,5) = 0 \rightarrow$ zero function
* $\pi_2^5(1,3,5,8,9) = 3 \rightarrow$ 5-ary projection function
* $f(n) = n+2 = S(S(n))$ $\quad S(n) \Rightarrow$ successor function

* function that adds m and n ; $m, n \in N$

$plus(m, 0) = \pi_1^1(m)$

$plus(m, n+1) = S(plus(m, n))$

$\log_m n = \log(m, n) = \mu(m, n, p)$

$= \text{least } p \text{ such that } m^p \geq n \text{ is true.}$

Godel Numbering (Godelization)

→ Encoding techniques which encodes string as a number.

→ Primitive recursive functions takes natural numbers as arguments and produce natural number as output.

→ Godelization technique extends these function to strings also.

→ Converts strings to unique number.

Let, $\Sigma = \{a_1, a_2, a_3, \ldots a_{r-1}\}$ be any alphabet, then,

any string $w \in \Sigma^*$ can be represented by integer in radix 'r'. Define function gnum from Σ^* to N as:-

if $w = a_{i_1}, a_{i_2}, \ldots a_{i_k}$ where, $k \geq 0$ and

$1 \leq ij \leq r-1$ for $j = 1, 2, \ldots k$

then,

$$gnum(w) = r^{k-1} \cdot i_1 + r^{k-2} \cdot i_2 + \ldots + r^1 i_{k-1} + i_k$$

$$gnum(e) = 0$$

gnum(w) represents the godel numbering of string w. This function converts each string into a unique integer & each integer corresponds to at most one string.

Example:-

Let $\Sigma = \{a, b, c\}$

→ rewrite Σ as $\{a_1, a_2, a_3\}$

Here,

$a_1 = a$, $a_2 = b$, $a_3 = c$ and $r = 4$

Now, godel number for strings e, a, ab, aab

gnum(e) = 0

gnum(a) = 1

gnum(ab) = $4 \cdot 1 + 2 = 6$

gnum(aab) = $4^2 \cdot 1 + 4^1 \cdot 1 + 4^0 \cdot 2$

$= 16 + 4 + 2 = 22$

gnum(abc) = $4^2 \cdot 1 + 4^1 \cdot 2 + 4^0 \cdot 3$

$= 16 + 8 + 3$

$= 27$

gnum(abaa) = $4^3 \cdot 1 + 4^2 \cdot 2 + 4^1 \cdot 1 + 4^0 \cdot 1$

$= 64 + 32 + 4 + 1$

$= 101$ //