Mid II

→ Kleene's Theorem :-

If a language is expressed by FA, TG, RE then it can also be expressed by other two as well

Part 1: If accepted by FA, then it can be accepted by TG.

Part 2: If accepted by TG, then it can be accepted by RE.

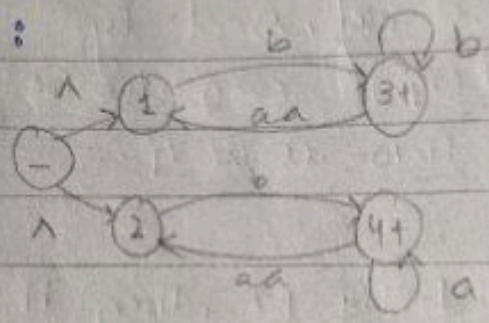
Part 3: If accepted by RE, then it can be accepted by FA.

→ Part 1: Every FA is also a TG. (conversion not required).

→ Part 2: Given TG, extract FA.

Obtaining RE from TG

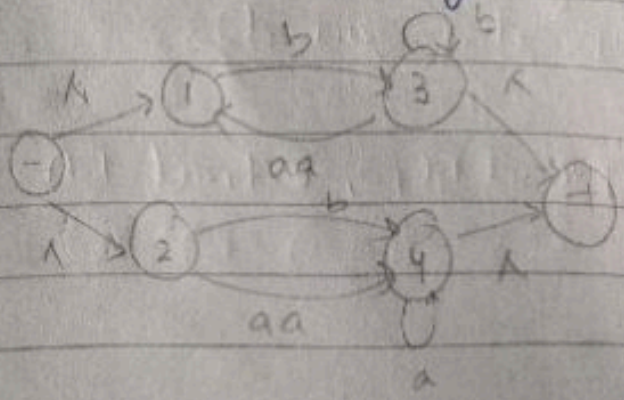
Step 1:



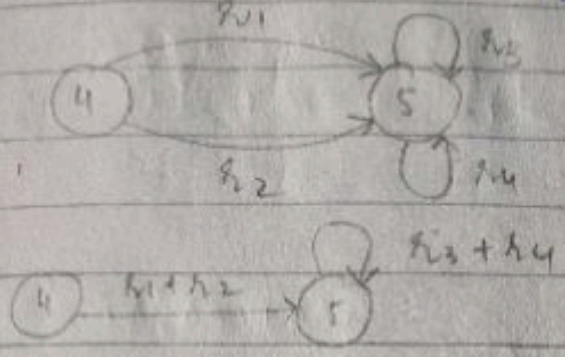
- Required if multiple initial states are there, then make new initial state connected new state by old by connecting through null.

Step 2:

More than 1 final state, then introduce a new final state by joining new state with old by null transition.

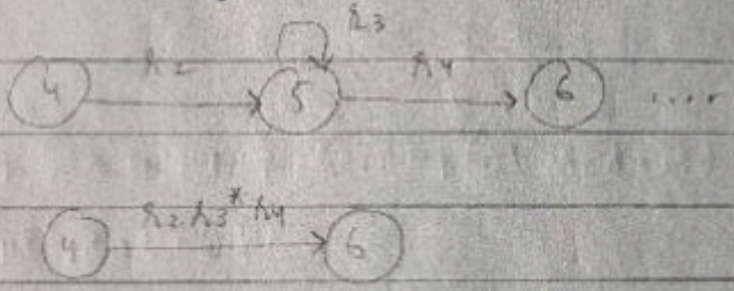


Step 3: Reduce states by adding parallel incoming edges

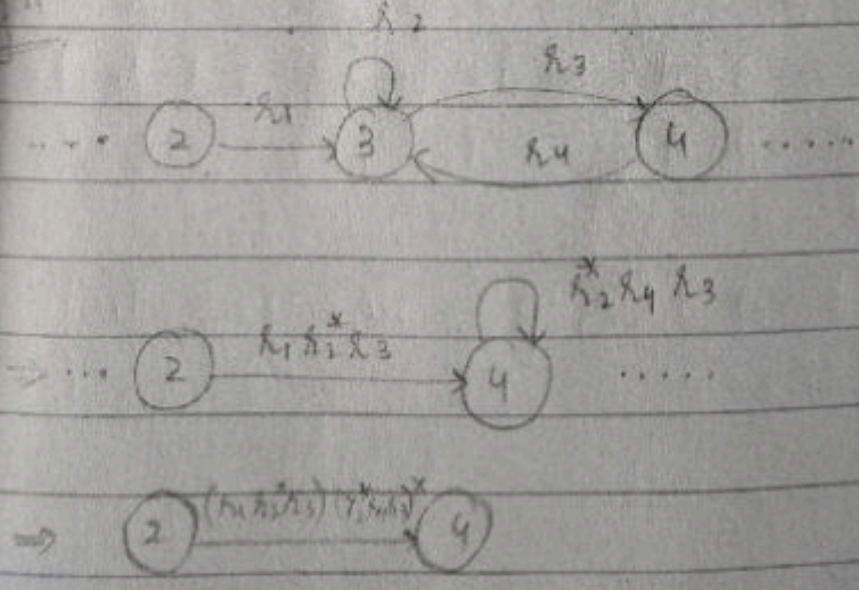


Step 4: By Pass and state elimination

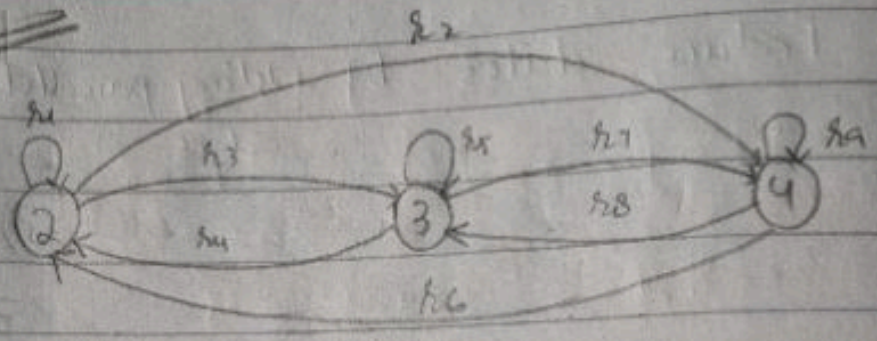
eg #



eg #

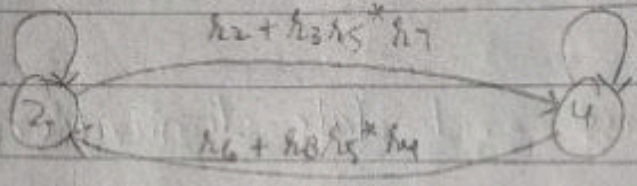


Eg#



$r_1 + r_3 r_5^* r_4$ $\qquad$ $r_9 + r_8 r_5^* r_7$

$=$

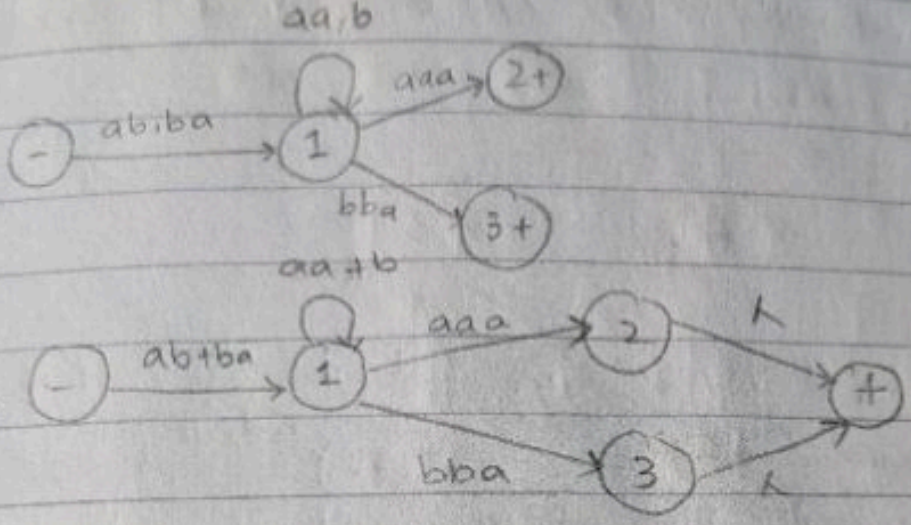


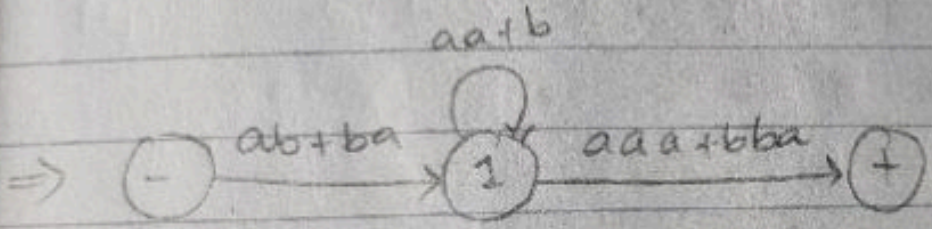
$=$

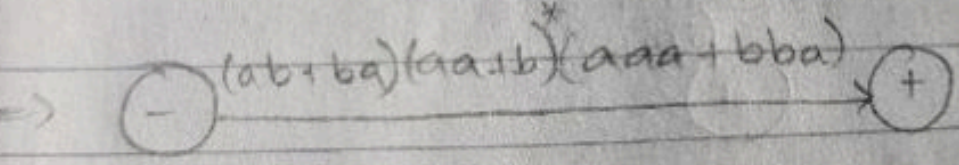
$(r_1 + r_3 r_5^* r_4) + (r_2 + r_3 r_5^* r_7)(r_9 + r_8 r_5^* r_7)^*$
$(r_6 + r_8 r_5^* r_4)$

Tuesday
5 Mar 19

Example:



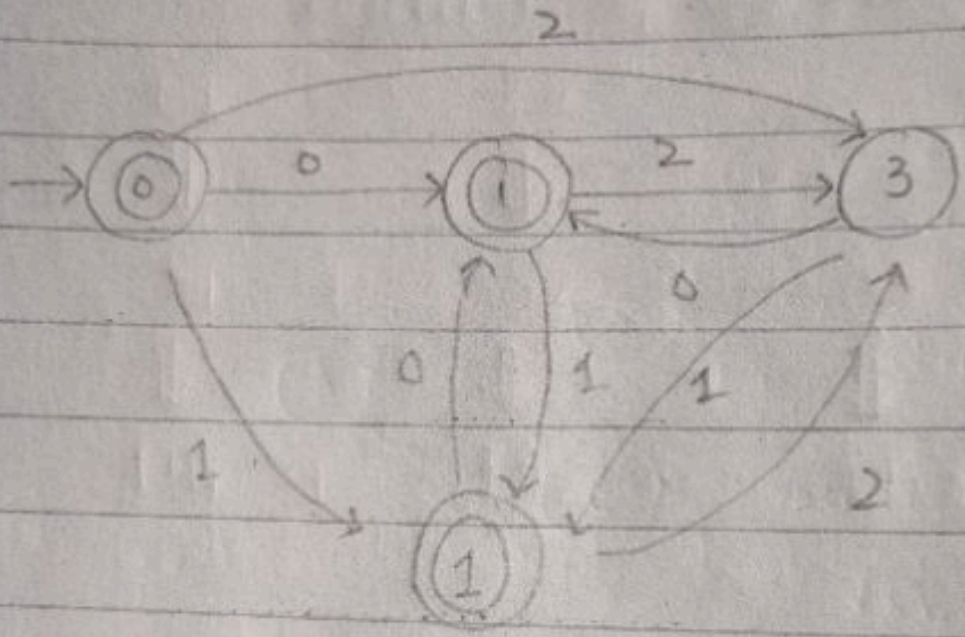




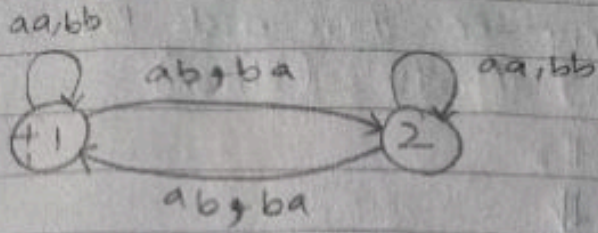
$R.E = (ab+ba)(aa+b)^*(aaa+bba)$

=> Cohen

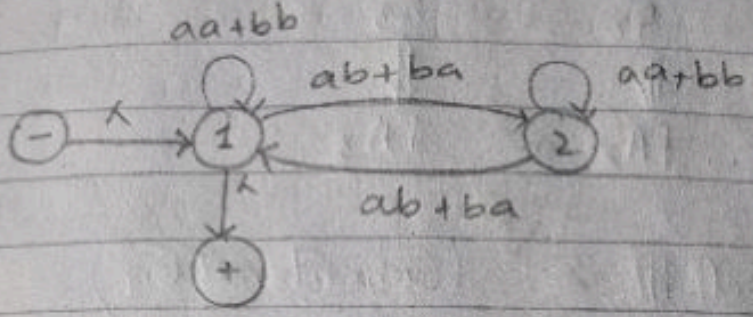
eg=#



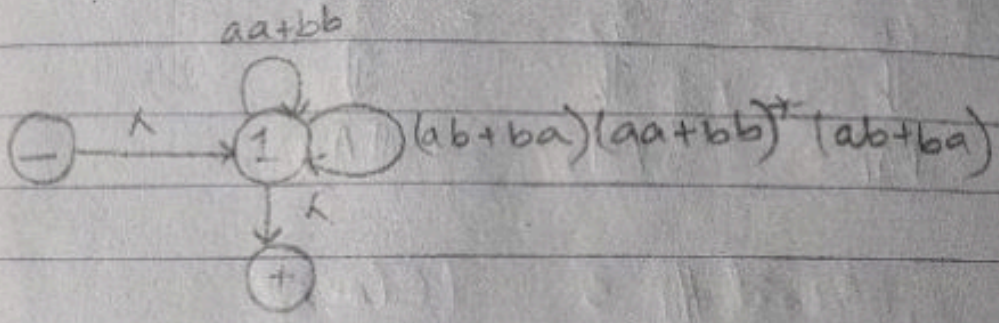
eg# Language EVEN-EVEN



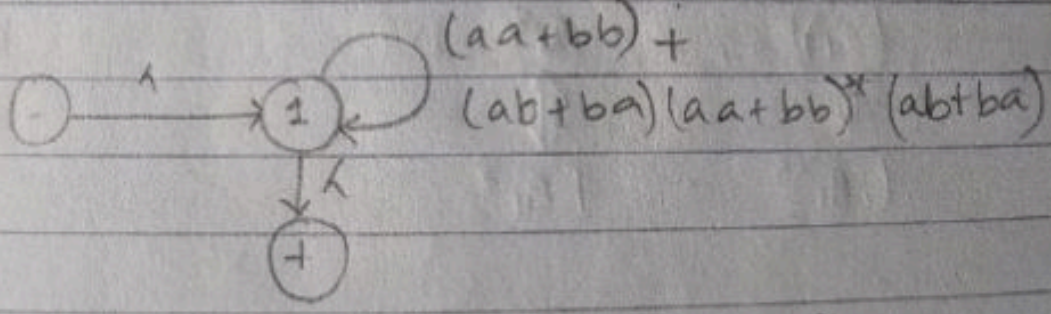
1

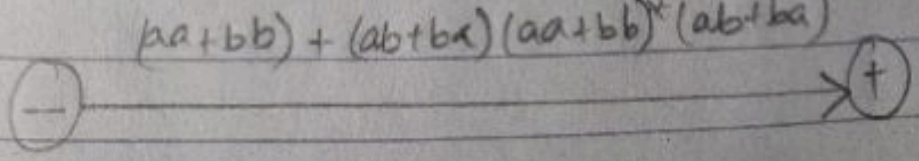


2



3





R.E = (aa+bb) + (ab+ba)(aa+bb)* (ab+ba)

Part 3: For every RE there exists FA.

Method # 01 (Union)

$$\begin{array}{ccccc} L_1 & & L_2 & & \\ \Downarrow & & \Downarrow & & \\ R_1 & + & R_2 & = & R_3 \\ \Downarrow & & \Downarrow & & \Updownarrow \\ FA_1 & + & FA_2 & = & FA_3 \end{array}$$

Method # 2 (Concatenation)

$$\begin{array}{ccccc} R_1 & \cdot & R_2 & = & R_3 \\ \Downarrow & & \Downarrow & & \Updownarrow \\ FA_1 & \cdot & FA_2 & = & FA_3 \end{array}$$

Method # 3 (Closure)

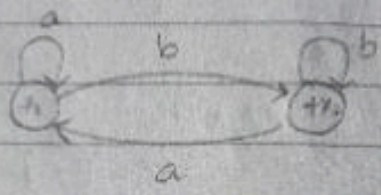
$$\begin{array}{ccc} (R_1)^* & = & R_i^* \\ \Downarrow & & \Updownarrow \\ (FA_1)^* & = & FA_1^* \end{array}$$

Method # 1 → Initial State FA_1 & FA_2
→ Final State FA_1 & FA_2
Union of Two FA's

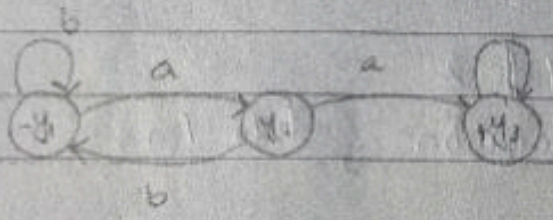
$r_1 = (a+b)^* \cdot b$

$r_2 = (a+b)^* \cdot aa(a+b)^*$

FA_1 :



FA_2 :

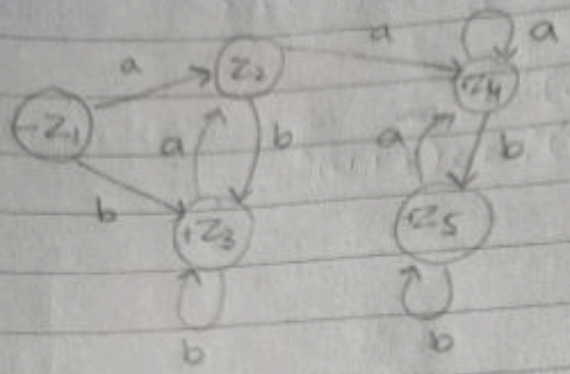


$r_3 = (a+b)^*b + (a+b)^* \cdot aa(a+b)^*$

FA_3 : Construct a transition table

Old states	a	b
$\rightarrow Z_1 = (X_1, Y_1)$	$Z_2 = (X_1, Y_2)$	$Z_3 = (X_2, Y_1)$
$Z_2 = (X_1, Y_2)$	$Z_4 = (X_1, Y_3)$	$Z_3 = (X_2, Y_1)$
$+Z_3 = (X_2, Y_1)$	$Z_2 = (X_1, Y_2)$	$Z_3 = (X_2, Y_1)$
$+Z_4 = (X_1, Y_3)$	$Z_4 = (X_1, Y_3)$	$Z_5 = (X_2, Y_3)$
$+Z_5 = (X_2, Y_3)$	$Z_4 = (X_1, Y_3)$	$Z_5 = (X_2, Y_3)$

Draw FA



Then check language

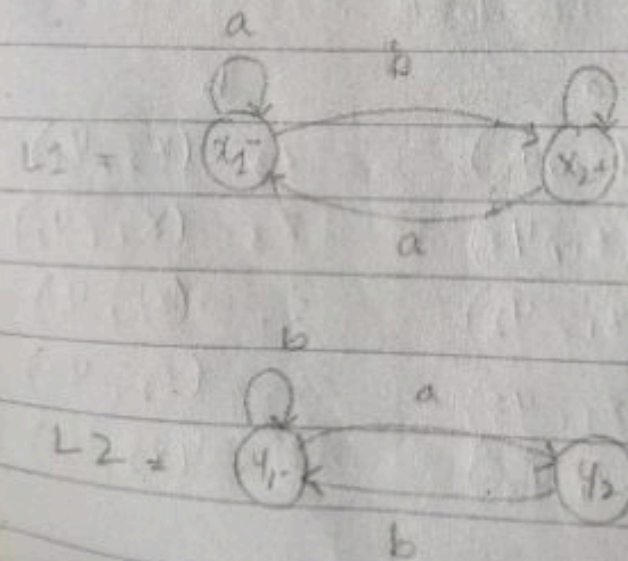
6/March/19

Method # 2

Concatenation of two FA's

$R_1 = (a+b)^*b$

$R_2 = (a+b)^*aa(a+b)^*$



(59) Cohen. chp# 7 "Kleene's Theorem"

$r_3 = (a+b)^* b \cdot (a+b)^* aa (a+b)^*$

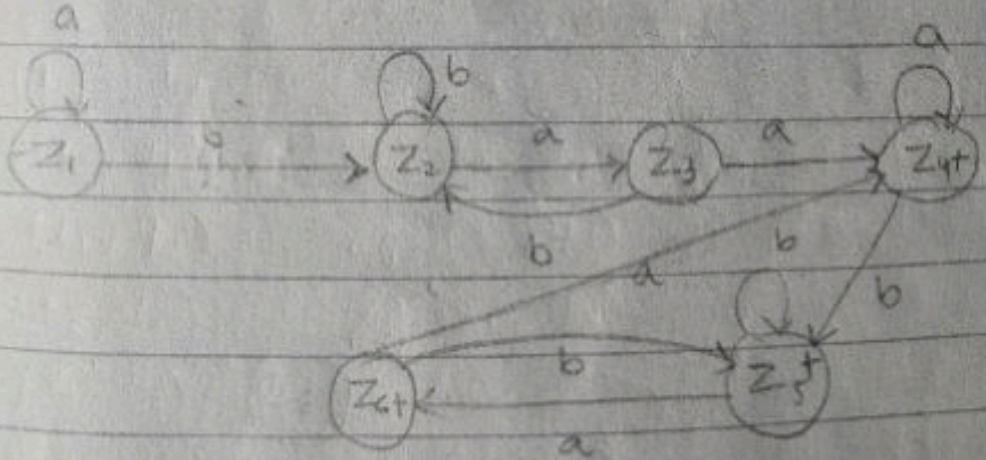
$L = \{baa, \dots\}$

Initial state starts from FA1
Final state → FA2
Final state of FA1 is initial state of FA2

} 2 jump points

Old states	New states a	b
$z_1 = x_1$	$z_1 = x_1$	$z_2 = (x_2, y_1)$ because x_2 is Final of FA1
$z_2 = (x_2, y_1)$	$z_3 = (x_1, y_2)$	$z_2 = (x_2, y_1)$
$z_3 = (x_1, y_2)$	$z_4 = (x_1, y_3)$	$z_2 = (x_2, y_1)$
$z_4 = (x_1, y_3)$	$z_4 = (x_1, y_3)$	$z_5 = (x_2, y_1, y_3)$
$z_5 = (x_2, y_1, y_3)$	$z_6 = (x_1, y_2, y_3)$	$z_5 = (x_2, y_1, y_3)$
$z_6 = (x_1, y_2, y_3)$	$z_4 = (x_1, y_3)$	$z_5 = (x_2, y_1, y_3)$

Final state jis mai y_3 aya hoga



Method #3 → Initial state of required FA is the final state as well

Closure of an FA

* Concatenation of an FA by itself

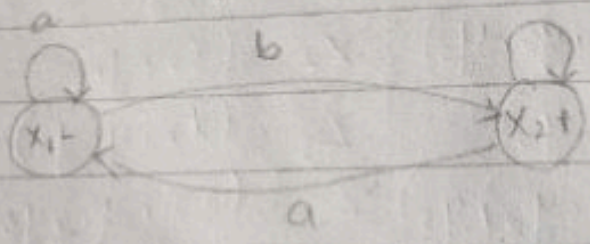
$r = ab$

$r \rightarrow (ab)^*$

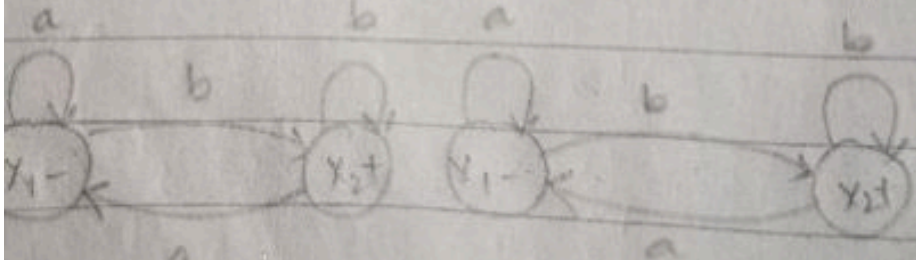
$\Uparrow$

$FA \cdot FA = (FA)^*$

$r = (a+b)^*b$



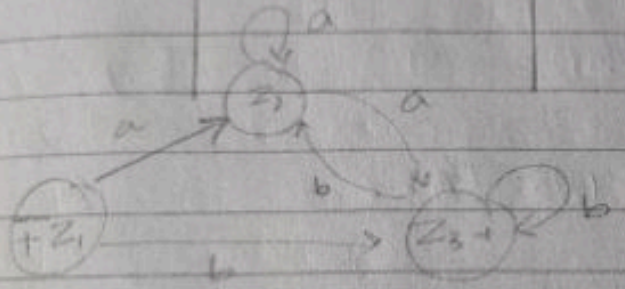
To form r^*, form replica



(6.1)

$FA^* = FA \cdot FA$

Old state	New state a	New state b
$Z_1 = X_1$	$Z_2 = (X_1, X_1)$	$Z_3 = (X_2, X_1)$
$Z_3 = (X_1, X_1)$	$Z_2 = (X_1, Z_2)$	$Z_3 = (X_2, X_1)$.

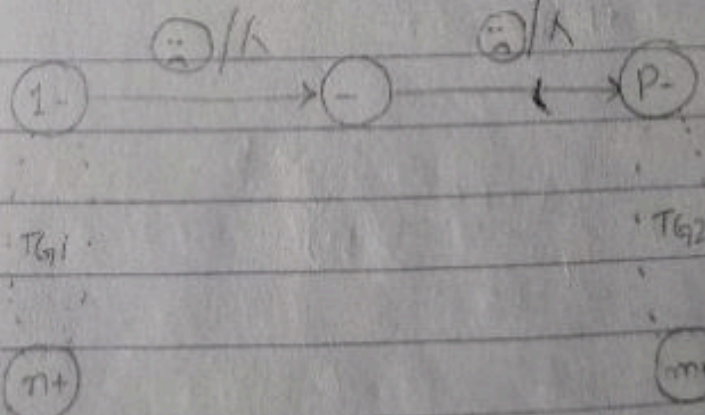


7th/March/19

* Union of two TG's:

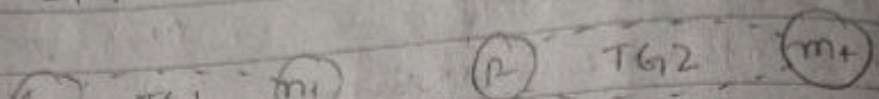
- L_1 & L_2 are expressed by TG1 & TG2

$= L_1 + L_2$

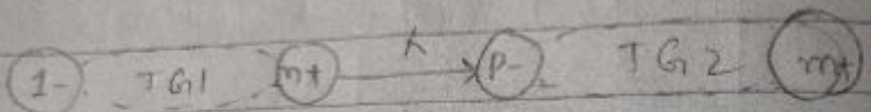


* Concatenation of two TG's:

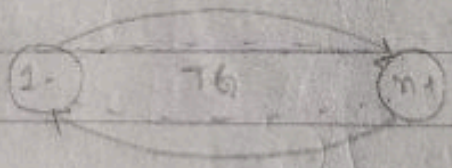
L_1 & L_2 expressed by TG_1 & TG_2



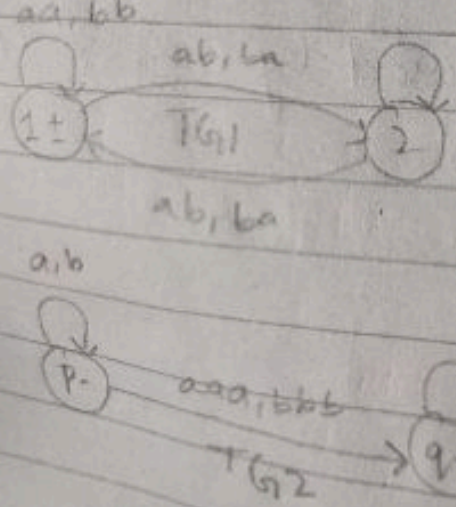
= Concatenation



* Closure of two TG's:

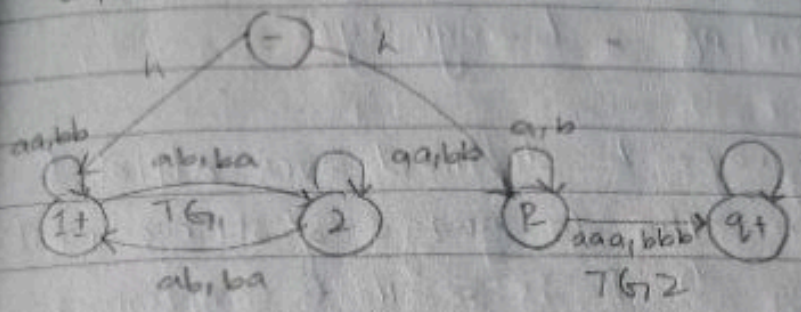


Eg#1

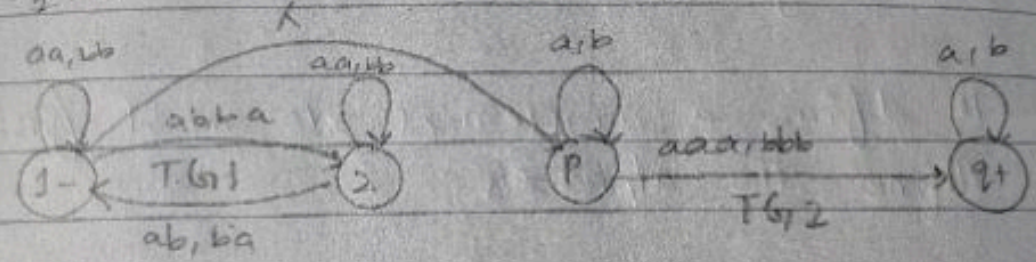


(5.3)

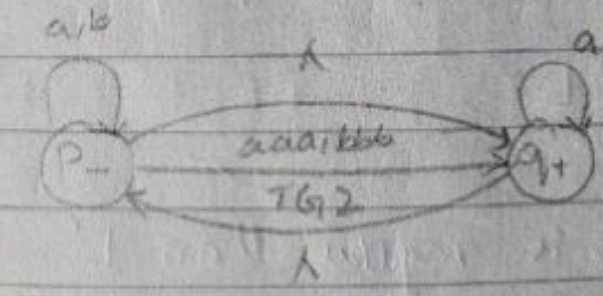
$L_1 + L_2$



$L_1 \cdot L_2$



$L2^*$



* Complement of a Language:

The Language L defined over alphabet Σ, then the language of strings not belonging to L is called Complement.

Denoted by L^c

- To define a complement we must first define the alphabets.

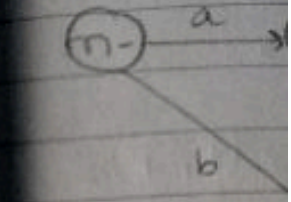
- Complement of a already complemented language gives the actual language.

eg: $(L^c)^c = L$

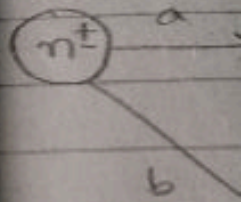
Theorem: If L is regular than L^c is also regular.

- To find a complement of an FA, we convert all final states to non-final states & all non-final states to final states.

Example



Complemer



Theorem
than

(Li^c U

* Complement of a Language:

The Language L defined over alphabet Σ, then the language of strings not belonging to L is called Complement

Denoted by L^c

- To define a complement we must first define the alphabets.

- Complement of a already complemented Language gives the actual language.

eg: $(L^c)^c = L$

Theorem: If L is regular then L^c is also regular.

- To find a complement of an FA, we convert all final states to non-final states & all non-final states to final states.

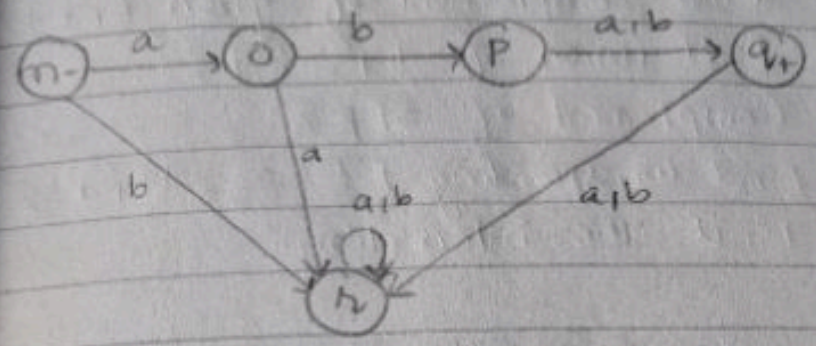
Example

Complemen

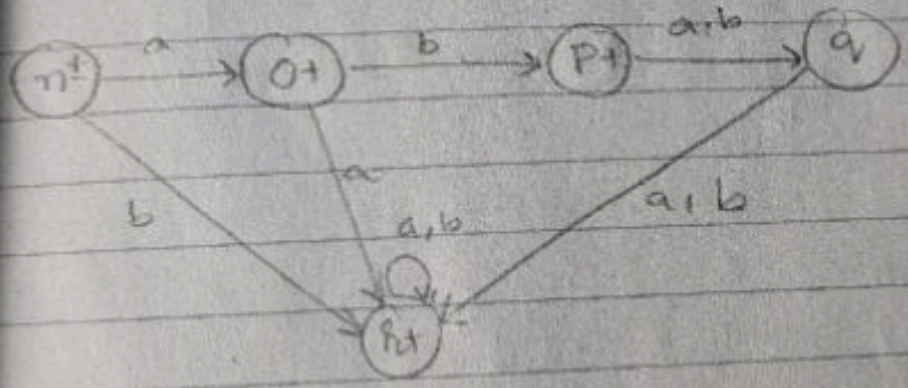
Theorem
than

$(L_i^c \cup$

Example $\Sigma = \{a, b\}$ RE : aba + abb



Complement



Theorem 2: If L_1 & L_2 are regular than their intersection is also a regular.

$$(L_1^c \cup L_2^c)^c = (L_1^c)^c \cap (L_2^c)^c = L_1 \cap L_2.$$

Proof of Theorem 2 Using Demorgan's Law

Step 1: Find complement of given L_1 & L_2

Step 2: Find union of L_1^c & L_2^c

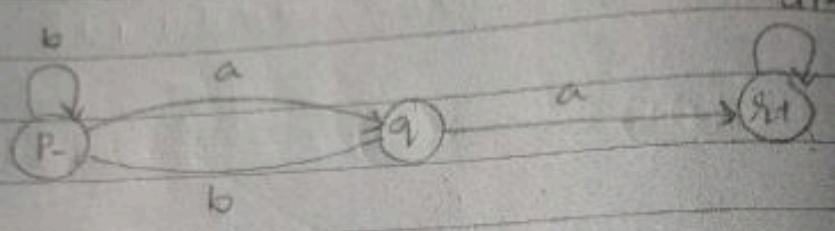
Step 3: Complement of step 2

Step 4: Find corresponding R.E of $(L_1^c + L_2^c)^c$

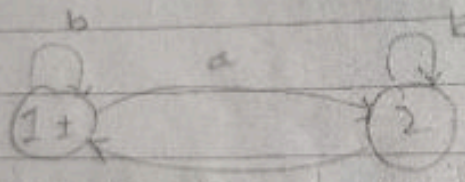
Step 5: Find the intersection

FA1

RE = $b^*(ab)^* a(a+b)^*$



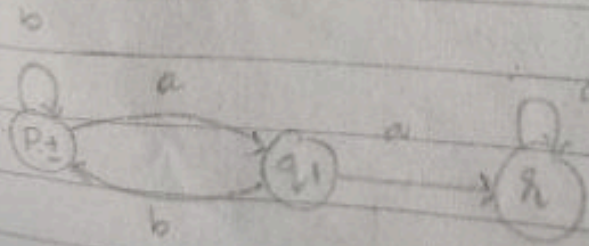
FA2



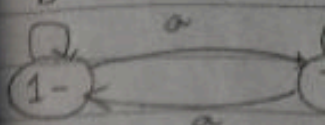
RE = $b^* + ab^* a$

Step 1:

FA_1^c:



FA_2^c:



Step 2:

Old States:

$Z_1 = (P, 1)$

$Z_2 = (P, 2)$

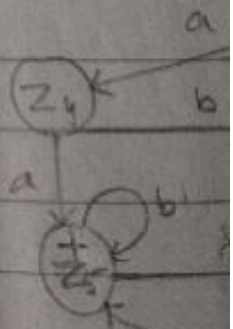
$Z_3 = (q, 1)$

$Z_4 = (q, 2)$

$Z_5 = (R, 1)$

$Z_6 = (R, 2)$

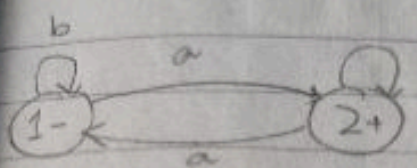
Step 3:



Lan:

L_2^c):

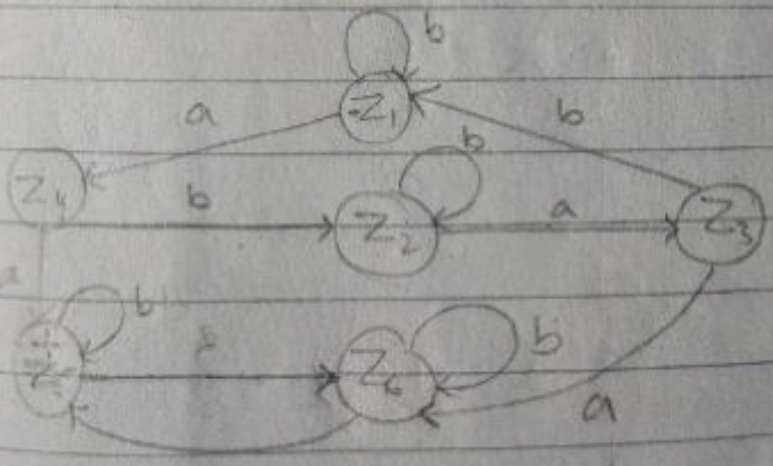
FA_2^c:

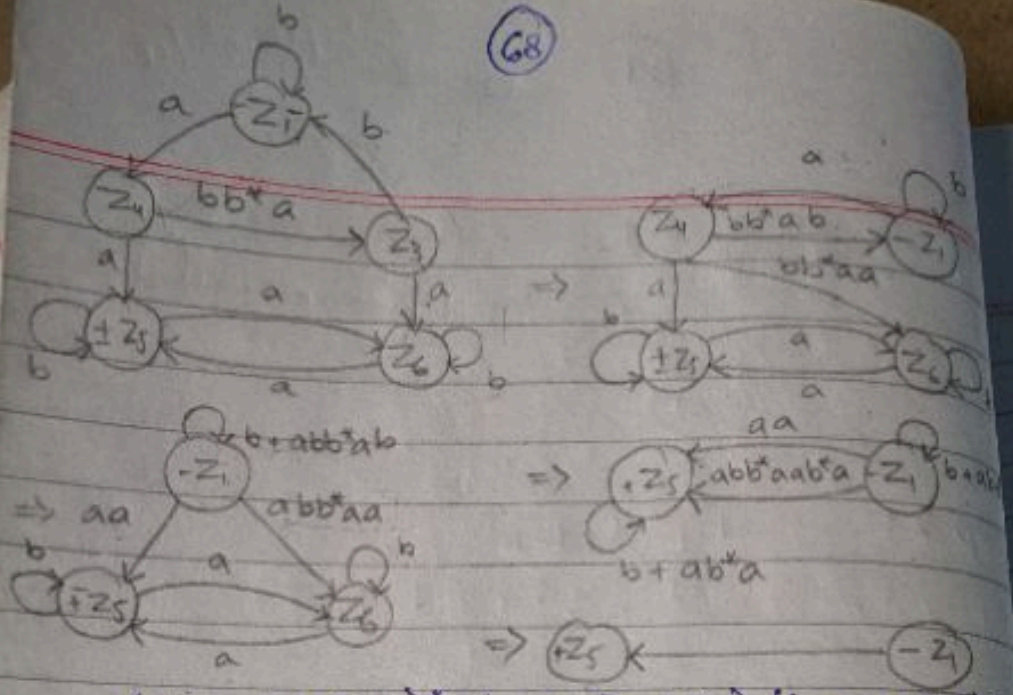


Step 2: $L_1^c \cup L_2^c$

Old States	New States a	b
$z_1 = (p, 1)$	$z_4 = (q, 2)$	$z_1 = (p, 1)$
$z_2 = (p, 2)$	$z_3 = (q, 1)$	$z_2 = (p, 2)$
$z_3 = (q, 1)$	$z_6 = (r, 2)$	$z_1 = (p, 1)$
$z_4 = (q, 2)$	$z_5 = (r, 1)$	$z_2 = (p, 2)$
$z_5 = (r, 1)$	$z_6 = (r, 2)$	$z_5 = (r, 1)$
$z_6 = (r, 2)$	$z_5 = (r, 1)$	$z_6 = (r, 2)$

Step 3: $(L_1^c \cup L_2^c)^c$





$(b + abb^*ab)^*a(a + bb^*aab^*a)(b + ab^*a)^*$

Step #5 $L_1 \cap L_2$

Old States	New States	
	a	b
$z_1 = p$	q	p
$z_2 = q$		

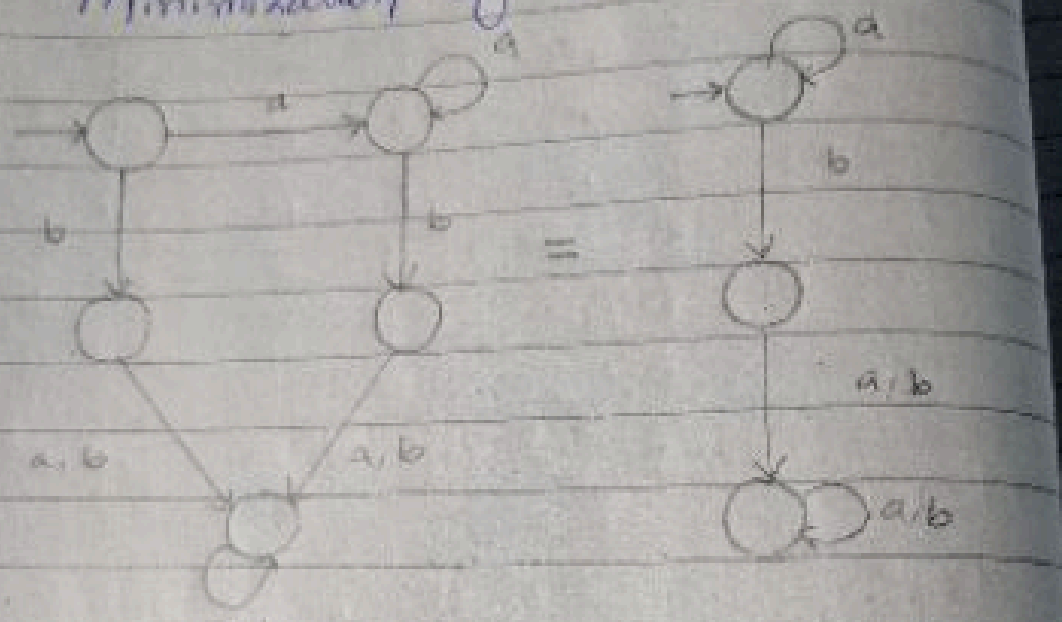
(69).

) What is the complement of language of an NFA. What is the language of the complement of the NFA ?

o— DFA ka complement uski language ka complement bh hoga.

11/March/19 (70)

Minimization of DFA:



- DFA can be minimised only once.
- Minimal DFA cannot be further minimised.
- NFA cannot be minimised only DFA can be minimised.

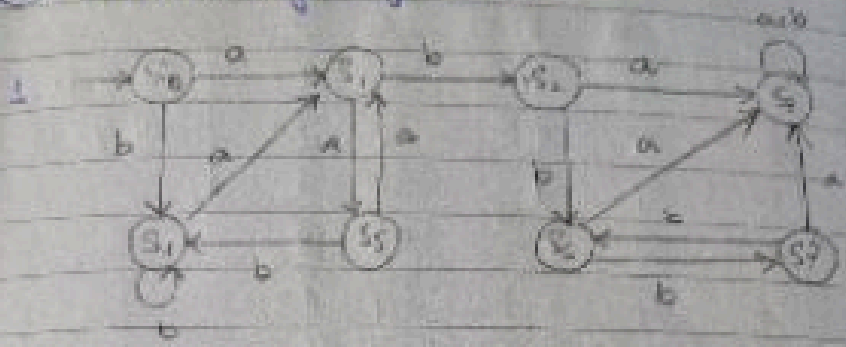
→ Process:

→ Get rid of inaccessible states

→ Group equivalent states.

(41)

(1) Partitioning Algorithm:-



		a		b	
Non final state	S_0	S_1	I	S_4	I
	S_1	S_3	I	S_2	II
	S_3	S_5	I	S_5	I
	S_4	S_2	I	S_4	I
	S_5	S_4	I	S_4	I
	S_6	S_5	I	S_7	II
Final state	S_2	S_5	I	S_6	I
	S_7	S_5	I	S_6	I

→ Pehle partition kara non final & final ka
→ Phir figure se dekh kr transition likhi
→ Phir numbering kari $(S_0, S_1, S_3, S_4, S_5) \rightarrow$ I
$(S_2, S_7) \rightarrow$ II
→ Draw new transition table.
S_1 & S_6 different hain
a → I , b → II

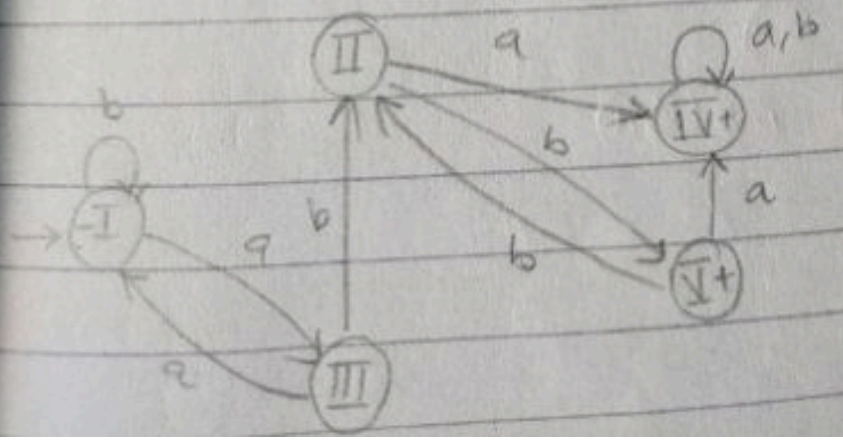
		a		b		Partitioning dethrongé
I	S_0	S_1	II	S_4	I	
	S_2	S_2	I	S_3	I	S_3 looks different
	S_4	S_1	II	S_1	I	
	S_5	S_1	II	S_4	I	
II	S_1	S_5	I	S_2	II	
	S_6	S_3	I	S_7	II	
III	S_2	S_3	I	S_5	II	
	S_7	S_3	I	S_6	II	

		a		b		
I	S_0	S_1	III	S_4	I	same
	S_4	S_1	III	S_4	I	
	S_5	S_1	III	S_4	I	
II	S_3	S_3	II	S_3	II	
III	S_1	S_5	I	S_2	IV	
	S_6	S_3	II	S_7	IV	
IV	S_2	S_3	II	S_6	III	same
	S_7	S_3	II	S_6	III	

		a		b		
I	S_0	S_1	III	S_4	I	same
I	S_4	S_1	III	S_4	I	same
I	S_5	S_1	III	S_4	I	same
IV	S_3	S_3	IV	S_3	IV	
III	S_1	S_5	I	S_2	II	
V	S_6	S_3	IV	S_7	II	
II	S_2	S_3	IV	S_6	V	same
II	S_7	S_3	IV	S_6	V	same

Now make new table with new states.

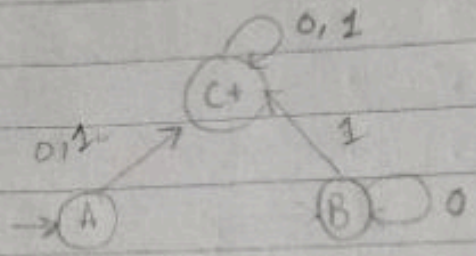
States	a	b
→ I	III	I
II	IV	V
III	I	II
+ IV	IV	IV
+ V	IV	II



(2) Myphill Nerode Theorem:

(TF Algorithm)

↳ Table Filling



i, j	A	B	C
A			
B			
C	✓	✓	

$Q_i \in F$ $Q_j \notin F$

Now pick unmark pairs & see their transitions.

let (A, B)

$\delta(A,0), \delta(B,0)$

(C, B)

$\delta(A,1), \delta(B,1)$

(C, C)

Step 1: D

Step 2: M

one s

state

Q:

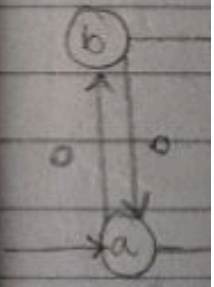
Step 3:

ob

Pick

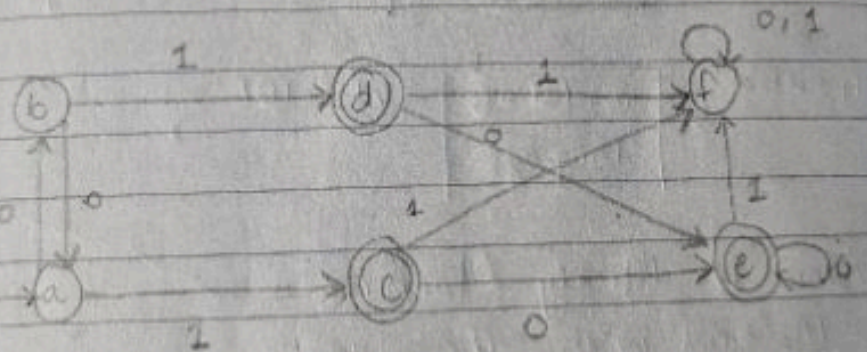
trans

mar



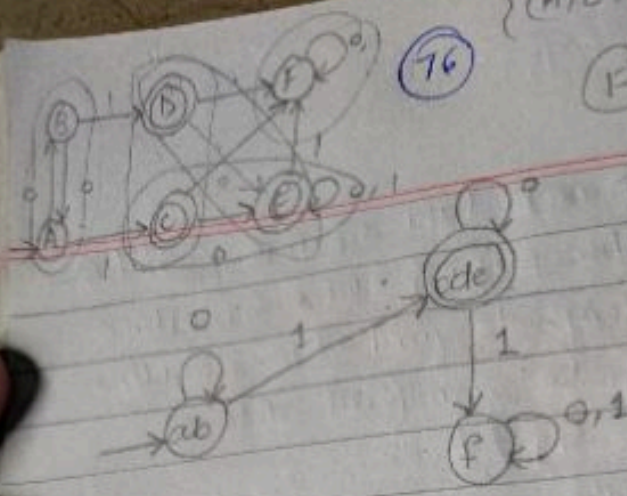
i	a
a	
b	
c	✓
d	✓
e	✓
f	✓

Step 1: Draw unmark table of pair of states.
Step 2: Mark the pair of states in which one state belongs to final state & another state belongs to non-final state
$Q_i \in F$ & $Q_j \notin F$
Step 3: Stop when no new mark state is obtained.
Pick the unmark pair, & see its transitions. If it results in previously marked pair, then mark that pair.

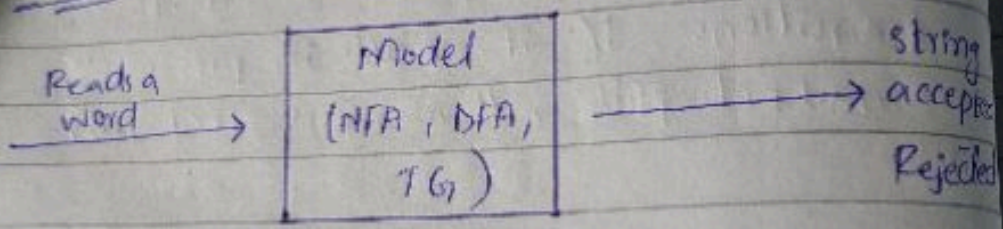


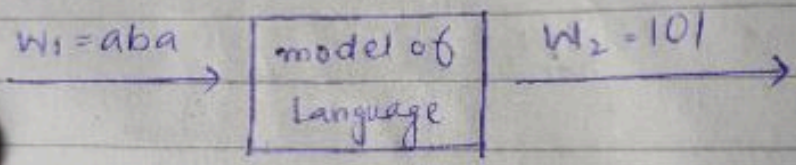
	a	b	c	d	e	f
a						
b						
c	✓	✓				
d	✓	✓				
e	✓	✓				
f	✓	✓	✓	✓	✓	

(b,0) ∪ (a,0) ✓



14/Mar/19





Moore & Mealy Machine

- Machines with the output
- Finite automata with output
- Required for complex operations

→ Moore Machines

$$\langle Q, \Sigma, \Gamma, \delta, \lambda \rangle$$

input — Output

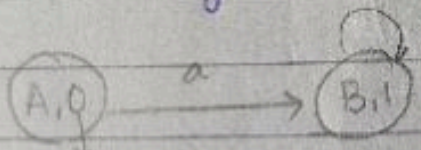
→ Moore Machine:

$\lambda : Q \to \Gamma^*$ → delta

$\langle Q, \Sigma, \Gamma, \delta, \lambda \rangle$

input | letters → output | characters

- Q = finite set of states
  q_0 → initial state
- Σ = alphabet for input strings
- Γ = alphabet of possible output characters
- Transition table shows for each state & each input letter what state is reached next
- An output table shows what character is printed by each state



$n \longrightarrow n+1 \longrightarrow$ output

n ↓ size of input

$n+1$ character is always $n+1$ s/c initial state also emits something

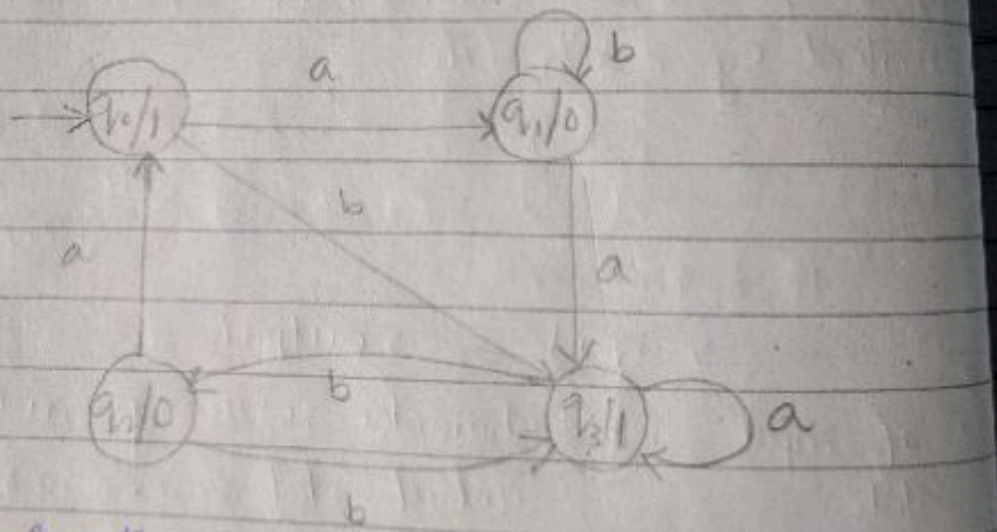
- No final state, since every input string creates an output string.

eg#1 Input : $\Sigma = \{a, b\}$

Output : $\Gamma = \{0, 1\}$

States : q_0, q_1, q_2, q_3

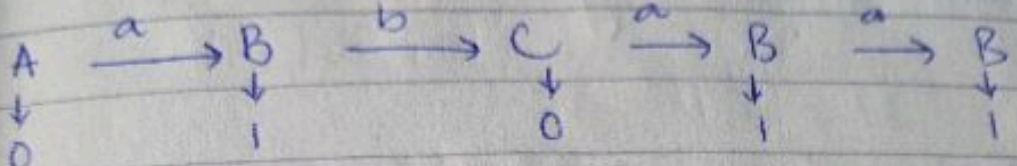
Old State	Next State		Output
	a	b	
$\rightarrow q_0$	q_1	q_3	1
q_1	q_3	q_1	0
q_2	q_0	q_3	0
q_3	q_3	q_2	1

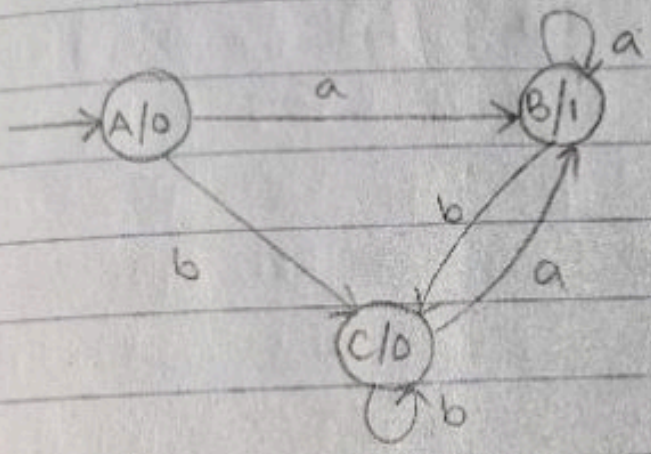


input: abab → output: 1 0 0 1 0

n → n+1

Behaviour of Moore Machine



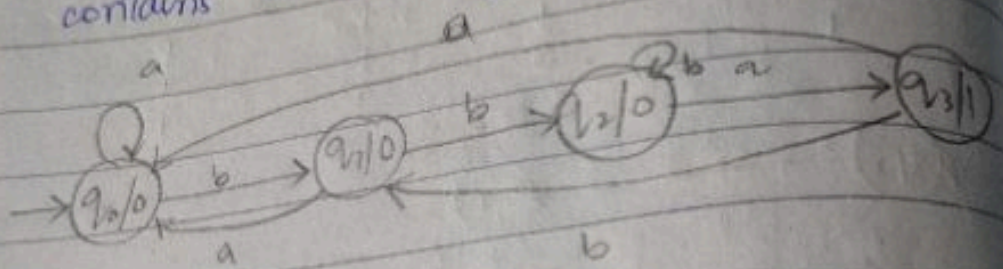


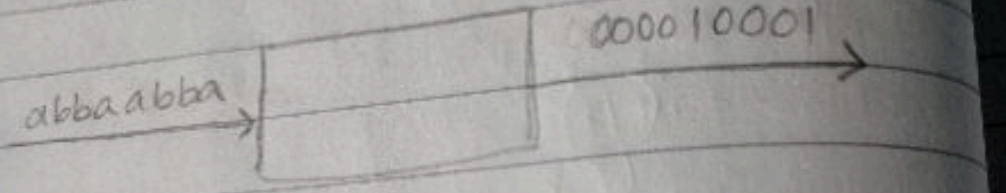
Old states	a	b	a	b
A	B	C	01	00
B	B	C	1	0
C	B	C	1	0

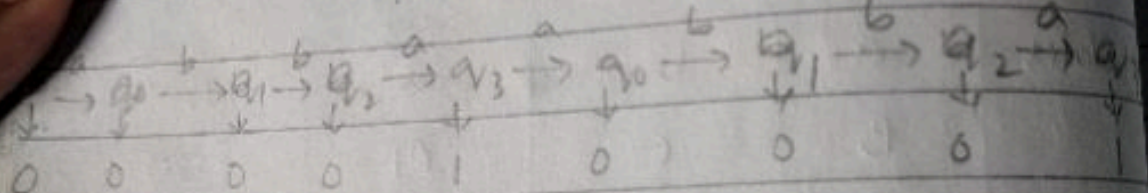
a b
0 11
1 0
1

Monday
18/Mar/19

Q Construct a Moore Machine that contains a substring bba

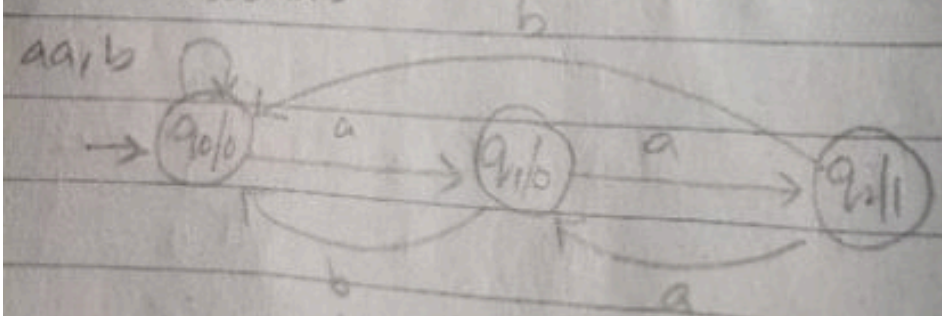






Q. string with aa

ababaab



Limitation

if length of string is n
then its output is always n+1

0110 ⟶ (0)1001

(complement and change most
bhi aasakta b/c of n+1

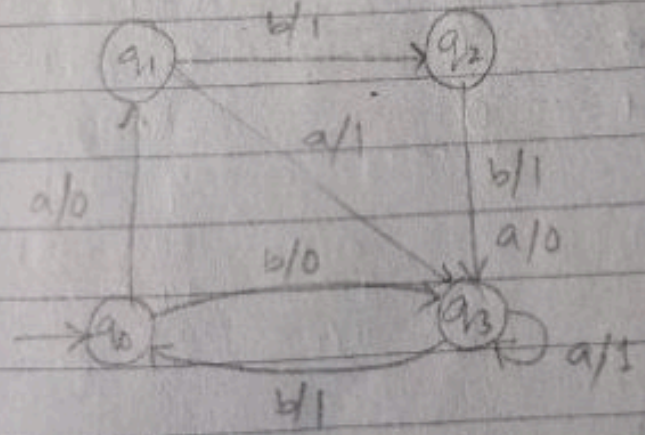
→ Mealy Machine: $\langle Q, \Sigma, \Gamma, \delta, \lambda \rangle$ → $\lambda : Q \times \Sigma \to \Gamma$ → output function.

Transition per output ayega.

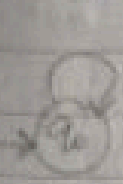
q_0 —a/0→ q_1

∴ No final state required.

Q. $\Sigma = \{a, b\}$, $\Gamma = \{0, 1\}$

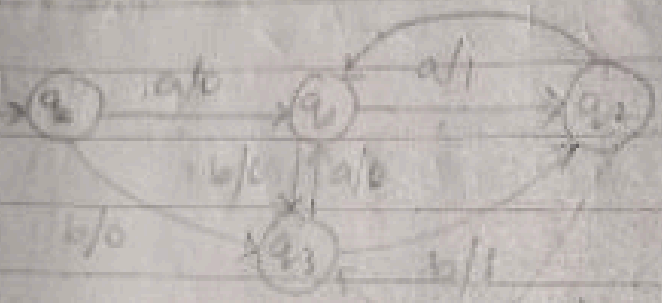


→ Complement machine using Mealy Machine



→ 0 per 1
→ 1 per 0

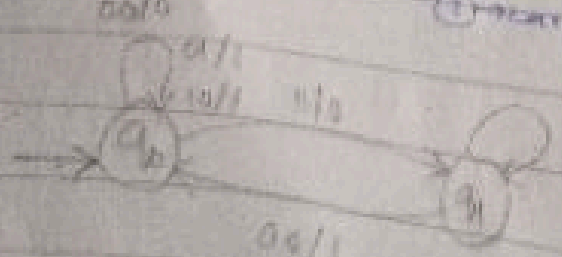
→ Construct a mealy machine consist of * ,
aa, ** or bb



→ A + B = o/p (Addition Machine)

A	B	o/p
0	0	0
0	1	1
1	0	1
1	1	0

(1 remaining)



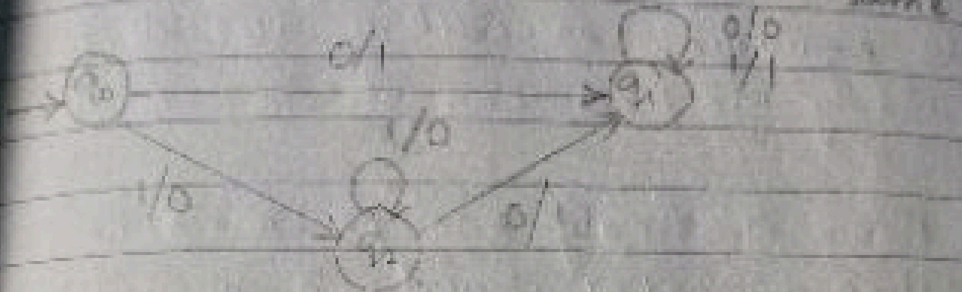
Operation → R to L
Machine → L to R
Top pehle inverse krenge
eg: 1011 → 1101
phr result check kr k inverse krenge

→ Increment Mealy Machine

LSB '0' has to be set digit change hua

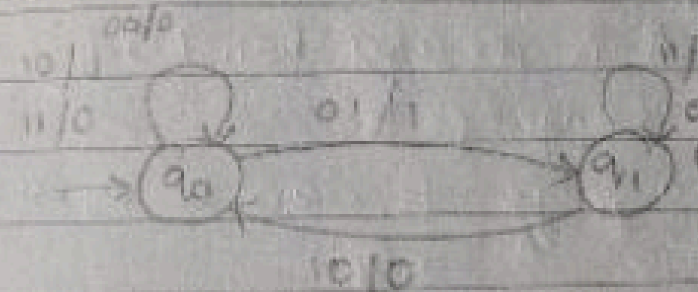
1000	1011
+ 1	+ 1
1001	1100

LSB '1' tou pehle zero ke different par same



→ Subtraction Mealy Machine (H.W).

Hint: increment + addition + complement



0	0	0
0	1	1
1	0	1
1	1	0

→ OR Base 9s subtraction machine

64 − 29 → 99 − 29 = 70 → 64 + 70 = 134 → 134 + 1 = 135; 99 − 70 = 39

19/Mar/19 (84)

→ Equivalent Machines:
Two machines are said to be equivalent if they produce same output on same input.

- Two different Moore Machines can be equivalent as well as two different Mealy Machines can also be equivalent

- Moore & Mealy can produce same output if initial character is ignored

Power of machine:
Machine A is powerful than Machine B for same Language L
because Language L can be expressed by Machine A only.

eg#

TM > PDA's > FA, TG, Moore, Mealy
R.E
Context-free Language
$a^n b^n c^n$

→ Conversion

Statement:
For every
Machine

Method:
To the

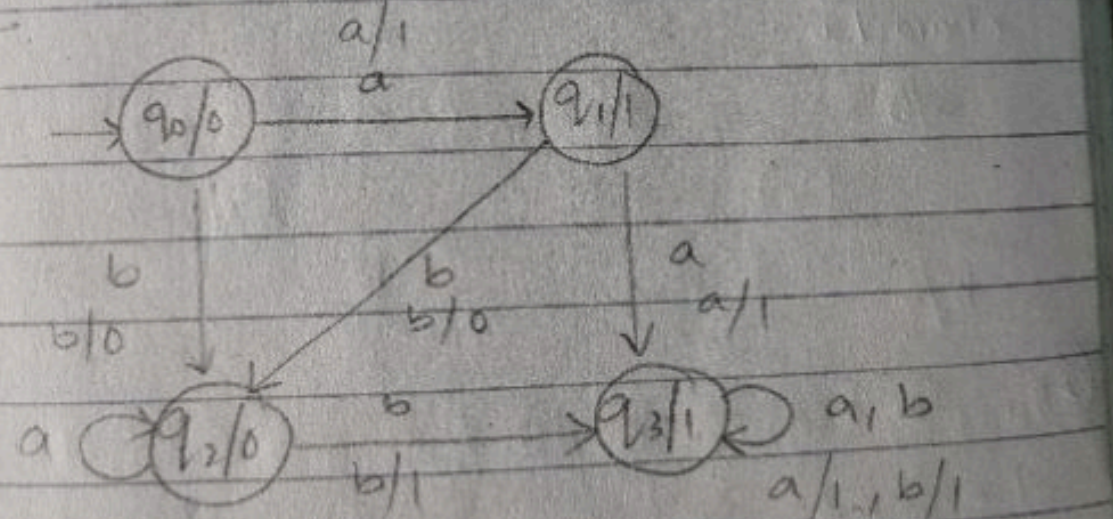
eg#

→ Conversion from Moore Machine to Mealy Machine:

Statement:
For every Moore Machine there is a Mealy Machine that is equivalent to it.

Method: Shift output character corresponding to the state to the incoming transition

eg #



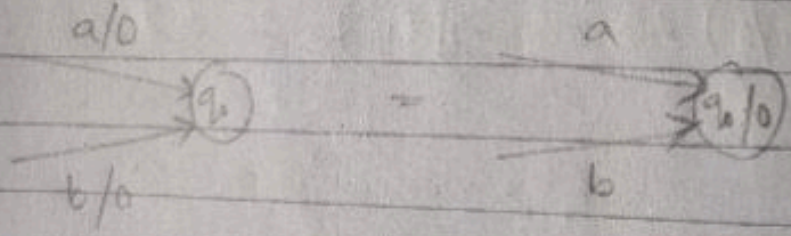
→ Conversion from Mealy Machine to Moore Machine :

Statement :

For every Mealy Machine there exist a Moore Machine that is equivalent to it.

Method :

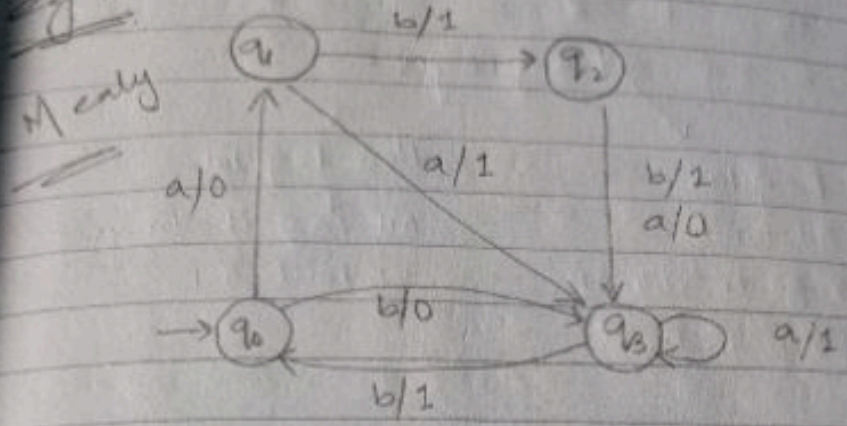
① Incoming transitions have same output character.



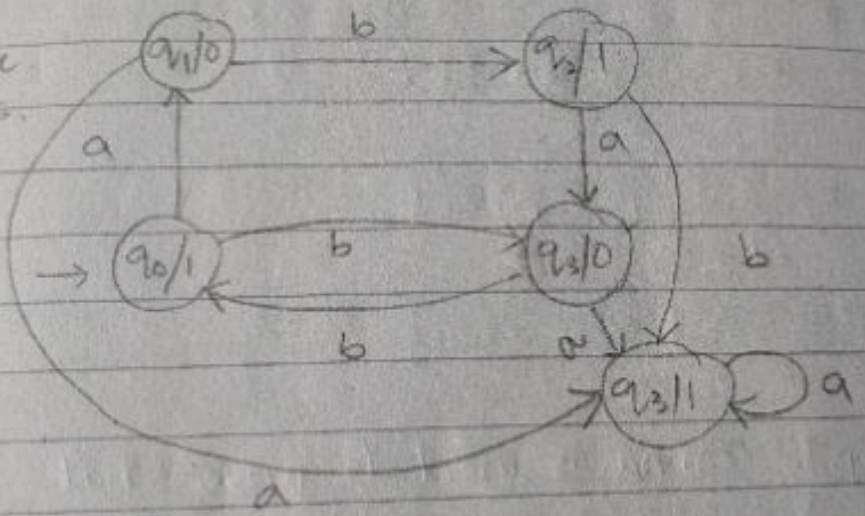
② For loop

Eg #

Mealy



Moore



Read ba:

Mealy: $q_0 \xrightarrow[0]{b} q_3 \xrightarrow[1]{a} q_3$

Moore: $q_0 \xrightarrow{b} q_3 \xrightarrow{a} q_3$; q_0 → 1 (ignore), q_3 → 0, q_3 → 1

} both gives same output.

25/Mar/19

a: Pumping Lemma

Languages

- Regular
  - All finite
  - Infinite with A.P. (Loop in FA)
- Non-Regular
  - All infinite Languages without A.P

eg① $L = \{ab, abab, ababab, abababab, \ldots\}$

$RE = a(ba)^*b$

② $L = \{a^p \mid p \text{ is prime}\}$

③ $L = \{a^n \mid n \text{ is even}\}$

Pumping lemma is used to prove that Language is non-regular using Prove by Contradiction Approach

Let 'L' is a regular language with 'K' states in its Finite State Machine.

Pick one W such that $|W| \geq K$ and split the W into three substrings i.e $W = x.y.z$ with following

conditions:

① $|xy| \leq K$

② $|y| \neq \varepsilon$

③ $W = xy^iz$

Examples: $\Sigma =$

Regular Languages

1) a^n ; $n \geq 1$
2) a^n ; n is even
3) a^n ; n is odd

eg: ① $L = \{a$

② arrest →

① a^n ;

$|W| \geq$

$W = \frac{a}{x}$

i=0 → a.a

i=1 → a.

i=2 → a.

i=3 → a.a

→ Pumping Lemma for complex Languages

conditions:

① $|xy| \leq k$

② $|y| \neq \varepsilon$ (y cannot be empty string).

③ $w = x y^i z \quad \forall i \geq 0 \rightarrow w \in L$.

Then L is Regular.

Examples: $\Sigma = \{a\}$

	Regular Language	Non-Regular Language
1)	a^n ; $n \geq 1$	1) a^n ; n is prime
2)	a^n ; n is even	2) $a^{n!}$; $n \geq 1$ $\{a, aa, \dots\}$
3)	a^n ; n is odd	3) a^{n^2} ; $n \geq 1$ $\{a^1, a^4, a^9, a^{16}, \dots\}$
		4) a^{2^n} ; $n \geq 1$ $\{a^2, a^4, a^8, \dots\}$

eg: ① $L = \{a^{n^2} \mid |w| = 4, \text{ where } n \geq 1\}$ → Reg

② a^{555} → Reg

① a^n ; $n \geq 1$.

$|w| \geq k \quad \therefore k = 2$

$w = \underbrace{a}_{x} \cdot \underbrace{a}_{y} \cdot \underbrace{aaaa}_{z}$

i=0 → a.aaaa ∈ L

i=1 → a.a.aaaa ∈ L

i=2 → a.aa.aaaa ∈ L

i=3 → a.aaa.aaaa ∈ L

// since it is regular it satisfies all i's.

$w = aa.aa.aa$

$w = a.aa.aaa$

} X

b/c $|xy| \not\leq k$

eg: $L = a^{n!}$ $n \geq 1$

Prove L is non-regular

$a^{n!} = \{a, aa, a^6, a^{24}, \ldots\}$

Let $k = 6$

$W = aaaaaa$

$|xy| \leq k$

$W = \underset{x}{aa} \cdot \underset{y}{aa} \cdot \underset{z}{aa}$

$i = 0 \Rightarrow W = aaaa \notin L$

hence proved non-regular

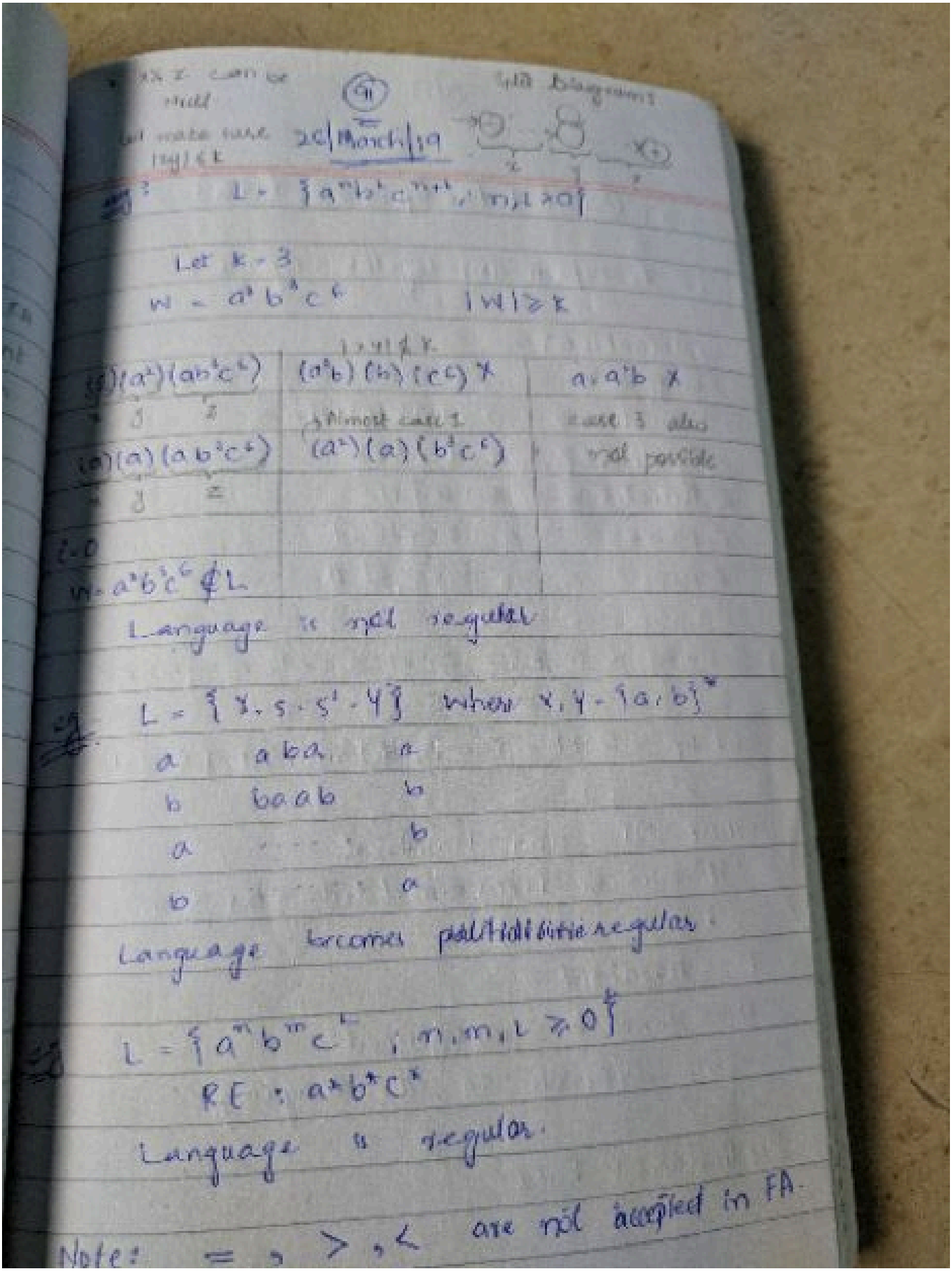
Limitations of
- cannot
- Comparision
eg: $W \cdot W^R$
(palindrome)

eg: $L = a^n b^n$

Let $k = 6$

$W = aaabbb$ $|W| \geq k$

Case #1	Case #2	Case #3
'y' from 'a's	'y' from 'b's	'y' from a's & b's
$W = \underset{x}{(a)}\underset{y}{(a)}\underset{z}{(abbb)}$	$W = \underset{x}{(aaab)}\underset{y}{(b)}\underset{z}{(b)}$	$W = \underset{x}{(aa)}\underset{y}{(ab)}\underset{z}{(bb)}$
$W = \_\_(aa)(abbb)$	$W =$	
$(\|y\| \neq \varepsilon$ & $\|xy\| \leq k)$	$i = 0,$	$i = 0,$
$i = 0,$		
$aabbb \notin L$		



Context Free Grammars (CFG) :-

:- To express non-regular language

Explaination :-

<Sentence> → (Noun Phrase)(Predicate)
<Noun Phrase> → (Article)(Noun)
<Predicate> → (Verb)
<Article> → a | the
<Noun> → Dog | Cat
<Verbs> → Run | Walk

Deriving a sentence from a set of grammar.

CFG = <NT, T, Productions, Start Syntax>

Where NT - non-terminal
"it can be transformed into other form."

T - terminal
"it cannot be transformed"
:- are alphabets in the language.

Productions : Rules.

Start Syntax : Star

eg E → E+E/E-E
E → T
T → 0|1|2| ....

CFG = <E, T> ...

eg S → aSb
S → λ

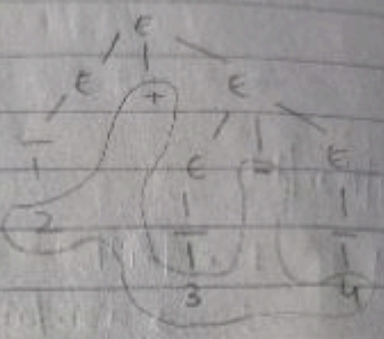
S → λ S → asb
→ ab

CFG = <
L = {λ, ab
Language

Start Syntax : Start Symbol (i.e. NT).

1) $E \to E+E/E-E$
$E \to T$
$T \to 0/1/2/\ldots\ldots/9$

CFG = $\langle E, T \}, \ldots \} \rangle$



2) $S \to aSb$ (NT) → Terminal
$S \to \Lambda$

Snake

$S \to \Lambda$

$S \to asb$
$\to ab$

$S \to aSb$
$\to aaSbb$
$\to aabb$

$S \to aSb$
$\to aaSbb$
$\to aaasbbb$
$\to aaabbb$

CFG = $\langle S, \{asb, \Lambda\}, \{asb, \Lambda\}, S \rangle$.
L = $\{\Lambda, ab, aabb, \ldots\ldots, \}$
Language = $\{ a^n b^n \}$

eg: $S \to Ab$
$A \to aAb$
$A \to \lambda$

$S \to Ab$
$\to aAbb$
$\to abb$

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

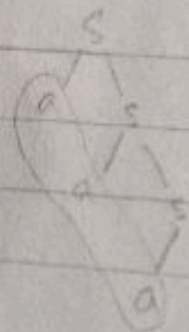
28/Mar/19

eg: $L = \{a^n b^n \mid n \geq 1\}$
$S \to aSb / ab$ → 2 predictions.

$S \to ab$
$S \to aSb \to aabb$
$S \to aSb \to aaSbb \to aaabbb$.
$L = \{ab, aabb, aaabbb, \ldots\}$

eg: $a^+ = \{a, aa, aaa, \ldots\}$
$S \to aS / a$

$S \to a$
$S \to aS \to aa$
$S \to aS \to aaS \to aaa$.



eg: $L = \{aa, ab, ba, bb\}$ → Finite

$S \rightarrow aa|bb|ab|ba$ → common way

R.E - $(a+b)(a+b)$
A A

$S \rightarrow A \cdot A$
$A \rightarrow a|b$.

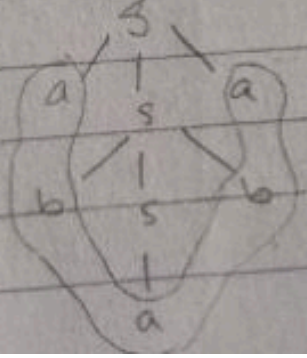
Note: For any language, we can have more than one CFG's.

▷: Syntax Tree ◁

Graphical representation of the derivation.

eg: $L = \{w \cdot w^r | w \in \{a,b\}^*\}$
$L = \{\lambda, a, b, aa, bb, aba, bab, \ldots\}$

$S \rightarrow aSa|bSb|\lambda|a|b$.



eg: $(a+b)^*$

$S \rightarrow aS \mid bS \mid \Lambda$

eg: $a(a+b)^*a + b(a+b)^*b$

$(S \rightarrow aSa \mid bSb \mid aS \mid bS \mid \Lambda)^*$

(OR)

$S \rightarrow aAa \mid bAb$

$A \rightarrow aA \mid bA \mid \Lambda$

eg: $L = \{a^n b^m c^p \mid n, m, p \geq 1\}$

$S \rightarrow$

(OR)

$S \rightarrow ABC$

$A \rightarrow aA \mid a$

$B \rightarrow bB \mid b$

$C \rightarrow cC \mid c$

eg: $L = \{a^n b^m c^n \mid n, m \geq 1\}$

$S \rightarrow aAc$

$A \rightarrow aAc \mid B$

$B \rightarrow bB \mid b$

Grammar

Left linear

eg:

$A \to A\alpha$

$\alpha \in T^*$

OR

$A \to B\alpha$

$\alpha \in T^*$

Right Linear

eg:

$A \to \alpha A$

$\alpha \in T^*$

OR

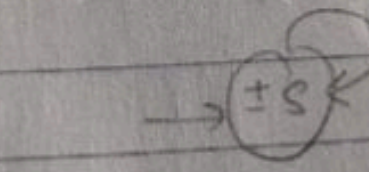
$A \to \alpha B$

$\alpha \in T^*$

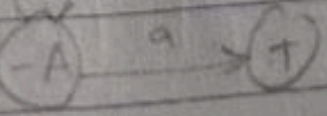
- If your grammar is either in Left linear grammar or Right Linear grammar than it express regular language.
- If the grammar is in Right linear form then we can built an FA.

eg: $(a+b)^*$

$S \to aS|bS|\wedge$



eg: $A \to aA|a$



Derivation

LMD

(Replacing Left most NT with its prediction in each step)

S → ABC
→ aABC
→ aaBC
→ aabC
→ aabcC
→ aabcc

RMD

(Replacing Right most NT with its predict. in each step).

S → ABC
→ ABcC
→ AbcC
→ Abcc
→ aAbcc
→ aabcc

→ Ambiguous

CFG is ambiguous if some string w belongs to L(G)