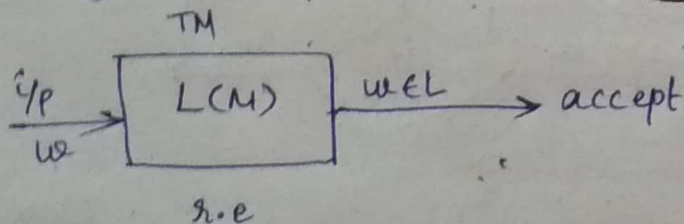
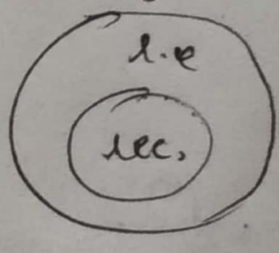
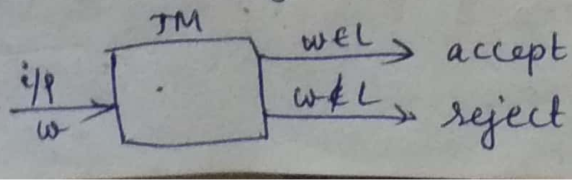
MOD - 6

* Rec. & r.e languages :
1) ~~Recursively enumerable / Type 0 lang.~~ : → generated by Type 0 gram
~~A lang.~~ L is said to be r.e if ∃ a TM that accept it,
i.e, ∃ a TM 'M' s.t. ∀ $w \in L$, $q_0 w \vdash_M^* x_1 q_f x_2$
with q_f as the final state,
x_1, x_2 → strings after processing w.
This definition says nothing abt. what happens for $w \notin L$. It may ~~happen~~ be that m/c halts in a non-final state / it never halts and goes to an infinite loop. → Turing recognizable lang.





2) Recursive lang. : → Turing decidable lang.
~~A~~ lang. L is said to be recursive if for some TM 'M' s.t.,
• if $w \in L$ then M accept w and halt.
• if $w \notin L$ then M reject w and halt on a non-final state.



(L is decided by a TM).

* Closure properties of recursive and r.e languages.

(1) The union of 2 recursive languages are recursive. (recursive language is closed under union).

Let L_1 be a recursive lang. accepted by M_1 & L_2 be the recursive lang. accepted by M_2.

Construct ~~Consider~~ a new TM that accepts $L_1 \cup L_2$, say M.

If M_1 accepts $w \Rightarrow$ M will accept.

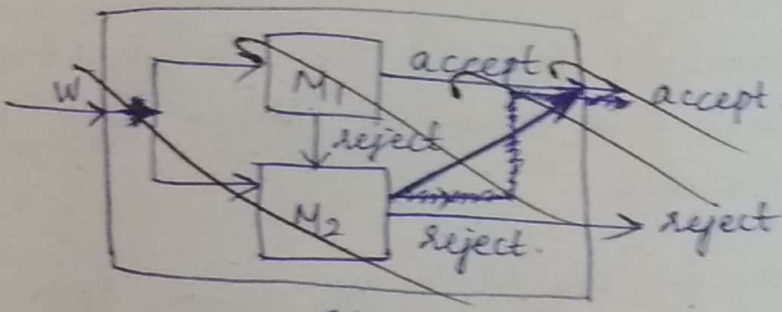
" " reject $w \Rightarrow$ M simulates M_2 and accept if M_2 accepts w.

If both M_1 and M_2 reject $w \Rightarrow$ M will reject w.

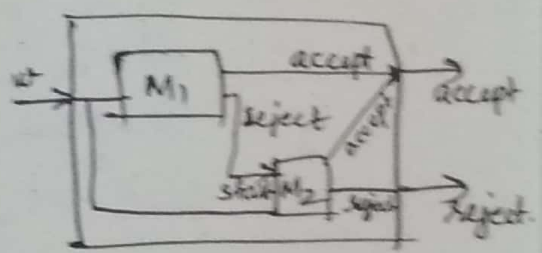
$\Rightarrow$ M is guaranteed to halt on all i/ps

$\Rightarrow$ language accepted by M is recursive

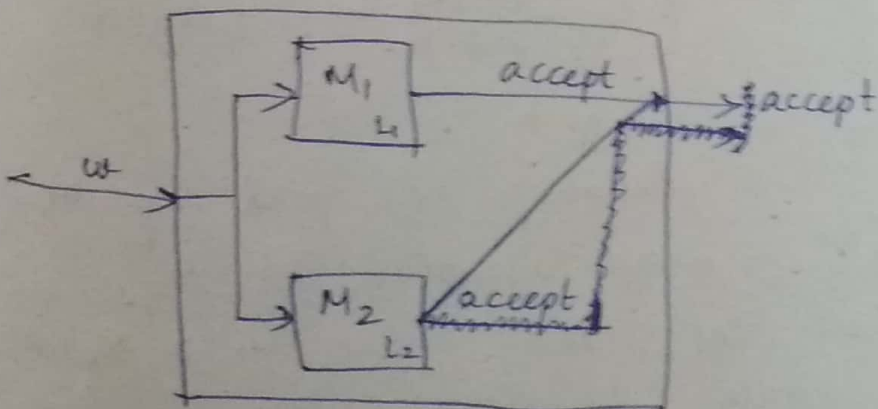
$\Rightarrow$ $L_1 \cup L_2$ is recursive.



M



(2) The union of 2 r.e languages is recursively enumerable. (r.e lang. is closed under union) (r-e)



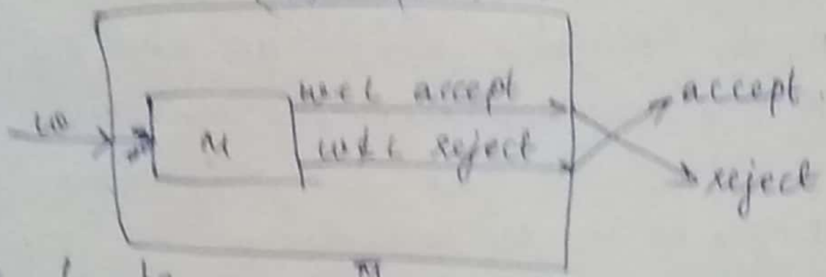
M

Let L_1 be a recursively enumerable language that is accept by M_1, ie $L(M_1) = L_1$; and L_2 be a r.e lang. accepted by M_2, ie $L(M_2) = L_2$.

We construct a new TM 'M' & it accept the language $L_1 \cup L_2$. $\Rightarrow L(M) = L_1 \cup L_2 \Rightarrow L_1 \cup L_2$ is r.e.

(3) If L is recursive then complement of L, i.e. $\bar{L}$ is also recursive (recursive lang. is closed under complementation)

$(L(\bar{M}) = \bar{L})$



Let L be a recursive language accepted by TM, 'M' and $\bar{L}$ is complement of L accepted by '$\bar{M}$'.

TM modifies as follows to create $\bar{M}$.

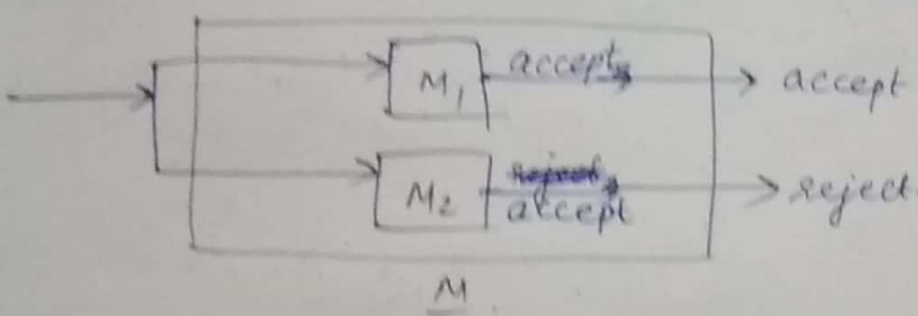
(i) accepting states of M are made non-accepting " of $\bar{M}$

(ii) all non-accepting states of M are made accepting states of $\bar{M}$.

$\because$ M is guaranteed to halt on all i/p's for w

$\Rightarrow$ $\bar{M}$ will also halt $\Rightarrow$ $\bar{L}$ is also a recursive language.

(4) If both L and $\bar{L}$ are r.e $\Rightarrow$ L is recursive.



Let L be a r.e language accepted by M_1.

$\Rightarrow$ $\bar{L}$ is the r.e language accepted by M_2.

Create a new TM, 'M' using M_1 & M_2 as shown in figure. $\therefore$ from figure it's clear that M halts on all i/ps of w.

$\Rightarrow$ $L(M) = L$ and L is recursive.

(5) Concatenation : If L_1 and L_2 are recursive $\Rightarrow$ $L_1 \cdot L_2$ is also recursive

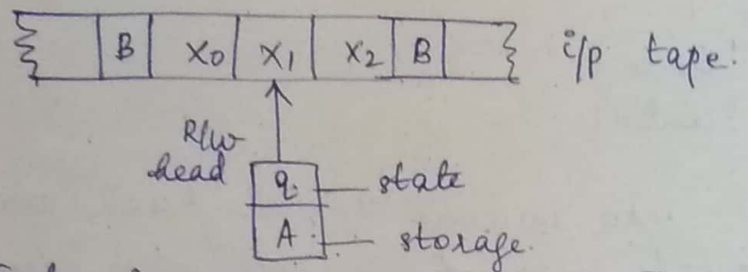
(6) Kleene Closure : L_1 is rec. $\Rightarrow$ L_1^* is also rec.

(7) Intersection : L_1 & L_2 are rec. $\rightarrow$ $L_1 \cap L_2$ is also rec.

* Programming techniques in TM.

1. Storage at state / finite control :

In this type of TM, states in the control unit not only represent current position of the pgm. but also holds a finite amt. of data.



A state in TM is a tuple [q, A] where q is a state and A is the stored value at state q.

The transition fn. 'δ' can be viewed as

$$\delta(\underbrace{[q, A]}_{\text{curr. state + stored value}}, \underbrace{X}_{i/p}) = (\underbrace{[p, B]}_{\text{next state + stored value}}, \underbrace{Y}_{\text{replacing symbol}}, \underbrace{D}_{\text{direction}})$$

Q. Design a TM that accepts strings generated by the lang. $10^* + 01^*$ using storage at the state concept.

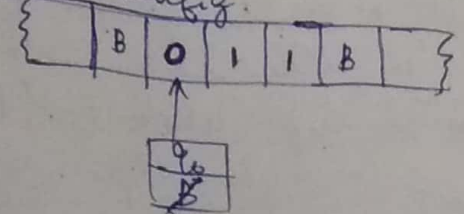
Let $L(M) = \{10^* + 01^*\}$

(i) state $q_0 \Rightarrow$ M has not read its 1st symbol yet

(ii) state $q_1 \Rightarrow$ " " read the 1st symbol and is checking that it doesn't appear elsewhere.

(iii) ~~state~~ storage at state q_1 remembers the first symbol read.

Initial config.



$[q_2, B] \rightarrow$ accepting state.

$\delta([q_0, B], 0) = ([q_1, 0], 0, R)$

$\delta([q_1, 0], 1) = ([q_1, 0], 1, R)$

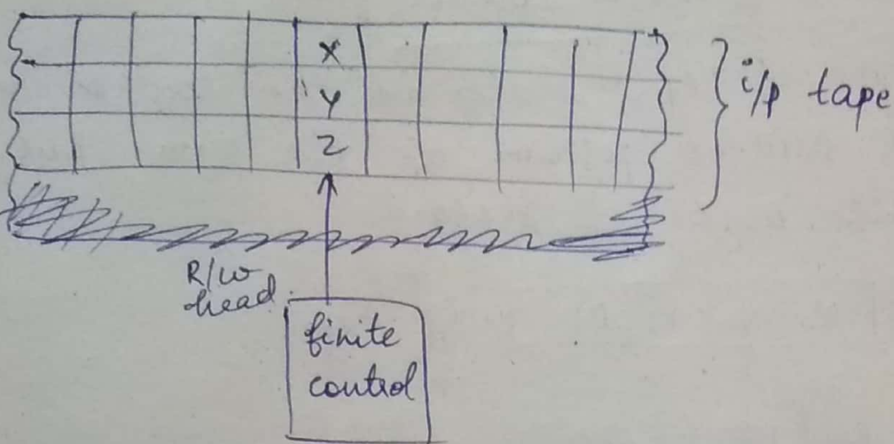
$\delta([q_1, 0], B) = ([q_2, B], B, R)$

$\delta([q_0, B], 1) = ([q_1, 1], 1, R)$

$\delta([q_1, 1], 0) = ([q_1, 1], 0, R)$

$\delta([q_1, 1], B) = ([q_2, B], B, R)$.

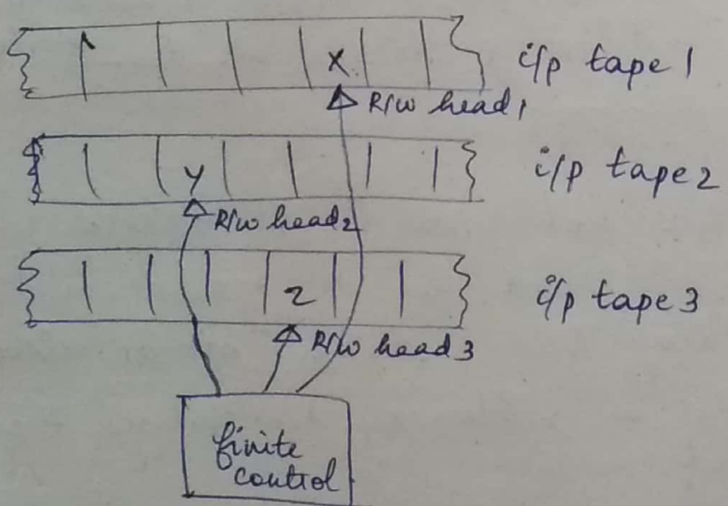
2. TM with multiple tracks on single tape:



i/p tape of TM is divided into several track. Each track is divided into several cells. Each cell can hold one symbol. The tape alphabet of TM consists of tuples with 1 component for each track.

$$\underbrace{\delta(q}_{\text{curr. state}}, \underbrace{[x, y, z]}_{\text{i/p symbols}}) = (\underbrace{p}_{\text{next}}, \underbrace{[A, B, C]}_{\text{replacing symbols}}, \underbrace{D}_{\text{direc}^n\text{ of head movt. (R/L)}})$$

3. Multi-tape TM:



This TM has finite control unit & multiple no. of i/p tapes. Each tape is divided into cells & each cell can hold 1 symbol of the finite tape alphabet(Γ).

Transition fn., 'δ' depends on:

(i) current state

(ii) symbol scanned by each tape head

In a 3-tape TM, δ can be written as,

$$\delta(q, [x, y, z]) = (p, [A, D_1], [B, D_2], [C, D_3])$$

Scanned by CamScanner

q – curr. state ; [x, y, z] – i/p symb. scanned by R/W head 1, 2, 3 resply.

p – next state ; D_1, D_2, D_3 – direcⁿ of head movts. in tape 1, 2, 3 resply.

A, B, C – replacing symb. in tape 1, 2, 3 resply.

In one move, a multi-tape TM does the following:

(i) Control enters into new state (p).

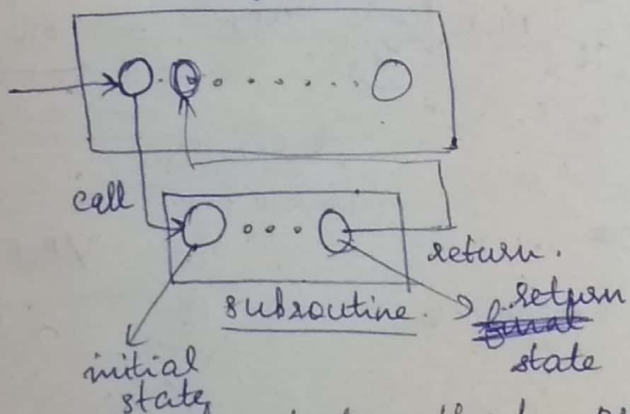
(ii) On each tape, a new symb. is written on the cell being scanned.

(iii) Each R/W head can move either R/L.

---

* <u>Subroutine</u> :

<u>Calling pgm.</u>



A subroutine is a <u>set</u> of states that perform some unique fn. This set includes a start state & a return state.

<u>Call</u> occurs whenever there is a transition to the initial state of the subroutine.

Q. Design a TM that performs multiplication.

i/p : strings with $0^m 1 0^n 1$ on i/p tape.

o/p : computation ends with $0^{m \times n}$ on the tape.

<u>Ans</u>: Assume i/p : $0^4 1 0^3$

<u>initial config</u> :

{... | B | B | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | B | B ...}

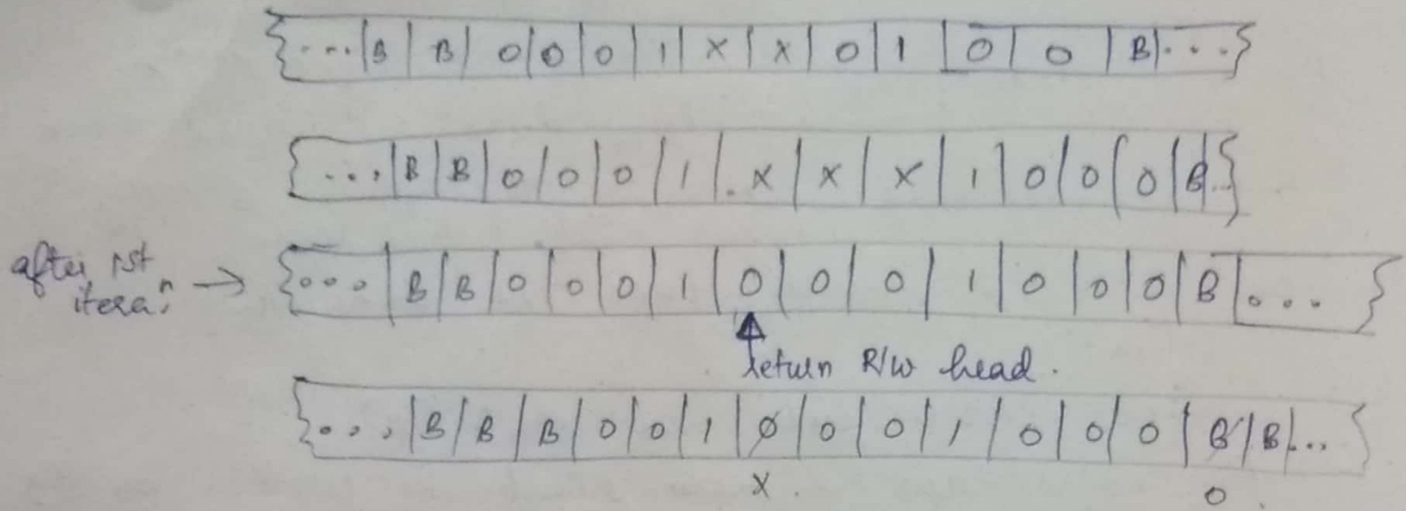
q_0 (under the first 0)

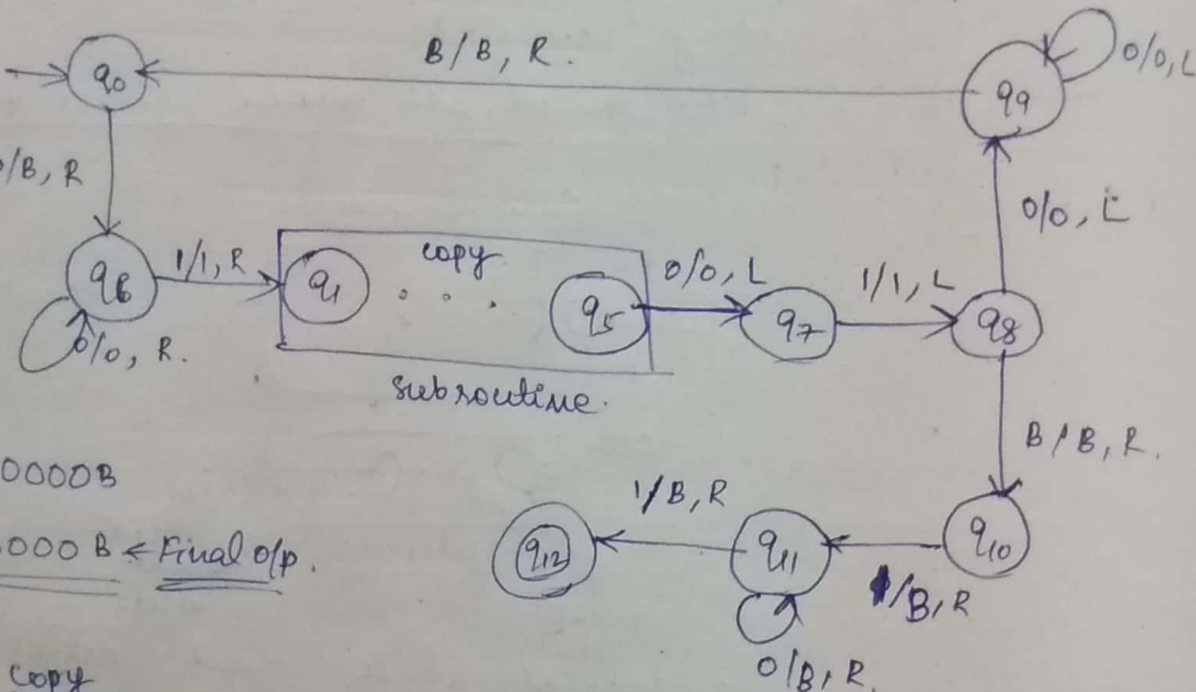
{... | B | B | B | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | B | B | ...}

call subroutine.

{... | B | B | B | 0 | 0 | 0 | 1 | x | 0 | 0 | 1 | 0 | B | ...} → in copy subroutine.

Scanned by CamScanner

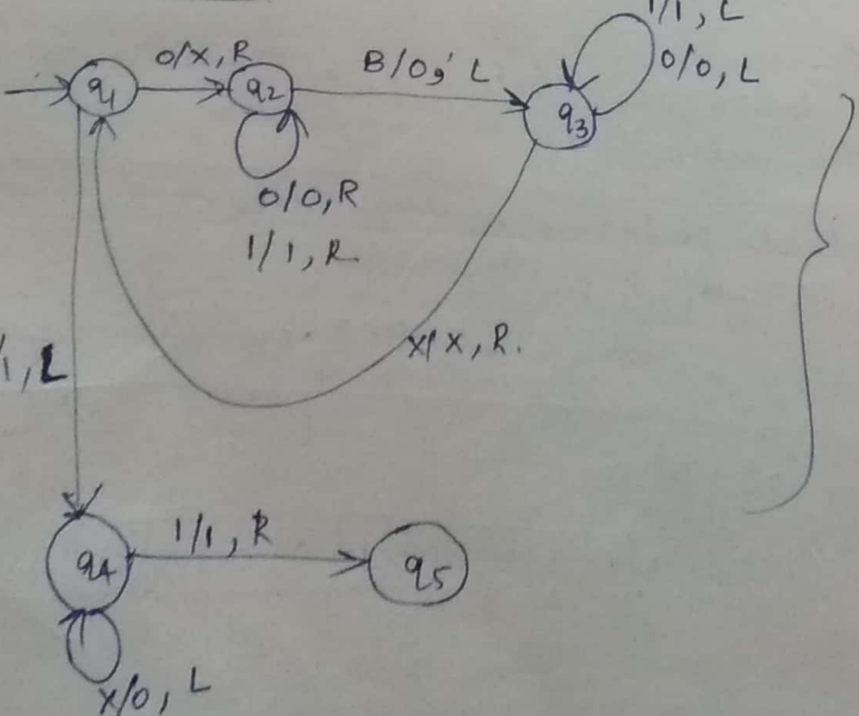




B 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 B

BBBBBB 0 0 0 0 0 0 0 0 0 0 0 0 B ← Final o/p.

Copy subroutine



copies the block of n zeros to the right end of i/p tape.

* Enumerating TM. :

A TM can be identified by binary strings.

A TM 'M' = $(Q, \Sigma, \Gamma, \delta, B, q_0, F)$ can be represented as a binary strings.

For that we must assign diff. integers to the state, tape symbols & direction.

- Each 'δ' ~~can be~~ in an enumerated TM can be represented as

$$\delta(q_i, X_j) = (q_k, X_l, D_m)$$

then code for each fn. is

$$\underbrace{0^i}_{q_i} 1 \underbrace{0^j}_{X_j} 1 \underbrace{0^k}_{q_k} 1 \underbrace{0^l}_{X_l} 1 \underbrace{0^m}_{D_m}.$$

The complete code ~~consists~~ for TM 'M' consists of set of quintuples (/transition code) separated by pair of ones.

$$C_1 \| C_2 \| C_3 \| \ldots \| C_n.$$

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

eg: $M = (Q, \Sigma, \Gamma, \delta, B, q_0, F)$

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

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

$\Gamma = \{0, 1, B\}$

$\delta(q_1, \underline{1}) = (q_3, \underline{0}, R)$

$\delta(q_3, \underline{0}) = (q_1, \underline{1}, R)$

$\delta(q_3, \underline{1}) = (q_2, \underline{0}, R)$

State

Enumerating symbols	Tape symbols	Direc^n.
$q_1 = 0$	$0 = 0$	$L = 0$
$q_2 = 00$	$1 = 00$	$R = 00$.
$q_3 = 000$	$B = 000$	

$C_1 = 0100100010100.$

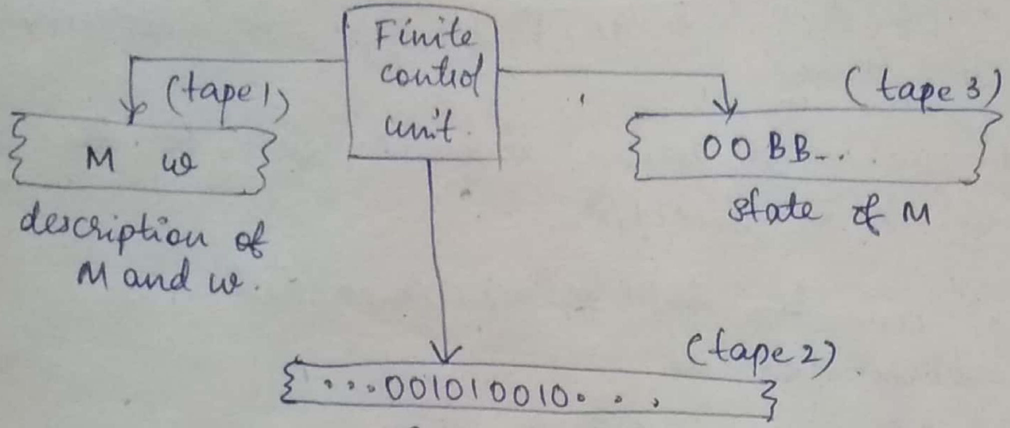
$C_2 = 0001010010100$

$C_3 = 00010010010100.$

TM, $M = C_1 \| C_2 \| C_3$

$\Rightarrow M = 01001000101001100010100100100110001001001 0100.$

* Universal TM (UTM): (M_U).



Tape symbols. (Fig—1)

→ TM is a special purpose computer. Once 'δ' is defined the machine is restricted to carry out one particular type of computation.

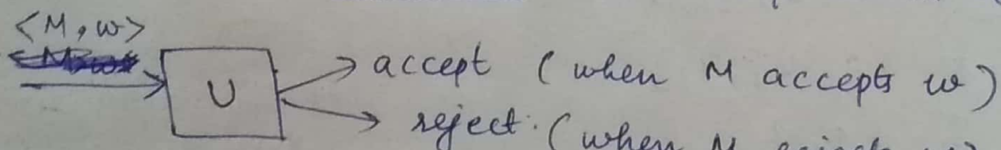
A reprogrammable TM is called universal TM (U).

→ UTM is an automation that, given as i/p to U,

(i) the description of any TM 'M'

(ii) a string 'w'.

⇒ UTM can simulate the computation of M on w.



* Organization of a UTM:

→ UTM is a multitape TM.

In Fig 1, UTM is a 3-tape TM.

(i) Tape 1 holds description of M and i/p w.

(ii) Tape 2 holds the simulated tape of M using the same format as the code of M.

(iii) Tape 3 holds the current state of M in which state q_i is represented by i no. of 0's.

* Operations of UTM :

Step 1 : Check if code for 'M' is valid for some TM 'M'. If not, halt without accepting.

Step 2 : Initialize Tape-2 to contain the i/p w in its coded form.

w = 011 ~~et~~ eg: ~~if w = 01B~~

Tape-2

let $\Sigma_1 = \{0, 1\}$.

{B|0|1|00|1|00|B|...}

i/p	0	1	B
coded format	0	00	000

Step 3 : Place start state of 'M' in Tape-3. Move the head of ~~UTM's~~ Tape-2 to the 1st simulated cell.

Step 4 : To simulate a move of M, the UTM searches on its Tape1 for a transition $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ s.t.

0^i is the state on Tape-3 (current state)

0^j is the tape symbol of M that begins at the position of R/w head on Tape-2.

The transition is :

(i) change content of Tape-3 to 0^k

(ii) replace 0^j with 0^l on Tape-2

(iii) Move R/w head on Tape-2 to the L/R depending on value of 0^m.

Step 5 : If M has no transition that matches the simulated state ~~symbol on Tape-3~~, and tape symbol in Step 4, then M halts in the simulated configuration of M.

Step 6 : If M enters accepting state ⇒ M_U accepts.

* Non-deterministic TM (NDTM)

An NDTM differs from a DTM by having a transition fn. δ s.t. for each state 'q' & tape symbol 'x', $\delta(q, x)$ is a set of triples.

$$\delta(q, x) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \dots (q_k, Y_k, D_k)\}$$

k → any +ve integer.

The computation of a NDTM is a tree of configurations that can be reached from the start ".

An NDTM can either –

(i) accept : if one or more node of the tree is in an accept configuration.

(ii) reject ; if ~~some~~ for some i/p, all branches reject ⇒ i/p is rejected.

(iii) decider : if all branches of the computation tree halt on all i/ps, then the NDA is called a decider.

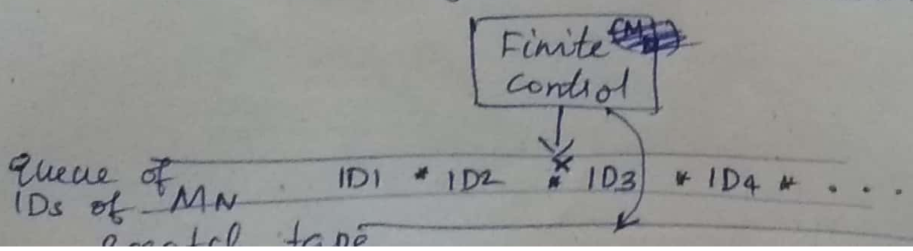
$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

$\delta : Q \times \Gamma \longrightarrow P(Q \times X \times \{L/R\})$

▷ Equivalence of NDTM and DTM ;

If M_N is an NDTM then ∃ a DTM M_D s.t. $L(M_N) = L(M_D)$

M_D is designed as a multitape TM.



The 1st tape of M_D holds sequence of ID's of M_N, including the state of M_N.

One ID of M_N is marked as current ID, whose successor ID's are in the process of being discovered. ~~At~~ and all ID's to the left of current ID have been explored & can be ignored subsequently.

To process current ID, M_D does the following.

1. M_D examines the state & scanned symbol of the current ID. Built into the finite control of M_D is the knowledge of what choices of move M_N has for each state & symbol.

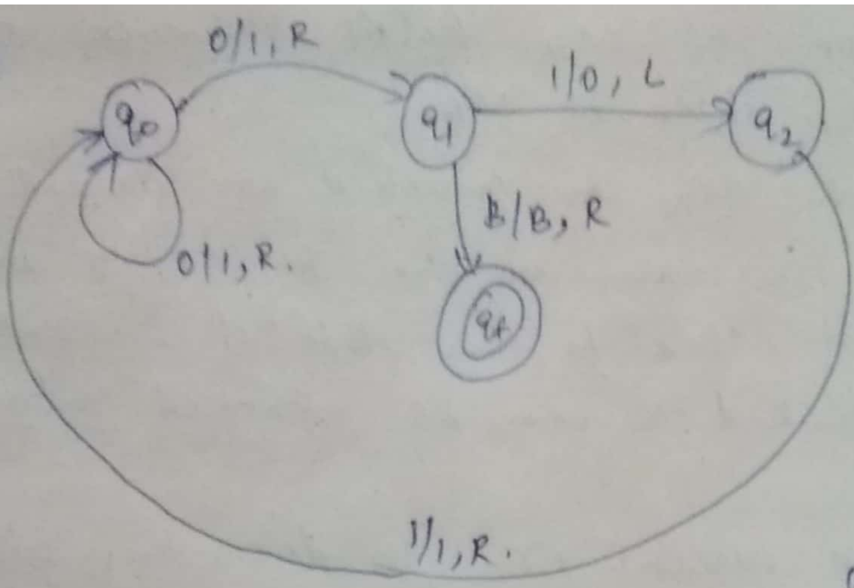
   If the state in current ID is accepting, then M_D accepts & simulates M_N no further.

2. If state isn't accepting, the state-symbol combination has k moves, then M_D uses its 2nd Tape to copy the ID and then makes k copies of that ID at the end of the sequence of IDs on Tape 1.

3. M_D modifies each of those k ID's according to a diff. one of the k choices of moves that M_N has from it current ID.

4. M_D returns to the marked, current ID, erases the mark, & moves the mark to the next ID to the right. The cycle then repeats with step 1.

eg NDTM



~~$L = (01)^* 0^*$~~

$L = 00^*(01)^*$

strings having atmost 1 ~~seq~~ 1

$0^*(01)^* 0^*$

00011 w = 0001100B

1110 1

~~000~~1~~00~~ q_f

0001100.

~~000~~1~~0~~1~~0~~1~~0~~1 01 B

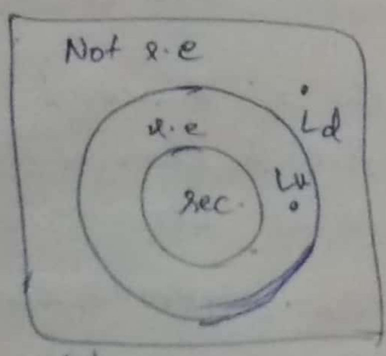
★ Decidability

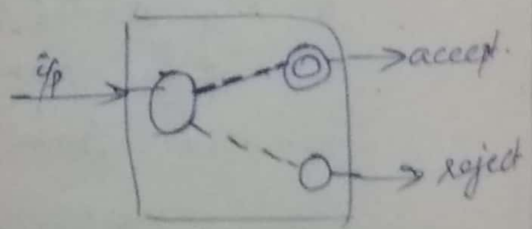
A lang. L is decidable if ∃ a TM which accepts it, and it halts on every i/p string w.

(i) Every decidable lang. is recursive.

(ii) " " " " turing acceptable.

Recursive lang. thus corresponds to an algorithm, a well defined sequence of steps to solve a prob.





decidable lang.

eg:- 1) Is a no. i/p prime.

2) Given a regular lang. L & string w. , check if $w \in L$.

3) Does a DFA accept the empty language

* Undecidable lang. → not recursive lang:

→ A decision prob. 'P' is called undecidable if lang. L of all 'Yes' instance to 'P' is not decidable.

→ Undecidable lang. can be r.e.

Recussively enumerable lang. thus corresponds to procedure

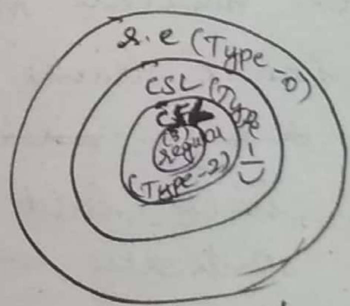
eg: 1) Halting prob.
2) Post Correspondence prob. (PCP)
3) Clique prob.
4) Vertex Cover prob.
5) 3 CNF Prob.

Type-3 grammar:

- generates regular lang.
- linear languages / grammar.
- pdt^n format $X \to a$
  $X \to aX$ where $X \in V$ and $a \in T$

eg: $G = (V, T, S, P)$ be a regular grammar then $P = \begin{cases} S \to a \\ S \to aS \end{cases}$

- accepted by FSA.



Chomsky Hierarchy

Type-2 grammar:

- generates CFL
- accepted PDA.
- CFG.
- pdt^n format. $V \to (V \cup T)^*$

V – non-terminal
T – terminal.

eg: $L = a^n b^n$ where $n \geq 1$

$G = \{V, T, S, P\}$ $P = \begin{cases} S \to aSb \\ S \to ab. \end{cases}$

Type-1 grammar:

- generate CSG
- accepted by LBA

$\alpha A \beta \to \alpha \gamma \beta$
$A \in V,\ \alpha, \beta, \gamma \in (V \cup T)^*$.

A string $\alpha Y \beta$ may be ϵ but A can't be empty.

The rule $S \to \epsilon$ is allowed if S doesn't appear on the RHS of any rule.

eg: $G = (V, T, S, P)$ $P = \begin{cases} S \to a\underline{B}aC \\ Ba \to cDcD. \end{cases}$

Type-0 grammar:

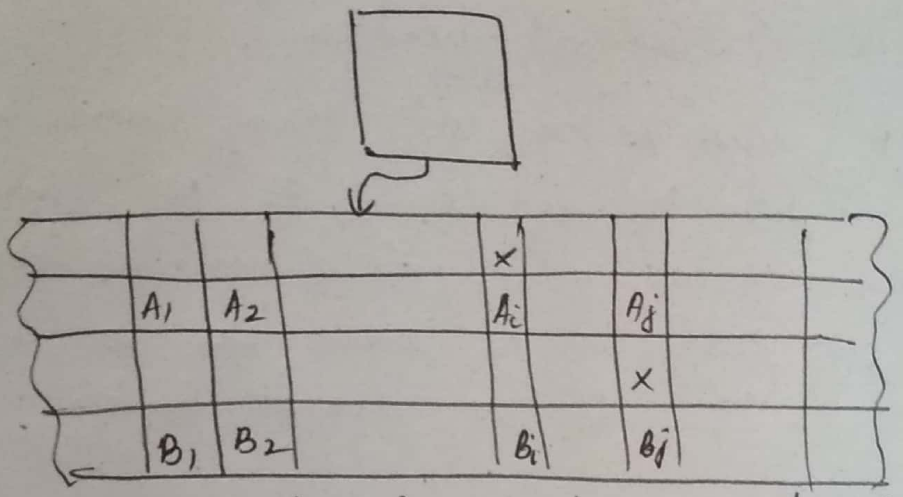
- generates r.e
- accepted by TM.
- prdtn. format. $\alpha \to \beta$. $\alpha, \beta \in (V \cup T)^*$ where α contains atleast one variable.

Q. Normal single tape can perform computations performed by multitape TM's.

Proof: Suppose L is accepted by a k-tape TM (say 'M'). We simulate M with a one-tape TM 'N' whose tape has k-tracks. Half of these tracks hold ~~only~~ ~~a single marker~~. the tapes of M, the other half hold ~~only~~ a single marker that indicates where the head for corresponding tape M is currently located.

Assume k = 2. ; 2nd and 4th track of 'N' hold the contents of 1st and 2nd tapes of 'M'.

Tracks 1 and 3 hold position of head of Tape1 and Tape 2 respectively.



Simulation of a 2-tape TM by 1-tape TM.

To simulate a move of 'M', 'N's head must visit the k head markers. To ensure N doesn't get lost, it must remember the no. of head markers to its left at all times; that count is stored as a compt. of N's finite control.

After visiting each head marker & storing the scanned symbol in a compt. of its finite control, N knows

(i) what tape symbols are being scanned by each of M's heads.

(ii) the state of M, which it stores in N's own finite control

Thus, 'N' knows what move 'M' will make.

N now revisits each of the head markers on the tape, change the symbol in each of corresp. tape of 'M' and moves the R/W head L/R, if necessary. Finally 'N' changes state of 'M' as recorded in its own finite control. ⇒ N simulated one move of 'M'.

We select as N's accepting states, all those states that record M's state as one of the accepting states of M. Thus whenever the simulated M accepts, N also accepts, N doesn't accept otherwise.

* **Halting problem** - Classical problem.

i/p to a TM (say 'M') is string 'w' then can we form an algo. to decide if M finishes the computing of 'w' in finite no. of steps.

ie, o/p = { yes, if M halts on w as i/p.
 no, if M doesn't halt.

~~Halti~~ OR.

Does TM finish computing of 'w' in finite no. of steps? The answer must be a yes / no.

▷ **Halting problem is undecidable.**

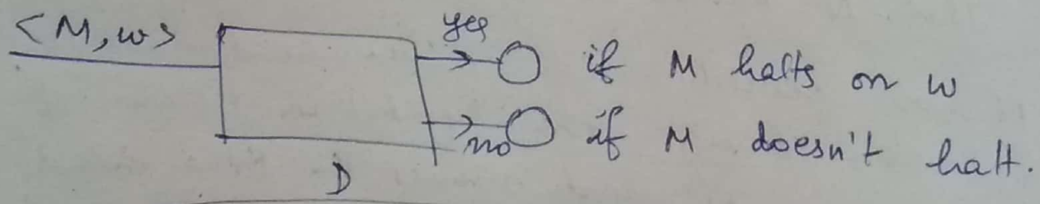
Proof. (by contradiction).

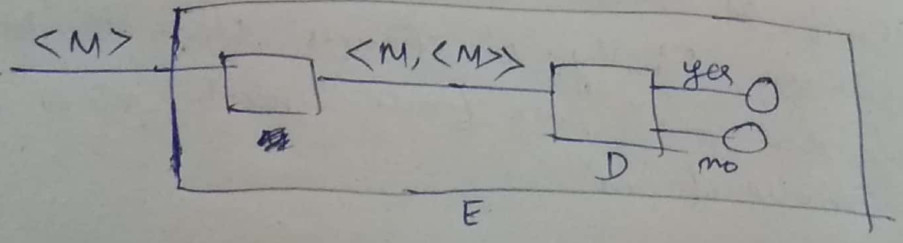
Suppose halting prob. is decidable.

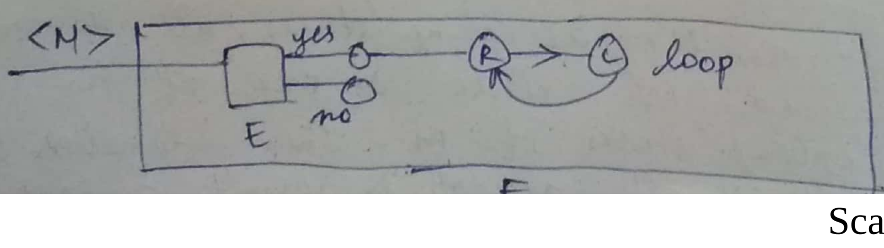
⇒ Existence of algo to decide if M halts on w for i/p M, w.

⇓ invoking Church / Turing thesis

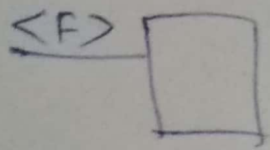
There is an existence of a TM that solves the halting prob. (say 'D')







modify the TM.



F

F halts on <F>
⇒ F doesn't halt on <F>

F doesn't halt on <F>
⇒ F halts on <F>

⇓

CONTRADICTION.

⇓

such an D doesn't exist

⇓

~~such~~
Halting prob. is undecidable.