Theory of Computation Notes

Contributor: Manoj Dhangar
[KMV (DU)]

Computer Science Notes

Ch-2 - Languages —

Λ → string with length 0

$\emptyset$ → A lang. with no. words

$L_F = \{0\}$ - finite Languages

$L_I = \{a, aa, aaa, aaaa, \ldots\}$ infinite lang.

Laxicographic order - (increasing order of length) —

Theory of formal languages —

The word formal refers to the facts that all the rules for lang. are explicitely stated in terms of word string of symbol can occur. The down formal rule use here emphasize. It is form of symbols that we intrested in not the meaning.

certain specify set of strings of chars. from the alphabates

Ex - I eat one apple

I eat three books →

In a formal languages. we must allow this string because it is grammtical correct.

Meaning is something that we not refer to in formed languages.

for exp. pallindrome $\Sigma \{a,b\}$ over alphabate

$\wedge$ - NULL

a	aaa
b	aba
aa	bab
bb	bbb

Kleene closure or kleene star —

Given an alphabate Σ a lang. is define in which any strings of letters from Σ is a valid even the same lang. is closure of the alphabate.

$\Sigma^* = \{ \wedge\ a\ b\ aa\ ab\ ba\ bb \ldots \}$

$\Sigma = \{a\}$

$\Sigma^* = \{ \wedge\ a\ aa\ aaa\ aaaa \ldots \}$

25/July

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

$\Sigma^* = \{\wedge\ a\ b\ ab\ ba\ aa\ bb \ldots\}$

$\Sigma = \emptyset$

$\Sigma^* = \{\wedge\}$

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

$\Sigma^* = \{a\ b\ aa\ ab\ ba\ bb \ldots\}$

↓

It also a closure the diff. is null $\wedge$.

$S = \{a, ab\}$

$S^* = \{\wedge\ a\ aa\ ab \ldots\}$

Consider a set $S = \{a\ b\ ab\}$

$S^* = \{\wedge\ aa\ ab\ ab\ aa\ bb\ ba\ aab\ bab\ abab\}$

$S^* = \{\wedge\ a\ b\ aa\ ab\ ba\ bb\ aaa\ aab\ aba\ abb\ baa\ bab\ bba\ bbb \ldots\}$

Same you can write

$S = \{0, 1, 01\}$

$S^* = \{\quad\}$

Theorem 1:- We have for any set S of string we have

$S^* = S^{**}$

$\Sigma^* = \{ \wedge\ a\ b\ aa\ ab\ ba\ bb \ldots \}$

$\Sigma^{**} = \{ \wedge\ a\ b\ aa\ ab\ ba\ bb \ldots \}$

$\Sigma^{**} \subset \Sigma^*$

$\Sigma^* \subset \Sigma^{**}$

problem 1 to 6, 9, 11 (do by self)

26 July

Ch-4 Regular Expression

Regular Exp.

$L_1 = \{ x^n \mid n = 0, 1, 2, 3 \ldots \}$ $\wedge$

$1, 3, 5, 7 \ldots$ a

b

$\{ \wedge\ x\ xx\ xxx \ldots \}$ a^*

b^*

$\Sigma = \{a\}$ aa^*

$\Sigma^* = \{ \wedge\ a\ aa\ aaa \ldots \}$ ab^*

ba^*

$aa^* = \{ a\ aa\ aaa \ldots \}$ a^*b

Some [it is compulsory in every part $(a+b)^*$

$a^*a = \{ \ldots \}$

$ab^* = \{ a\ ab\ abb\ abbb \ldots \}$

$b^*a = \{ a\ ba\ bba\ bbba \ldots \}$

$a^*b^* = \{ \wedge\ a\ b\ aa\ ab\ ba\ bb \ldots \}$

$b^*a^* = \{ \wedge\ a\ b\ aa\ bba\ bbb \ldots \}$

a+b → either a or b not null

(a+b)(a+b) → aa ab ba bb same
→ aa+ab+ba+bb

(a+b)(a+b)(a+b) = aaa aab abb baa bab ... all combination of 3

(a+b)* = { ∧ a b aa ab ba bb ... }

a(a+b)* = { a aa ab ... }

(a+b)*a = { a aa ba ... }

a*b*] not same = { ∧ a b ab bb }
(ab)*] = { ∧ ab abab ababab ... }
↳ complete string

ab*a = { a aa aba abba abbba ... }
a(aa)* = { a aaa aaaaa ... }
(aa)*a = { ... }

a(a+b)*a =
a(a+b)* + (a+b)*a =

30 July

Σ = { a b c }
T = { a c ab cb abb cbb abbb cbbb ... }

All the words in T begin with a & c and then are followed by some no. of b.

Same as written
$(a+c)b^*$

Consider a language

Some or Expression

$L = \{aaa\ aab\ aba\ abb\ baa\ bab\ bba\ bbb\}$

$(a+b)(a+b)(a+b) \neq (a+b)^3$

All words that begin a and end with b

$a(a+b)^*b$	aa^*b^*b
ab	ab
aab	aab
abb	abb
aaab	aaab
aabb	aabb
abab	~~abab~~
abbb	abbb

All these expression are language defining expressions and these are Regular expression.

The corresponding languages that define are

Λ
a
b
+
(
)
*

The symbols are appear in regular exp. are the letters of alphabets signs. The symbol of Λ string (*) operator and the + sign.

Set of reg. expression is defined by the foll. rules.

Rule-1 - Every letter of sigma Σ can be made into a reg. expression.
NULL itself is a reg. expression.

Rule-2 - If r_1 and r_2 are reg. expression then r_1 and r_2

(i) (r_1)
(ii) $r_1 r_2$
(iii) $r_1 + r_2$
(iv) r_1^*

are reg. expre.

Rule-3 - Nothing else is a reg. expression.

An expression $(a+b)^* a (a+b)^*$ all strings with a and all without a

$(a+b)^*$

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

The Language of all words that have atleast 2a

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

All the words with exactly two a's

$b^* a a b^* \quad b^* a b^* a b^*$

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

If S and T are set of strings of letter whether they are finite or infinite then define the product ST have all combination of string from S concatenated with a string from P in that order

For exp—

Set S as {a, aa, aaa}

T = {bb, bbb}

then ST is

= {abb, abbb, aabb, aabbb, aaabb, aaabbb}

Languages associated with Reg. expression

31/July

Lang. is associated with regular expre.

Following rules define the language associated with any regular exprs.

Rule-(1) The language associated with reg. expression that is just a single letter is that one letter itself along and the language associated with NULL symbol ∧ is just NULL a ward language $\wedge = \{\wedge\}$

Rule-(2)- If R_1 is a reg. exp. associated with the language L_1 and R_2 associated with the Language L_2. then

(i) the Reg. expression $(R_1)(R_2)$ is associated with the product language L_1, L_2. i.e. the language L_1 times L_2

(ii) the Reg. exp. $(R_1 + R_2)$ is associated with the language form by the union of the sets L_1, L_2

(iii) the language asso. with reg. expression $(R_1)^*$ give L_1^* klean closere of the set L_1^* as a set of words.

There collection of rules proves recursively there is some language associated with reg. expression.

If L is a finite Language then L can be define by a reg. expression in other words are all finite lang. are regular.

$L_1 = \{a\ ab\} = a + ab$

$L_2 = \{ab\ ba\ bb\} = ab + ba + bb$

Set of strings of A and B's that at a some point contain a double letter

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

$aa(a+b)^*$

$(a+b)^* aa$

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

$(a+b)^* bb (a+b)^*$

$(a+b)^* (aa+bb)(a+b)^*$

Ex- $(A+B^*)^*$ $(a+b)^*$

both are same or diff.

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

Same

The language of all words without a 'aa'. Your Reg. must give ∧

$b^*(abb^*)^*(\wedge + a)$

Q Aug –

* Even - Even (Language)

$[aa + bb + (ab + ba)(aa + bb)^*(ab + ba)]^*$

$(aa)^*$
$(bb)^*$
$(ab + ba)(aa + bb)^*(ab + ba)$

~~baaa~~	aaaa
baab	~~aaab~~
baba	~~aaba~~
~~babb~~	aabb
bbaa	~~abaa~~
~~bbab~~	abab
~~bbba~~	abba
bbbb	abbb

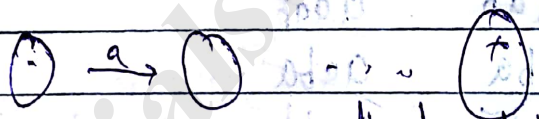
Ch-4 - Q- 1 to 11

6 Aug **Finite Automata (Automaton) [FA]**

PC's are setup on a playing board. Dice are thrown and a no. is generated at random depending on the no. the pc's on the board must be rearranged in a fashion completely specified by the rules. Everything is determine by the dyes. We know that PC's are finite.

State | Initial State

Finite automata is also called regu. acceptor



initial state — final state / Accepting state

if we reach initial to final then input is accepted else rejected

Finite Automata — because no. of possible states and no. of letters are both finite. Automaton is change of states is totally govern by the input.

Finite Automata – A finite automata is a collection of 3 things –

(1) A finite set of states, one of which is designated as initial state called the start state, and some (may be none) of which are designated as final states.

(2) An alphabet Σ of possible input letters

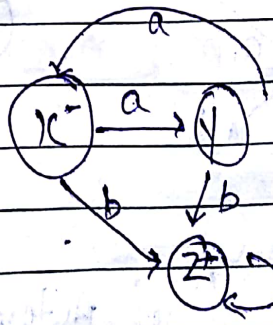
(3) A finite set of transitions that tell for each state and for each letter of the input alphabet which state to go to next.

Ex- Consider

set of state — set of alphabate

	a	b
→x	y	z
y	x	z
z+	z	z

(Transition table)



a a
a b b
a a a b

accepted

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

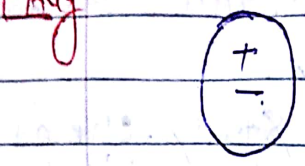
TutorialsDuniya.com

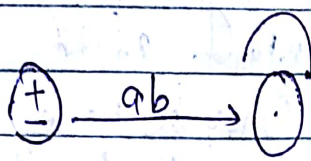
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- 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

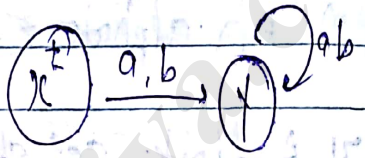
Please Share these Notes with your Friends as well

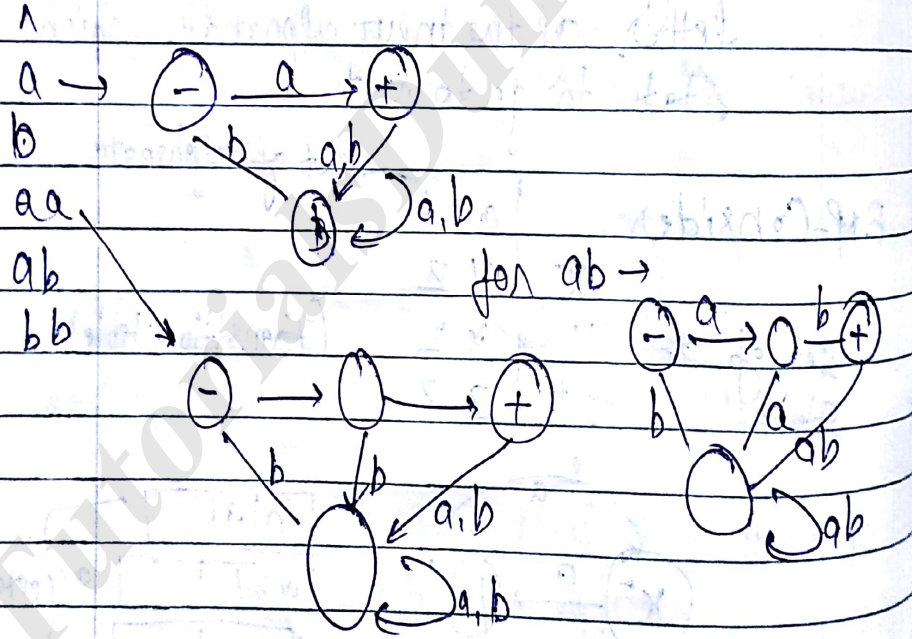
7 Aug


→ initial and final state is same mean null is accepted

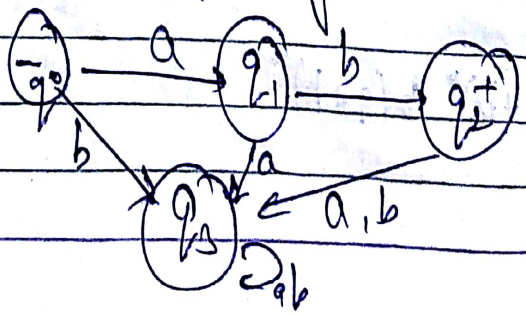

for this fig. only null is accepted

	a	b
$x^{\pm}$	y	y
y	y	y



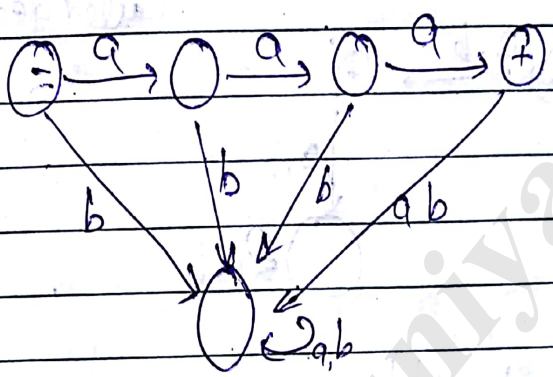


DFA (Deterministic finite Automata) →

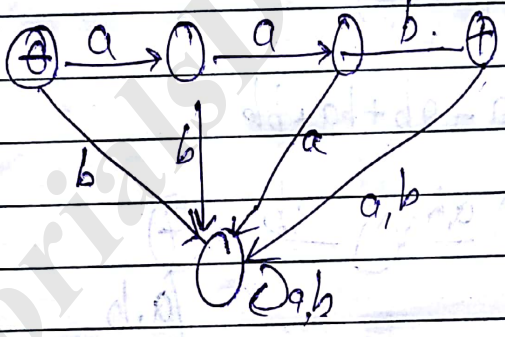


	a	b
q_0	q_1	q_3
q_1	q_3	q_2
q_2	q_3	q_3
q_3	q_3	q_3

aaa —



aab
aba
abb



aba →

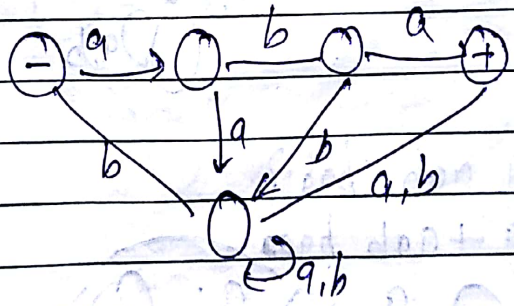
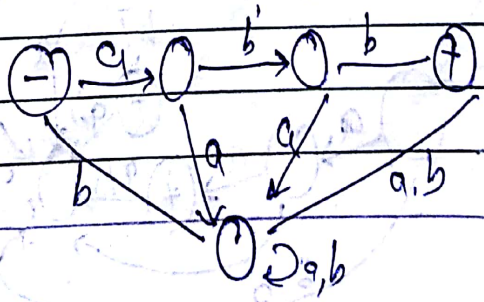
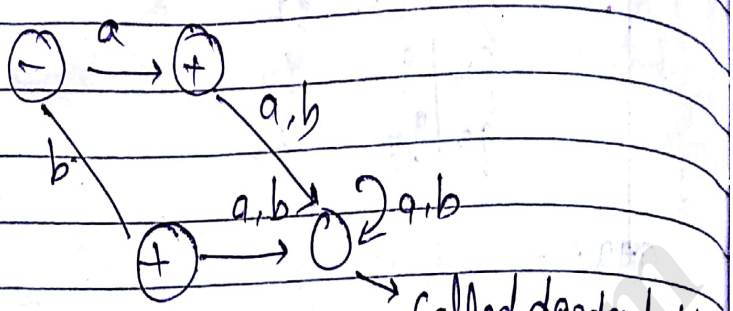


abb →

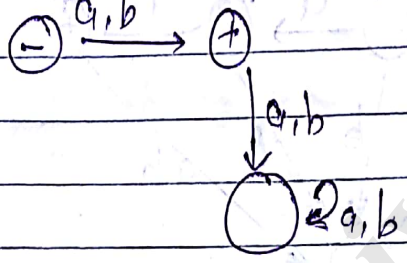


(6)

for a+b

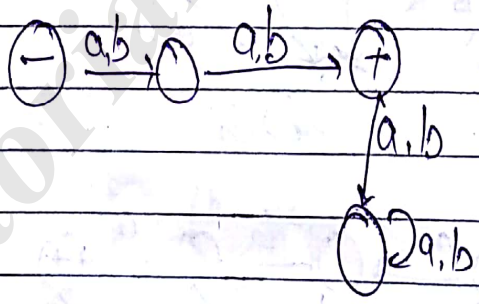


OR



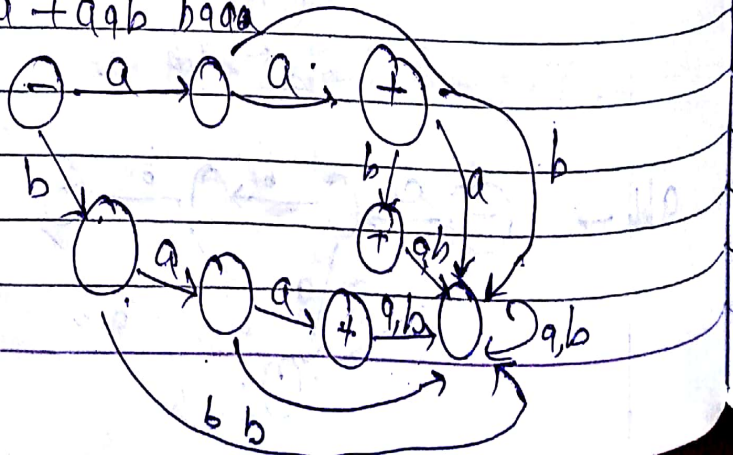
Try these [aa+ab+ba+bb
(a+b)(a+b)

& Ayg aa+ab+ba+bb



{aa. aab baa}

aa+aab baa

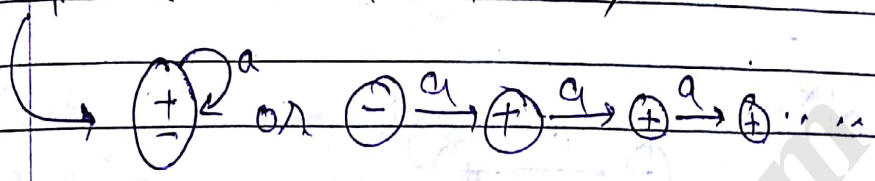


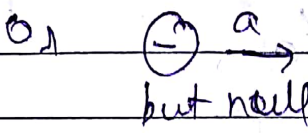
$\wedge$

a

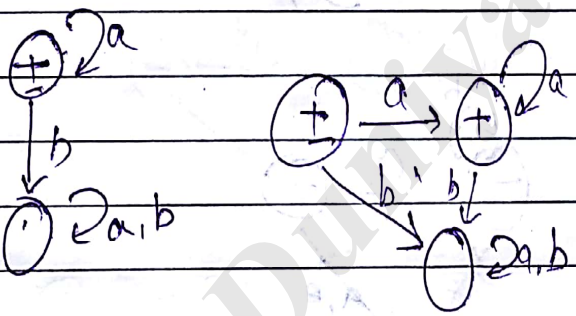
b

$\{\wedge, a, aa, aaa, aaaa, \ldots\}$

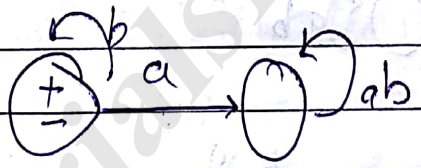




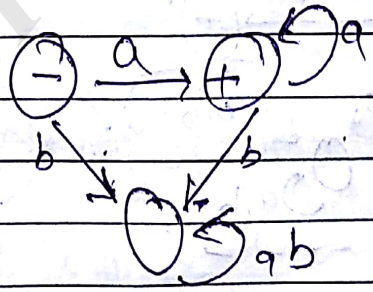
a^* →



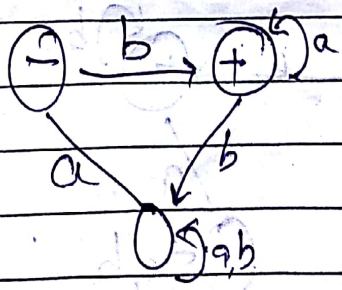
b^* →



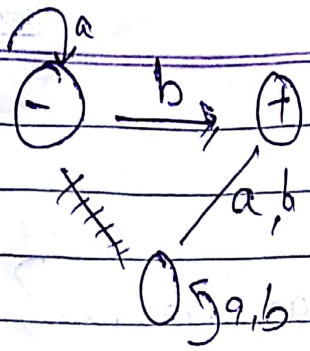
aa^* →
a^*a



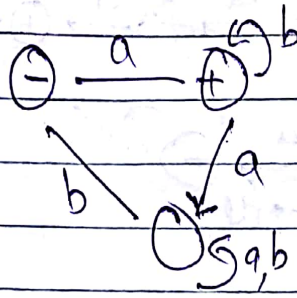
ba^* →



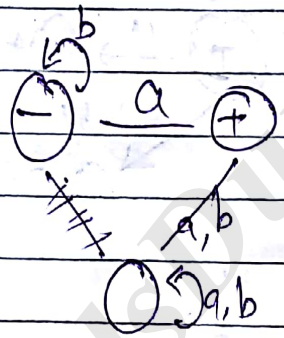
$a^*b \rightarrow$



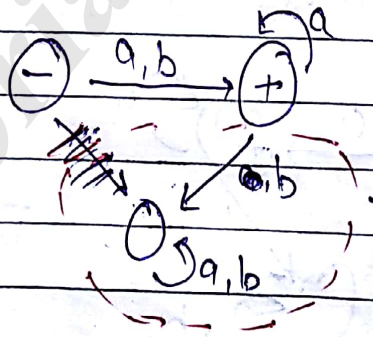
ab^*



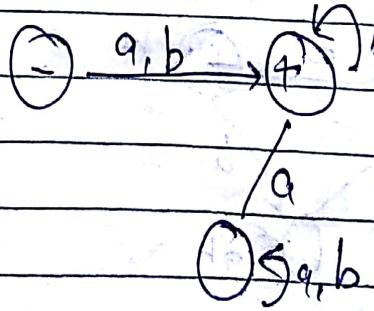
b^*a



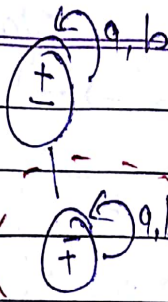
$(a+b)a^*$



$(a+b)b^*$

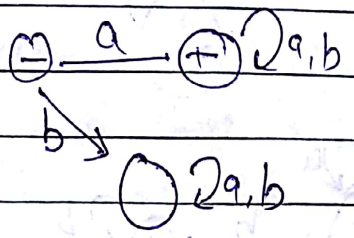


$a^* b^*$ →

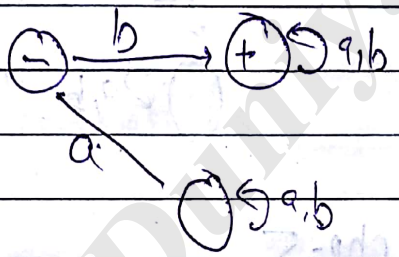


$(a+b)^*$

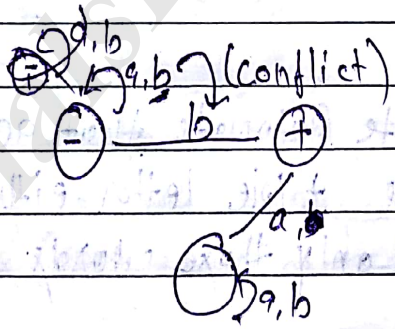
$a(a+b)^*$ →



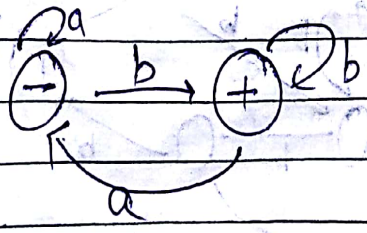
$b(a+b)^*$
$(a+b)^* b$
$(a+b)^* a$



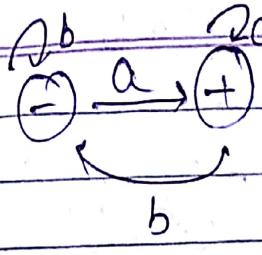
$(a+b)^* b$ →



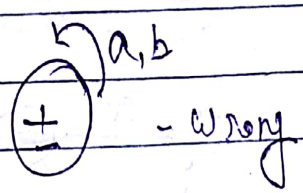
It is not deterministic in nature means path is not define beco2 b is there.



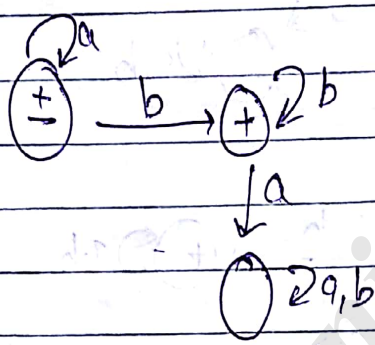
$(a+b)^*a \rightarrow$



$a+b^* \rightarrow$



- Wrong

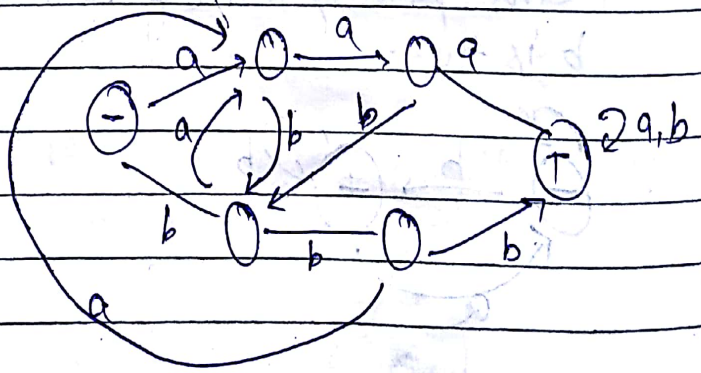


Q-1 to 8 chp-5

9/Aug –

Finite Automata that accept all strings that contains either triple letter either aaa, bbb and only those strings

$(a+b)^*(aaa+bbb)(a+b)^*$



Definition of transition Graph — (TG) — Invented by John Myhill

A transition Graph (TG) is a collection of three things —

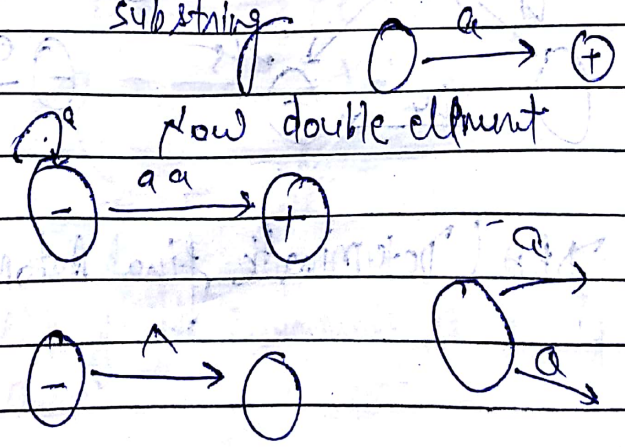
(1) A finite set of states at least one of which is designated as the start state and some.

(2) An alphabet (Σ)

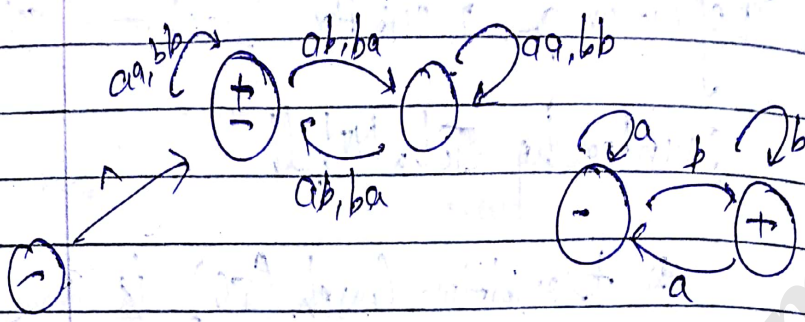
(3) A finite set of transition that show how to go from some states to some others based on reading specified sub strings of input letters, (possibly even the null string)

$\Lambda \in \Sigma$

substring



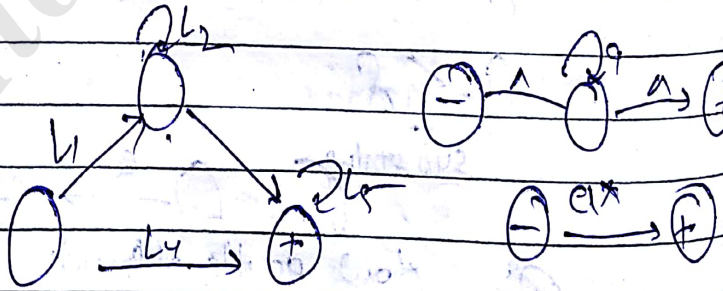
A TG like this



GTG (Generalize Transition Graph) —
is a collection of three things-

(1) A finite set of state in which atleast one a star state and some (may be none are final state)
(2) An Alphabet Σ of in input letters
(3) Directed edges connecting some pair of states. each label with a regular expression.

For ex — It will allow a*



DFA (Deterministic final Automata)

Chapter-7 (Kleen's theorem)

Any lang. can be defined by reg. expression, finite automata or TG can be define by all three methods.

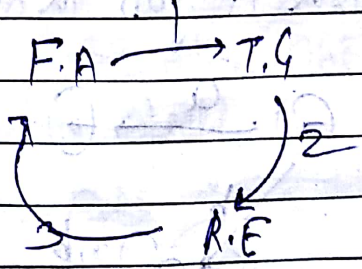
RE
FA
TG

21/8/18 three parts—

part-1— Every lang. can be define by finite automata and also by TG.

part-2— Every lang. that can be define by TG can also be define by regular expression.

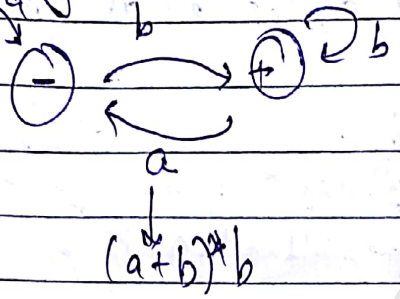
part-3— Every lang. that can be define by regular expression that can by finite automata.



every finite automata is itself already a TG.

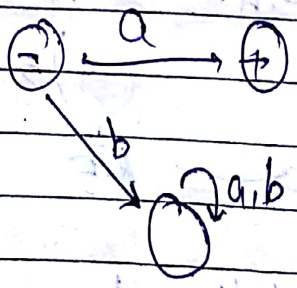
Any lang. that can be define F.A that already been defined by TG.

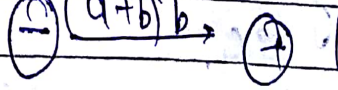
Turning TG into Regular expression



This part will be or proof done by converting - constructive algo. which means we represent a procedure that starts out a TG and end sub with a reg. expression that define the same language.

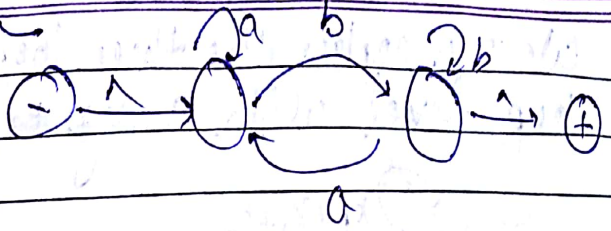
Any algo. must satisfy a criteria (i) it must work every constable TG and it must guarantee to finish it job in a finite time



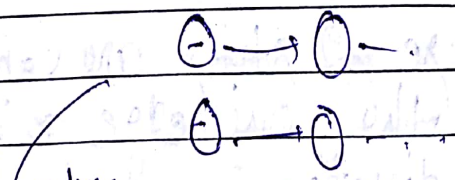


Reduce graphs above

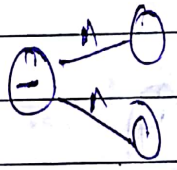
Page No. 106 (given on algo.)



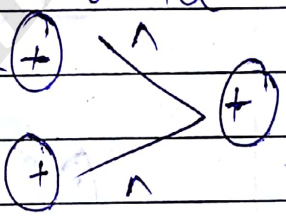
We can double - or +



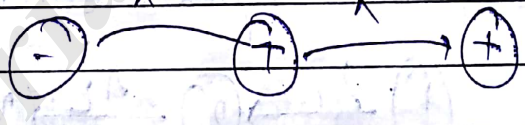
reduce



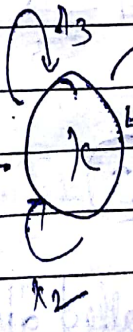
reduce



29/8/18



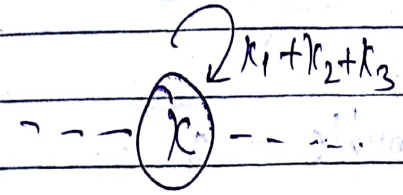
Suppose T has some state that has more than one loop circling have to itself.



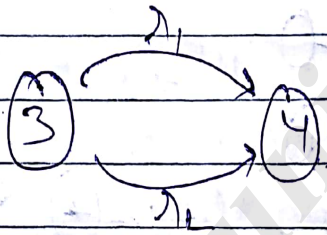
CLASSTIME | Page No.
Date

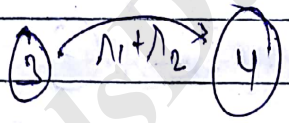
(25)

We can replace the three loop by 1 loop level with a reg. expression.

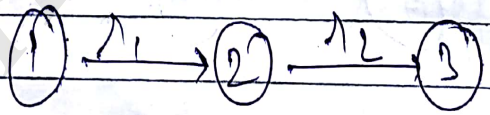


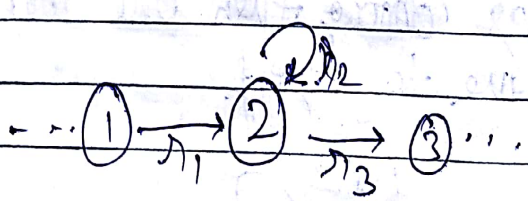
Suppose 2 states are connected by more than one edge going in the same direction.





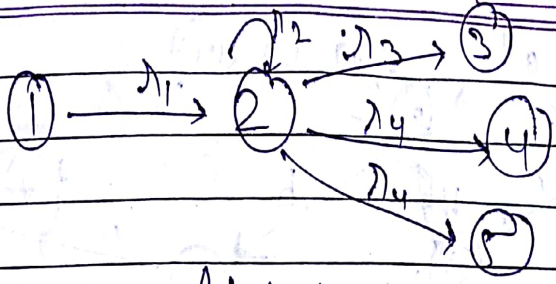
Bypass and state elimination exp —



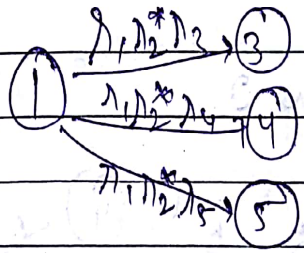


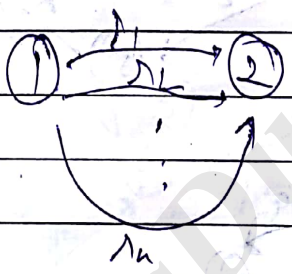
edges goes in parallel direction we can add them if final state is same as well initial

We can eliminate anything as we want but the final result will same



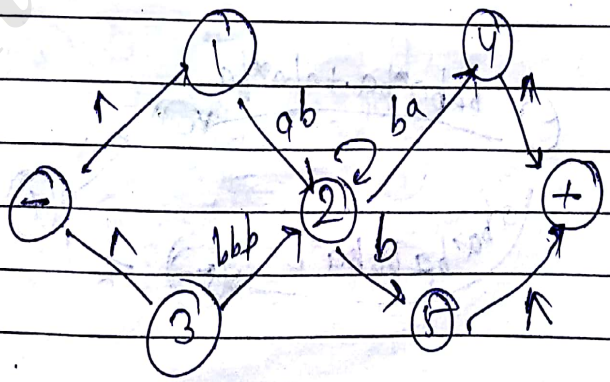
Here we can eliminate λ_2





1 $\xrightarrow{\lambda_1+\lambda_2+\cdots+\lambda_n}$ 2

Consider a T-Graph



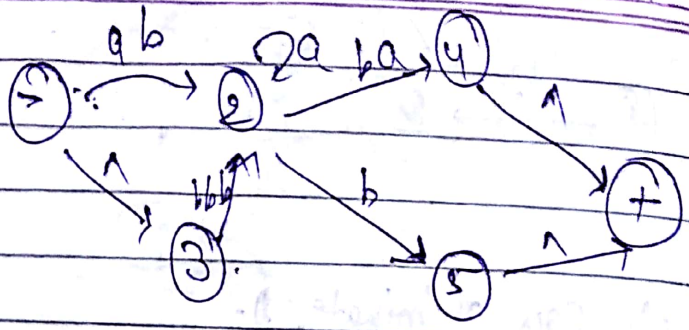
if we eliminate 1 then result is

TutorialsDuniya.com

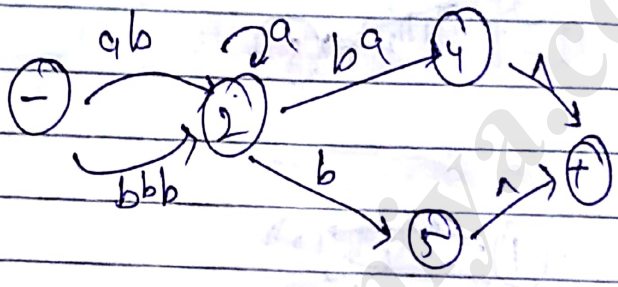
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- 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

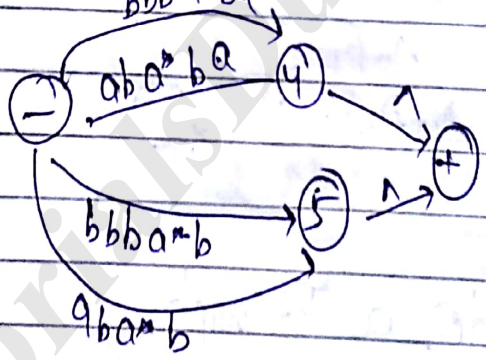
Please Share these Notes with your Friends as well

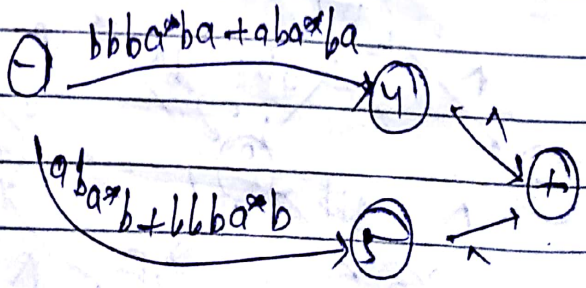


If we remove 3

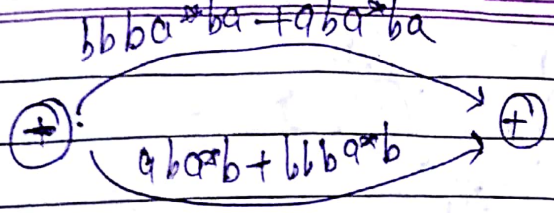


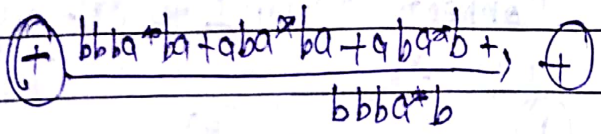
remove 2



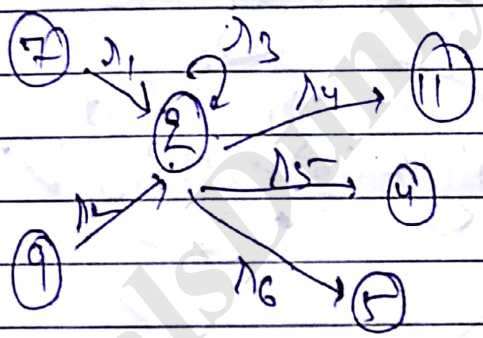


Remove 4 then 5 then result is

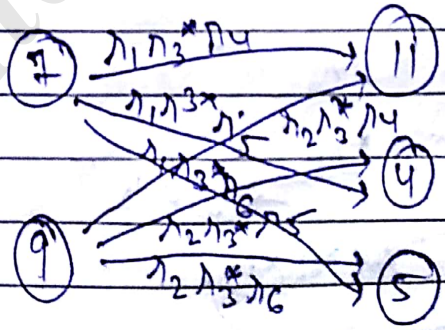




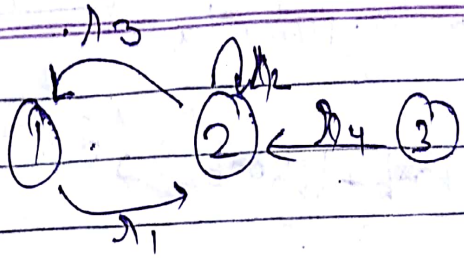
Consider another case



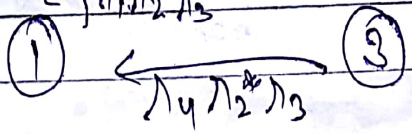
eliminate state 2



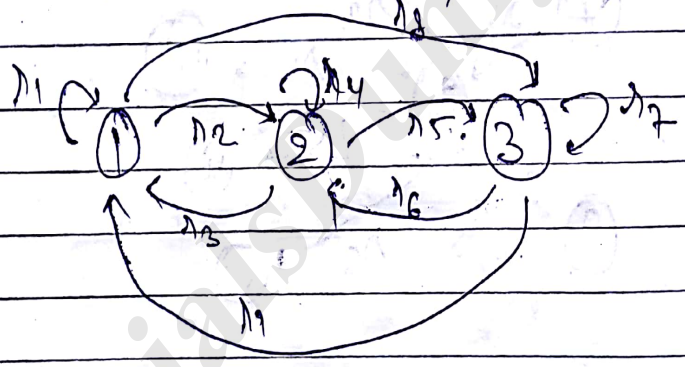
We have some special cases



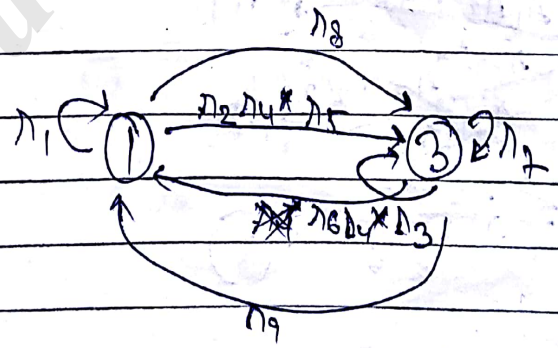
Remove 2 state
Connect incoming to outgoing edges

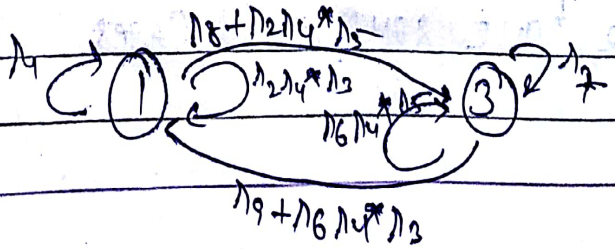


Consider another case

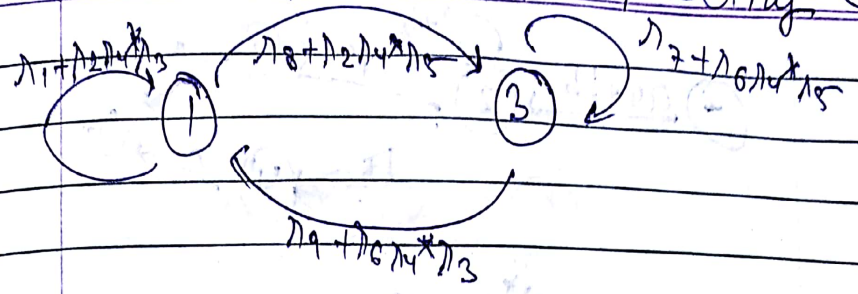


Eliminate state 2

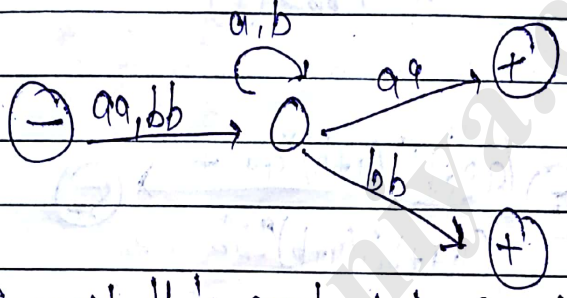




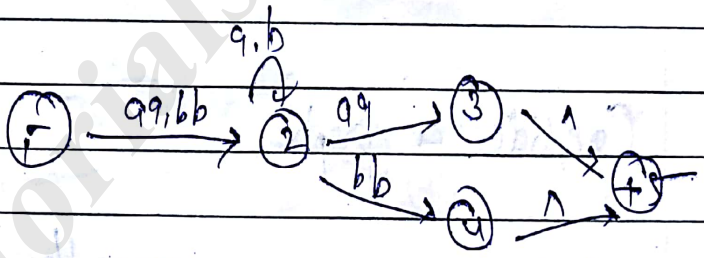
(c) Remove parallel edges by applying addition



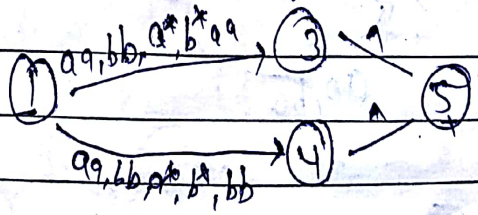
Consider a graph



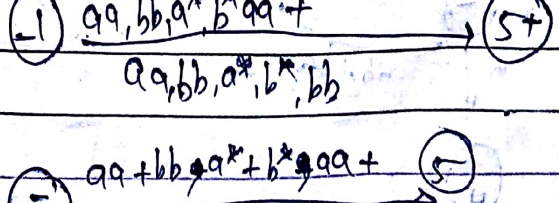
Convert this graph into regular exp.
We can have only one final state so add transition edge
$aa+bb+a^*+b^*+aa+bb$

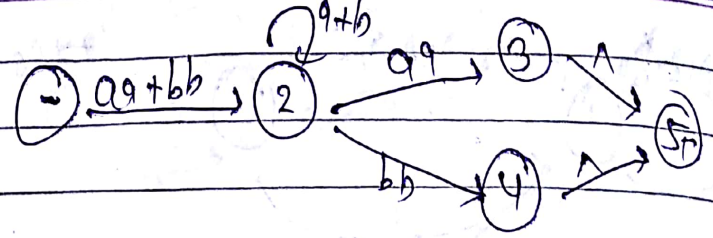


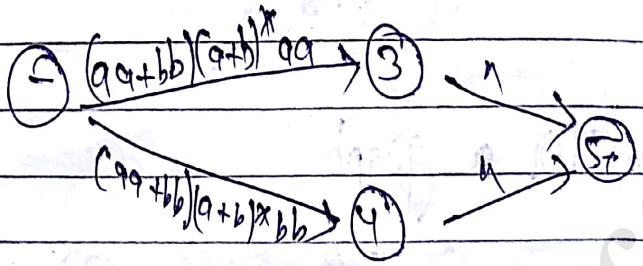
Remove 2 because we want final ⊖ — ⊕

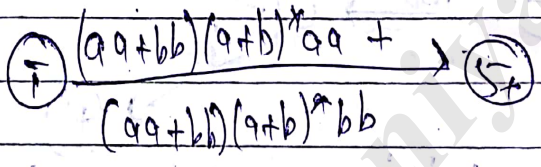


Next page



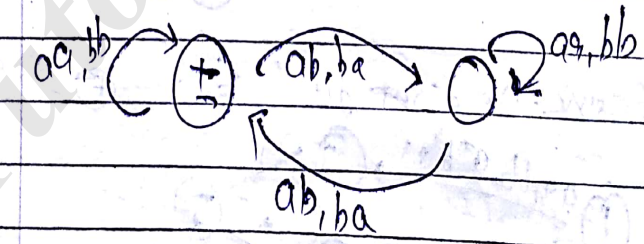


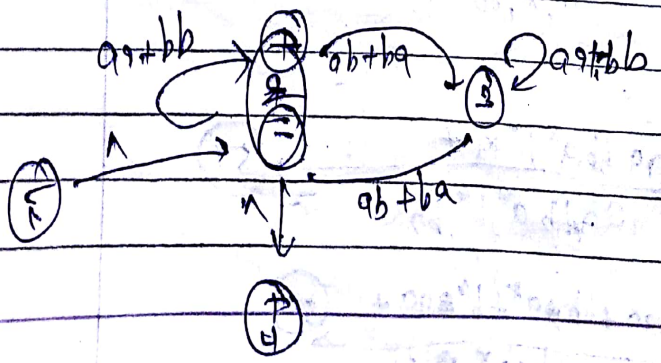




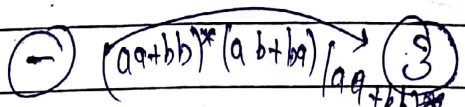
Common

Consider a graph



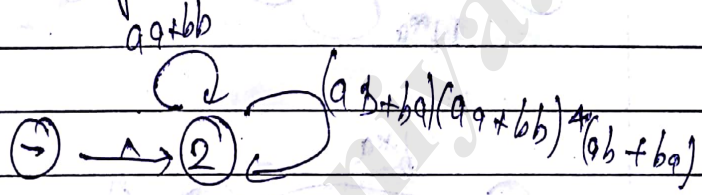


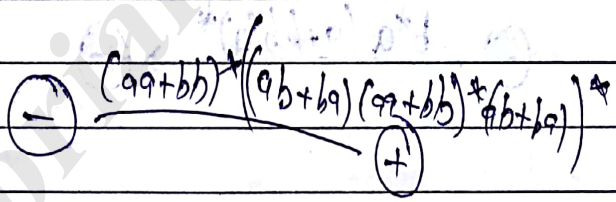
eliminate 2 and 3



(4)

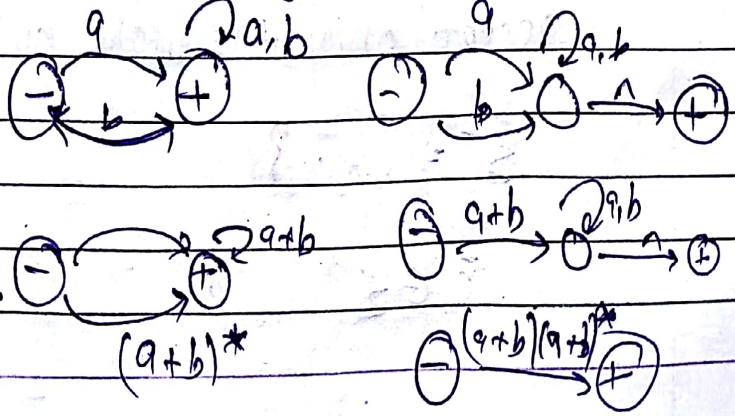
eliminate 3 first



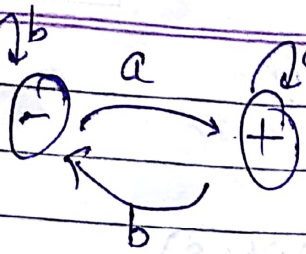


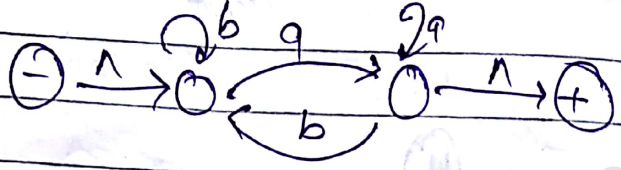
30/8

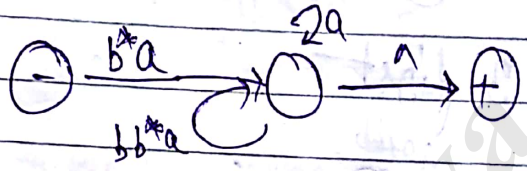
Consider this graph into Reg. Exp.

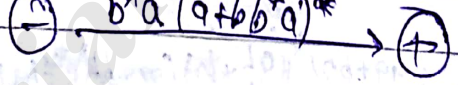


Q-









5/9/18

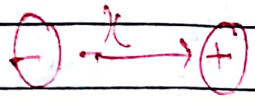
Kleen's theorem

Part 3- Rule No-1- If there is a F.A that accept only particular letter of the alphabet. there is F.A that accept only the word null.

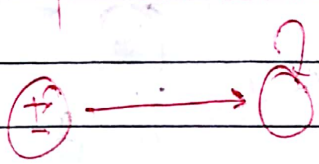
$\Sigma = \{ \ldots \}$

$x \in \Sigma$ (1)

You can draw a F.A that accept x other inputs are invalid



F.A accept null i.e. ∈ NFA of string you can declare that

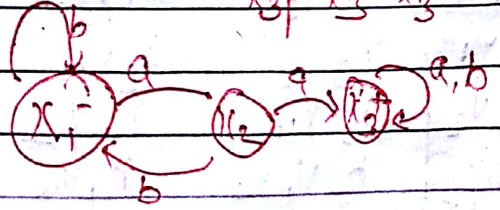


Rule No-2 — If there is a F.A called FA1 that accept the language define by Reg. Exp. that is R_1 and there is a F.A called FA2 that accept the language define by Reg. Exp is R_2 Hunt then is a finite Au. that we shall calle FA3 that accept lang. define by Reg. Exp. R_1+R_2

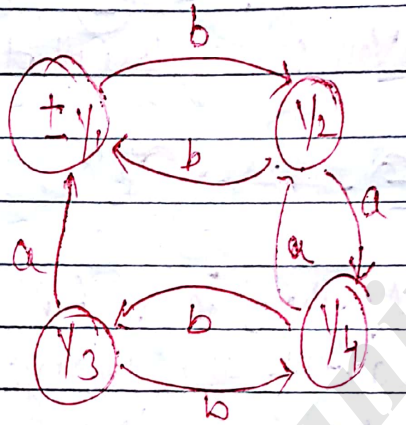
Q. Consider FA1

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

FA1	a	b
x_1	x_2	x_1
x_2	x_3	x_1
x_3	x_3	x_3



FA2	a	b
$\pm Y_1$	Y_3	Y_2
Y_2	Y_4	Y_1
Y_3	Y_1	Y_4
Y_4	Y_2	Y_3



The final the new machine accepts will be union of these two languages

$$L_1 + L_2 = FA_1 + FA_2$$

FA_3

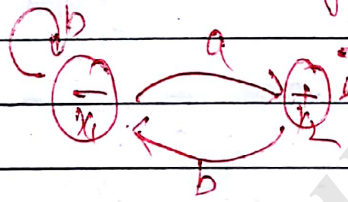
	a	b
$\pm(X_1Y_1)$	X_2Y_3	X_1Y_2
(X_1Y_2)	X_2Y_4 (final)	X_1Y_1
X_2Y_3	X_3Y_1	X_1Y_4
X_1Y_4	X_2Y_2	X_1Y_3
X_3Y_1	X_3Y_3	X_3Y_2
X_2Y_2	X_3Y_4	X_1Y_1 already have
X_1Y_3	X_2Y_1	X_1Y_4 already have

Max Rows = 4 × 3 = 12

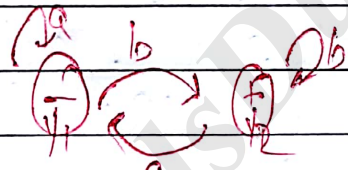
	a	b
$x_3\ y_3^+$	$x_3\ y_1$	$x_3\ y_4$
$x_3\ y_2^+$	$x_3\ y_4$	$x_3\ y_1$
$x_3\ y_4^+$	$x_3\ y_2$	$x_3\ y_3$
$x_2\ y_1^+$	$x_3\ y_3$	$x_1\ y_2$
$x_2\ y_4^+$	$x_3\ y_2$	$x_1\ y_2$

make graph of this table (self)

Q FA1 and FA2 are given (FA1 + FA2) Union



$(a+b)^*a$



$(a+b)^*b$

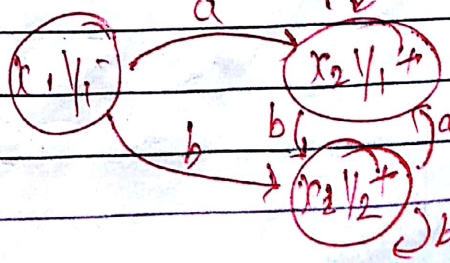
	a	b
x_1^-	x_2	x_1
x_2^+	x_2	x_1

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

	a	b
$x_1\ y_1^-$	$x_2\ y_1$	$x_1\ y_2$
$x_1\ y_2^+$	$x_2\ y_1$	$x_1\ y_2$
$x_2\ y_1^+$	$x_2\ y_1$	$x_1\ y_2$

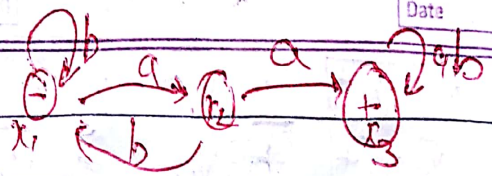
(Graph)

FA3 (FA1 + FA2) Union

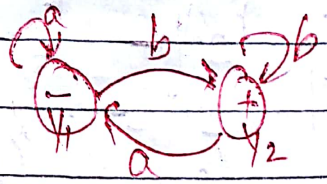


CLASSTIME | Page No.
Date

Q- FA1



FA2



Find union.

	a	b
x_1	x_2	x_1
x_2	x_3	x_1
x_3^+	x_3	x_3

	a	b
y_1	y_1	y_2
y_2	y_1	y_2

Rule No. 3 :- If there is a finite Auto. FA1 that accept the lang. define by Reg. Exp. R_1 and FA is FA2 that accept lang. define by Reg. Exp. R_2 then there is FA FA3 then lang. define by Concation $R_1 R_2$ with the product lang.

11/9/18

Regular languages—

All finite languages are Regular. but every regular are not or every lang. is not regular.

It is not Reg.

Finite. - R.L. $\{a^n b^n \ n=0,1,2,3\ldots\}$

$\{a^*b \quad aabb \quad aaabbb \ldots\}$

Reg. - $a^*b^* = \{\lambda \ a \ b \ aa \ ab \ bb \ldots\}$

A language that can be define by a reg. exp. is called a reg. language

Some infinite lang. are regular.

If L_1 & L_2 are reg. languages then L_1+L_2, L_1L_2, $(L_1L_2)^*$ are also reg. languages, we know that L_1+L_2 means the lang. of all words is either L_1 or L_2.

L_1^* means strings concation of arbitrarely many factors from L_1.

Set of reg. lang. is closed under union, concath and Kleen closure.

all these lang. are regular	L_1 $\quad$ r_1	$L_1+L_2 = r_1+r_2$
	L_2 $\quad$ r_2	$L_1L_2 = r_1r_2$
	L_1+L_2	$L_1^* = r_1^*$

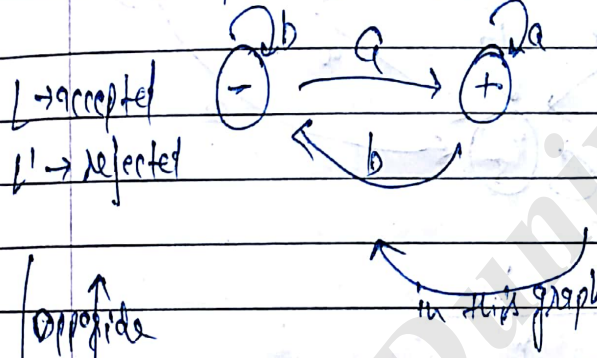
$L_1 L_2$

L^*

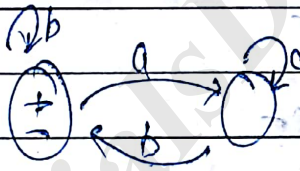
L^-

L'

if L is a reg. lang. then L' is also a reg. language in other words the set of reg. lang. is closed under complimentation.



opposite



L' → accepted
L → rejected

If L is a reg. lang. then there is some finite Auto. that accept L some of the states of this F.A are final states or most atleast not. Let us reverse the final status of each state.
i.e. if it was a final state make it non final state and non final make a final state it becomes Complement.

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

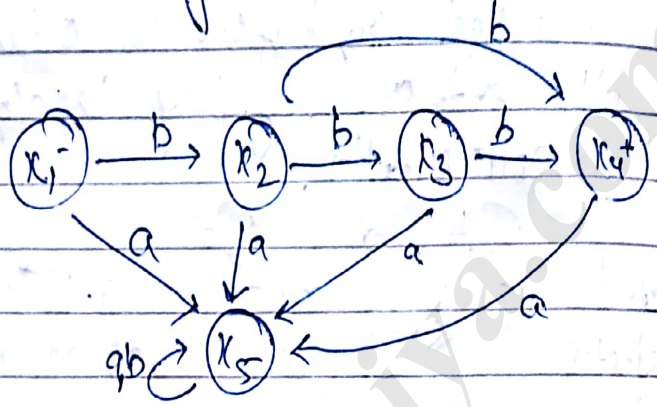
- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- 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

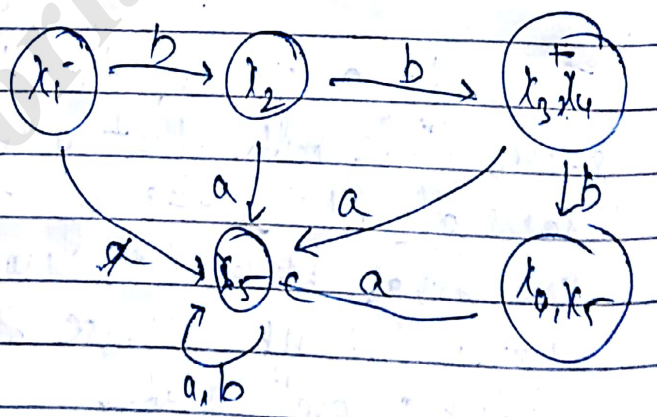
If an input string formerly ended in a non final state it now ends in final state and vice-versa.
Therefore, this machine accepts exactly the lang. $\bar{L}$

18/9/18

NFA to DFA



	a	b
$\to q_1$	q_5	q_2
q_2	q_5	$q_3 q_4$
$q_3 q_4$	q_5	$q_4 q_5$
q_5	q_5	q_5



Convert each of foll. NFA into FA

Reg. languages –

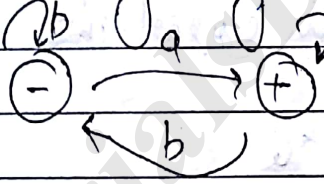
Complements & intersection

If L is lang. over the alph. Σ define its complement L' to be the lang. of all string of letter Σ will not words in L.

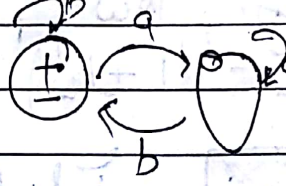
E.g. If L is the lang. over the alph. Σ then have AA in them
L' is lang. of all words that do not have AA

L = (a+b)* aa (a+b) reg. lang
L' = is not a reg. lang. (not compulsory)

If L is a reg. lang. then L' is also reg. lang.



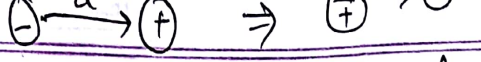
L = accepting (a+b)*a
{a aa ba

L' = 

{Λ b ab bb

If L is a reg. lang. and there is some FA that accept lang. L some of the state of this F.A final states and most likely some are not.

☆ To do complement of a lang. remove (+) and add their initial state eg.



Let us reverse the final states of each state, i.e. if it was a final state make it a non-final stage and if it is a non-final stage make it a final state.

This new machine accept all input strings which were not accepted by original final Auto.
(All words in L')

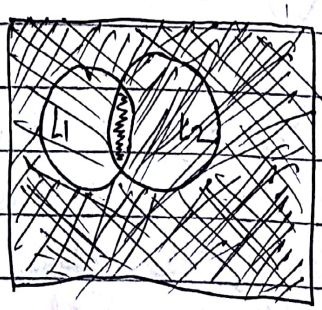
Theorem-12- If L1 and L2 are reg. languages then L1 ∩ L2 also a reg. lang.
In other words a set of closed lang. under closed intersection.

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

$L_2 = (a+b)^*$

By De morgan's law

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



If L_1 L_2 are reg then $L_1 \cap L_2$ is also reg.

Λ → string with 0 length

ϕ → string with no string

CLASSTIME | Page No. 420
Date 18/09/2018 (45)

Q4/9/18 $L_1 \cap L_2$

$(L_1' + L_2')'$

Λ_1 Λ_2

FA_1 FA_2

FA_1' FA_2'

$FA_1' + FA_2'$

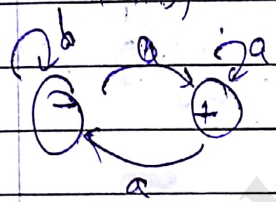
$(FA_1' + FA_2')'$

R.E (Reg. Exp.)

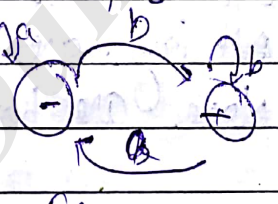
Q – Consider $L_1 = (a+b)^* a$ & $L_2 = (a+b)^* b$ find $L_1 \cap L_2$ ∴ F.A, RE

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

L_2 : $(a+b)^* b$

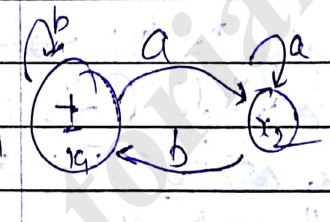


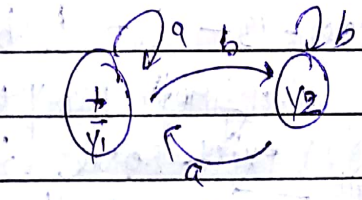
FA_1



FA_2

FA_1'





16

	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

8:30
9:0

$FA_1' + FA_2'$

	a	b
$x_1 y_1$	$x_2 y_1$	$x_1 y_2$
$x_2 y_2$	$x_2 y_1$	$x_1 y_2$

Make a graph

$L_1 \cap L_2 = \phi$

Ch-9 Page-185
Q- first 10 Q. TRY

→ $b(a+b)^*a$

Q- Consider find $L_1 \cap L_2$

L_1 : $(a+b)^*$

L_2 : $b(a+b)^*$

Non-Regular languages —

A lang. cannot be define by a reg. exp. is called a non-reg. lang.

by kleen's theorem F.A is not possible for NRL as by well as Transition Graph.

All lang. are either Reg. or non-reg. None are both.

25/9/18

Theorem-13 — Let L be any reg. lang that has infinitely many words then there exist some three strings x, y and z

Null — $y \neq \Lambda$ not the null string

i.e, all strings are the form xy^nz where $n = 1, 2, 3, \ldots$ are words in L.

$xy^nz \in L$

Consider If L is reg. lang. there is F.A that accepts words in L

Now Consider,

$w \in L$

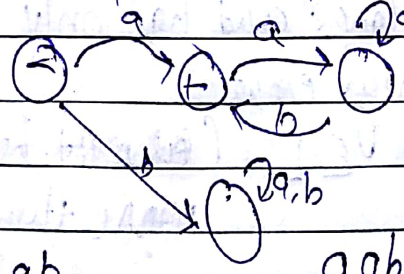
that have w be some words in L that ≥ more letters in it there are some states in machine

When this word generate the path to the machine the path cannot visit a new state for a new letter. i.e, a must at some point revisit state that it has been before.

Now, Let's break w into three parts

(i) Call parts x start letter with w starting with beginning that lead up to first state that is revisited.

(ii) Starting the letter after the substring x let y denote the substring the w that travels around the ckt coming back to the same state the ckt began because there must be substring y cannot be the null string y contains the letters of w for exactly one loop around the ckt.



aaaab
x y

aabab
x y

(iii) Let z be the rest of w starting with the letter after the substring y and y to the end of string w.

z should be null. path for z could also possibly loop around

aaaab $xy^n z \in L$

x y z

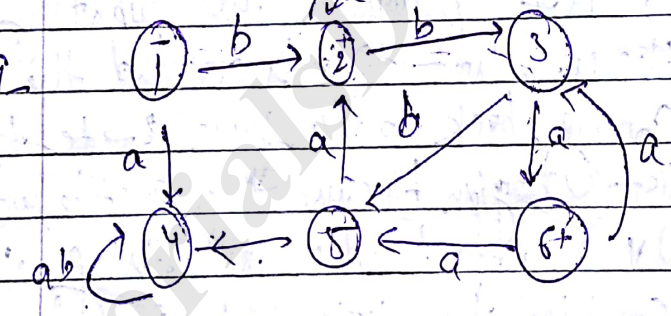
aaaaaaab

x y y y z

This idea of pumping of y substring is called pumping lemma. We are repeating y we are pumping substring y.

$a^n b^n$. n = 1 2 3 4 — —

26/9



Consider a graph this machine accepts infinite language and has only 6 states and pumping lemma

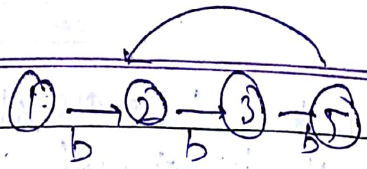
$w \in L$ (w must have 6 or more than 6 words)

Consider w to be bbbababa

break w into three parts x, y & z

bbbababa

x y z



bb bababa
x y z

before revisit ckt (revisiting) rest of string in z

$xyz \in L$
$xyyz \in L$
$xyyyz \in L$

(pumping lemma) $xy^n z \in L \quad n = 1, 2, 3, 4 \ldots$

pumping Lemma is only valid on Reg. Lang.

Q- Consider another lang —
$a^n b a^n$ xyz
$y = a$
$a^{10} b a^{10}$
$a^{11} b a^{10} \notin L$ } because an should
$a^{10} b a^{11} \notin L$ } be equal $a^{10} b a^{10}$

We cannot have y that can be pumped into the string is not there for not reg.

Q- $a^n b^n a b^{n+1} \quad n = 1, 2, 3 \ldots$

Let L be an infinite lang. excepted by FA with N no. of states then for all words w in L that have

Ch-10 is not in syllabus
Ch-11 " " "

more than 1 letters there are strings x, y & z where y is not null.

$|x| + |y| \leq n$ (x)

length of x in y does not exceed n s.t $w = xyz$ and all strings of the form $xy^nz \in L$ $n = 1, 2, 3 \ldots$

Pumping Lemma

Ex-195

Show that lang. pallindrome is not reg.

Q- Consider the lang. prime
prime = $\{a^p \mid p \text{ is prime}\}$
is not regular

Q. Show the lang. is non-regular -
(i) $a^n b^{n+1}$
(ii) $a^n b^n a^n$
(iii) $a^n b^{2n}$
(iv) $a^n b^* a^n$

Part-2

Pushdown Automata theory

Ch-12 Context free Grammar -

Syntax as a method for defining lang -

Consider an exp.

$$\frac{\frac{1}{2}+9}{4+\frac{8}{21}+\frac{5}{3+\frac{1}{2}}}$$

Consider arithmetic exp.(AE) You are given some rule

Rule-(i) Any no. is in the set of AE

(ii) If x and y are in AE then so are

x, y ∈ AE

(x)

(-x)

(x+y)

(x-y)

(x*x)

(x/y)

(x**y)

(3+4*5)

(3+4)*5) (3+(4*5))

Terminals (T)

Non-terminals (NT)

A context free Grammer is a collection of three things-

(i) An alphabet Σ of letters call terminals from which we are going to make string that will words of lang.

(ii) A set of symbols called non-terminals. One of symbol which is capital S studying for start here.

(iii) It finite set of production of a form out NT into finite strings of terminals or non-terminals (NT) while the strings of T & NT can consist of only T or of only NT of any mixture of T and NT or even the empty string.

1/10/19 Consider rules for the formal arithmetic exp.

Start → AE
AE → (AE+AE)
AE → (AE-AE)
AE → (AE*AE)
AE → (AE/AE)
AE → (AE**AE)
AE → (AE)
AE → -(AE)
AE → any number

Any number
Rule 1 - Any no → FD
Rule 2: FD → FOOD
Rule 3: FD → 123-9
Rule 4: OD → 0123-9

(3+(4*5))

Start → (AE)

⇒ ((AE+AE))

⇒ ((AE+(AE*AE)))

⇒ ((any no.+(any no.+any no.)))

⇒ ((3+(4*5)))

Any no → FD

→ 3

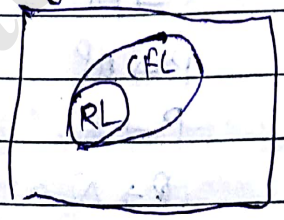
Any no → FD

→ F0D

→ 20

The lang. generated by CFG is a set of all strings of terminal that can be produced from the start symbol using the productn as substitutn

A lang. generated by CFG is called CFL



EVERY RL IS CFL

(start) S → A {n}

S → a {a}

S → a

S → b {a b}

S → A

S → a

S → b {n a b}

All finite Lang. are CFL

a^*

$S \to aS$

$S \to \wedge$

b^*

$S \to bS$

$S \to \wedge$

$S \to aS$

$\to a$

$S \to aS$

$\to aaS$

$\to aa$

$\therefore S \to \wedge$ ~~aa^*~~

$S \to aS$

$S \to a$

$S \to aA$

$A \to aA$

$A \to \wedge$

---

ab^*

$S \to aB$

$B \to bB$

$B \to \wedge$

$(a+b)^*$

$S \to aS$

$S \to bS$

$S \to \wedge$

$a(a+b)^*$

$S \to aS_1$

$S_1 \to aS_1$

$S_1 \to bS_1$

$S_1 \to \wedge$

$(a+b)^*a$

$S \to Pa$

$P \to aP$

$P \to bP$

$P \to \wedge$

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

$S \to PaP$

$P \to aP$

$P \to bP$

$P \to \wedge$

$(aa+bb)^*$?

a^*b^*

$(a^*+b)^*$

$(aa+b)^*$

try these

$(aa+bb)^*$

$S \to aaS$

$S \to bbS$

$S \to \wedge$

a^*b^*

$S \to pq$

$p \to ap$

$p \to \wedge$

$q \to bq$

$q \to \wedge$

$(a^*+b)^*$

$S \to pS$

$p \to (ap+b)p$

q10 RE RL CFL

finite

Infinite

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

$S \to XaaX$

$X \to aX$

$X \to bX$

$X \to \wedge$

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

$S \to XaX ab$

$X \to aX$

$X \to bX$

$X \to \wedge$

a^*b^*

$S \to aS$

$S \to \wedge$

$S \to bS$

$S \to \wedge$

$S \to AB$

$A \to aA$

$A \to \wedge$

$B \to bB$

$B \to \wedge$

$(a^*b^*)^*$

$S \to \wedge$

$(AB)^*$

$A \to aA$

$A \to \wedge$

$B \to bB$

$B \to \wedge$

$S \to ABS$

$S \to \wedge$

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

$(a^*+ab)^*$

$S \to AS$

$S \to BS$

$S \to \Lambda$

$A \to aA$

$A \to \Lambda$

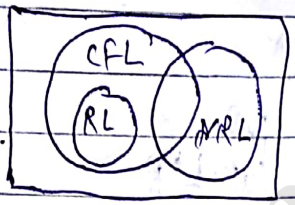
$B \to ab$

for Non-Reg. Languages

$a^n b^n$ $n = 1, 2, 3 \ldots$

$S \to aSb$

$S \to ab$



(NRL) $a^n b^n c^n$ $n = 1, 2, 3, \ldots$

$S \to aXbYc$

$X \to ab$

$Y \to bc$

Consider a CFG

all even a

$S \to aSa$

$S \to \Lambda$

all even b

$S \to bSb$

$S \to \Lambda$

odd length palindrome

$S \to aSa$

$S \to bSb$

$S \to a$

$S \to b$

even either aa/bb

even palindrome

$S \to aSa$

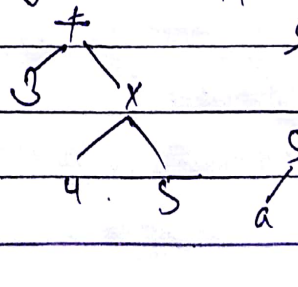
$S \to bSb$

$S \to \Lambda$

(even length)

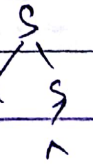
If we add $S \to \Lambda$ it will give all types of palindrome

$3+4*5$



$S \to aS$

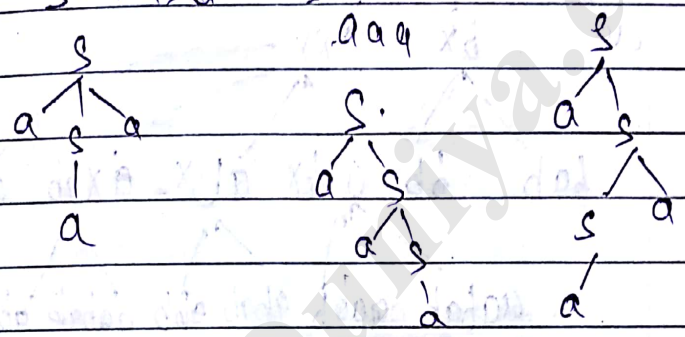
$S \to \Lambda$



If CFG is called ambigious If for atleast one word in the lang. that is generates there are 2 possible derivn of the word that corresponds to diff. syntax trees. If CFG is not ambigious it is called unambigious.

$S \to aSa$ $S \to a$

aaa



10/10

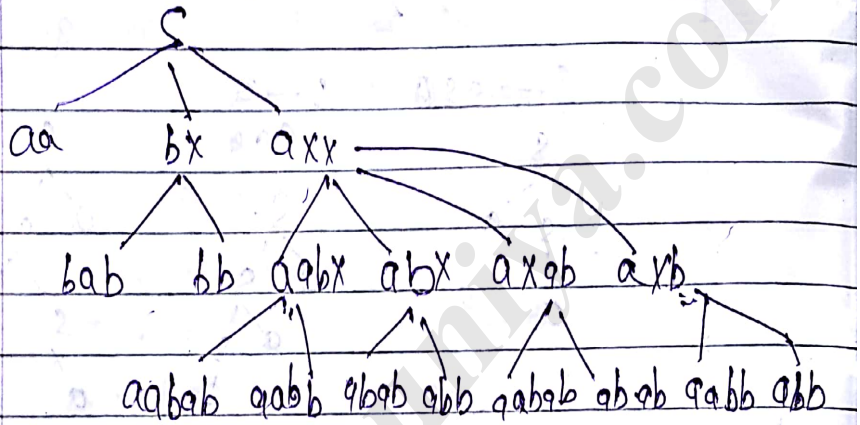
The total lang. tree (TLT)

It is possible to depict the genn of all the words in the lang. of CFG simultaneously in one big tree (possibly infinite tree). In the lang. infinite. For a given CFG we define a tree with the * symbol(s) as a root whose nodes are working strings of terminal and non-terminals the decendents of each node are all the possible results of applying every applicable string, one at a time.

If string of all terminals is a terminal node in a tree, the resulting tree is called TLT

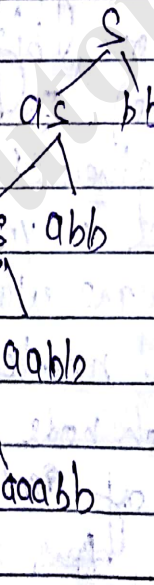
$S \to aa \mid bX \mid aXX$

$X \to ab \mid b$



Q- 1, 2, 3, 4, 5, 7, 8, 11, 13, 16, 17

Q- $S \to aS \mid bb$ $\quad a^*bb$



a^*S

Q- Give CFG for $(baa+abb)^*$

$S \to (X+Y)^*$

$S \to X^*$

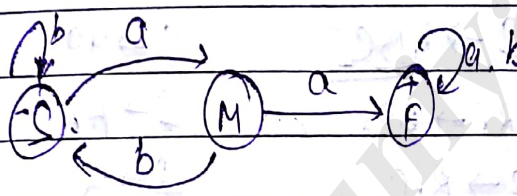
$S \to Y^*$

$S \to \wedge$

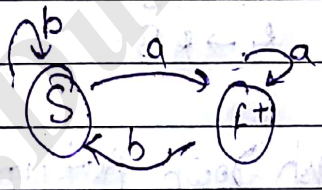
Ch-13 is not in syllabus

{ Strings of length 2 ∀ $\Sigma(a+b)^*$ }

aa
ba
aa aa
ab aa
ba aa
bb aa
aaa aaa



$S \to aM$
$S \to bS$
$M \to aF$
$M \to bS$
$F \to aF$
$F \to bF$
$F \to \wedge$



$S \to aF$
$S \to bS$
$F \to aF$
$F \to bS$
$F \to \wedge$

CNF (Chomsky Normal form)

If { $S \to A$
$A \to a$
Then → $S \to a$

If L is a lang. generated by CFG then there is another CFG that generates all the non-null words

If all of rules production are of one of the basic form

$NT \rightarrow$ str. only NTs

$NT \rightarrow T$

If a CFG has only production of form

$NT \rightarrow$ str. of exactly two NT's

$NT \rightarrow T$

$S \rightarrow ABC$

$R_1 \rightarrow AB$

$S \rightarrow R_1C$

$R_1 \rightarrow AB$

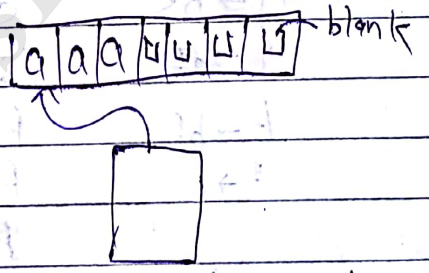
$S \rightarrow aa$

$S \rightarrow AA$

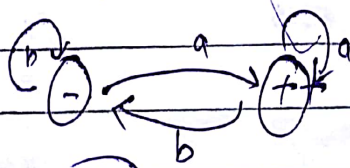
$A \rightarrow a$

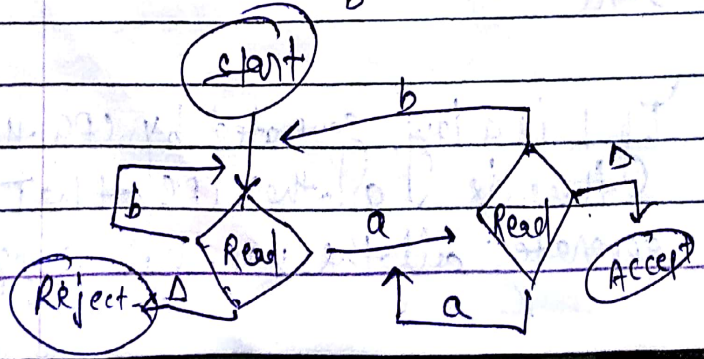
Push Down Automata (PDA)

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



Consider a finite automata that accept all words in the letter ending a



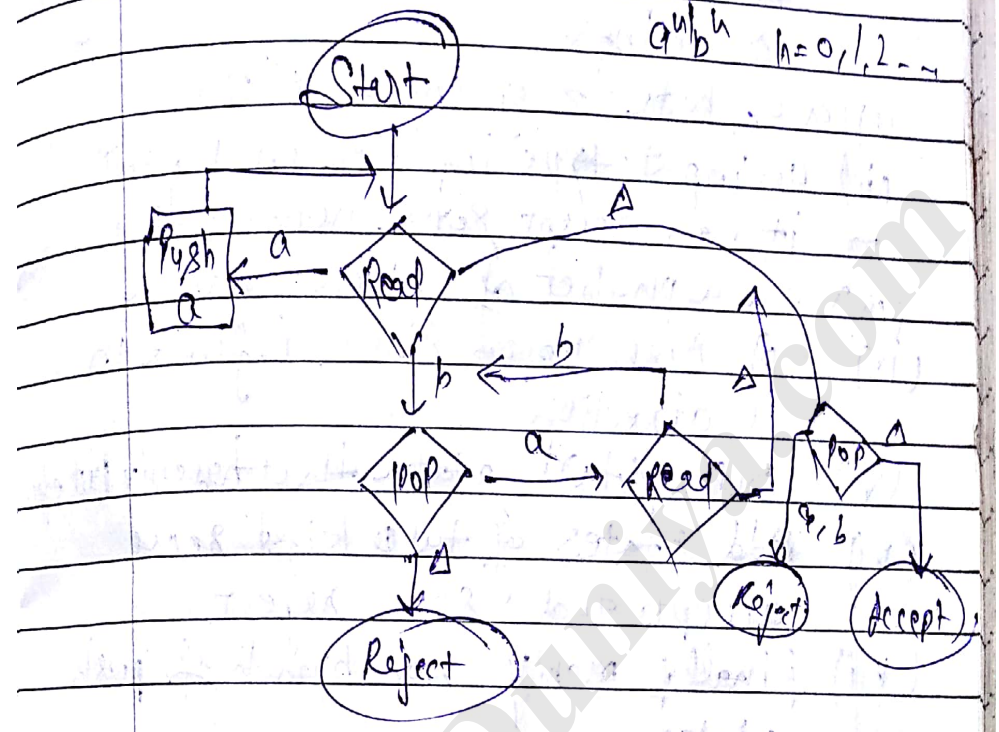


CLASSTIME | Page No. | Date | 61

PDA –

Consider a PDA

$a^n b^n$ $n = 0, 1, 2 \ldots$



$a^n b^{2n}$ aaa bb bb bb

$a^{2n} b^n$

$a^n b^{2n+1}$

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

for every CFL = PDA

11/10 Consider a PDA

N PDA (Non-Deterministic Push Down Automata)

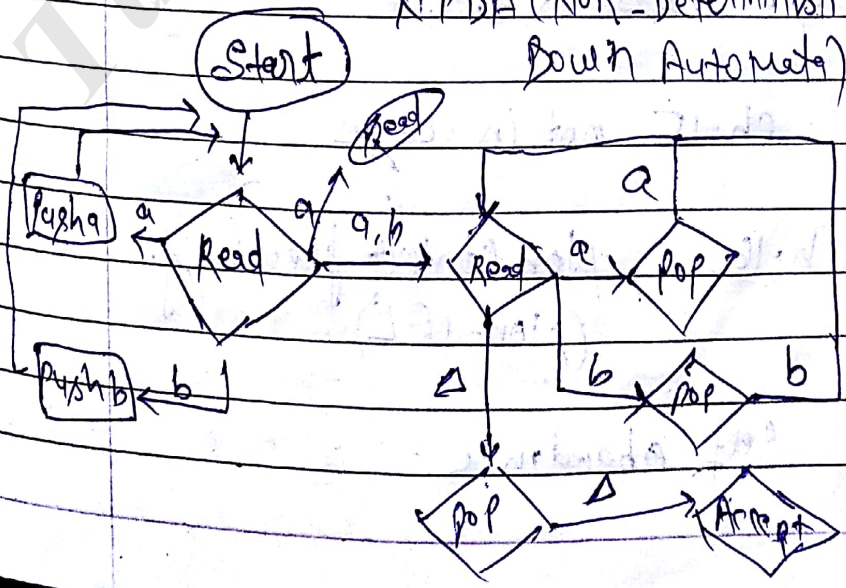


Fig. given on page no- 302

→ PDA

A pushDown Automata collection of 8 things.

(i) a alphate Σ of input letters
(ii) an input type infinite in 1 direct^n
→ it can accept some non-Reg. lang.
(iii) An alphabet of stack char.
(iv) A push Down stack, infinite in 1 direction.
(v) One start state that has only Out edg.
(vi) Add states of two kind some accept and some reject.
(vii) Finally many non-branching push states
(viii) Finally many branching states of two type one read, one pop.

Theorem For every lang. L there is some PDA that accept it.

Q - 1, 2, 3, 4, 5, 6, 7, 8

Ch-15 not in course

Ch-16 Non Context free lang. (Non-CFL)

Self abandance -

Now consider CFG which is in CNF all is production are of 2 forms

$NT \rightarrow NTNT$

$NT \rightarrow T$

Let G be a CFG in CNF and let's call production $NT \rightarrow NTNT$ as live production and $NT \rightarrow T$ as dead production. Every time we apply a live production we increase no. of NT by 1 and ... dead we decrease no. of NT by 1.

Consider for eg.

$S \rightarrow b$ (dead)

Suppose G has p - live prodⁿ

q → dead prodⁿ

Because any derivation that derived resize live prodⁿ must have atmost p+1 live prodⁿ

Now consider a case

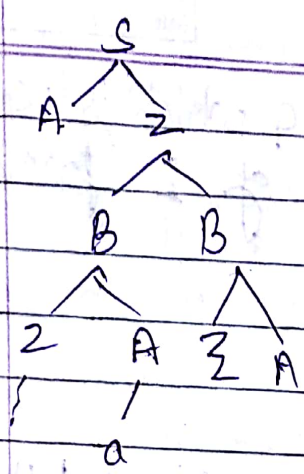
$S \rightarrow AZ$

$Z \rightarrow BB$ $\quad B \rightarrow b$

$B \rightarrow ZA$

$A \rightarrow a$

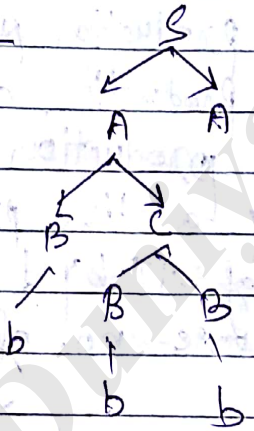
Qs-1



second Z will derive from first Z.

$S \to AA$
$A \to BC$
$C \to BB$
$A \to a$
$B \to b$

Case-2



The second B is not descented from first B

Because a grammer is in CNF replace every NT with 1 or 2 symbols and the derivn tree of each word has the property that ever node has 1 or 2 desendents such a tree is called binary tree and in an ∞ way, we can always find an exp. of case 1.

Theorem If G is a CFG in CNF that has p line and q dead prod^n and w is a word generated by G that has more than 2^p letters $|w| > 2^p$ there is somewhere in every derivn tree for string w there is an exp. of

TutorialsDuniya.com

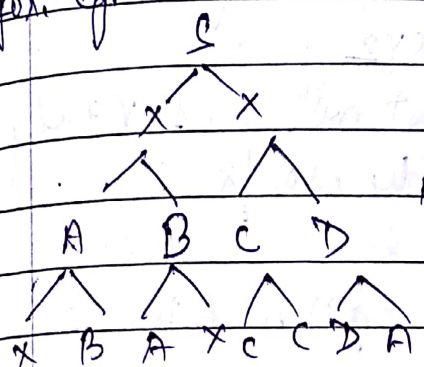
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- 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

some NT call it z being used twice where the 2nd z is descended from first z.

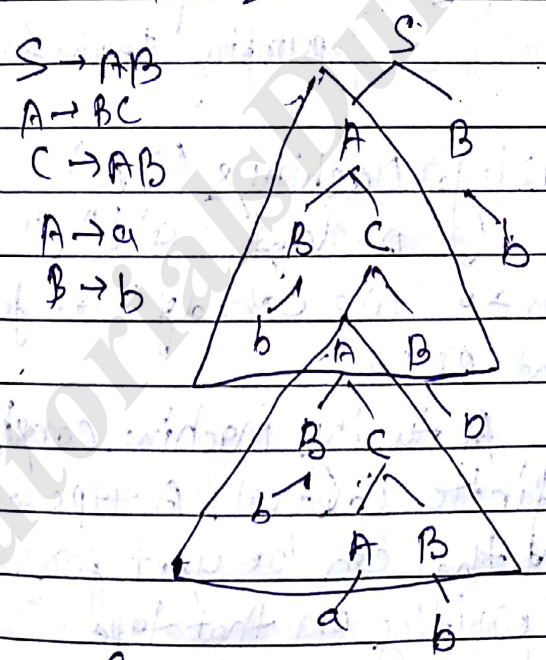
for eg.

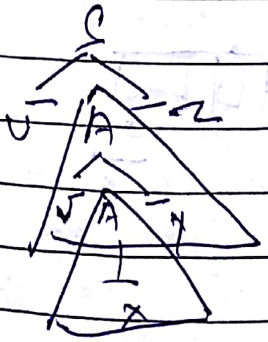


If bottom most has >2 letters the tree must have more than p+1 nodes.

Now consider a CFG
$|w| > 2^p$

$S \to AB$
$A \to BC$
$C \to AB$
$A \to a$
$B \to b$





If G is any CFG in CNF with p line prodn and w is any word generated by G with length $\geq 2^p$ then we can breakup w into five sub strings

$w = uvxyz$

s.t x is not null and v and $y \in \Lambda$ and s.t all the words

$uvxyz$

$uvvxyyz$ for $n = 1, 2, \ldots 3$

$uvvvxyyyz$

$\vdots$

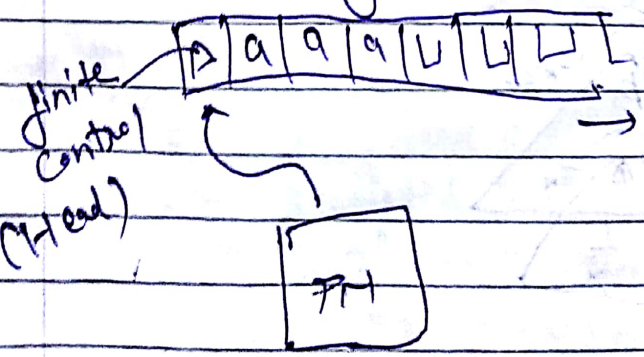
$uv^n xy^n z$

Can also be generated by G. This is pumping lemma for CNF.

22/10 Turing Machines (TM)

A lang. $a^n b^n c^n$ where $n \geq 0$ we can definite for it FA and PDA.

A turing machine consist of a finite control. A tape and a Head that can be used for reading and writing on that tape.



The TM seem to form a stable and maximal class of computer devices in terms of computⁿ they can perform. The idea of an algo. must be equiv. to the (notion) TMs are design to generally satisfy the foll. three criteria —

(i) They should be automata i.e. there functi and conds. as the devices previously studied

(ii) They should be as simple as possible to describe define formally and reason about.

(iii) They should be as general as possible in terms of computⁿ as they carried out.

Definitⁿ of TM —

TM is a quintuple $(K, \Sigma, \delta, s, H)$

K - is finite set of states

Σ - is an alphabet containing blank symbol ($\sqcup$), left hand symbol ($\triangleright$) but not containing the symbol ($\rightarrow \leftarrow$)

$s \in K$ is the initial state

$H \subseteq K$ set of halting state (final)

δ q, $\delta(q,a) \to q'$, The transⁿ func is a fun. from $(K-H) \times \Sigma \to K \times (\Sigma \cup \{\leftarrow, \rightarrow\})$ s.t.

(a) $q \in K-H$, if $\delta(q, \triangleright) = (p, b)$ then $b = \triangleright$

(b) $q \in K-H$ and $a \in \Sigma$ if $\delta(q, a) = (p, b)$ then $b \neq \triangleright$

Consider

▷	a	a	a	⊔	⊔	⊔	.

$\triangleright \underline{a}\, bab \sqcup \sqcup \sqcup$ $\quad \delta(q_0, a) \to (q, \sqcup)$

$\triangleright \sqcup \underline{b}ab \sqcup \sqcup$ $\quad \to (q_1, \sqcup)$

$\triangleright \sqcup b \underline{a} b \sqcup$ $\quad \to (q_1, \triangleright)$

$\triangleright \sqcup \underline{\triangleright} a b \sqcup$

$\triangleright \sqcup \sqcup \underline{a} b \sqcup$ — represent position of Head

$\triangleright \sqcup \sqcup b \underline{b} \sqcup$

$\triangleright \sqcup \sqcup \sqcup \underline{b} \sqcup$

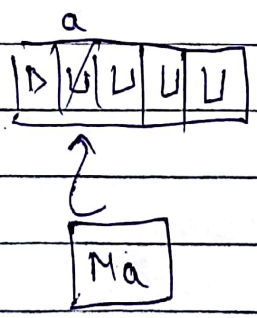
$\triangleright \sqcup \sqcup \sqcup \underline{\sqcup} \sqcup$

24/10 Annotation of TM —

Tabular form of a TM are quite complex and hard to interrupt when a annotation is needed for TM i.e. for more graphic and transparent. We ~~use~~ using Pictorial notation, ~~etc~~

The basic machines ~~etc~~
(i) Symbol writing machines
(ii) Head moving machines

A writing machine
$\Sigma = \{a, b\}$

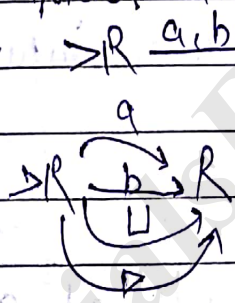


The TM's will be combined in a way suggesting of a structure of a FA. Individual machines are like the states of F.A

The machines may be connected to each other in the way that the states of FA connected together.

The connection of one machine to another is not pursued until the first machine halt. Another machine is then started from its initial state with the tape and the head position as they will left by the first machine.

Consider a machine



▷aUU
▷aUU
▷aUU

Q. ▷aaUU >RR

▷aaUU
▷aaUU
▷aaUU

Q. ▷aaUU >bRa→R

▷aaUU
▷baUU
▷baUU
▷baUU

A Machine that finds first blank sq. to the right

▷aaaaUU >R
▷abaaaUU

$\triangleright aaqLaqL \quad > L^{2\sqcup}$

A machine which find first Non-blanks sq. to the Right

$\triangleright LLLaqLL$

$> R^{2\sqcup}$ → to move right until non-blank symbol.

or $R\sqcup$

$> L^{2\sqcup}$ → to move left

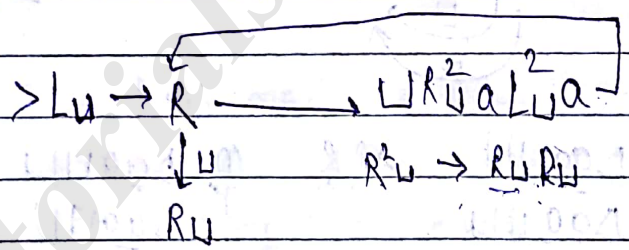
or $L\sqcup$

Copying Machine — (C)

C starts with input w

$\sqcup w \sqcup$

$\sqcup w \sqcup w \sqcup$



$\triangleright \sqcup aaa\sqcup\sqcup \quad \triangleright \sqcup aaa\sqcup w$

$\triangleright \sqcup aaa\sqcup w \quad \triangleright \sqcup\sqcup aa\sqcup\sqcup w$

$\triangleright \sqcup\sqcup aa\sqcup\sqcup w$

$\triangleright \sqcup\sqcup aa\sqcup w\sqcup$

$\triangleright \sqcup\sqcup aaw\sqcup\sqcup$

$\triangleright \sqcup\sqcup aaw\sqcup w$

$\triangleright \sqcup\sqcup aawaw$

$\triangleright \sqcup\sqcup aawaw$

$\triangleright \sqcup\sqcup aawaw$

$\triangleright \sqcup aaaw\sqcup w$

$\triangleright \sqcup aaaw\sqcup\sqcup$

$\triangleright \sqcup aaawaw$

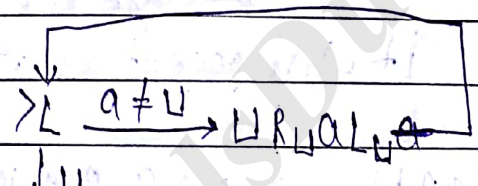
$\triangleright \sqcup aaawaw$

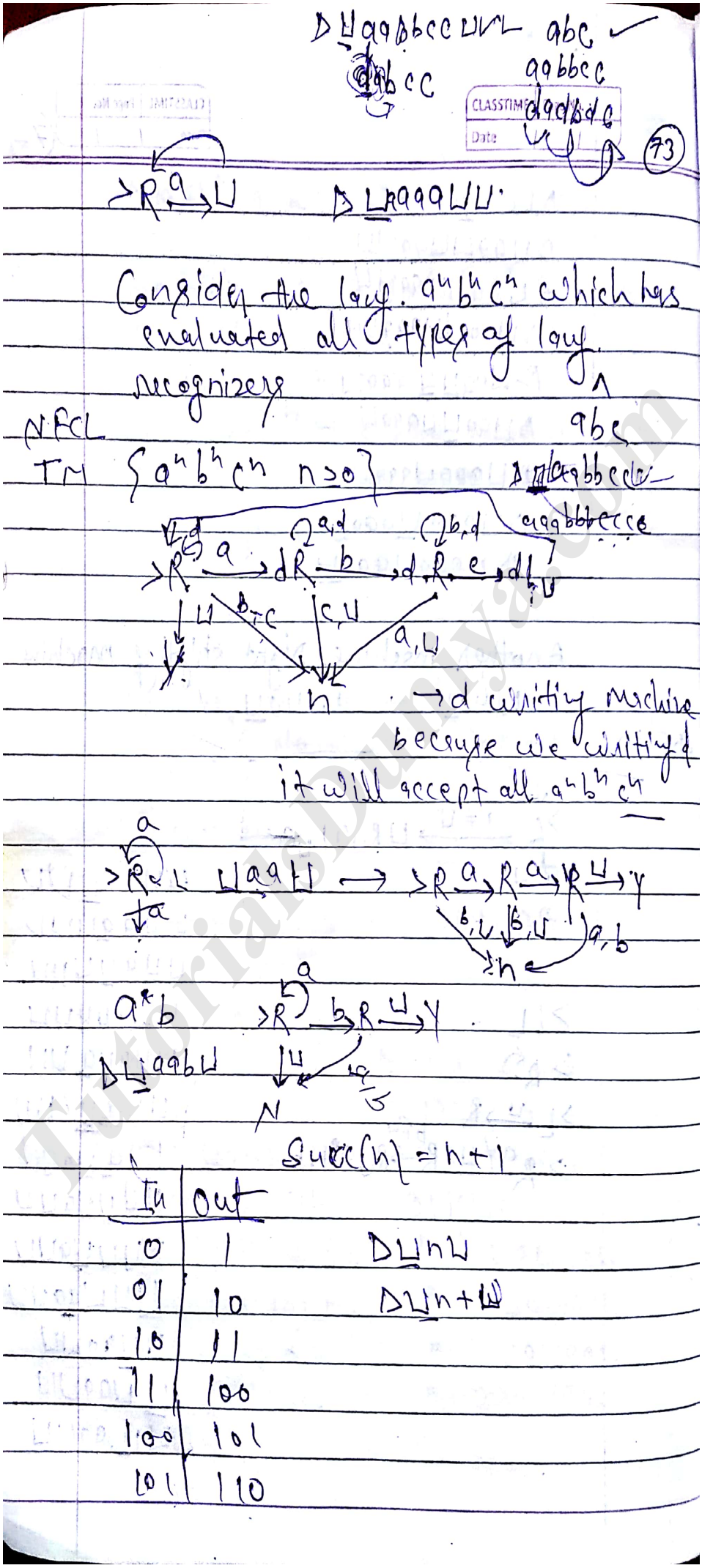
$\triangleright \sqcup aaawaw$

$\triangleright \sqcup aaawaw$

Another machine Right shifting machine

25/10





Consider the lang. $a^n b^n c^n$ which has evaluated all types of lang.

recognizers

NFCL

TM $\{a^n b^n c^n \; n \geq 0\}$

→ writing machine because we writing & it will accept all $a^n b^n c^n$

$a^* b$

Succ(n) = n+1

In	Out	
0	1	ΔLnW
01	10	ΔLn+1W
10	11	
11	100	
100	101	
101	110	

30/10

Recursive functions

Let M be a TM

$M = \{K, \Sigma, \delta, s, H\}$, and let

$\{a, b\}$ $\Sigma_0 \subseteq \Sigma - \{\sqcup, \triangleright\}$

$\{0, 1\}$ $w \in \Sigma_0^*$

Now, suppose that M halts on input w

$$(s, \triangleright \sqcup w \sqcup) \vdash_M^* (h, \triangleright \sqcup y)$$

then y is called the output of M on input w

$M(w) = y$

$M(w)$ is defined only if M halts on input w.

$y \in \Sigma_0^*$

Now Let $f: \Sigma_0^* \to \Sigma_0^*$

We say that M computes fun. F if for all $w \in \Sigma_0^*$

$M(w) = f(w)$ (input)

i.e. an eventually holds for all w and when it does hold its tape contain the string $\triangleright \sqcup f(w) \sqcup$

A func. f is called recursive if there is a TM M that computes f.

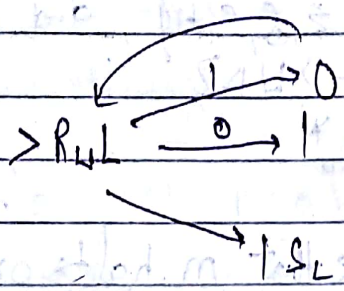
Ex - 4.2.2

* any lang we have TM for that that is recursively lang —

i/p ▷ ⊔ w ⊔ ∈ Δ ⊔ w ⊔ w ⊔

o/p ▷ ⊔ w w ⊔ Σ

↳ we want ∈ Σ



▷ ⊔ 0 | 0 | 0 | ⊔

Recursively Numerable languages —

If a TM decide a lang. and computes a function it can be regarded thought of Σ and algo that performs correctly and reliably some computational tasks

Let M a TM. Let Σ_0^*

Let L a subset of Σ_0^*

We say that M semi decide ∈ all if for any string $w \in \Sigma_0^*$ the foll. is true, $w \in L$ —

If and only if M holds on input w

A Lang L is recursively enable if and only if there is a TM M that semi decide L

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