Name - Varun Gupta

Roll No - 2023101108

THEOREY ASSIGNMENT - 2

A2)

1.) $L = \{ a^n b^m \mid m = n^2 \}$

Let L is a Context free Language.

Let a string $w = a^n b^{n^2} \in L$.

Using Pumping lemma (with pumping length = n)

$\because$ $w = uvxyz$

$|vy| \geq 1$ & $|vxy| \leq p$

$uv^i xy^i z \in L$.

Now $vxy = a^k$ or b^k or $a^m b^n$; $k \leq p$ & $m+n \leq p$

Case-1:- $vxy = a^k$, $k \leq p$

$|vy| = c$ $(c \geq 1)$

$w' = uv^0 xy^0 z = uxz$

$= a^{n-c} b^{n^2}$

But $w' \notin L$ as $(n^2 \neq (n-c)^2)$

Case-2:- $vxy = b^k$

$|vy| = c$ $(c \geq 1)$

$w' = uxz = a^n b^{n^2 - c}$

$w' \notin L$ as $n^2 - c \neq n^2$

Case-III: $vxy = a^m b^n$, $m+n = k \leq p$

$vy = a^{m'} b^{n'}$; $0 \leq m' \leq m$, $0 \leq n' \leq n$

but $1 \leq m'+n' = k' \leq k \leq p$

Since $|vy| \geq 1$

and $w^0 = uv^0xy^0z = uxz = a^{p-m'} b^{p^2-n'}$

$(p-m')^2 = p^2 + (m')^2 - 2pm'$ acc to Pumping Lemma

For $w^0 \in L$, we need

$p^2 + (m')^2 - 2pm' = p^2 - n'$

$\Rightarrow w^0 = a^{p-m'} b^{(p-m')^2} \Rightarrow w' \in L$

but then for $i=2$;

$w^{*2} = uv^2xy^2z = a^{p+m'} b^{p^2+n'}$

$(p+m')^2 = p^2 + (m')^2 + 2pm'$

$w^{*2} \in L$ if $p^2 + (m')^2 + 2pm' = p^2 + n'$

$\Rightarrow (m')^2 + 2pm' = n'$ ——①

from ①, $n' = -((m')^2 - 2pm')$ ——②

From ① & ② ~~Contradict each other~~ $w', w'' \in L \Rightarrow m' = 0$

but then $|vy| = 0$ $n' = 0$

It Contradicts the statement of Pumping lemma

~~Case ① Failed~~ ∴ L is not Context Free

Ans 1) $L = \{ xy \mid x, y \in \{0,1\}^*, |x| = |y|, x \neq y \}$,

To Proove it a Context free language we have to find a PDA that recognizes it or Grammar which can generate it.

Let L_1 is a context free language such that

$$L_1 = \{ xy \mid x, y \in \{0,1\}^*, |x| = |y| \}$$

Grammar, G_1:

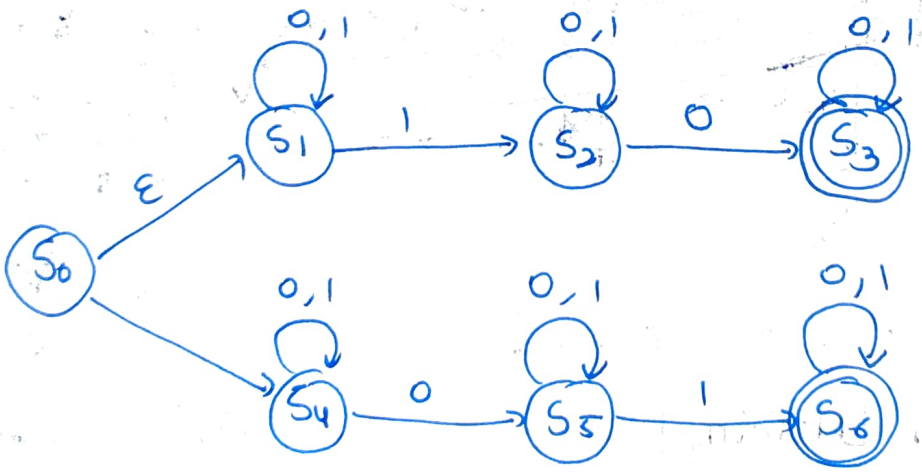
$$S \longrightarrow 0S0 \mid 1S1 \mid 0S1 \mid 1S0 \mid \varepsilon$$

L_1 can be generated by G_1.

Let L_2 be language, $L_2 = \{ xy \mid x, y \in \{0,1\}^*, x \neq y \}$

It is a regular language which can be proved as I am giving a NFA for it.

Also, $|x| > 0$ & $|y| > 0$ as $|x| = |y|$ also $|x| = |y| = 0$ is not possible bcz in that case $x = y$ but $x \neq y$.



Now,

$L_1 \cap L_2 = \{xy \mid x, y \in \{0,1\}^* \text{ and } |x| = |y|, x \neq y\}$

Now this is how 'L' is defined,

$$L = L_1 \cap L_2$$

$L_1 \longrightarrow$ Context Free, $L_2 \longrightarrow$ Regular Language

$L \longrightarrow$ Context Free Language.

Proof for Union of CFL and Regular Lang is RL :

$L_1 \longrightarrow$ CFL

$L_2 \longrightarrow$ RL

$L = L_1 \cup L_2$

for L_1 there will be a pushdown Automata P_1 that recognizes L_1. Since L_2 is regular $\exists$ a Finite Automata A that recognizes L_2.

Let's construct a new pushdown Automata P that recognizes $L_1 \cup L_2$ using P_1 & A.

States: All states of P_1 & A also a new start q_s.

Transistion: For P_1 & A transition would be same

For new start state q_s;

for every transition in A we can move to the corresponding state of A without using stack.

Add an epsilon transition from q_s to the start state of P_1.

Accept: Accepting state of P will be accepting states of P_1 & A.

Stack:- When P simulates A, it does not use its stack. when running P_1 use stack as per its rule

The new PDA can either simulate the A to accept strings from L_2 or P_1 to accept strings from L_1

∴ PDA P recognizes $L_1 \cup L_2$

∴ $L_1 \cup L_2$ is CFL

~~Or, We can also find~~

Alternate:-

(Reference from Stack Exchange)

$S \longrightarrow AB \mid BA$

$A \longrightarrow 0 \mid 0A0 \mid 0A1 \mid 1A0 \mid 1A1$

$B \longrightarrow 1 \mid 0B0 \mid 0B1 \mid 1B0 \mid 1B1$

L{ This will generate the required Language L.

Ans 3)

$E \longrightarrow I$

$E \longrightarrow E + E$

$E \longrightarrow E * E$

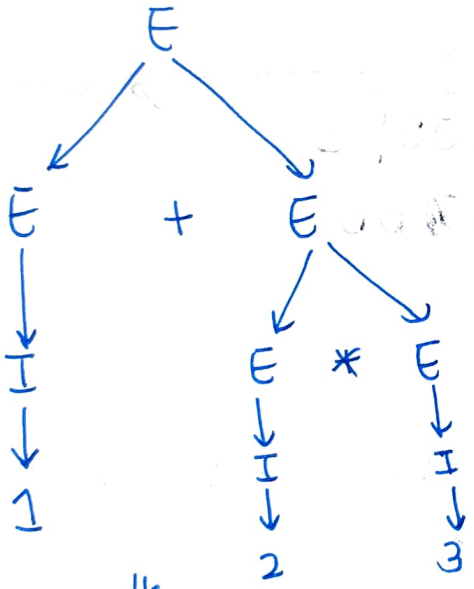
$E \longrightarrow (E)$

$I \longrightarrow \varepsilon \mid 0 \mid 1 \mid - - - - \mid 9$

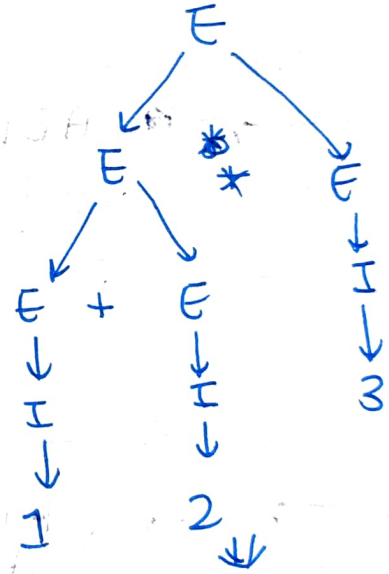
A Grammar is said to be ambiguous if we can get two leftmost/rightmost derivation/Parse tree for a same string.

Let consider string, $w = 1 + 2 * 3$

Leftmost Parse Tree :-



1 + 2 * 3



1 + 2 * 3

∴ Grammar is Ambiguous

Q

Ans 4)

$$A \longrightarrow BAB \mid B \mid \varepsilon$$
$$B \longrightarrow 00 \mid \varepsilon$$

For Converting to Chomsky Normal Form the production rules must be of form :-

$$A \longrightarrow BC$$
$$A \longrightarrow a$$
$$S \longrightarrow \varepsilon$$

Here S is our Start Variable and adding $S \longrightarrow A$

⇒ Removing $B \longrightarrow \varepsilon$ and $A \longrightarrow \varepsilon$

$$S \longrightarrow A \mid \varepsilon$$
$$A \longrightarrow BAB \mid BA \mid AB \mid A \mid B \mid BB$$
$$B \longrightarrow 00$$

⇒ Removing $A \longrightarrow A$ (trivial) & $A \longrightarrow B$, $S \longrightarrow A$

$$S \longrightarrow BAB \mid BA \mid AB \mid BB \mid 00 \mid \varepsilon$$
$$A \longrightarrow BAB \mid BA \mid AB \mid 00 \mid BB$$
$$B \longrightarrow 00$$

⇒ Removing $A \longrightarrow BAB$ & $A \longrightarrow 00$ & $B \longrightarrow 00$

$$S \longrightarrow CB \mid BA \mid AB \mid BB \mid DD \mid \varepsilon$$
$$A \longrightarrow CB \mid BA \mid AB \mid DD \mid BB$$
$$B \longrightarrow DD$$
$$C \longrightarrow BA$$
$$D \longrightarrow 0$$

Chomsky Normal Form

Ans 5) To Proove: PDA with 2 Stack is more Powerfull than the PDA with 1 Stack.

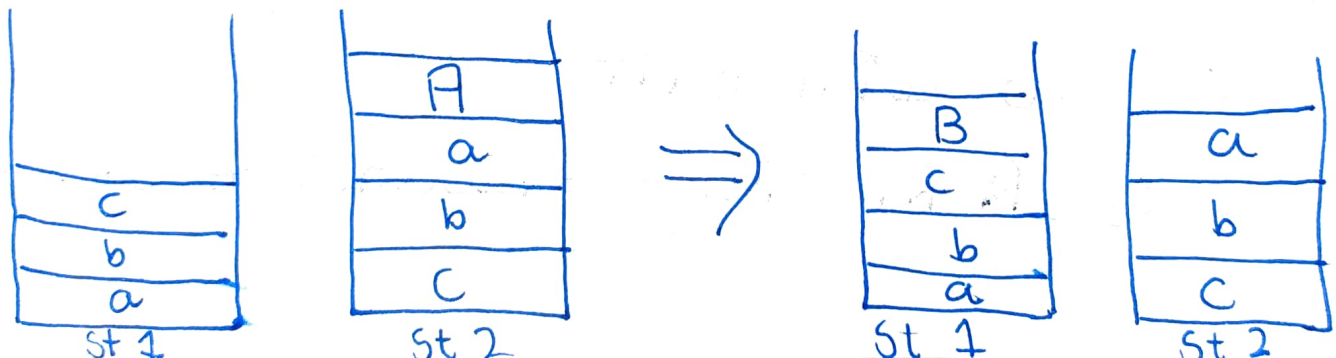
Proof:

Claim:- PDA with 2 Stack is equivalent to Turing Machine.

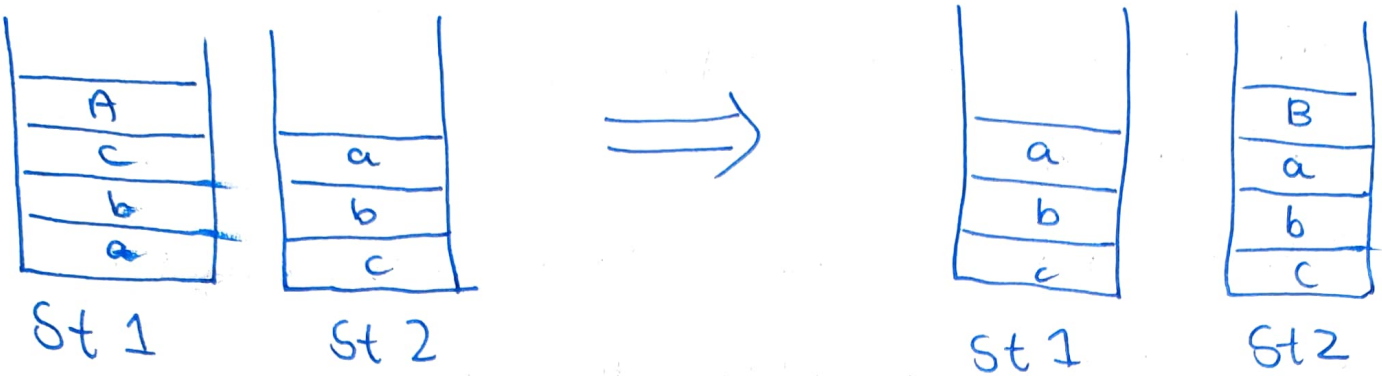
Proof:- Stack 1 will represent the portion of tape to the ~~left~~ and Stack 2 will represent the portion of the tape to the right of the current head pointer as well as the current position.

Initially, Stack 1 is empty and Stack 2 contain the complete string with current position is at the top of Stack 2.

Transition: Let A be any tape signal & B is the signal written & then move right. Top element of Stack 2 is ~~popped~~ (which is A) & B is ~~popped~~ pushed in the Stack 1.



Similarly if it move left reading A and writing B then,



If Turing Machine extends the accepting or rejecting stak then 2 Stack PDA will also accept & reject respectively.

Hence, we can say that 2Stack PDA is equivalent to TM.

Now Claim PDA with 2 Stack is ~ TM and PDA with 1 Stack will accept Context Free Language.

CFL ⊂ TM. TM will accept extra strings than that in CFL.

Note:- TM transition are used like it is done in Sipser. $a \to b, L/R \Rightarrow$ Read a, Write b, Move L/R

Ans 6) $f(\#\langle x \rangle) = \begin{cases} \#\langle x/2 \rangle & \text{if } x \text{ is even} \\ \#\langle 3x+1 \rangle & \text{otherwise} \end{cases}$

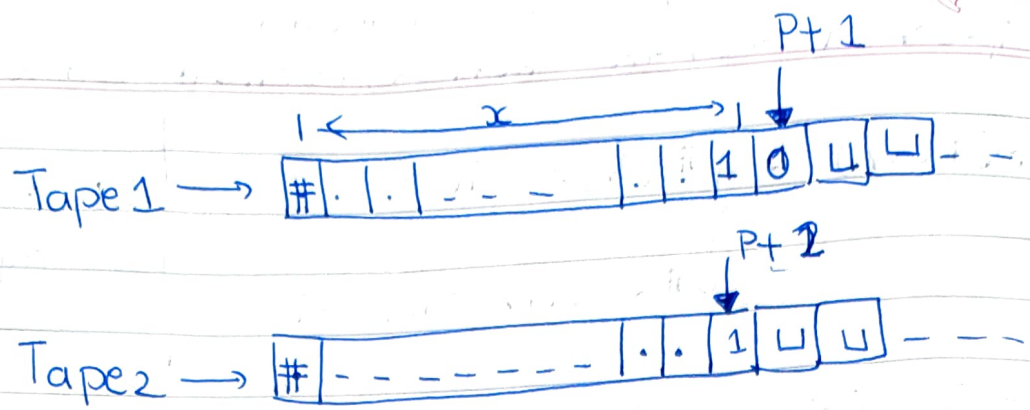
$\sqcup \to$ Blank Symbol

Mult Tape Turing Machine (a variant of TM) which is equivalent in power with Normal TM.
We are moving 3 Tape Turing Machine. Tape's also has ~~2 Tape~~ 1, Tape2 ~~are normal Tape Tape 3 head~~ has a don't move/stay (S) option with L/R.

Tape 1 $\longrightarrow$ Input String with # in front
Tape 2 $\longrightarrow$ Input String Copy with # in front
Tape 3 $\longrightarrow$ 1

First move the head pointer of Tape 1 & 2 to the last character in string. If it is '0' then the input is even in that case we need to output $x/2$ ∴ ~~Replace~~ Do "$0 \to \sqcup, L$". It will be same as shift ~~Left~~ Right Operation. Output is Tape 1.

If the last character is '1' then the input is odd in that case we need to output $3x+1$.
Now $3x+1 = (2x) + (x) + (1)$
What we will do is "$1 \to 1, R$ and $\sqcup \to 0, L$"
Basically it is left shift equivalent to '$2x$'.
Now, Tape 1 $\longrightarrow 2x$, Tape 2 $\longrightarrow x$
We just need to add Tape 1 & Tape 2 taking Tape 3 as carry bit (it initially only contains 1 which will make results $3x+1$)



~~Now we will see it with the help of State Diag~~

Now as we are dealing with Tapes redefining Transition defn as,

$(a_1, a_2, a_3) \longrightarrow (b_1, b_2, b_3)$, ~~L/R/S~~ L/R/S | L/R/S | L/R/S

↳ Basically don't change the Tape

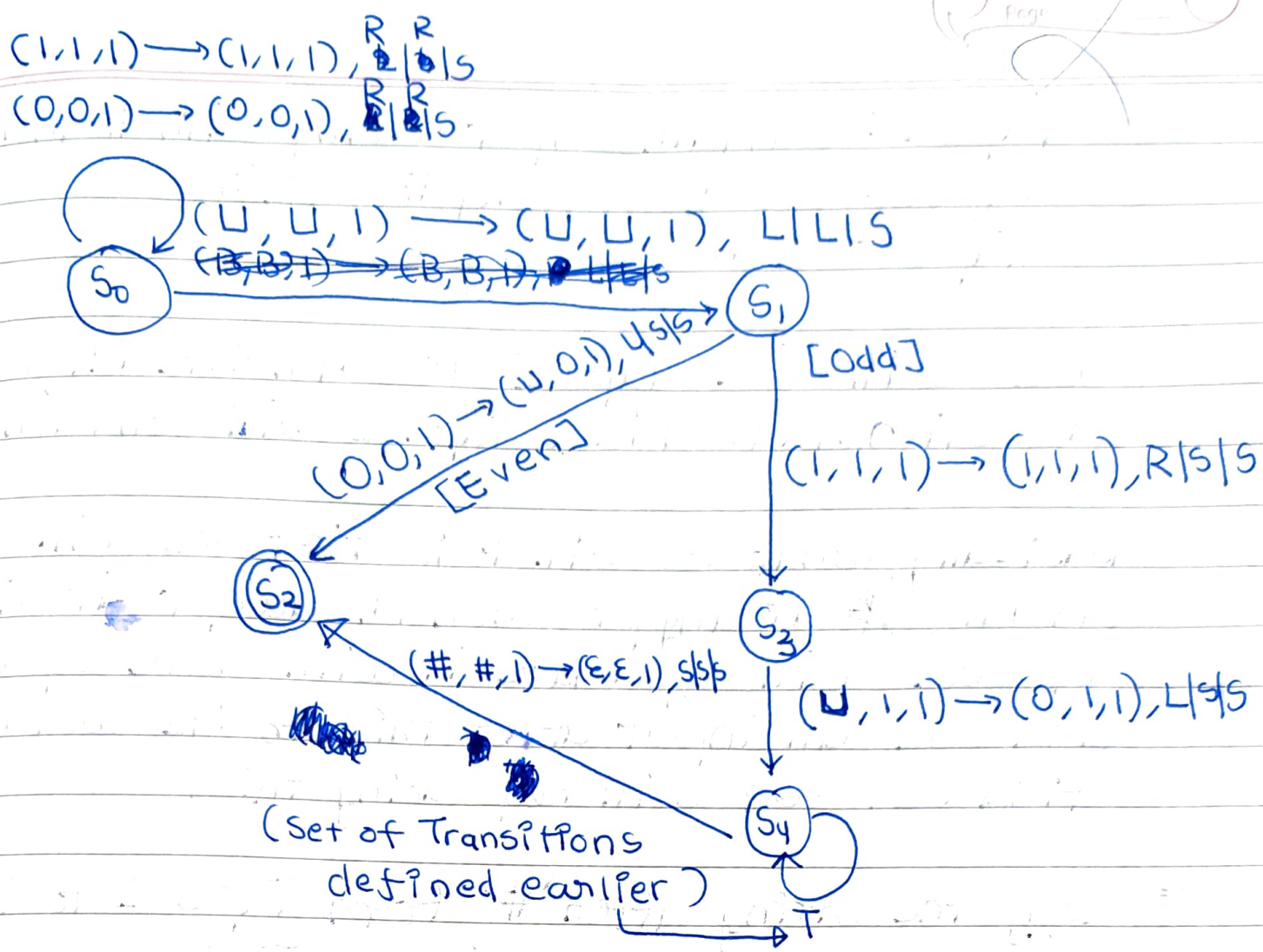
Now Transitions, ($\varepsilon \longrightarrow$ Repⁿ No Writeup)

① $(1,1,1) \longrightarrow (\varepsilon, \varepsilon, \varepsilon)$, L|L|S
② $(1,0,1) \longrightarrow (0, \varepsilon, 1)$, L|L|S
③ $(0,1,1) \longrightarrow (0, \varepsilon, 1)$, L|L|S
④ $(1,1,0) \longrightarrow (0, \varepsilon, 1)$, L|L|S
⑤ $(0,0,1) \longrightarrow (1, \varepsilon, 0)$, L|L|S
⑥ $(1,0,0) \longrightarrow (1, \varepsilon, 0)$, L|L|S
⑦ $(0,1,0) \longrightarrow (1, \varepsilon, 0)$, L|L|S
⑧ $(0,0,0) \longrightarrow (0, \varepsilon, 0)$, L|L|S

Defining these set of Transition as Set T

Now as Output Tape 1 → Contain Output which is 3x+1 in this Case

We will provide State Diagram ~~for~~ for the Turing Machine described as a Solution for this problem. ↓


(1,1,1) → (1,1,1), R|R|S
(0,0,1) → (0,0,1), R|R|S
(U, U, 1) → (U, U, 1), L|L|S
S0
S1
(0,0,1) → (U,0,1), L|S|S
[Even]
[Odd]
(1,1,1) → (1,1,1), R|S|S
S2
S3
(#,#,1) → (ε,ε,1), S|S
(U,1,1) → (0,1,1), L|S|S
S4
(Set of Transitions defined earlier)
T

Ans 7)

1.) $R_1 \cup R_2 \longrightarrow$ Recursive

Both R_1 & R_2 are Recursive ∴ ∃ Turing Machine M_1 & M_2 that decide R_1 & R_2 respectively.

Construct Turing Machine M than on input w:

→ Runs M_1 on w. If M_1 accepts M accepts.

→ If M_1 rejects, run M_2 on w. If M_2 accepts, M accepts

→ If both machine reject, M reject.

Since both machines halt on all inputs, M will also halt on all inputs meaning $R_1 \cup R_2$ is recursive.

2.) $RE_1 \cup RE_2 \longrightarrow$ Recursive Enumerable

Both RE_1 & RE_2 are recursive enumerable ∴ ∃ Turing Machine M_1 & M_2 that enumerate RE_1 & RE_2 respectively.

Conside M that on input w:

→ Simultaneously run M_1 & M_2 in parallel

→ If M_1 accepts w, then M accepts w.

→ If M_2 accepts w, then M accepts w.

If either M_1 or M_2 accepts, M will accept & Since Turing Machine can run indefinitely, M is guranteed to accept if w is in $RE_1 \cup RE_2$, Thus $RE_1 \cup RE_2$ is recursively enumerable.

3) $R_1 \cup RE_2$ ⟶ Recursively Enumerable

Consider M than on Input w:

→ Run M_1 & on w

→ If M_1 accept, M accepts

→ If M_1 rejects, run M_2 & (the enumerator for R_2) in Parrallel

→ If M_2 accepts w at any point, the M accepts.

Since M will accept if w is in either R_1 or RE_2 it recognizes $R_1 \cup RE_2$ but may run indefinately

∴ this is Recursively enumerable

4) $R_1 \cap RE_2$ ⟶ Recursively Enumerable

Construct a Turing Machine M that on input w:

→ Run M_1 on w

→ If M_1 rejects, M rejects

→ If M_1 accepts, run M_2 in parallel

→ If M_2 accept then M accept.

The construction works because M will only accept w if both M_1 accepts & M_2 accepts. Since M_2 may run indefinitely but M_1 always halts, the intersection $R_1 \cap RE_2$ is recursively enumerable.

Ans 8.) We are known that if S belong to either Language no prefix of S belongs to either Language.
What My Interpretation of this is Let suppose a string is accepted by M_1 & then any prefix can not be accepted by M_1 or M_2 (that any prefix of S can not be of type S_1 or S_2).

M_1 ⟶ accepts string of type S_1
M_2 ⟶ " " " " S_2
M_3 ⟶ will partition string S into S_1 & S_2
(basically in may case

S_a ⟶ M_3 ⟶ $\$ S_1 \$ S_2 \$ S_{31}$ - -- --

It has 3 Tapes.

Tape 1 : Contain input string with # in beg
Tape 2 : " only #
Tape 3 : " " "

We will use Tape 2 to stimulate M_1 & Tape 3 to stimulate M_2. Tape 1 will contain output

Basically what we will do is first we will run Input on M_1, at any pt when M_1 accepts it will mark S_1 we will separate it with some separator (#) and use all input after this to M_2 which will find S_2. We will keep doing this.

⇒ Copy the string after '#' from Tape 1 to Tape 2 when M_1 accept the string S_1 then add a new '#' in Tape 1 & shift other elements. Also replace previous '#' with $. Now copy string after '#' from Tape 1 to Tape 3. when M_2 accept the string S_2 then add a new '#', shift & replace any previous '#' with '$'. Repeat these steps untill a complete string is done processing.

Explaining all this with the help of example,

S_1 = chipi

S_2 = chapa

$S = S_1 S_2 S_1 S_2 \ldots$

= chipichapa chipichapa - - - -

Initially ⇒ Tape 1 = #chipichapa chipichapa...
Tape 2 = #
Tape 3 = #

Copy Input after # to Tape 2 & stimulate it on M1

Tape 1 = # chipi chapa chipi chapa - - -
Tape 2 = # chipi chapa chipi - - -
Tape 3 = #

Put # in that place & Replace Previous # with $.

Tape 1 = $ chipi # chapa chipi chapa - - -
Tape 2 = # chipi chapa chipi - - -
Tape 3 = # ~~chapa~~

Copy String after # in Tape 1 to Tape 3 & stimulate it on M2.

Tape 1 = $ chipi ~~chapa$~~ # chapa chipi chapa
Tape 2 = # chipi chapa chipi - - -
Tape 3 = # chapa chipi chapa - -

Put # in that place 2 Replace previous # with $

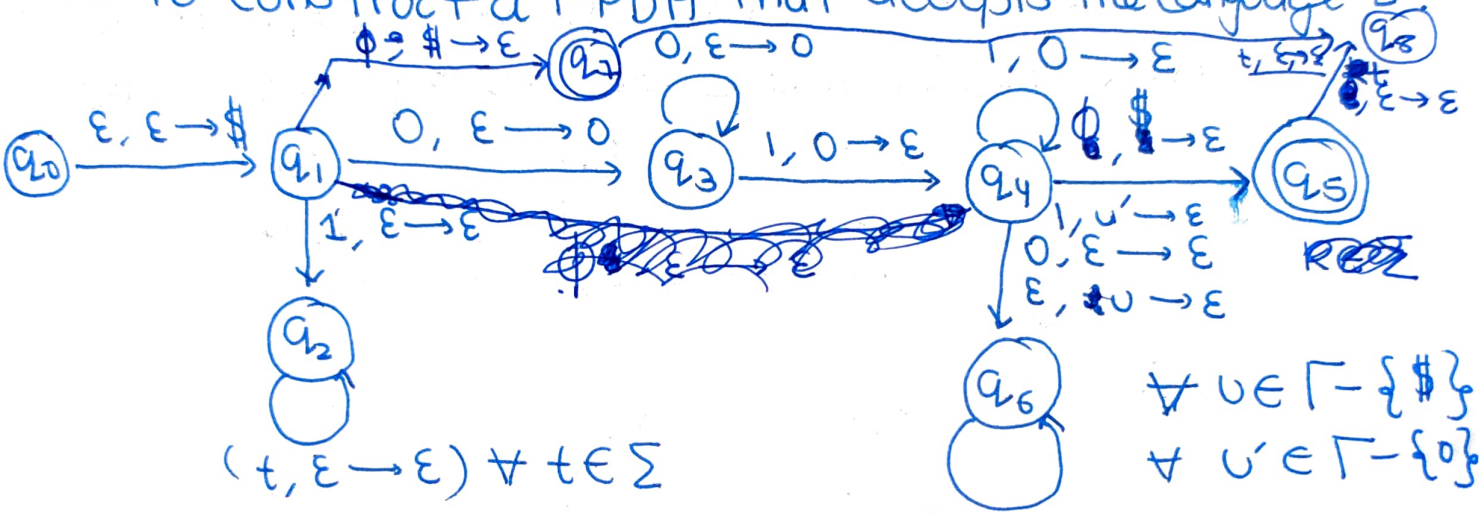
Tape 1 = $ chipi $ chapa # chipi chapa - -
Tape 2 = # chipi chapa - - -
Tape 3 = # chapa chipi chapa - - -

Now alternate between these two methods.

final state:
Tape 1 = $ chipi $ chapa $ chipi - - - - - #

Ans 9) $L = \{0^n 1^n \mid n \geq 0\}$

We have to construct a FPDA that accepts the language L



$\varepsilon \longrightarrow$ Epsilon [Nothing]

$\phi \longrightarrow$ End of string

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

Accept state = q_5, q_7 | Start state q_0

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

$q_8 \longrightarrow$ Reject State.

Ans 10) $G = (V, \Sigma, R, S)$

$V = \{A, B\}$, $\Sigma = \{0, 1\}$, $S = A$

R:-

$A \longrightarrow 0A0 \mid 0B1 \mid 1A0 \mid 1B1$

$B \longrightarrow 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid 1 \mid 0 \mid \varepsilon$

G is the required CFG that generates the given collection of strings, B.

$B = \{uv \mid u \in \Sigma^*, v \in \Sigma^* \mid \Sigma^*, 2\ |u| \geq |v|\}$

The string is contructed by always adding either a 0 or a 1 to either side, so we are keeping track of the midway point of our string. Termination canbe done by selecting B with 0 occurs iff we add 1 to the second half and could be end by picking $0 \mid 1 \mid \varepsilon$.