Unit :- 1

Automata Fundamentals

PAGE NO :
DATE : / /

Introduction to formal proof :

A formal proof refers to a systematic & proper demonstration of properties, theorems or statements related to formal languages, algorithms & other computational models.

We have 4 methods for this.

1) Deductive proof
2) Reduction to definition
3) Other theorem forms
4) theorems that appear not to be if then statemen

(1) Deductive Proof :

It consists of a sequence of statement whose truth leads us from some initial statement called hypothesis

ex :- 1 if A then B. ← Here B is deduced from A

ex :- 2 If $x \geq 4$ then $2^x \geq x^2$

Here hypothesis statement is : $x \geq 4$
& conclusion statement is : $2^x \geq x^2$

Step: 1

if $x=1$ then $x \neq 4$ & $x \neq 4$

So $(2^1 \geq 1^2) = (2 \geq 1)$ Can be true

Same can be for $x=2$

Step: 2

if $x=3$ then $x \neq 4$ & $x \neq 4$

So $(2^3 \geq 3^2) = (8 \geq 9)$ will be false

Step: 3

if $x=4$ then $x \geq 4$ condition will be true,

Here, now,

$(2^4 \geq 4^2) = (16 \geq 16)$ will be true.

Conclusion can be:

If hypothesis is true, then Conclusion is tru

(2) Reduction to Definition:

If not clear in hypothesis to start proof. then convert all terms in the hypothesis to their definitions.

ex: Let 'S' be a finite subset of some infinite set 'U' & 'T' be the complement of 'S' w.r.t. 'U'. Then 'T' is infinite.

Let's Consider:

$U = \{1, 2, 3, 4 \ldots\}$
$S = \{1, 2, 3\}$
$T = \{4, 5, 6, 7 \ldots\}$

Actual statement	New statement
→ 'S' is finite	→ $\|S\| = x$, x is an integer
→ 'U' is infinite	→ For no integer value y. $\|U\| = y$
→ 'T' is complement of 'S'	→ $S \cup T = U$ $S \cap T = \phi$

PAGE NO.:
DATE: / /

Step: 1

$\|S\| = x$

$\|T\| = z$ (proof by contradiction)

Step: 2

$S \cup T = U$ (where U is infinite)

Step: 3

$x + z = U$

But this is false w.r.t. step-2.

Step: 4

From the contradiction of step-2 & step-3 we can say that

'T' is infinite.

(3) Other theorem forms:

There are 2 types of forms.

1) if - then
2) if & only if

(1) If - then form:

if H then C

Here if there is H then ⇒ C will be as..

- H implies C
- H only if C
- C if H
- whenever H holds C follows.

ex: if $x \geq 4$ then $2^x \geq x^2$

- $x \geq 4$ implies $2^x \geq x^2$
- $x \geq 4$ only if $2^x \geq x^2$
- $2^x \geq x^2$ if $x \geq 4$
- whenever $x \geq 4$ holds $2^x \geq x^2$ follows

(2) If & only if form:

It has 2 forms

- x if & only if y $(x \leftrightarrow y)$
- x is equivalent to y $(x \equiv y)$

or

y is equivalent to x $(y \equiv x)$

$x \leftrightarrow y$ / $x \equiv y$

step:1 if-part : Asume y to prove x

step:2 Only if part : Assume x to prove y

ex: Let x be the real number. Then $\lfloor x \rfloor = \lceil x \rceil$ if and only if x is an integer.

step:1 if-part

Assume x is an integer. like 81

$\lfloor 81 \rfloor = 81$ ——— (1)

$\lceil 81 \rceil = 81$ ——— (2)

from (1) & (2) we can prove that

$\lfloor x \rfloor = \lceil x \rceil$

Step: 2 Only if part

Assume $\lfloor x \rfloor = \lceil x \rceil$ to prove x is an integer.

Let's take $x = 95.5$

$\lfloor 95.5 \rfloor = 95 \Rightarrow \lfloor x \rfloor \leq x$ ——— (1)

$\lceil 95.5 \rceil = 96 \Rightarrow \lceil x \rceil \geq x$ ——— (2)

we have assumed that

$\lfloor x \rfloor = \lceil x \rceil$ ——— (3)

Put the value of (3) in (1)

$\lceil x \rceil \leq x$ ——— (4)

By comparing (2) & (4)

$\lceil x \rceil \geq x$ & $\lceil x \rceil \leq x$ are contradiction,

Thus x should be an integer

(4) Theorems not to be in if-then statement

In this type hypothesis is not there.

Eg:

→ $\sin^2\theta + \cos^2\theta = 1$

→ $a^2 + b^2 = c^2$

★ Inductive Proof:-

1) Basis: Prove $S(i)$ for an initial value i, $i=1$

2) Inductive: • Assume $S(k)$ is true.
• Prove that $S(k+1)$ is also true

To prove $S(n)$

Eg: $1+4+7+\ldots+(3n-2) = \frac{n(3n-1)}{2}$ $S(n)$

Basis: $n=1$

$1 = \frac{n(3n-1)}{2} = \frac{1(3-1)}{2} = \frac{2}{2} = 1$ Proved

Inductive:

$n = k$

$\therefore 1 + 4 + 7 + \dots + (3k-2) = \frac{k(3k-1)}{2}$ ——— (1)

Assume that (1) is true.

now, Consider $n = k+1$

$\underbrace{1 + 4 + 7 + \dots (3k-2)}_{(1)} + (3(k+1)-2)$

$= \frac{(k+1)(3(k+1)-1)}{2}$

$\therefore \frac{k(3k-1)}{2} + 3k + 3 - 2 = \frac{(k+1)(3(k+1)-1)}{2}$

$\therefore \frac{k(3k-1)}{2} + (3k+1) = \frac{(k+1)(3k+3-1)}{2}$

$\therefore \frac{k(3k-1)}{2} + (3k+1) = \frac{(k+1)(3k+2)}{2}$

$\therefore \frac{k(3k-1) + 2(3k+1)}{2} = \frac{(k+1)(3k+2)}{2}$

$\therefore 3k^2 - k + 6k + 2 = 3k^2 + 2k + 3k + 2$

$\therefore \boxed{3k^2 + 5k + 2 = 3k^2 + 5k + 2}$ Proved.

Thus S(n) is proved

★ Finite Automata :-

Suppose $S = \{x, y\}$

Then

Language

- Finite
  - L_1 = Length 2
  - $\{xx, xy, yx, yy\}$
- Infinite
  - L_2 = at least one 'x'
  - $\{x, xx, xxx, xy, xyx, xyy, \ldots\}$

Automata is a model which is used to check that either given string can be considered as a part of lenyuage or not.

There are 2 types of finite automata

1) Deterministic finite Automata (DFA)
2) Non-deterministic finite Automata (NFA)

✱ DFA:

If the machine read an input string one symbole at a time then it is called DFA. There is one path for specific input from current state to the next state.

It is not accepting null input.

DFA has some taminologies like:

Q = Set of all finite states

Σ = Set of alphabets

δ = Transition $(Q \times \Sigma)$

q_0 = Starting state

F = Set of final states.

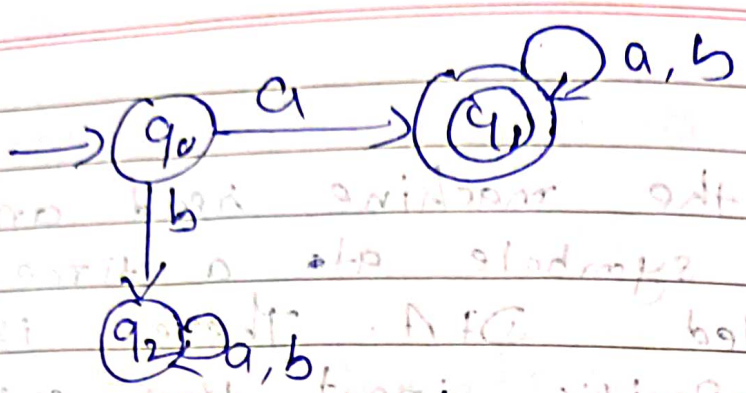
Note that $F \subseteq Q$

Ex:

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

Construct DFA for language where string starting with a a

{a, ab, aa, aba, aaa, ~~ba~~}



Here,

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

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

$Q \times \Sigma \rightarrow Q$ also $Q * \Sigma \rightarrow F$

$\{q_0, q_1, q_2\} \times \{a, b\} \rightarrow \underbrace{\{q_0, q_1, q_2\}}_{Q}$ or $\underbrace{\{q_1, q_2\}}_{F}$

$q_0 a \rightarrow q_1$

$q_0 b \rightarrow q_2$

$q_1 a \rightarrow q_1$

$q_1 b \rightarrow q_1$

$q_2 a \rightarrow q_2$

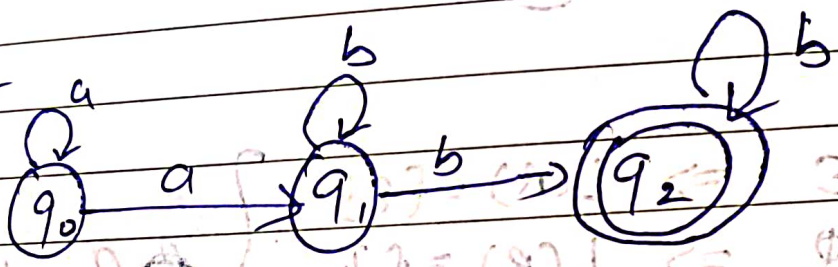
$q_2 b \rightarrow q_2$

✱ NFA :-

for any input in transition if there are multiple possible next states then it is called NFA/NDFA. NFA can also reach or accept null value.

It also have Q, Σ, q_0, δ & F same as DFA.

Ex:-



Here suppose,

$$Q \times \Sigma \rightarrow Q$$

$$\therefore \{q_0, q_1, q_2\} \times \{a, b\} \rightarrow \{q_0, q_1, q_2\}$$

$q_0 a \rightarrow \{q_0, q_1\}$

$q_0 b \rightarrow \phi$

$q_1 a \rightarrow \phi$

$q_1 b \rightarrow \{q_1, q_2\}$

$q_2 a \rightarrow \phi$

$q_2 b \rightarrow \{q_2\}$

Unit - 2

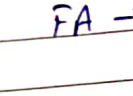
Regular Expressions & Languages

✦ Regular Expression:-

Regular expression is used to represent a regular language which can be accepted by finite automata

Ex:- $L = \{\varepsilon, a, aa, aaa, aaaa \ldots\}$
$= a^*$ ← Regular expression (means ∪ or any no of 'a')

FA →



If

$R = \varepsilon \Rightarrow L(R) = \{\varepsilon\}$
$R = \phi \Rightarrow L(R) = \{\}$
$R = a \Rightarrow L(R) = \{a\}$

} Premitive

- Union of RE is also regular
- Concatenation of RE is also regular
- if 'a' is regular then a^* is regular
- if R is regular then (R) is also regular

✦ RE for Finite Language:

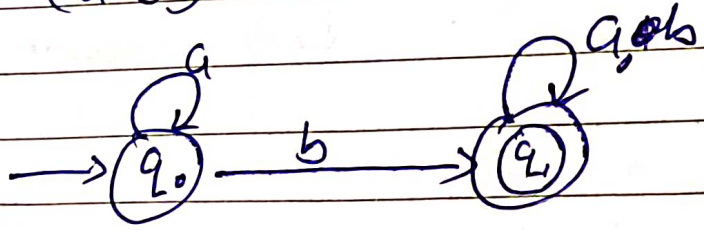
Language	RE
No string $\{\}$	ϕ
Length 0 $\{\varepsilon\}$	ε / λ
length 1 $\{a, b\}$	$(a+b)^*$
length 2 $\{aa, ab, bb, ba\}$	$(aa + ab + bb + ba) = a(a+b) + b(a+b)$ $= (a+b)(a+b)$

for $\Sigma(a,b)$

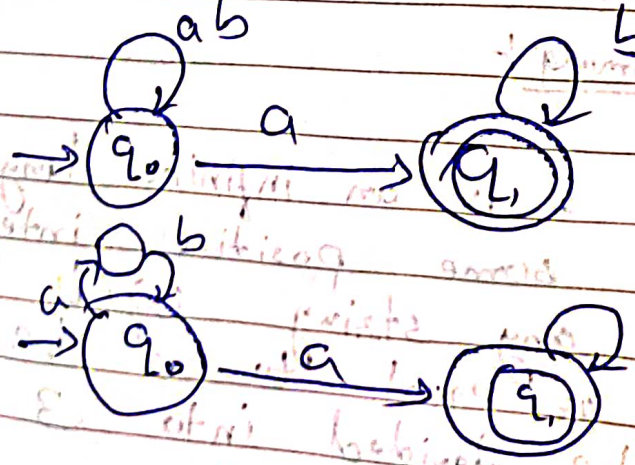
Language	RE
length 3	(a+b) (a+b) (a+b)
At most 1 {ε, a, b}	(ε+a+b)
At most 2	(ε+a+b) (ε+a+b)

★ RE To NFA :

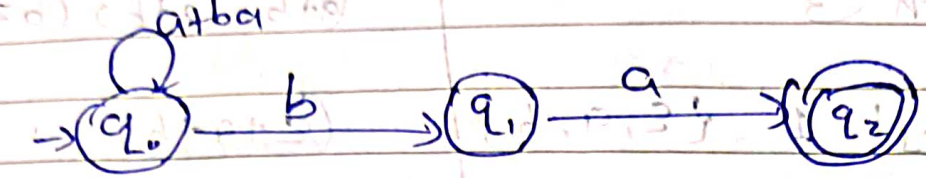
1) $a^*b\,(a+b)^*$

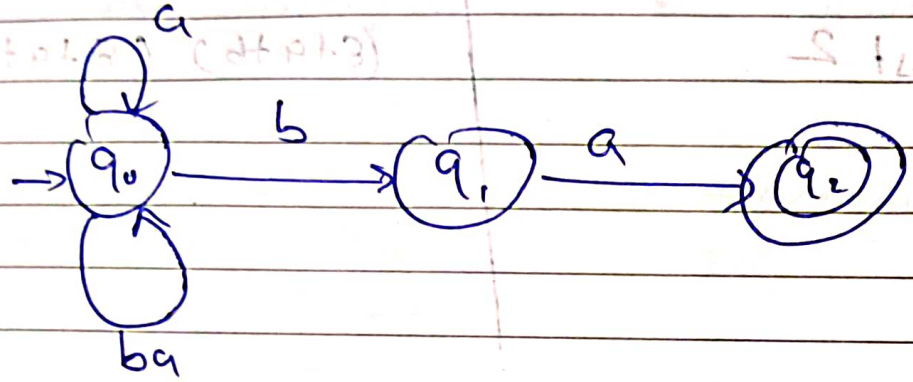


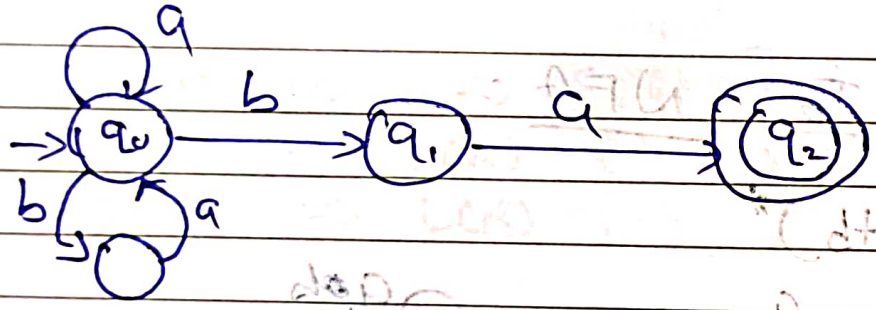
2) $(ab)^*ab^*$



3) $(a+ba)^* ba$







★ Proving Languages not to be regular:

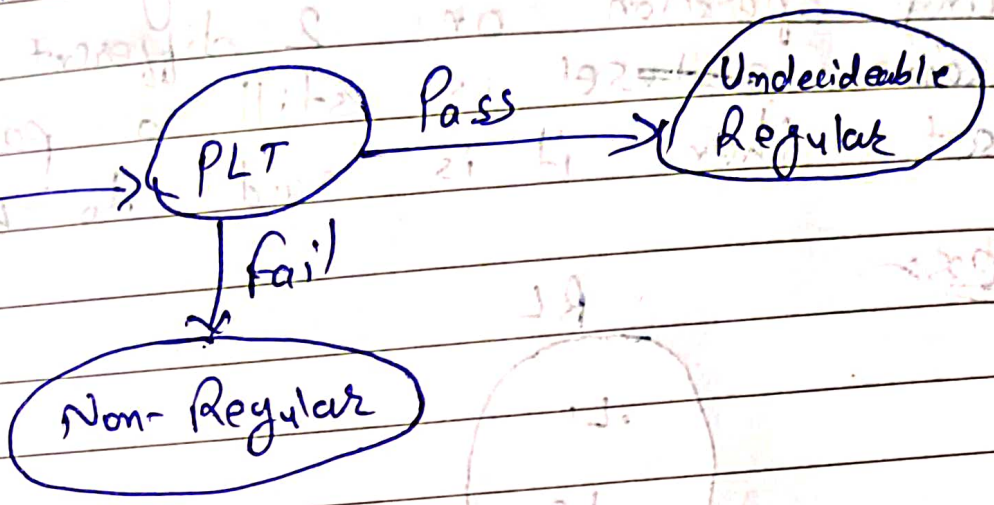
- **Pumping Lemma:**

If L is an infinite language then there exists some positive integer 'n' such that any string $w \in L$ has length greater than or equal to 'n' i.e. $|w| \geq n$ then w can be devided into 3 parts, $w = xyz$ satisfy following conditions.

i) for each $i \geq 0$, $xy^iz \in L$

ii) $|y| > 0$

iii) $|xy| \leq n$



Ex :-

$L = a^n b^{2n}$, $n \geq 0$

for $n = 2$

$\omega = \underset{x}{aa}\ \underset{y}{bb}\ \underset{z}{bb}$

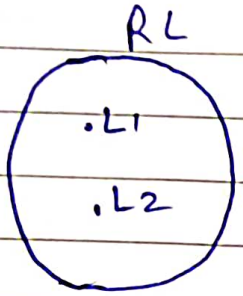
for $i = 2$ $\quad \omega = \underset{x}{aa}\ \underset{y}{bbbb}\ \underset{z}{bb} \notin L$

Thus, this is not satisfying pumping Lemma theorem. that's why this language is not Regular.

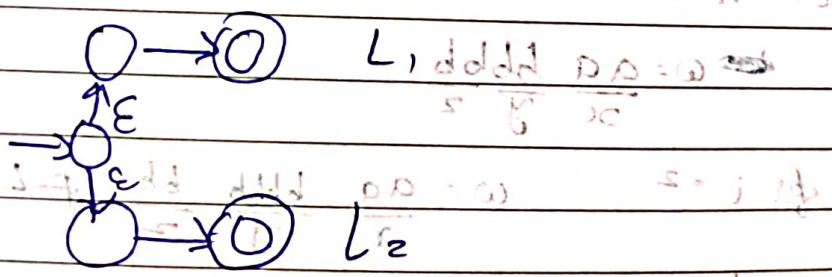
★ Closure Properties of Regular Languages:

If output we got by applying any operation on 2 different parts of same ~~state~~set is still a part of the set then it is said to be closed.

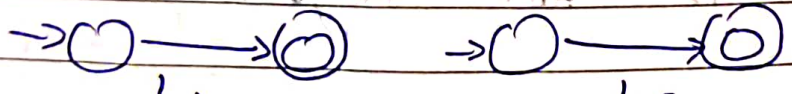
ex:-



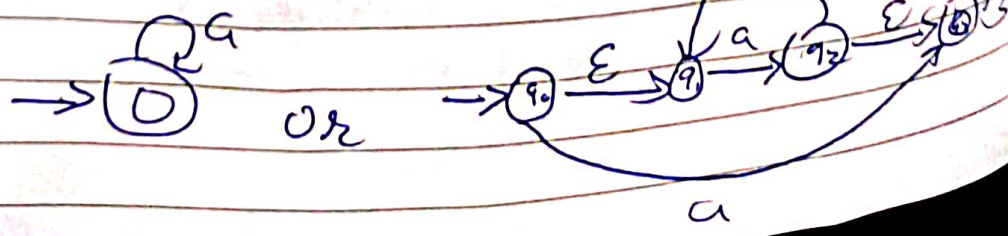
1) $L_1 \cup L_2$ will be closed



2) $L_1 \cdot L_2$ will be closed



3) a^*



PAGE NO. :
DATE : / /

4) Complementation $\bar{L} = \Sigma^* - L$

Convert initial state to final state & final state to initial state. This will be DFA too.

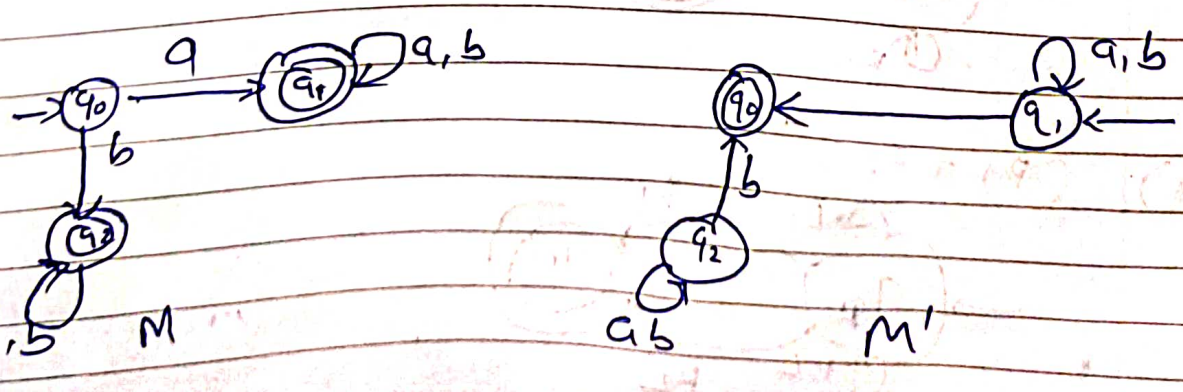
5) Intersection $L_1 \cap L_2$

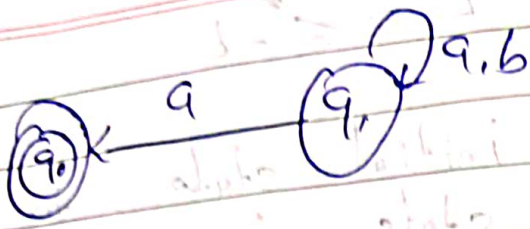
6) Difference $L_1 - L_2$

7) Reverse of Language L^R

- Make initial state of M as final state of M'
- final state of M become initial state of M'
- Reverse the direction of edges of M to make M'
- No change in loop & remove unnecessary states

ex :- L_1 = {set of all strings over (a,b) starting with a}
= $a(a+b)^*$

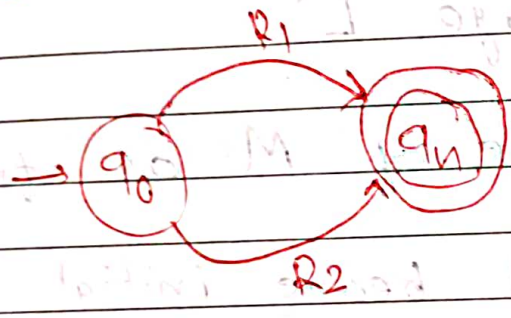




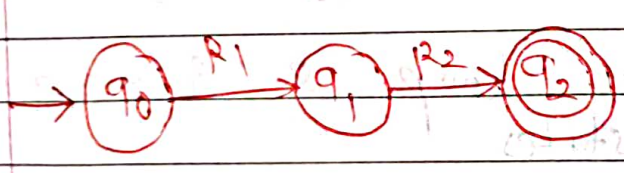
M' (as q_2 is unnecessary state)

Basic Regular Expression

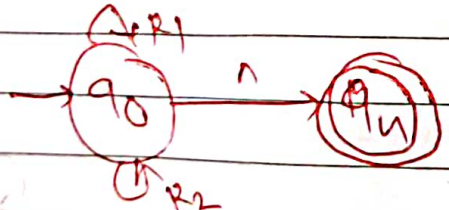
1) $(R_1 + R_2)$



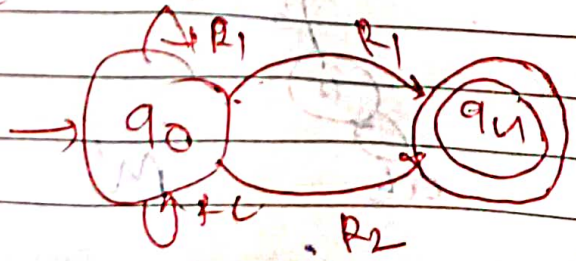
2) $R_1 R_2$



3) $(R_1 + R_2)^*$



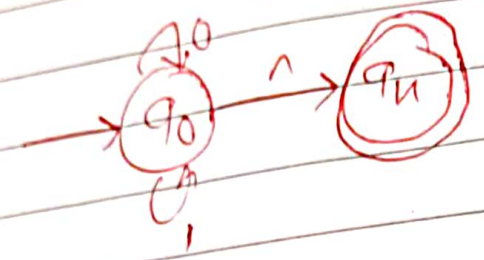
4) $(R_1 + R_2)^+$



* A string can contain all combination of 1's or 0's.

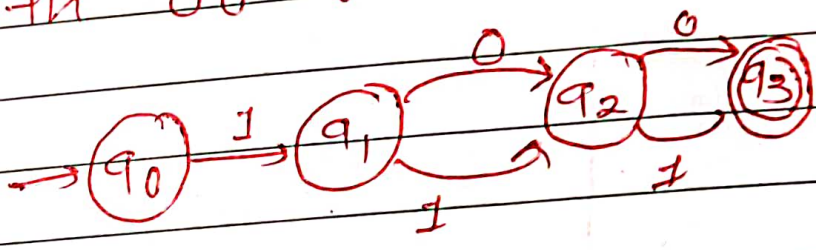
(NFA)

$(0+1)^*$

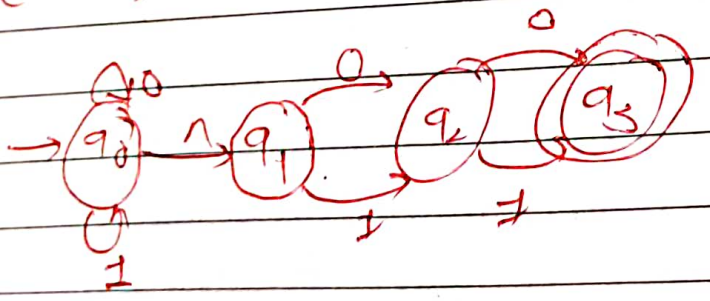


* A string accept all the string which ~~begin and~~ end with 00 or 11

$1\,(00+11)$



$(0+1)^*\,(00+11)$

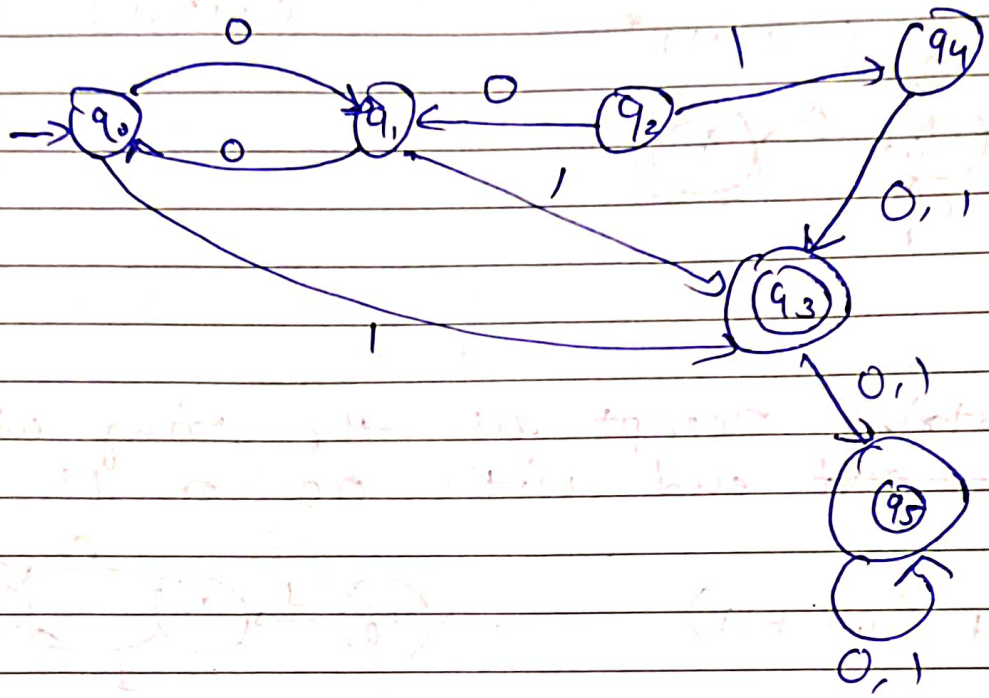


* W.d a r.e where $\Sigma = \{a, b\}$ such that the 3rd character from right end of the string is always a.

$(a+b)\,a\,(a+b)\,(a+b)$

PAGE NO.:
DATE: / /

✱ Minimization of DFA :-



Step :- 1

state	I/P = 0	I/P = 1
q_0	q_1	q_3
q_1	q_0	q_3
q_2	q_1	q_4
q_3	q_5	q_5
q_4	q_3	q_3
q_5	q_5	q_5

PAGE NO. :
DATE : / /

Step: 2

Here, q_2 & q_4 are unreachable states so we will remove it.

state	I/P = 0	I/P = 1
q_0	q_1	q_3
q_1	q_0	q_3
q_3	q_5	q_5
q_5	q_5	q_5

Step :-3

make table for non-final state q_0 & q_1

state	I/P = 0	I/P = 1
q_0	q_1	q_3
q_1	q_0	q_3

Both o/p states are different so ~~it~~ we will keep it as it is.

Step :- 4

Make table for final states q_3 & q_5

state	I/P = 0	I/P = 1
q_3	q_5	q_5
q_5	q_5	q_5

Both o/p is q_5 so we can remove q_5 & Consider 1 final state q_3

Step: 5

Make table for final state

State	I/P = 0	I/P = 1
q_3	q_3	q_3

Here, q_5 is removed so self loop of q_5 will be applicable for q_3

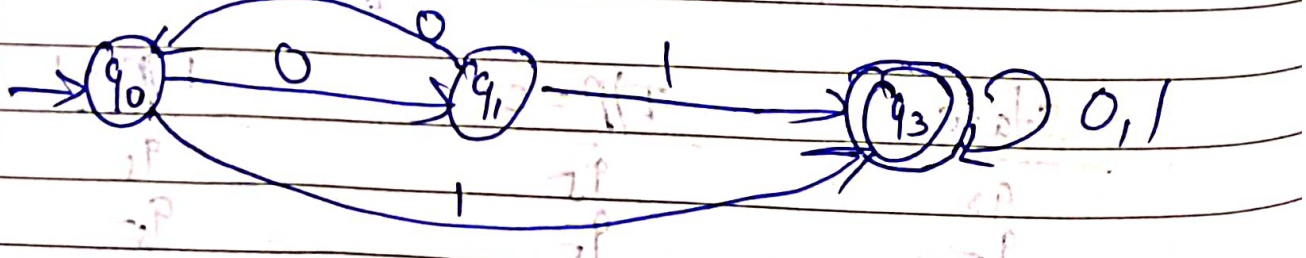
State: 6

Merge table of final & non-final states

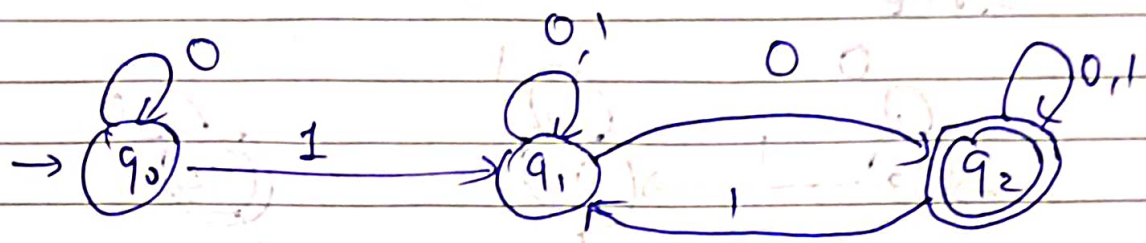
state	I/P=0	I/P=1
q_0	q_1	q_3
q_1	q_0	q_3
q_3	q_3	q_3

State: 7

Construct DFA



★ NFA to DFA :-



Step :- 1

State	I/P = 0	I/P = 1
q_0	q_0	q_1
q_1	$\{q_1, q_2\}$	q_1
q_2	q_2	$\{q_1, q_2\}$

Step :- 2

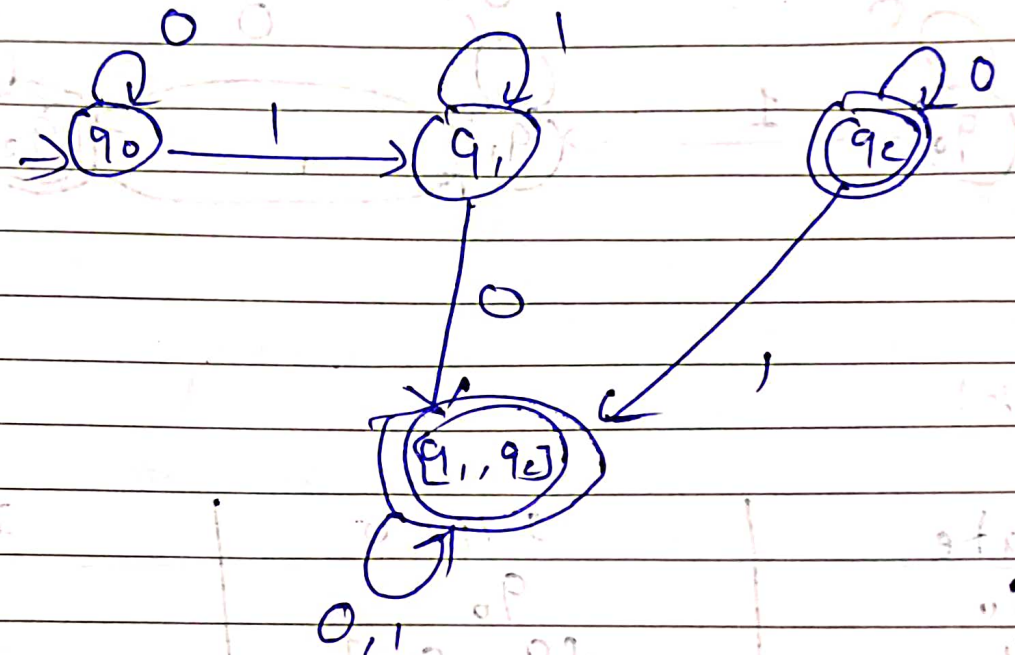
Consider $\{q_1, q_2\}$ as a single state.

$$\delta(\{q_1, q_2\}, 0) = \delta(q_1, 0) \cup \delta(q_2, 0)$$
$$= \{q_1, q_2\} \cup \{q_2\}$$
$$= \{q_1, q_2\}$$

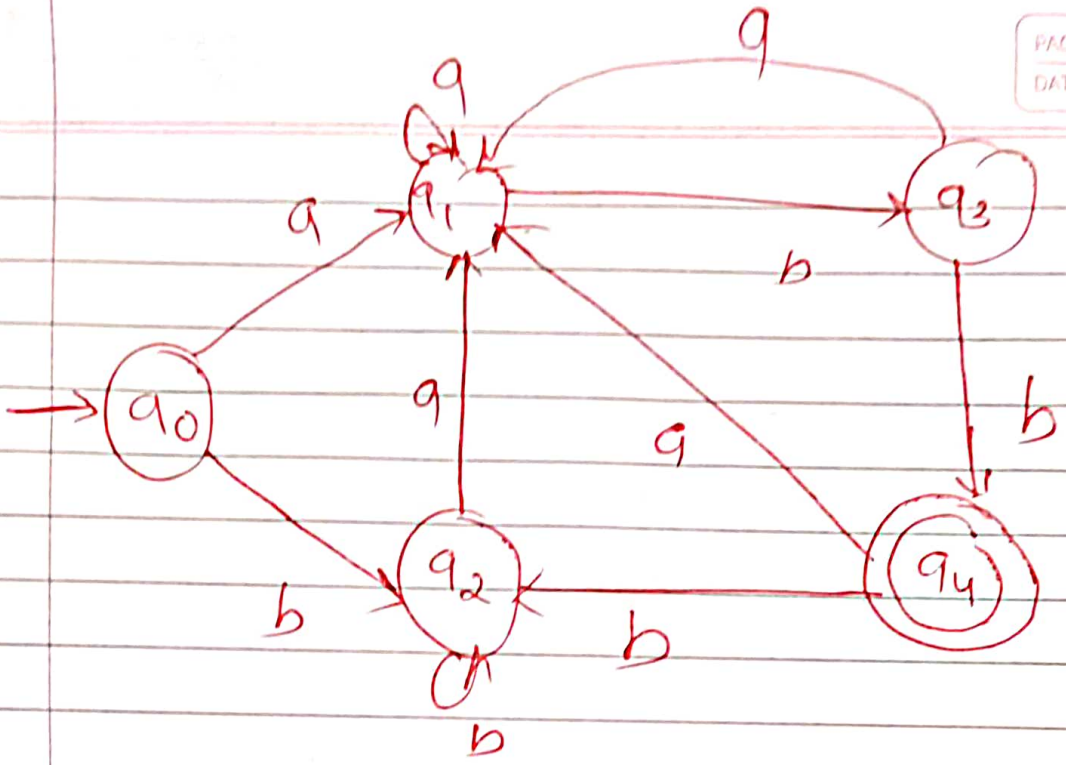
$$\delta(\{q_1, q_2\}, 1) = \delta(q_1, 1) \cup \delta(q_2, 1)$$
$$= \{q_1\} \cup \{q_1, q_2\}$$
$$= \{q_1, q_2\}$$

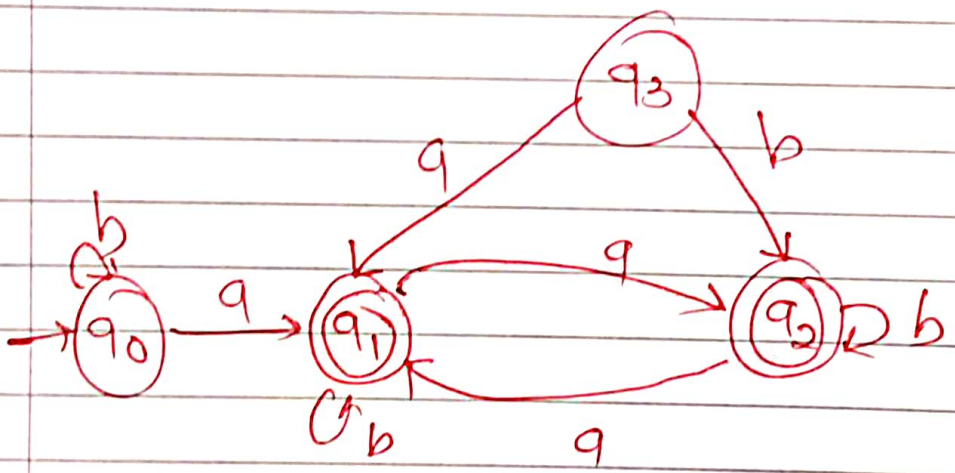
step:2

Consider $[q_1, q_2]$ as a final state



State	I/P = 0	I/P = 1
q_0	q_0	q_1
q_1	$[q_1, q_2]$	q_1
q_2	q_2	$[q_1, q_2]$
$[q_1, q_2]$	$[q_1, q_2]$	$[q_1, q_2]$


q
q
q
q
q
b
q
b
q
b
b
b
q_0
q_1
q_2
q_3
q_4


q_3
q
b
b
q
q
b
q
b
q_0
q_1
q_2

Unit : 3

"Context Free Grammar & Languages"

PAGE NO
DATE / /

☆ CFG :-

Context free grammar is a formal grammar which is used to generate all possible strings in given formal language.

$$G = (V, T, P, S)$$

G → Grammar
T → Finite set of Terminal symbols
P → Set of Production Rules
S → start symbol
V → Finite set of non-terminal symbols

ex :-

$$L = \{ wcw^R \mid w \in (a,b)^* \}$$

Production Rules

$S \rightarrow aSa$ ——— (1)
$S \rightarrow bSb$ ——— (2)
$S \rightarrow aSb$ ——— (3)
$S \rightarrow bSa$ ——— (4)
$S \rightarrow c$ ——— (5)

Generate, $S \Rightarrow abbcbba$

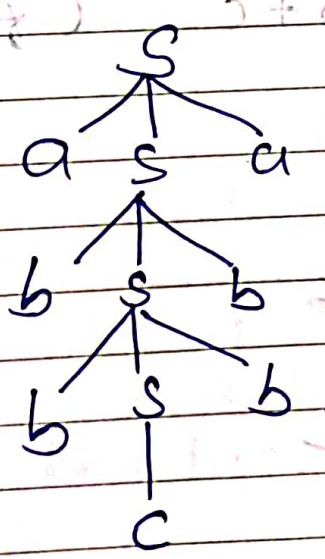
from (1)

$S \rightarrow aSa$
$S \rightarrow abSba$ (From (2))
$S \rightarrow abbSbba$ (from (2))
$S \rightarrow abbcbba$ (from (5))

★ Parse Tree :-

Parse tree is the graphical representation of the given production rules for a given context free grammar.

ex :- From example given above for cfg, Parse tree will be as follow



PAGE NO.:
DATE: / /

Ex :-

Production Rule:

$E \to E+E$ ——— (1)
$E \to E*E$ ——— (2)
$E \to a|b|c$ ——— (3)

Input : $a * b + c$

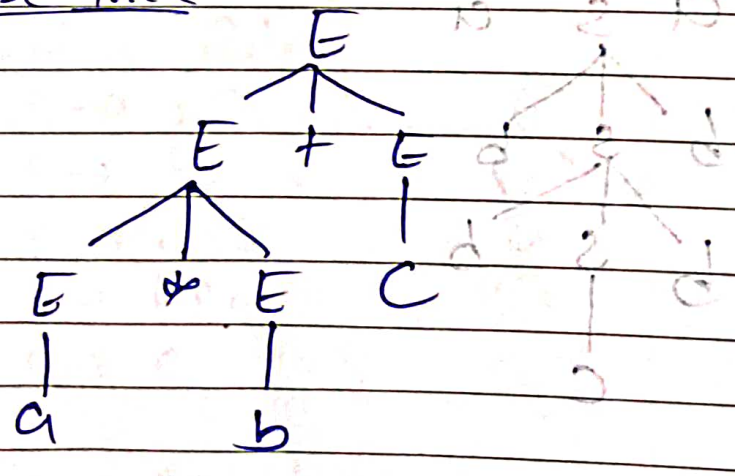
CFG :-

$E \to E+E$

$E \to E*E+E$ (from (2))

$E \to a*E+E$ (from (3))
$E \to a*b+E$ (from (3))
$E \to a*b+c$ (from (3))

Parse tree



Ambiguity in grammar & languages:

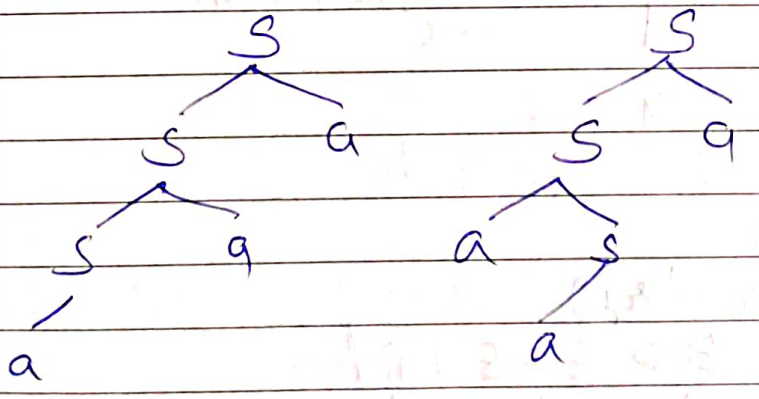
In ambiguous grammar there is more than one derivation for any word that belongs ~~thro~~ to grammar is possible

In unambiguous there is exactly one derivation for any word that belongs to grammar.

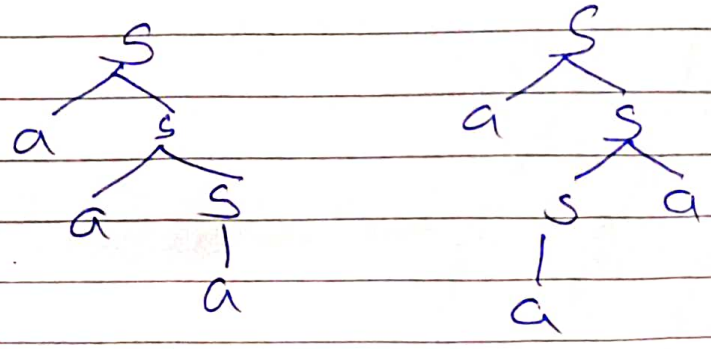
ex: G: $S \to Sa \mid aS \mid a$

w: aaa

Left most derivation



Right most derivation



Here we got 4 different parse trees for same grammar & word. That's why this is called ambiguous grammar.

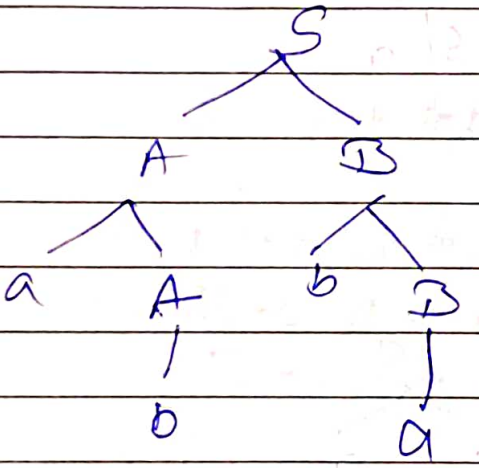
ex:-

CT: $S \to AB$

$A \to aA \mid b$

$b \to bB \mid a$

w: abba



this is only possible parse tree. That's why this is unambiguous grammar.

ex:- (For try)

CT: $E \to E+E \mid id$

w: id + id + id

PAGE NO :
DATE / /

* Conversion of Ambiguous to Unambiguous:

Ex: $E \rightarrow E+E \mid id$

w: id + id + id

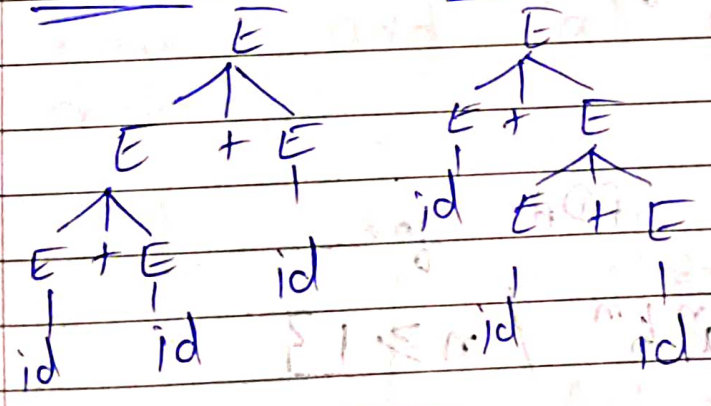
This is ambiguous grammar because here 2 trees are possible. To convert it to unambiguous grammar we can replace one node so another tree will not possible. Just like...

$E \rightarrow E+T \mid T$

$T \rightarrow id$

Before

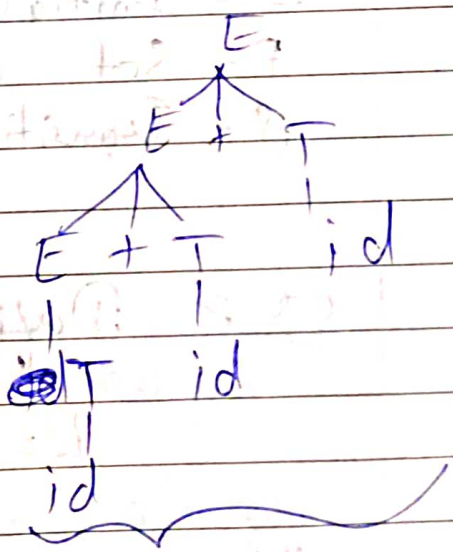
LMDT



RMDT

Ambiguous

After



Unambiguous

★ Pushdown Automata:

Pushdown Automata (PDA) is a finite automata with extra memory called stack which helps PDA to recognize CFL.

It has 7 touples.

Q = Set of states
Σ = Set of input symbols
Γ = Set of pushdown symbols (to push & pop from stack)
q_0 = Initial state
Z = Initial pushdown symbol
F = Set of final states
δ = Transition

ex :- Define PDA for

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

Here,

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

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

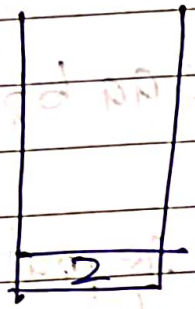
$$\Gamma = \{a, Z\}$$

Consider $n=3$ (for example only)

then string will be

aaabbb

Initially pointer will be on first a & there will be Z in stack.



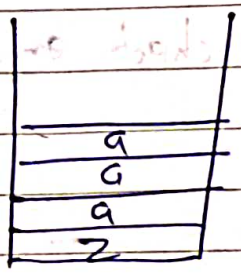
Here, Z is initial symbol there is no input symbol in stack. so we can also consider that stack is empty.

Now, add 'a' from pointer into stack.

Now,

string = aabbb

Add second 'a' from pointer to stack & repeat for next 'a' also.



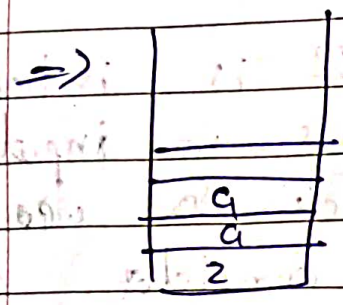
Now

string = aaabbb

Now according to language

No. of 'a' = No. of 'b'

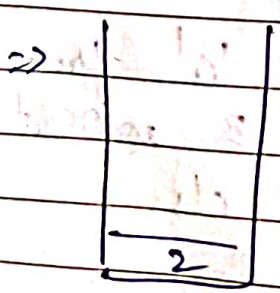
So to check language, we will pop each 'a' on every occurance of 'b'



String = aaa bbb

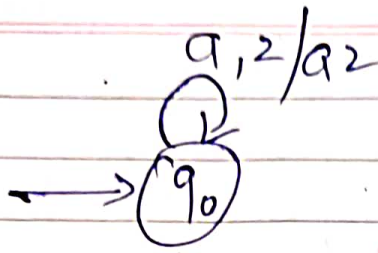
We will repeat process untill stack will be empty again

String = aaa bbb



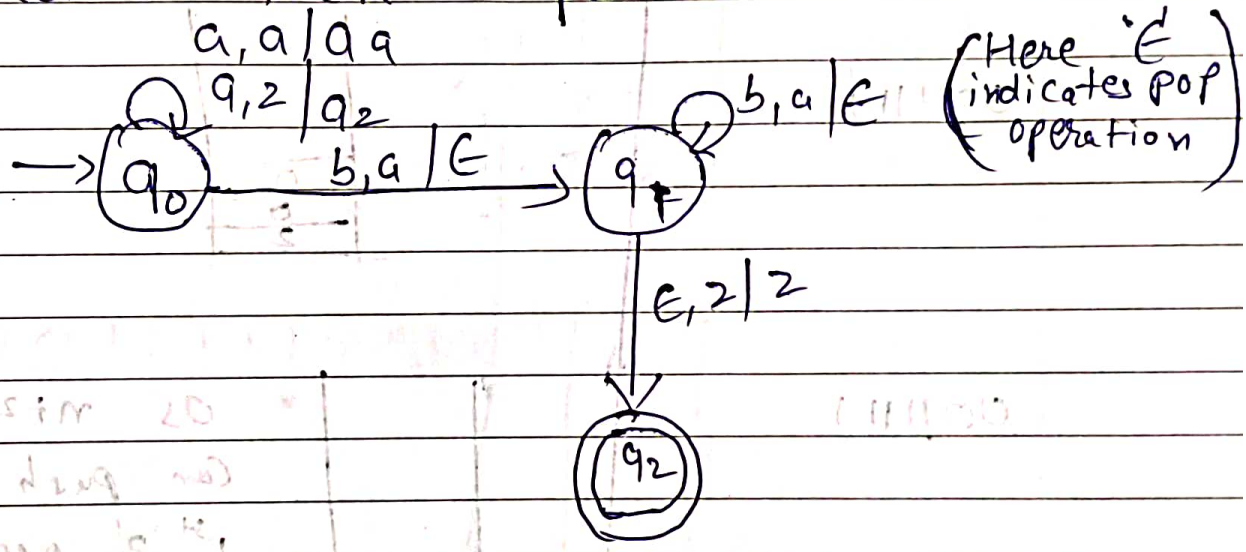
string = aaa bbb ε

(We can assume 'ε' to check empty stack)



Here, "a,2/a2" can be read as

if there is 'a' as an input & '2' is on the top of the stack then put 'a' on '2'



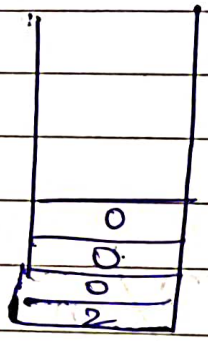
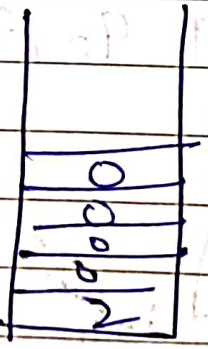
Ex :- $L = \{0^n 1^m / n \geq 1, m \geq 1, m > n+2\}$

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
$\Sigma = \{0, 1\}$
$\Gamma = \{0, 2\}$

Consider string where $n = 2$,
Thus we can assume $m = 5$
Or $m \geq 5$

string = 0011111

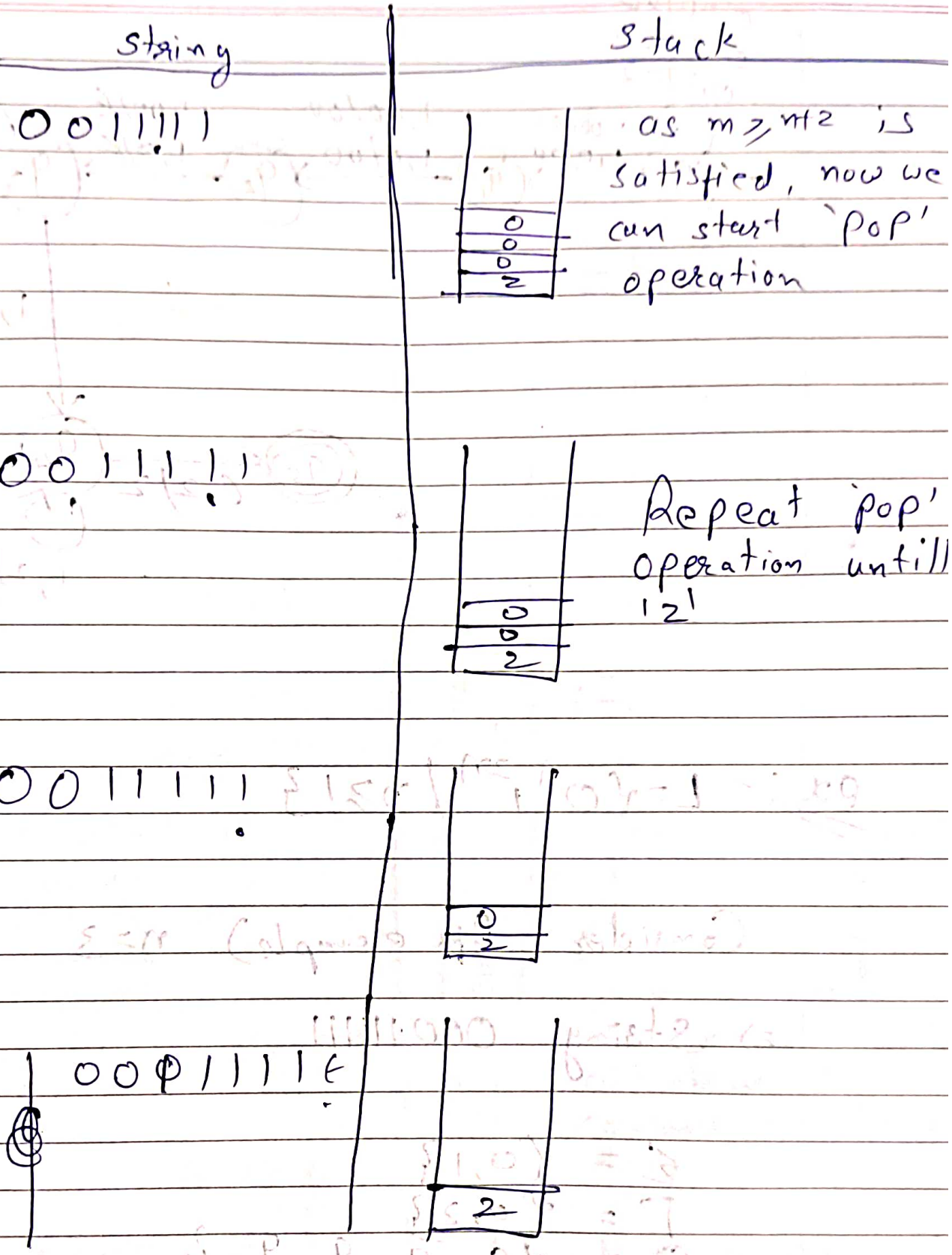
Pointer is on 0

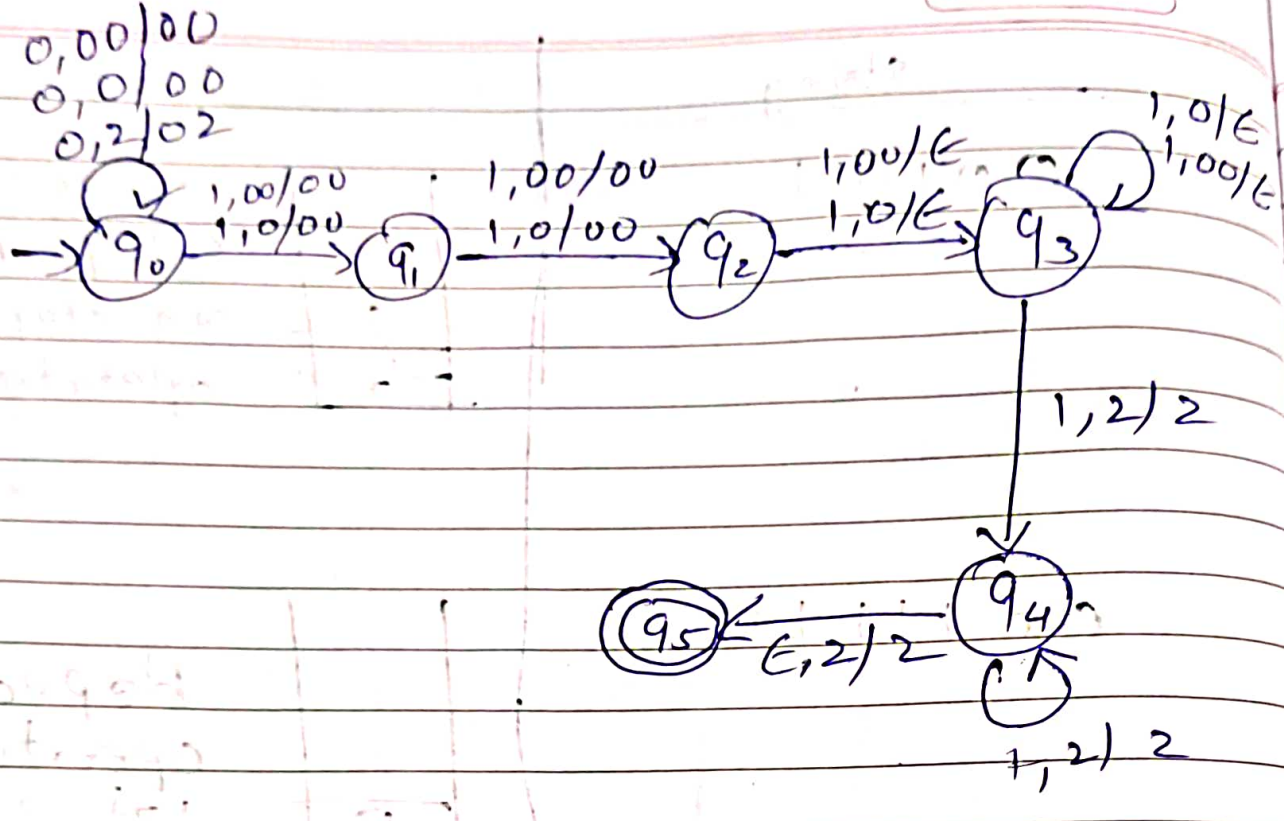
string	stack
0011111	
0011111	
0011111	

as $n+2 \leq m$, we can push '0' on 1st '2' occurance of '1'

String

Stack





ex:- $L = \{0^n, 1^{2n} / n \geq 1\}$

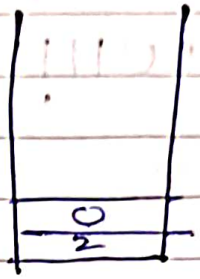
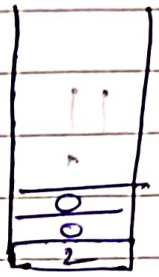
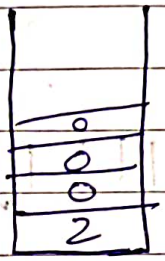
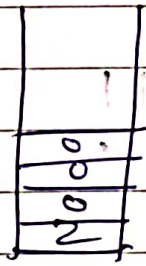
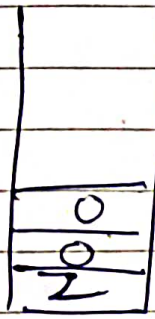
Consider (for example) $n = 3$

⇒ String: 000111111

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

$\Gamma = \{0, 2\}$

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

String	Stack
0 0 0 1 1 1 1 1	
0 0 0 0 1 1 1 1 1	
0 0 0 0 1 1 1 1 1	
0 0 0 1 1 1 1 1	 Here, we won't do any operation on 1st occurance of '1'. Because according to given language we want '2' occurance of '1' together
0 0 0 1 1 1 1 1 1	 Now, we'll do pop operation as we have two '1' together

String

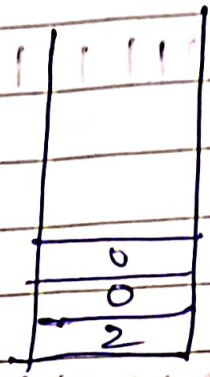
000111111

000 1111 11

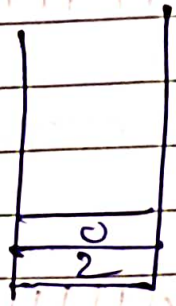
000 111 1110

000 11111

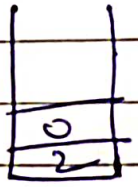
Stack



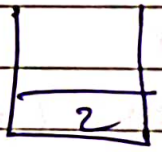
Ignore this '1'



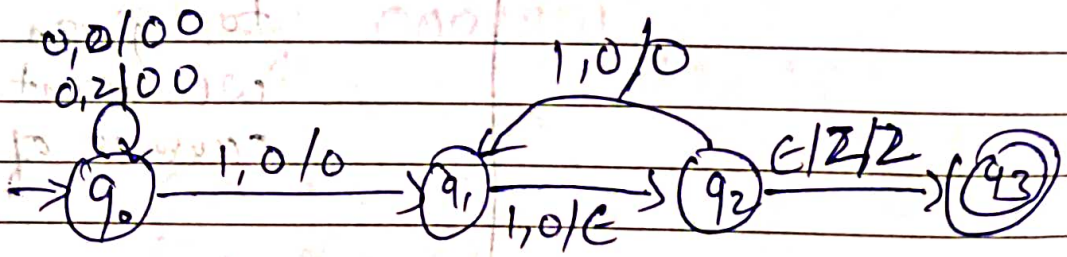
Pop ope.n



Ignore '1'



Pop operation



Example for practice :-

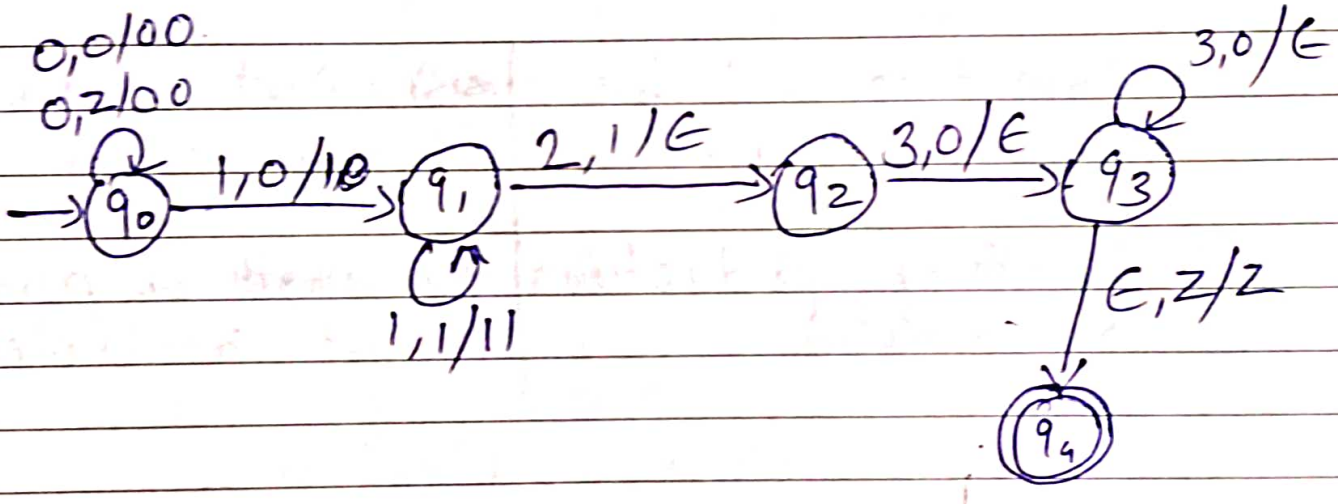
$$L = \{ 0^n 1^m 2^m 3^n \mid n \geq 1, m \geq 1 \}$$

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

$$\Sigma = \{ 0, 1, 2, 3 \}$$

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

"make stack by yourself"



Unit :- 4

"Properties of CFL"

PAGE NO. :
DATE : / /

★ Normal forms of CFL

Chomsky NF	Greibach NF
1) LHS will be one variable & RHS will be either maximum 2 variable (non terminal) or 1 terminal e.g. $A \to BC / a$	1) LHS will be one non terminal & RHS will be terminal followed by multiple non terminals e.g. $A \to aX$ $(a \in T, X \in V^*)$
2) No. of steps to generate string $= 2n-1$	2) No. of steps to generate string $= n$
3) Parse tree will be always binary	3) Not always binary parse tree
4) Length of production is restricted.	4) Length of production is not restricted

★ Pumping Lemma for CFL's

Pumping Lemma for CFL is used to prove that given language is not a context free.

Consider L is CFL having pumping length n such that any string $w \in L$ of length $\geq n$ can be written as

$|w| \geq n$ Then

We can break w into 5 substring like $w = uvxyz$ such that

$|vxy| \leq n$

$|vy| \neq \epsilon$

For all $k \geq 0$,

$uv^k xy^k z \in L$

Ex:-1) $L = \{a^N b^N c^N / N \geq 0\}$

Turing Machine:

In PDA we can compare 2 inputs as one comparison as we have one single stack in that case.

Where we have more than one comparison (e.g. $a^n b^n c^n d^n$), we can take 2 stacks or also implement queue to improve calculation power. Such machine is called turing machine

$$FA < PDA < TM$$

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

where,

$Q \rightarrow$ No. of finite states

$\Sigma \rightarrow$ Input Symbols set

$\Gamma \rightarrow$ Symbols allowed on input/output tape

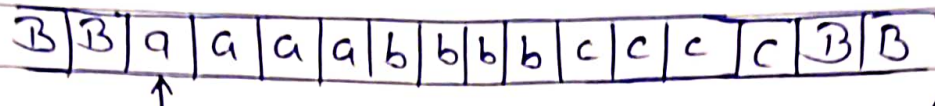
$\delta \rightarrow$ Transition $(\delta = Q \times \Gamma \times (L, R))$

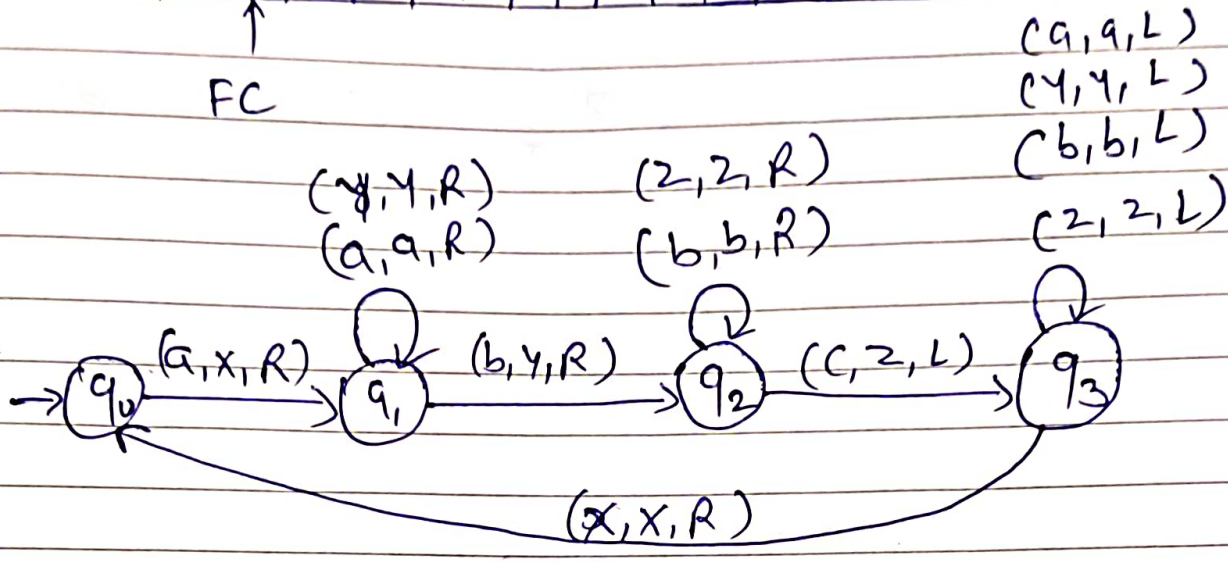
$q_0 \rightarrow$ Initial state

$B \rightarrow$ Blank state

$F \rightarrow$ Final state

Ex: Design TM for $(a^n b^n c^n / n \geq 1)$





1) B B X a a a Y b b b Z c c c B B

2) B B X X a a Y Y b b Z Z c c B B

3) B B X X X a Y Y Y b Z Z Z c B B

4) B B X X X X Y Y Y Y Z Z Z Z B B