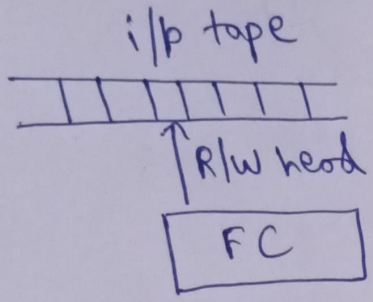
Turing Machine and unbecidability

Difference b/w TM & other machines like FA, PDA -

→ FA, PDA can move in one direction i.e left to right but TM can move in both the direction.

→ TM can read a symbol from the tape as well as write on it.

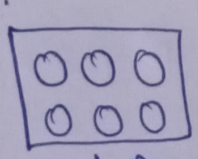


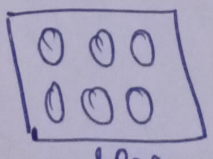
We will take simple example,

Q Suppose $a^n b^n \mid n \geq 1$. Design TM for this?

To give a basic idea, think of TM as a man who doesn't know to count.

Suppose we have two room, 1st room has white ball and 2nd room has black ball, now man has to count.

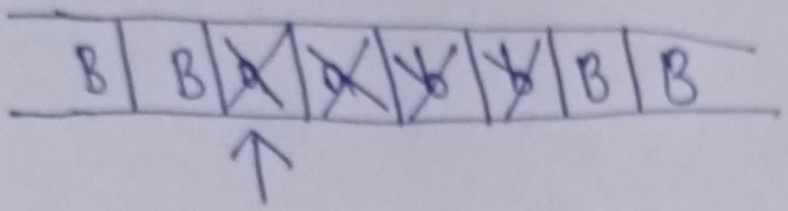

1st Room

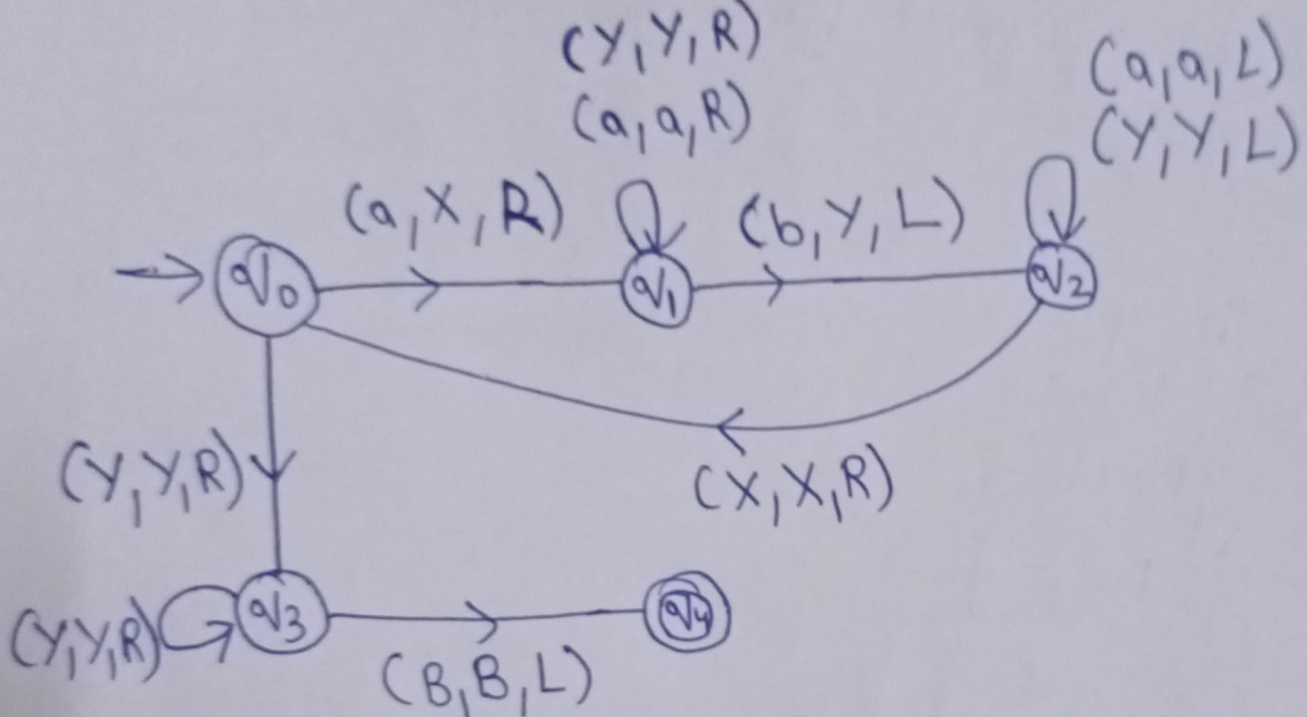

2nd Room

He can do by marking 1 ball in 1st room and then marking 1 ball in 2nd room and keep on repeating untill all balls one marked, thereby concluding that

1st room and 2nd room have same number of balls.

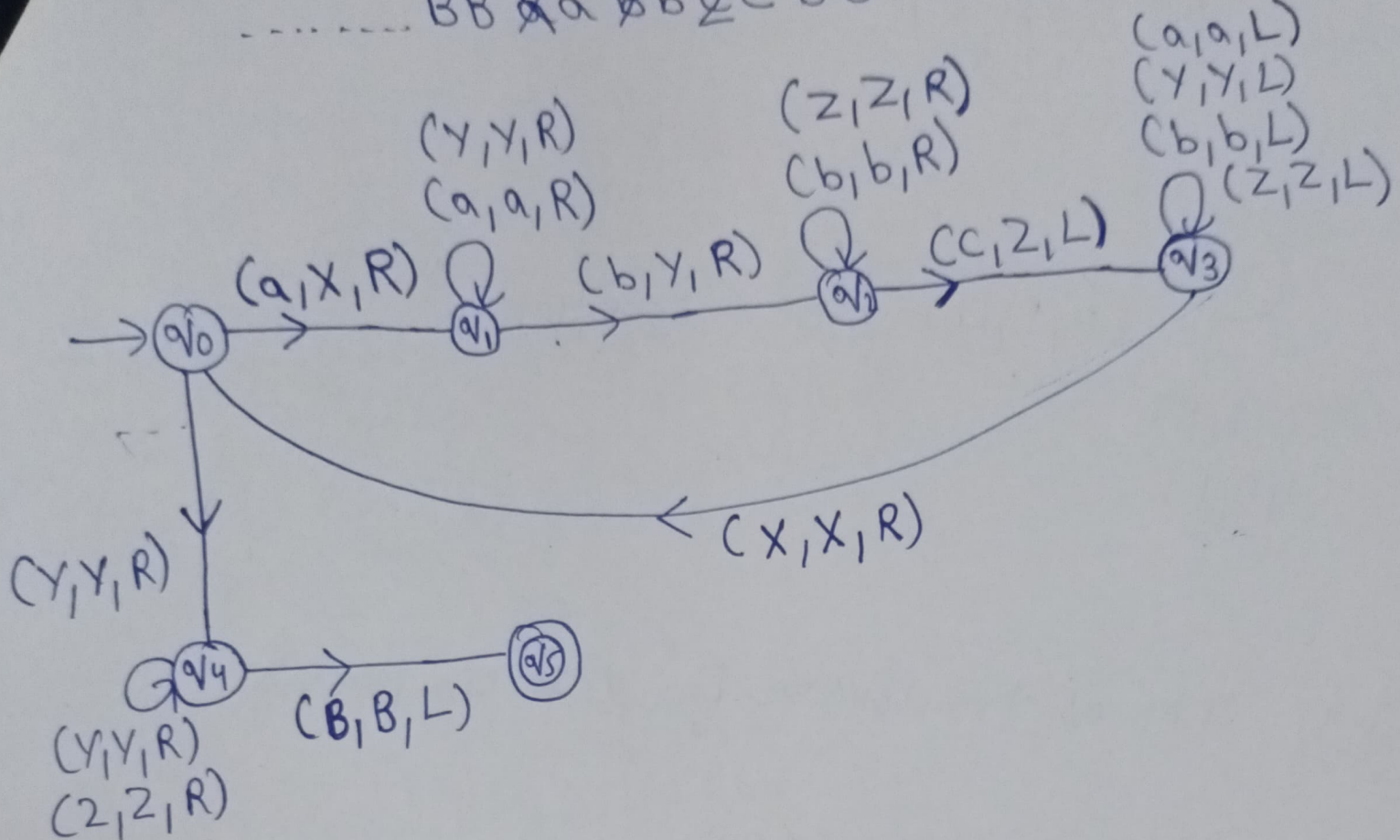
In similar way,





Ex 2 $L = \{ a^n b^n c^n \mid n \geq 1 \}$

..... BB X a X b Z c BB .....



for Transition function,

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

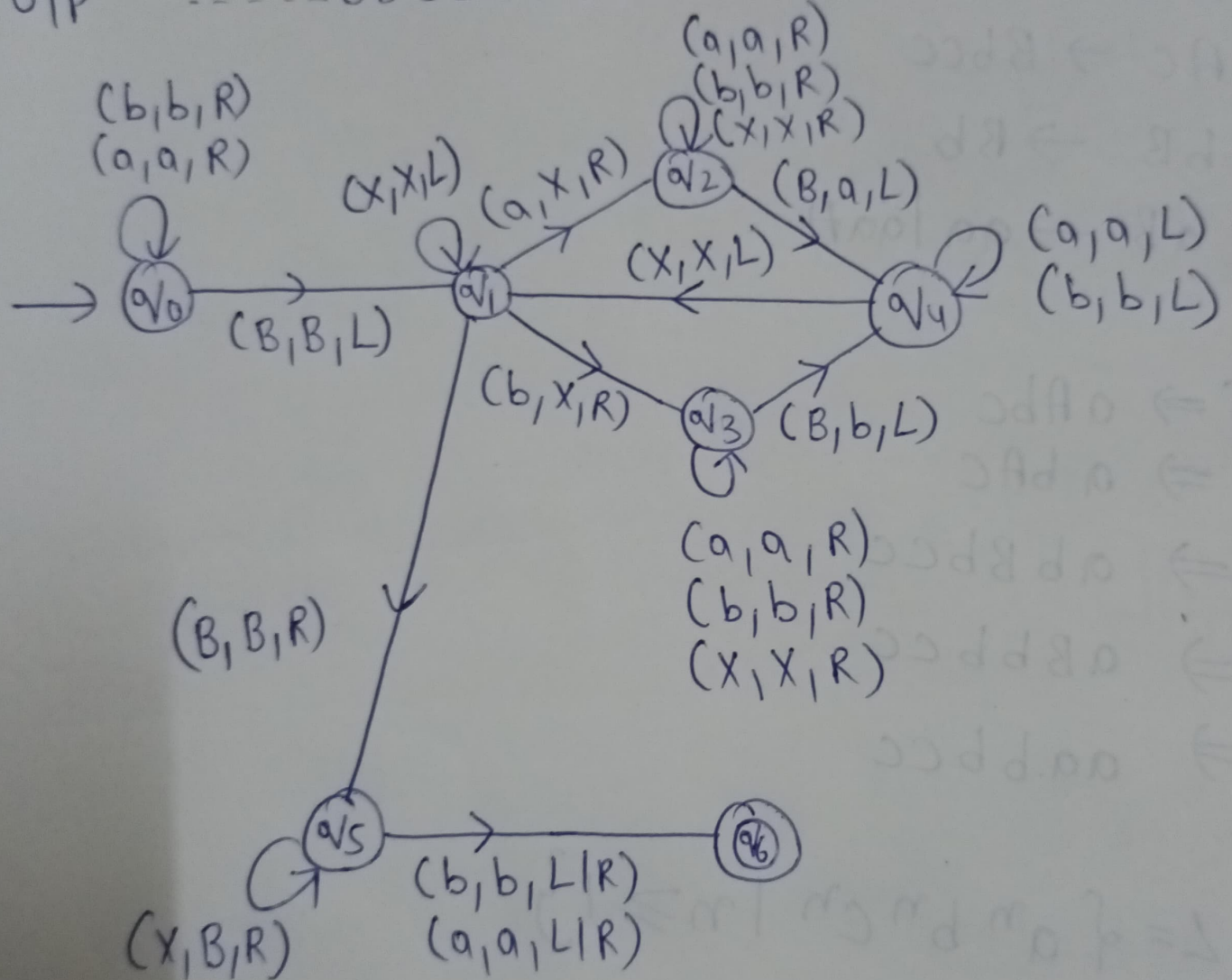
$\delta(q_0, Y) = (q_4, Y, R)$

and so on.

Q Design a TM to reverse string consisting of a's and b's.

I/p ..... BB aabb BB .....

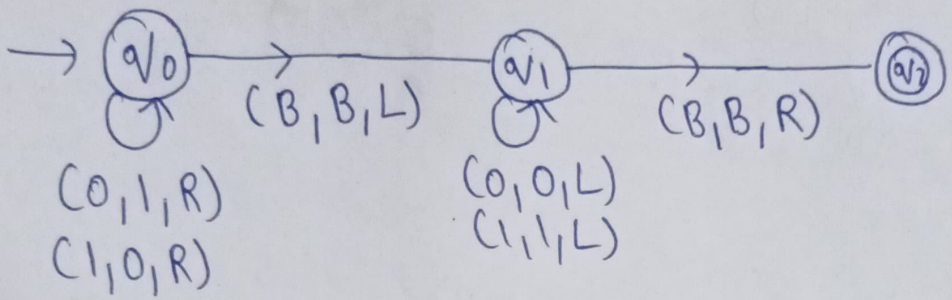
O/p ..... BB bbaa BB .....



TM as Computer of Integer function -

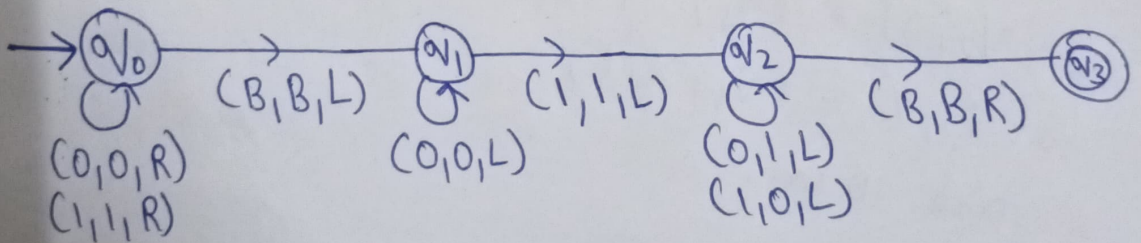
Ex3 TM to find 1's complement of a no.

... B B Ø Ø Y Y B B ...
1 1 0 0



Ex4 TM as transducer for 2's complement.

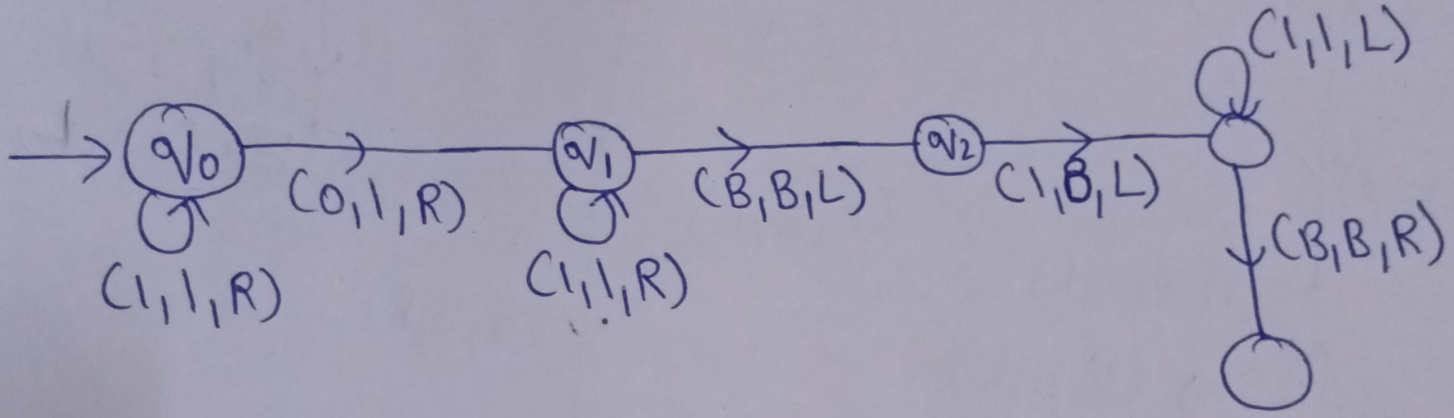
... B B Y Ø Y Ø Ø B B ...
0 1 1 0 0



TM as an adder -

3 = 111 4 = 1111 in unary system

I/p ..... BBB 111 0 1111 BBB .....

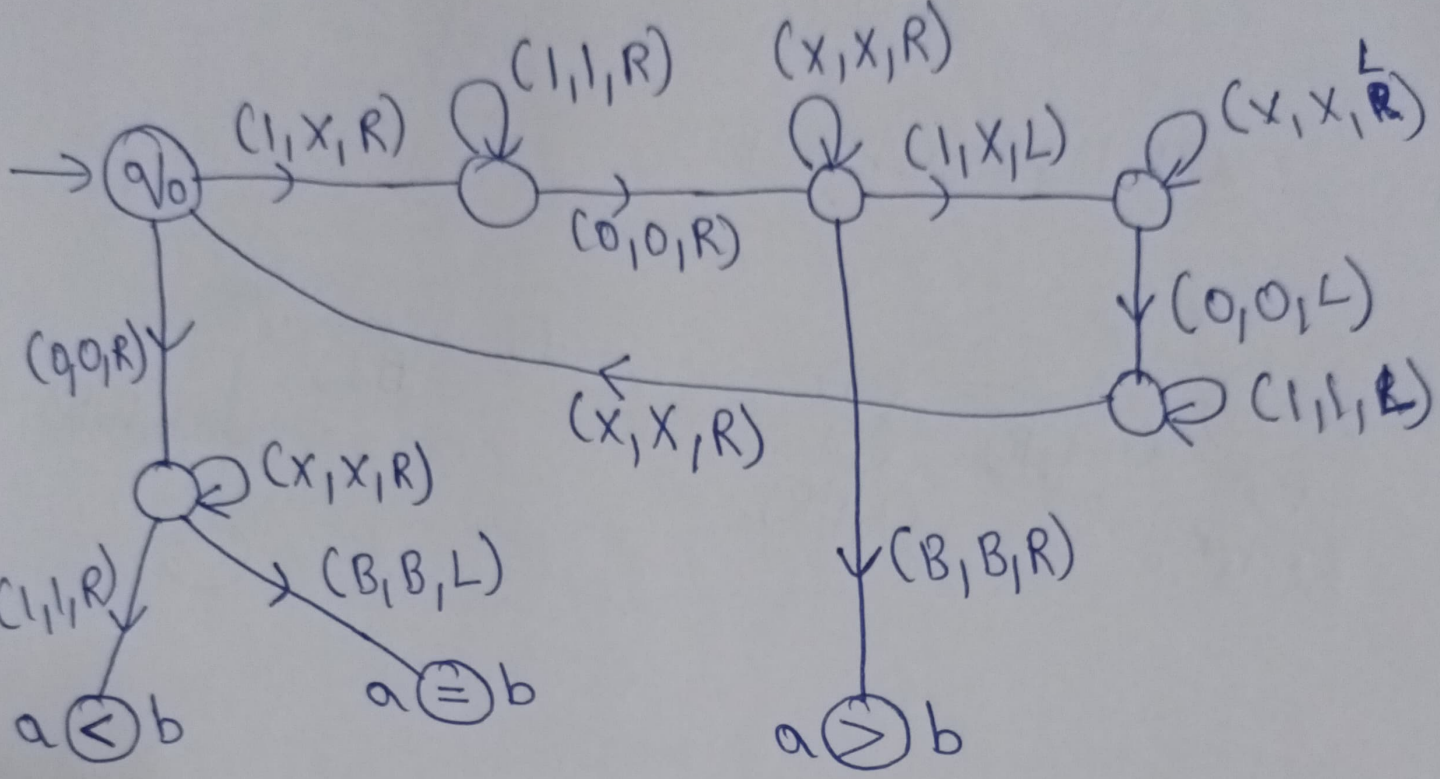


Final → ..... BBB 1111111 BBB .....

we need to concatenate 1's by removing 0 in b/w.

4) TM as comparator –

---BB*111O*111BB---



∵ Adder & comparator which are basic operations can be done than every mathematical operation can be done by having cascading of one or more TM. So, we can say TM is mathematically complete.

TM as a copier

I/p : ... BB 111 BB ....

o/p : ... BB 1111111 BB .....

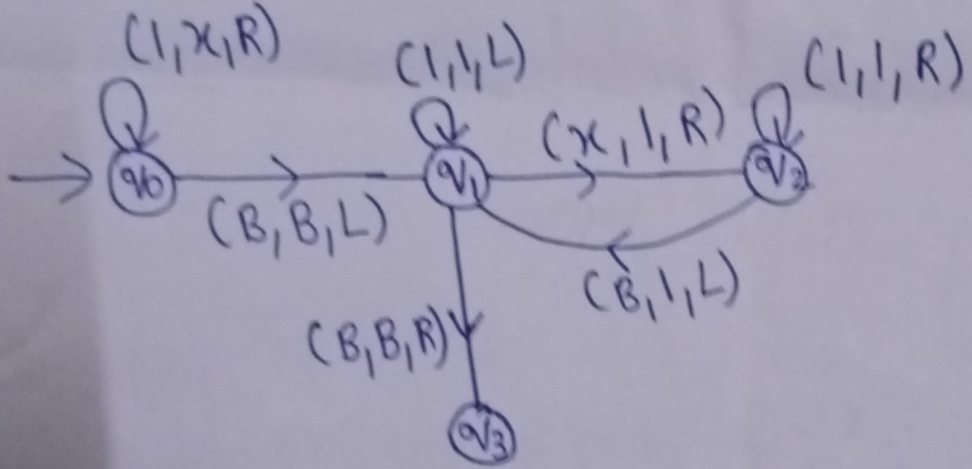
I/p tape ... BB 111 BB ....

↓

... BB x x x BB ....

↓

... BB 1111111 BB ....



Note - Using the copier, we can perform multiplication

Standard TM –

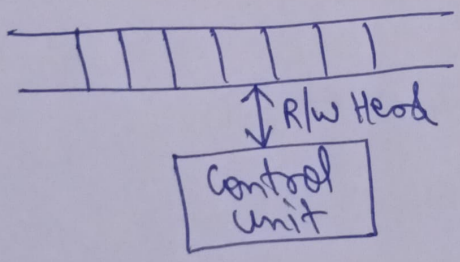
mathematically expressed as,

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

Q : set of states
Σ : i/p alphabet
Γ : Tape alphabet (set of alphabet that can be written on tape)

δ : transition function

q_0 : intial state
B : rep blank
F : set of final states (can be empty in case of transducer)



Deterministic TM :

$\delta : Q \times \{\Sigma\} \rightarrow Q \times \Gamma \times \{L, R\}$

It is partially deterministic since we have atmost one move for each configuration.

Summary –

→ Tape is unbounded, so any no of left & right moves possible.

→ every i/p or o/p will be given on tape.

→ Anything computation that can be carried out by mechanical means can be performed by some turing machine.

→ No one has yet suggested a problem, solvable by digitable computer, for which turing machine program cannot be written.

Modifications to standard Turing M/C –

1) TM with stay option –

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

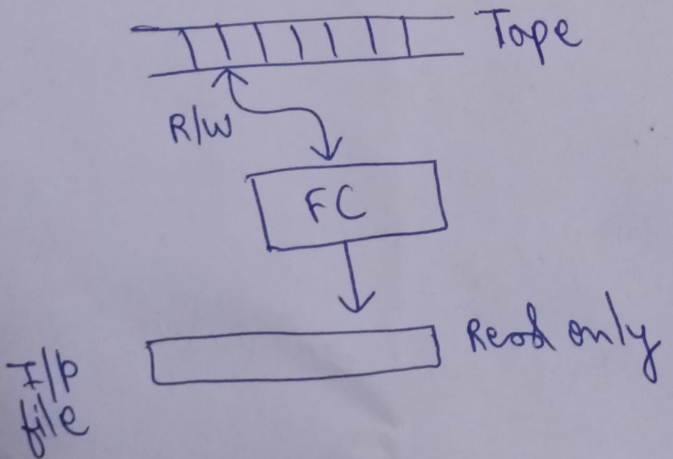
2) TM with semi-infinite tape –

---BBB|acb BBB....

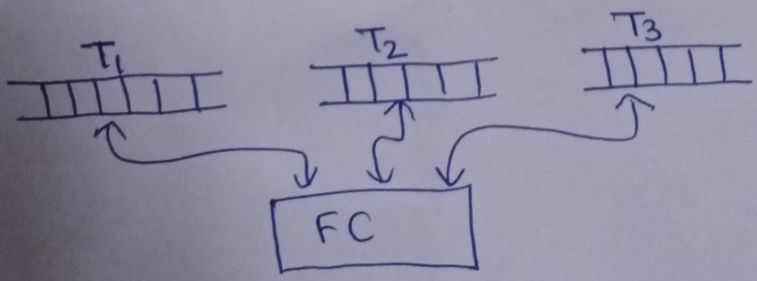
We will limite the tape from one side

3) Offline TM –

In this, we will have input in separate file and no modification to the input.



4) Multitape TM –

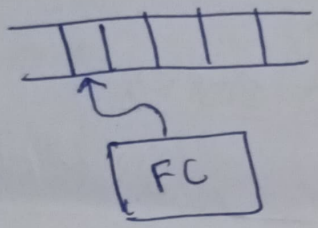


$$\delta : Q \times \Gamma^n \longrightarrow Q \times \Gamma^n \times \{L, R\}^n$$

Time Complexity may improve for some operation but power will remain same.

5) Jumping TM -

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \times \{n\}$$



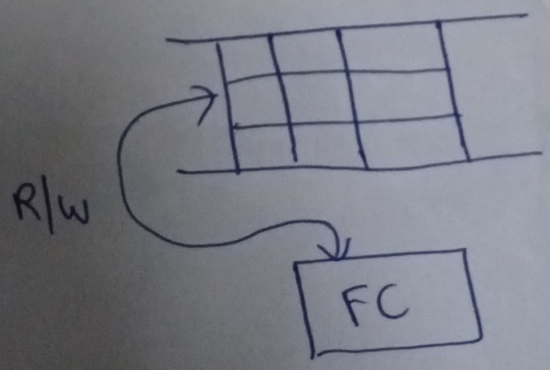
Instead of moving one step left or right, we can move n step at a time

6) Always writing TM -

. connot leave symbol as it

7) Multi-dimensional TM -

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



8) Non-Deterministic TM -

$$\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

ND-TM $\underset{\text{power}}{=}$ D-TM

9) Non-Erasing TM -

- i/p connot be changed to blank
- so instead of writing blank, use other symbol

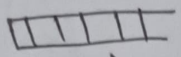
10) NPDA with two independent stacks -

$$\delta : Q \times (\Sigma \cup \epsilon) \times \underset{\text{(top of stack)}}{\Gamma} \times \Gamma \longrightarrow 2^{Q \times \Gamma^* \times \Gamma^*}$$

Linear Bounded Automata - (LBA)

→ We are not able to extend the power of TM by any kind of modifications.

→ But can we limit the power of TM by restricting the way we are going to use the tape.

NDTM +  → PDA

stack

(limit power of tape as stack)

TM + Tape (finite size) → FA

i.e whatever i/p but memory is finite

→ Now, If we restrict TM such that it uses the tape where i/p is present only.

ex- [aabb]

↑ ↑

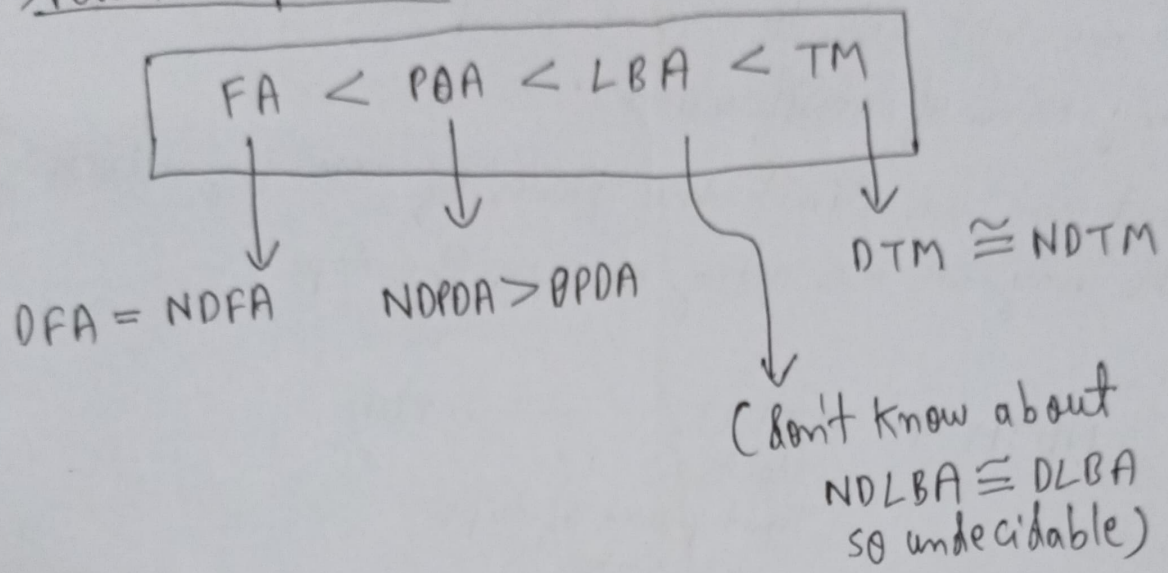
LBA = TM + tape (input size)

i.e size of tape will depend on size of i/p

therefore, TM will get opportunity to use large memory for large i/p and small memory for small i/p.

ex- $a^n b^n c^n$ can be accepted by LBA but not by PDA.

→ Power Comparison -



Context Sensitive Language & Grammer -

CSG ← generator — CSL — acceptor → LBA

A grammer is CSG if all productions are of form,

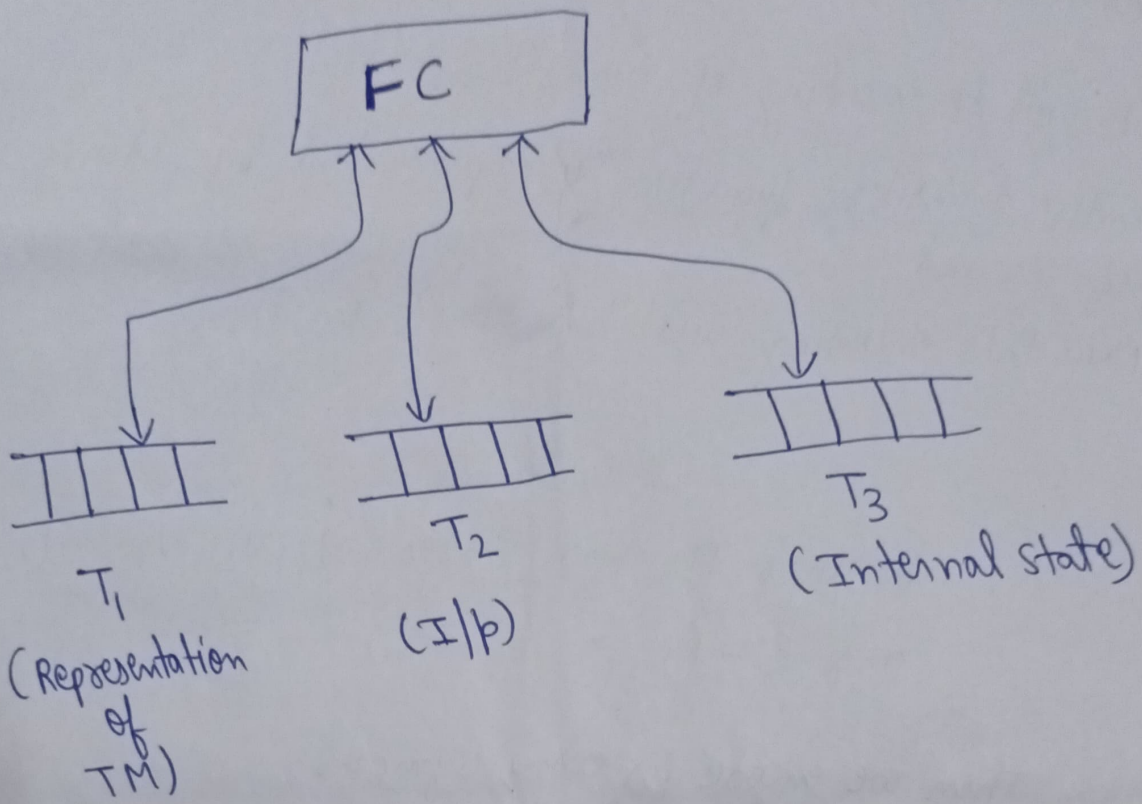
$$x \to y$$

where $x, y \in (V \cup T)^+$ and

$$|x| \leq |y|$$

Note - Here, derivation will always expand or remain same (non-contracting grammer always)

Universal TM and Encoding of TM -



→ has 3 tapes
→ UTM will take TM and store in one Tape T_1 which is representation of some other TM.
→ Whatever i/p is given in TM is stored in Tape T_2
→ Tape T_3 is used for rough work means UTM will trace out behaviour of T_1 on the i/p T_2.
→ Suppose initially we are in q_1 and see first symbol of i/p on T_2, then go to T_1 and see for this configuration, approximate changes we need to make.

Encoding of TM -

How we will save entire TM in one tape T_1?

- We will do encoding of transitions.
- Entire transition function is represented by strings of 0's and 1's.
- Now this encoding will be stored on T_1.

Ex- Original TM has states $(q_1, q_2, q_3 \ldots)$ and $\Gamma = (a_1, a_2, a_3 \ldots)$

Then we encode for every symbol

$q_1 - 1$	$a_1 - 1$	$L - 1$
$q_2 - 11$	$a_2 - 11$	$R - 11$
$q_3 - 111$	$a_3 - 111$	
⋮	⋮	

We can use '0' as separator

Suppose transition func^n,

$$\delta(q_1, a_1) = (q_2, a_2, R)$$

$$(q_1, a_1, q_2, a_2, R)$$

$$\downarrow$$

$$\underbrace{10101101101}_{\text{1st transition}}\underbrace{00\ldots\ldots}_{\text{2nd transition}}$$

Summary -

→ Every TM can be represented as string of 0's and 1's as every transition func^n can be encoded

→ $\Sigma = \{0, 1\}$

Σ^* → universal language

TM is one of string in this language

→ Every TM can be written as string of 0's and 1's but every string of 0's and 1's is not a TM.

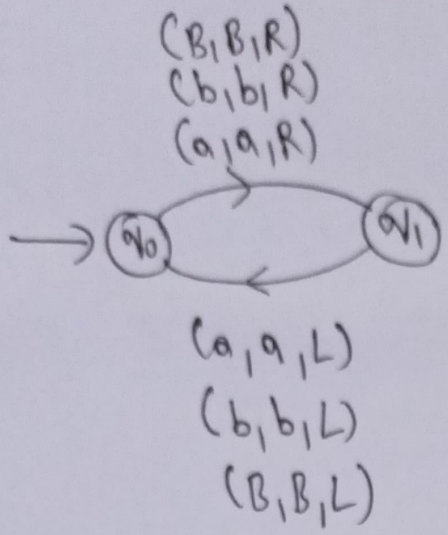
→ TM = string of 0's & 1's = binary number
= decimal number

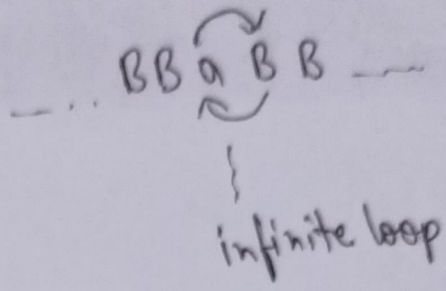
0^{th} TM, 1^{st} TM, ... TM$_i$

Challenge - Not every binary string is a TM

ex- 10 → not a valid TM binary encoding

Non-Halting TM





→ This does not happen in PDA or FA because we move only in one direction, if i/p is over so halt but in TM we can move in both direction.

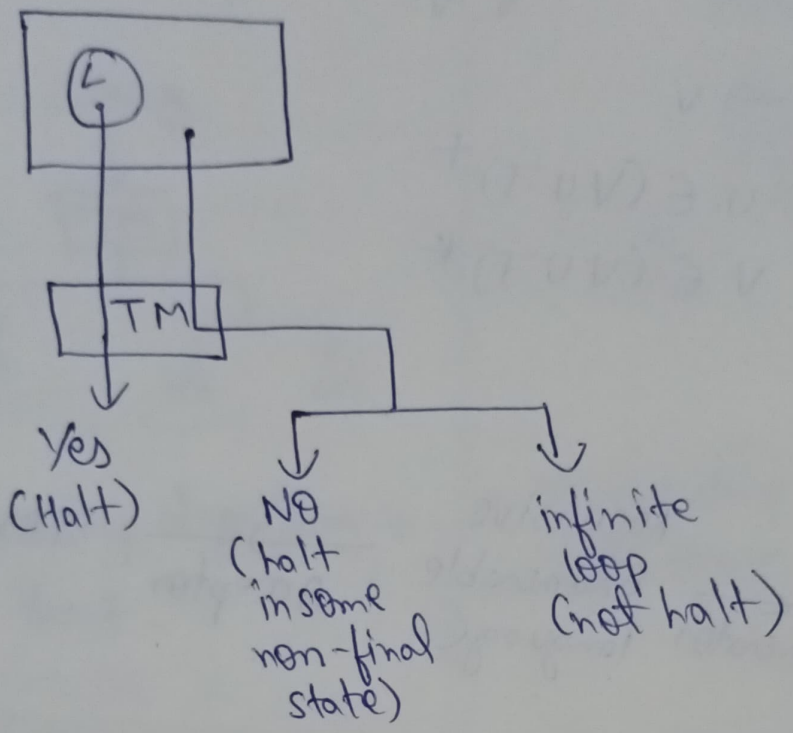
This gives rise to halting problem.

→ Given an i/p, if it is in language it will halt and say it is in language.

→ when i/p is not in language it will never halt or might halt, you might think if i give some time it might halt but then how much time is not known, also sometimes you might think TM has fallen in infinite loop and you might stop it but TM has not fallen in infinite loop it might be doing some real work.

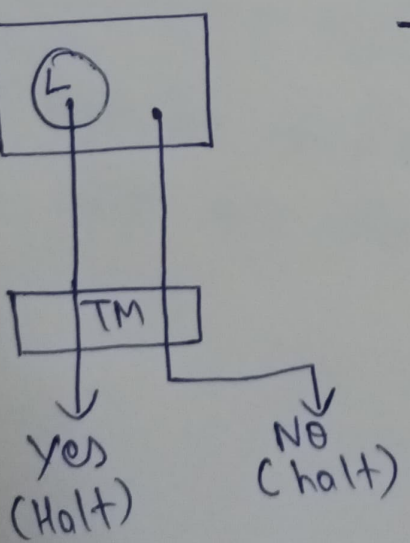
Recursive Enumerable and Recursive Language –

→ Σ^*



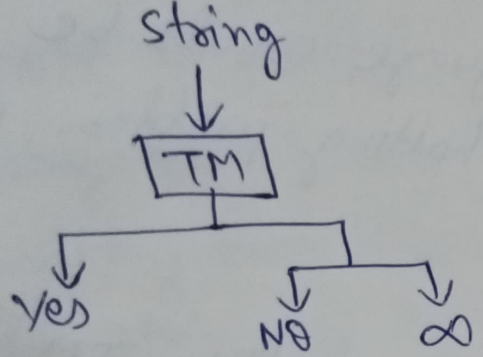
If there is a TM (halting or non-halting) for a language then the language is recursive enumerable.

→ Σ^*



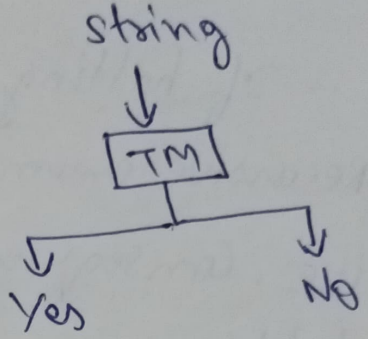
Then there is a subclass of language where turing machine always halts (halting TM), then such language is called recursive language.

Recursive Enumerable

- TM (we never know it will halt or not)
- 

- Membership algorithm is not there.

Recursive

- TM which always halt
- 

- Membership algorithm is there.

Unrestricted Grammer -

→ Grammes which will generate recursive enumerable languages.

→ if all productions are of form,

$$U \rightarrow V$$

where $U \in (V \cup T)^+$

$V \in (V \cup T)^*$

→

Unrestricted Grammer ← generator — Recursive Enumerable language → Accepter — TM

TM Halting Problem –

Given description of TM 'M' and input 'w'. Does 'M' when started with 'w' as its input eventually halts?

→ Theorem – If halting problem were decidable, then every Recursive enumerable language would be Recursive. Consequently, the halting problem is undecidable.

→ There is no algorithm to prove whether TM will halt or not. So, it is undecidable.

Important Theorem on Recursive & Recursive Enumerable –

a) → If a language L and its complement $\bar{L}$ are both RE, then both languages are recursive

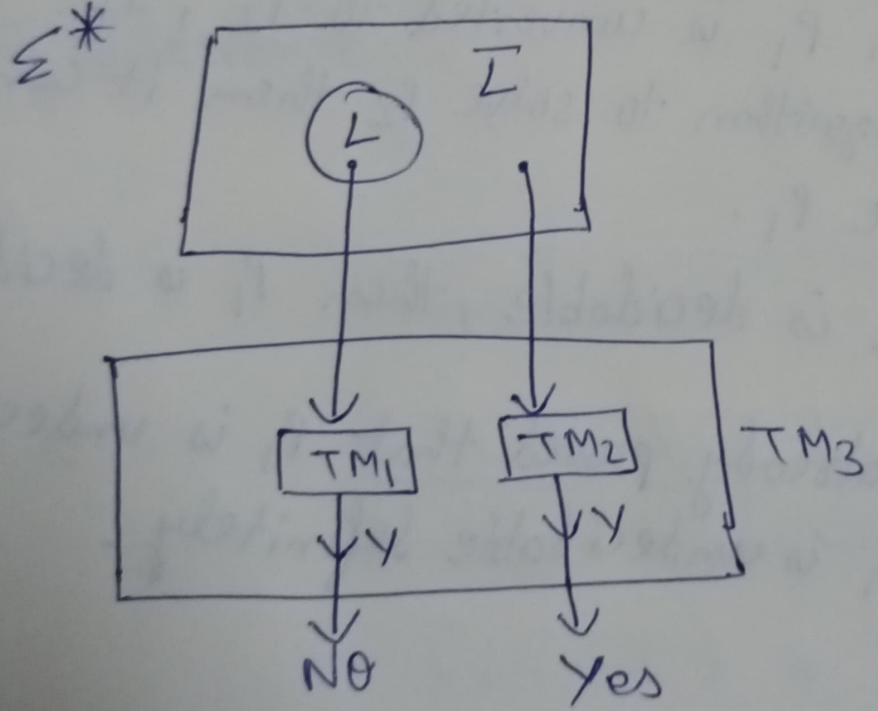
b) → If L is recursive and $\bar{L}$ is also recursive and consequently both are RE.

Proof

a) L & $\bar{L} \rightarrow RE$

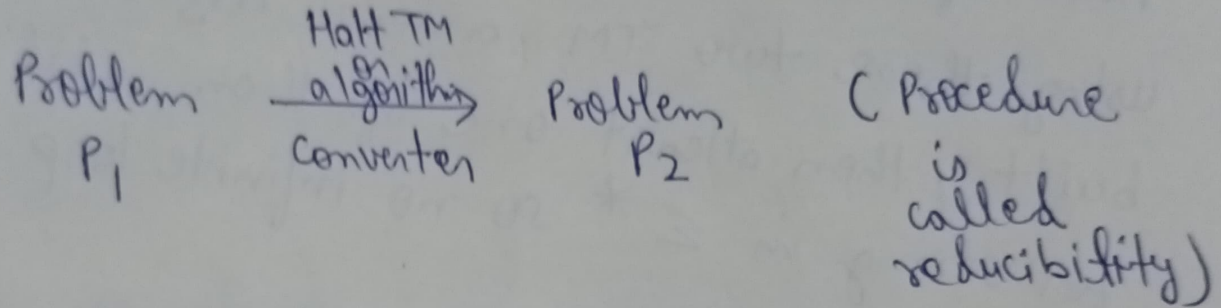
There will be TM_1 & TM_2 for both which halt if words is present in L and $\bar{L}$.

using these two TM, a new TM_3 will be built, then atleast one of them will halt for string in Σ^* so no infinite loop.



Decidability –

- talks about existence of algorithm.
- Algorithm ⟶ TM (only which halts always)
- Prove Problem is undecidable?



→ If problem P_1 is converted to P_2, if there is algorithm to solve P_2 then it can also solve P_1.

→ So, if P_2 is decidable, then P_1 is decidable

→ If it is already proved that P_1 is undecidable then P_1 is undecidable definitely.

Undecidable Problems on TM Halting Problem -

→ **State Entry Problem** -

Given a TM, state $q \in Q$ and $w \in \Sigma^+$, decide whether or not the state 'q' is even entered when TM is applied to w. This is undecidable.

→ Given a TM 'M', whether or not 'M' halts if started with blank tape. This is undecidable.

→ Almost any problem related to R.E language is undecidable.

Post - Correspondence Problem — (PCP)

Given two seq of 'n' strings on some alphabet Σ, say $A = w_1 w_2 \cdots w_n$ and $B = v_1 v_2 \cdots v_n$, we can say there exists a PC-soln for pair (A, B) if there is a non-empty sequence of integers $i, j, \cdots k$ such that

$$w_i w_j \cdots w_k = v_i v_j \cdots v_k$$

PC problem is to device an algo that will tell us for any (A, B), whether or not there exist a PC-soln.

Ex- $A =$

w_1	w_2	w_3
a	ab	bba

$B =$

v_1	v_2	v_3
baa	aa	bb

soln is (3 2 3 1)

$A \rightarrow w_3 w_2 w_3 w_1 =$ bbaabbbaa

$B \rightarrow v_3 v_2 v_3 v_1 =$ bbaabbbaa

So, how can we prove PCP is undecidable?

PCP (P_1) ⟶ convert P_2 (Ambiguity Problem)

Strings can be derived in many ways so ambiguity problem which is already undecidable so PCP is also undecidable.

Q

	A	B
1	1	111
2	10111	10
3	10	0

A → (2113)

11

Q Language generated by CSG -

$S \rightarrow abc \mid aAbc$

$Ab \rightarrow bA$

$Ac \rightarrow Bbcc$

$bB \rightarrow Bb$

$aB \rightarrow aa \mid aaA$

$S \Rightarrow aAbc$

$\Rightarrow abAc$

$\Rightarrow abBbcc$

$\Rightarrow aBbbcc$

$\Rightarrow aabbcc$

$L = \{ a^n b^n c^n \mid n \geq 1 \}$

Decidability Table

*	RL	DCFL	CFL*	CSL	Recursive Language*	L REL
1) Does $w \in L$ (membership)	D	D	D	D	D	UD
2) Is $L = \phi$? (Emptiness)	D	D	D	UD	UD	UD
3) Is $L = \Sigma^*$ (Completeness)/Totality	D	D	UD	UD	UD	UD
4) Is $L_1 = L_2$ (Equality)	D	UD	UD	UD	UD	UD
5) Is $L_1 \subseteq L_2$ (Subset)	D	UD	UD	UD	UD	UD
6) $L_1 \cap L_2 = \phi$ (Intersection finiteness)	D	UD	UD	UD	UD	UD
7) Is L_1 finite? (Finiteness)	D	D	D	UD	UD	UD
8) If complement of L'a language of same type or not?	D	D	UD	D	D	UD
9) Is intersection of two languages of same type	D	UD	UD	UD	UD	UD
10) Is L a regular language	D	D	UD	UD	UD	UD

Date
Page

Page

Closure Property Table

Property	RL	DCFL	CFL	CSL	Recursive Language	Recursive Enumerable Language
Union	✓	×	✓	✓	✓	✓
Intersection	✓	×	×	✓	✓	✓
set difference	✓	×	×	✓	✓	×
Complementation	✓	✓	×	✓	✓	×
Concatenation	✓	×	✓	✓	✓	✓
Kleene Closure (L^*)	✓	×	✓	✓	✓	✓
Kleene Plus (L^+)	✓	×	✓	✓	✓	✓
Reversal	✓	×	✓	✓	✓	✓

FREEMIND

Date

Page

Not closed under -

→ RL - subset, superset relation, infinite operation

→ CFL - Intersection, set difference, complement Quotient, Inverse substitution.

→ DCFL - union, concatenation, Kleene closure, Homomorphism, Reversal, intersection, substitution, set difference

→ CSL - ~~xxxx~~ Homomorphism, substitution

→ Recursive Language - Homomorphism, substitution Quotient with RL

→ Recursive Enumerable Language - set-difference, complement

Note-

- NO languages are closed under subset and infinite union.
- R.L are not closed under infinite union and infinite intersection.
- complement of non-regular is always non-regular.
- If something is closed under union and complement then it will be surely closed under intersection.