20/7/23

Chu ✓ Formal language & Automata theory (TOC)

↓Imp

- language
- machine
- grammers

Formal language → basic thing = symbols

Σ = {a, b, c} (Here, symbol can be anything)

↓ finite set of ~~alphabets~~ input symbols

String – collection of symbols will make any string. (sum of word) [we can create infite sequence of i/p does matter. no of words] & where

Σ = {0}, {ε, 0, 00, 000, 0000 ... ∞} – strings possible

↓ symbol ↓ set of

Σ = {0, 1}, {ε, 0, 1, 00, 01, 10, 11, 000, 001, ... ∞}

- The smallest string is 'ε'. It is a '0' length string. the smallest string possible for any lang is 'ε'.

Σ = {a, b}

{ε, a, b, aa, bb, aaa, aab ... ∞}.

meaning is not imp in string, what is imp is sequence of chars like

aab ≠ baa

aaab (aaba, $a^4b^3b^4a^3$
→ a^3b, → a^2ba, → $a^4b^7a^3$ (it's based on sequence as b^3 & b^4 are consequitive then we can write this)

If 'ε' is given we can find the sequence from that //

language – collection of strings makes language, over some rules.

L1: All the strings starting with 'a' where $\Sigma = \{a, b\}$ ~~(ba)~~

$= \{a, a^2, a^3, ab, aab, abb, \dots \infty\}$

• i.e Language is somehow subset of total no. of strings possible, i.e, $L \subseteq \Sigma^*$

L2: All strings ending with 'ab', where $\Sigma = \{a, b\}$

$= \{ab, aab, a^2ab, a^3ab, bab, b^2ab, b^3ab, \dots \infty\}$

L3: All strings of length exactly 3 where $\Sigma = \{a, b\}$

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

L4: All string where ~~there~~ no. of a is divisiable by 2 $\Sigma = \{a, b\}$ $(b^*(\epsilon + a^2 + a^4 + \dots + \infty) b^* +$
[∵ a^1ba^3, (a^4) ; a^3ba^5 (a^8)] $(\epsilon + a^2 + a^4 + \dots) b^* (\epsilon + a^2 + a^4 + \dots)$
$+ (\epsilon + a^1 + a^3 + a^5 \dots) b^* (\epsilon + a^1 + a^3 + \dots)$

~~$\{\epsilon, a^2 b\}$~~
$= \{\epsilon, b, a^2b, a^4b, a^8b, \dots \infty\}$ $b^3, b^4, b^6 \dots$
aba, abaa. ...

L5: All string whose binary converted desimal value is divisiable by 5 where $\Sigma = \{0, 1\}$

$= \{\epsilon, 0, 00, \dots 101, 1010, 1111 \dots \infty\}$
[As the length/val of $\epsilon = 0$]

• The no. of strings in a language is called it's cardinality.

|L| = cardinality → no. of strings in lang.

25/7/23

CW

• Σ^0 means what are the strings possible of length '0' using Σ, $= \epsilon$
• Σ^1,
• $\Sigma^2 = \{aa, bb, ab, ba\}$
:
:

∵ If $\Sigma = \{a, b\}$

$\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \dots = \{\epsilon, a, b, a^2, b^2, ab, ba \dots \infty\}$
(All possible strings using Σ)
||
can be represented as → Σ^* = By using Σ what are the strings possible / various strings possible using Σ.

• Σ^* is called Kleene closure.

- $a^* = a^0 \cup a^1 \cup a^2 \cdots \infty$ = all possible strings using a; length of strings
$= \{\epsilon, a, a^2 \cdots \infty\}$

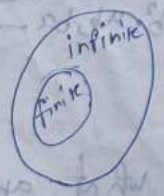
- ~~$(a+b)^*$~~ a^+ = positive closure = All possible strings using a but excluding the 0 length string $= a^1 \cup a^2 \cup a^3 \cdots \infty$
$= \{a, b, a^2, a^3, b^2 \cdots \infty\}$

- $\Sigma^* = \{a, b\}$
$\Sigma^+ = \{a, b, a^2, b^2 \cdots \infty\}$

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

- $\Sigma^* = \{\epsilon, a, b, c, a^2, b^2, c^2, ab, abc, bc, ca, \cdots \infty\}$

- By using Σ the total no. of langs possible is $\boxed{2^{|\Sigma^*|}}$ [Any lang is a subset of Σ^* (Power set concept), if $|\Sigma^*| = n$, then the total no. of langs possible $= 2^n$]

- 

$2^{|\Sigma^*|}$

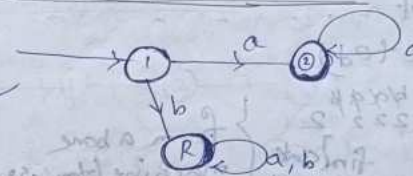
long. length $\leq 10^{10^{10}}$ ~~infinite~~ finite lang.
long length $\geq 10^{10^{10}}$ ~~infinite~~

L_1 = ~~*~~ (all) string starting with 'a'; $\Sigma = \{a, b\}$
$= \{a, ab, a^2b, a^3b \cdots \infty\}$

From YX

If you want to search an particular element in b/w the infinite no. array that contains a no of elements you will always get an ans NOT FIND!

finite automata (machine)



- There are 5 tuples in finite automata-
$(Q, \Sigma, \delta, q_s, f)$
  - Q: finite set of states
  - δ: transition function
  - q_s: starting state
  - f: set of final states

$Q = \{1, 2, R\}$ (no. of circles)

$\Sigma = \{a, b\}$ (the i/ps above transitions)

the arrow means transition

$q_s = 1 \subseteq Q$ (the state before which there is an arrow without q i/p)

$f = \{2\}$ (the state that has double ~~states~~ circles)

only one finite automata in any finite

There is only 1 ~~finite automata~~ starting state in every finite automata,

(more than 1 final state is also possible)

- Our starting state can also be final.

- whatever the strings that are possible for it should be accepted by that 'fa' machine. (Accepted)

not (Rejected)

Accepted → 1 2 2 2 2 2 → final state accepted

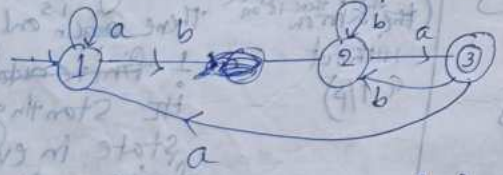
Rejected → 1 R R R → not final state → Rejected

from a bone machine/transition diagram or finite automata

- one transition contain only 1 i/p or more.
- R → intermediate state (Rejected state)
- self loops can contain more than 1 i/ps.

$\Sigma = \{P\}$

$= \{\epsilon, P^2, P^3 \dots \infty\}$



$a^3 b^2 a$ → accepted

$a^3 b^2$ ← Rejected

aaabba

1 2 3 1 1 1 2 3

For $a^3 = 1, b^2 = 2$ it $a^4 = 1, b^4 = 2$ also.

All strings ending with 'ba'

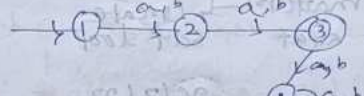
∴ This is it's language (by accepted string we can say that, this 'fa' is language above)

4: All strings with length exactly 2, $\Sigma = \{a, b\}$

$= \{a^2, b^2, ab, ba\}$

There must be an ∞ no. of strings but finite automata give finite O/P always that is the power of it

Srideton method



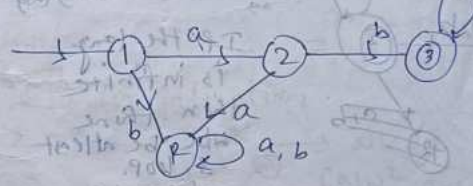
For length = 1 States = 2
length = 2 states = 3
As one state is free

While drawing fa, 1st complete pro smallest string & then consider the not possible string & then

L2: All strings Starting with ab, $\Sigma = \{a, b\}$

$= \{ab, aab, aba^2, aba^3 \dots \infty\}$; $= ab\Sigma^*$

$= ab(\epsilon, a, b, aa, ab \dots)$; $= ab, aba \dots$ or $ab(a+b)^*$

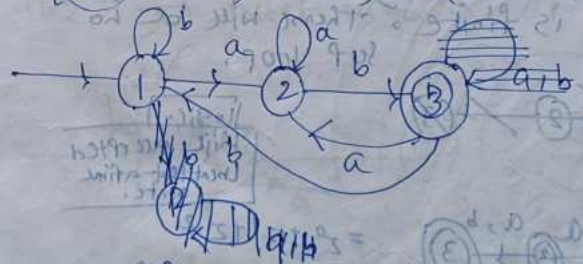


There should be 1 R State or 0. But not more than 1.

Start building from smallest string

L3: All strings ending with 'ab', $\Sigma = \{a, b\}$

$= \{(a+b)^*ab, aaab, a^2ab, a^3ab \dots \infty\}$; $= \Sigma^*ab$ or $(a+b)^*ab$



Rejected state may come, may not be come

- Start from the smallest string of that language & 1st build srideton then add logic.

i) going to the result → Create new state

ii) not going to the result → create rejected state (R)

iii) If the i/p is a or b or whatever but it doesn't matter → create self loop

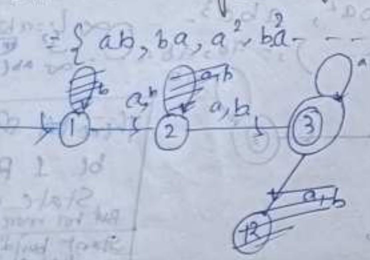
(or) If the i/p string doesn't affect your result

26/7/23

CW

L1: All strings whose length at least 2, $\Sigma = \{a,b\}$

$= \{ab, ba, a^2, ba^2 \dots \infty\}$ (unrestricted) $= \Sigma^2\Sigma^*$ or $(a+b)^2(a+b)^*$

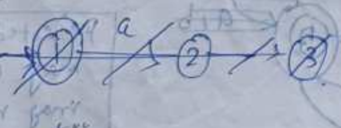


If the lang is infinite then there will be atleast 1 loop.

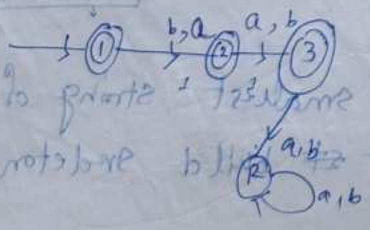
L2: All strings whose length at most 2, $\Sigma = \{a,b\}$

$= \{\epsilon, a, b, a^2, b^2, ab, ba\}$ → (restricted is finite)

If the lang is finite there will be no self loop.



logic - while accepted check for a final state.



$= \Sigma^0 + \Sigma^1 + \Sigma^2$ (or) $\epsilon + (a+b) + (a+b)^2$

- length atleast n, total no. of states = n+1
- ... atmost n, total no. of states = (n+1)+1 → reject state
- ... exact n ... (n+1)+1 → reject state

Substring

abba → ϵ, a, ab, abb, bba, ba, b, bb

L3: For all strings, substring, $\Sigma = \{a, b\}$ having 'aa' as a substring (ex, here)

$= \{aa, aaa, a^2aa, a^3aa, baa, b^2aa,$ baab ... $\}$

the substring can also be a string itself

or $= \Sigma^* aa \Sigma^*$ (or) $(a+b)^* aa (a+b)^*$

- $ab\Sigma^*$ → concatination
- $\Sigma\Sigma\Sigma^* = \Sigma^2\Sigma^*$ → atleast 2 length (length 2) (or) $(a+b)^2 (a+b)^*$
- $\Sigma^0 + \Sigma^1 + \Sigma^2$ → atmost 2
- $\Sigma^* ab$ → will have no self loop at end
- $ab\Sigma^*$ → must have a self loop at end

my requirement is min 2 aa so from ③ → ①

deterministic finite automata (DFA)

↓

minimized DFA = the lesser no. of states required

L4: All strings ~~are~~ of count of a, ∧ will be equal to the count of b where $\Sigma = \{a, b\}$ [abba ✓, abbx sequence doesn't matter]

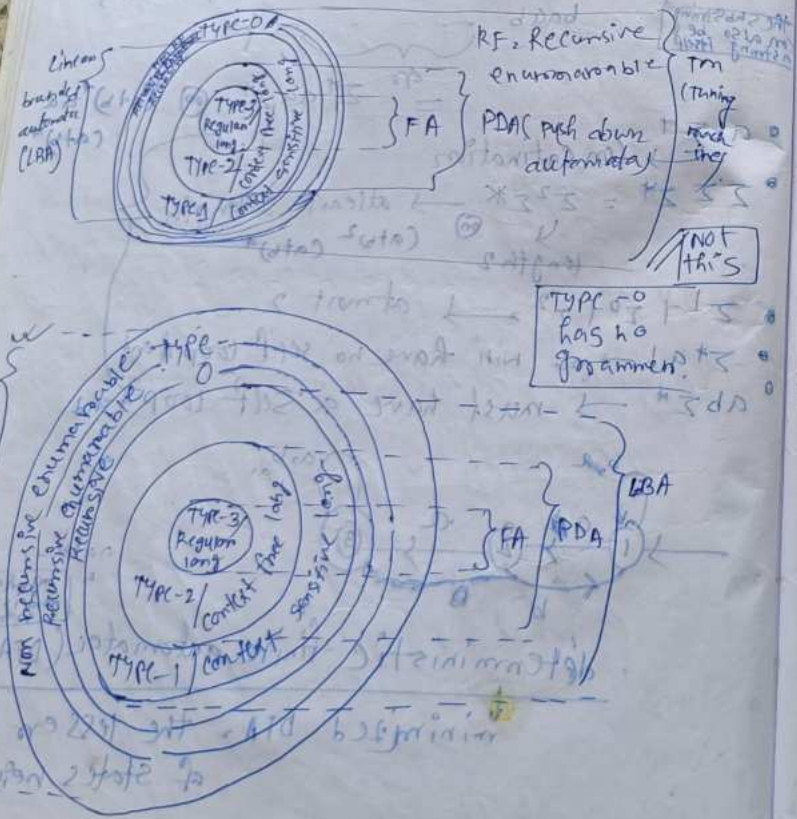
$\{n_a = n_b, \Sigma = \{a, b\}\}$

n_a = count of 'a'

this isn't a Regular lang. For this FA can't be constructed.

[$L = \{a^n b^n \mid n \geq 1\}$, $\Sigma = \{a, b\}$, a^2b^2 ✓, abba ✗]

Chomsky's classification → (that says types of langs)



Regular lang – the lang for which ~~this~~ a ~~last~~ FA exists.

* L is regular automata iff there exist a FA for that.
* If my lang is finite, then we can always construct finite automata, ie. that lang is always regular [that means, all finite langs are regular]. But in some cases also ~~is~~ exist for infinite lang., these are also regular lang.
* Type 2, Type-1, Type-0 all are infinite lang. [TYPE-3 ~~can~~ consists finite langs, but it can mainly also consist some infinite langs]

Assignment

① Construct a finite automata (minimized DFA) for the lang. below –

L1: All the strings where 2nd last char will be a. (Ex – aab, aa [a^2]) (Regular exp – $(a+b)^* a (a+b)$)

L2: All the strings where 2nd last char will be b. (Ex – $\Sigma^* b aa$, $\Sigma^* bab$, $\Sigma^* bbb$ ($\Sigma^* b \Sigma^2$)

(Regular exp – $(a+b)^* b (a+b)(a+b)$) → 1/8/25

① L: All strings where 2nd char will be b

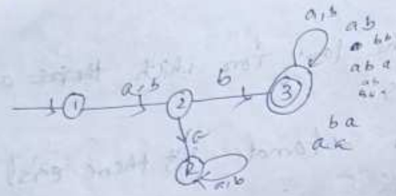
$\Sigma = \{a, b\}$

$= \{ab, a^2ba, abb, aba^2, abb^2, \ldots\}$

, $b\Sigma^*$

or $(a+b)\, b\, [(a+b)^*]$ → any combination of a, b

anything from a or b

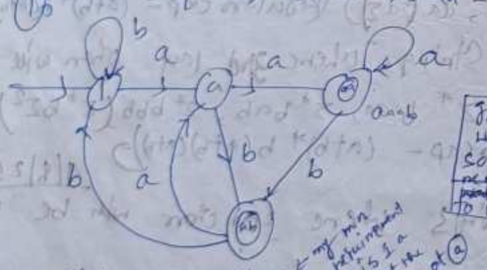


- $(a+b)^* \, b \, (a+b)^* \longrightarrow$ b is the substring
  $(\Sigma^* \, b \, \Sigma^*)$
  any combination of (a, b)

Remember 2 -
create new state
self loop ~ waiting

Assignment

(1) = { aa, ab, bab, b^2aa ... ∞ }



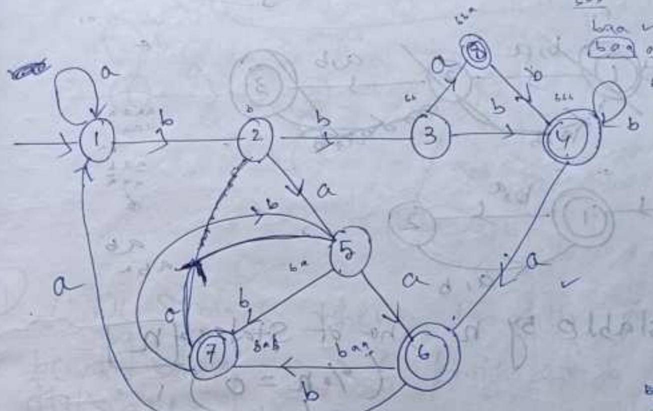
1st abb - but with not perfect, so we have to remember means to go back to 1st

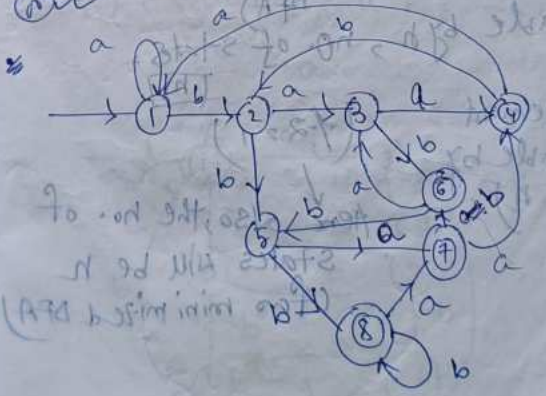
- Last from last nth position, so the no. of total states = 2^n
- From starting nth position, so, no. of total states = (n+1)

(2) $\Sigma = \{a, b\}$

= { bbb, baa, bab, bba, ~~ba^3, ba^4, bb^5~~, ~~babba~~, a^3bbb, a^5bba, b^5b^3 ... ∞ }

$= \Sigma^* \, b \, \Sigma^2 \Rightarrow (a+b)^* \, b \, (a+b)^2$



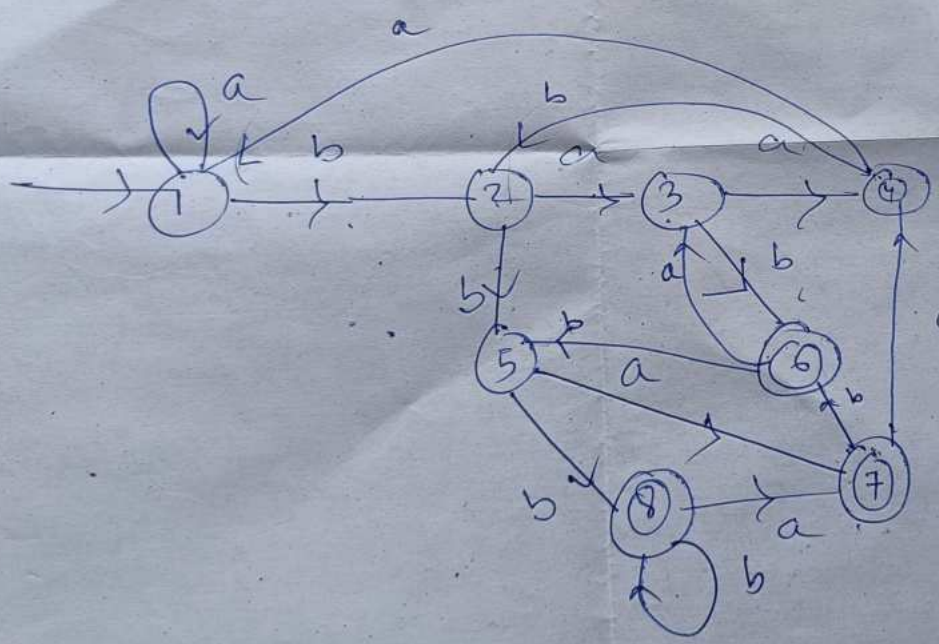


Smallest string a or b → this check is done for once for every smallest string is enough after that you can check more after 'a' or 'b' but they will be done easily

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

$\Sigma^* b \Sigma 2$

$= (a+b)^* b (a+b) 2$



required one a or b if you done any one from for one a or one b with the another will be also done

bbb

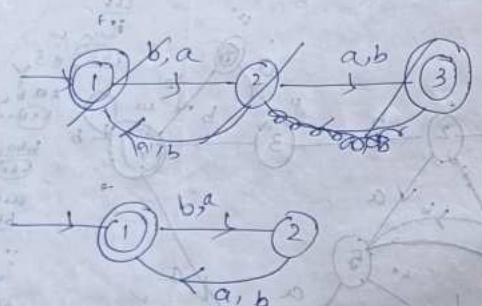
baa

bba

for every smallest string is enough after

② L: All strings where length is divisible by 2, $\Sigma = \{a, b\}$.

$= \{\epsilon, a^2, b^2, ab, ba, a^4, aaab, bbba \dots \infty\}$

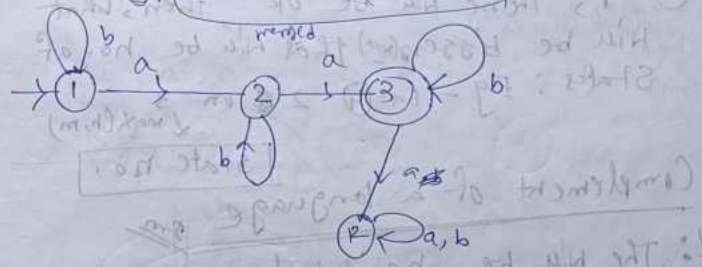


- divisible by n, no of states = n ($\%n = 0$)
  ↓ no. of states (for minimized DFA)
- Not divisible by n, no. of states = n
  ↓ (as it is the complement of divisible by n)
  ($\%2 = 1$)
  ↓ Here also, the no. of states will be n (for minimized DFA)

③ L: All strings where no. of a will exactly 2, $\Sigma = \{a, b\}$

$= \{aa, aab, aab^2 \dots \infty\}$

$= aab^*, b^*aa, ab^*a, b^*a^2b^*$



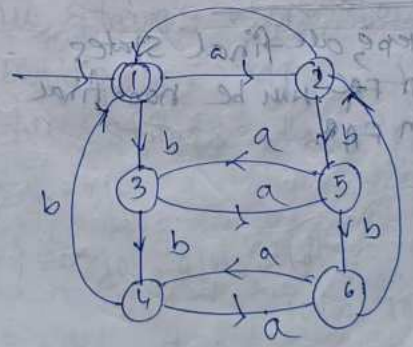
④ L: All strings where length of a will be divisible by 2 & length of b will be divisible by 3, $\Sigma = \{a, b\}$

Regex $= (\epsilon + a^2 + a^4 + \dots \infty) \cdot (\epsilon + b^3 + b^6 + \dots \infty)$

$= \{\epsilon, aabbb, a^4b^6 \dots \infty\}$

$a^2, b^3,$

- total 6 states $2 * 3$ $(n * m)$



← for AND

a^2 done

b^3 done

now do rest of states symmetric

(5) L: All strings where length of a is atleast '3' & length of b is almost '4'. $\Sigma = \{a, b\}$

$(n+1) + ((n+1)+1)$

$3+4+6$

$= 24$

(6) If, there will be 'OR' then, what will be biggest(num) that will be no. of states: Eg - Ex-(4) 2 & 3 on $\max(n, m)$

State no.

Complement of a language

L: The will be no 'b' at 2nd positions. All strings), $\Sigma = \{a, b\}$

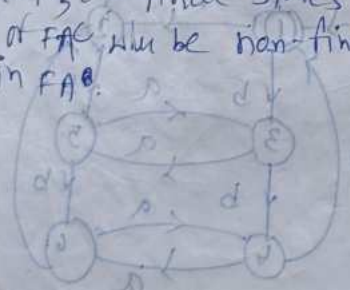
L^c: All strings where ~~no~~ 'b' is in 2nd pos.

When there will be a concept "no"/"not" = then we will go for L^c

How to construct the complementary automata of finite automata

$L \longrightarrow FA$ → Here, all final states of FA will be non-final in FA@.

$L^c \longrightarrow FA^c$



(1) L: All strings where no. of 'a' is not divisible by '2'.

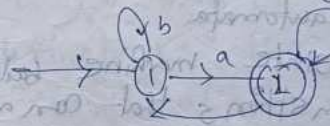
L': All strings where no. of 'a' is divisible by '2'. * reg ex = b^*(no of a div by 2) b^* + (∈ + a + aa + ... ∞)

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

= { ∈, a^2, a^4 .... b^2, a^2b .... ∞ }

(no of a div by 2) b^* (no of a div by 2) ∪ (rev)

⇓ Complement of this



Assign

(3) All strings where '101' will not be a sub string, $\Sigma = \{0, 1\}$.

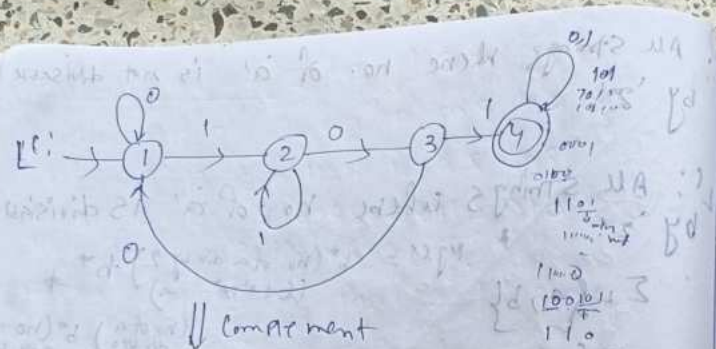
L: All strings where '101' will not be a substring, $\Sigma = \{0, 1\}$

L^c: All strings where '101' will be a sub string.

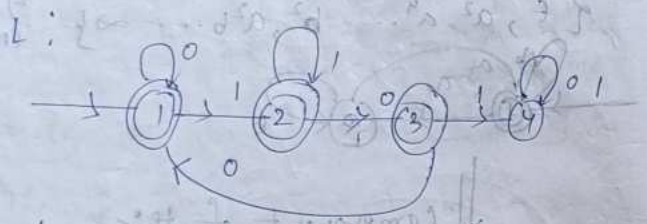
$\Sigma = \{0, 1\}$

= { ~~000~~ 101, 0101, 0^2101, ... 01101, 10101, 10100, 101111, 1^2101 .... ∞ }

= $\Sigma^* 101 \Sigma^*$



∥ Complement



~~Formal language~~ — In automata

DFA — A DFA is a state machine built up of states & transitions that can accept or reject a finite string made up of a series of symbols and evaluate it to a predefined language across a predefined set of characters. We represent states with circles & transitions with directed arrows. Every state must have each symbol radiating from it, or it will not be characterized as a DFA. The machine is known as a deterministic finite machine since it has a finite no. of states. It is also called as minimized DFA as have the lesser no. of states are require to draw the machine. A FA is called DFA if lesser no. of states are required to draw the machine.

Formal language — In automata theory, a formal language is a set of strings of symbols drawn from a finite alphabet. A formal lang. can be specified either by a set of rules that generates the langua-age, or by a formal machine that accepts the language.

String — A string is a finite sequence of symbols selected from some alphabet. It generally denoted as w e.g. for alphabet $\Sigma = \{0,1\}$, $w = 010101$ is a string. length of a string is denoted as $|w|$ and is defined as the numbers of positions for the symbol in the string. For the above example length is 6.

language — A language is a set of strings from some alphabet (finite or infinite). In other words, any subset L of Σ^* is a language in toc.

Finite automata — it is a simple idealized machine used to recognize patterns within input taken from some character set (or alphabet) C. The job of an FA is to accept or reject an input depending on whether the pattern defined by the FA occurs in the input. [a automata with help of which we can build a reg. lang.

Skeleton method - In this method you put all known factors into skeleton file. Then you use it as base for most of the subassemblies and parts that make up the assembly.

Substring - It is any part of the string.

Regular language - [It is a language that can be expressed with a regular expression or a deterministic or non-deterministic finite automata or state machine.] //

A language is said to be a regular language if & only if some finite state machine / FA recognize it.

Complement of a language - The complement of a language is defined in terms of set difference from Σ^*, that is $L' = \Sigma^* - L$. And the complement language (L') of L has all strings from Σ^* except the strings in L. Σ^* is all possible strings over the alphabet Σ.

Finite & Infinite language - A finite language is any set L of strings of finite cardinality $|L| < \infty$.

An infinite language is any set L of strings of infinite cardinality $|L| = \infty$. (1)

Ch 2

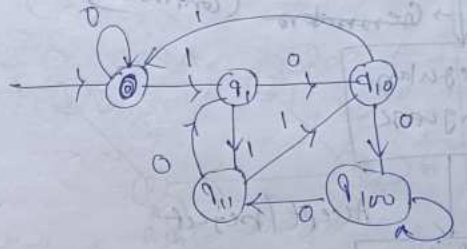
2/3/23

(1) Prof

L: All strings in which all ~~converted~~ decimal converted values / whose decimal converted value will be divisible by 5, $\Sigma = \{0, 1\}$

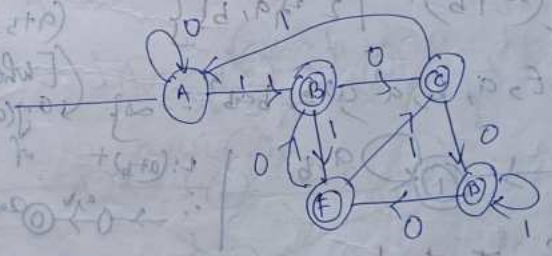
Sol:

- $\Sigma = \{0, 1\}$
- $= \{0, 00, 000 \dots 101, 1010, 1111 \dots \infty\}$



$q_1, q_{10}, q_{11}, q_{100}$ Here we are considering mod values. Here we have to remember all states separately as 100 & 101 have different meaning

(2) L: All strings, whose decimal, which is not divisible by 5, $\Sigma = \{0, 1\}$ (Complement of (1))



(3) L_1, $7n+2$ | $\Sigma = \{1\}$, $n \geq 0$

Smallest string $= 1^2 = 11$

$= \{1^2, 1^9, \dots \infty\}$

Converted values / whose decimal [illegible]
value will be divisible by 5, $\Sigma = \{0, 1\}$

Ans

- $\Sigma = \{0, 1\}$
  - $\{0, 00, 000 \ldots 101, 1010, 1111 \ldots\}$

Divisible by 5

Remainders lying range = 0 to divisor-1
= 0 to (5-1)
= 0 to 4

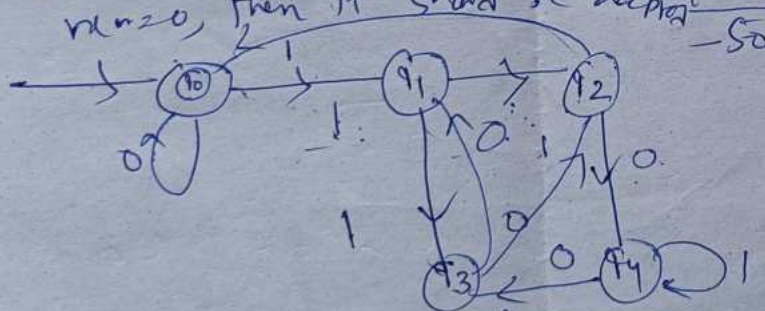
$q_0 \rightarrow$ rem 0
$q_1 \rightarrow$ rem 1
$q_2 \rightarrow$ rem 2
$q_3 \rightarrow$ rem 3
$q_4 \rightarrow$ rem 4

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

$5\%5 = 0$
$6\%5 = 1$
$7\%5 = 2$
$8\%5 = 3$
$9\%5 = 4$
$10\%5 = 0$

So rem 0 is final state

divisible by 5, means rem = 0, then it should be accepted



i/p = 101	rem $5\%5 = 0$
i/p = 110	rem $6\%5 = 1$
i/p = 111	rem $7\%5 = 2$

i/p = 1000	$8\%5 = 3$
i/p = 1001	$9\%5 = 4$
1010	0

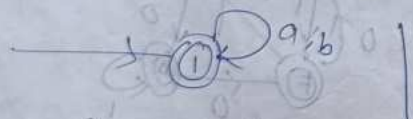
i/p = 0	rem = $0\%5 = 0$
i/p = 1	rem = $1\%5 = 1$
i/p = 10	rem = $2\%5 = 2$
i/p = 11	rem $3\%5 = 3$
i/p = 100	rem $4\%5 = 4$

(3) Regular Grammar → Generators (controllers) → Regular language → Regular Expression (represented by) → Finite automata (Representative) ← Accept / Reject

(4) $L = \{(a+b)^* \mid \Sigma = \{a, b\}\}$

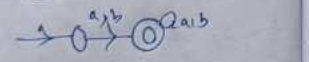
$= \{\epsilon, a, b, a^2, ab, ba, b^2 \dots \infty\}$

$(a+b)^* = \Sigma^*$ [where $\Sigma = \{a, b\}$ any combination of a, b]



$L = (a+b)^+$

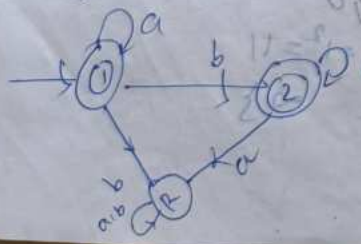
$\therefore$



(5) $L = \{a^* b^* \mid \Sigma = \{a, b\}\}$

$= \{\epsilon, a, b, ab, aa, aabbb \dots \infty\}$

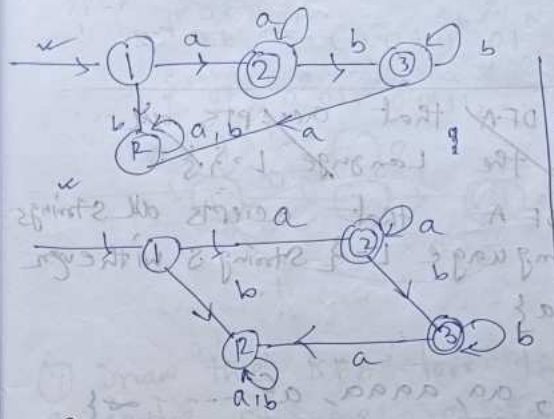
~~Smallest string = ε??~~

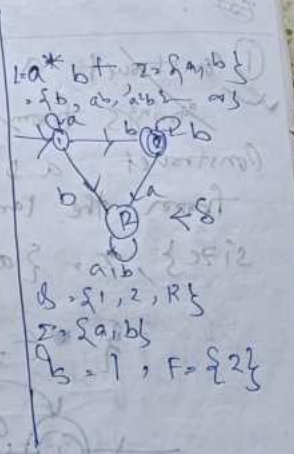


(3) $L = \{a^+ b^* \mid \Sigma = \{a, b\}\}$

$= \{a, ab, aa \dots a^2 b^3 \dots \infty\}$

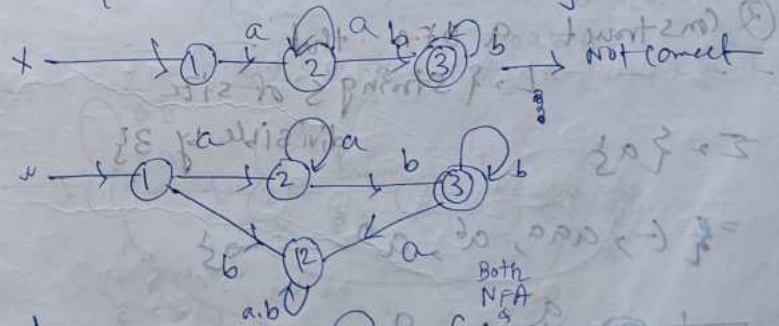
$a^+ = \Sigma^*$ except ε where $\Sigma = \{a\}$

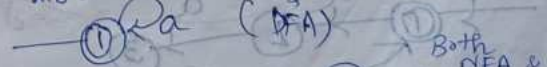
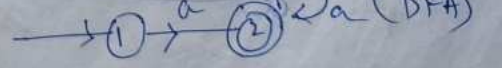




(5) $L = \{a^+ b^+ \mid \Sigma = \{a, b\}\}$

$= \{ab, a^2 b^2, ab^2 \dots \infty\}$



- a^+ →  Both NFA & DFA
- a^+ →  Both NFA & DFA

Assign

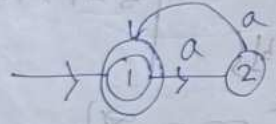
Finite Automata, language search problems from net & soln

DFA - Learn

(1) Construct a DFA that accepts all strings from the Language L = {}

Construct a DFA that accepts all strings from the language L = { strings with even size }, Σ = {a}

= { ε, aa, aaaa, a^6, ... ∞ }

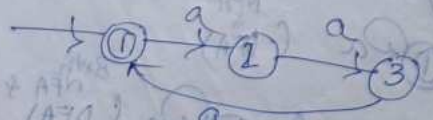


(2) Construct a DFA that

L = { strings of size divisible by 3 }

Σ = {a}

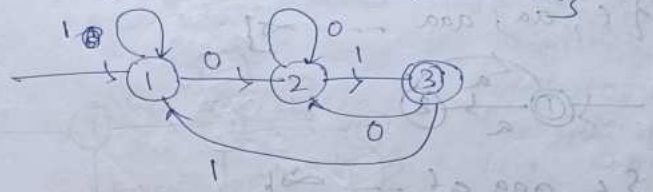
= { ε, aaa, a^6, a^9, ... ∞ }



(3) Draw the DFA for the language accepting strings ending with '01' over input alphabet Σ = {0, 1}

(a+b)* or (a+0+1)* 01

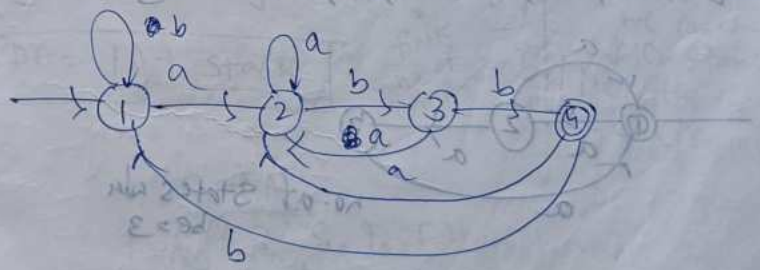
= { 01, 0^2 01, 0^7 0111, ... ∞ }



(4) Draw the DFA for the language accepting strings ε strings ending with 'abb' over input alphabet Σ = {a, b}

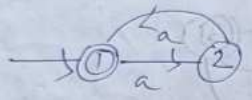
= { abb, a^2 abb, b^2 abb, aaabb, ... ∞ }

= (a+b)* abb

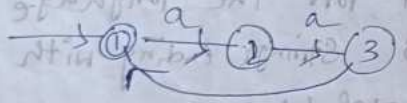


⑤ Construct a DFA that accept the strings from the language L = {strings with size multiples of 2 or 3}, $\Sigma = \{a\}$

$L1 = \{\epsilon, aa, aaa \dots \infty\}$

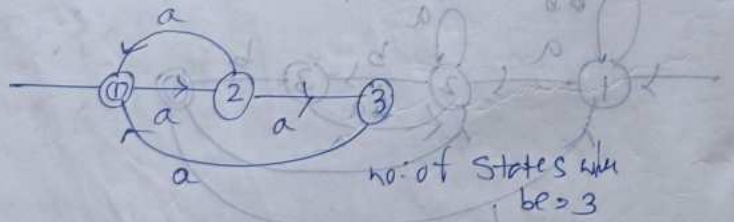


$L2 = \{\epsilon, aaa, a^6 \dots \infty\}$



~~L1 ∪ L2~~ L1 ∪ L2 (3 states)

$= \{\epsilon, aa, aaa, a^4, a^5, a^8, a^7 \dots \infty\}$



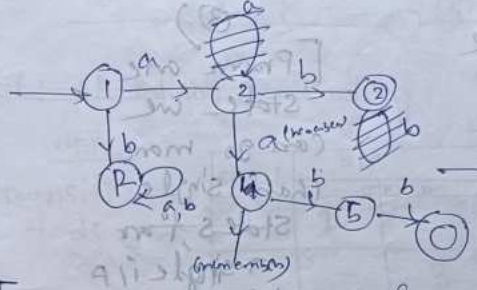
Not too needed

CM — infinite level of comparison — 8/8/23

$L = \{a^n b^n \mid n \geq 1, \Sigma = \{a, b\}\}$ → can't construct finite automata

$L = \{a^* b^* \mid \Sigma = \{a, b\}\}$ → can construct finite automata

$= \{ab, a^2b^2 \dots \infty\}$ [If the counting is infinite then we can't construct a fa.]



[As we have to remember how much a is coming & how much b is coming but we can't know how we can remember, so we can't construct i.e the comparison is infinite]

[We can construct fa when the counting is ~~infinite~~ but it's comparison is infinite (e.g div by 2 & 3)]

finite no of comparison

PDA = FA + stack

FA $(Q, \Sigma, \delta, I_s, F)$

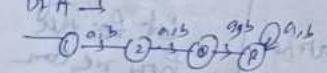
- Without output
  - DFA
  - NFA (Nondeterministic fa)
  - ∈-NFA
- With output
  - moore machine
  - mealy machine

DFA

① (It has single choices)
[From one state we can go single state, for a single i/p we can go to a single state] (DFA)

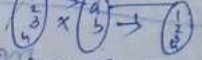
a a b
1 - 2

DFA →



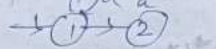
② ~~When~~ what will ~~be~~ happen for each and every states will be given / written.

③ $\delta: Q \times \Sigma \to Q$



[~~In one~~ ~~State we go~~ ~~applying~~ For ~~each~~ any

NFA



① (~~That~~ It has more than 1 options to choose ① or ②)

[From one state we can go more than single states for a single i/p we can go to multiple states (NFA)]

a a b
1 → 2 | 1 → 1, 2 | 2 → 3

② what will ~~be~~ happen for each & every states will not be given here / written here

② $\delta: Q \times \Sigma \to 2^Q$

$\{\phi, \{1\}, \{2\}, \ldots\}$ subsets possibly

①

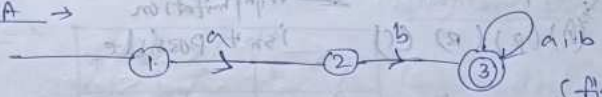
NFA to DFA Conversion ex.

① L: All strings starting with 'ab'. Construct NFA.

= { ab, aba, abb ... ∞ }

→ ab Σ* or ~~Σ~~ ab (a+b)*

NFA →



(figure isn't needed) (not so necessary)

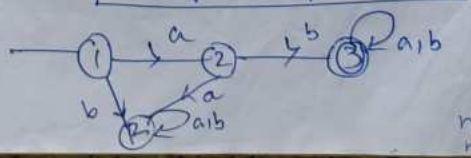
~~② L: All strings ending~~

NFA transition table →

States ↓	i/p a	b
→ 1	2	–
2	–	3
* 3	3	3

DFA transition table →

States ↓	a	b
→ {1}	{2}	R
{2}	R	{3}
* {3}	{3}	{3}
R	R	R



[{1,2} → not 1, 2 i/p]

→ [In DFA transition table one Scope can hold only 1 i/p, if it's more than 1 then it's not DFA]

(AB CR)

(AR) (C)

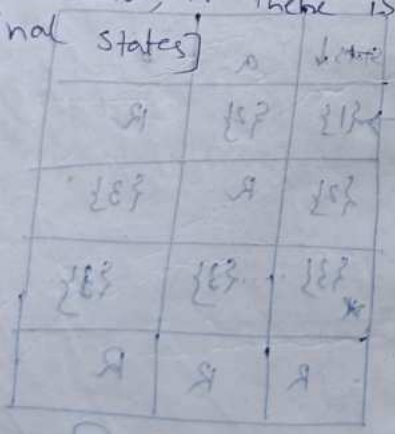
(AR) (B) (C)

(A) (R) (B) (C)

[A's & R's pattern is in a set ... in a set]

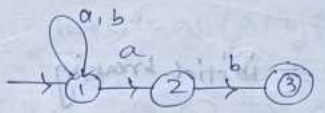
← minimization isn't possible

Here there is 1 final state, so we can't write that it is minimized already. If there is more the 1 final states then we have to understand this can be minimized, so we have to perform minimization. [this may be minimized / may not be but we have to perform minimization, if there is more than 1 final states]



② L: All strings ending with 'ab', Σ = {a, b}

= (a+b)*ab



Done

NFA →

states	a	b
1	{1, 2}	1
2		3
3		

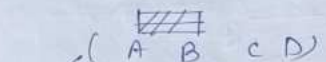
[for every conversion cases, make 2 to 3 table, if NFA → DFA → make 2 tables (NFA, DFA (into it rename)) [Already minimized DFA]

→ make 3 tables (NFA, DFA (into it rename), minimized table)

if (- NFA → DFA → make 1 table (DFA (into it rename)) [DFA is already minimized]

→ make 2 tables (DFA (into it rename), minimized DFA)

(2) minimization of this fa



= (A B) (C D) ← initial breaking
(non final) (final)

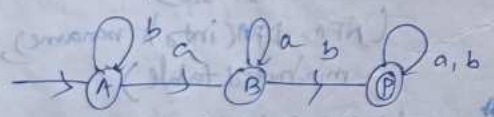
= (A) (B) (C D)

= (A) (B) (P)

← B, C are at same states so we have to break
[CD → single state as it's not breakable]

NOW table,
DFA (minimized)

State ↓	a	b
→ A	B	A
B	B	P
* P	P	P



↑ minimized DFA

(3) L: All strings where 'ab' is a substring.

Imp

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

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

[At NFA the no. of states ≤ no. of states at DFA]

[As we are not not writing each states (i.e. if I get a* at first, b* at end, this kind of things & also here there is no restricted state)

NFA →

States ↓	o/p	
	a	b
→ 1	1, 2	1
2	—	3
* 3	3	3

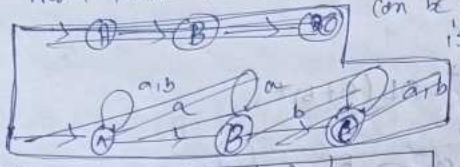
DFA →

States ↓	a	b
→ {1} A	{1,2} B	{1} A
{1,2} B	{1,2} B	{1,3} C
* {1,3} C	{1,2,3} D	{1,3} C
* {1,2,3} D	{1,2,3} D	{1,3} C

[join 1, 2 = {1,2}, {1,2,3} join 1,3,3 = {1,3}]

[a over both {1,2} b over both {1,2} like that way]

[the pattern of 2 final states at is same, from there we can understand 2 final state can be 1 as the pattern is same so we have to perform minimization (by this logic we can understand when minimization is needed)]

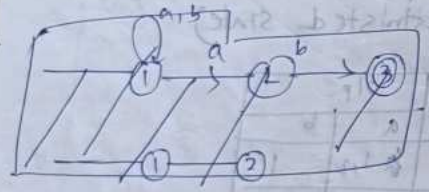


Rename →

states	a	b
→A	B	A
B	B	C
*C	D	C
*D	D	C

[output pattern is also same so both are final states]

PTO

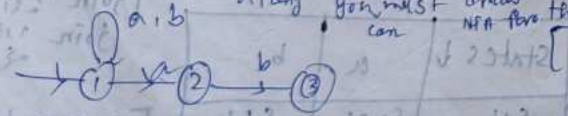


* a/b ← ATM

• power of any FA is same type of

[power = How many langs it covers]

[For how many langs FA exists (if you can draw DFA for along you must can draw NFA for that] → [But this is not possible for every machines]



(transition diagram)

[FA is basically a t.d, but it can also be represented in transition table]

Σ_1 = {1, 2, 3}

Σ_2 = {a, b}

δ :

To represent a δ there are 3 techniques:

→ transition diagram
→ " table
→ " function

states	a	b
→1	1,2	1
2	-	3
*3	-	-

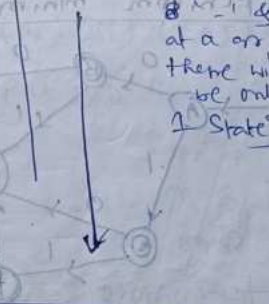
[→ = starting state, * = final state]

δ(1, a) = 1, 2
δ(1, b) = 1
δ(2, b) = 3

} not so need

1 is starting state & 3 is final state

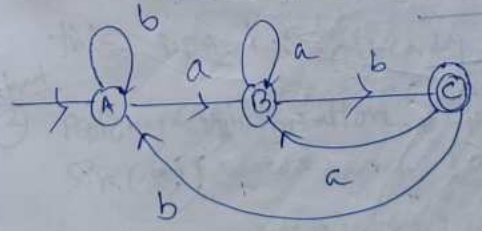
(NFA) [at DFA it will be like no matter if at a or b there will be only 1 state]



states	a	b
→{1} A	{1,2} B	{1} A
{1,2} B	{1,2} B	{1,3} C
{1,3} C	{1,2} B	{1} A
*		

[{1,2} → new state]

[the state that will carry the final state at NFA will be final state at DFA]



[For state {1,2} if we apply a on both we can get {1,2} & if we see in NFA the sum of state 1 & 2 over a is 1,2 & b is 1,3 & it is matching with DFA's {1,2} state]

Assignment → finish all assignments & submit

9/8/23

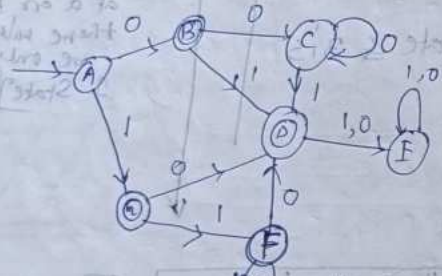
CK

Assignment

① Construct DFA that aren't constructed by us (DFA that we can't construct) (Search from net & note it down & try to sol; if u can't leave it as it is).

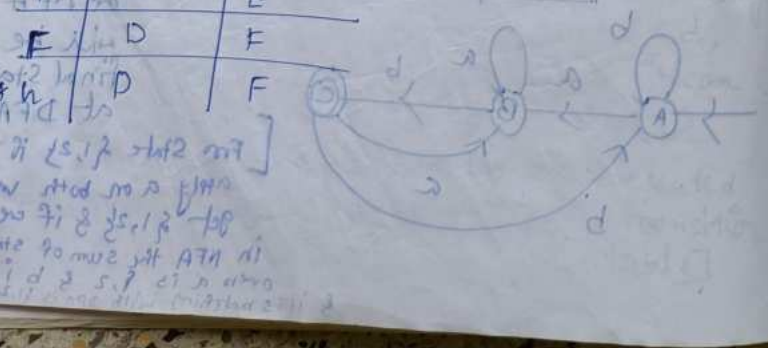
(Quantitative examples fns)

① Perform minimization



tell is it minimized or not?

	0	1
A	B	C
B	C	D
C	C	D
D	E	E
E	E	E
F	D	F
D	D	F



(ABCDEFG)

→ (AD)

= (AC EF) (BDG)

= (AE) (CF) (BDG) (A & F same set)

= (AE) (CF) (BG) (D) (E & F's pattern are same)

= (AE) (CF) (BG) (D) [D's pattern is different]

(*) = P Q R D

after minimizing we get = (AE) (CF) (BG) (D)

After renaming we get = P Q R D

	0	1
→ P	R	R
* R	Q	D/D
Q	Q	D
* D	P	P
P	P	P
Q	D	Q
* R	D	Q (new)

AE = P
CF = Q
BG = R

[Replacement should be same, otherwise you need to break further]

(*) = (AE) (CF) (BG) (D)

= (A) (E) (C) (F) (B) (G) (D)

[CF isn't in same set]

this DFA is already minimized

[Here, though the pattern of 'BG', 'DE', are same but of 2 sides only 2 (send by the pattern is actually not same if final & non final state, if we have to remove we will remove whichever!]

Assignment ② Perform minimization for Sin (SS)

** Though it is not applicable here, but can be applicable for others cases, pattern isn't same during the partition of minimization.

Regular expression — The mechanism to represent a regular language:

- if is a RE.

(1) $\Sigma = \{a\}$, a is a RE.

(2) $\Sigma = \{a, b\}$, a.b is RE. ↳ concatenation

(3) $\Sigma = \{a, b\}$, a+b is RE ↳ union (i.e either or)

4) $\Sigma = \{a\}$, Kleene closure, a^* is RE

5) $\Sigma = \{a\}$, positive closure, a^+ is RE.

- $a^+ = aa^* = a^*a$
- $(a^* + b^*)^* = (a+b)^*$
- length atleast 2, $\Sigma = \{a\}$ (infinite lang) — R.E
  $[a^2a^*$ or $aa^+ \mid (a+b)^2(a+b)^*$ or $(a+b)(a+b)^+ \leftarrow$ aa a* or aa+
  $[\because$ fm Kleene closure, $\Sigma^n \Sigma^* = \Sigma^{n-1}\Sigma^+]$
- , $\Sigma = \{a, b\}$ (infinite)
  $= (a+b)(a+b)(a+b)^*$
- exactly 2, $\Sigma = \{a, b\}$ (finite lang)
  $= (a+b)(a+b)$
  $= (aa + ab + bb + ba)$
- atmost 2, $\Sigma = \{a, b\}$
  $= \{\epsilon, a, b, aa, ab, ba, bb\}$
  $R.E = (\epsilon + a + b + aa + ab + ba + bb)$
  $= (\epsilon + (a+b) + (a+b)(a+b))$

[+ → OR, . → AND (concatenate)]

- Substring ab → R.E = $(a+b)^* ab (a+b)^*$
- len is divisible by 2 = $\Sigma = \{a, b\} = (\epsilon + (a+b)^2 + (a+b)^4 + \cdots$ (All strings)

① L: Starting & ending with same char, $\Sigma = \{a, b\}$

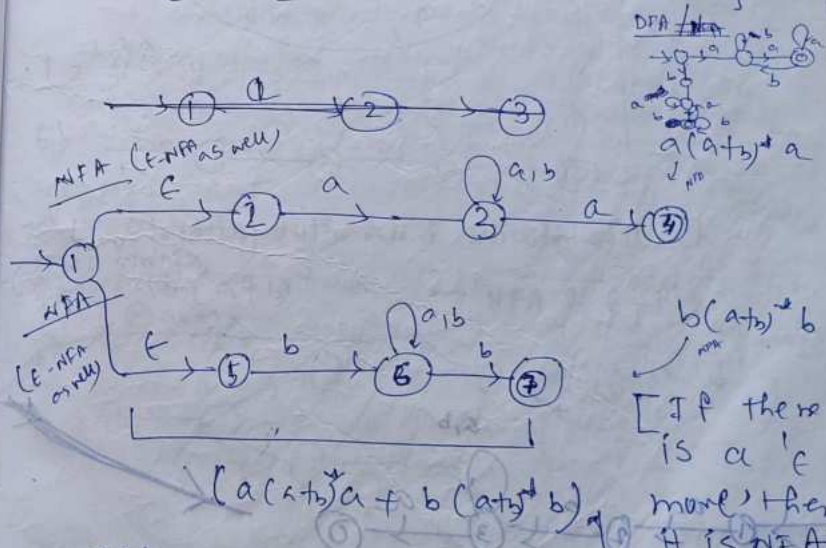
ε-NFA = ~~$a(a+b)^*a$~~

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

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

(from this NFA ≈ ε-NFA) (Same as dfa but not least (bcz of bb))

$= \{aa, bb, ab^2a, aa^3a \cdots\}$



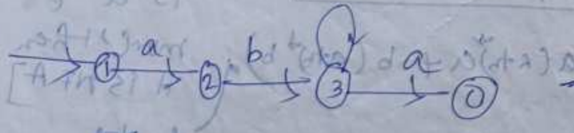
[If there is a 'ε' more' then A is NFA]

ε-NFA

aaa
1 2 3 4
↓
no i/p
but we are moving towards (2) → so it is non deterministic

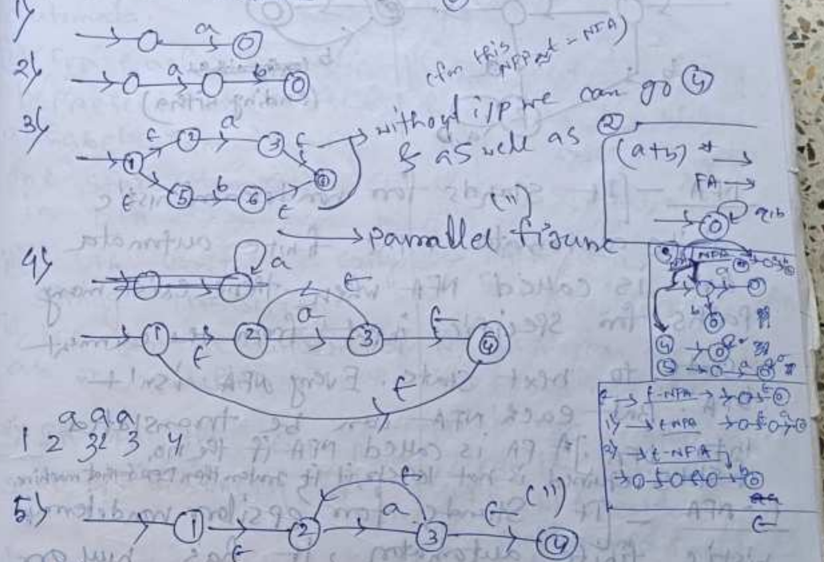
Convert ε-NFA to DFA (as all fa's lang's Power is same)

Done (PTO)



NFA

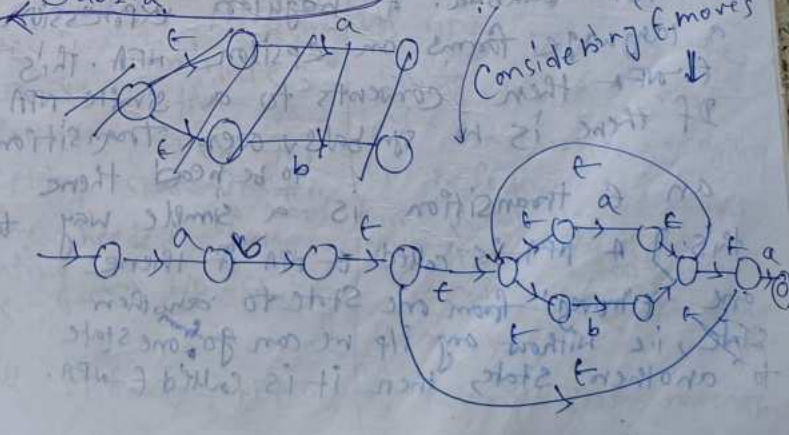
- It is a R.E, NFA = →◎ (ε-NFA as well) → (finals ①②③) All ε-NFA is NFA but vice versa is not correct



L: Starting with ab & ending with 'a',

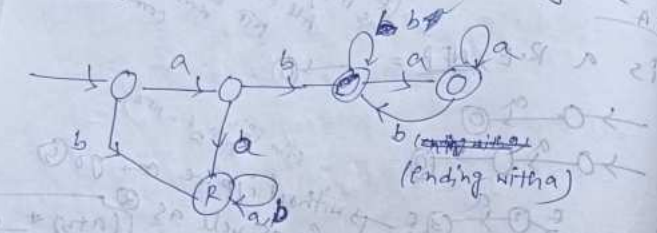
(Q) All strings ab (a+b)* a → NFA ? (ε-NFA as well)

Considering ε-moves



DFA

Smallest string case



NFA — It stands for nondeterministic finite automata. The finite automata is called NFA when there exist many paths for specific input from the current state to next state. Every NFA isn't DFA. But each NFA can be translated into DFA. An FA is called NFA if the no. of states required is not lesser, it is greater than DFA of that machine.

ε-NFA — It stands for epsilon nondeterministic finite automaton, it has null or epsilon transitions from one state to another. It is also called a null NFA or an NFA lambda. A regular expression for a language forms an epsilon NFA. This ε-NFA then converts to a simple NFA. If there is no symbols over transitions, to be read there an ε transition is a simple way to do this.] A NFA is called ε NFA if there are ε moves from one state to another state, i.e without any i/p we can go from one state to another state, then it is called ε-NFA.

DFA

i) It stands for deterministic finite automata.

ii) For each symbolic representation of the alphabet, there is only one state transition in DFA.

iii) DFA can't use empty string transition.

iv) DFA can be understood as one machine.

v) In DFA, the next possible state is distinctly set.

vi) DFA is more difficult.

vii) It rejects the string in case it terminates in a state that is different from the accepting state. (A state is present)

viii) time needed for executing an i/p string is less.

ix) All DFA are NFA.

NFA

i) It stands for non deterministic FA.

ii) no need to specify how does the NFA react according to some symbol.

iii) NFA can use Empty string transition.

iv) NFA can be understood as multiple little machines computing at the same time.

v) In NFA, each pair of state and input symbol can have many possible next states.

vi) It is easier to construct

vii) NFA rejects the string in the event of all branches dying or refusing the string. (A state isn't present)

viii) time needed for executing an i/p string is more.

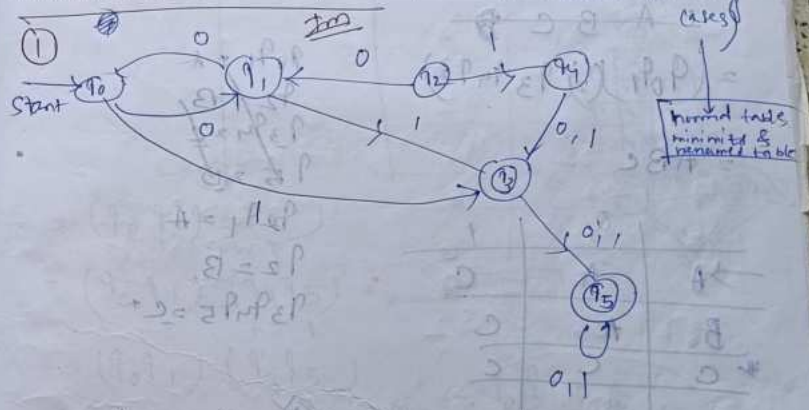
ix) Not all NFA are DFA.

DFA	NFA
xi) DFA requires more space.	xi) NFA requires less space than DFA.
xii) Dead configuration is not allowed.	xii) Dead configuration is allowed.
xiii) $Q \times \Sigma \to Q$, i.e next possible state belong to Q.	xiii) $Q \times (\Sigma \cup \epsilon) \to 2^Q$ i.e next possible state belongs to power set of Q.
xiii) Back tracking is not allowed here.	xiii) Back tracking is not allowed. always possible here.
xiv) conversion of Regular exp to DFA is difficult.	xiv) Conversion of Regular exp to NFA is simpler compared to DFA.
xv) ϵ move isn't allowed here.	xv) ϵ move is allowed in NFA.
xvi) It allows only one more from single o/p alphabet.	xvi) There can be choices (more than one move) for single i/p alphabet.

DFA minimization – It is also called optimization of DFA & use partitioning algo. It is a task of transforming a given deterministic (mechanism) finite automata into an equivalent DFA that has a minimum no. of states. Here 2 DFAs are called equivalent if they recognize the same regular language. (fg. from pic 4)

If a DFA table consists more than 1 final states then we need to understand that minimization (breaking) is needed. while the replacement of old final states into newname final states will be same then we have to understand that the minimization is complete.

minimization of DFA (2 tables are required for every cases)



required table minimized & renamed table

	0	1
→ q_0	q_1	q_3
q_1	q_0	q_3
q_2	q_1	q_4
* q_3	q_5	q_5
q_4	q_3	q_3
* q_5	q_5	q_5

You can construct DFA from NFA that will be direct dfa, but if you are constructing dfa by intuition it is said, to construct minimized dfa, you can do this minimization often reaching some dfa, as though you are following dfa making approach, but it wouldn't may be not minimized so do this

$(q_0 q_1 q_2 q_3 q_4 q_5)$

$= (q_0 q_1 q_2)(q_3 q_4)(q_5)$

~~$= (q_0 q_1 q_2)(q_3)(q_4)(q_5)$~~

$= (q_0 q_1)(q_3 q_4 q_5)(q_2)$

~~$= A\ (q_0 q_1)(q_2)(q_3 q_4 q_5)$~~

$= A\ B\ C\ D$

$= (q_0 q_1)(q_2)(q_3 q_4 q_5)$

$= ABC$

$q_0 q_1 = A$

$q_2 = B$

$q_3 q_4 = C$

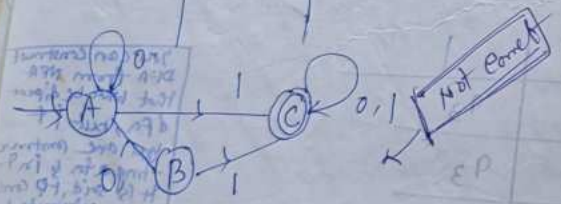
$q_5 = D$

$q_0 q_1 = A$

$q_2 = B$

$q_3 q_4 q_5 = C$

	0	1
→ A	A	C
B	A	C
* C	C	C



q_2 is not direct from q_1 & hence q_4 is not direct from q_1. so let's overlook of them, we can reach q_3 & q_5 from q_1 but not q_2 & q_4.

minimization $= (q_0 q_1 q_3 q_5)$

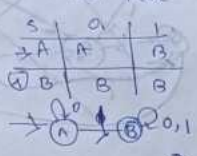
$= (q_0 q_1)(q_3 q_5)$

$= (q_0 q_1)(q_3 q_5)$

A B

minimized table

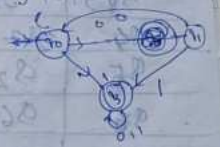
S	0	1
→ A	A	B
* B	B	B



	0	1
→ q_0	q_1	q_3
q_1	q_0	q_3
* q_3	q_5	q_5
* q_5	q_5	q_5

correct

	Correct	not correct
→ q_0	q_1	q_3
q_1	q_0	q_3
* q_3	q_5	q_5



$= (q_0 q_1 q_3 q_5)$

$= (q_0 q_1)(q_3 q_5)$ Not correct

$= (q_0 q_1)(q_3 q_5)$

~~$= A\ B$~~ $(q_0 q_1)(q_3)(q_5)$

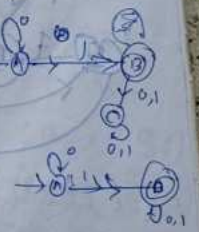
$= A\ B\ C$

$q_0 q_1 = A$

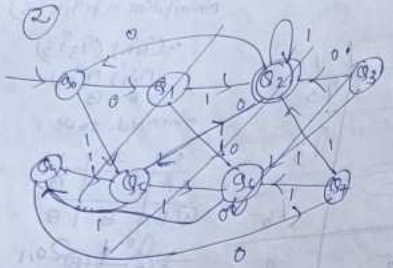
q_3 ~~q_5~~ $= B$

$q_5 = C$

	0	1
→ A	A	B
* ~~A~~ B	C ~~B~~	~~B~~ C
* C	C	C

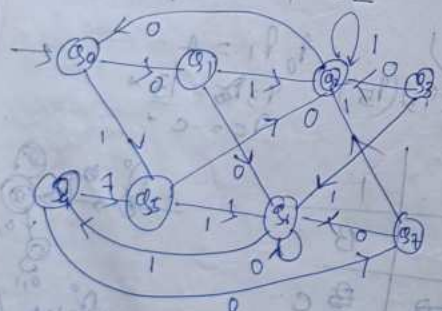


(2)



	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
(*) q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

→ not reachable (nobody is going towards q_3)



[Here final state is 1 but the pattern of many states are equal so we need to minimize]

	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
(*) q_2	q_0	q_2
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

$(q_0 q_1 q_2 q_4 q_5 q_6 q_7)$

$= (q_0 q_1 q_4 q_5 q_7)(q_2 q_6)$

$= (q_0 q_1 q_4)\ (q_1 q_5 q_7)\ (q_2 q_6)$

$= (q_0 q_4)\ (q_1 q_5 q_7)\ (q_2 q_6)$

$= A\ B\ C$

$= (q_0 q_4)\ (q_1 q_5 q_7)'\ (q_2)\ (q_6)$

$= (q_4 q_0)\ (q_1 q_7)\ (q_5)\ (q_2)\ (q_6)$

$= (q_0 q_4')\ (q_2')\ (q_1 q_7')\ (q_5')\ (q_6')$

	0	1
→ A	B	B
B	C	C
*C	A	C
C	C	A
*C	C	C

$= A\ B\ C\ D\ E$

q_2 = final

q_0 not q_2 F

q_6 = not final

q_6 q_4 not not

C C

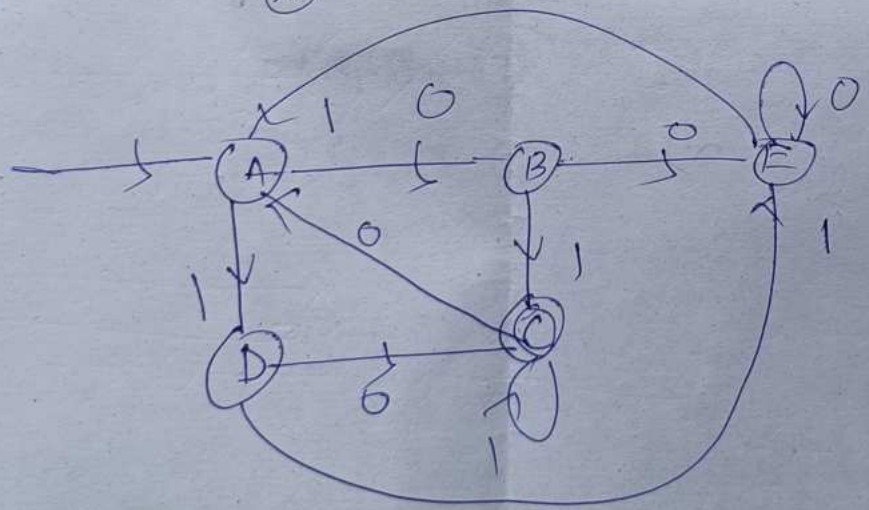
	0	1
→q_0	q_1	q_5
q_1	q_6	q_2

$(q_0 q_1 q_2 q_4 q_5 q_6 q_7)$

$= (q_0 q_4)\ (q_2)\ (q_5 q_6)\ (q_1 q_7)$

$= (q_0 q_4)\ (q_2)\ (q_5 q_6)\ (q_1 q_7)$

$= A\ B\ C\ D\ E$

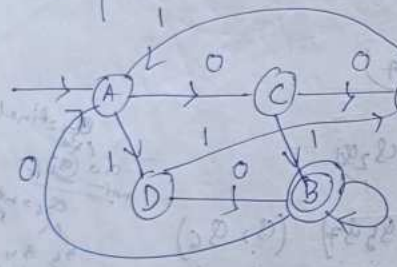


not reachable (nobody is going towards q_3)

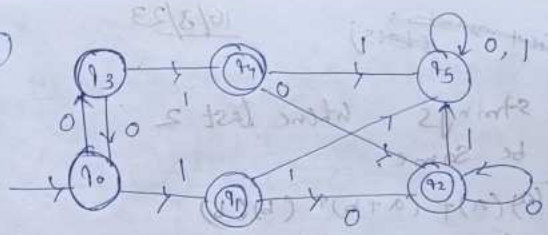
[Even final state is 1 but the pattern of many states are equal so we need to minimize]

$C \mid C \mid C = A\ B\ C\ D\ E$

	0	1
→ A	C	D
C	E	B
* B	A	B
D *	B	E
E	E	A



3)



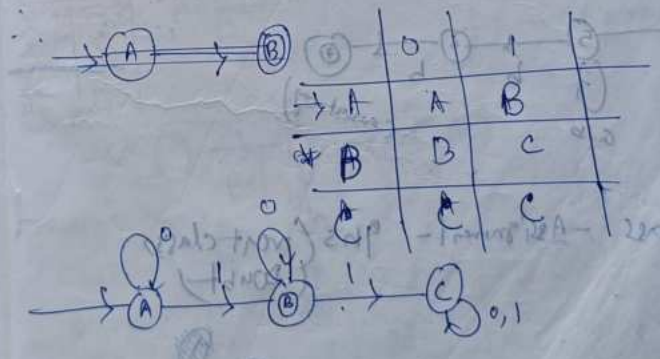
	0	1
→ q_0	q_3	q_1
* q_1	q_2	q_5
* q_2	q_2	q_5
q_3	q_0	q_4
* q_4	q_2	q_5
q_5	q_5	q_5

see the o/p pattern if any doubt then see o/p for i/p

$(q_0\ q_1\ q_2\ q_3\ q_4\ q_5)$

$= (q_0\ q_3)\ (q_1 q_2 q_4)\ (q_5)$

$= A \quad B \quad C$



	0	1
→ A	A	B
* B	B	C
C	C	C

16/8/23

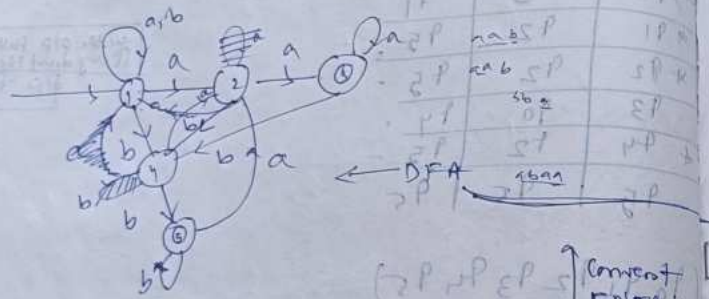
Assignment (Next week classes)

Prob (1) L: $ All strings where last 2 chars will be same.

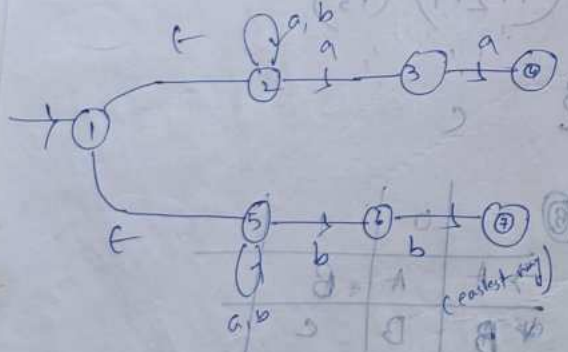
R.E = $(a+b)^*(a)(a)+(a+b)^*(b)(b)$

$= (a+b)^*(aa+bb)$ (or) $\Sigma^*(aa+bb)$

Smallest = aa, bb

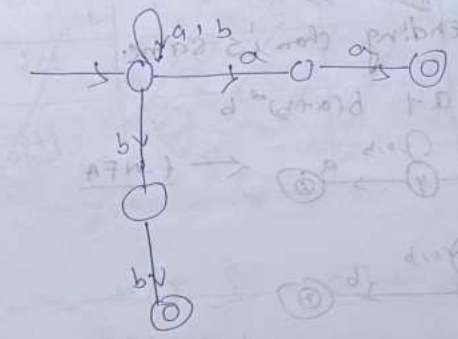


ε-NFA →

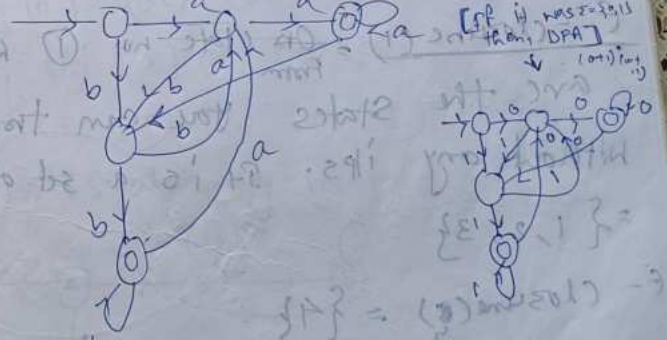


Hint: next class - Assignment - 9hrs (next class) (Doubt)

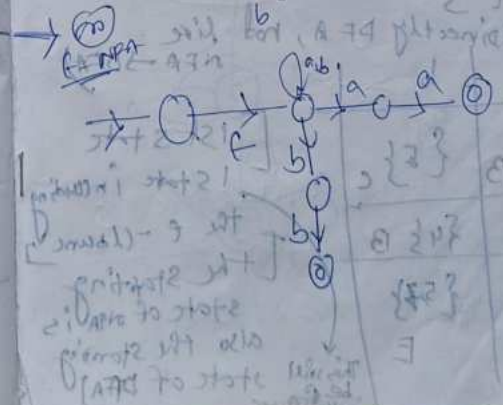
NFA → Hence NFA ≈ ENFA & also



DFA,



[If in terms of DFA]



b

ε-NFA → DFA (last 2 chars will be same)

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

ε-NFA - done

DFA transition table -

S	a	b
→ {1,2,5} A	{2,3,5} B	{2,5,6} C
{2,3,5} B	{2,3,4,5} D	{2,5,6} C
{2,5,6} C	{2,3,5} B	{2,5,6,7} E
*{2,3,4,5} D	{2,3,4,5} D	{2,5,6} C
*{2,5,6,7} E	{2,3,5} B	{2,5,6,7} E

minimization -

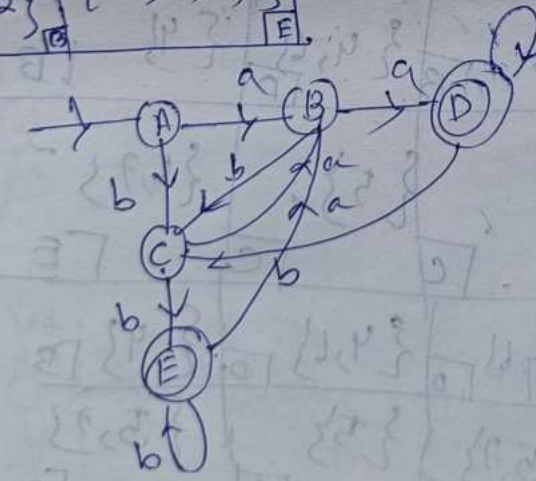
(A B C D E)

= (A B C) (D E)

= (A) (B) (C) (D) (E)

this is already minimized DFA.

∴ DFA →

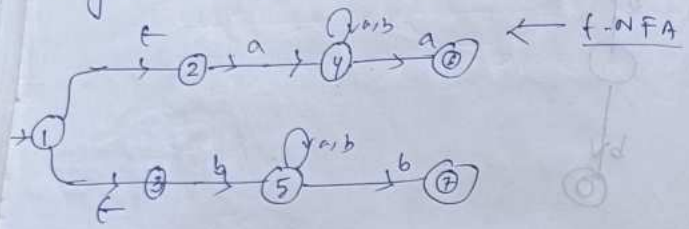


22/8/23

ε-NFA → DFA

① starting & ending char is same.

Reg Ex - $a(a+b)^*a + b(a+b)^*b$



← ε-NFA

abab ← Rejected

ε-closure(1) = from state no ① what are the states you can travel without any i/ps. It is a set of states.

= {1, 2, 3}

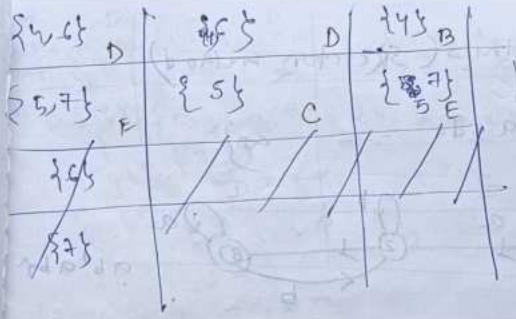
ε-closure(4) = {4}

DFA (Directly DFA, not like NFA → DFA)

States ↓	a	b
→ {1,2,3} A	{4} B	{5} C
{4} B	{4,6} D	{4} B
{5} C	{5} C	{5,7} E
{4,6} D	{5} C	{4} B
{5,7} E	{5} C	{5,7} E

[1st state 1 state including the ε-closure of the starting state of NFA, is also the starting state of DFA]

This will be final states



[Only closure includes ε moves if there is any given i/p like a/b then we have to consider a, b only for a/b we can reach where ε there ε moves]

(ABCDE)

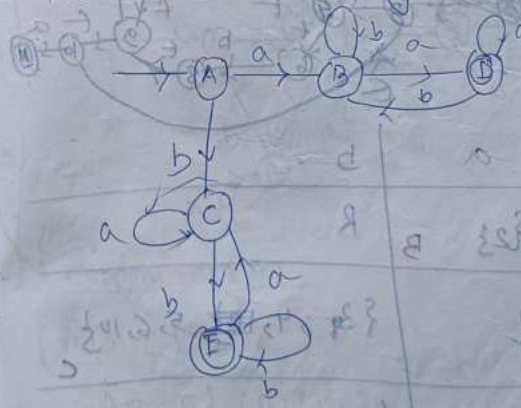
= (ABC) (DE)

= (A) (BC) (DE)

= (A) (B) (C) (DE)

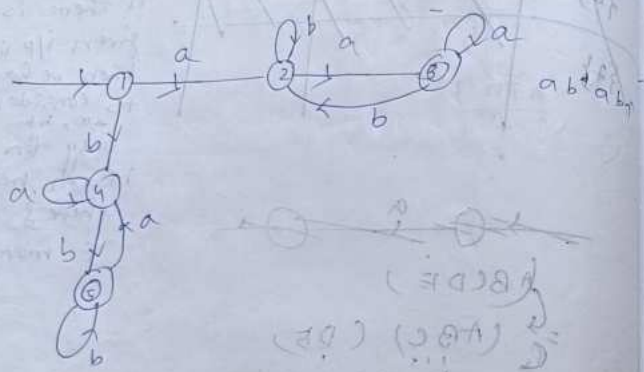
= (A) (B) (C) (D) (E)

the DFA is already minimized.



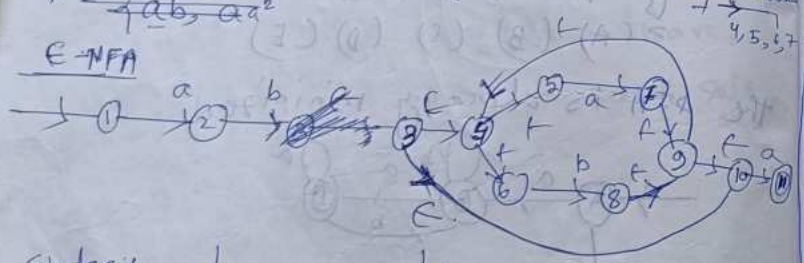
DFA for this (Skeleton method)

= {aa, bb, a ada ... a}



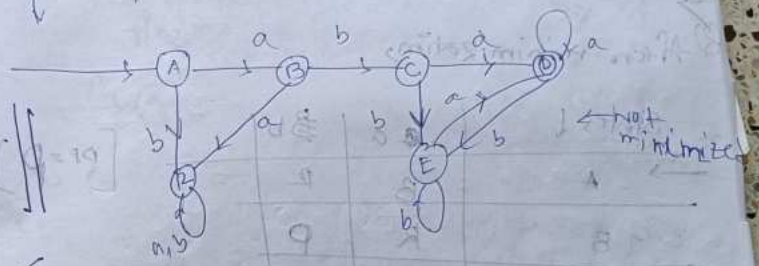
(2) a(a+b)*ba ← Reg EA

E-NFA

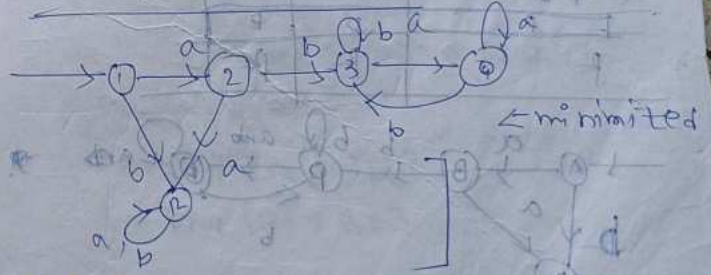


States	a	b
→ {1} A	{2} B	R
{2} B	R	{3, 4, 5, 6, 10} C
{3, 4, 10} {3, 4, 5, 6, 10} C	{7, 10, 5, 6, 11} {4, 5, 6, 7, 9, 10, 11} D	{4, 5, 6, 8, 9, 10} E

D	D	E
{4, 5, 6, 7, 9, 10, 11}	{4, 5, 6, 7, 9, 10, 11} same	{4, 5, 6, 8, 9, 10, 11} same
{4, 5, 6, 8, 9, 10} E	{4, 5, 6, 7, 9, 10, 11} D same	{4, 5, 6, 8, 9, 10, 11} E same
R	R	12



DFA (Skeleton method)



minimization

(AB CDE 12)

= (AB CE) (D) (12)

= (AB) (CDE) (12)

= (A) (B) (CDE) (12)

PTO

(A)(B)(D)(CE)(12) = (A)(B)(C)(DE)(12) = (A)(B)(P)(12)

Arden's Theorem / Lemma (applicable in automata) — Regular exp.

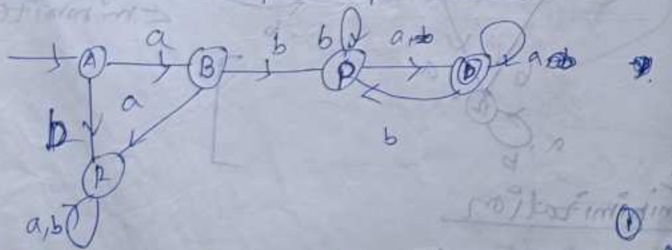
Pumping Lemma (If pumping lemma isn't satisfied by any lang. then this language isn't regular language)

PDA

After minimization,

States ↓	a	b
→ A	B	R
B	R	P
P	D	P
* D	D	P
R	R	R

[DF = P]



ε-closure - ε closure for a given state q is a set of states which can be reached from the states q with only ε-moves including the state q itself. ε-closure(1) = {1, 2, 3} (mr ex)

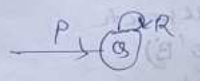
29/8/23

not needed
[(a+b+c) → [abc] or [a-c]

RE for a variable ⇒ RE → [a-z A-Z] [a-z A-Z 0-9]*

Arden's theorem

From finite automata to reg ex.
This is used to get reg ex from finite automata.



$Q = P + Q \cdot R$

$Q = P \cdot R^*$

[What is self loop this will be start later]

Prove

$Q = P + QR$

$= P + (P + QR)R$ [as $Q = P + QR$ (substitution method)]

$= P + PR + QR^2$

$= P + PR + (P + QR)R^2$

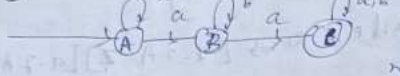
$= P + PR + PR^2 + QR^3$

⋮

$= P(\epsilon + R + R^2 + R^3 + \dots \infty)$ [∵ $P \cdot \epsilon = P$] [as we are replacing Q so $QR^3 = R^3$]

$= P \cdot R^*$ (Proved)

(1) Imp



[the reg ex of final state will be our ans reg ex]

$A = \epsilon + A \cdot a$ — (i)

$B = A \cdot a + B \cdot b$ — (ii)

$C = B \cdot a + C \cdot (a+b)$ — (iii)

(If we will apply a at A then we will reach to 'B')

Soln of (iii),

$C = B \cdot a \cdot (a+b)^*$ [$\because q = p \cdot R^*$]

[this isn't reg ex as B is a state]

from (ii)

$B = Aa \cdot b^*$

from (i)

$\therefore A = a^*$

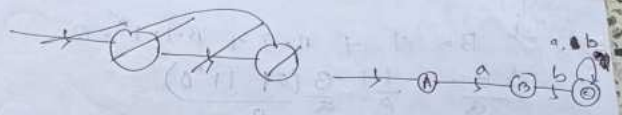
$\therefore$ from (iii) $\Rightarrow C = a^* a b^* a (a+b)^*$

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

$\therefore$ Reg ex of this finite automata

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

(2)



$A = \epsilon$ — (i)

$B = A \cdot a$ — (ii)

$C = B \cdot b + C \cdot (a+b)^*$ — (iii)

[we don't write reg reject state here in Arden's theorem]

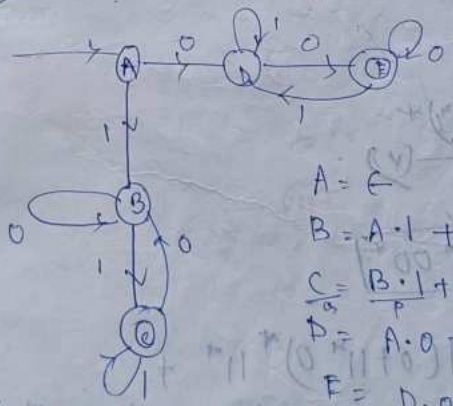
$\therefore C = B \cdot b (a+b)^*$

$= A \cdot ab (a+b)^*$ (from (ii))

$= ab (a+b)^*$ (from (i))

(3) Starting & ending with same char.

Imp



$A = \epsilon$ — (i)

$B = A \cdot 1 + B \cdot 0 + C \cdot 0$ — (ii)

$C = B \cdot 1 + C \cdot 1$ — (iii)

$D = A \cdot 0 + D \cdot 1 + E \cdot 1$ — (iv)

$E = D \cdot 0 + E \cdot 0$ — (v)

$\therefore C = B \cdot 1 \cdot 1^*$ — (vi)

replacing (v) in (ii) into (iii)

$\Rightarrow B = 1 + B\cdot 0 + B\cdot 1\cdot 1^*\cdot 0$

$\Rightarrow \frac{B}{q} = \frac{1}{p} + \frac{B}{q}\frac{(0+11^*0)}{R}$

$\therefore B = 1\cdot(0+11^*0)^*$ — (VII)

from (II) & (VIII),

$C = 1(0+11^*0)^*11^*$

from (V),

$E = D00^*$ — (IX)

replacing (IX) & (I) in (IV),

$D = 0 + D1 + D00^*\cdot 1$

$\frac{D}{q} = \frac{0}{p} + \frac{D(1+00^*1)}{q\ R}$

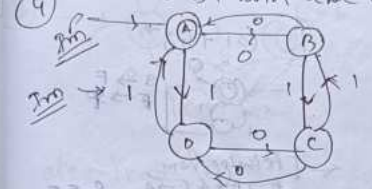
$\therefore D = 0(1+00^*1)^*$ — (X)

from (X) & (I),

$E = 0(1+00^*1)^*00^*$

$\therefore$ Reg ex $= (1(0+11^*0)^*11^* + 0(1+00^*1)^*00^*)$

(4) no of 0's & 1's both are divisible by 2 (i.e even no. of 0's & 1's)



[we are taking C and eleminating B, D as, if we eleminate B, keeping C then we have to encounter only B, when we are searching any path from C → A through B, same goes to D. But if we will take B or D then we have to encounter 2 states, for B (C,D) & for D (C,B) so, we are using C]

[If you are deleting B you have to go through B, i.e you have to find a path that encounters B like that D-too]

$A = \epsilon + B\cdot 0 + D\cdot 1$ — (I)

$\frac{A}{q} = \frac{A\cdot 0}{p} + \frac{C\cdot 1}{q\ R}$ — (II)

$\frac{C}{q} = \frac{B\cdot 1}{p} + \frac{D}{q}\cdot\frac{0}{R}$ — (III)

$\frac{D}{q} = \frac{A\cdot 1}{p} + \frac{C\cdot 0}{q\ R}$ — (IV)

from (II),

$B = A01^*$ — (V)

from (III),

$C = B10^*$ — (VI)

from (IV),

$D = A10^*$ — (VII)

from (I), ← not correct

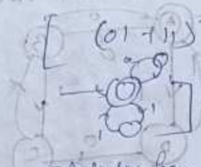
$A = \epsilon + A01^*0 + A10^*1$

$\Rightarrow \frac{A}{q} = \frac{\epsilon}{p} + \frac{A}{q}\frac{(01^*0 + 10^*1)}{R}$

$\therefore q = (01^*0 + 10^*1)^*$ $\therefore$ reg ex $= (01^*01 + 10^*1)^*$

Short cut ;

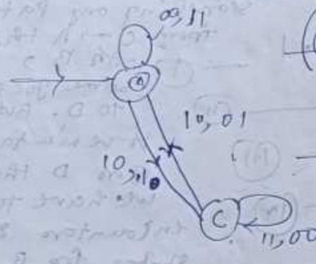
{ Path →
{ Circuit →
loop



(01 + 11)*

S → F
F → F

← [Actually here A → A & C → A both are same] & S → F, F → F



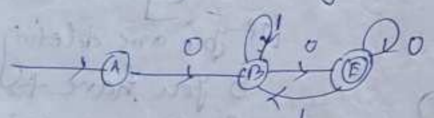
~~((00 + 11)* + (10 + 01)(11 + 00)*~~

~~(10 + 01)~~

→ S → F (00, 11, (10+01)(11+00)* (10+01))

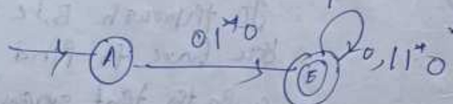
(00 + 11 + (10 + 01) (11 + 00)* (10 + 01))

Starting & ending with '0'.



→ [A → E through B → v]

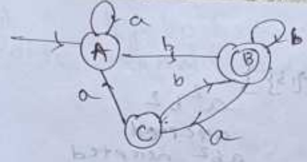
↓ Shortcut



$01^*0\,(0 + 11^*0)^*$ → [to go E → E]

S → F → [to go A → E]

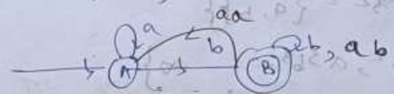
5



← use short cut and solve.

[just keep starting & final state]

eleminate C,



path A → B through C → λ

B to A → λ

$(b + ab + aa\,a^* b)^*$ → (to go B → B we need that loop)

a^*b → through this we can go to B. Cos B is final state then we have to consider the point of view from B.
S → F

[Put itself's state (B) normal (normal self loop),
Put all other state through the state to it proper (self loop → +)]

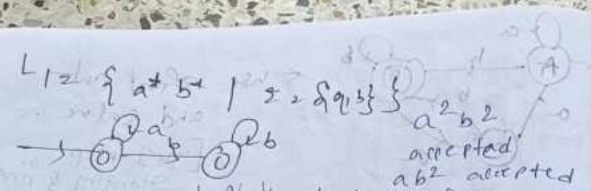
5/10/23

Cw

Pumping lemma (for regular language)

This is a negativity test for RL. It says the lang isn't Regular lang. It can't give the conclusion about a lang is Regular or not. But it conclude a lang isn't regular lang. Pumping lemma is a test to prove that a lang isn't a regular lang.

$L_1 = \{ a^* b^* \mid \Sigma = \{a, b\} \}$



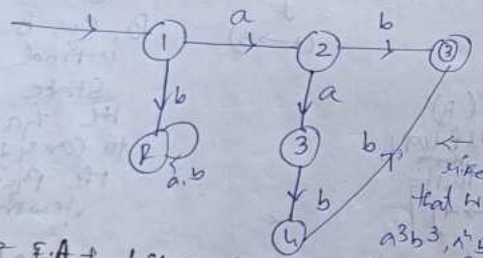
$a^2 b^2$ accepted
ab^2 accepted

↳ infinite level comparison

$L_2 = \{ a^n b^n \mid n \geq 1, \Sigma = \{a, b\} \}$

$= \{ ab, a^2 b^2, a^3 b^3 \cdots \infty \}$

$a^2 b^2$ accepted but, ab^2 isn't accepted.



since that way $a^3 b^3$, $a^4 b^4$ ... $a^n b^n$. no: of a's & b should be equal but we can't do that by self loops like this way (as how many comparison can we do till infinity..!!)

PDA = FA + 1 stack

① $L = \{ a^p \mid p \text{ is prime} \}$

try, (this is not a regular lang)

no, $\Sigma = \{a\}$

$= \{ a^2, a^3, a^5, a^7, a^{11} \cdots \infty \}$

we can't construct a DFA for this, as for as we have to remember states, for a^7, a^{11} ... ∞ there are infinite level comparison. so, it's not a Regular lang, as for this lang finite automata doesn't exist.

Pumping lemma

Pumping lemma language, says if any lang is regular then, there will be pumping length (p) (p is any int value) & a string $w \in L$, & we can divide $w = xyz$ such that -

i) $|xy| \leq p$

ii) $|y| \geq 1$

iii) $xy^i z \in L$

[$|w|$ should $\geq p$]

Proof:-

$L = \{ a^n b^n \mid n \geq 1, \Sigma = \{a, b\} \}$ isn't regular.

Suppose L is Regular.

let, $w = a\,a\,a\,b\,b$

$= x\,y\,z$

$p = 3$

(w should be $\geq$ p as it will be divided into 3 parts)

(i) $|xy| \leq p$

$\therefore 2 \leq 3$ (satisfied)

(ii) $|y| \geq 1$

$\therefore y \geq 1$ (there should be any element in y) (satisfied)

① ②

let,

$i = 2$, xy^2z = aaabb

$= a^3b^2 \notin L$

(not satisfied)

Pumping lemma isn't satisfied so it isn't regular lang.

let,

w = aaabbb (x y z) P = 4

(i) $y \le 4$

(ii) $y = 2 (>1)$

(iii) $i = 2$, xy^2z

= aaababbb

$= a^3bab^3 \notin L$

[y^i → means can we create any loop/self loop. If we can't form loop the lang is infinite if it can't be a R.L]

This isn't a R.E

① **Push down automata** (the langs in exam will be from there - from solved eg in class)

PDA
- DPDA (Deterministic PDA)
- NPDA (nondeterministic PDA)

[Here we 1st read DPDA then NPDA (but in R.E we 1st learn ε-NFA, NFA, DFA)]

(the powers of NPDA is greater than DPDA)

(It itself a strong machine than DPDA, so there is no conversion from NPDA → DPDA)

the lang. for which PDA exists, is called context free language (CFL).

PDA (7 tuples)

$(Q, \Sigma, \delta, q_s, F, \gamma, Z_0)$

Z_0 → initial stack symbol; γ → finite set of stack symbols

$[Z_0 \subset \gamma]$

Stack

[If Top = Z_0, Stack is empty]

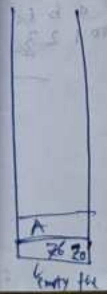
[the Stack helps us to remember the symbols just like remaining of states]

① $L_1 = \{a^nb^n \mid n \ge 1, \Sigma = \{a, b\}\}$

$= \{ab, a^2b^2, \ldots \infty\}$ (Skeleton method isn't applicable here)

[when b will arrive we pop out A & will do nothing (ε)]

aabb

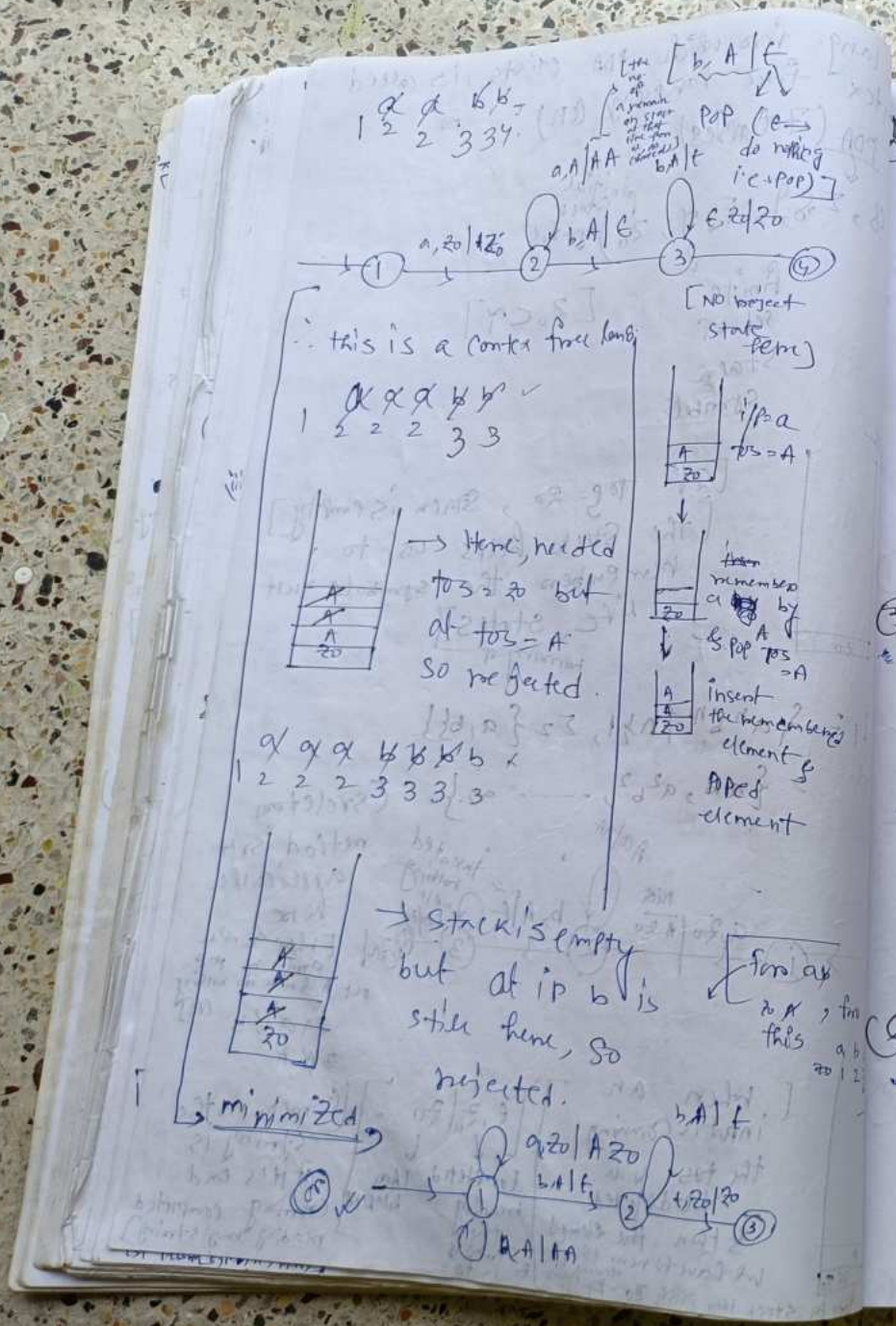


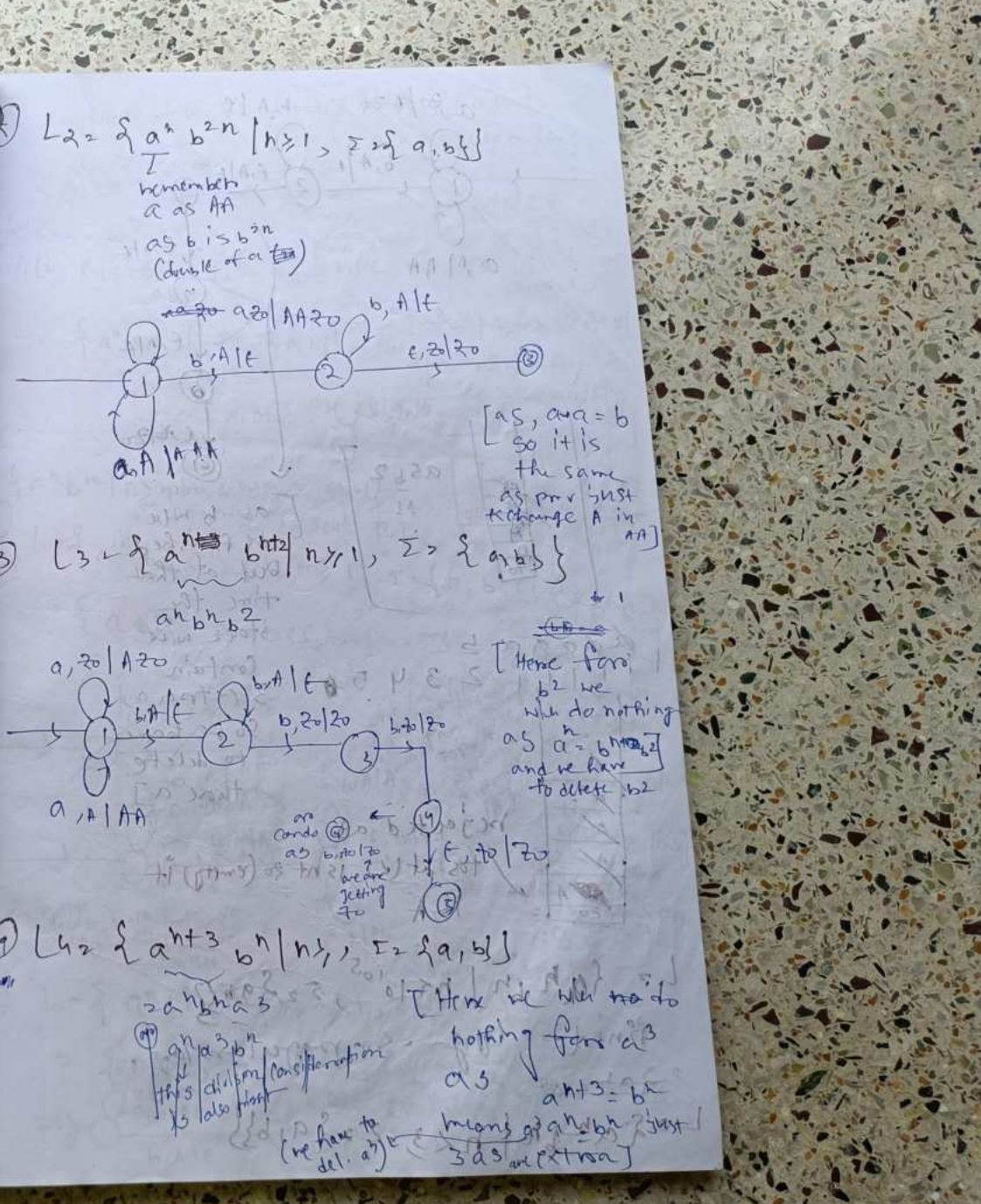
[when an input is coming the tos will be deleted & then we have to rem. the element it pushing]

[ε, Z_0|Z_0 completed reading the string & Z_0 is tos, then placed]

[ε means the string is at it's end (completed reading my string)]

Empty the stack then push Z_0


a, Z0 | AZ0
b, A | ε
ε, Z0 | Z0
a, A | AA
b, A | ε
pop (ε → do nothing i.e. pop)
[NO reject state here]
this is a context free lang.
→ Here, needed tos = Z0 but at tos = A so rejected.
remember a by A & pop tos = A
insert the remembered element & poped element
→ stack is empty but at ip b is still there, so rejected.
minimized
a, Z0 | AZ0
b, A | ε
1, Z0 | Z0
a, A | AA
b, A | ε


(2) L2 = { a^n b^2n | n ≥ 1, Σ = {a, b} }
remember a as AA
as b is b^2n (double of a)
a, Z0 | AAZ0
b, A | ε
ε, Z0 | Z0
a, A | AA
[as, aa = b so it is the same as prev just exchange A in AA]
(3) L3 = { a^n b^n+2 | n ≥ 1, Σ = {a, b} }
a^n b^n b^2
a, Z0 | AZ0
b, A | ε
b, Z0 | Z0
b, b | Z0
a, A | AA
ε, Z0 | Z0
[Here for b^2 we will do nothing as a^n = b^n+2 and we have to delete b^2]
(4) L4 = { a^n+3 b^n | n ≥ 1, Σ = {a, b} }
= a^n b^n a^3
[Here we will do nothing for a^3 as a^n+3 = b^n means a^n = b^n just 3 as are extra]
(we have to del. a^3)

remember a as AA
as b is b^{2n} (double of a)

$a z_0 | AAz_0$ b, A|ε

[NO reject state here]

a context free lang

(~~correct~~ in copy)

~~not correct here because &~~ like PA sequence ~~also~~ matters here also. So if you think, ~~for~~ Eg - $a^n b^m c^n \simeq a^n c^n b^m$ it's not correct imagine the stack for $a^n b^m c^n$, b^m can be absorbed but ~~for~~ if you do $a^n c^n b^m$ then c will 'cut' a but what about b^m ?!

(4) isn't correct here, correct PDA is in copy. because, $a^{n+3} b^n \simeq a^n a^3 b^n$ → this is same as $a^n b^n a^3$ ~~becau~~, this isn't blundering the sequence because imagine the stack for $a^n a^3 b^n$, if say a=2 then in stack '5 a' will be present & '2 a' will be cut out by '2b' & 3a' will still remain in the stack that will be absorbed or can say popped out, ~~same thi~~ this is same for $a^n b^n a^3$. so they are equivalent.

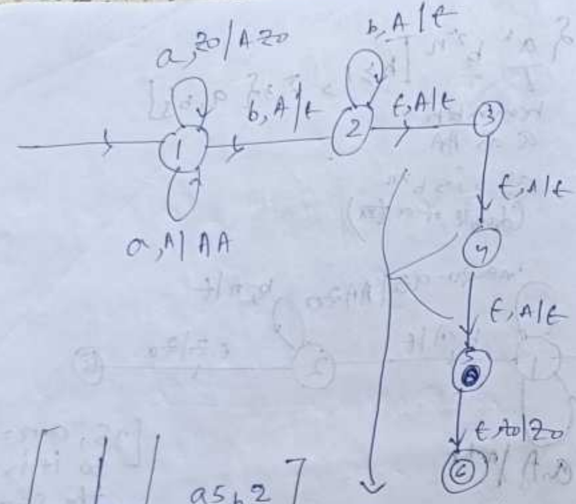
(3) isn't correct here, correct PDA is in copy. Here also $a^n b^{n+2} \simeq a^n b^n b^2$ → same as $a^n b^2 b^n$ but $a^n b^2 b^n$ will give complexity so no need, ~~it~~ $a^n b^n b^2$ is already arranged & we always try to arrange/organize i/ps so that there will be no complexity. now, here we can't draw PDA like the (4)th as in (4)th a^{n+3} ~~will~~ were already pushed & b^n was comming so we were cutting a^n for b^n & popping out rest a^3. But here a^n is already pushed & b^{n+2} is comming so, we will cut off a^n for b^n. & when rest of b^2 will come it will see 'z_0' (empty) in stack (stack empty), so we will absorbe 'b' by seeing it as 'z_0' (like copy).

When building PDA always imagine the stack 1st.

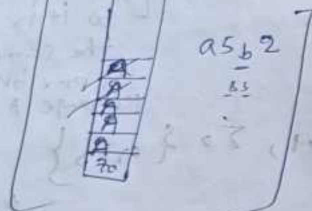
a3

$a^{n+3} = b^n$ means $a^3 a^n = b^n$ just 3 a's are extra]

(we have to del. a^3)



[as b will be finished but at that time the stack will contain 3 extra a. So we have to delete those a]



a a a a a b
1 1 1 1 1 2 3 4 5

rejected as tos at (e) isn't z0 (empty) it is A

$L = \{a^n b^n \mid n < 10^{10^5}, \Sigma = \{a, b\}\}$
finite lang. So regular lang.

~~= {ab, a²~~

$L = \{a^n b^n \mid n < 3, \Sigma = \{a, b\}\}$

$= \{ab, a^2 b^2\}$ finite lang, so it is regular lang.

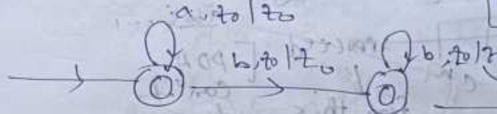
7/9/23

chm

- All RLs are CFL. But vice versa isn't correct.

$L = \{a^* b^* \mid \Sigma = \{a, b\}\} \Rightarrow \{a^n b^m \mid n, m \geq 0, \Sigma = \{a, b\}\}$ (RL)

[as no comparison so, nothing here]



[this is PDA]
[All RLs are CFL]

- $\{a^n b^m \mid n > m \geq 0, \& n \neq m, \Sigma = \{a, b\}\}$ comparison — not R.L.

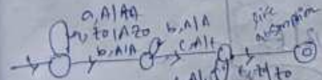
(1) $L = \{a^n b^m c^n \mid n, m \geq 1, \Sigma = \{a, b, c\}\}$ $(a^n b^* c^n)$

$= \{abc, aabcc \ldots\}$

[then b will come we will not pop 'a' as when 'c' will come we will do this]
[as, there is comparison of a & c]

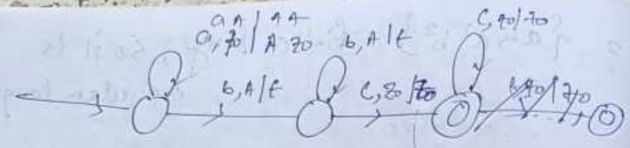
what we want we are only putting this b on top of A. It we only pop A remember it through A

we also have this in 4 steps too



(2) $L = \{a^n b^m c^m \mid n, m \geq 1, \Sigma = \{a, b, c\}\}$

$= \{abc, aabbc \ldots\}$ ε, z0/z0 → no need



③ $L = \{a^n b^{n+m} c^m \mid n, m \geq 1, \Sigma = \{a,b,c\}\}$ — Imp

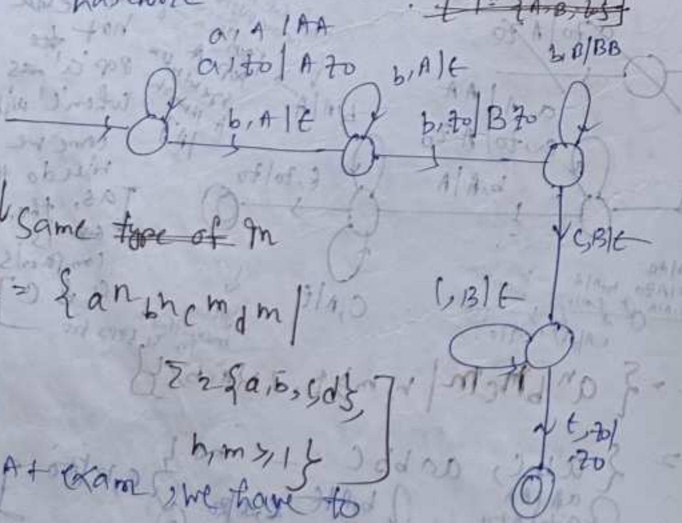
$\{a^1 b^2 c,\ a^2 b^3 c^2 \ldots\ \infty\}$

$a^2 b^4 c^2 \ldots\ \infty$ — $a^2b^2c \ldots$

$a^n b^m b^n c^m$ → proceed — for this PDA isn't possible [PDA can't do 2 level infinite comparison]

$a^n b^n$ — $b^m c^m$ → No 2 level ∞

comparison has done — another comparison has done — comparison (DPDA)



↓ same type of m

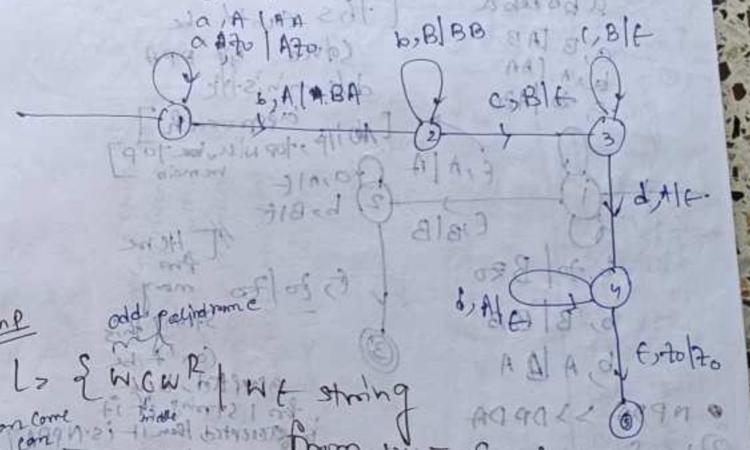
$[L = \{a^n b^n c^m d^m \mid \Sigma = \{a,b,c,d\},\ n,m \geq 1\}]$

(At exam, we have to mention tuples) ——— scrutiny

$\Gamma = \{A, B, z_0\}$
$\Sigma = \{a, b\}$

④ $L = \{a^n b^m c^m d^n \mid n,m \geq 1, \Sigma = \{a,b,c,d\}\}$ — Imp

$a^n b^m c^m d^n$



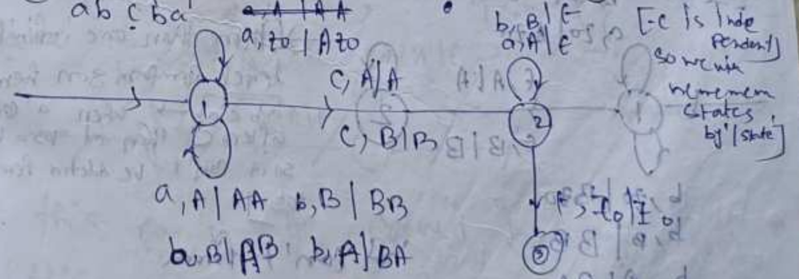
Imp odd palindrome

⑤ $L = \{w c w^R \mid w \in$ string from $\Sigma = \{a,b\}\}$

[a can come 1st, b can come 1st]

b a c a b — w^R ⟹ Reverse of w

(or) a b c b a



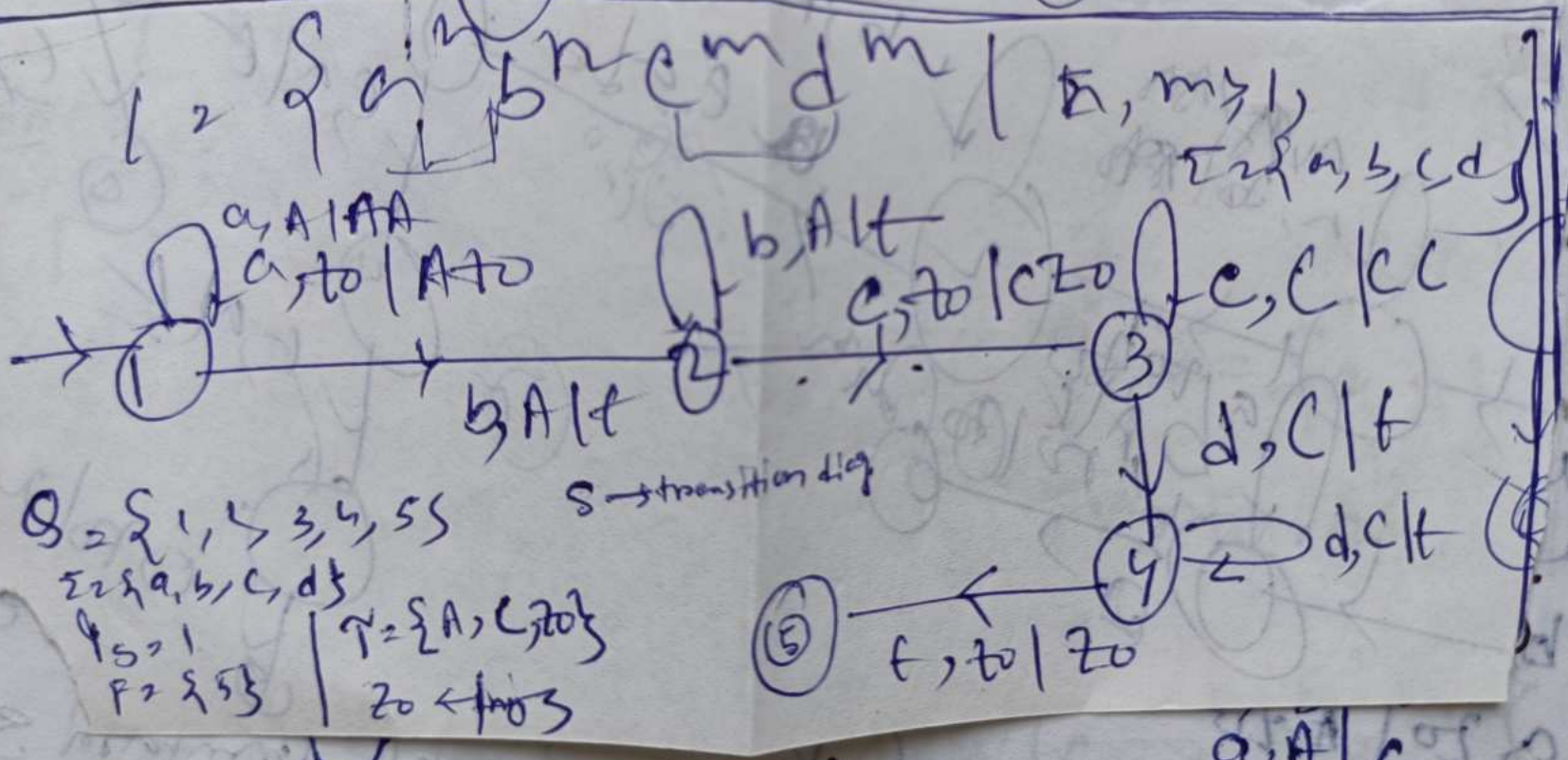
[c is independent so we can increment states by 1 state]

[If there is only one 'a' or 'b' comes, → [a,A|ε / b,B|ε], if there is as many a/b comes, → [a,A|ε / b,B|ε]]

Imp

odd palindrome

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



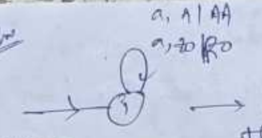
$Q = \{1, 2, 3, 4, 5\}$
$\Sigma = \{a, b, c, d\}$
$q_s = 1$
$F = \{5\}$

$\Gamma = \{A, C, Z_0\}$
$Z_0 \leftarrow$ ~~this~~ 5

S → transition diagram

[-c is

12/9/23



→ this not NPDA
this is DPDA

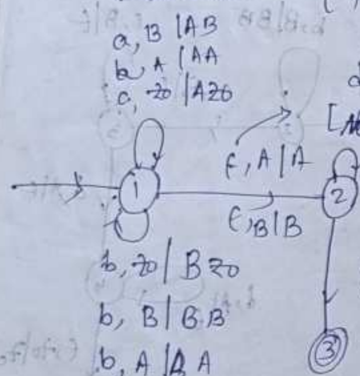
NPDA

① L → {w w^R / w ∈ string formed by {a, b}} (even palindrome)

NPDA (NCFL)

abbbaba

[this can't be solved by DPA deterministic approach]



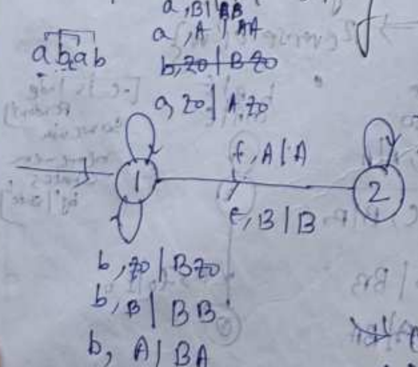
[No i/p, top will be top remain]

[Here for many string this can't be accepted, but for 1 string if it is accepted then it is NPDA]

• NPDA >> DPDA

② L → {wcw | w ∈ string formed by {a, b}}

abab



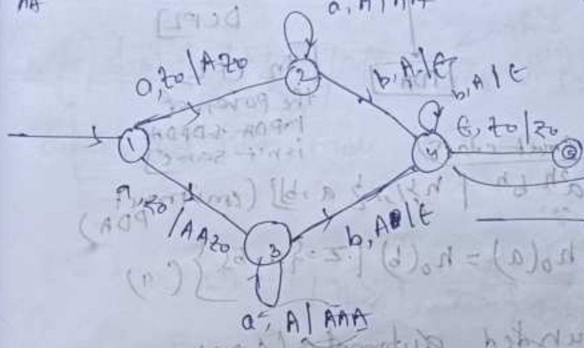
→ [more than one infinite level comparison here
ab c ab → when 'a' comes after 'c' then at TOS = b so it can't be deleted from TOS]

can't form 1 PDA (DPDA / NPDA)

this isn't a CFL. Even ww isn't a CFL.

③ L → {a^n b^n ∪ a^n b^{2n} / n ≥ 1, Σ = {a, b}}

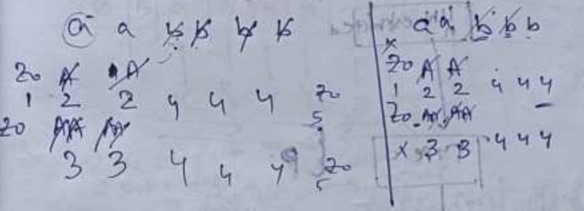
aabbbb



← this is NPDA (for this DPDA isn't possible)

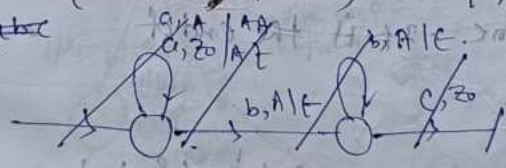
[moving 2 transitions as after great state of both trans. is same]

→ [while popping for 'AA' we will pop one A as there is double no of bs, the another A will be deleted after that but it will be deleted first.]



more than 1 level of infinite comparison

④ L → {a^n b^n c^n | n ≥ 1, Σ = {a, b, c}}



aabbcc

this can't be solved by PDA. For this we need turing machine / (LBA)]

Here there is more than 1 level of infinite comparisons.

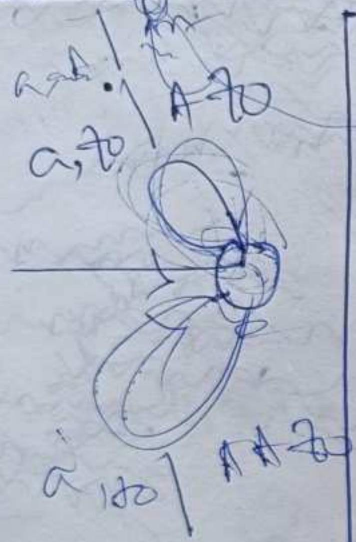
i.e you can't merge 2 states / transactions instead of checking that the operations on both are same or not regardless (though their next transaction/state is same!

a, A | AAA

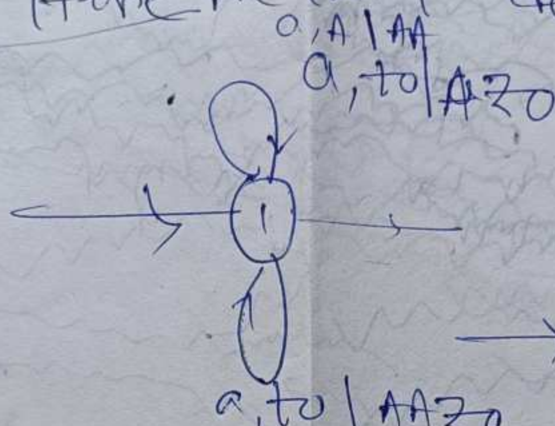
(a) a b b b b | a a b b b

Z0 A A | Z0 A A

for 'AA' we will pop one A as there is double no of



Here we can't do,



as the meaning of both is diff.

X

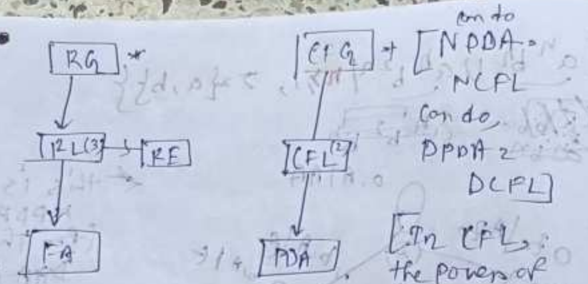
→ this isn't NPDA, this is indicating DPDA

this con... is same! ... transaction)

[this we need tuning machine/(LBA)]

Here there is more than

infinite ...

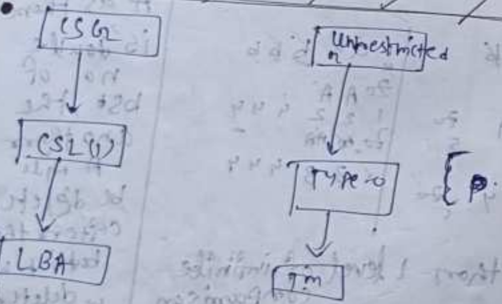


NW Assignment (next class)

(5) $L = \{a^{2n} b^n \mid n \geq 1, \Sigma = \{a, b\}\}$ (construct PDA)

(6) $L = \{a^n b^n \mid n_o(a) = n_o(b) \mid \Sigma = \{a, b\}\}$ (")

Linear bounded automata (LBA)



- All machines are the subset of turing machine. at the end of the logic.
- WNP is D on N that is depended on machine, that is not depended on language.
- ~~FA < RP~~ NTM ≥ DTM ≥ NPDA ≥ DPDA ≥ FA

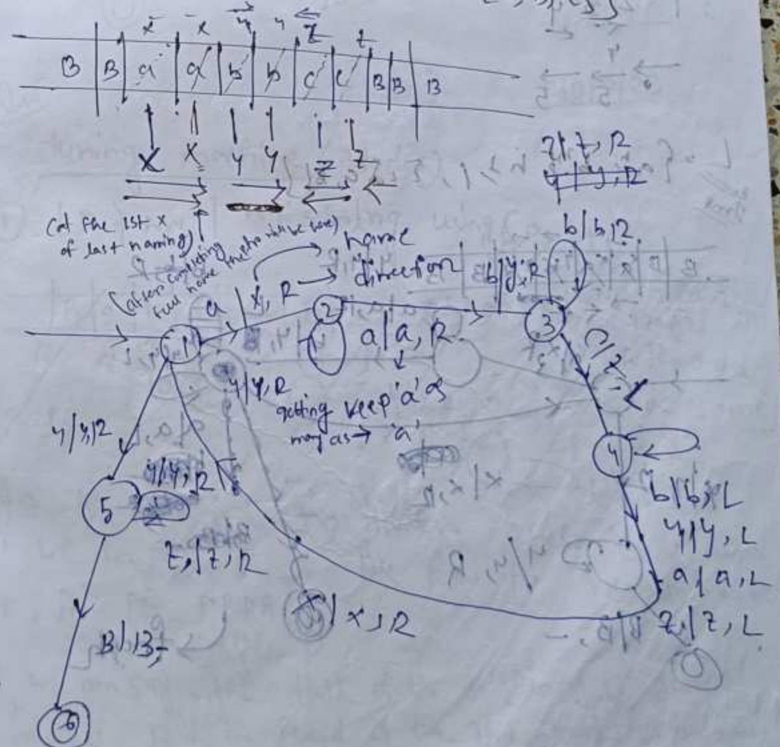
Turing machine (TM)

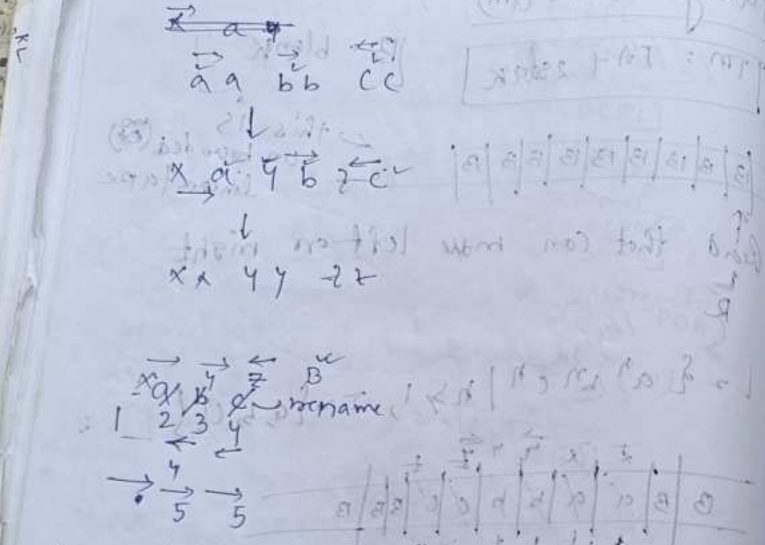
- TM = FA + 2 stack

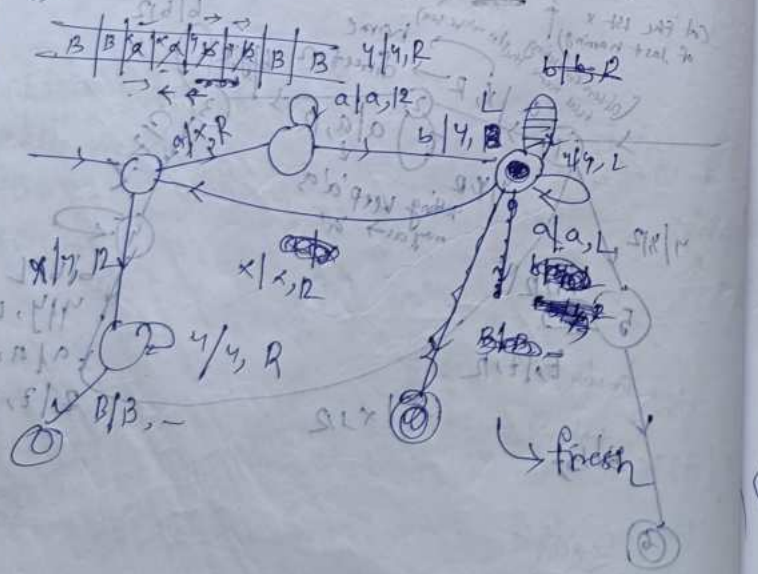
[B = blank]

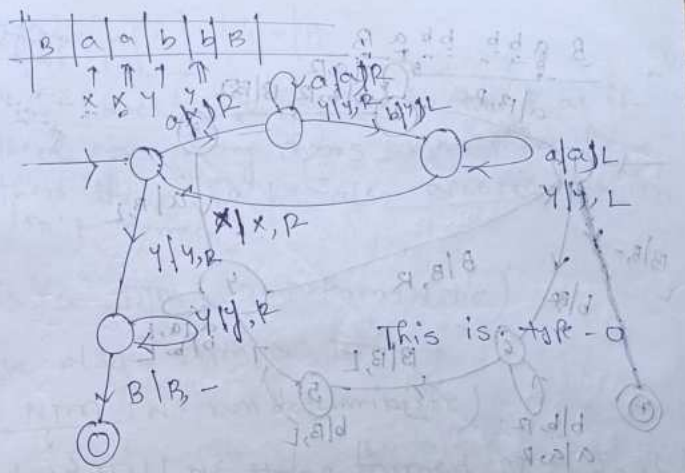
Read that can move left or right (L R)

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





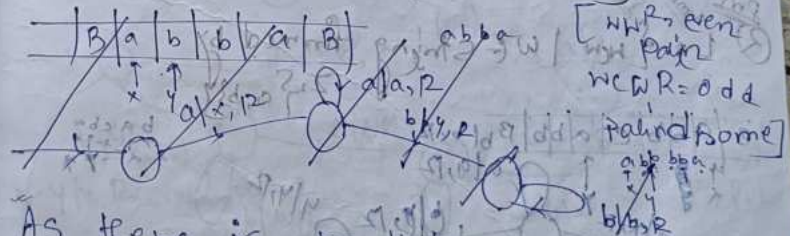




(3) 10/9/23

tuning machine (7 tuples)

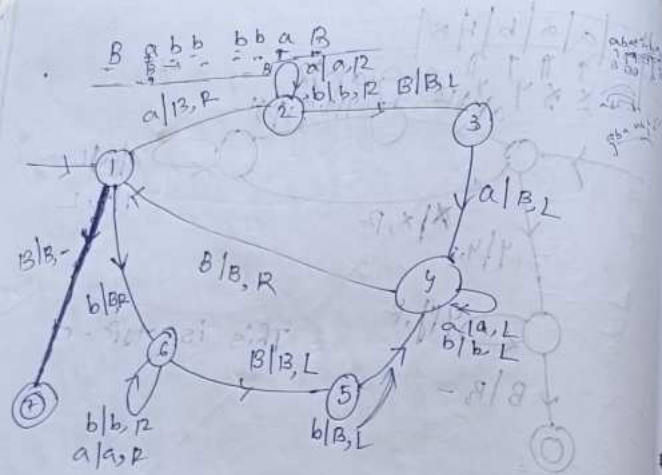
① $L = \{wwR \mid w \in \text{string using } \{a, b\}\}$

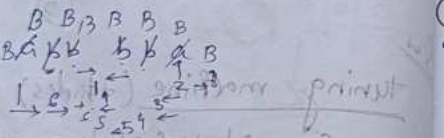


[wwR, even palindrome
wcwR = odd palindrome]

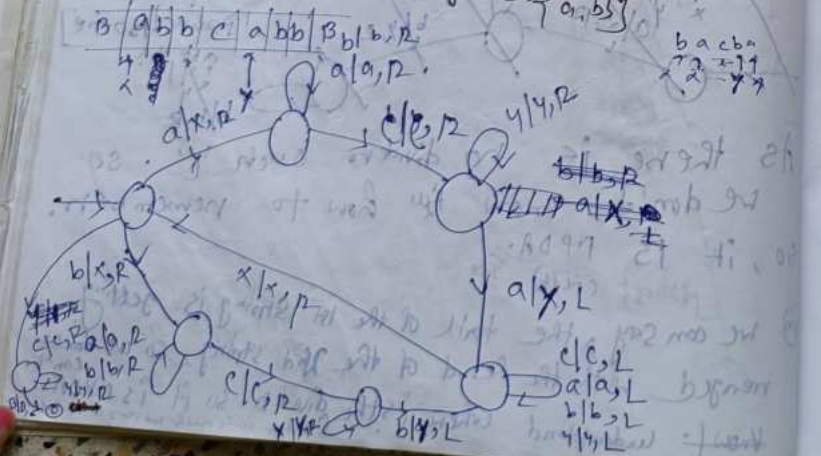
As there is no divider such 'c'. so we don't know the how to remember.
so, it is NPDA.

② we can say, the tail of the 1st string is getting merged with the head of the 2nd string, so we can't understand where is the divider. so it is NPDA.





L = {WW | W ∈ string formed by Σ = {a, b}}



we have to go further to the R as if there are many chars more than here then this will be also accepted, which isn't correct.

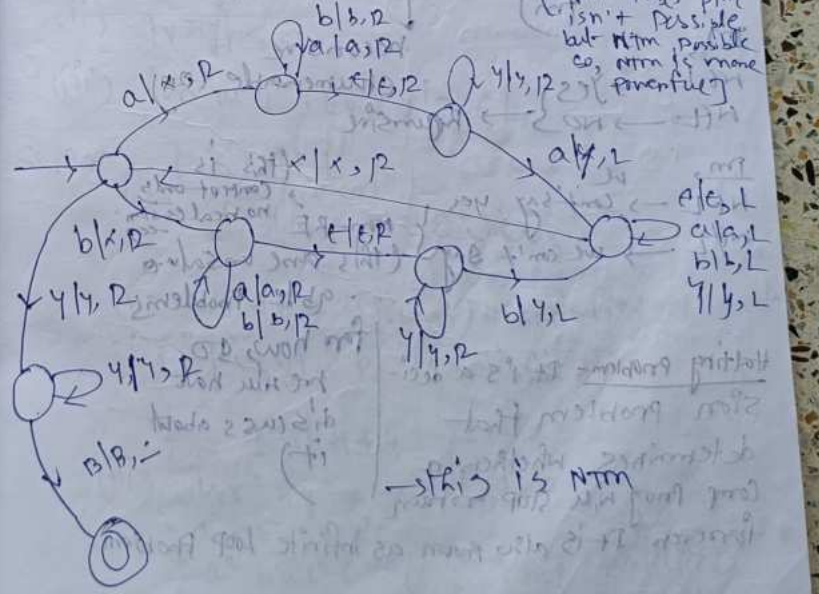
This is DTM. (D = Deterministic)

Here also NTM ≥≥ DTM.

NTM (N: non deterministic)

L = {WW | W ∈ string formed by Σ = {a, b}}

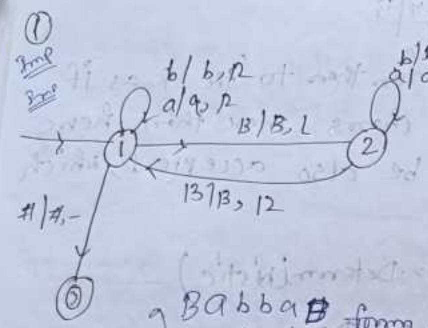
TM TMP



[for this DTM isn't possible but NTM possible so, NTM is more powerful]

→ This is NTM

(1)



BabbaB → forms an infinite loop } Halting Problem

for this, TM
- w∈L → yes/Accept
- w∉L → you can't say

(concept: Partial TM)

(this is a problem that actually not showing halts)

→ recursive enumerable (RE)

TM
- w∈L → yes
- w∉L → NO } → Recursive

TM
- w∈L → can't say yes
- w∉L → we can't say } Non-RE (this are unsolvable problems)

(this is concept only no real custom eeg)

Halting Problem – It is a decision problem that determines whether a comp. prog will stop or run forever. It is also known as infinite loop Problem.

(for now, so we will not discuss about it)

Pumping lemma for CFL (to check it is not CFL)

$L = \{a^n b^n c^n \mid n \geq 1, \Sigma = \{a,b,c\}\}$ → this isn't CFL

If L is a CFL, pumping length = p
we consider a string $w \in L$, $|w| \geq p$,
and $w = uvxyz$ [p is any integer]

(i) $vy \neq \epsilon$
(ii) $|vxy| \leq p$
(iii) for all i, $uv^i x y^i z \in L$

Suppose L is CFL.
let, p=4, w = aabbcc (u v x y z)

(i) $vy \neq \epsilon$ (vy=3) (satisfied)
(ii) $|vxy| \leq p$ (satisfied)

4≤4

(iii) uv^2xy^2z let, i=2
$= a\,abab\,b\,c^2c$
$= a^2bab^2c^3 \notin L$ (not satisfied)

∴ This isn't CFL.

Grammer 10/10/23

- $L = \{a^n b^n c^n \mid n \le 10^5, \Sigma = \{a,b,c\}\}$ this is not infinite lang of it is finit lang.
- TM > CFL

Grammer → Role: Grammer generates language.

(N, T, P, S) → 4 tuples

N → set of non terminals
P → set of production rules
S → starting symbol (this is non terminal)
T → set of terminals

$S \subseteq N$

Eg- of grammer.

① Production rule

P → aQb
aQ → abD
D → ε

collection of Production rules is called grammer

- Set of non terminals. rep. by Cap letters those who are not NT are terminals. (except ε → this is a part of production rule)

N = {P, Q, D}
T = {a, b, ε, f}
S = P

- If Starting symbol isn't given, the grammer is wrong.

formation of lang → this need strings

a → P X → there should be
P → a ✓ atleast one Non terminal
A → a ✓ at LHS, there
ab → PQad ✗ can be terminals
PAD → PQab ✓ non terminals with it.

- String: set of terminals (string doesn't hold N.T)

Derivation

P → aQb [the PR start with starting state is the 1st start of our grammer]
→ abDb [∵ aQ → abD] [Replace the Production rule of N.T]
→ abefb [∵ D → ef]

the mechanism of forming strings from Production rules is called derivation.

② grammer → A → aA, A → ε [ε is also terminal]

As the, starting state is A, we have to choose the production rule A → aA. We always have to choose that PR where LHS is our starting symbol.

A → aA
→ a [∵ A → ε]

A → aA
→ aaA [∵ A → aA]

⇒ a^+ ← ∴ {a, a², a³ ... ∞}

this can be written like
A → aA | ε as the LHS is exactly the same.

the process of by which we are getting a string is called derivation.

③ $S \to aB$

$B \to aB \mid bB \mid \epsilon$ (this is CFG) (Starting state aB (a meaning) action that any combination of a and b aB|bB|ε)

derivation

$S \to aB$ | $S \to aB$, $B \to \epsilon$

$\to aB$ [$B \to aB$]

$\to abB$ [$B \to bB$]

$\to ab$ [$B \to \epsilon$]

$\therefore \{a, a^2, a^3, ab, ab^2, ab^3 \ldots\}$

lang (R.L) = Starting with a: [loop will give us closure]

$a(a+b)^*$

④ $a^+ \to$ R.L

$A \to aA \mid a$

$S \to aB \mid A$

$B \to a$

$a^+ \to RL$

$A \to aA \mid \epsilon$

⑤ $a(a+b)^+ \to$ Reg. L

$S \to aB$

$B \to aB \mid bB \mid a \mid b$

actually this forming [extra]

⑥ $S \to asb \mid ab$

derivation's tree

aabb → string

$S \to asb$

$\to aasbb$

$\to aaasbbb$

$\to aaabbb$

$\therefore$ lang$_L$ $\{a^n b^n / n \geq 1, \Sigma = \{a, b\}\}$ (CFL)

⑦ $S \to asb \mid \epsilon$

$\therefore$ lang$_L$ $= \{a^n b^n \mid n \geq 0, \Sigma = \{a, b\}\}$ (CFL)

Regular grammer: Imp

collection of Production rules (def. grammer)

i) LHS, only 1 non terminal (no terminals)

ii) RHS $(N \cup T)^*$ [any combination of NT & T]

$\therefore N \to (N \cup T)^*$

$|N| = 1$

iii) Either left linear or right linear but not both.

Imp $A \to Aa \mid \epsilon$ → left linear [the recursions are at left side]

Imp $A \to aA \mid \epsilon$ → Right linear [the recursions are at right side]

A → aA | Ab | ε ✗

A → aA | bA | ε ✓ (Right linearity is more preferable)

(8) abb

A → aB → (too a NT)

B → bB | ε

(m) A → aB
B → bD
D → b

(9) abb(a+b)*

A → aB
B → bD
D → bP
P → aP | bP | ε

| abb(a+b)*
A → abbB
B → aB | bB | a | b

(m) better
need formation of * so a new NT
S → abbA
A → aA | bA | ε
this is loop (actually for forming (a+b)*)

(16) (a+b)* ab

A → aA | bA | ab
~~A → ab~~

(m) S → A ab
A → Aa | Ab | ε

(a+b)+ ab
A → aA | bA | aab | bab

(m) S → Aab
A → Aa | Ab | a | b

(5) ab(a+b)+

S → abD
D → aD | bD | a | b

ab(a+b)*
S → abA
A → aA | bA | ε

(11) a* b*

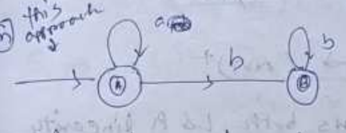
S → aA
A → bA | ε

[(11111)* → 1 ... 5 states
(111+1111)* → 2 states,
from where the repeating is it's acceptance is starting total no. of States will be → that state + 1]
best way to do it → draw diagram / machine

(12) a* b*

~~S → SA~~
~~S → aA | ε~~
~~A → bA | ε~~

(m) this approach

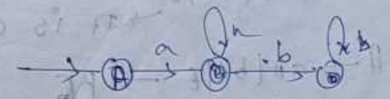


A → aA | bB | ε
B → bB | ε

S → AB
A → aA | ε
B → bB | ε

(13) a+ b*

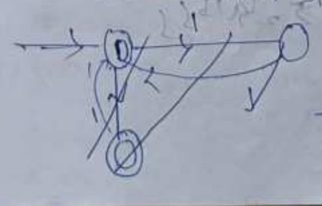
S → AB
A → aA | a
B → bB | ε

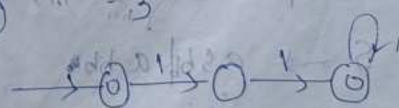


A → aB | ~~bB~~ | ~~a~~
B → ~~bB | ε~~ aB | bD | ε
D → bD | ε

(14) Σ = { (11+111)* | Σ = {1} }

⇒ { ε, 11, 111, 1111, 11111 ... ∞ }
2 states ∴ no. of states → 2 + 1 = 3





$A \to 11A \mid 111A \mid \epsilon$

11/10/23

CFL

↓

CFG (N, T, P, S)

$N \to (NUT)^+$

follows both LG A linearity

$(a+b)^*$

$A \to aA \mid Ab \mid \epsilon$ → this isn't a grammar of regular 1.

$A \to aA \mid bA \mid \epsilon$ → It is CFG (there is no such restriction for CFG)

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

grammar,

$S \to asb \mid ab$

② $L_2 = \{a^n b^{2n} \mid n \geq 1, \Sigma = \{a, b\}\}$

$S \to asbb \mid abb$

③ $L = \{a^{n+2} b^n \mid n \geq 1, \Sigma = \{a, b\}\}$

$S \to aasb \mid aaab$

④ $L = \{a^n b^m \mid n \geq m, n, m \geq 1, \Sigma = \{a, b\}\}$

$S \to asb \mid as \mid as$

a can be greater than no of b so

$[a^n b^m \mid n, m \geq 1, \Sigma = \{a, b\}]$ → regular

$[a^n b^m \mid n, m \geq 1, \Sigma = \{a, b\}]$ → this is PDA

$\approx [a^n b^n]$ [for equal ones asb]

⑤ $L = \{a^n, b^m \mid n > m, n, m \geq 1, \Sigma = \{a, b\}\}$

$S \to asb \mid aab \mid as$

this is one 'a' is extra

bs for it this isn't happen many a's will be extra

$S \to asb \mid aab$

$A \to aA \mid a$

④ Other ways;
Best way → as it is increasing count of A. So a is in 1st place so AB

S → AB
B → aBb | ab
A → aA | a

Exact (common)
② L = {a^n b^m | n ≤ m, n, m ≥ 1, Σ = {a, b}}

S → aBA → as it's for increase b's count & as b is behind A so BA
B → aBb | ab
A → bA | b

another way, otherway
S → asb | aAb
A → Ab | ∈

S → asb | aAb
A → aA | ∈

S → asb
→ aasBb
→ aaaAbbb
→ aaaabbb
→ aaaaAbbbb
→ aaaaabbbb

S → aAb

if S → asb | A } sb, this is wrong
A → aA | a

S → asb
→ a^4 S b^4
→ a^4 Ab^4
→ a^4 ab^4

S → aa ✗ this is wrong
→ a

PDA

⑦ L = {a^n b^m | n ≠ m, n, m ≥ 1, Σ = {a, b}}
Imp

S → A B
A → aA | a
B → aBb | ab
S2 → BD
D → bD | b
S → S1 | S2

n > m or n < m
doing it non equal

S1	S2
a^n b^m, n ≠ m	a^n b^m, n < m

⑧ L = {a^n b^m c^m d^n | m, n ≥ 1, Σ = {a, b, c, d}}
Imp
≈ a^n b^m c^m d^n

S → asd | aAd
A → bAc | bc

a & d's counting is same & also before b & after C there is a & d.

⑤ L = {a^n b^{n+m} c^m | m, n ≥ 1, Σ = {a, b, c, d}}

a^n b^{n+m} c^m
a^n b^n b^m c^m

S → asdb | abA
A → bAc | bc

⑨
S → S1 . S2
S1 → aS1b | ab
S2 → bS2c | bc

→ NCFL

(10) L = {WW^R | W ∈ string formed over (a, b)}

Sol (even palindrome)

S → S₁ S₂ [ε isn't a palindrome string] [so terminating cond. added]

S₁ → a S₁ b | ab

S₂ → b S₂ a | ba

abba, aabbaa, ababbaba ...

S → a S a | b S b | bb | aa

(11) L = {WCW^R | W ∈ string formed over (a, b)}

Imp (odd length palindrome)

S → a S a | b S b | C

RLG ⊆ CFG

We can simplyfy the grammer.

Imp Rmp

E → E + E | E * E | a

2 + 3 * 5 = ?

a + a * a = ?

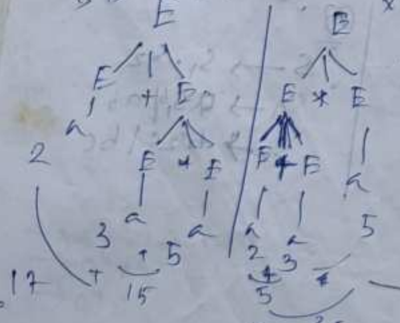
LM (Left most) derivation

E → E + E
→ a + E [∵ E → a]
→ a + E * E [∵ E → E * E]
→ a + a * E

(As we have expanded the left most NT.)

(In derivation at 1 time there ...)

↳ this is correct ans

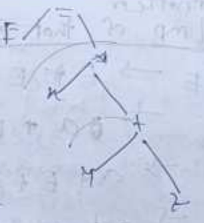


✗ this is not correct → a + a * a ans

← this is incorrect grammer but here there are ambiguities (more than 1 distinct derivation tree)

tree like structure —

E = x + (y + z)



Here we can also see that this 2 trees are giving 2 diff. results. though we know that 1st one is correct but it isn't known to the grammer. So, this grammer is ambiguous.

31/10/23

Cl

S → abc | aAabc

A → aa

derivation → {abc, aaabc}

Rm (right most) derivation

Imp

E → E + E
→ E + E + E [E → E + E]
→ E + E + a [∵ E → a]
→ E + a + a
→ a + a + a

(this explanation is not that was for this, the derivation tree will be ambiguous)

Ambiguous grammer — to generate 1 same string from 1 same grammer if we find — more than 1 LMD

or

more than 1 RMD

or

1 LMD & 1 RMD

(or) Imp To generate 1 same string from 1 same grammer if we find — some more than 1 distinct derivation tree.

(or) From any grammer if we can construct any string by more than 1 distinct derivation tree, then it is ambiguous grammer.

another LMD of that same grammer

$E \to E+E$ (1 more LMD)
$\to a+E$ [$\because E \to a$]
$\to a+E+E$ [$\because E \to E+E$]
$\to a+a+E$ [$\because E \to a$]
$\to a+a+a$

$E \to E+E$
$\to E+E+E$
$\to a+E+E$
$\to a+a+E$
$\to a+a+a$

RMD — same LMD & RMD

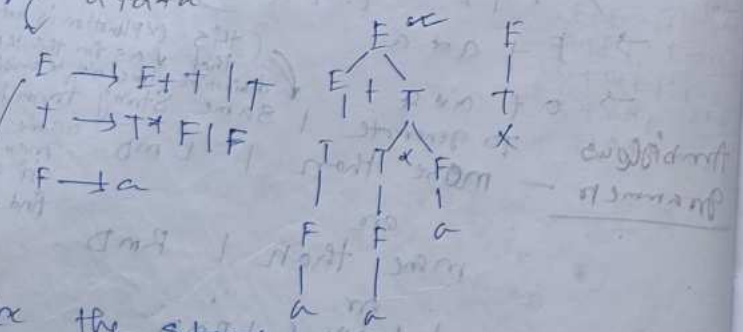
$E \to E+E$ (1 more LMD)
$\to E+E+E$
$\to E+E+a$
$\to E+a+a$
$\to a+a+a$

$E \to E+E$
$\to E+a$
$\to E+E+a$
$\to E+a+a$
$\to a+a+a$

① Ambiguity

$E \to E+E \mid E*E \mid a$
$a+a*a$

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to a$



Here the starting state is E
So, first it will calculate '+' then in power level there will be '*' so it will do mul after '+'. So, it's non-ambiguous.

CSG (TYPE - 1)

A can't contain ε

$(N \cup T)^+ \to (N \cup T)^+$

eg ① $S \to abc \mid aAbc$
$Ab \to bA$
$bA \to bb$

CFG ⊆ CSG

CSG is a formal grammer that allows the left & right sides of P rules to be surrounded by ε & NT symbols (the combination of ε & NT symbols)

a Ab c
a bA c
a bb c

we have to check the context before placing (checking NT as well as T before & after NT). So it is context sensitive.

② $S \to aAb$
$aAb \to b$

Unrestricted grammer (type - 0)

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

$S \to aAbc \mid abc$
$A \to b$

$= \{abc\}$

$S \to aAbc$
$Ab \to bA$
$Ac \to Bbcc$
$bB \to Bb$
$aB \to aa \mid aaA$

$S \to aAbc$
$\to abAc$
$\to abBbcc$
$\to aBbbcc$
$\to aabbcc$

HW

(2) L = { WCW | W ∈ string formed by Σ = {a, b} }

(3) L = { WW | W ∈ string formed by Σ = {a, b} }

(2) L = WCW

S → ab | ba

(2) S → aA | bB
A → aA | bB | X
B → aB | bA | Y
Xa → aX
Xb → bX
X → c
Yb → bY
Ya → aY
Y → c

(3) S → aSa | bSb | ε
S → A
Aa → aA
Ab → bA
AX → Xa
BX → Xb
Xa → aX
Xb → bX
X → ε

• Derivation tree = Parse tree

Left recursion (removal)

A → AaB : (NT) of left side is starting state. 1st NT of Right side.

E → E + T | T → left recursion
T → T * F | F → left recursion
F → a → no left recursion

Left recursion removal is converting a left recursion grammar into right recursive grammar.

(1) A → Aα | β

Soln
A → βA'
A' → αA' | ε
no left recursion

(2) A → Aα₁ | Aα₂ | Aα₃ | β₁ | β₂ | β₃

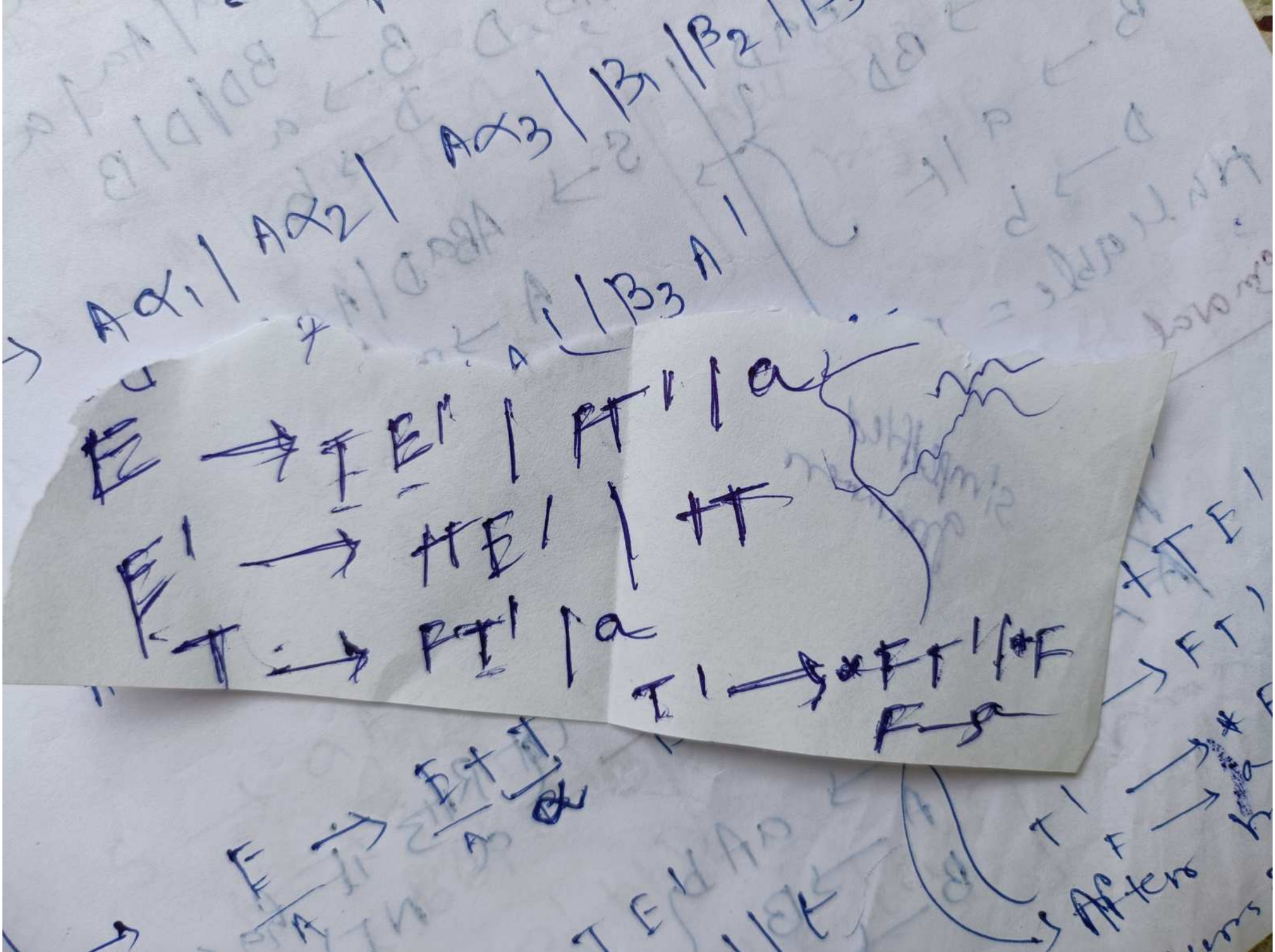
A → β₁A' | β₂A' | β₃A'
A' → α₁A' | α₂A' | α₃A' | ε

E → E + T | T

E → TE'
E' → +TE' | ε
T → T * F | F
T → FT'
T' → *FT' | ε

∴ E → TE'
E' → +TE' | ε
T → FT'
T' → *FT' | ε
F → a

After removing left recursion.


Aα1 | Aα2 | Aα3 | B1 | B2 | B3 A |
E → TE' | FT' | a
E' → +TE' | +T
T → FT' | a
T' → *FT' | /F | a

Simplification of any grammer — 1) not ambiguous is removed Left recursion is removed Left factored

② Left factored : (at RHS there is a common combination of NT & T)

① $S \to ieS \mid ieses$

$S \to iesS'$

$S' \to es \mid \epsilon$

[ies = Common]

③ Null Production removal (at RHS, it has a null)

① $S \to ABaD$, $A \to BD$, $B \to a \mid \epsilon$, $D \to b \mid \epsilon$

$S \to ABaD \mid BaD \mid AaD \mid ABa \mid aD \mid Ba \mid Aa \mid a$

$A \to BD \mid D \mid B$

$B \to a$

$D \to b$

nullable NTS = AB, D

② $S \to ABaD$, $A \to BD$, $B \to a \mid \epsilon$, $D \to b$

$S \to ABaD \mid AaD$

$A \to BD \mid D$

$B \to a$

$D \to b$

Nullable = B

⑤ Removal of unit Production (at RHS it has a NT)

① $A \to D$, $A \to B$, $S \to aAb$, $B \to d$

$S \to aAb$

$A \to B \mid d$

$B \to d$

[unit removal Production] [③ Removal of useless Production]

⑥ Removal of useless Production

If there is a Production rule which has no meaning in that grammer remove it

① $S \to aAD$, $A \to d$, $B \to d$

② $S \to aAb$, $A \to d \mid b$, $B \to d$

Simplify any grammer (use ④, ⑤, ⑥) (by default)

① $S \to aA \mid aBB$ (this is basically for (FOR))

$A \to aaA \mid \epsilon$

$B \to bB \mid bbD$

$D \to B$

Nullable = A

① $S \to aA \mid aBB \mid a$

$A \to aaA \mid aa$

$B \to bB \mid bbD$

$D \to B$

[null Production remove]

② $S \to aA \mid aBB \mid a$

$A \to aaA \mid aa$

$B \to bB \mid bbD$

$D \to bB \mid bbD$

[removal of unit Production]

③ $S \to aA \mid aBB \mid a$

$A \to aaA \mid aa$

[removal of useless Production]

Simplified grammer,

S → aA|a
A → aaA|aa

} we can use left recursion here, but it isn't that much needed.

~~Normal form~~ → this is also for CFG

CNF & GNF

↓ this format

A → BD
A → a

A → aX (X ∈ NT)

[X = X is combination of NTs]

- To convert into CNF & GNF we need to use ④, ⑤, ⑥.

① A → aBaD ← CFG → CNF

B → a|PD
D → a
P → b

Apply all 3 simplifications 1st.

⇓ Converting it into CNF

A → QBQB | mQD | RD
Q → a
B → a|PD
D → a
P → b
M → QB
R → mQ

A → RD
R → mQ
M → QB
Q → a
B → a|PD
D → a
P → b (now this is in CNF)

How many no. of NTs & Prod. rules are reduced to make it CNF.

② S → abcde

S → AXCDE (CFG)

n + (n − 1) > 2n − 1 → no. of " . . . .

- what is GNF is already CNF. But vice versa isn't correct.

CFG → GNF (conversion is much complex)

① S → XB | AA
A → a | SA
B → b
X → a

[If we have to make GNF then we must have to do ④, ⑤, ⑥ CNF. If ④, ⑤, ⑥ isn't needed then only CNF is needed & it's CNF then just convert.] this already in

this is CNF already, now making it GNF

S → aB | AA
A → a | aBA | XA
B → b
X → a

[give any one NT, it's value, i.e., put the value of it to it's corresponding NT - but for any 1 NT]

S → aB | AA
A → aR | aBAR
R → aAR | ε
B → b
X → a

A' · R [left recursion removal ①]

⇓

S → aB | AA
A → aR | aBAR | a | aBA [Null Prod. removal ③]
R → AAR | AA
B → b
X → a

⇓ making it in GNF format

S → aB | aRA | aBARA | aA |

A → aR | aBAR | a | aBA

R → aRAR | aBARAR | aAR | aBAAR |
aRA | aBARA | aA | aBAA
B → b
X → a

Type of langs (3 | 2 | none of these) 7/11/23

① L = {a^p | p is prime no., Σ = {a}}

② L = {$a^n b^m$ | n, m ≥ 1, Σ = {a, b}} → CFL (PDA)

③ L = {$n_a(a) = n_b(b)$ | Σ = {a, b}} → RL (PDA) [Infinite lang]

④ L = {$a^n b^{n+m} c^m$ | n, m ≤ 10^5, Σ = {a, b, c}} → CFL (PDA) ↳ (finite lang ⇒ R.L)

Mealy machine & moore machine

6 tuples (for both machines)

- DFA = δ : Q × Σ → Q
- NFA = δ : Q × Σ → 2^Q
- ε-NFA = δ : Q × (Σ ∪ ε) → 2^Q or 2^Q
- DPDA = δ : Q × Σ × Γ → Q × Γ*
- NPDA = δ : Q × Σ × Γ → $2^{Q × Γ*}$
- TM = δ : Q × Σ → Q × X × {L, U, R}

(5 tuples for DFA, NFA, ε-NFA; 7 tuples for DPDA, NPDA, TM)

[δ(1, a, Z₀) → (1, AA)]
[At LHS only i/p, At RHS state symbols]

(i/p state) (i/p) (o/p state)

→ (Q, Σ, O, δ, q₀, X) → there is no final state

- O: set of output symbols
- δ: state transition func
- X: o/p transition function

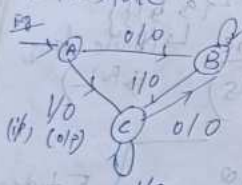
Mealy m – It is a finite state machine whose o/p values are determined by both prev state & i/p taken on the state. Add ②, ③, ④.

Moore m – only by it's prev state. Add ②, ③, ④.

Mealy machine

① $X: S \times Z \rightarrow O$

output depends on the present state & the input taken on the state.



Transition table

present state / present state	i/p = 0 NS	o/p	i/p = 1 NS	o/p
→ A	B	0	C	0
B	B	1	C	0
C	B	0	C	1

② It doesn't influence next state

③ o/ps are shown on the transitions.

④ Less complex

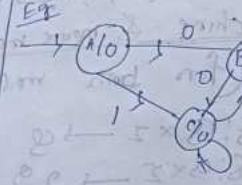
⑤ No. of states can be greater than or equal to Moore machine

Mealy machine can have greater no. of states than Moore machine, but the reverse isn't possible.

Moore machine

① $X: S \rightarrow O$

The o/p is independent on the present state.



Transition table

present state	next state i/p = 0	next state i/p = 1	o/p
→ A	B	C	0
B	C	B	1
C	C	B	0

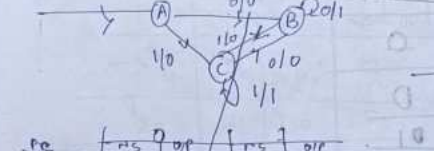
② It influences next state

③ o/ps are shown on the states

④ more complex

⑤ more no. of states

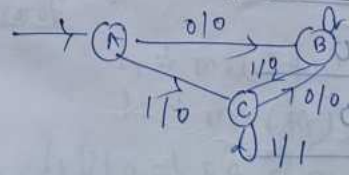
Convert moore machine into mealy machine



PS	NS i/p = 0	o/p	NS i/p = 1	o/p
→ A	B	0	C	0
B	C	0	B	1
C	C	0	B	1

PS	NS i/p = 0	NS i/p = 1	out
→ A	B	C	0
B	C	B	1
C	C	B	0

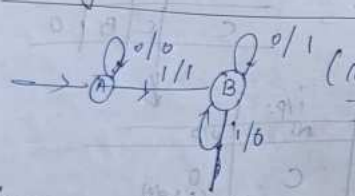
Convert mealy machine into moore machine



PS	NS i/p = 0	out	NS i/p = 1	out
→ A	B	0	C	0
B	B	1	C	0
C	B	0	C	1

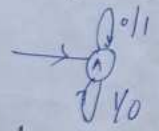
PS	NS 0	NS 1	output
→A	B_0	C_0	0
B_0	B_1	C_0	0
B_1	B_1	C_0	1
C_0	B_0	C_1	0
C_1	B_0	C_1	1

Binary → 2's Complement



(consider I/Ps from Right side)

Binary → 1's complement



Mealy → moore Conversion

PS	NS 0	out	NS 1	out
→q_1	q_1	0	q_2	0
q_2	q_2	1	q_3	0
q_3	q_2	0	q_3	1

PS	NS 0	NS 1	~~out~~ out
→q_1	q_1	q_{20}	0
q_{20}	q_{21}	q_{30}	0
q_{21}	q_{21}	q_{30}	1
q_{30}	q_{20}	q_{31}	0
q_{31}	q_{20}	q_{31}	1

21|11|23

Closure Property (Imp for gate) Not that much imp for exam

$a \in z$, $b \in z$, $a+b \in z$ → $a \& b$ (ints) are closed under '+'

For regular lang

Union

Prof L_1 & L_2 are reg. lang.

∴ then $L_1 \cup L_2$ is also RL (∵ this is closed under union)

Proof

$L_1 \Rightarrow m_1$ (FA)

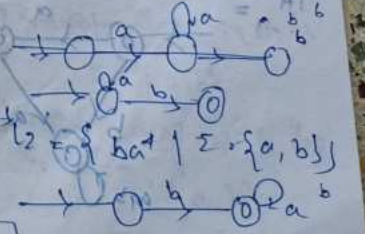
$L_2 \Rightarrow m_2$ (FA)

$L_1 \cup L_2 \Rightarrow$ FA also

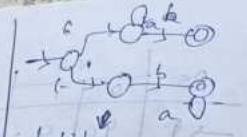
∴ as

∴ for Union operation the RL is closed (Proved)

$L_1 = \{a^*b \mid \Sigma = \{a,b\}\}$



$L_2 = \{ba^* \mid \Sigma = \{a,b\}\}$



2) Complement – if L is reg. then

L^c is also RL.

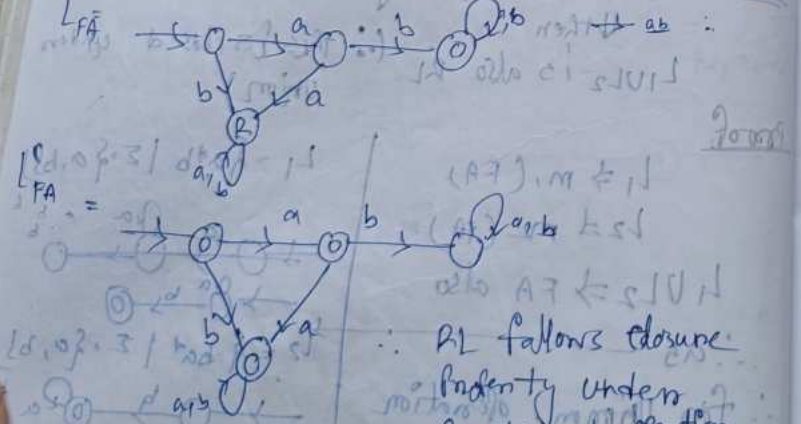
Proof As L is RL so a m(FA) will exist for that.

Let, L = ∀ strings starting with 'ab'.

L^c = Not any string starts with ab

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

From the FA of L we can produce a FA from L^c too. (write steps to make L^c from L)



∴ RL follows closure property under complement operation. (Proved)

- RL follows closure property for every operations (except subset operation)

3) Intersection – L_1 & L_2 are RL

⇓ then

$L_1 \cap L_2$ is also RL

Proof, given, L_1 is RL

L_2 is RL

$\overline{L_1 \cap L_2}$

$\overline{L_1} \cup \overline{L_2}$ (demorgan's law)

$(RL^c \cup RL^c) \in$ [∵ Union is RL's (welcome)]

$(RL^c) \in$

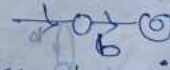
= RL (proved)

∴ RL follows closure property under intersection operation.

[$L_1 = \{a^n b \mid \Sigma = \{a, b\}\}$ → b, ab, aab...

$L_2 = \{ba^n \mid \Sigma = \{a, b\}\}$ → b, ba...]

$L_1 \cap L_2 = \{b\}$



4) Reversal – L^R = ∀ strings starting with 'ab'

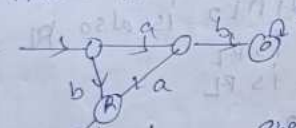
$\Sigma = \{a, b\}$

L^R = ∀ strings ending with 'ba'

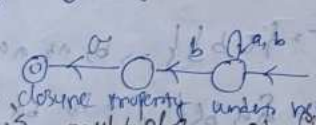
$\Sigma = \{a, b\}$

If L is RL $\xrightarrow{\text{then}}$ L^R is also RL.

If FA is exist for L then we can produce FA for L^R also from FA of L.

L_{FA} = 

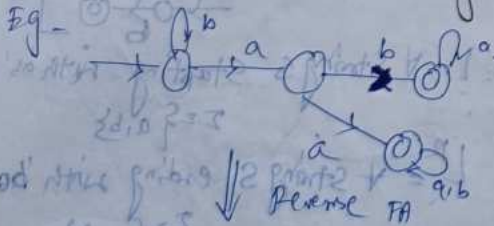
Steps → 1) reverse all directions
2) Starting state → F State
3) Final State → S State.
4) If R State is not treated (no note on the machine), then remove it

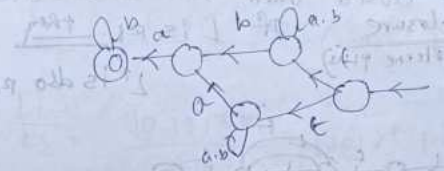
L^R_{FA} = 

L^R_{FA} =

∴ RL follows closure property under reversal operation.

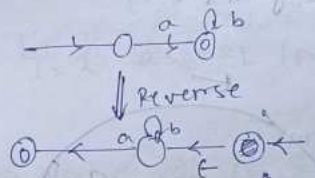
- If there is multiple finite state, Create a new state for Starting State & Connect all the final states with that new state (Starting state).

Eg- 



(1st type of making)

Eg - (making NFA by forcibly)

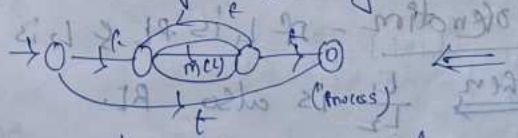


(2nd type of making)

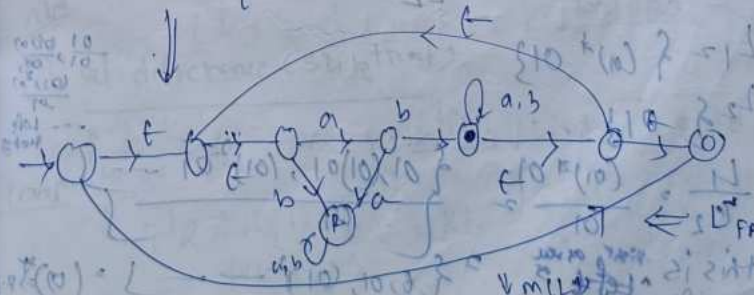
5) Kleene closure - If L is RL $\xrightarrow{\text{then}}$ L^* is also RL
(Kleene Star)

If L is (any)
If FA ~~is RL~~ exist for L then it will also (FA) exist for L^*.

L = all strings Starting with 'ab'.



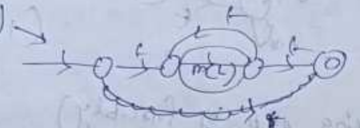
L_{FA} (Prev) m(L)



RL is closed under kleene closure (proved)

6) positive closure - If L is RL then L^+ is also RL
(kleene plus)

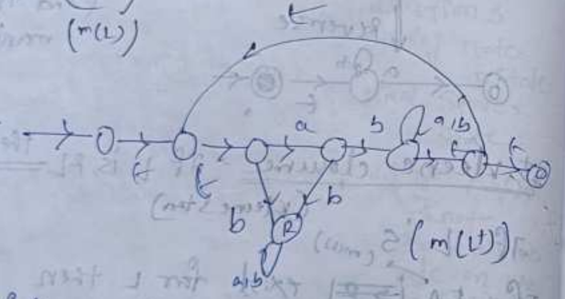
(process) →



L = ∀ strings starting with 'ab'

L_{FA} ⟹ (proc) (m(L))

L^+_{FA} ⟹



(m(L⁺))

∴ RL follows closure property under positive closure (proved).

7) Quotient operation - If L_1 is RL & L_2 is also RL then $\frac{L_1}{L_2}$ is also RL.

$L_1 = \{(01)^* 01\}$

$L_2 \{01\}$

$\frac{L_1}{L_2} = \frac{(01)^* 01}{01} = \frac{\{01, (01)01, (01)^2 01 \ldots\}}{01}$

$= \{\epsilon, 01, (01)^2 \ldots\} = (01)^*$

(this is right, as well as left quotient)

∴ RL is closed under quotient operation (proved)

$\frac{L_1}{L_2} = \frac{(011)^* 01}{01} = (011)^*$ by right quotient

$\frac{L_1}{L_2} = \frac{(011)^* 01}{01} = (101)^*$ right quotient

= ~~{01, (011)01, (011)² 01 ...}~~

= {ε} → left quotient → this is also a RL

∴ $\frac{L_1}{L_2}$ is also RL (proved)

• φ is also a reg. lang FA of φ = → O ◎

L_1 {match, math, ~~math~~ marth}

L_2 = {m}

R.Q. = φ L.Q. = {atch, ath, arth}

$\frac{L_1}{L_2} = \frac{(011)^* 01}{01} = \epsilon, 101, (101)^2 \ldots$

$= (101)^*$ → left quotient

22/11/23

8) set difference (subtract)

If L_1 & L_2 are RL then $L_1 - L_2$ is also reg.

Proof $L_1 - L_2 = L_1 \cap \overline{L_2}$ = RL

(RL ∩ RL^c)

RL is closed under substitution

(c) Homomorphism - $L = \{0,1\}$ [L is a lang]

(i) $L = \{0, 11\}; 010\}$

$H(0) = ab$

$H(1) = b^+$

$H(L) = \{ab, b^+, abb^+ab\}$ → this is RL.

(d) Inverse Homomorphism

$L = \{0, 1, 010\}$ ← this is inverse h. from the above eg's point of view otherwise it is h.

$L = \{ab, b^+, abb^+ab\}$

$h^{-1}(ab) = 0$

$h^{-1}(b^+) = 1$

$h^{-1}(L) = \{0, 1, 010\}$ → this is RL

• NO lang is closed under subset (not RL too).

$L = \{a^n b^n \mid \Sigma = \{a, b\}\}$ → RL

$L_1 = \{a^n b^n \mid \Sigma = \{a, b\}\}$ → not RL

CFL

(1) Union - If L_1 & L_2 are CFL then $L_1 \cup L_2$ is also CFL

• L_1 is CFL $S_1 \rightarrow$ } CFG of L_1

L_2 is " $S_2 \rightarrow$ } CFG of L_2

$L_1 \cup L_2$ CFG $S \rightarrow S_1 | S_2$

$S_1 \rightarrow$

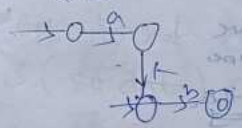
$S_2 \rightarrow$

$L_1 \cup L_2$ is CFL is closed under union.

(e) Regular Lang

(i) Concatenation - If L_1 & L_2 are reg then $L_1 L_2$ is also reg.

Proof



CFL same

2) Concatenation - If L_1 & L_2 are CFL then $L_1 \cdot L_2$ is also CFL.

As L_1 & L_2 both are CFL, so there exist a CFG for both. From those CFGs that we can construct CFG for $L_1 \cdot L_2$. So $L_1 \cdot L_2$ also will be CFL.

L_1 is CFL $S_1 \rightarrow$ } CFG of L_1

L_2 " $S_2 \rightarrow$ } CFG of L_2

$L_1 \cdot L_2$ CFG $S \rightarrow S_1 S_2$

$S_1 \rightarrow$

$S_2 \rightarrow$

CFL is closed under concatenation

3) Kleene Star:

L is CFL, so CF grammar of L^* $S \to S_1 S \mid \epsilon$ — $S_1 \to$ CFG of L

↓ this is CFL

CFL is closed under Kleene closure

4) Kleene Plus

L^+ .....

Grammar of L^+ $S \to S_1 S \mid S_1$

$S_1 \to$ .....

CFL is closed under positive closure

5) Reversal: If L is CFL then L^R is also CFL

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

$S \to aSb \mid ab$

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

$S \to bSa \mid ba$

CFL is closed under reversal.

6) CFL ~~complement~~ Intersection

$L_1 = \{a^n b^n c^m \mid n, m \geq 1\}$, $\Sigma = \{a, b, c\}$ → CFL

$L_2 = \{a^m b^n c^n \mid n, m \geq 1\}$, $\Sigma = \{a, b, c\}$ → CFL

$L_1 \cap L_2 = \{a^k b^k c^k \mid k \geq 1\}$, $\Sigma = \{a, b, c\}$ → this is not CFL

∴ CFL is not closed under CFL.

~~7) Complement~~ $L_1 \cap L_2$

7) Complement of $L_1 \cup L_2$: $= (L_1^c \cap L_2^c)^c$ [Suppose, L_1 & L_2 are closed under complement]

$= (CFL_1{}^c \cap CFL_2{}^c)^c$ [as CFL is closed under complement $CFL^c = CFL$]

$= (CFL_1 \cap CFL_2)^c$ [CFL is not closed under 'n'] = not CFL

CFL is not closed under complement

8) Set difference: $L_1 - L_2$

$L_1 \cap L_2^c$

CFL ∩ non CFL

↓ definitely not CFL [Intersection of CFL is not closed under 'n']

- CFL is not closed under I (intersection) S (set diff.) C (complement).

9) Quotient operation - CFL is closed under Quotient operation.

$L_1 = \{a^n b^m \mid n \geq 1, \Sigma = \{a, b\}\} \to CFL$

$L_2 = \{a^k b^{2k} \mid k \geq 1, \Sigma = \{a, b\}\} \to CFL$

$\frac{L_1}{L_2} = \{a^k b^k \mid k \geq 1, \Sigma = \{a, b\}\} \to$ also CFL

- ~~Union~~ DCFL
- Union $a^n b^n \to$ DCFL

$a^n b^{2n} \to$ DCFL

$a^n b^{2n} \cup a^n b^n \to$ NCFL not DCFL

∴ NCFL is closed under union but here, it is not NCFL, so DCFL isn't closed under union.

- Complement $L_1 \cap L_2$

$\Sigma_1 \cup \Sigma_2$

- DCFL ∪ DCFL [∵ are not closed under ∪]
= non DCFL

[Complement DCFL is closed under complement]

⇒ DCFL (proved)

[Chart by Sir of C.P. - Pmp]

Concatenation: the trail of string & the front of 2nd string is setting ...

→ eg $L_1 = \{a^n b^n \mid n \geq 1, \Sigma\{a,b\}\}$
$S_1 = \{ab \ldots\}$
$L_2 = \{b^n a^n \mid n \geq 1, \Sigma\{a,b\}\}$
$S_2 = \{ba \ldots\}$
$L_1 \cdot L_2 = abba$

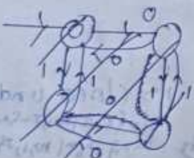
Qns from net

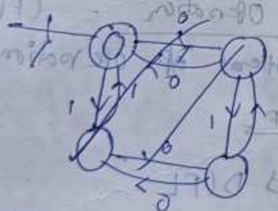
① write a reg ex for accepting all the string having 2 consecutive 0 & 2 consecutive 1.

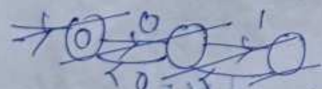
② make a dfa for accepting all st the string ending with 1 & having 2 consecutive 0s if 0 appears followed by 1 & write its reg ex.

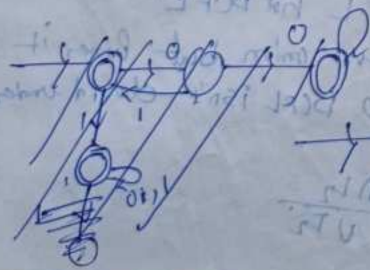
① strings = {ε, 0011, 1100, (00)(11)², (00)²11, 11 00 11, ... ∞}

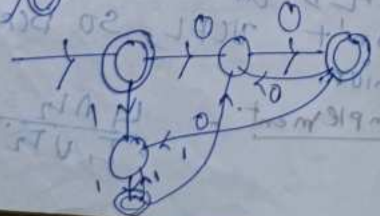
(00|11)



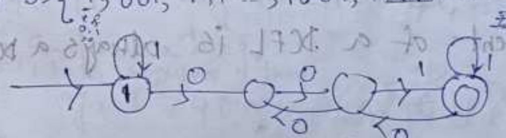








② S = {1, 001, 1², 1³ ..., 1001, 100011 ... ∞}



PYQS (TOC) - 2022

Q-A

1) a) DPDA is less Strong (powerful) (or lesser) in power than NPDA.

b) A grammer generates a language

c) A language is a collection of all possible strings over some alphabet.

d) Type 1 grammer are context sensitive grammer.

e) Stack in PDA has unlimited storing capability.

f) Complement of a DCFL is always a DCFL. T

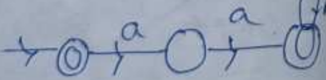
g) moore machine has 0 number of final states.

h) turing machines for recursive enumerable language never guarantee a reject for the strings not of language.

i) only the finite language are regular F

j) minimized DFA of $L:(aa+aaa)^*$ where $\Sigma=\{a\}$ has 3 number of states.

$\{\epsilon, aa, aaa, a^4, a^5, \dots \infty\}$



Q-B

2) What is derivation? Explain RMD & LMD. When a grammer is called ambiguous grammer? Give an example & explain.

Done,

LMD - Here, the given i/p is scanned & then replaced with the production rule from left side to right.

RMD - - - - - - - - r to l.

(Eg & derivation - Done for both)

Done, Done

3) Using pumping lemma prove that $L: \{a^p \mid p \text{ is any prime number}, \Sigma=\{a\}\}$ is not a regular lang.

Suppose, L is regular lang.

Lets say, $P = 4$ [$\because |w| \geq P$], $w = a^5 \in L$ [let $p=5$]

$w = a\ aaa\ a$ ($x\ y\ z$)

(i) $|xy| \leq P$

$4 \leq 4$

$\therefore$ It is satisfied

(ii) $|y| \geq 1$

$3 \geq 1$

$\therefore$ It is satisfied

(iii) let, $i=2$, $xy^iz = xy^2z = a(aaa)^2a = a^8 \notin L$

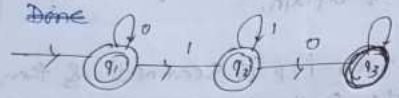
let $i=4$, $xy^iz = xy^3z = a(aaa)^4a = a^{14} \notin L$

$\therefore$ It isn't satisfied

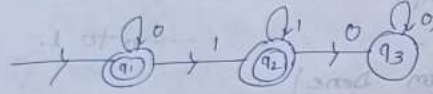
($\therefore$ This is not a reg lang (proved))

4) Write Arden's theorem. Using Arden's theorem find Regular Expression for the finite automata given below.

~~Done~~



Done



$q_1 = \epsilon + q_1 \cdot 0$ — (I)

$q_2 = q_1 \cdot 1 + q_2 \cdot 1$ — (II)

$q_3 = q_2 \cdot 1 + q_3(0+1)$ — (III)

$\boxed{q_1 = 0^*}$ — (IV)

$q_2 = 0^* 1 + q_2 \cdot 1$ [from (IV)]

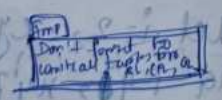
$\Rightarrow \boxed{q_2 = 0^* 1^*}$ — (V)

$\therefore$ Reg ex $= (0^* + 0^* 1^*)$ or $0^* (\epsilon + 1^*)$

5) Construct a minimized DFA for a lang given below -

L: All strings, not having '101' substring, over alphabet $\Sigma = \{0, 1\}$

Done



6) Prove that - "Context free languages are closed under concatenation".

Done

7) Construct a turing machine for the language below -

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

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

Same as: $L = \{a^n b^n \mid n \geq 1 \ldots\ldots\}$

$a^m b^m \mid m \geq 1 \ldots\ldots\}$

Rest = Done

8) Simplify the following context free grammar, where S is starting symbol, terminals are $\{b, d, e, f\}$

$S \rightarrow ABCd$

$A \rightarrow Bc$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow Pc \mid \epsilon$

(1) Null production removal

Nullable = A, B, C

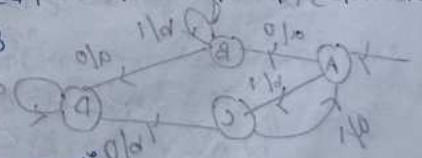
$\therefore$ After null production removal the grammar is

$S \rightarrow ABCd \mid BCd \mid Acd \mid ABd \mid cd \mid Ad \mid Bd \mid d$

$A \rightarrow Bc \mid c \mid B$

$B \rightarrow bB \mid b$

$C \rightarrow Pc \mid P$



(2) Removal of unit production

$\therefore$ After removal of unit prod. the grammar is

$S \rightarrow ABCd \mid BCd \mid Acd \mid ABd \mid Cd \mid Ad \mid Bd \mid d$

$A \rightarrow Bc \mid Pc \mid P \mid bB \mid b$

B → bB | ∈ b
C → pC | p

③ Removal of useless prod.

After removing useless prod. the grammar is [there is no unit/ε-production rule]

S → ABCd | BCd | ACd | ABd | Cd | Ad | Bd | d
A → BC | pC | p | bB | b
B → bB | b
C → pC | p

Ans-c

9) Consider the lang. below to answer the following questions-

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

a) Construct a PDA which accepts L.

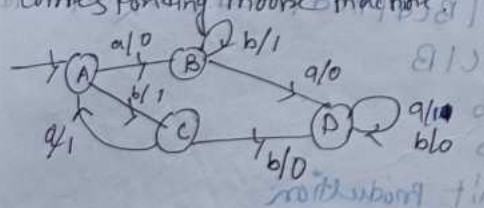
b) Write CFG which generates L.

a) Done, b) Done

10) a) Write a note on moore & mealy machine & highlight the basic differences.

Done

b) Convert the mealy machine (given below) into its corresponding moore machine



transition table of mealy machine

PS	a		b	
	NS	O/P	NS	O/P
→ A	B	0	C	1
B	D	0	B	1
C	A	1	D	0
D	D	1	D	0

← modify machine

transition table of moore machine

PS	NS		O/P
	a	b	
→ A_0	B_0	C_1	0
A_1	B_0	C_1	1
B_0	D_0	B_1	0
B_1	D_0	B_1	1
C_1	A_1	D_0	1
C_0	A_1	D_0	0
D_1	D_1	D_0	1
D_0	D_1	D_0	0

11) a) ~~Longest~~ Construct a Turing machine to accept the language give below-

L: {WW | where 'W' means strings formed over inputs {a, b}} Done

b) Is it a deterministic or non deterministic TM?

12) A regular exp is given for a lang. Done

L: $ab(a+b)^*a$ where $\Sigma = \{a, b\}$

a) Construct ∈-NFA for L. Done

b) Convert ∈-NFA to DFA. Done

c) Write a Right linear grammar for L.

$ab(a+b)^+a$

$S \rightarrow abBa$

$B \rightarrow aB|bB|\epsilon$

14) a) Construct a minimized DFA for the lang.-
L: All strings, where last two symbols are same, over alphabet $\Sigma = \{0,1\}$ Done

b) write context free grammar for the lang- L: $\{WcW^R \mid$ where 'W' means strings formed over inputs $= \{a,b\}$, 'c' is a terminal and W^R is the reverse of W$\}$ Done

12) Write short notes on followings:-

a) CNF (Chomsky's Normal form) - A CFG is in CNF if all production rules satisfy one of the following conditions:

i) Start symbol generating ϵ for eg, $A \rightarrow \epsilon$

ii) A non-terminal generating two n-t, for eg $S \rightarrow AB$

iii) A n-t generating a terminal, for eg $S \rightarrow a$

Eg i) $G1 = \{S \rightarrow AB,\ S \rightarrow c,\ A \rightarrow a,\ B \rightarrow b\}$

It is satisfying (ii), (iii) so it's CNF.

ii) $G2 = \{S \rightarrow aA,\ A \rightarrow a,\ B \rightarrow c\}$

But it isn't satisfying the rules mentioned, so it isn't CNF.

b) GNF (Greibach NF) - A CFG is in GNF iff the RHS of each route has 1 terminal followed by zero or more n-ts

eg- $A \rightarrow aB$, where $a \in T$ & $B \in N^*$.

For converting given a grammar into GNF we take the help

But, if $A \rightarrow aaB$, $a \in T$ & $B \in N^*$ it is NOT GNF.

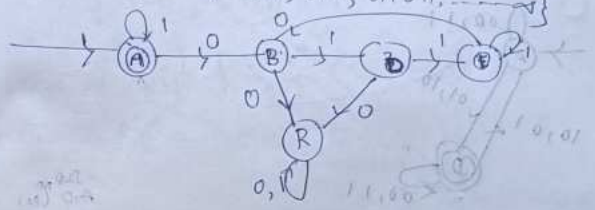
c) Removal of left recursion, d) CSG Done, Done

TOC (CT) → 2023

Q1) Construct minimized DFA for -

L: All strings where every occurrence of '0' is followed by '11', $\Sigma = \{0,1\}$ (min 11)

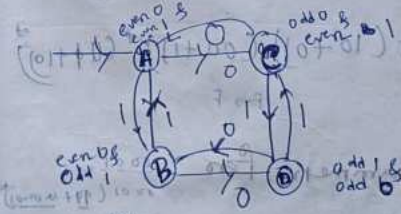
strings = $\{\epsilon, 1, 11, 011, 011011, \ldots\}$



Q2) Construct minimized DFA for -

L: All strings with odd no. of 0's & odd no. of 1's, $\Sigma = \{0,1\}$ (no. of 0 & 1 is divisible by 2)

strings = $\{01, 10, 000111, 010101 \ldots\}$



Q3) Write Regular expression for the language in G_2. ~~Using shortcut method~~

$A = \epsilon + C \cdot 0 + B \cdot 1$

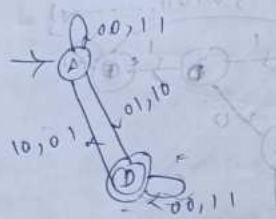
$B = A \cdot 1 + D \cdot 0$

$C = A \cdot 0 + D \cdot 1$

$D = B \cdot 0 + C \cdot 1$

But by using Arden's theorem getting the such of this is little lengthy. So we are using shortcut method (drawing paths & circuits).

So, by using shortcut we get,



The reg ex = ~~(00+11)* 01~~

= ~~(00+11 + (10+01)(00+11)* (01+10)~~

~~otherwise 01, 000111, 10, 111,~~

$= (00+11)^*(01+10)(00+11+(10+01)(00+11)^*(01+10))^*$

Write Regular grammer for ~~00 01~~ lang ing.

Reg ex- By applying Arden's theorem we get

$A = \underset{R}{\epsilon} + \underset{Q}{A} \cdot \underset{P}{1}$ — (I)

$B = A \cdot 0 + E \cdot 0$ — (II)

$D = B \cdot 1$ — (III)

$E = D \cdot 1 + E \cdot 1$ — (IV)

$R = B \cdot 0 + D \cdot 0 + B(0+1)$ — (V)

$E = D \cdot 1 \cdot 1^*$ — (VI)

$D = B \cdot 1$ — (VII)

$B = A \cdot 0 + E \cdot 0$ — (II)

$A = 1^*$ — (VIII)

Putting (VI), (VIII) in (II)

$B = 1^* \cdot 0 + D \cdot 1 \cdot 1^* \cdot 0$ — (IX)

Putting (VII) in (IX)

$B = 1^*0 + B \cdot 1 \cdot 1 \cdot 1^* 0$

$\therefore B = 1^*0(111^*0)^*$ — (X)

Putting (X) into (VII)

$D = 1^*0(111^*0)^*1$ — (XI)

Putting (XI) into (VI)

$E = 1^*0(111^*0)^*111^*$

Reg ex $= 1^*0(111^*0)^*111^*$

$S \to A\,1\,B\,C$

$A \to$

RG → $S \to A0B \mid 1A0$

$A \to 1A \mid \epsilon$

$B \to 11A0B \mid \epsilon$

$A = A1^* + \epsilon \cdot 0$ — (VII)

$A = \epsilon + A \cdot 1$ — (I)

$B = A \cdot 0 + E \cdot 0$ — (II)

$D = B \cdot 1$ — (III)

$E = D \cdot 1 + E \cdot 1$ — (IV)

$R_2 = B \cdot 0 + D \cdot 0 + R(0+1)$ — (V)

$\boxed{A = 1^*}$ — (VI)

$E = B \cdot 1 \cdot 1 + E \cdot 1$ [By (II)]

$= (A \cdot 0 + E \cdot 0)11 + E \cdot 1$ [By (II)]

$= A011 + E011 + E \cdot 1$

$= A \cdot 011 + E(011+1)$

$E = A011(011+1)^*$

$\boxed{E = 1^*011(011+1)^*}$ [By (VI)]

Reg R - $1^* + 1^*011(011+1)^*$

$S \rightarrow A + B$

$A \rightarrow 1A \mid \epsilon$

$B \rightarrow A011 \mid E$

$E \rightarrow 011E \mid 1E \mid \epsilon$

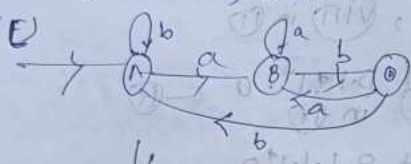
Q5) Find Regular expression for -

L: All strings, not ending with 'ab'

$\bar{L}$: All strings ending with 'ab'

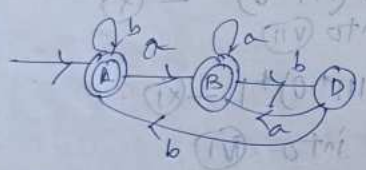
Regex - $(a+b)^*ab$

DFA (E)



Complement

DFA (F)



applying Arden's theorem we get,

$A = \epsilon + A \cdot b + D \cdot b$ — (I)

$B = A \cdot a + B \cdot a + D \cdot a$ — (II)

$D = B \cdot b$ — (III)

$D = B \cdot b$ — (IV)

Putting (III) on (II)

$B = A \cdot a + B \cdot a + B \cdot b \cdot a$

$= Aa + B(a + ba)$

$\therefore B = A \cdot a (a+ba)^*$ — (IV)

$A = \epsilon + A \cdot b + B \cdot b \cdot b$

$= \epsilon + A \cdot b + A\, a(a+ba)^* bb$

$= \epsilon + A(b + a(a+ba)^* bb)$

$\therefore A = (b + a(a+ba)^* bb)^*$ — (V)

putting (V) on (IV)

$B = (b + a(a+ba)^* bb)^* a(a+ba)^*$

$\therefore$ Regex - $(b + a(a+ba)^* bb)^* +$

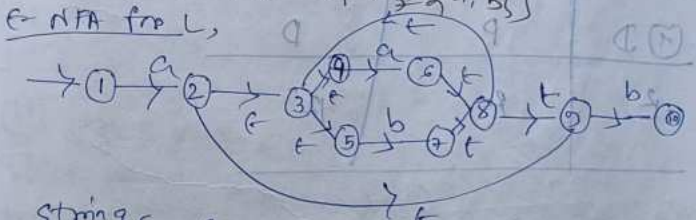
$(b + a(a+ba)^* bb)^* a(a+ba)^*$

$= (b + a(a+ba)^* bb)^* (\epsilon + a(a+ba)^*)$

Q6) Construct ϵ-NFA for -

$L = \{ a(a+b)^* b \mid \Sigma = \{a, b\} \}$

ϵ-NFA for L,



strings - {ab, aa²b, aabb ... ∞}

Q7) Convert ϵ-NFA of Q6 into DFA.

DFA transition table →

St	a	b
→ {1} A	{2,3,4,5,9} B	ϕ
{2,3,4,5,9} B	{3,4,5,6,8,9} C	{3,4,5,7,8,9,10} D
{3,4,5,6,8,9} C		

{3,4,5,6,8,9} [C]	{4,5,6,8,9} [C]	{3,4,5,7,8,9,10} [D]
(*) {3,4,5,7,8,9,10} [D]	{3,4,5,6,8,9} [C]	{3,4,5,8,9,10} [D]
R	P	Q

minimization –

A B C D R

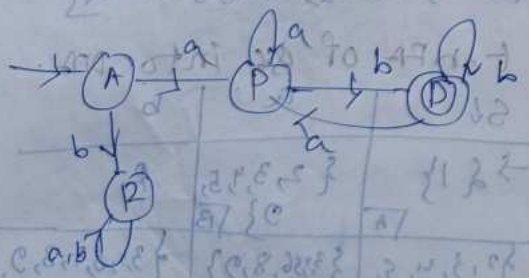
= (A B C) (D) (R)

= (A) (P) (D) (R)

Renamed table –

SN	a	b
→ A	P	Q
P	P	D
(*) D	P	D
R	R	Q

∴ DFA –



By considering it (3,6) NFA we will also get the same ans.

PYQ (Jun) – 2021

Ans A

i) Choose correct alternative –

i) $L = a^{n!} \mid 1 \le n \le 1000$ – This is a reg. lang.

ii) Which of the following defines string 'w' of lang. that can be generated by the following CFG?

$S \to xy$

$x \to ax \mid ba \mid a$

$y \to ya \mid yb \mid a$

w has atleast two consecutive a's

iii) Consider the pairs of reg ex given below –

(I) $0^*(10^+)^* \& (1^*0)^+(1^+)$

(II) $(b^* + 5^*) \& (b+5)^*$

(III) $(a^+ + b)^* \& (a+b)^*$

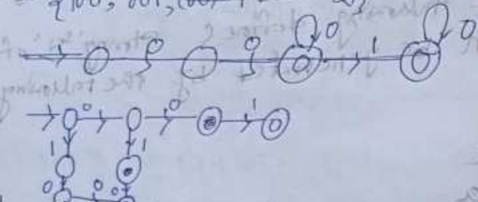
(IV) $(pq)^* p \& p(qp)^+$

Which of the above pairs represent equivalent reg ex.

(II) & (III) only

iv) The no. of states in minimal DFA which accepts all strings that contain at least 2 0's & at most one 1 over the alphabet = {0,1} is – 8 7 6

S- {100, 001, (00)^2 + ---- ∞}



v) Which of the following is context free lang.?

$L1$ = Complement of $\{a^i b^j c^k \mid i \neq j \; \& \; j \neq k\}$

$L2 = \{ww \mid w \in (0,1)^*\}$

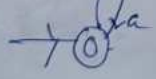
$L3 = \{1^{n!} \mid n \geq 0\}$

$L4 = \{0^n 1^n 0^n 1^n \mid n \geq 0\}$

L1 & L4 only

vi) What will be the no. of states in the minimal DFA which accepts the lang. represented by regular exp. (a+aaa)* over the alphabet {a}? 1

S = {ε, a, aaa, aa, aaaaa, aaa ---- ∞} a* or a+



vii) Match the correct automaton given in y for the given transition func in x.

x	y
i) Q×Σ → Q	A. NFA
ii) Q×Σ → 2^Q	B. DFA
iii) Q×Σ → Q × {L,R}	C. NFA with null moves
iv) Q × {Σ ∪ ε} → 2^Q	D. Turing machine
	E. 2 way DFA
	F. PDA

i – B, ii – A, iii – E, iv – C

viii) Consider the following DFA.

What is the language accepted by above DFA?

A) Strings over Σ = {0,1} contains '01' as a substring.

B) Strings ---------- '01' as ending string.

C) ---------- '01' as starting string.

D) None of these.

S = {01, 0001, 101, 010, 0111, ---- ∞}

(ix) Which of the following lang. is regular?

$\{a^n b^{2m} c^k \mid n, m, k \geq 0\}$

(x) Which of the following langs are DCFL?

$L_1 = \{ww^R \mid w \in \{a,b\}^+$, and w^R is reverse of $w\}$

$L_2 = \{ww^R x \mid w, x \in \{0,1\}^+\}$

Neither of L1 & L2

Q.n 3

2) Construct a DFA for the lang given below – L; All strings in which last 2 symbols are same, over alphabet $\Sigma = \{0, 1\}$

Done

3) Using pumping Lemma prove that – L; $\{WW | W$ where W means string $\}$ from Σ over $\Sigma = \{a, b\}$ is not a CFL.

Let, L is CFL.

Now, let, $P = 4$ [$\because |W| \geq P$], W = aabaab $\in L$

W = a a b a a b
    u v x y z

(i) $vy \neq \epsilon$

$vy = aba \neq \epsilon$

$(|W| = 3)$ (satisfied)

(ii) $|vxy| \leq P$

$\Rightarrow$ aba = 4 (satisfied)

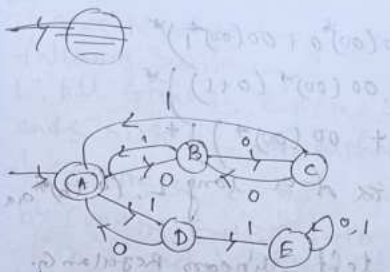
(iii) let, $i = 2$

$\Rightarrow uv^i xy^i z = uv^2xy^2z$

$\Rightarrow$ aaababaaab $\notin L$

∴ this isn't CFL (Proved) (not satisfied)

4) Using Arden's theorem find Reg ex. for the finite automata given below –



$A = \epsilon + B \cdot 1 + C \cdot 1 + D \cdot 0$ — (I)

$B = A \cdot 0 + C \cdot 0$ — (II)

$C = B \cdot 0$ — (III)

$D = A \cdot 1$ — (IV)

$E = D \cdot 1 + E \cdot (0+1)$ — (V)

from (II)

from (I)

$A = \epsilon + A \cdot 0 + C \cdot 0 + C \cdot 1 + A \cdot 1$ [from (I), (IV)]

$\Rightarrow A = \epsilon + C(0+1) + A(0+1)$

$A = [\epsilon + C(0+1)] \cdot (0+1)^*$ — (VI)

from (III)

$C = B \cdot 0$

$\Rightarrow C = (A \cdot 0 + C \cdot 0) \cdot 0$ [from (II)]

$\Rightarrow C = A \cdot 0 \cdot 0 + C \cdot 0 \cdot 0$

$\therefore C = A \cdot 00 (00)^*$ — (VII)

from (I) $A = \epsilon + A \cdot 0 + C \cdot 0 + C \cdot 1 + A \cdot 1$ [from (II), (IV)]

$= \epsilon + A \cdot 0 + A \cdot 00(00)^* 0 + A \cdot 00(00)^* 1 + A \cdot 1$

$= \epsilon + A(0 + 1 + 00(00)^* 0 + 00(00)^* 1)$

$\therefore A = (0+1+00(00)^*0+00(00)^*1)^*$

$= \{(0+1)+00(00)^*(0+1)\}^*$ (iii)

$= (0+1)[\epsilon + 00(00)^*)]^*$

b) Consider Reg ex of a lang L: $(a+b)^*a(a+b)+$.

a) write down left linear Regular G.

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

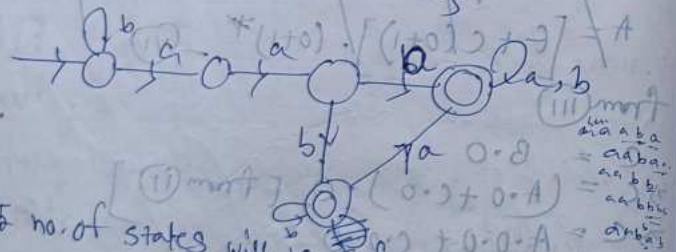
$S \to ABD \quad A aaD$

$A \to Aa|Ab|ab|\epsilon$

$D \to Da|Db|a|b$

b) What will be number of states in the minimized DFA that accepts L?

S: {aab, aaa, aaab, aaaa, a^2aab, a^2aabb, a^2aaab^3 ... a}



∴ no. of states will be 6 here.

c) Discuss closure properties of RL & Deterministic CFL language under operations- union, Intersection, complement, concatenation & Reverse. Done

Q 6-c

7) Consider the lang. below to ans. the following questions-

L: All strings, either starts with '00' or ends with '11', over alphabet $\Sigma = \{0,1\}$.

a) Construct ϵ-NFA which accepts L.

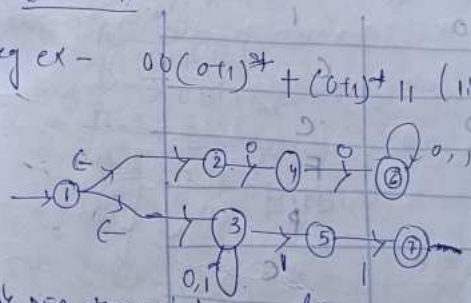
b) Convert ϵ-NFA into DFA. perform minimization if required.

c) Write down Regular grammers which generates L.

L: All strings, either starts with '00' or ends with '11', over alphabet $\Sigma = \{0,1\}$

a) ϵ-NFA

reg ex - $00(0+1)^* + (0+1)^*11$ (like starting & ending with same char & also start en tast 2 chars the same)



b) DFA transition table -

St	0	1	
→{1,2,3}	{4,3}	{3,5}	A
{4,3}	{6,3}	{3,5}	B
{3,5}	{3}	{3,5,7}	C
⊛{6,3}	{6,3}	{3,5,6}	D

{3} E	{3} E	{3,5} C
{3,5,7} F	{3} E	{3,5,7} F
{3,5,6} G	{6,3} D	{3,5,8,7} H
{3,5,6,7} H	{6,3} D	{3,5,6,7} H

minimization - (A B C D E F G H)

= (A B C E) (D F G H)

= (A) (B) (C) (E) (D G H) (F)

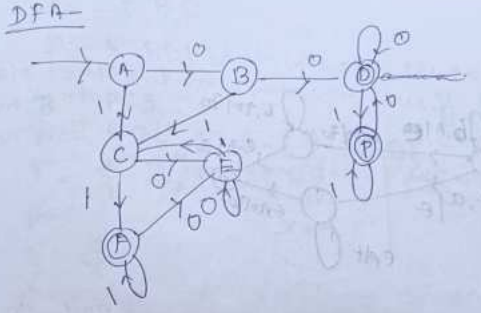
= (A) (B) (C) (E) (D) (G H) (F)

= (A) (B) (C) (E) (D) (P) (F)

minimized table -

SV	0	1
→ A	B	C
B	D	C
C	E	F
* D	D	P
E	E	C
* F	E	F
* P	D	P

DFA -



Reg ex for L: 00(0+1)* + (0+1)*11

Regular grammar for L =

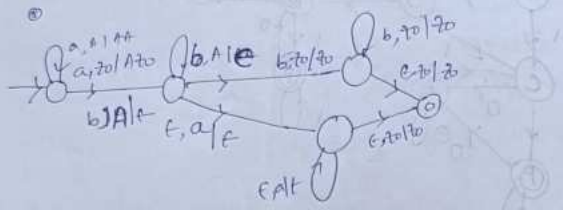
S → A + B

A → 00D

D → 0D | 1D | ∈

~~B → D1 | D1~~

~~A~~ B → 0B | 1B | 11



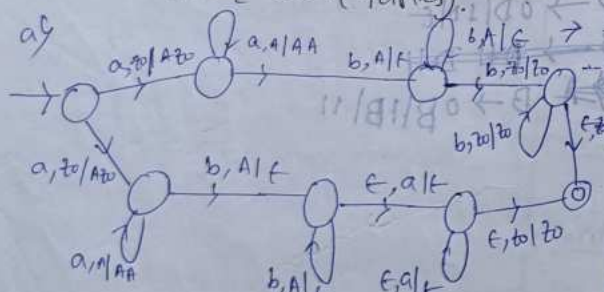
8) A context free lang is given below -

L; $\{a^n b^m / n \neq m$ (comparison) where $\Sigma = \{a, b\}\}$

a) Design a PDA to accept all valid strings of L.

b) write down the Grammer which generates L. (mention all 4 tuples). (N, T, P, S)

a)



b) $L: \{a^n b^m \mid n \neq m, \Sigma = \{a, b\}\}$

$S \rightarrow aSb \mid aS \mid bS \mid S_1 \mid S_2$

$S_1 \rightarrow AB$

$A \rightarrow aA \mid a$ (increasing # of a's)

$B \rightarrow aBb \mid ab$

S_1	S_2
$n > m$	$n < m$

$S_2 \rightarrow BD$

$D \rightarrow bD \mid b$ (increasing the # of b's)

this is also part of 5 tuples

N = {S, S_1, S_2, A, B, D}

T = {1, a, b}

P = set of production rules

S' = S

↓

starting symbol

9) a) Construct a TM to accept the lang. given below. L: $\{WCW\}$ where 'W' means strings formed over inputs = {a, b} & c is a terminals. Done

b) Explain Halting problem with an example. Done

10) a) write down differences b/w DFA & NFA. Done

b) Compare Recursive & Recursive enumerable languages.

Recursive	Recursive enumerable
i) If a string belongs to a lang we can say it is accepted & if it doesn't belong to the lang we can tell it is rejected.	i) If a ... accepted, but we can't ensure/ tell that it is rejected.
ii) It is called decidable lang.	ii) It is called recognizable lang.

(iii) This is no halting problem

(iv) It is closed under complement operation

Eg - done

(iii) Halting problem exists here.

(iv) It isn't closed under complement operation

Eg - done

c) Design a moore machine to determine the residue of mod 2 of the i/p treated as a binary string.

Not in syllabus

!!) a) Consider the grammer below & ans.

P: $E \to E+E \mid E*E \mid a \mid b \mid c$, where $N = \{E\}$, $T = \{a, b, c, +, *\}$ & E is the starting symbol. Perform LMD to generate a string "a+b+c", Is it an ambiguous grammer? Justify your ans.

b) write the grammer for the lang below -

$L = \{a^n b^m c^n t^m \mid n, m \geq 1\}$ & $\Sigma = \{a, b, c\}$

Done

P: $E \to E+E \mid E*E \mid a \mid b \mid c$

String: a+b+c

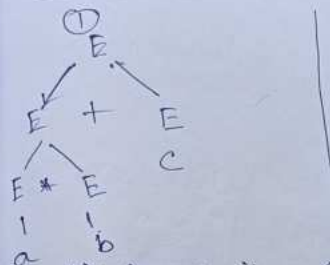
LMD

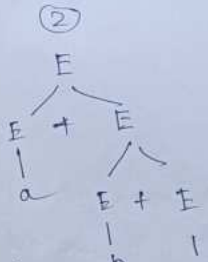
$E \to E+E$ [$\because E \to E+E$]
$\to E+E+E$ [$\because E \to E+E$]
$\to a+E+E$ [$\because E \to a$]
$\to a+b+E$ [$\because E \to b$]
$\to a+b+c$ [$\because E \to c$]

LMD ②

$E \to E+E$ [$\because E \to E+E$]
$\to a+E$ [$\because E \to a$]
$\to a+E+E$ [$\because E \to E+E$]
$\to a+b+E$ [$\because E \to b$]
$\to a+b+c$ [$\because E \to c$]

Derivation trees

① 

② 

As the grammer is ambiguous. There exist more than 1 distinct derivation tree & also there exist more than 1 LMDs. So, it is an ambiguous grammer.

Just for me

[1st justification is the exact justification, as those 2 trees are LMDs but how can we understand that it is LMD or RMD by just seeing the picture, because who tells it knows that whether it is LMD or RMD but the viewers doesn't know that, so more than 1 distinct derivation tree - Statement is more reliable].