Theory of Computation Notes

Contributor: Riya Goel

KMV (DU)]

Computer Science Notes

CLASSTIME | Page No.
Date / /

23/8/18 TOC

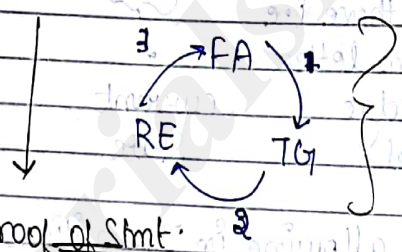
Ch-7
Kleen's Theorem

→ There are 3 seperate ways of defining a language ② generation by regular expression and acceptance by Finite Automata, and acceptance by ③ Transition Graph. ①

* Language generators
  - R.E
  - F.A
  - T.G

* Language acceptors
  - DFA
  - NFA

Theorem- Any language that can be defined by regular expression or finite automata or transition Graph can be defined by all the 3 methods.



all 3 are equivalent.

Proof of Stmt.
(divided into 3 parts)

- Part I says every language that can be defined by Finite Automata can be defined by Transition Graph.
- Part II says every language defined by a TG can also be defined by a regular

CLASSTIME | Page No.
Date / /

expression

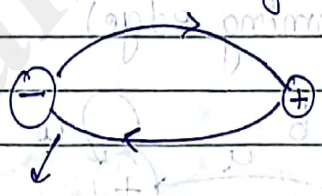
- Part III says every language that can be defined by R.E can also be defined by F.A.
  {making graph from expressions}.

Proof of Part-I :-
Every F.A is itself already a transition graph.
F.A = restricted transition graph.

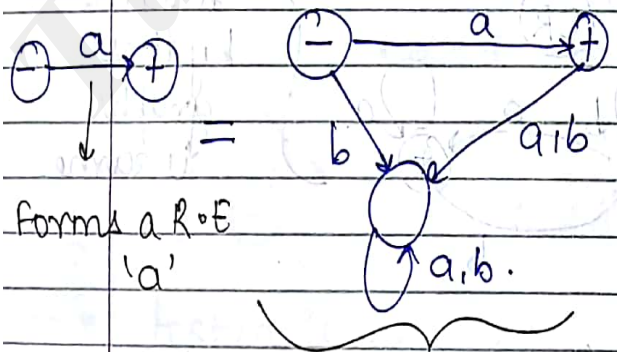
Proof of Part-II :-
This proof can be given by constructive algorithm.

→ Constructive Algorithm- We present a procedure that starts out with a transition graph & ends up with a regular expression that defines the same language.

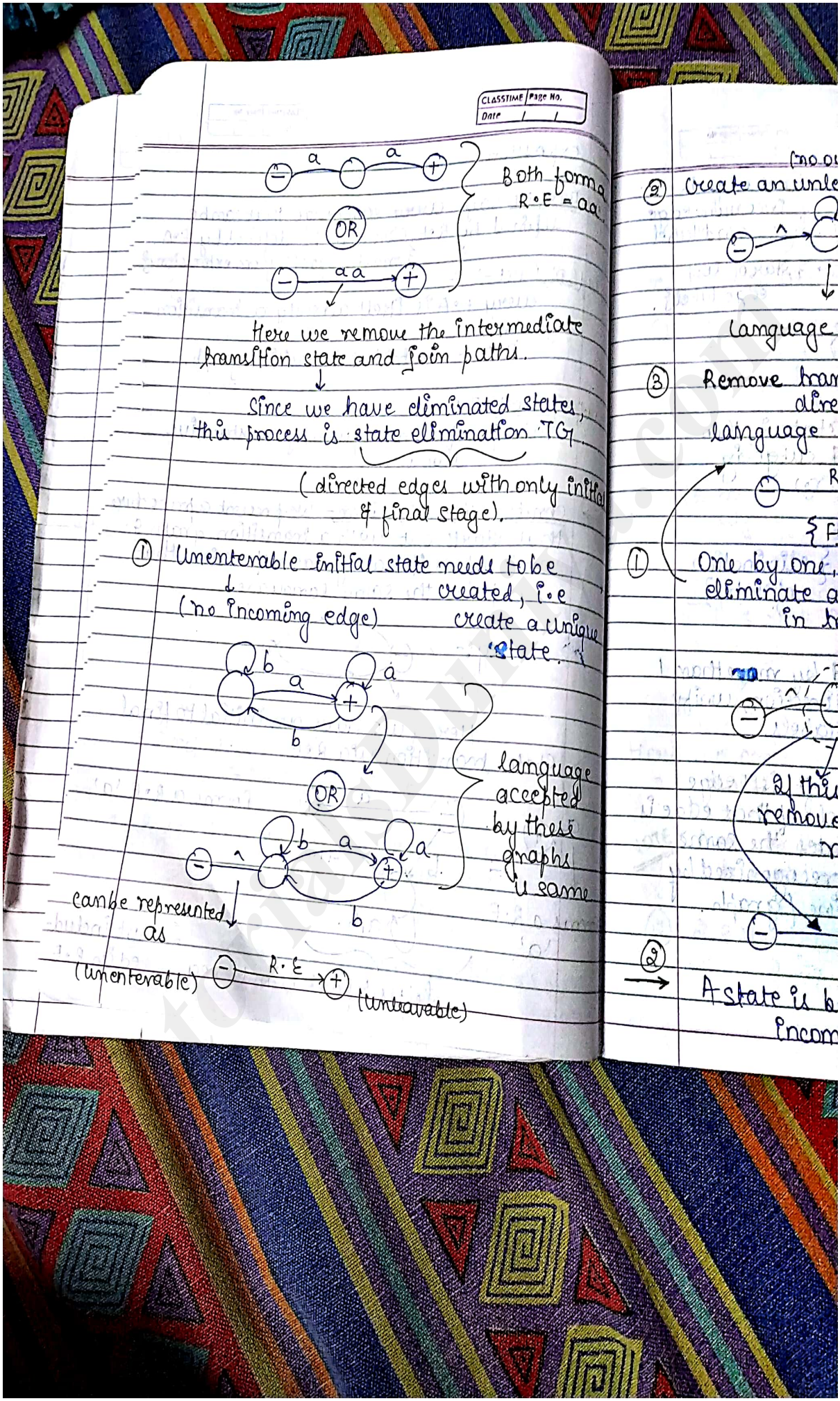
eg :-



Here, we have an initial to final State transition via R.E

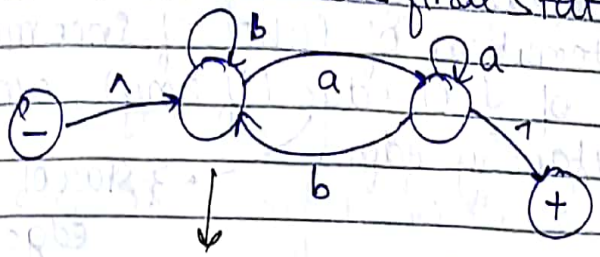


∴ not included
left out part is not required in R.E



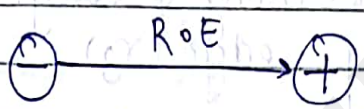
(no outgoing edge)

(2) Create an unleavable final state.



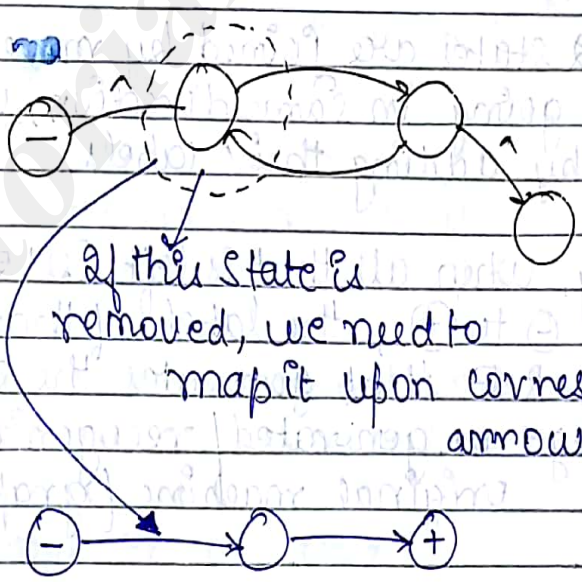
Language accepted is same.

(3) Remove transition states or remove a directed edge from graph s.t language accepted remains same



{Final Steps}

(1) One by one, in any order bypass & eliminate all non ⊖ve and ⊕ states in transition graph.



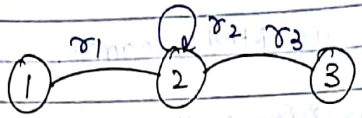
If this state is removed, we need to map it upon corresponding arrow (label)

(2) → A state is bypassed by connecting each incoming edge to outgoing edge.

CLASSTIME | Page No.
Date / /

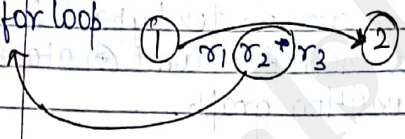
and the label of each resultant edge is concatenation of label of incoming edge, label of loop edge (if any), and label of outgoing edge. → {star of loop edge label}.

eg:-



If we remove state 2, then we include incoming & outgoing edge (r_1) (r_3)

Here, we have r_2^* for loop label

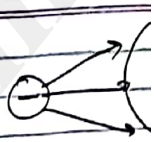


{elimination of states}

③ When 2 states are joined by more than 1 edge going in same direction, unify them by adding their labels.

④ Finally, when all that is left is 1 edge from ⊖ to ⊕, the label of that edge is a R.E that generates the same language as generated/recogonized by original machine/graph.

eg:-



* Cases of Ob

① TL represen
It that
circling b

r

(incoming edge)

Step 1 - Re

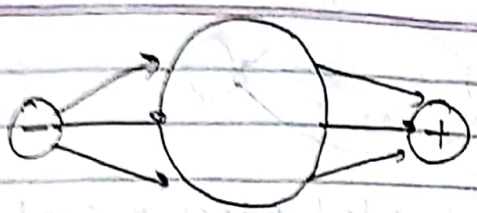
Here we can't
no fixe

NOTE:- Here we
haven't

② 2 states a
go

CLASSTIME | Page No.
Date / /

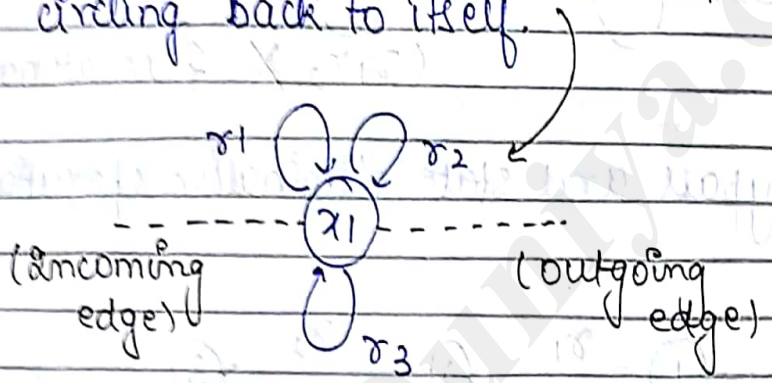
eg:-



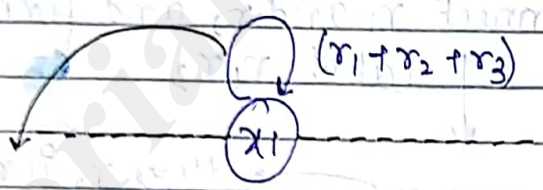
T (a graph)

* Cases of Obtaining A Graph :-

① T (represents a graph) has some states inside it that have more than 1 loop circling back to itself.



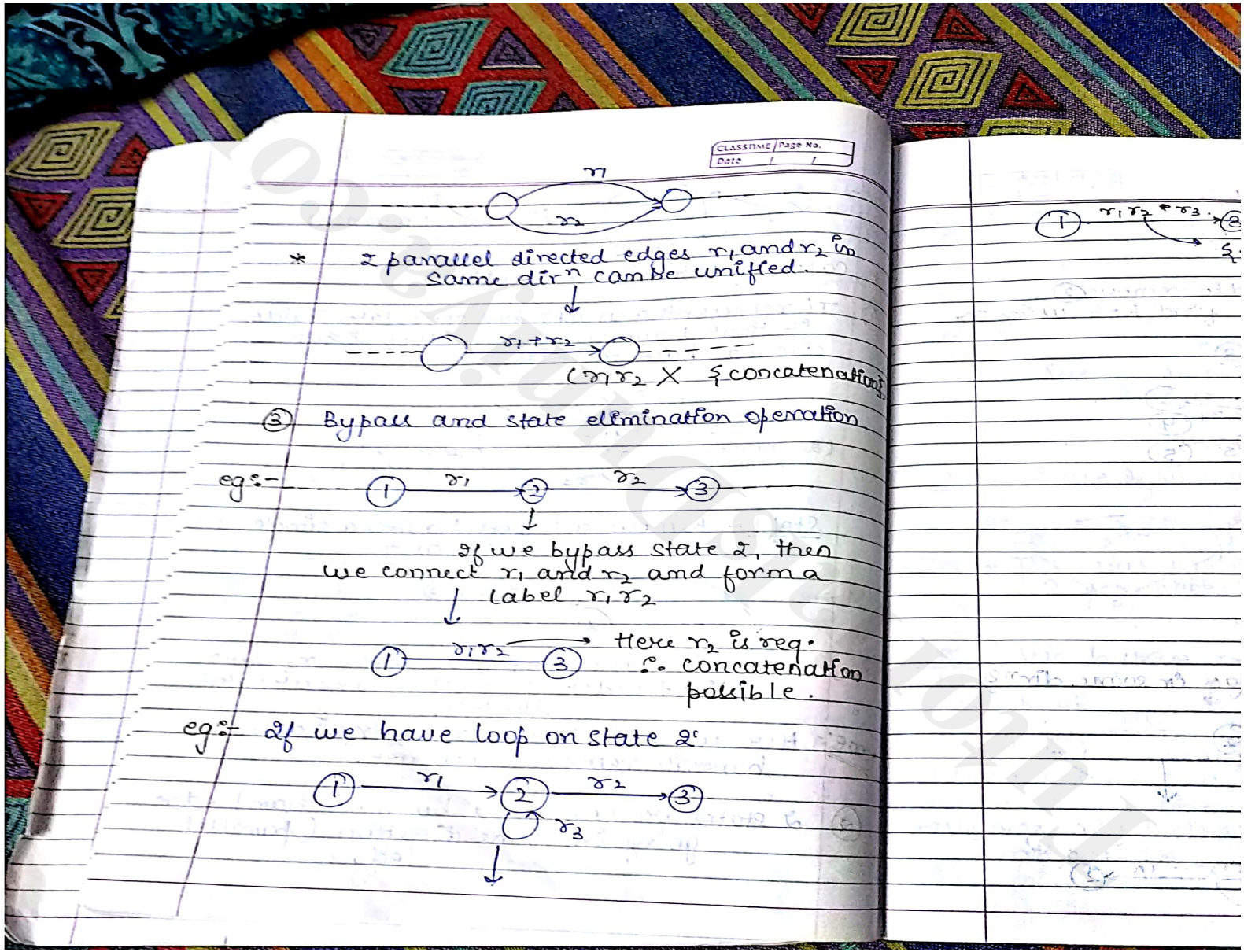
Step 1 - Remove all loops having a single loop labelled as ↓

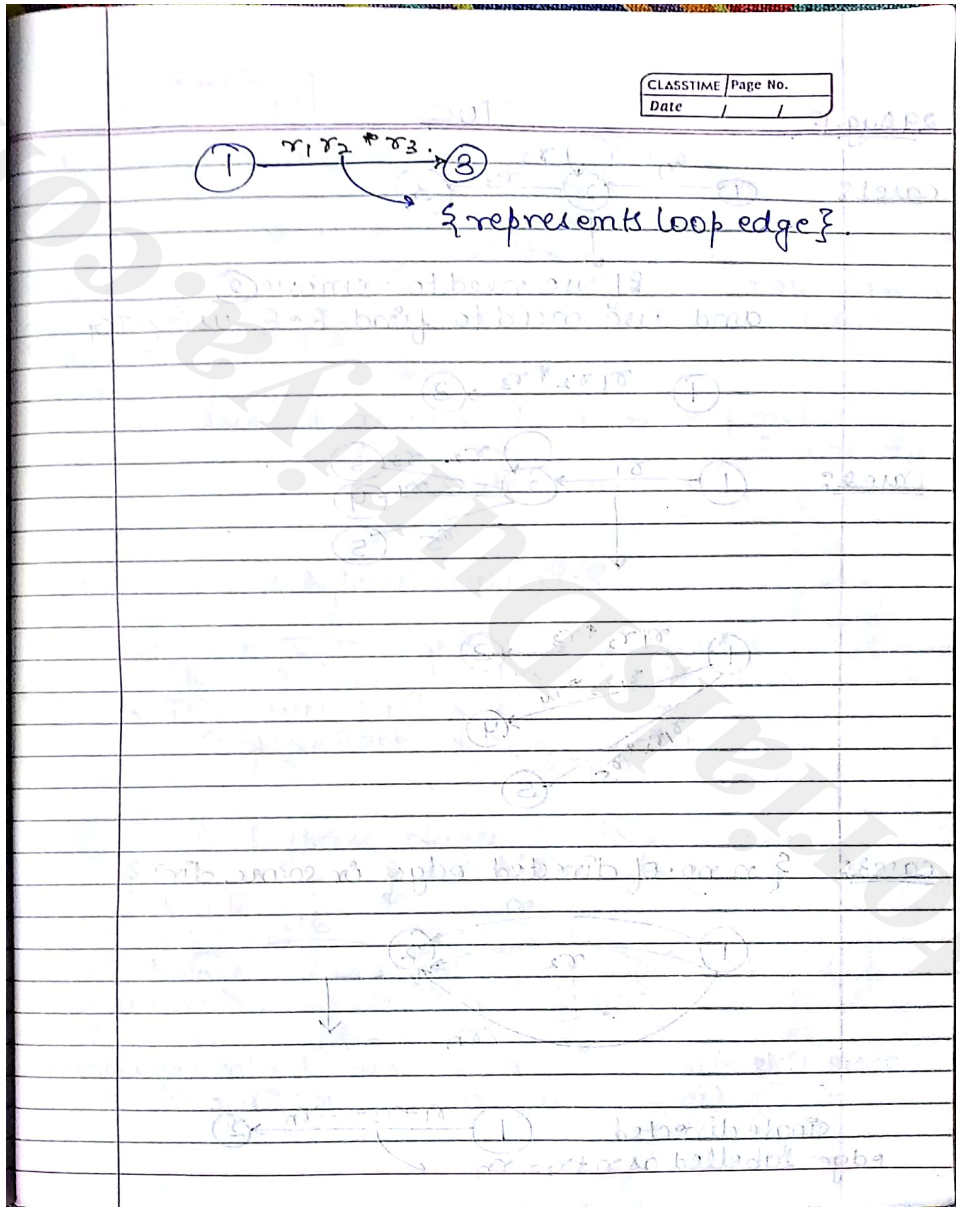


Here we can't have $r_1 r_2 r_3$ as r_1, r_2, r_3 have no fixed order ∴ can't be concatenated like this.

NOTE: Here we won't have $(r_1 + r_2 + r_3)^*$ as we haven't removed loop yet.

② 2 states are connected by more than 1 edge going in same direction (parallel edges)





TOC

29 Aug 18

Case1:

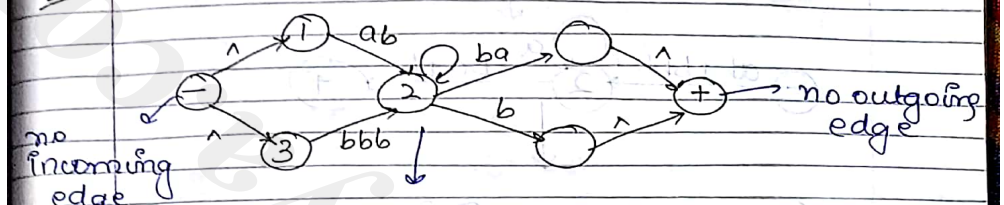
Case2:

Case3: { n no. of directed edges in some dirn }

single directed edge labelled as $x_1 + x_2 + \dots + x_n$

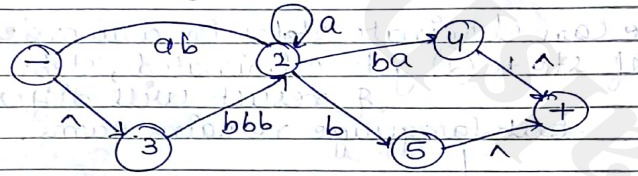
CLASSTIME | Page No.
Date / /

Q Consider a graph :-

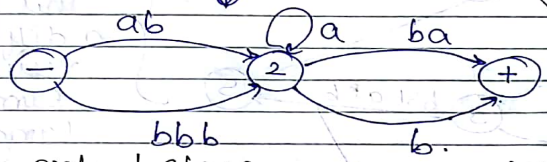


We can remove states in any order and we are left with (1) and (4) states

I { we bypass state (1) }



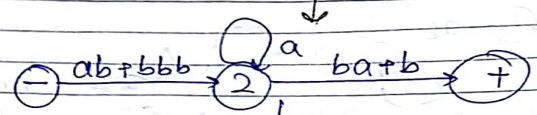
II { We bypass stages (3), (4), (5) }



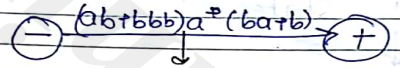
{ We do this by removing only 1 stage at one step.

can't skip state (3), (4), (5) in one step.

Now removing parallel edges first

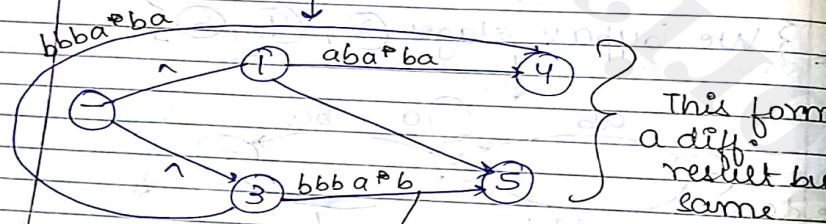


{ Bypass state (2) }



This forms the final R.E

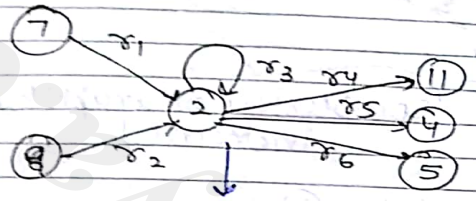
NOTE:- We can eliminate states in any order. If state (2) gets eliminated, steps & result will differ but language remains same.



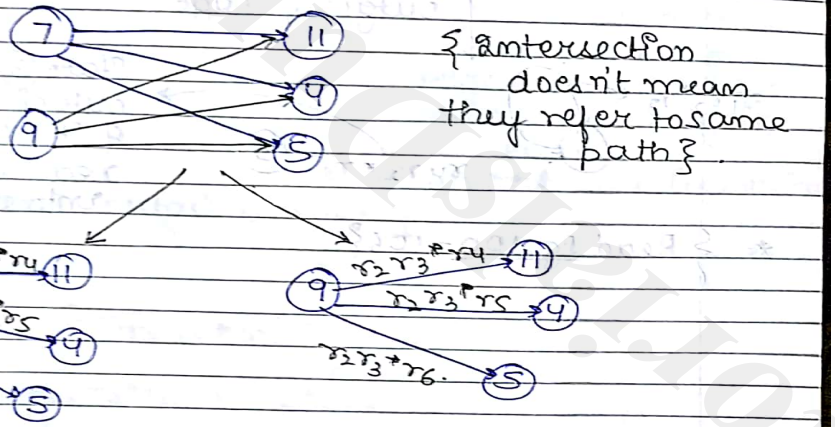
We have a path here as (3) → (2) → (5)

CLASSTIME | Page No.
Date / /

Q consider another graph :-



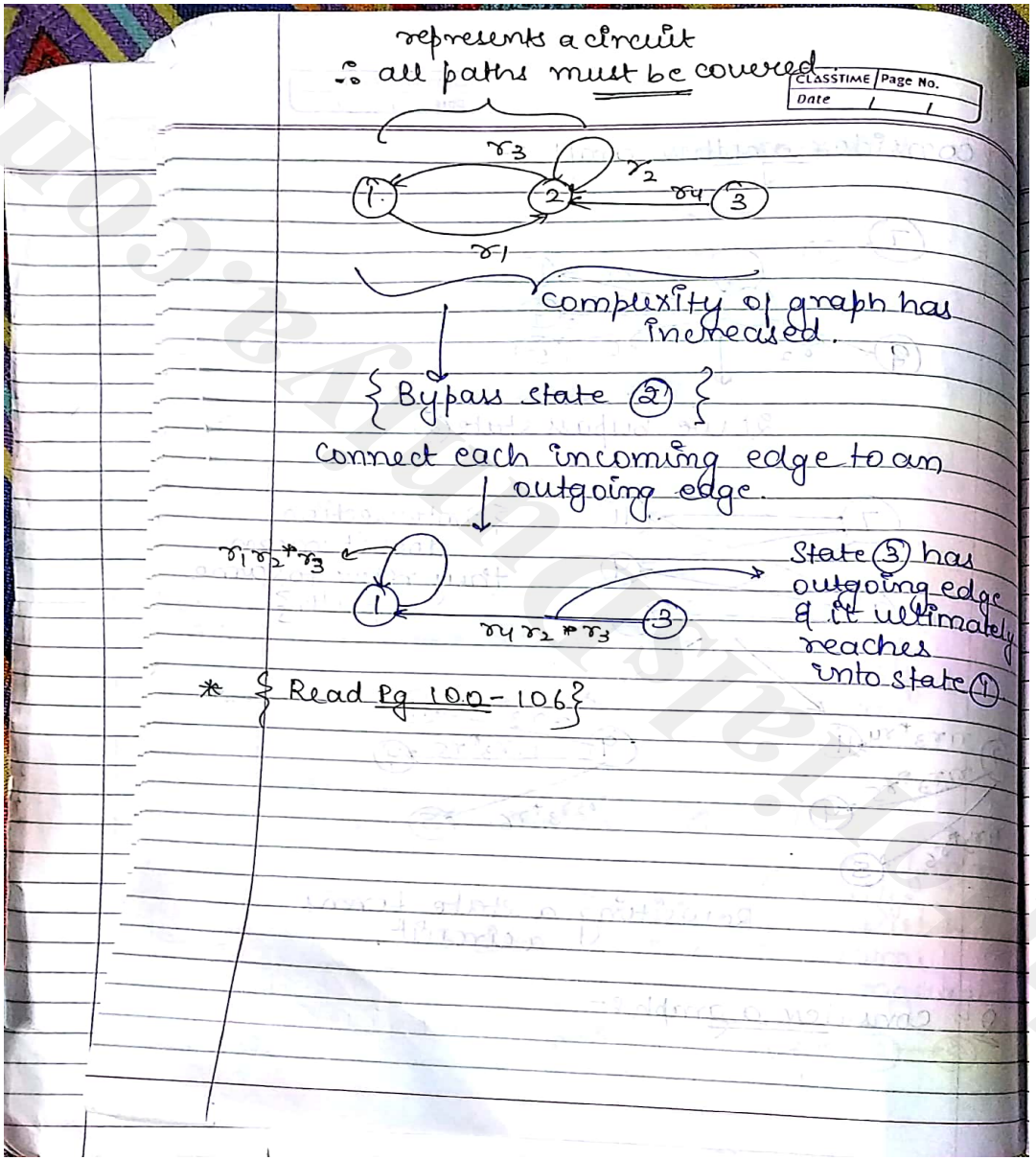
If we bypass state (2)



{ intersection doesn't mean they refer to same path }.

Revisiting a state forms a circuit.

Q consider a graph :-



TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

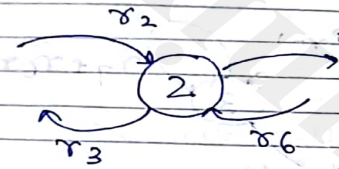
Please Share these Notes with your Friends as well

30 Aug, 18

CLASSTIME | Page No.
Date / . /

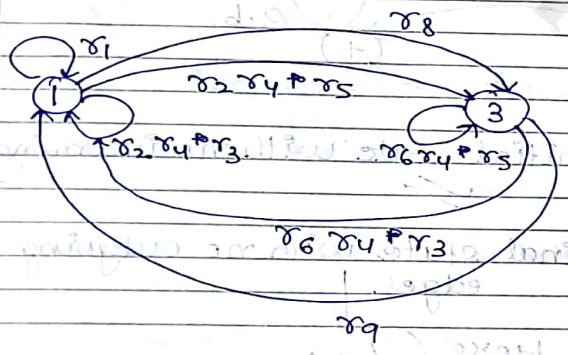
Q Consider a graph ↓

I Bypass state (2).

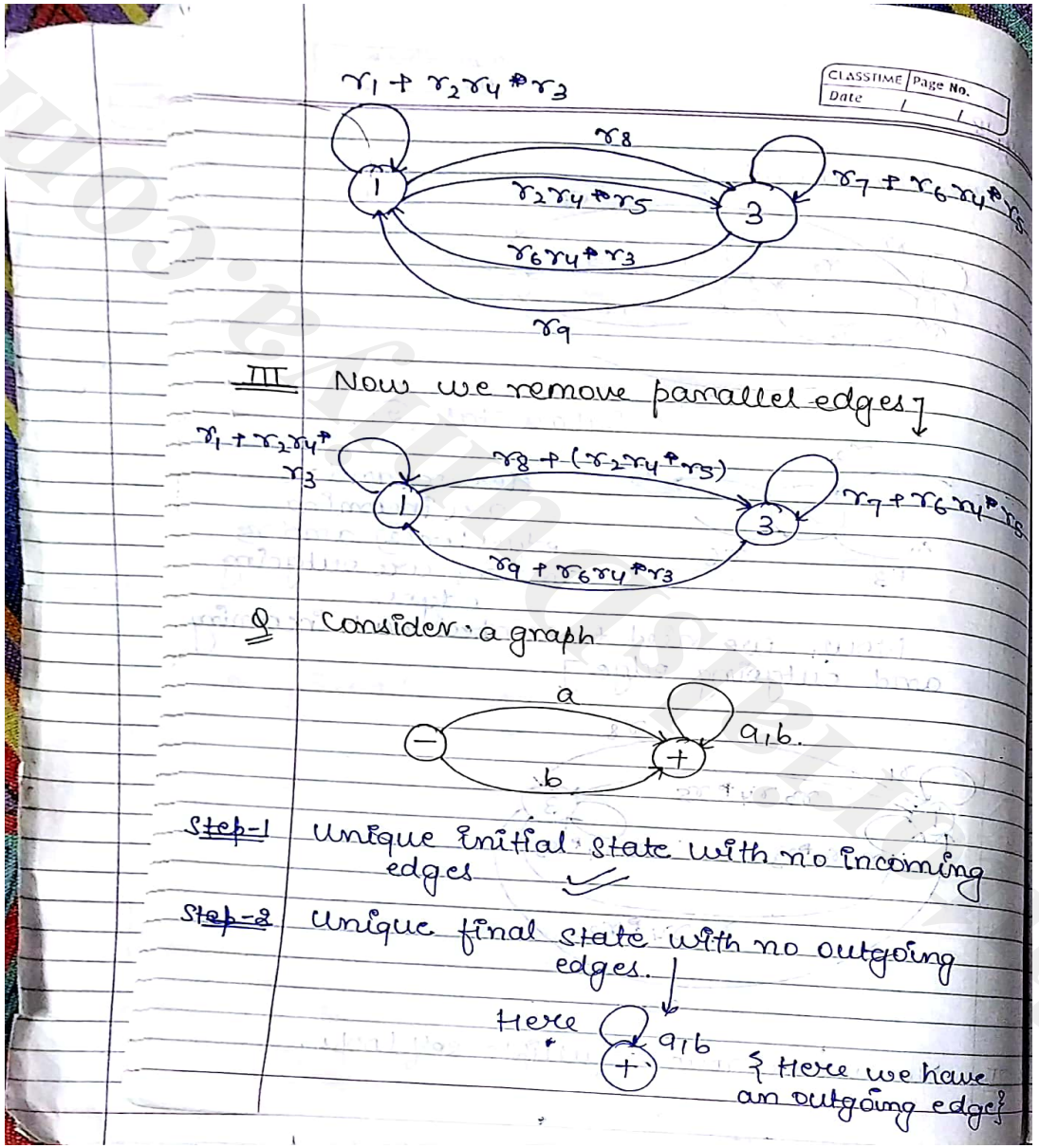


Here r_2 and r_6 are incoming edges for (2) and r_5 and r_3 are outgoing edges.

Now, we need to connect each incoming and outgoing edge ↓

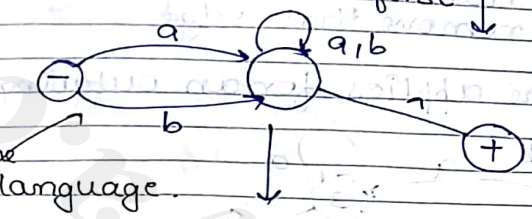


II Now we remove multiple self loops.



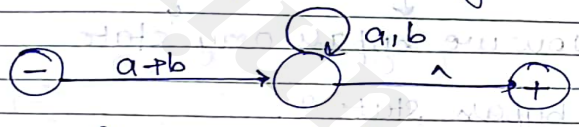
CLASSTIME | Page No.
Date / /

∴ We remove it first ↓

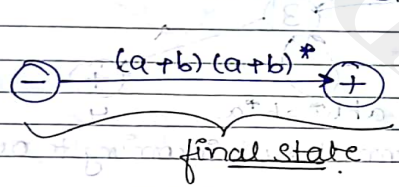


They define same language.

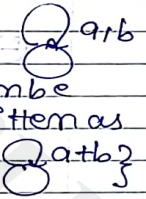
Remove parallel edges.



Remove self loops and bypass middle state.



{ as can be written as

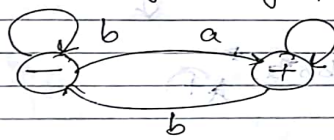


}

NOTE:

→ loop edge can both be considered as incoming or outgoing edge. depending upon initial & final state.

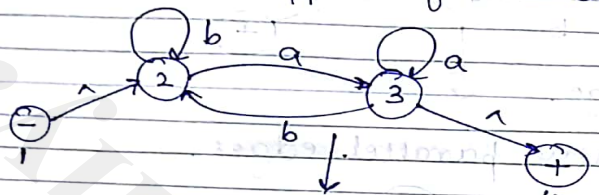
Q Convert the given graph into R.E.



CLASSTIME | Page No.
Date / /

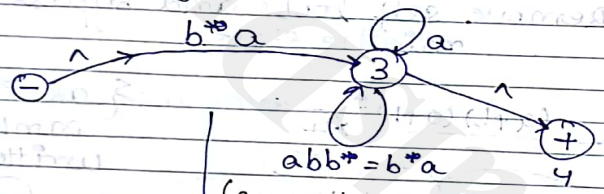
Here we have an incoming edge
∴ remove that edge

and same applies for an outgoing edge



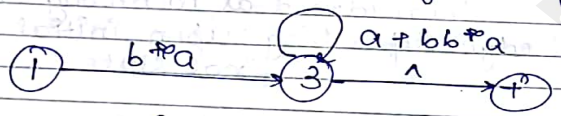
Now we bypass any state

First bypass state 2.

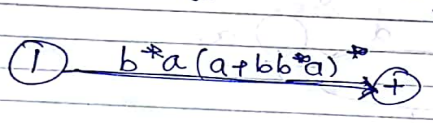


(connect each incoming to outgoing)

Now make a single self loop.

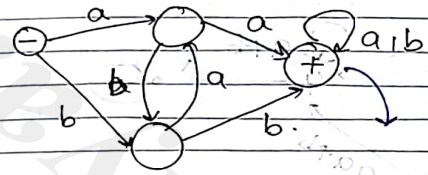


Bypass state (3)

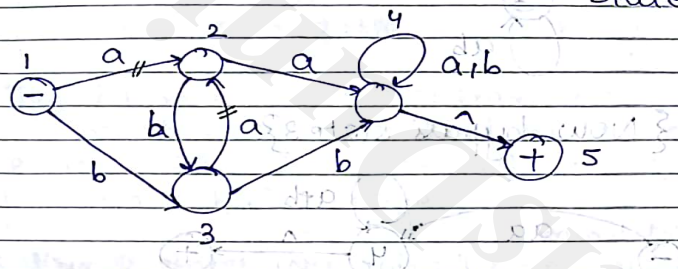


CLASSTIME | Page No.
Date / /

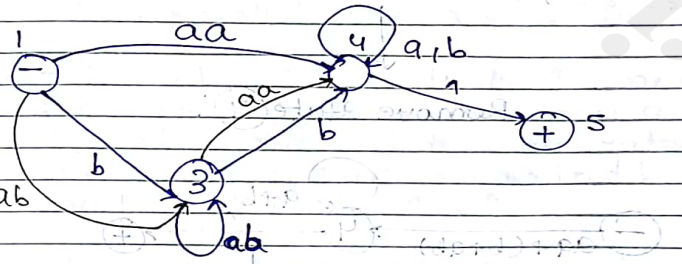
Q Consider a graph ↴



We have an outgoing self loop in final State.



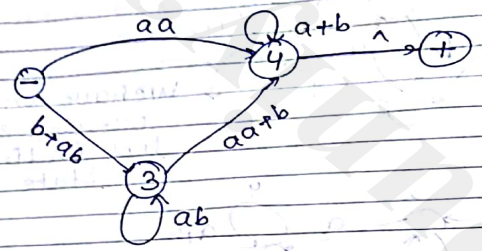
{ Bypass State 2 } ↴



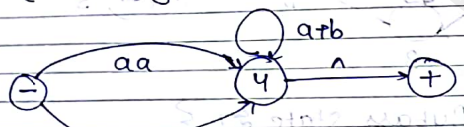
{ Bypass state 3 }

CLASSTIME | Page No.
Date / /

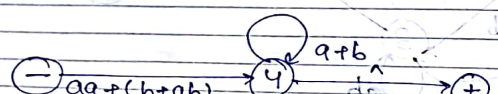
Before bypassing, remove all parallel edges.

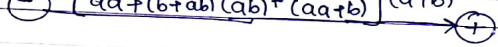


{Now bypass state 3}



Remove state (4).





Constructive A

NOTE- The constructive al
all transition gra
that define exac

Step1 - Create a unique
unique unlea

Step2 - One by one, in a
all non (−ve) o
transition.

Step3 - The label of ea
concatenation
edge, with la
on outgoing

Step4 - When 2 states ar
going in same
adding thei
{ parallel

Step-5 - Finally when al
from (−) to (+)
edge represents
same languag
machine } i.e

* Proof of Part III :-
Recursive app
(defi

(hardest part)

Constructive Algorithm

CLASSTIME | Page No.
Date / /

NOTE - The constructive algorithm that proovves that all transition graphs can be turned in R.E that define exact same language.

Step1 - Create a unique unenterable (−) state & a unique unleavable (+) state.

Step2 - One by one, in any order bypass & eliminate all non (−ve) or (+ve) states in the transition graph.

Step3 - The label of each resultant edge is the concatenation of label on incoming edge, with label on loop (if any) & label on outgoing edge.

Step4 - When 2 states are joined by more than one edge going in same direction, unify them by adding their labels.
{ parallel edges are unified }

Step-5 - Finally when all that is left is one edge from (−) to (+) state, the label on the edge represents R.E that generates the same language as recognized by original machine { i.e F.A }.

* Proof of Part III :- This proof is given by Recursive approach & constructive algo. (definition) at the same time.
(handest part)

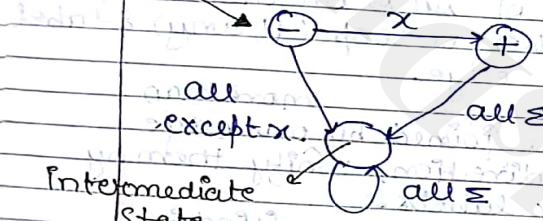
Theorem.

CLASSTIME | Page No. | Date

Every R.E can be built up from the letters of the alphabets (Σ) and $\wedge$ (null) by repeated application of certain rules that are :-

- Addition (+)
- concatenation (.)
- closure (*)

Rule1 :- There is a F.A that accepts any particular letter of the alphabet, and there is a F.A that accepts only a word null ($\wedge$)



Σ may contain any no. of letters i.e $x \in \Sigma$

neglected / rejected paths.

F.A that accepts $\wedge$

neglected / rejected paths.

6 Sept, 18

CLASSTIME | Page No. | Date / /

- Proof of part III

$R \cdot E \longrightarrow F \cdot A$

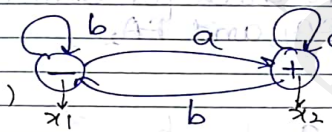
Rule 2:- If there is a finite automata FA_1 that accepts (Union the language defined by Regular Expression of F.A) r_1 and there is a finite automata called FA_2 that accepts language defined by Regular expression r_2, then there is a F.A FA_3 that accepts language defined by R.E $r_1 + r_2$

r_1 —— FA_1 } union of 2 F.A

r_2 —— FA_2

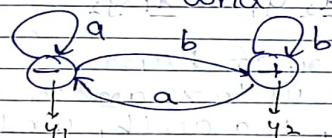
$r_1 + r_2$ —— FA_3

eg:-



(FA_1) $(a+b)^* a$ (r_1)

and



(FA_2) $(a+b)^* b$. (r_2)

⟶ Now for FA_1 we draw transition table

	a	b
x_1 −	x_2	x_1
x_2 +	x_2	x_1

⟶ Now transition table for FA_2

CLASSTIME | Page No.
Date / /

	a	b
y_1^-	y_1	y_2
y_1^+	y_1	y_2

→ Now we design FA_3

↓

ababab...

↙

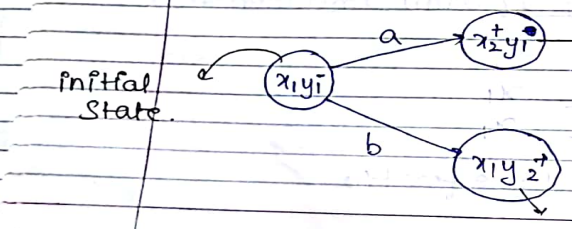
We need to know the last letter of string so that we can decide whether it belongs to FA_1 or FA_2

Thus, we need to check for combinations in both FA_1 and FA_2

(1)

	a	b
x_1y_1	x_2y_1	x_1y_2

If we check for x_1 in FA_1 it takes us to x_2 and x_1 for a & b. Similarly for y_1 to a and b, we have y_1 and y_2.

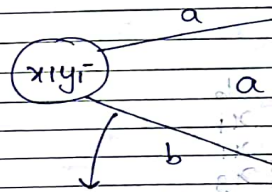


If this is final state it will be accepted by FA_1 as x_2^+ is final state

If this is final state, it is accepted by FA_2.

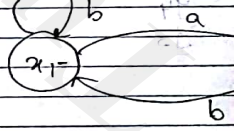
Now we consi
the new

z_1	$x_1y_1^-$	x_2
z_2	$x_1y_2^+$	x_2
z_3	$x_2y_1^+$	x_2



FA_3 that accepts $R_1 \cap R_2$.

Q Consider FA_1



and FA_2 as

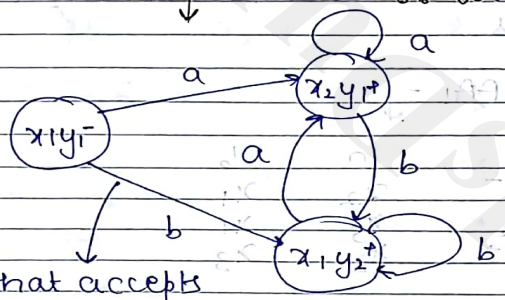
CLASSTIME | Page No.
Date / /

Now we consider x_2y_1 and x_1y_2 as the new combination

↓

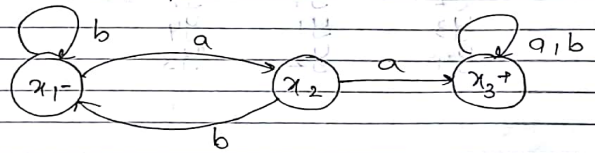
		a	b
z_1	$x_1y_1^-$	x_2y_1	x_1y_2
z_2	$x_1y_2^+$	x_2y_1	x_1y_2
z_3	$x_2y_1^+$	x_2y_1	x_1y_2

No new combination
∴ We stop here.

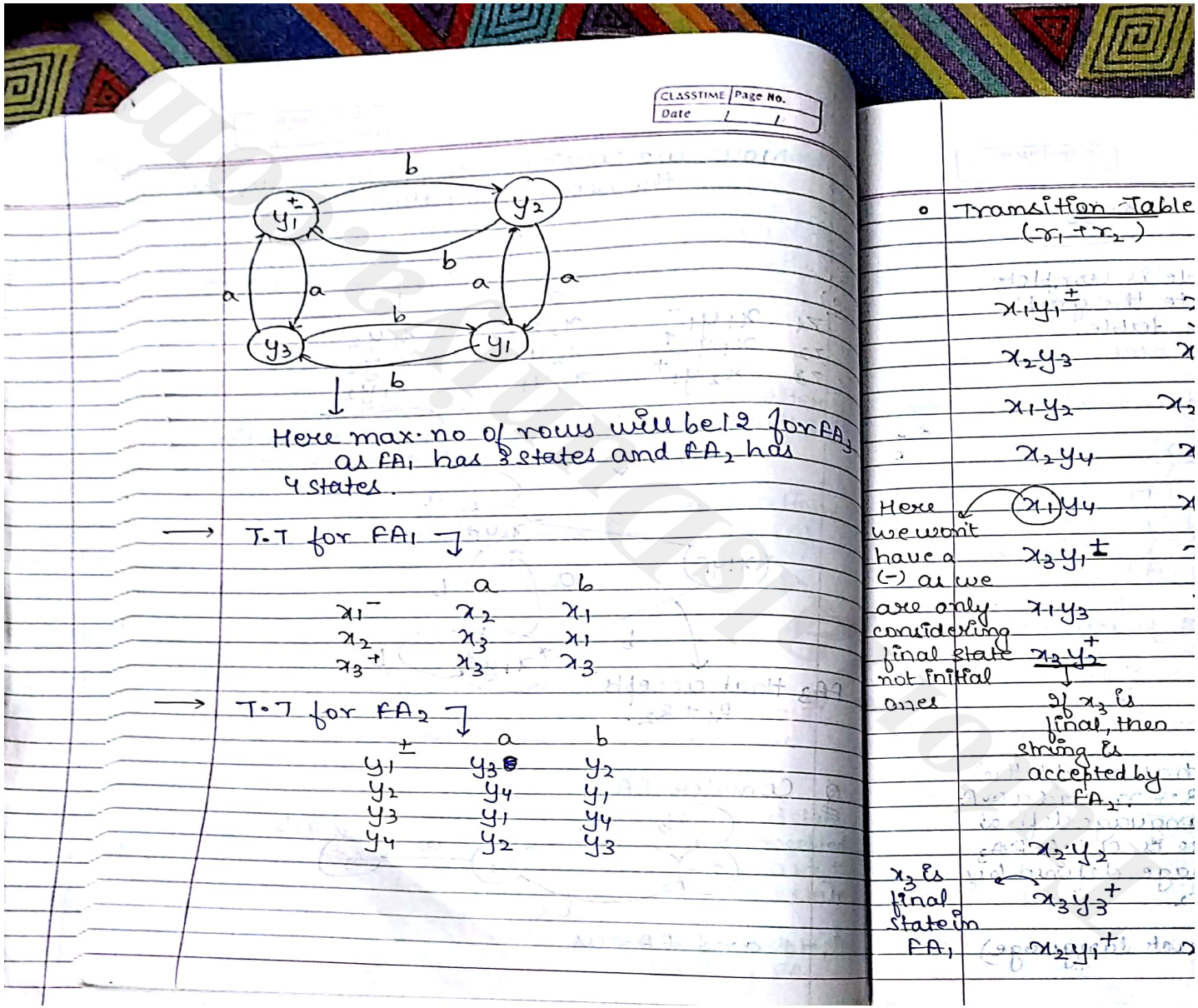


FA_3 that accepts $R_1 + R_2$.

Q Consider FA,



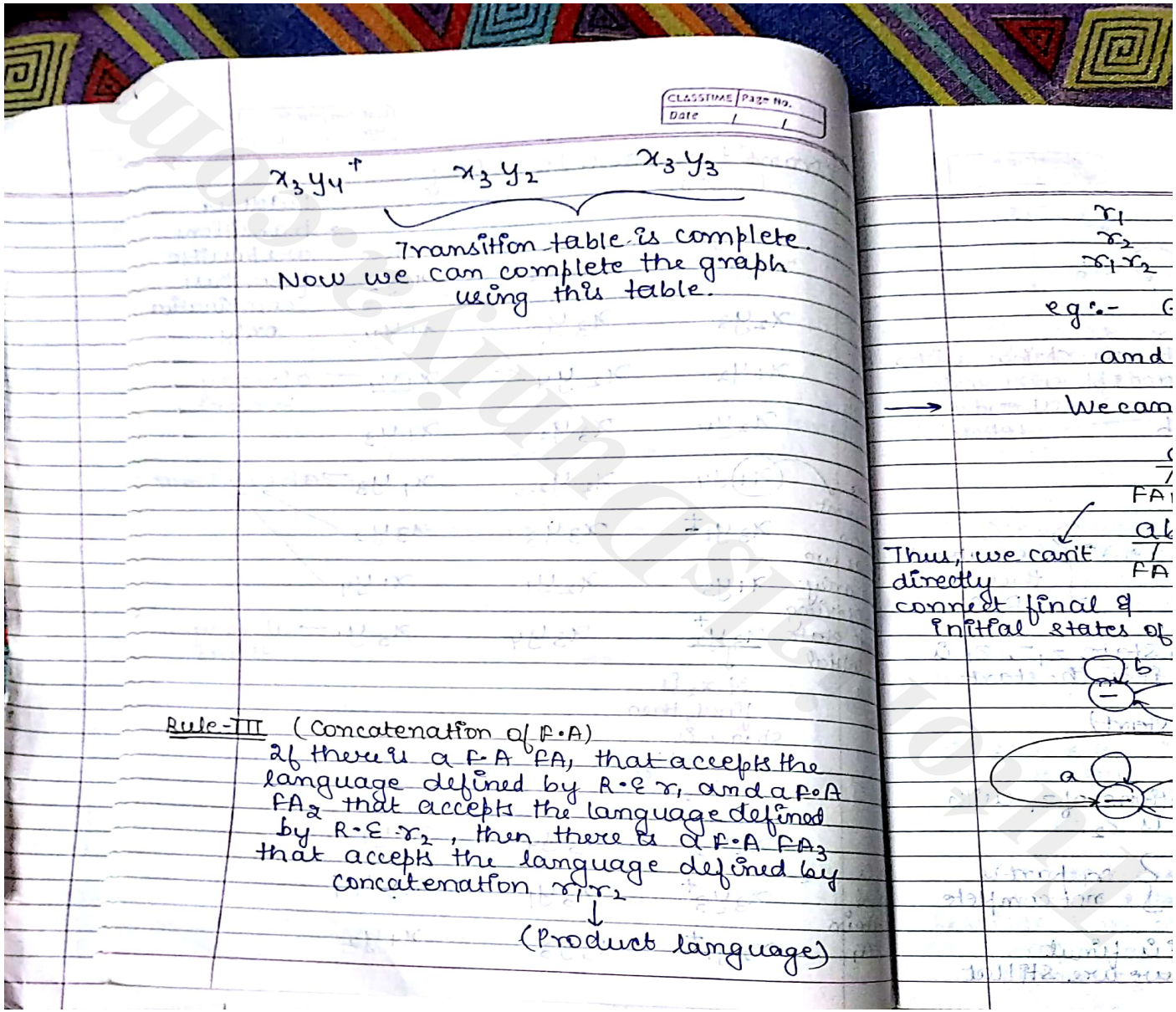
and FA_2 as



CLASSTIME | Page No.
Date / /

- Transition Table for FA3 ↓ $(x_1 + x_2)$

	a	b	
$x_1y_1^{\pm}$	x_2y_3	x_1y_2	Further transitions are possible from these combination only.
x_2y_3	x_3y_1	x_1y_4	
x_1y_2	x_2y_4	x_1y_1 → already present.	
x_2y_4	x_3y_2	x_1y_3	
Here we won't have a (-) as we are only considering final state not initial ones — x_1y_4	x_2y_2	x_1y_3 → already present	
$x_3y_1^{\pm}$	x_3y_3	x_3y_2	
x_1y_3	x_2y_1	x_1y_4	
$x_3y_2^{+}$ — If x_3 is final, then string is accepted by FA_2.	x_3y_4	x_3y_1 → already there.	
x_2y_2	x_3y_4	x_1y_1	
x_3 is final state in FA_1 — $x_3y_3^{+}$	x_3y_1	x_3y_4	
$x_2y_1^{+}$	x_3y_3	x_1y_2	



$x_3 y_4$ $x_3 y_2$ $x_3 y_3$

Transition table is complete. Now we can complete the graph using this table.

Rule-III (Concatenation of F.A)

If there is a F.A FA_1 that accepts the language defined by R.E r_1 and a F.A FA_2 that accepts the language defined by R.E r_2, then there is a F.A FA_3 that accepts the language defined by concatenation $r_1 r_2$

↓

(Product language)

r_1

r_2

$r_1 r_2$

eg:-

and

We can

FA

Thus, we can't directly connect final & initial states of

b

a

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

CLASSTIME | Page No.
Date / /

r_1 FA1
r_2 FA2
$r_1 r_2$ FA1 FA2 → FA3 overlapping not possible.

eg:- Given ab*
and b*a
and string is abbba

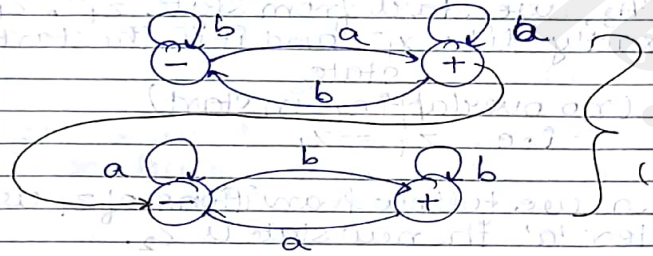
→ We can have many possibilities.

abbba
FA1 FA2

abbba
FA1 FA2

Here problem is at which letter we need to change from FA1 to FA2.

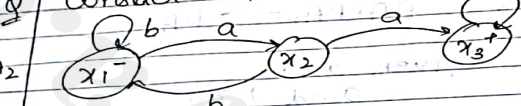
Thus, we can't directly connect final & initial states of 2 automata.



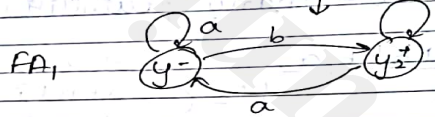
(can't directly do this).

7 Sept, 18

Q Consider FA_1 as



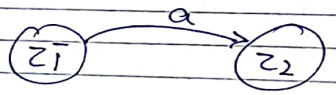
The machine that accepts only strings with 'aa' in them and FA_2 accepts all words at end in letter 'b'



$r_1 \rightarrow (a+b)^* aa (a+b)^*$
$r_2 \rightarrow (a+b)^* b$
} concatenate these 2 F.A

→ For FA_3, we start from state z_1^-. z_1 is exactly like x_1^- and it is the start state
(no overlapping in start)
i.e $z_1 = x_1$

When we follow transitions of z_1 with letter 'a' the new state is z_2.

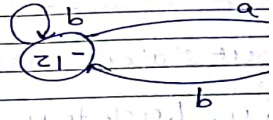


} r_1 part is not complete

→ Now $z_2 = x_2$ and x_3^+ is final
And with letter b, we are still at

z_1^-

→ If we are in z
letter a, th
and if b is



→ Now for state
still running
we have begi
(z_3 is th

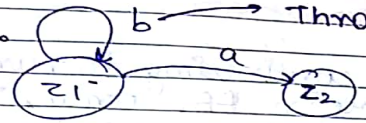
$z_3 =$

→ If we are in
we have n

If we have a
then it might
possible that r_1 i
completed
(When $z_3 = x_3$

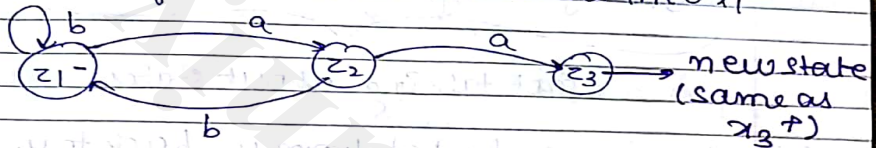
CLASSTIME | Page No.
Date / /

z_1^- ∴



Through b, we are only in z_1^-

→ If we are in z_2 (intermediate state) with a letter a, then we are into x_3^+ and if b is read, we come into x_1^-



→ Now for state z_3:- z_3 is x_3 and we are still running on FA_1 and/or y_1 and we have begin to run on FA_2.
(z_3 is the deciding state)

$z_3 =$ { $x_3 \longrightarrow FA_1$; $y_1 \longrightarrow FA_2$ } → x_1 is still not complete

→ If we are in state z_3 and we read an 'a', we have now 3 possible interpretations for the state

If we have a then it might be possible that x_1 is completed.
(When $z_3 = x_3^+$)

If we have an 'a' in z_3 i.e y_1, then we are in y_1^- acc. to FA_2
(When $z_3 = y_1^-$)

CLASSTIME | Page No.
Date / /

• Possibilities -

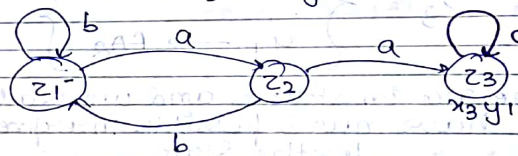
(1) We are continuing to run of state x_3 and are in FA_1 only. The string is still running on FA_1.

(2) We have just finished FA_1 & are now in y_1 beginning to run on FA_2

not talking about states.

(3) We have looped from y_1 back to y_1 while already running on FA_2 i.e starting from y_1, reading 'a' and looping back to y_1.

Possibilities of next state are only x_3 and y_1



→ If we are in state z_3 and we read a 'b'

↓

From z_3, if b is read we have $x_3 y_1$ and along with that a new state.

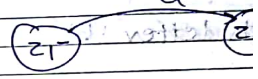
∴ for b, we

• Possibilities -

(1) We are still in

(2) We have just fi
now in y

(3) We are now
reached thei



→ (x_3, b) =

role

→ (y_1, b) →

→ If we are in
we h

(1) Remaining
or

CLASSTIME | Page No.
Date / /

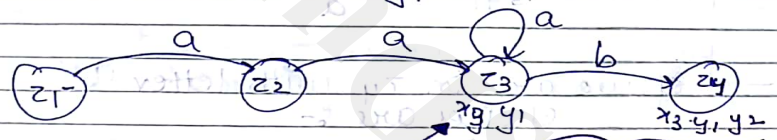
∴ for b, we must have a new state which is z_4 (or y_2 +)

- Possibilities -

(1) We are still in x_3 continuing to run on FA_1

(2) We have just finished running on FA_1 & are now in y_1 on FA_2.
+

(3) We are now in y_2 on FA_2 having reached their via y_1



→ $(x_3, b) \longrightarrow x_3, y_1$ due to overlapping.

as soon as we come on x_3, role of y_1 comes automatically.

→ $(y_1, b) \longrightarrow y_2$ ∴ 3 choices ↓

$\{x_3, y_1, y_2\}$

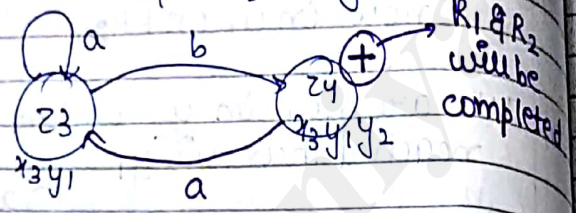
→ If we are in z_4, and we read letter 'a' we have choices as :-

(1) Remaining in x_3 & continuing to run on FA_1

(2) Having just finished FA_1 & beginning at y_1 in FA_2

(3) Having moved from y_2 back to y_1 in FA_2

Thus, choices of states are only b/w x_3 and y_1



→ If we are in z_4 with letter 'b' choices are :-

(1) Remaining in x_3 & continuing to run on FA_1.

r_2 is completely inside r_1.

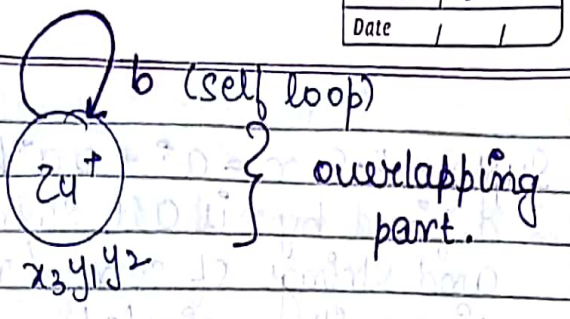
eg:- $(a+b)^*a$ and aa^* :)

completely in $(a+b)^*a$

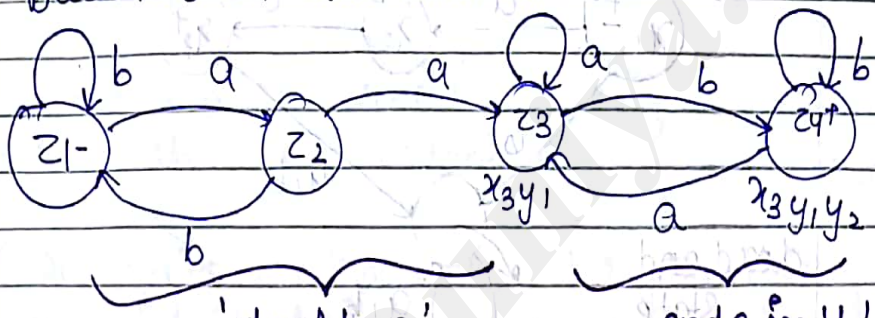
(2) Having just finished FA_1 & beginning at y_1 in FA_2.

(3) Having loop back from y_2 to y_2, running on FA_2

→ We hav
exactl
back
b
z-
Rule-4 :- If
that
R then
accep
If
NOTE:



→ We have produced a machine that accepts exactly those strings that have a front section with a 'aa' followed by a back section that ends in 'b'.



Rule-4 :- If R is a R.E and FA$_1$ is a finite automata that accepts exactly language defined by R then there is a F.A called FA$_2$ that will accept exactly the language defined by R^* {Kleen's closure}.

r F.A

r^* $(F.A)^*$ } F.A closure

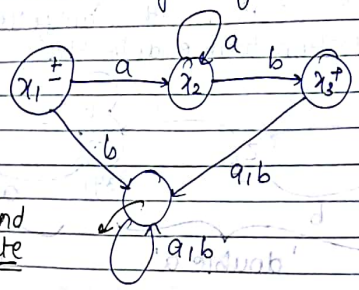
If r doesn't have null, then r^* will still have $\wedge$

NOTE:- The language defined by r^* must always contain null and is denoted by $(\pm)$

CLASSTIME | Page No.
Date / /

eg:- Suppose R.E $r = a^* + aa^*b$ the language defined by r is all strings of only a's and strings of some (not zero) a's ending with a single b.

The automata given for this is ↓



{Rule 4 says closure of alphabets}

Q Pg 142 (Q1 - Q5)

14 Sept 18

Regular

A language that ca
as regular langua

- All finite languages
- Any combination of expression is not

eg:- $\{a^n b^n$ n
$\{\wedge$ ab aabb

Result:- All infinite langu
Some infinite lang

→ If L_1 and L_2 are r
L_1L_2 and L_1^* are

$L_1 \longrightarrow RL$
$L_2 \longrightarrow RL$
then $(L_1 + L_2) \longrightarrow$
$L_1L_2 \longrightarrow$
$L_1^* \longrightarrow$

TG | FA

FA_1

FA_2

14 Sept. 18

Regular Language

Q. Consider

A language that can be defined by R.G is known as regular language.

- All finite languages are regular.
- Any combination of a and b for which regular expression is not possible.

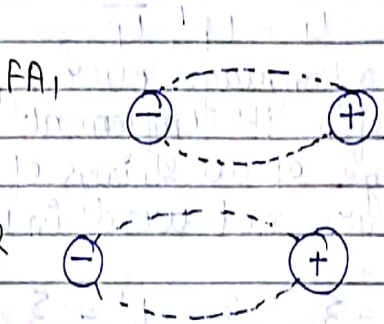
eg :- $\{a^n b^n \quad n = 0, 1, 2, \ldots\}$
$\{\wedge \quad ab \quad aabb \quad aaabbb \ldots\ldots\}$

Result :- All infinite languages are not regular. Only some infinite languages are regular.

→ If L_1 and L_2 are regular languages then $L_1 + L_2$, $L_1 L_2$ and L_1^* are also a regular languages.

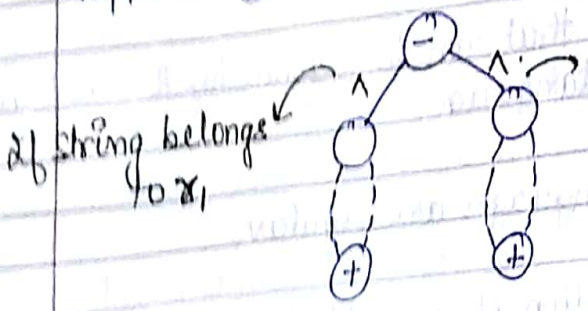
$L_1 \longrightarrow RL \quad r_1$ (Regular expression).
$L_2 \longrightarrow RL \quad r_2$ (Regular expression)
then $(L_1 + L_2) \longrightarrow RL \rightarrow r_1 + r_2$
$L_1 L_2 \longrightarrow RL \rightarrow r_1 r_2$
$L_1^* \longrightarrow RL \rightarrow r_1^*$

TG/LFA

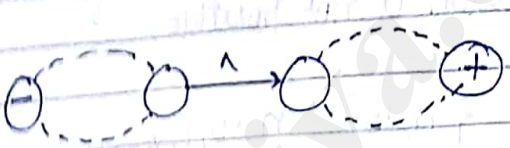


CLASSTIME | Page No.
Date / /

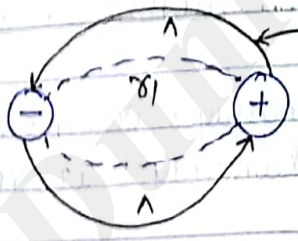
TG | FA ① $r_1 + r_2 \rightarrow$



② $r_1 r_2$



③ $r_1^+ \rightarrow$



to include more than 1 n.

0 or more no. of r_1 is accepted.

Imp:
- R.L are closed under addition ($L_1 + L_2$), concatenation ($L_1 L_2$), L_1^* (closure)
- Complement & intersection

→ Complement of a language.

$L_1 = L_1', \overline{L_1}$

If L is a language over alphabet Σ, we define its complement L' to be the language of all strings of letters from Σ that are not words in L.

$L = \{\ \}$, $L' = \{\ \}$ → not in L.

CLASSTIME | Page No.
Date / /

eg:- $\Sigma = \{a, b\}$

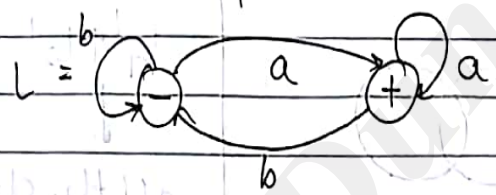
$L = \{a \quad ab \quad abb\}$

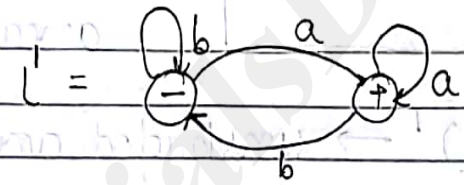
$L' = \{\wedge \quad baa \quad ba \quad bb \quad aaa \ldots\}$

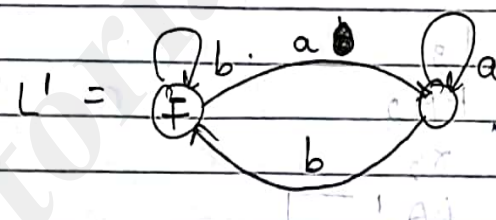
→ $(L')' = L$

Theorem 11 :- If L is a Regular language then L' is also R.L in other words, the set of Regular languages are closed under complimentation.

eg:- Consider a Graph :-







→ If L is a R.L then there is some F.A that accepts language L. Some of the States of this F.A are final states and most likely some are not.

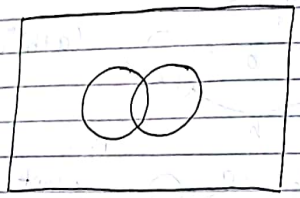
Let's reverse the final status of each state i.e if it was final state make it non final state and it was a non final state then

make it final state.

Theorem 1.2 :- If L_1 and L_2 are R.L then L_1 intersection L_2 is also a R.L. In other words the set of R.L are closed under intersection.

By De morgan's law, for set of any kind, it says.

$L_1 \cap L_2 = (L_1' + L_2')'$



$L_1' + L_2'$ → all the shaded areas.

$(L_1 + L_2)'$ → unshaded area.

L_1	L_2
FA_1	FA_2
x_1	x_2
FA_1'	FA_2'

$FA_1' + FA_2'$

R.E ← F.A = $(FA_1' + FA_2')'$

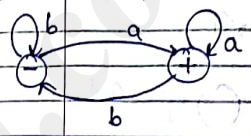
26 Sept 18 TOC

Q3 Consider languag
$L_1 = (a+$
(Pg 185) $L_2 = (a+$

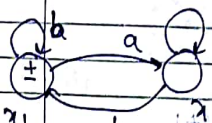
$L_1 \cap L_2 = ($

$= ($

FA_1



FA_1'



	a	b
x_1 ±	x_2	x_1
x_2	x_2	x_1

x_1, y_1 ±
x_1, y_2 +
x_2, y_1 +

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

26 Sept 18 TOC

CLASSTIME | Page No.
Date / /

Q3 Consider languages.
(Pg 185)

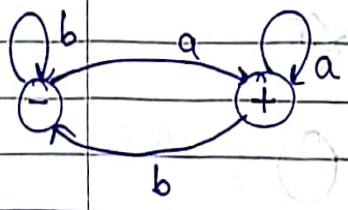
$L_1 = (a+b)^*a$

$L_2 = (a+b)^*b$.

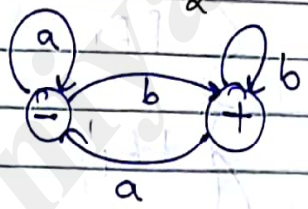
$L_1 \cap L_2 = (L_1' + L_2')'$ {Demorgan's law}.

$= (FA_1' + FA_2')'$

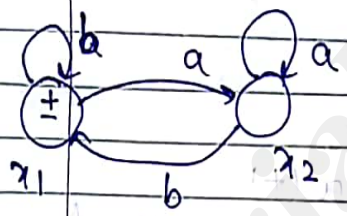
FA_1



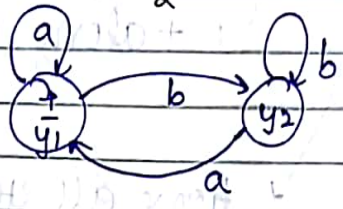
FA_2



FA_1'



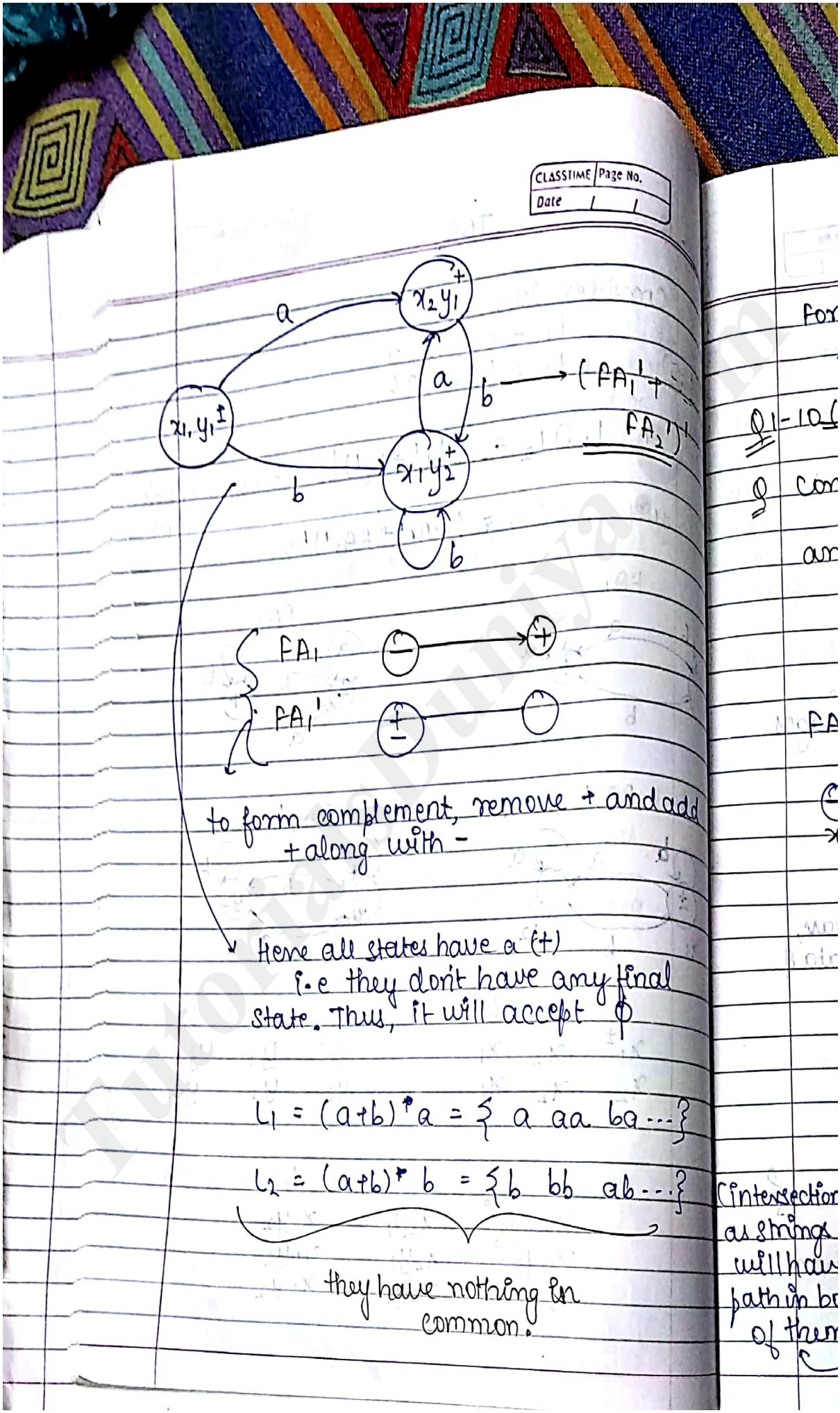
FA_2'



	a	b
x_1 ±	x_2	x_1
x_2	x_2	x_1

	a	b
y_1 ±	y_1	y_2
y_2	y_1	y_2

	a	b
$x_1 y_1$ ±	$x_2 y_1$	$x_1 y_2$
$x_1 y_2$ +	$x_2 y_1$	$x_1 y_2$
$x_2 y_1$ +	$x_2 y_1$	$x_1 y_2$



CLASSTIME | Page No.
Date / /

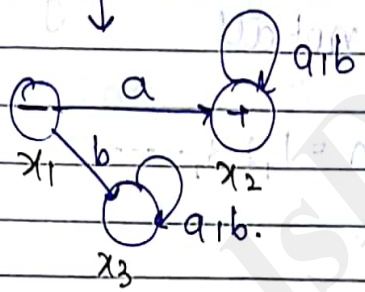
For common strings, both of them will have path for those strings.

Q1-10 (Pg 185)

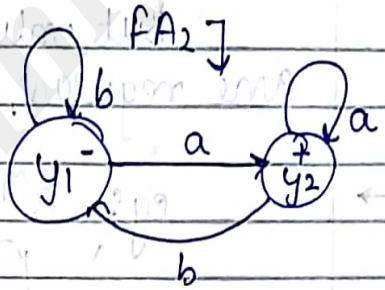
Q Consider language L_1 → all words that begin with a
and L_2 → all words that end with a.

$$L_1 = a(a+b)^*$$
$$L_2 = (a+b)a^*$$

FA1 ↓



FA2 ↓



$x_1 y_1^-$ start from this state.

$x_2 y_2^+$ end at this state

	a	b
(intersection) $x_1 y_1^-$	$x_2 y_2^*$	$x_3 y_1^*$
as strings $x_2 y_2$	$x_2 y_2$	$x_2 y_1$
will have $x_2 y_1$	$x_2 y_2^+$	$x_2 y_1$
path in both of them		

$x_2 y_2$ (+)

we will make graph & then define language.

CLASSTIME | Page No.
Date / /

x_1 - a b
x_2 + x_2 x_3
x_3 x_3 x_3

y_1 - a b
y_2 + y_2 y_1
y_2 y_1 y_1

NOTE:-

U
RL IL/NRL

All finite languages are regular.
But only some infinite languages are regular, and not all

→ eg:- $a^n b^n$ $n = 1, 2, 3, \ldots\ldots$

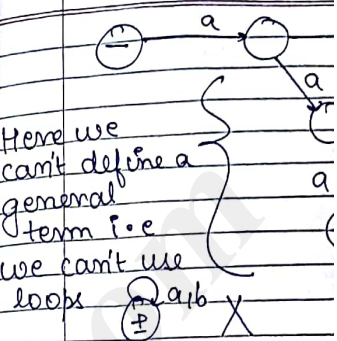
If this language is not regular, then we can't define RE, automata & graph for them.

(Kleen's Theorem)

$a^n b^n = \{ab, aabb, aaabbb \ldots\ldots\}$

Since we don't have null here thus we can't use a state as

(±) X



Here we can't define a general term i.e we can't use loops (±) a/b X

as loop means a*
b's m

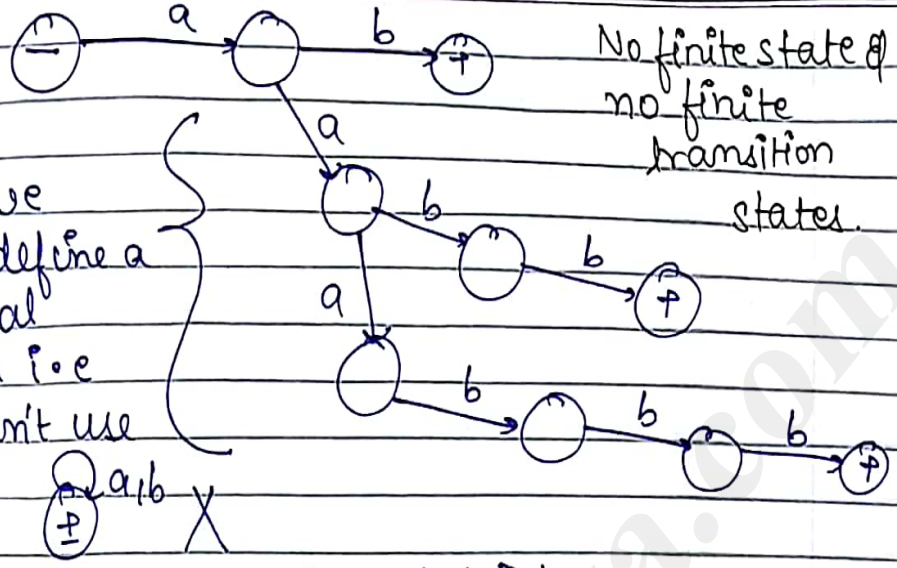
For an FA

∴ All Infi

$a^n b^n$
$a^n b^{n+1}$
n

* Counting Lemm

CLASSTIME | Page No.
Date / /



No finite state & no finite transition states.

Here we can't define a general term i.e we can't use loops a,b

as loop means a^* and b^* but no. of a's & b's must be same.

For an FA → Q (finite)
Σ ab
δ (finite)

∴ All infinite languages are not regular.

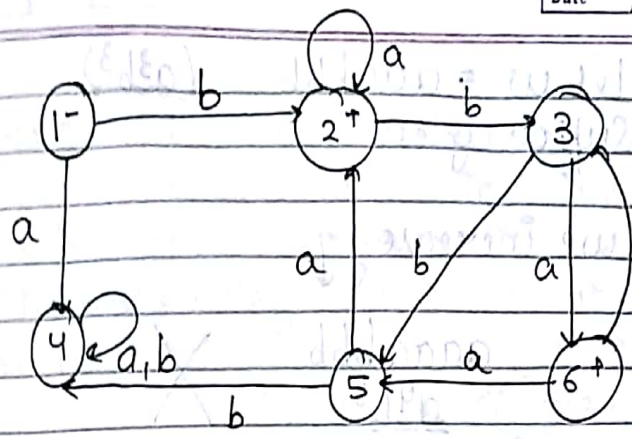
$a^n b^n$
$a^n b^{n+1}$ } for n = 1, 2, 3, ...

→ not regular.

* Counting Lemma / Theorems

CLASSTIME | Page No.
Date / /

Q.



Consider a graph $xy^nz \in L$

$W = \underbrace{bbb}_{x}\underbrace{aba}_{y}\underbrace{baba}_{z}$

$w \in L$

$|w| > N$

$xyz \in L$

Now (Pumping y)

$\underbrace{bbb}_{x}\underbrace{aba}_{y}\underbrace{aba}_{y}\underbrace{baba}_{z}$

$\therefore xyyz \in L$

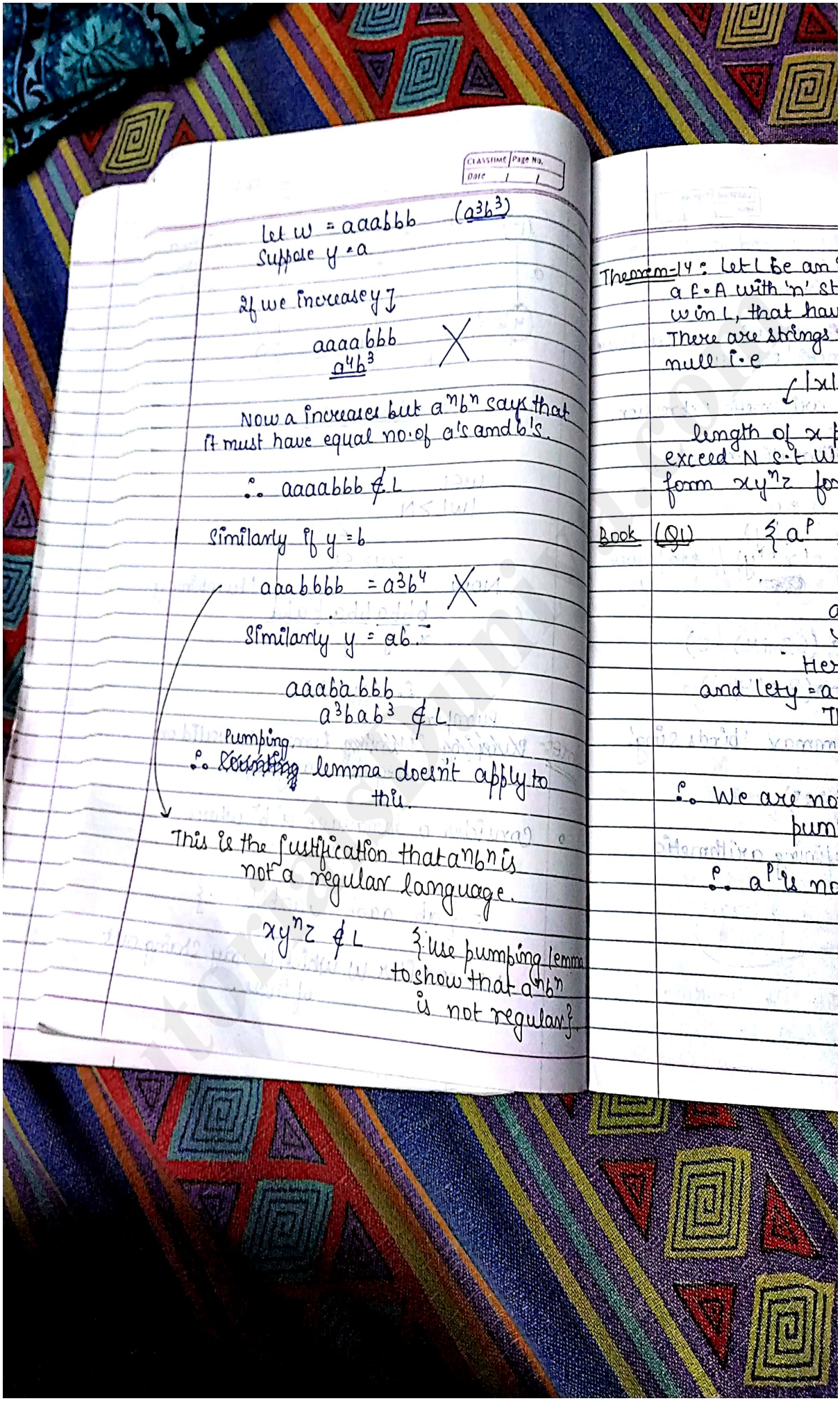
Pumping.

NOTE:- Pumping Lemma is valid on regular languages.

- Consider a language $a^n b^n$ where $n = 1, 2, 3 \ldots\ldots$

$= \{ab \quad aabb \quad aaabbb \ldots\}$

We consider w which is any string out of these.



Let $w = aaabbb$ (a^3b^3)

Suppose $y = a$

If we increase y]

$aaaabbb$

a^4b^3 ✗

Now a increases but a^nb^n says that it must have equal no. of a's and b's.

$\therefore aaaabbb \notin L$

Similarly if $y = b$

$abaabbbb = a^3b^4$ ✗

Similarly $y = ab$

$aaababbb$

$a^3bab^3 \notin L$

Pumping lemma doesn't apply to this.

This is the justification that a^nb^n is not a regular language.

$xy^nz \notin L$ } Use pumping lemma to show that a^nb^n is not regular }

Theorem-14 : Let L be an
a f.A with 'n' st
w in L, that hav
There are strings
null i.e
$\{|x|$
length of x
exceed N s.t w
form xy^nz fo

Book (Q1) $\{a^p$

Her
and let $y = a$

$\therefore$ We are no
pum

$\therefore a^p$ is no

CLASSTIME | Page No.
Date / /

Theorem-14 : Let L be an infinite language accepted by a f.A with 'n' states. Then for all words w in L, that have more than n letters There are strings x, y and z where y is not null i.e

$|x| + |y| \leq N$ → no. of states.

length of x plus length of y doesn't exceed N s.t $w = xyz$ and all strings of the form $xy^n z$ for $n = 1, 2, 3, \ldots$ are in L.

Book (Q1) $\{a^p, p \text{ is prime}\}$.

a^3, a^5, a^7

Here a is getting repeated and let $y = a$

Then $a^5 . a = a^6$

not a word in language

∴ We are not able to a 'y' that can be pumped.

∴ a^p is not a Regular language.

CFG

Context free Grammar.

→ Consider an arithmetic expression. We can write diff. rules for arithmetic expression.

Rule 1:- Any no. is in set of arithmetic expression.

Rule 2:- If x and y are in arithmetic expression, then so are

(x)	x * y
(x)	(x / y)
(x+y)	(x ** y) // x to power y
(x-y)	

eg:- $\frac{3+4}{5}$ → ((3+4)/5)

3+4/5 → (3+(4/5))

→ Consider the word grammar 'birds sing'.

- Rules of grammar (Pg 227)

→ Consider a model for defining arithmetic expression.

Start → (AE)

↓

arithmetic expression.

CLASSTIME | Page No.
Date / /

- AE can be → AE = (AE+AE)
  AE = (AE-AE)
  AE → any no.

eg:- ((3+4)/5)

start → (AE)
→ (AE/AE)
→ ((AE+AE)/AE)
→ ((3+4)/5)

terminal + non term. (RHS)

* Rule 1: Any no → FD
* Rule 2: FD → FD.OD
* Rule 3: FD → 1,2,3,....9
* Rule 4: OD → 0,1,2,....9

Production rules to generate a no.

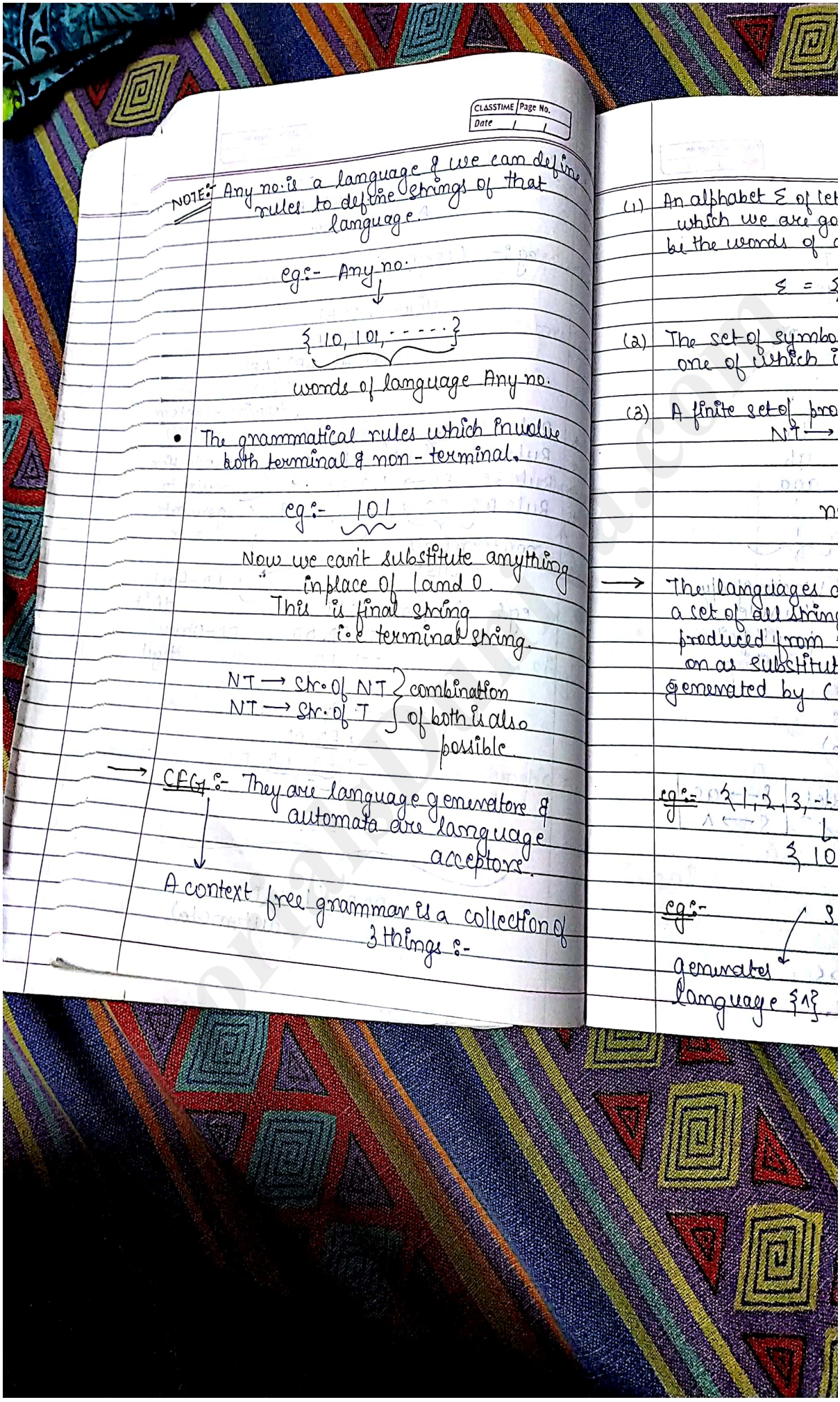
non terminals (LHS)

FD - first digit
OD - Other digit

eg:- Any no. → FD
→ FD.OD
→ 10

Any no. → FD
→ FD.OD
→ FD.OD.OD
→ 101

PDA (Push Down automata)



NOTE:- Any no. is a language & we can define rules to define strings of that language.

eg:- Any no.

↓

{10, 101, ----}

words of language Any no.

- The grammatical rules which involve both terminal & non-terminal.

eg:- 101

Now we can't substitute anything inplace of 1 and 0.
This is final string
i.e terminal string.

NT → Str. of NT } combination
NT → Str. of T } of both is also possible

→ CFG:- They are language generators & automata are language acceptors.

↓

A context free grammar is a collection of 3 things :-

(1) An alphabet Σ of let
which we are go
be the words of

Σ = {

(2) The set of symbo
one of which

(3) A finite set of pro
NT →

n

→ The languages
a set of all string
produced from
on as substitut
generated by

eg:- {1, 2, 3, -
{10

eg:- S

generates
language {1}

(1) An alphabet Σ of letters called terminals from which we are going to make strings that will be the words of a language.

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

(2) The set of symbols called non-terminals, one of which is start symbol S (start)

(3) A finite set of productions of the form
NT $\longrightarrow$ finite strings of terminals and/or non terminals.

no restriction of rules on RHS
∴ Context free.

$\longrightarrow$ The languages defined/generated by CFG is a set of all strings of terminals that can be produced from start symbol S. using production as substitution & the language generated by CFG is called 'CFL'

(Context Free Language)

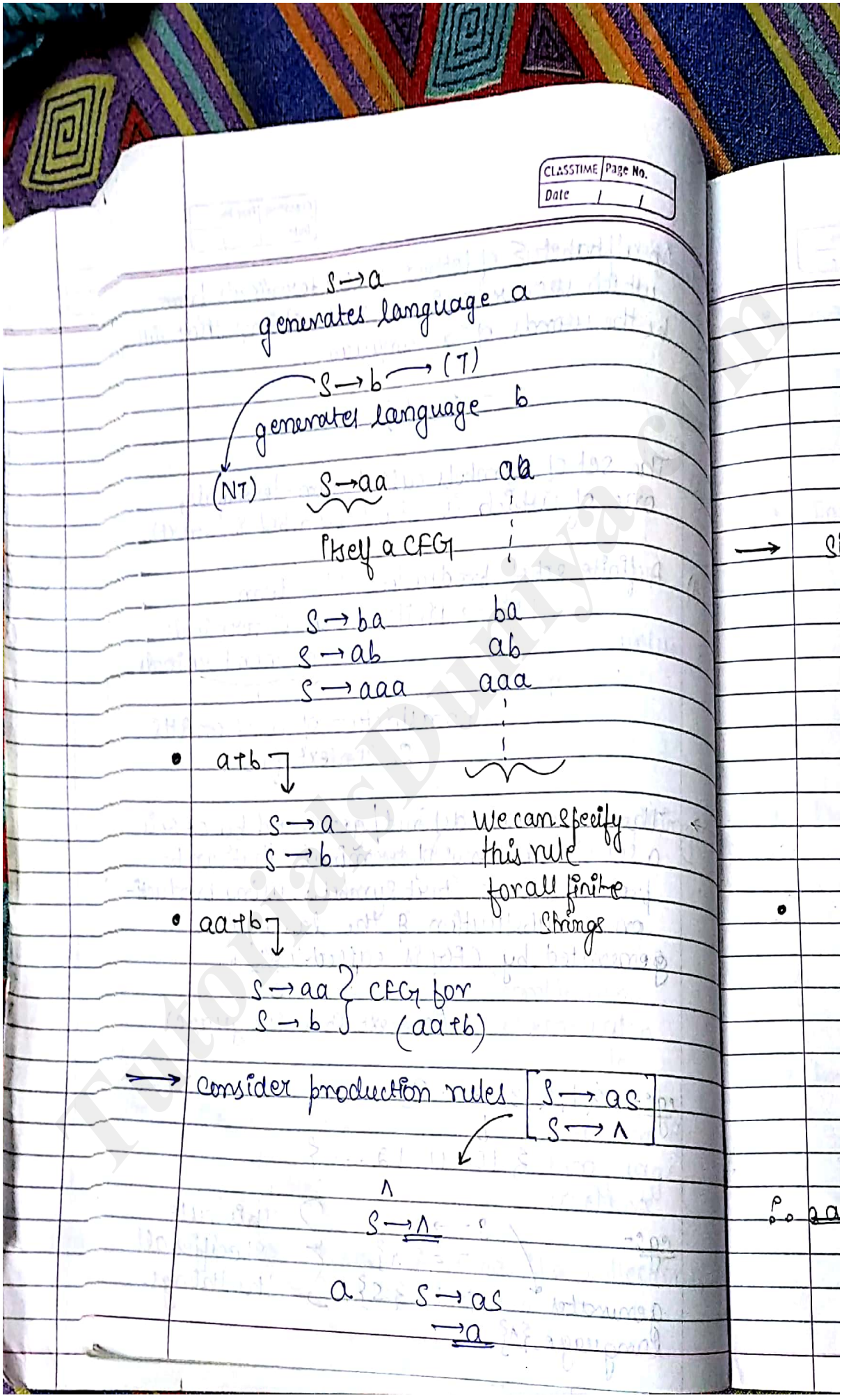
eg:- $\{1, 2, 3, \ldots 9\}$

$\{10\ 11\ 12 \ldots\}$

eg:- $S \longrightarrow 1$
$T = \{1\}$
$NT = \{S\}$

This rule specifies all the things.

generates language $\{1\}$



$S \to a$

generates language a

$S \to b \longrightarrow (T)$

generates language b

(NT) $S \to aa$ ab

Itself a CFG ⋮

$S \to ba$ ba

$S \to ab$ ab

$S \to aaa$ aaa

⋮

• a+b

$S \to a$

$S \to b$

We can specify this rule for all finite strings

• aa+b

$S \to aa$ } CFG for

$S \to b$ } (aa+b)

⟶ Consider production rules $\begin{bmatrix} S \to aS \\ S \to \wedge \end{bmatrix}$

$\wedge$

$S \to \wedge$

a $S \to aS$

$\to a$

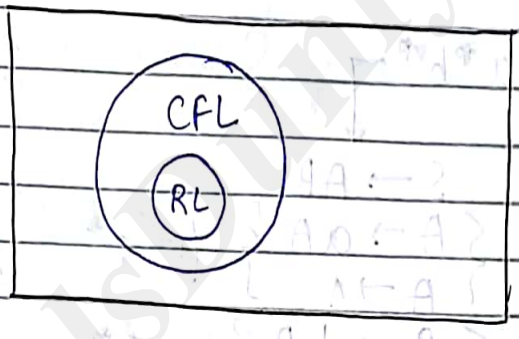
CLASSTIME | Page No.
Date / /

aa $S \to aS$
$\to a.aS$ } for $S \to aS$ and $S \to \wedge$ }.
$\to aa$

$\therefore$ $S \to aS$
and $S \to \wedge$ represents a^*.

→ Similarly, b^* ↓

$S \to bS$
$S \to \wedge$



• for aa^* ↓

$S \to aA$ → compulsory a
$A \to aA$ } represents a^*
$A \to \wedge$ }

a $S \to aA$
$\to a\wedge = a$

aa $S \to aA$
$\to aaA \to aa\wedge$
$\to aa$

$\therefore aa^*$

aaa $S \to aA$
$\to aaA \to aaaA \to aaa$

CLASSTIME | Page No.
Date / /

- For ab^* →

$S \to aA$
$A \to bA$
$A \to \wedge$

- For ba^* →

$S \to bA$
$A \to aA$
$A \to \wedge$

- For a^*b^* →

$S \to AB$
$\{A \to aA, A \to \wedge\}$ for a^*
$\{B \to bB, B \to \wedge\}$ for b^*

- For $(a+b)^*$ →

$\wedge$
a
b
aa
ab
⋮

$S \to aS$
$S \to bS$
$S \to \wedge$

$S \to aS$
$\to abS \to ab$

- For $(aa+bb)^*$ ↓

$S \rightarrow aaS$
$S \rightarrow bbS$
$S \rightarrow \wedge$

- For $(a^*b^*)^*$ ↓

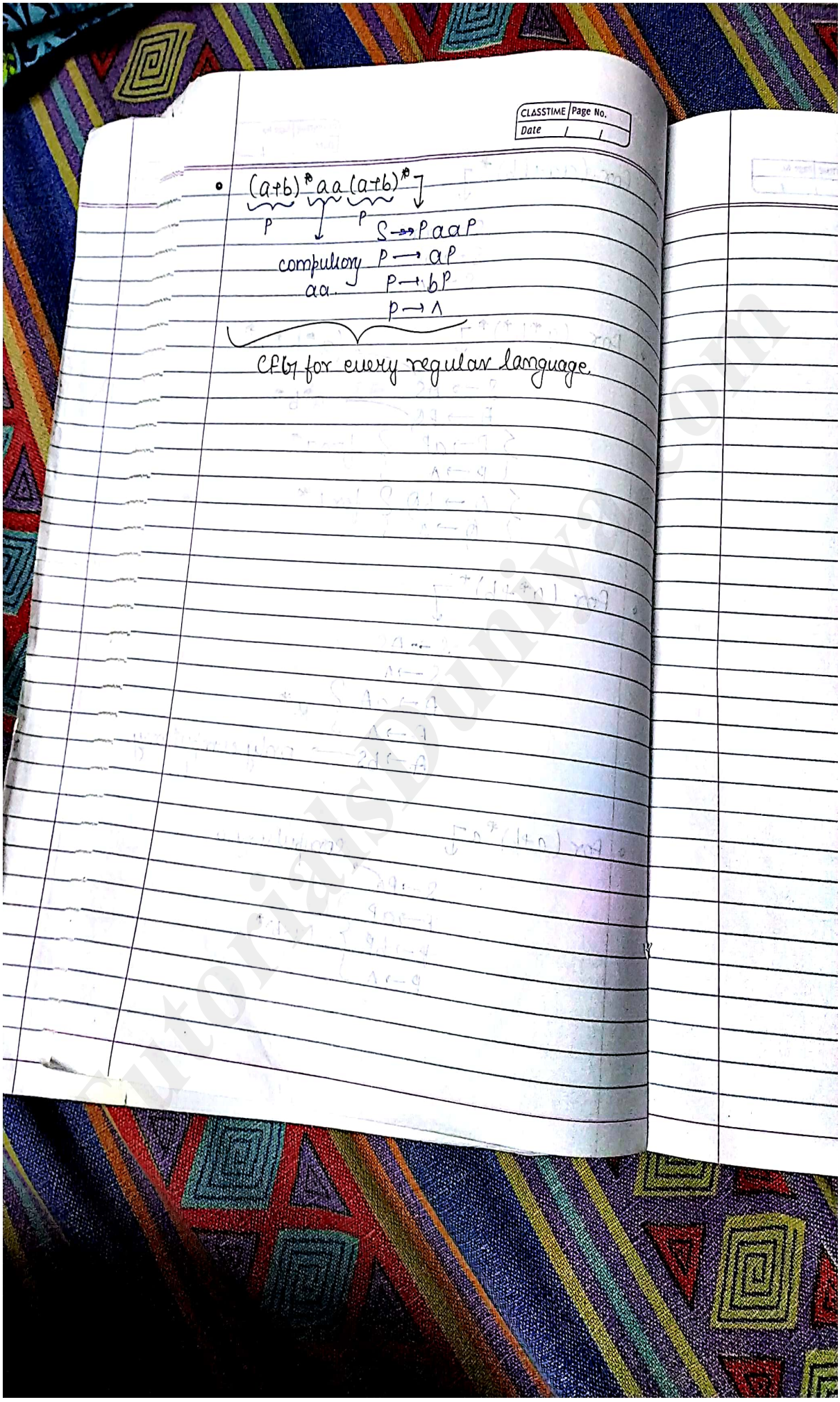
$S \rightarrow AS$ → $(a^*b^*)^*$
$A \rightarrow PQ$ → a^*b^*
$P \rightarrow aP$, $P \rightarrow \wedge$ } for a^*
$Q \rightarrow bQ$, $Q \rightarrow a$ } for b^*

- For $(a^*+b)^*$ ↓

$S \rightarrow AS$
$S \rightarrow \wedge$
$A \rightarrow aA$, $A \rightarrow \wedge$ } a^*
$A \rightarrow bS$ → only compulsory b.

- For $(a+b)^*a$ ↓

$S \rightarrow Pa$ → compulsory a
$P \rightarrow aP$, $P \rightarrow bP$, $P \rightarrow \wedge$ } $(a+b)^*$



TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well