Ch#1

Theory of Automata

=> The word automata is derived from Greek word "αὐτόματα" which means "self acting".

=> Also mean automatic machine or self-controlled.

=> Automata Theory is the study of abstract computing devices or machines.

=> Automaton = an abstract computing device.

=> In an area of CS. deals with the study of abstract machines (mathematical models) as well as the computational problems that can be solved using them.

- Different names:-
  - Theory of computation
  - Theory of Turing machin
  - Theory of compiles science
  - Theory of formal languages

Why we study TOA?

=> It allows us to think systematically about what machine do without going into hardware details.

=> Learning of Languages and Computational techniques.

=> Designing of theoretical models for machines.

What are pillars of Theory of automata?

=> There are three pillars of TOA:

L: o Language
A: o Automata
G: o Grammer

(1) Role of Alan Turing:-

=> A pioneer of automata theory
- o Father of modern Computer Science
- o English mathematician

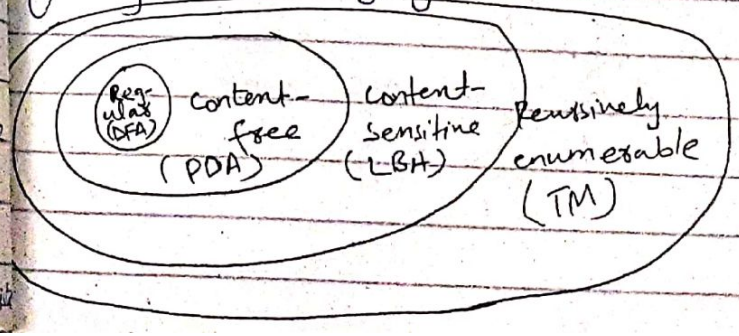
Studied abstract machines called Turing machines even before computers exist

Historical Perspective : TOA

1930s	o Alan Turing studies "Turing machines" o Decidability o Halting Problem
1940 - 1950s	o "Finite Automata" machines studied o Noam chomsky proposes the "chomsky hierarchy" for formal languages.
1969	Cook introduces "intractable" Problem or "NP- hard" Problems
1970	modern Computer Science : Compile Computational & Complexity theory evolve.

The chomsky hierarchy

=> A containment hierarchy of classes of formal languages:



Terminology

(i) Letters:

=> Characters / Symbols out of which we build languages for machine.

Example:

a, b, c, d, ...

0, 1, 2, 3, ...

(ii) Alphabet:

=> A finite, non-empty set of symbols / letters called alphabet.

=> Σ (Sigma) is used to denote an alphabet.

Examples:

- Binary: $\Sigma = \{0, 1\}$
- All lowercase letters: $\Sigma = \{a, b, c, \ldots, z\}$
- Alphanumeric: $\Sigma = \{a\text{-}z, A\text{-}Z, 0\text{-}9\}$
- DNA molecule letters: $\Sigma = \{a, c, g, t\}$

(iii) String:

=> Collection of alphabets, concatenation of letters OR sequence of letters without follow any rule is called String.

Example:

aa, bb, ab.

Words:

=> when a string is according to some rules then that string is called words.

for example:

- Rule we have: a string begin with a and end with b

$L_1 = \{ab, bb, bab, \ldots\}$

Language:

=> A set of strings with rules is called called language.

=> L is said to be a language over alphabet Σ,

only if $L \subseteq \Sigma^*$

=> this is because Σ^* is the set of all strings (of all possible length including 0) over the given alphabet Σ.

Examples:

1- Let L be the language of all strings consisting of n 0's followed by n 1's:

$L_1 = \{\varepsilon, 01, 0011, 000111, \ldots\}$

2) Let L be the language of all strings of with equal number of 0's and 1's:

$L = \{\varepsilon, 01, 10, 0011, 1100, 0101, 1010, \ldots\}$

vi) Empty language:-

=> ϕ denotes the empty language.

- $L = \{\varepsilon\}$ ~
- $L = \phi$ (false)

v) Empty String:-

=> String that has no letter, also known as Null String

=> denoted by Λ, λ or ε.

=> It's length is zero (0).

iv) Length of String:-

=> The number of letter in a string is called length of the string.

=> it is denoted by |s|.

Example:

s = abab

|s| = 4

length(s) = 4 or length(abab) = 4

Reverse of String:

=> Also called palindrom string.

=> Reverse of a string is obtained by writing the letters of string in reverse order.

=> it is denoted by Rev(s) or s^r or Reverse(s) e.g aa, aba

Example:
S = abab
Rev(s) = baba
Reverse (s) = baba

x Power of alphabet:

(ii) => Determines that the strings made from alphabet will be of length equal to power of alphabet.

=> Let Σ be an alphabet

- Σ^k = the set of all strings of length k.
- Σ^* = set of all possible strings from alphabet Σ
  $\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$
- Σ^+ = set of all possible strings from Σ except $\{\}$
  $\Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$

(xi) Example:

$\Sigma = \{a, b\}^2$
$= \{aa, ab, ba, bb\}$
Total no of letters in alphabet = n^m ← length/ power
$= 2^2 = 4$

xii> Power of String:-

- Determines the length of string.

Example:
$(bab)^2 = babbab$
$ba^2b = baab$

Note:-
Sets: $\phi = \{\} \neq \{\lambda\}$
Set size: $|\{\}| = |\phi| = 0$
Set size: $|\{\lambda\}| = 1$
String length:
$|\lambda| = 0$

Kleene Closure:

=> Also known as Kleene closure, Kleene Star or Kleene operator.
- Denoted by Σ^*
- it is undetermined power, represent infinite number of terms can be made including empty string
  e.g. $a^* = \{\lambda, aa, aaa, \dots\}$

- it is the infinite set of all possible strings of all possible lengths including ε.

Kleene Plus:

=> Also known as Kleene closure +ive, positive closure.

=> Denoted by +.

=> it is undecimined power, represent infinite number of terms can be made except empty string.

e.g

$a+ = \{a^* - \lambda\}$

Note:

$(a+b)^*$

=> Means any no of As and any number of bs

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

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

OR Operation:

=> a+b

either a or either b.

Lexicographic Order:

=> Method of sequencing a language in which strings are grouped by their length.

e.g strings of shortest length first.

Difference blw formal language & informal language.

Formal language	Informal language
o Concerned with just rules / syntax not with meaning. o Also known as syntatic e.g Language starts with w and end with o	o Concerned with rules and meaning both o Also known as Semantic language

=> Automata focus on formal language.

Q How do automata accept or reject the input?

- There are different methods with which we can describe a language.

Descriptive Definition:

=> It is one of the language defining methods.

=> In this method, we simply describe the condition imposed on its strings / words.

Language Name = { Definition }.

Example:

L_1 = { set of all strings that starts with a and ends with a }

Q. Define a pallindrome language with descriptive definition?

Palindrome = { λ, and all strings x such that reverse(x) = x }

e.g. Palindrome = a, b, aa, bb, aaa, bbb, ... }

What is recursive definition?

=> It is one of the language defining methods.

=> In this method we specify some objects in the set. The number of basic objects specified must be finite.

=> Second, we give a finite number of rules for constructing more objects in the set from the one we already know.

=> Third, we declare that no objects except those constructed in this way are allowed in the set.

Example: Recursive definition for the even numbers:

Rule 1: 2 is in P-Even.

Rule 2: if x is in P-Even, then so is x+2

Rule 3: The only elements in the ~~step~~ set P-even are those can be produced from the two rules above.

P-Even = { 2, 4, 6, 8, 10, ... }

Example:

Recursive definition for palindrome:-

Rule1: ^ , a and b are in palindrome

Rule2: If w ∈ Palindrome, then so are awa and bwb

Rule3: No other string in the palind. rome unless it can be produced by rules 1 and 2.

ImP Points:

Regular expression → RE → generators

Regular language → FA → DFA NFA

Regular Grammar

Primitives RE:

1) ϕ is Empty set
2) ∈ is null string
3) a ∈ Σ set of input string.

Operators
o + Union
o . Concatination
o Kleene closure
o Plus closure

Regular Expressions:-

=> it is one of the language defining methods.

=> The language accepted by finite automata are easily described by simple expressions called regular expressions.

=> The regular expression is the most effective way to "represent any language".

=> The language accepted by some regular expression is known as "Regular language"

=> In this method, language is represented in terms of strings. (Power, Concatenation, Union, ANY)

Example:

String contains only a letter

L = {^, a, aaa, aaaa, ...}

R = a*

Example:

Strings starts with a and contains any b letters

L = {a, ab, abb, abbb}

R = ab*

String contains : a or b
L = {a, b}
R = a+b

String contains any a or any b.
R = (a+b)*

Strings start with and ends with a
L = {aa, aaa, aba, aaba ...}
R = a(a+b)* a

Definition:

⇒ Let R be a regular expression over alphabet Σ. if R is:
- ε, is regular expression, denoting the set {ε}
- ϕ is the regular expression denoting the set empty {}
- For each symbol a ∈ Σ, a is a regular expression : denoting the set {a}
- Union of two RE is also regular. R1UR2 = regular
- Concatenation of two RE is also reg. R1R2
- Kleene closure of regular expression is also regular R*
- if R is regular, (R) is also regular.
- Nothing else, Repeat 1 to 7 recursively.

Q. Define a RE for language that contains the substring ba?
- (a+b)* ba (a+b)*

Q. All strings which do not contain the substring ba?
- a*b*

Q. All the strings which do not contain the substring 00.
0+1*+1*+1*01*

Q. All the strings which do not Contain the substring 101
0*1*000*1*0*

→ The set of all strings over {a,b,c} do not contain substring ac.
RE = (c*(a+(bc*))*)

Language having at least one a
$(a+b)^* a (a+b)^*$

Language of all words that have at least two a's
R.E = $(a+b)^* a (a+b)^* a (a+b)^*$

Language of all words that ha at least one a and at lea one b
R.E = $(a+b)^* a (a+b)^* b (a+b)^* +$
$(a+b)^* b (a+b)^* a (a+b)^*$

Language of all words that have at least one a or at least one b.
R.E = $(a+b)^* a (a+b)^* + (a+b)^* b (a+b)^*$

The language of even length.
R.E = $((a+b)\cdot(a+b))^*$

The language of odd length
~~R.E = $((a+b)\cdot(a+b))^*$~~

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

String of length 2 start with a
R.E = $aa+ab$

String all start and end with double letter
R.E = $(aa+bb)(a+b)^*(aa+bb)$

All words that contain exactly one a or one b
R.E = $b^*ab^* + a^*ba^*$

All words don't end with ba
R.E = $(a+b)^*(ab+bb+aa)$

All words do not begin with b
R.E = $a(a+b)^* + a$

All words that contain at least one double length
R.E = $(a+b)^*(aa+bb)(a+b)^*$

String in which no bab occurs
a^*b^*
or $a^*(b^*aa^+a^*)^*b^*a^*$

String in which no abb occurs
$b^*(a+ab)^*$

String all words of length >=3
$(a+b)(a+b)(a+b)(a+b)^*$

Language starts with a ends with b and having odd length.
RE = $a((a+b)(a+b)(a+b))^*)b$

Set of all those strings whose 3rd symbol from the right end is a 0.

$(0+1)^* 0 (0+1)(0+1)$

All those string having even length strings and starting with a & odd length string with b.

$E \cdot E = [aa + bb + (ab+ba)(aa+bb)^*(ab+ba)]^*$

$b(E \cdot E) + a(E \cdot E)(ab+ba)(E \cdot E)]$

String contains even no of 0's

$(0+1)^* 0 (0+1)^* 0 (0+1))^*$

String nothing ends with b or ba

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

String in which every 0 is immediately followed by 11

R.E: $(011+1)$

R.E that contains even a's & even b's

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

String that contains even no of a's

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

R.E that contains even no of b's

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

Strings not ending in 01

$(0+1)^*(00+11+10)$

All Strings having even no of 0's

R.E = $[1^* + (01+10)(00+11)^*(01+01)\cdot]^*$

Language that contains odd no of b's

R.E = $(a^*ba^*)(a^*ba^*ba^*)^*$

Language that contains odd no of a's & b's

$= (Even)^* (ab+ba)(Even)^*$

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

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

Language which has Length 4 and start & end with different letter

R.E: $a(a+b)(a+b)b + b(a+b)(b+a)a$

Language which do not contain ab.

RE = b^*a^*

Language of all strings at least 1 a & 1 bb

$(a+b)^+$

Language of all strings that have at least 1 a's & 1 bb

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

Language that has exactly 1 a & must start with b

$(b+ba+baa)^+ aaa (b+ba+baa)^*$

Language of all strings that do not contain bbb's

$(\lambda + b + bb)(a+ab+abb)^*$

or $a^*(b+\lambda)a^*ba^*ba^*$

Language of all strings in which aaa is substring or bbb but not both

$(\lambda+b+bb)(a+ab+abb)^* aaa (\lambda+b+bb)(a+ab+abb)^*$

$+(\lambda+a+aa)(ba+baa)^* bbb (\lambda+a+aa)(ba+baa)^*$

R.E for all strings that have exactly one double letter in them.

$(b+\lambda)(ab)^* aa (ba)^*(b+\lambda) + (a+\lambda)(ba)^* bb (ab)^* (a+\lambda)$

All strings do not end in double letter

$(a+b)^*(ab+ba)+a+b+\lambda$

Construct a regular expression for all words in which a appears triple if at all, this means every clump of a contains 3 or 6 or 9 or 12 a's

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

All strings that have odd no of a's & odd no of b's

Even-Even $(ab+ba)^*$ Even-Even

String with next-to-last symbol is 0.

$(0+1)^*0(0+1)$ or $(0+1)^*01^*01^*$

String ending with 0.

$(0+1)^*0$

String with even no of 0's & odd no of 1's

$= 1(\text{Even-Even}) + 0[E\cdot E(01+10)E\cdot E]$

Language which do not contain a double letter

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

All string except those with

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

String with odd no of 1's

$= (0|10^*)(0^*10^*10^*)^*$

String ending in 1 & not containing 00.

$= (01+1)^+$

All strings with no occurrences of 00

$1^*(011)^*(0+1)^*$

All strings with exactly one occurrence of 00

$1^*(011)^*00(11^*0)^*1^*$

All the string not containing 000.

$(1+01+001)^*(\lambda+0+00)$

All strings in which every run of 0's has length at least 3.

$(1+0000^*)^*$

All strings in which every substring 000 appears for every 1.

$(1+01+001)^*0^*$

L_1: Start with Zero and has odd length or start with 1 has even length

$= 0[(0+1)(0+1)]^* + 1[(0+1)(0+1)]^*(0+1)$

Odd - Odd

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

or $(a+b)(aa+bb+ab+ba)^*$

The language of all words without double a.

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

The language of even no of a's, followed by odd no of b's & or even no of a's followed by b's followed by odd no of a's

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

Strings that start with aa, end with bb, and have alternating sub-string ba in between.

$a(ab)^+b$

or $aa(ba)^*bb$

String contain at most one b.

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

The even length of strings both a's & b's

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

=> All words
Substrings bba and abb.
$= a^*(baa^*)^*b^* + b^*(a^*ab)^*a^*$

The odd length strings of a's & b's

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

String that do not contain both a and b in them

RE = a^*+b^*

Language in which either a come before b or b come before a.

$= a^*+ba^*$ or
$= a^*(\lambda+b)$

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

Language that do not contain the substring ac

$(c^*(a+(bc))^*)$

Book definition:

- The language defining symbols we are about to create are called regular expressions.
- regular expressions itself recursively.
- The language associated with this expressions are called regular language.

Finite Automata

=> Automata is plural of Automaton. Automata means self-controlled machine.
=> It is a method for defining language.
=> it a graphical method for the representation of regular language.
=> Finite automata is also called finite automata machine.
=> A finite automata has 5-tuple, where
- States → final no. → F
  → initial → Q
- alphabets → Σ
- Transition (δ)

=> it is acceptor of regular language

1) Alphabets:

- An alphabet Σ of possible input letters, from which are formed strings that are to be read one letter at a time.

2) States:

- A finite automata contains finite no of states.
- States are like a place.

Types of States

- Initial state
- Final state
- Intermediate state

Initial state: (q_0)

=> Every finite automata always start with q_0, that is called start state or initial state.

=> In FA there will be only one initial state. it state that the machine naturally starts in before it reads any input. - it is entry point.

=> It is represented by: → (q_0) or (−) or 1

Final state:

=> In FA the minimum number of final state should be 1 and maximum number of final states can be more than 1.

=> It means we may have more then 1 final states in FA.

=> At final state machine may stop or may not stop depend on our language (input).

=> It is represented by:

⊕ or ◎

If is state where the machine halts when it has no input left. Also called accepting state.

Q (States / normal states)

- These are the states other then initial and final states.
- These are such states from with our FA neither start nor finish.
- These are represented by: ○

3) Transition / movement:-

- Transition or movement that tell for each state and for each letter of the input alphabet which state to go to next.
- it is represented with arrow → sign.

Types of FA

- FA
  - NFA (non-deterministic)
  - DFA (deterministic)

DFA: $(Q, \Sigma, \delta, v_0, F)$

- Output already known. (has only one next state)

o Deterministic means that each string and thus each state sequence, is unique.

o In DFA, for each input symbol, one can determine the state to which the the machine will move. Hence, it is called deterministic finite automata.

o The FA are called DFA if the machine is read an i/p string one symbol at a time.

⇒ Remember 3 points for DFA.

i) No empty string

ii) No of transaction from each st is equal to number of letters give in alphabet.

for example:

if $\Sigma = \{a, b\}$. This alphabet ha two letter a & b.

That means from every state there wil be two transaction.

iii) it may also contain reject stat

Dead state / Dead end state

o it is also called reject or trap state.

o once the machine enters a dead state, there is no way for it to reach an accepting state.

o it never allows us to reach final state.

To accept string:

All the input string is scanned & the last state is accepting.

To reject a string:

All the input string is scanned & the last state is non-accepting.

Proper Definition:-

o It is a type of FA that used to define language in which next state is already known.

o it consists of 5 parts.

Limitations:

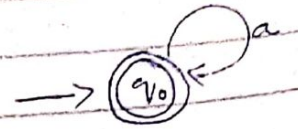
o Limited memory

o for Linear power not suitable

o String without comparison

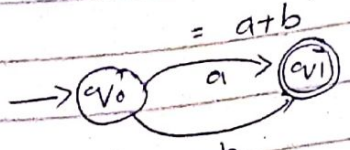
DFA that accept any no of a's

RE = a*



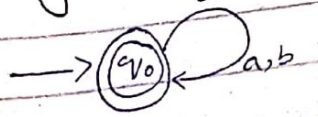
DFA that accept either a or b.

= a+b

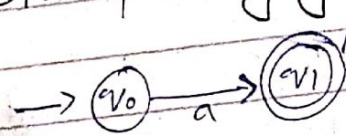


DFA that accept any no of a's & any no of b's

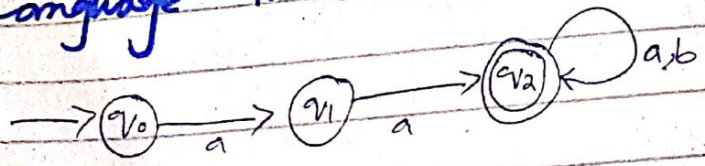
= (a+b)*



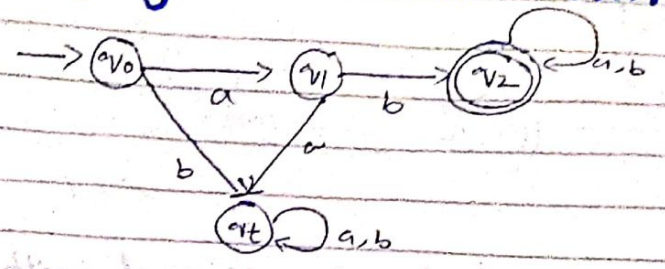
DFA for language that start with a



Language that start with aa

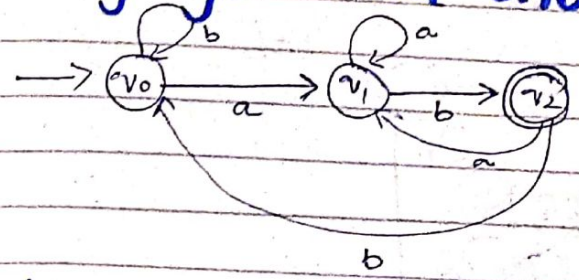


Language that start with ab



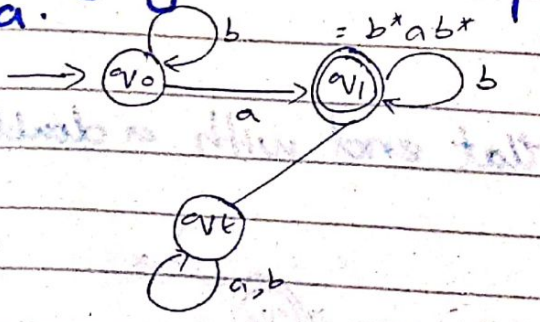
Language that end with ab.

= (a+b)* ab

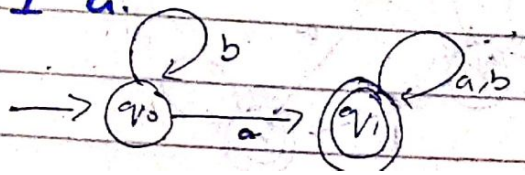


Language that accept exactly one a.

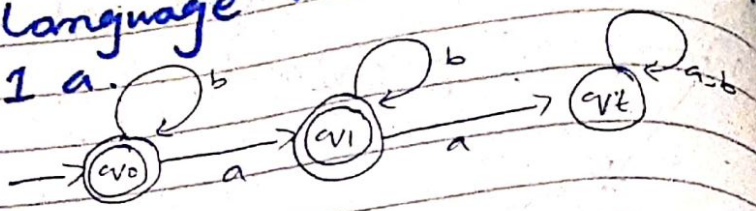
= b*ab*



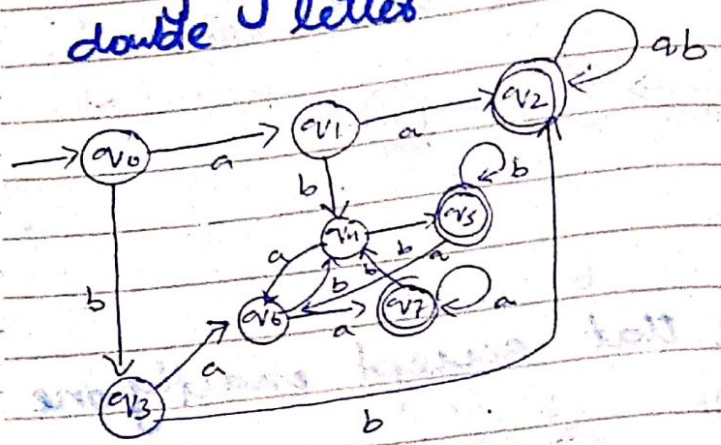
Language that accept at least 1 a.



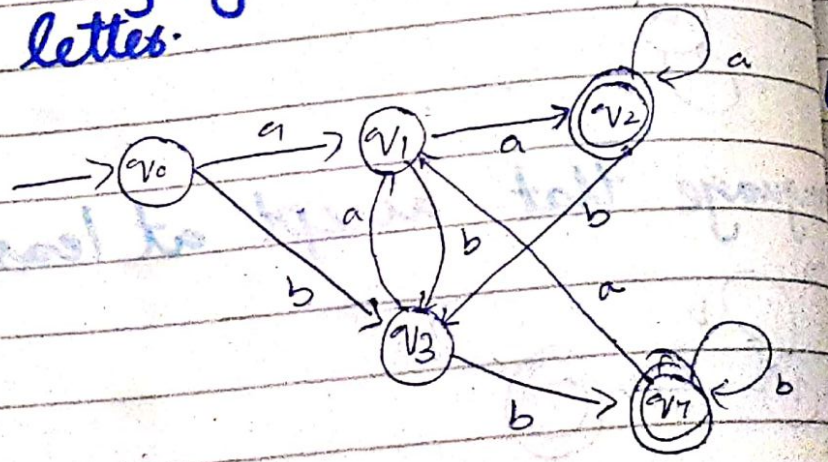
Language that accept at most 1 a.



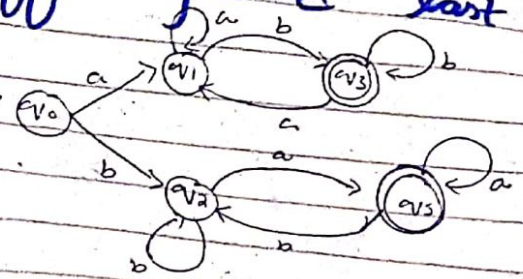
Language that begins or end with double letters



Language that end with a double letters.

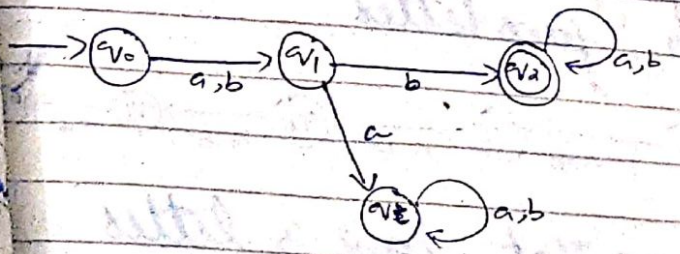


Language that accept strings having different first & last letters

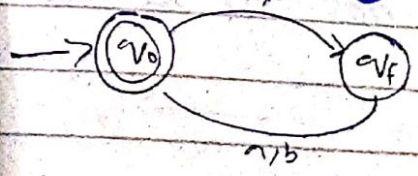


Language that accept b as a 2nd letter

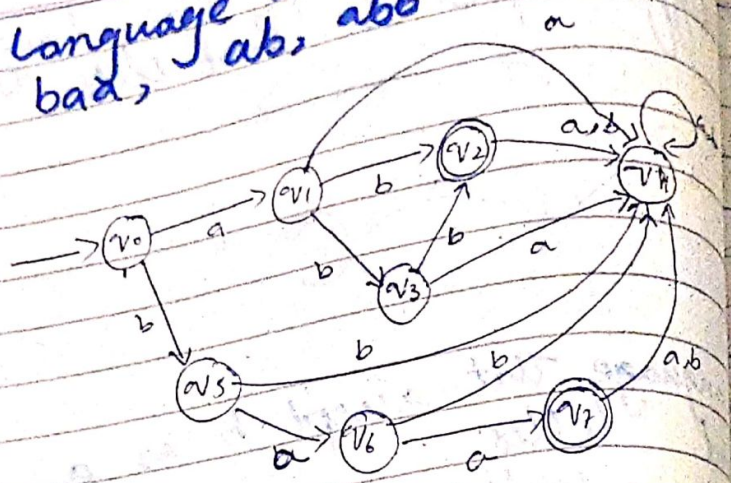
$= (a+b)^* b (a+b)^*$



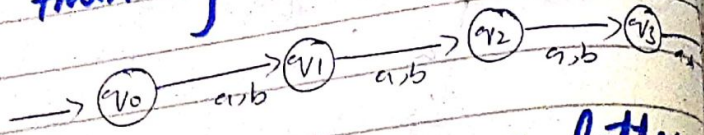
Language that accept letters of even length.



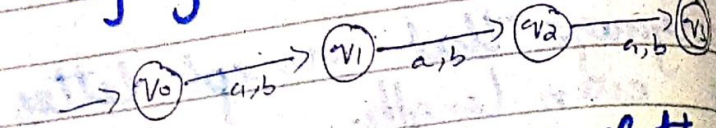
Language that contains only
baa, ab, abb



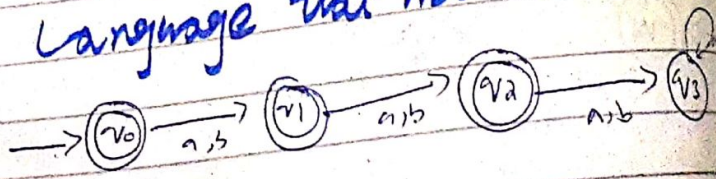
Language that have 4 or more than four letters



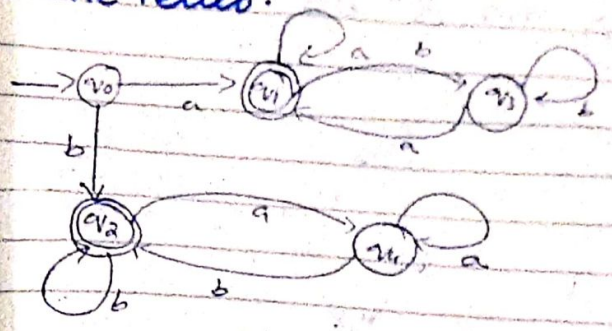
Language that have 3 letters



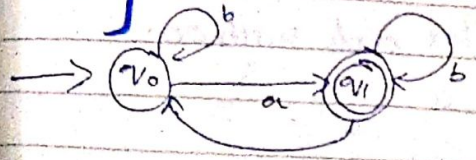
Language that not 3 letters



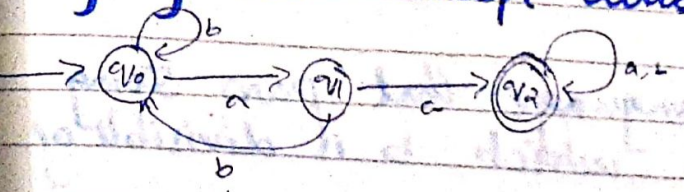
Language that start & with same letters.



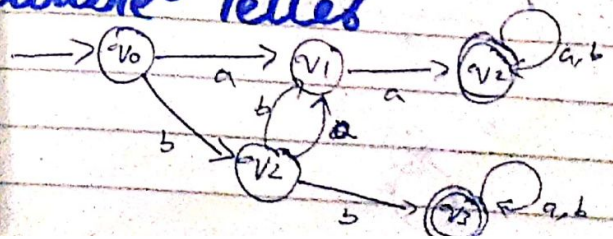
Language that accept odd no of a's



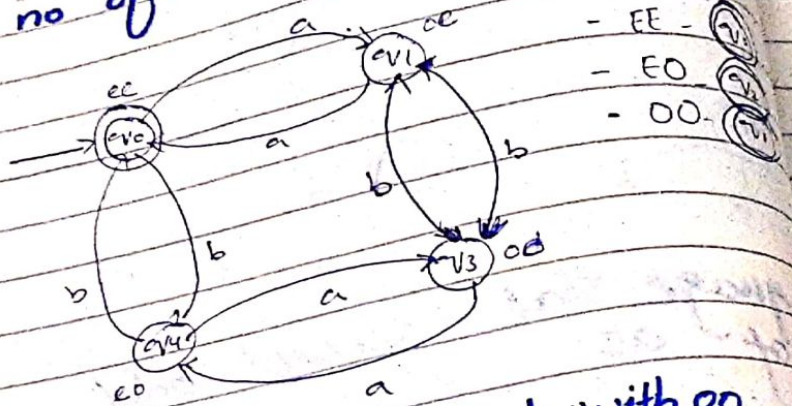
Languag that accept double a



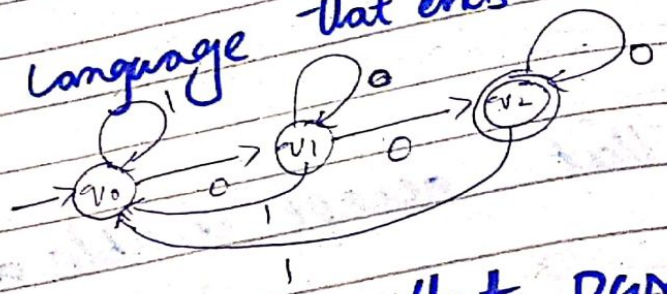
Language that Contains one double letters



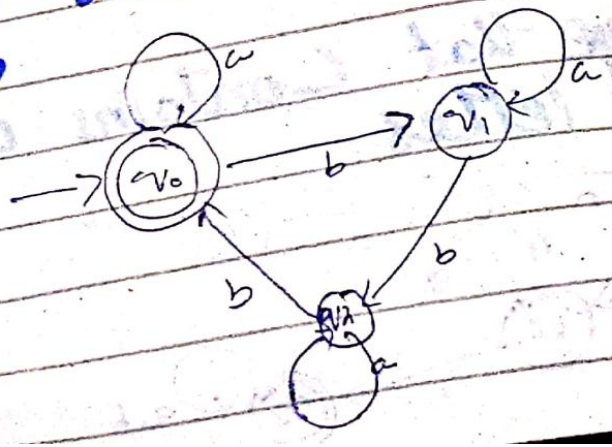
Language that accept
no of a's & b's



Language that ends with 00

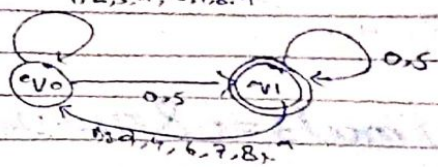


Language that accept string in which b is divisible by 3

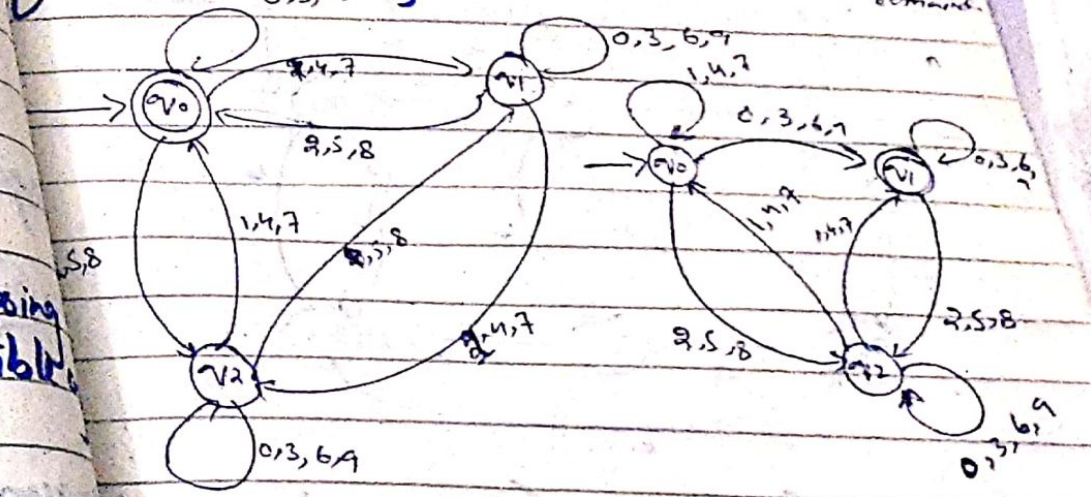


Language in which string is divisible of 5.

$\Sigma = \{0, \ldots 9\}$

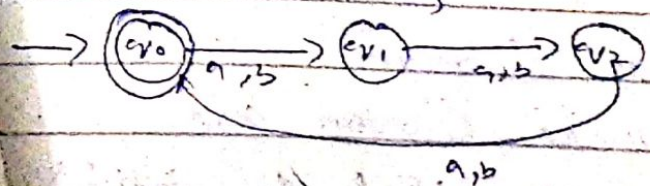


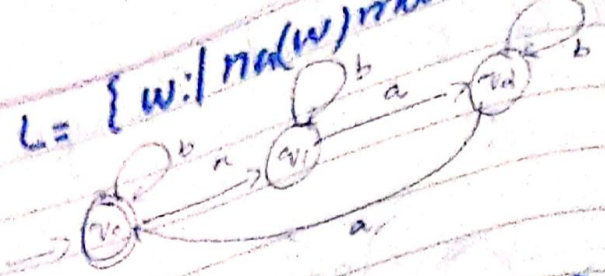
Language that accept string of divisible of 3



Language $L = \{w : |w| \bmod 3 = 0\}$

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



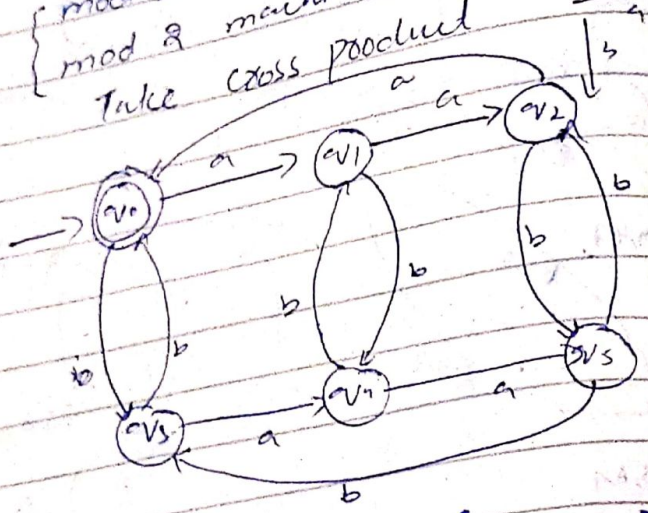


$L = \{w : |na(w) \bmod 3 = 0 \ \& \ n_b(w)$

Trick

{ mod 3 machines 0, 1, 2 → a
{ mod 2 machines 0, 1 → b

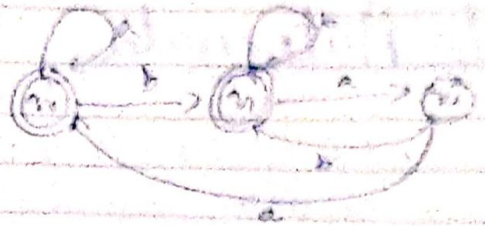
Take cross product



Special case: Complemen

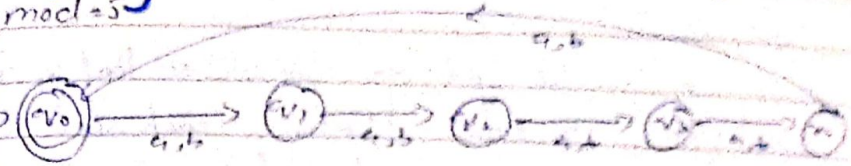
- Do not Contain
- First make Contain case
- Then make final to initial initial state to final

Language that do not end with ba

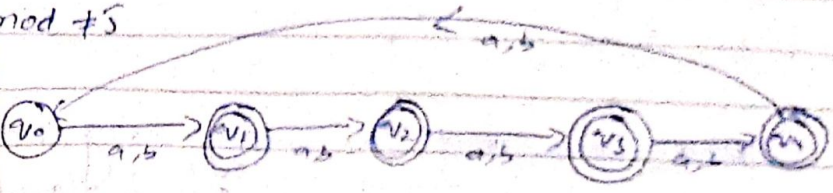


Language $L = \{w : |w| \bmod 5 \neq 0\}$

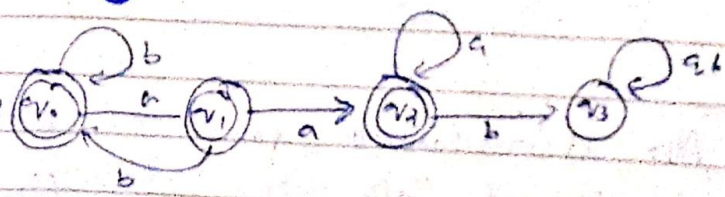
mod = 5



mod ≠ 5



Language do not contain aab.

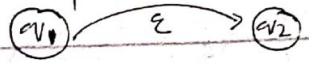


Non-deterministic finite Automata

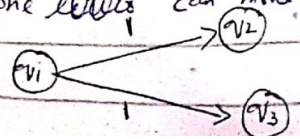
⇒ it is a collection of three things:
- A finite set of states with one start (-) and some final states (+).
- An alphabet Σ of possible input letters
- A finite set of transition that determines how to proceed from each state to the states along edges labeled with one letters of the alphabet

Flexibilities

- Allow transition of an empty string λ or ε that's why it is called λ transition or epsilon transition.



- Also allows transition from one state to different states given a character.
  - one letter can move to more than one state, need state is nd deterministic



non-deterministic transition

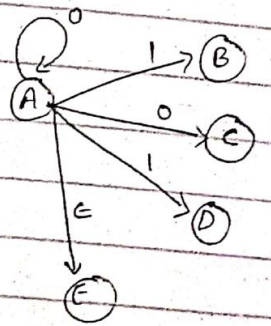
- Not necessary to pass every alphabet from every state.
- No trap state exist because characters that do not need we do not pass them simply.

Note:
- Every FA is an NFA.
- But every NFA is not FA.
- By Kleene theorem, every TG has an equivalent FA.

Therefore:
languages of FA's ⊂ languages of NFA's ⊂ languages of TG's = languages of FA's

⇒ it is simple and easy to design because there are less restrictions.

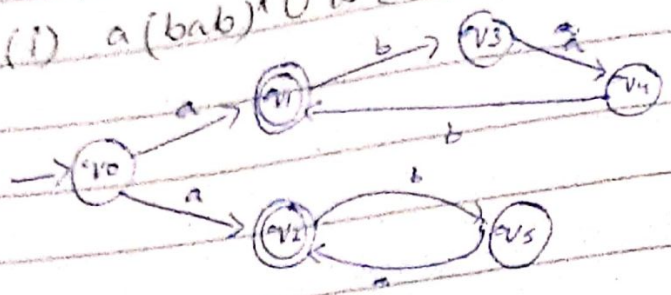
e.g



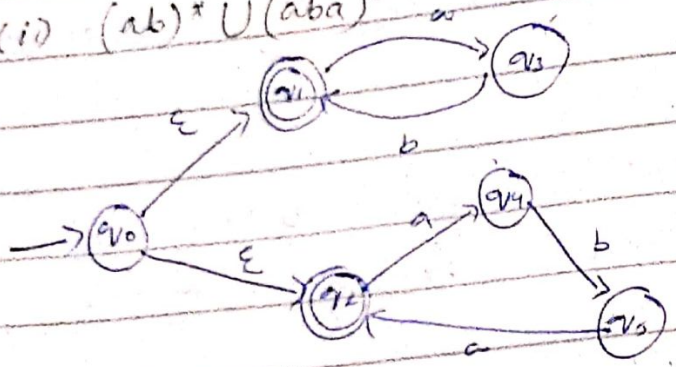
Example:

Construct NFA.

(i) $a(bab)^* \cup a(ba)^*$



(ii) $(ab)^* \cup (aba)^*$

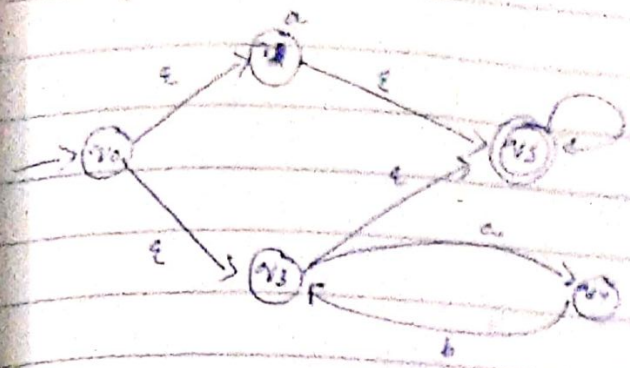


(iii) {foo language that omits at least one symbol $\sigma \in \Sigma$}

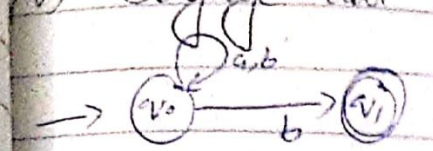
$\Sigma = \{a, b, c\}$

$(a \cup b)^* \cup (a \cup c)^* \cup (b \cup c)^*$

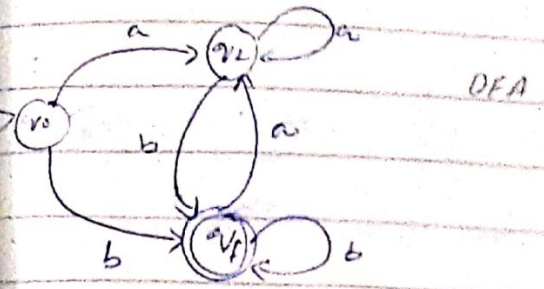
(iv) $(a^* + (ab)^*)b^*$



(v) Language that ends with b.

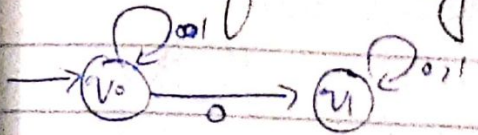


NFA



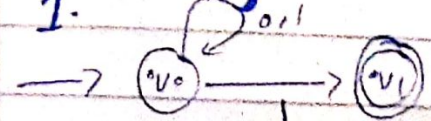
DFA

(vi) Set of all strings containing 0

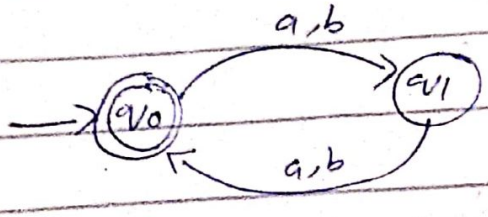


(vii) Set of all strings and with

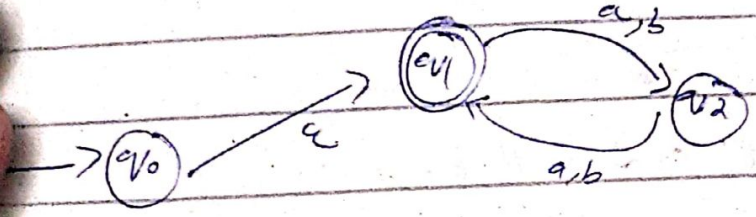
1.



Language that pass even letter strings

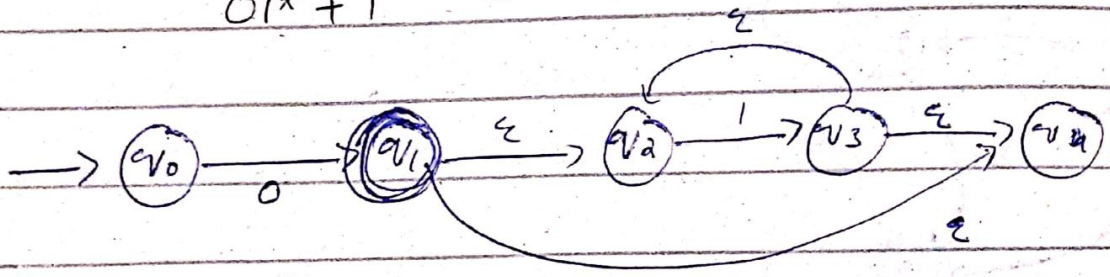


OR



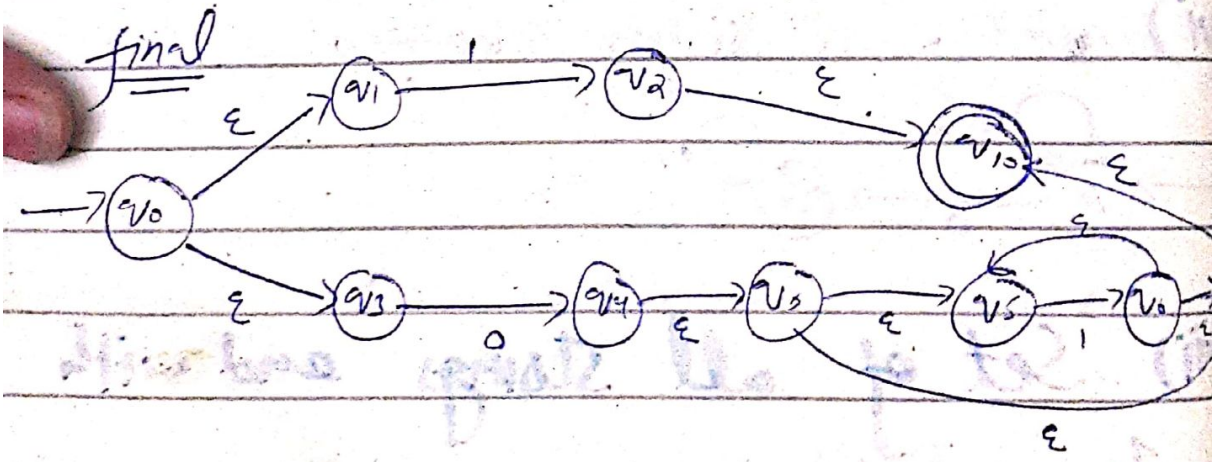
Constoud NFA foo regular expression:

01* + 1

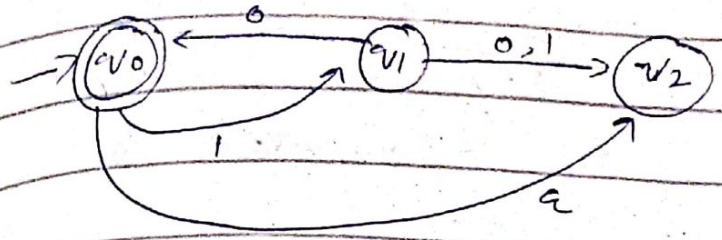


foo 01*

final

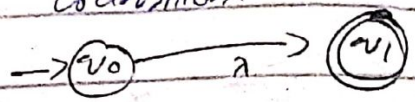


* for language $L(m) = \{(10)^n \cdot n >= 0\}$

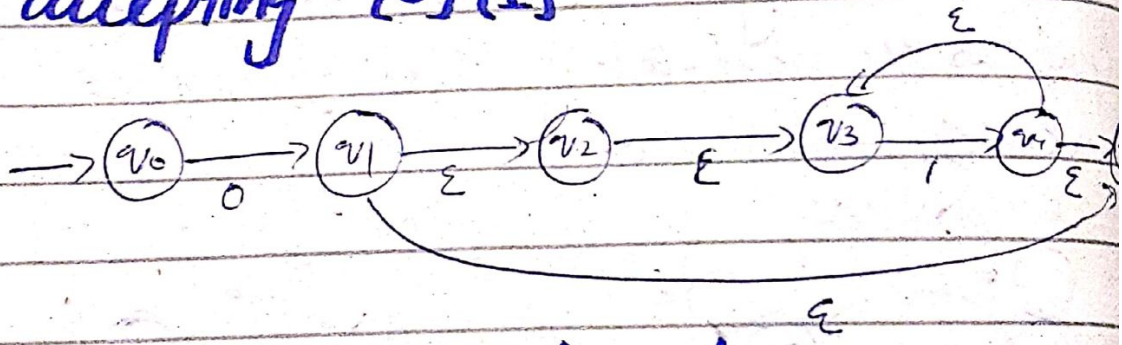


*Difference between DFA and NFA.

DFA	NFA
=> One letter cannot move to more than one state, there is only one output so the next state is already determined.	=> Letter can move to more than one state, next state is not determined.
e.g. $R = (a+b)^*a$	e.g. $R = (a+b)^*a$
=> Dead state is required.	=> Dead state is not required.
e.g. $R = a$	e.g. $R = a$

3) Do not allow ε or Null transition	3) Allows Null / Epsilon Transition. 
4) Sometimes hard to construct as Compared to NFA.	4) Easy to construct as Compared to DFA.
5) Requires more Space.	5) Requires less Space.

Give an example of an NFA accepting $\{0\}\{1\}^*$



Use of finite automata:

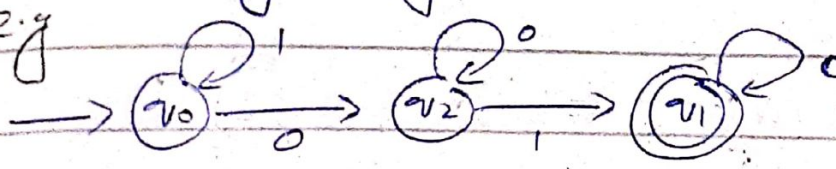
- for building different kinds of software including:
  - Lexical analysis Component of a Compiler and systems for verifying the Correctness of Circuits or protocols

What is Transition diagram?

=> A transition diagram or state diagram for a DFA $A = (Q, \Sigma, \delta, v_0, F)$ is a directed graph defines as:

- There is a node for each state in Q, which is represented by the circle.
- There is a directed edge from node v to node p labeled a if $\delta(v, a) = p$.
- In the start state, there is an arrow with no source.
- Accepting states or final states are indicating by a double circle.

e.g.



What is Transition table?

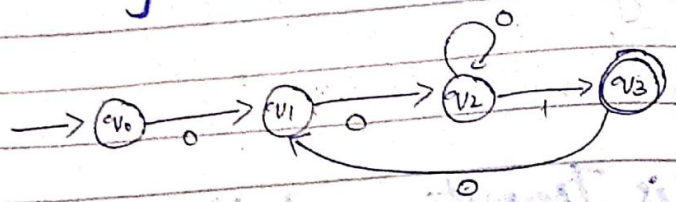
=> A transition table is a conventional, tabular ~~expression~~ representation of a function like δ that takes two arguments and returns a value.

=> The rows of the table correspond to the states, and the columns correspond to the inputs.

⇒ The entry for the row corresponding to state v and the column corresponding to input a is the state $\delta(q, a)$.

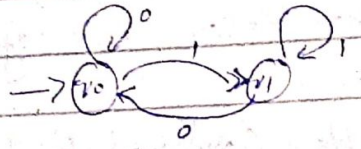
δ	0	1
$\to v_0$	v_2	v_0
$*v_1$	v_1	v_1
v_2	v_2	v_1

NFA for language in which before every 1 there is at least one zero



DFA for binary number is even:

v_0 – ends with 0 – even
v_1 – ends with 1 – odd



What is transition graph?

- it is a method to define a language

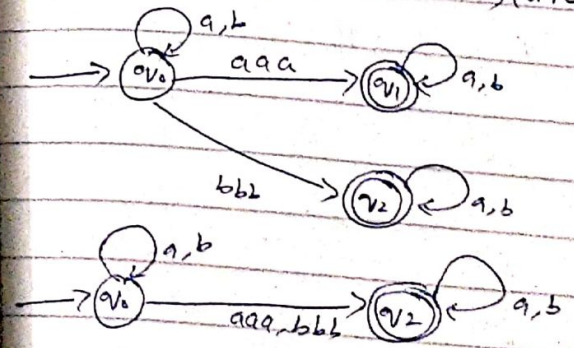
Rules:

- TG can have more than one initial states / final states
- Dead end state is not required.
- One letter can move to more than one state
- Can read more than one characters at a time (it allows substring transition)
- Every TG is NFA.
- it allows epsilon / null / lambda transition

it allows substring

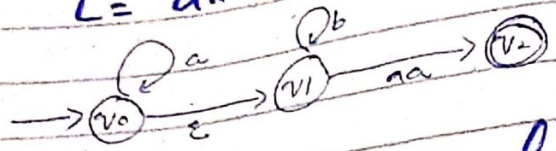
Draw TG that accepts all words with triple aaa or bbb over $\Sigma = \{a, b\}$.

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



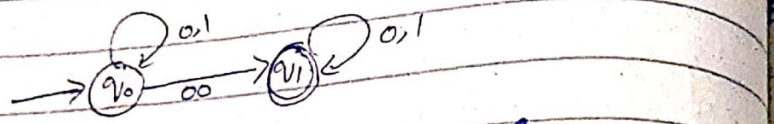
Q. Construct TG for:

$L = a^*b^*aa$



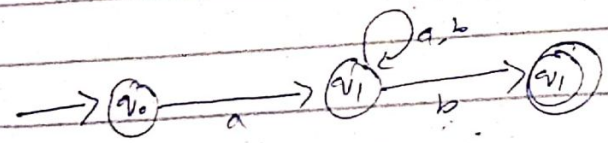
Q Construct TG of a language that accepts the string having 00 as substring.

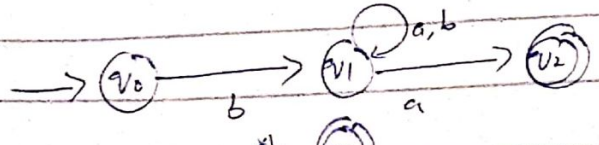
$R.E = (0+1)^*00(0+1)^*$

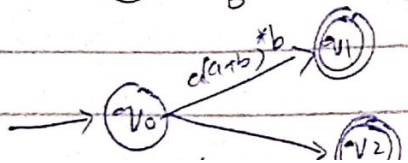


Construct TG of a language that accepts the strings having double letter starting and ending letter?

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

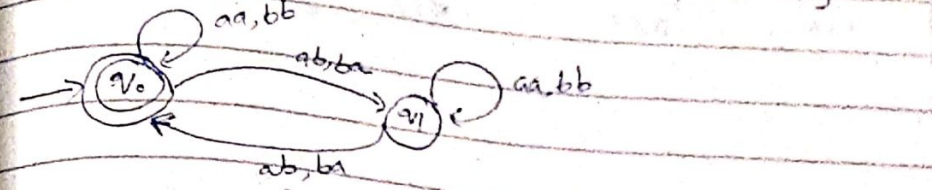




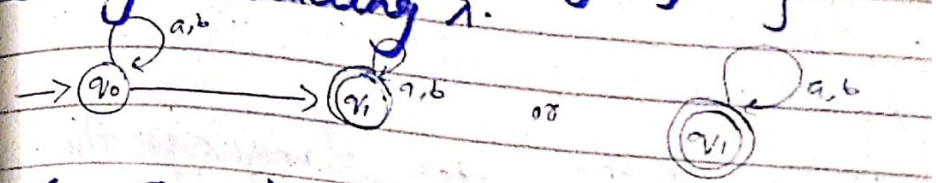


Transition graph for even no of a's & even no of b's.

$R.E = [aa+bb+(ab+ba)(aa+bb)^*(ab+ba)]^*$

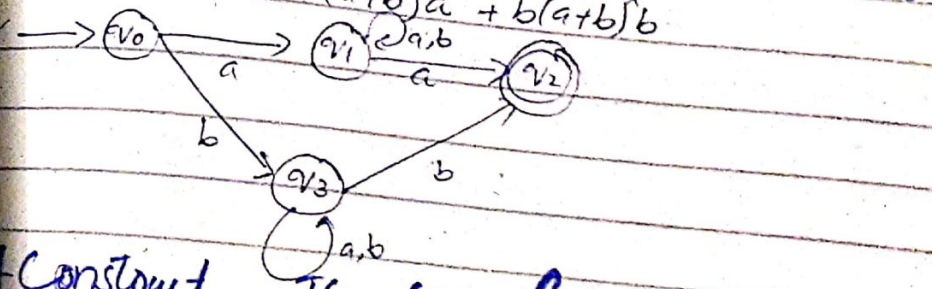


#Construct TG for language of all strings including λ.



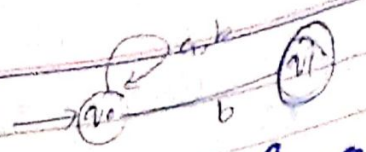
#Construct TG for language that starts and ends with same letter.

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



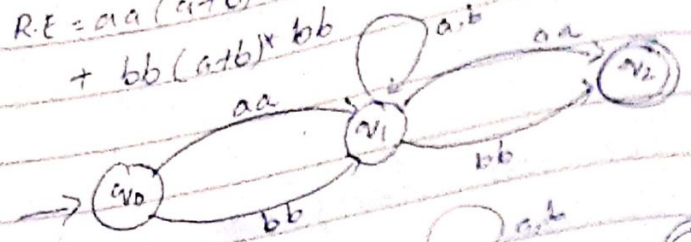
#Construct TG for language that ends with b.

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



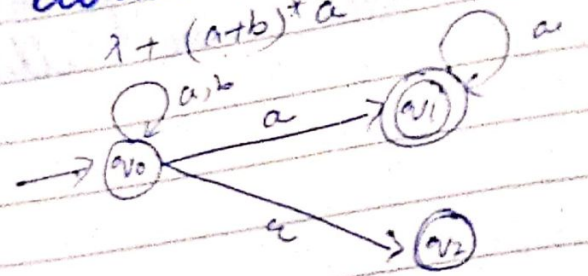
TG for language that ends and start with double letter.

R.E = aa(a+b)*aa + bb(a+b)*bb + bb(a+b)*aa + aa(a+b)*bb



Construct TG for language that do not end with b.

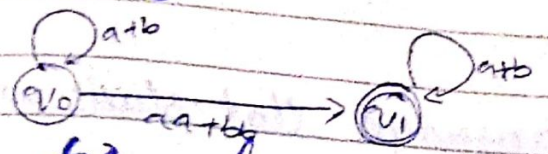
λ + (a+b)*a



GTG

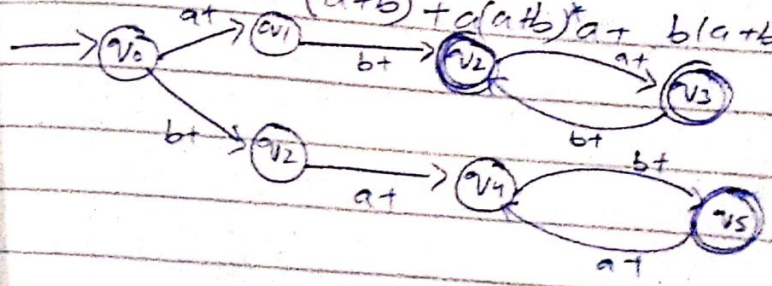
- GTG consists finite numbers of states.
- At least one start state and some (may be none) final states.
- GTG consist finite set of input letters from which input strings are formed.
- Only one arrows for all states
- GTG pass regular expressions.

Construct GTG for language that contains either double a or double b.

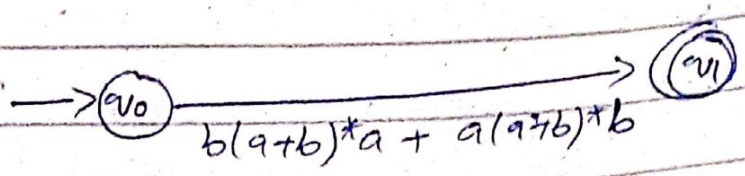


GTG for language that beginning and ending with same letter.

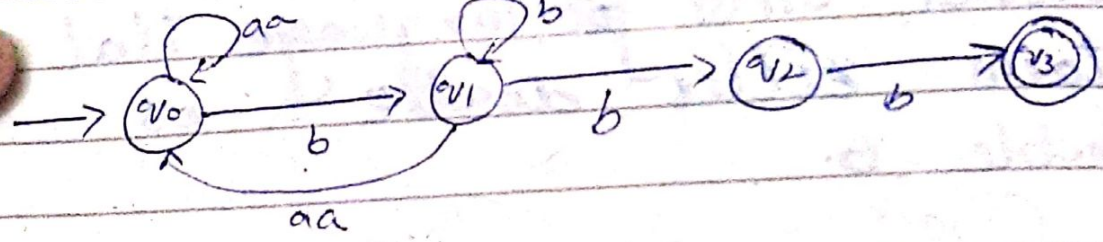
(a+b) + a(a+b)*a + b(a+b)*b



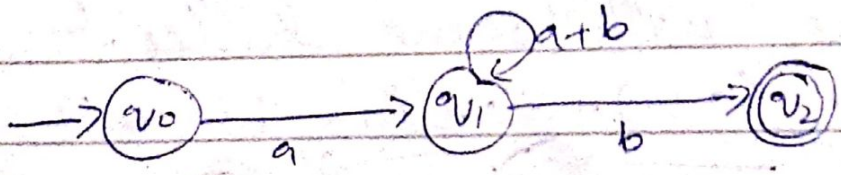
GTG for language that begin and ending in different letter?



TG for language in which a occurs a and b occurs more than and with three or equal to three



GTG for language that starts with a and end with b.



Part Papers

1) Define regular expression?
2) Difference btw kleene star closure and plus?
3) what is difference b/w words and strings?
4) Define NFA. Give an example?
5) Difference b/w deterministic and non-deterministic finite automata?
6) what is null string?
6) What is palindrome?
7) What is the concept of valid and invalid alphabet?

o While defining an alphabet of letters consisting of more than one symbols, no letter should be started with any other letter of the same alphabet.
o One letter should not be the prefix of another. However, a letter may be added in the letter of same alphabet. One letter may be the suffix of another.

Ex:

Σ = {a,b} (valid alphabet)
Σ = {a,b, cd} (valid alphabet)
Σ = {a,b, ac} (invalid alphabet)

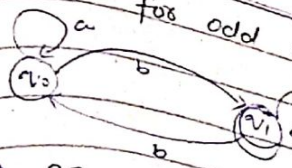
Q8) What is difference blw an alphabet & of an element of a set? whether alphabet is an element of a set or it is a set of itself?

⇒ An alphabet is a set in this
⇒ The elements of an alphabet are called letters.

for example:
Binary alphabets Σ = {0,1} Here 0,1 are the letters of binary alp

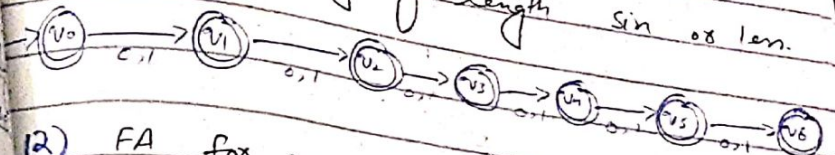
9) Define Kleene star?
10) How diagrams of FA's are Creat
11) What are the basic rules to build FA's?
12) What is difference blw (a,b) and (a+b)?
13) How diagram of FA are Created

(9) DFA for odd no of b's



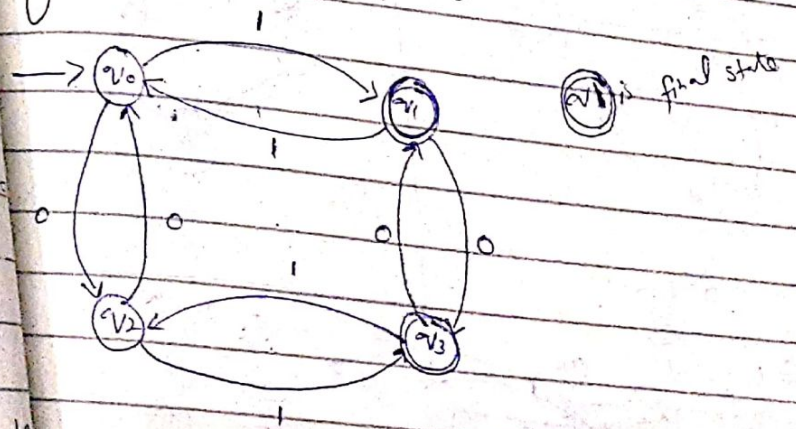
10) RE for language end with 0
RE = (0+1)*0

11) FA for strings of length 5 or less.



12) FA for even a's and even b's
Solved already.

(13) FA for even no of 0's and odd no of 1's

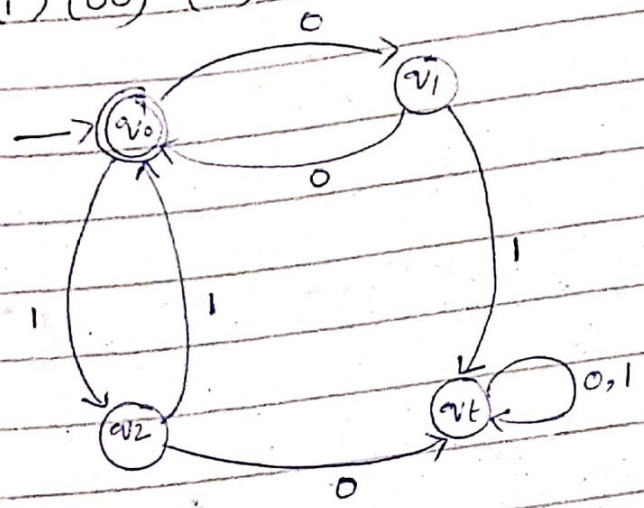


(N) DFA for $(a+b)^*(aa+bb)(a+b)^*$

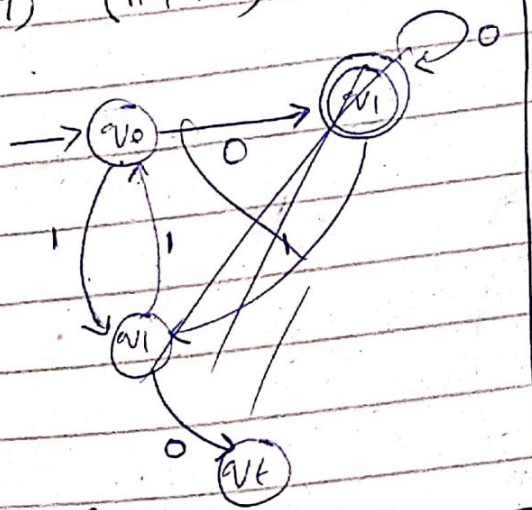
Solved

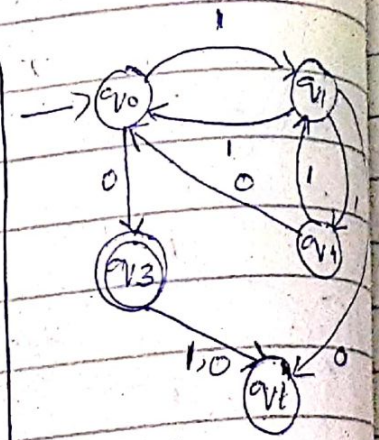
15) DFA for

(i) $(00)^*(11)^*$

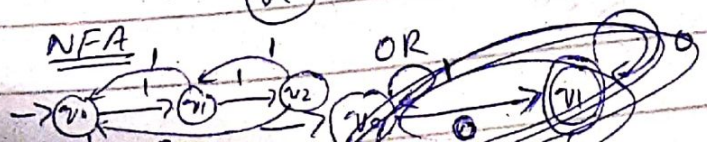


(ii) $(11+110)^*0$

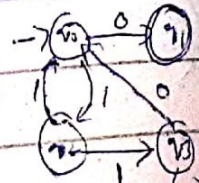




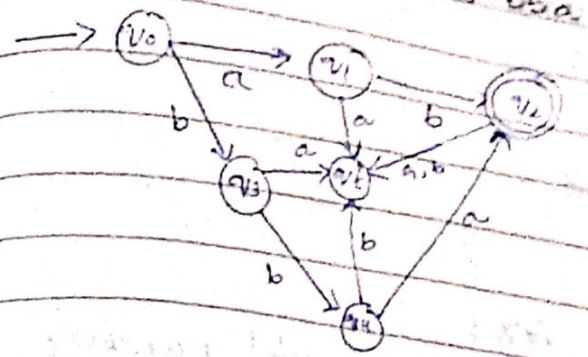
NFA



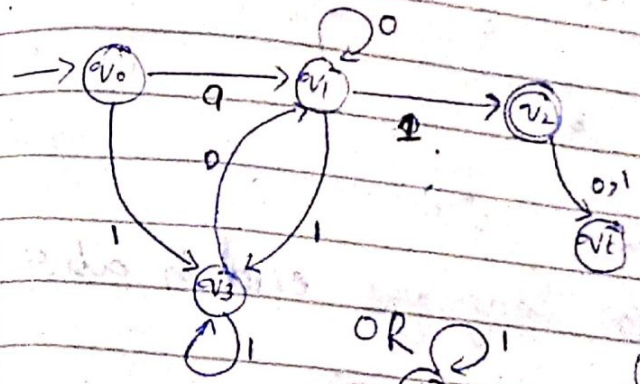
OR



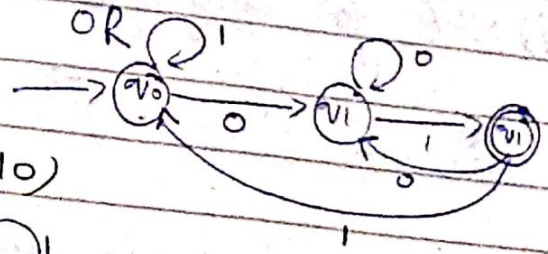
(iii) String ends ab or bba



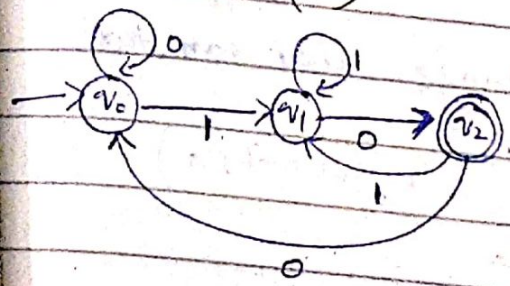
(iv) $(0,1)^*(\phi\phi)$



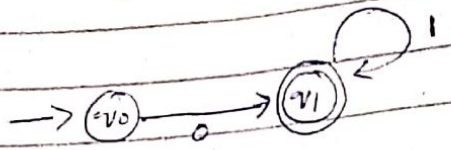
OR



(v) $(0,1)^*(10)$

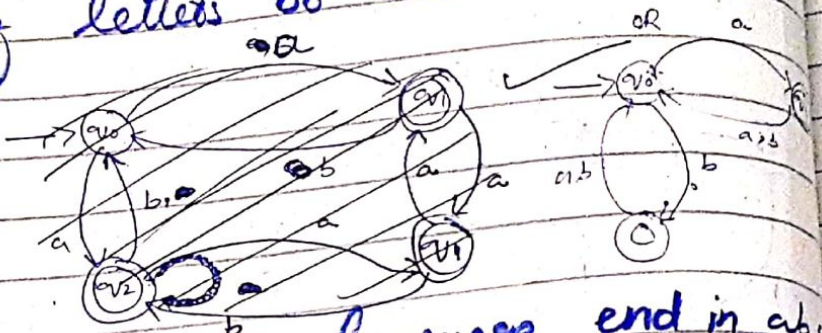


Give an example of NFA that accept 01*

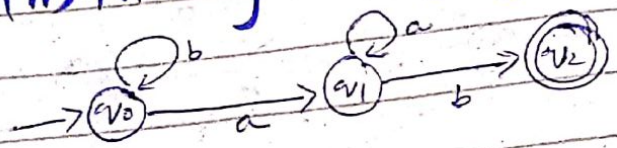


Build FA for:

(i) either have an odd number of letters or end in a.

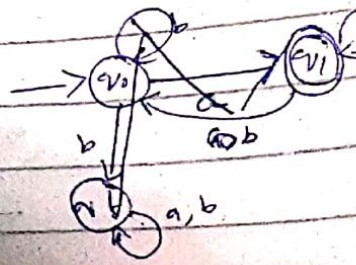


(ii) NFA for language end in ab



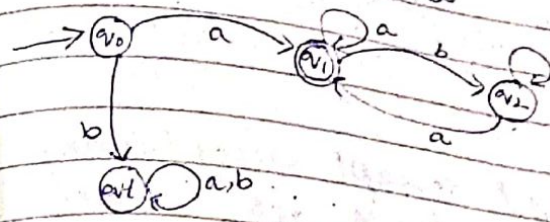
iii) FA for language that can start and end in a.

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

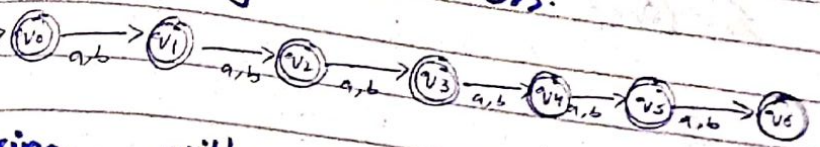


FA for language that starts and ends with a.

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



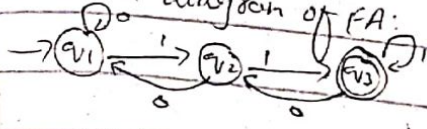
Strings of length six or less.



String with next-to-last symbol is 0.

(0+1)*0(0+1)

Given a diagram of FA: (PP-2017)



RE = 0*11(000*11)*1*

(1) 000111 is accepted and 10110 is rejected.

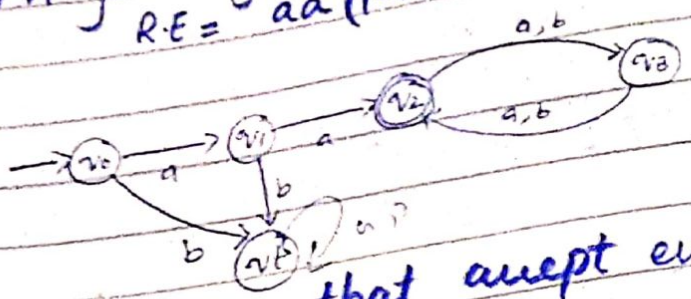
(2) {11, 0111001, ...} etc

RE = (0*11((00)11)1*)*

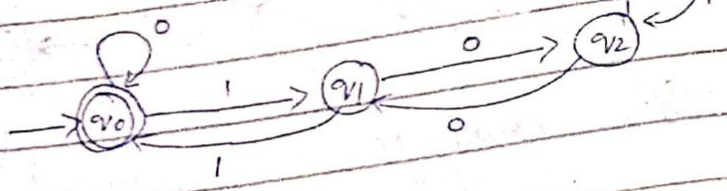
(3)

FA for regular expressions:-

R.E = aa((a+b)(a+b))*



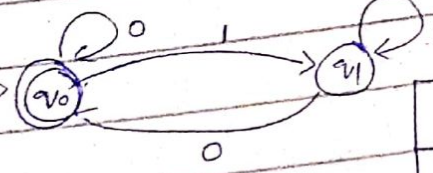
Make FA that accept every binary number divisible by 3. for when $\Sigma = (0,1)$



	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

FA for every binary no divisible by:

= (2) remainder (0,1)

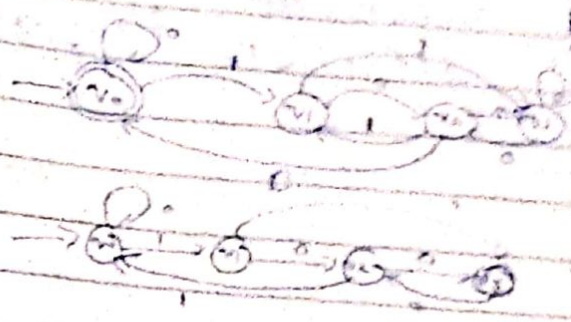


TT

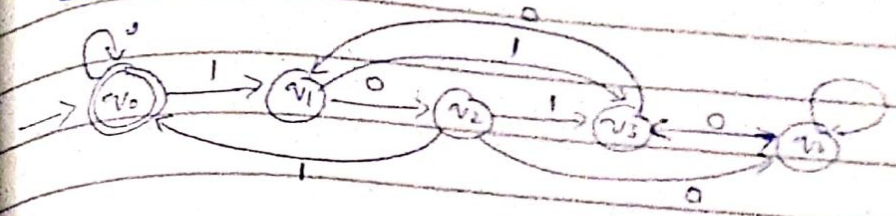
	0	1
q_0	q_0	q_1
q_1	q_0	q_1

divisible by 4

	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

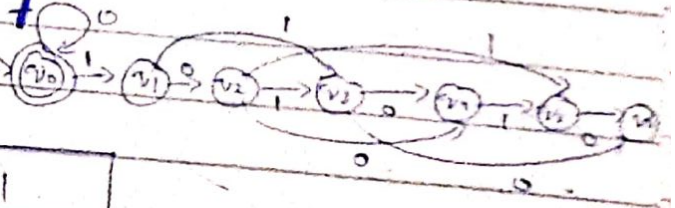


divisible 5



divisible by 7

TT



	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_6	q_0
q_4	q_1	q_2
q_5	q_3	q_4
q_6	q_5	q_6

Application of FA

- word process Prog
- Digital logical design
- Lexical analyzer
- Switching circuit
- design
- Text editors etc

Decision Properties of RL | CFG

There are the following decision properties

(1) Finiteness - it states that language accepted is finite or not

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

Q is the finite set of states.

(2) Membership - To check that a particular string accepted or not by a given FA

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

(3) Emptiness - To check whether a given FA accepts empty language or not

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

(4) Equivalence

NFA ⊆ DFA

Definition:

=> Decision property for a class of language is an algorithm that takes a formal description of a language.

Define closure property

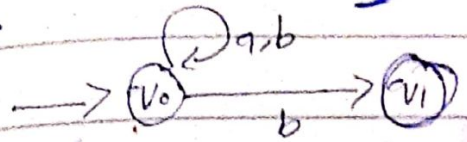
=> Certain operations on regular languages which are guaranteed to produce regular language.

=> Closure refers to some operation on a language, resulting in a new language that is of some "type" as originally operated

Converting NFA to DFA

- Let T be the new transition table for DFA.
- If the transition of start state over some alphabet is null, then perform the transition of start state over that input alphabet to a dead state in the DFA.
- In DFA, the final state will be all the states which contain F (final states of NFA.

Convert the following NFA to DFA.



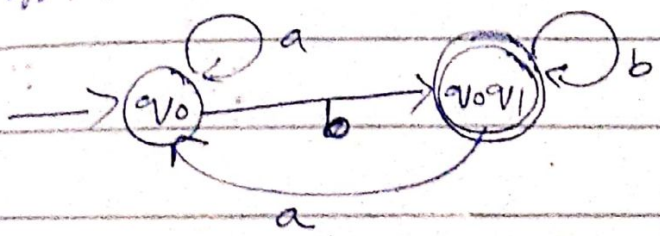
$Q = \{q_0, q_1\}$

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

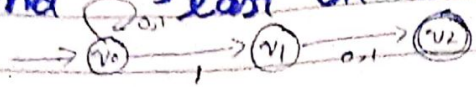
$F = \{q_1\}$

q_0 = initial state

δ	a	b
q_0	q_0	q_0q_1
q_0q_1	q_0	q_0q_1

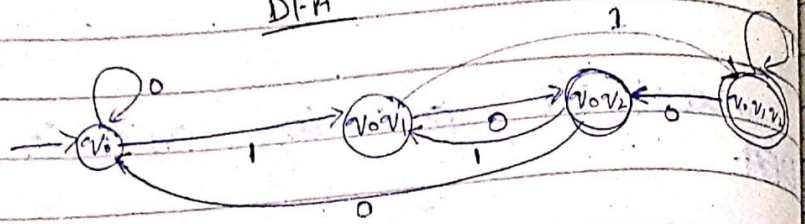


(i) NFA of all binary string whose and last bit is 1.

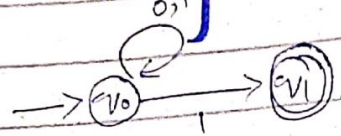


δ	0	1
q_0	q_0	q_0q_1
q_0q_1	q_0q_2	$q_0q_1q_2$
$q_0q_1q_2$	q_0q_2	$q_0q_1q_2$
q_0q_2	q_0	q_0q_1

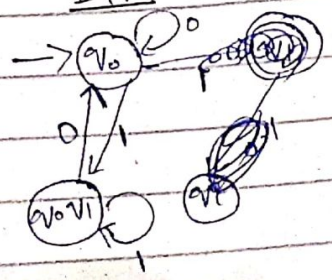
DFA



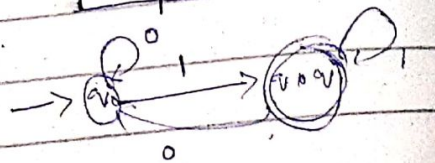
(ii) Set of all strings end with



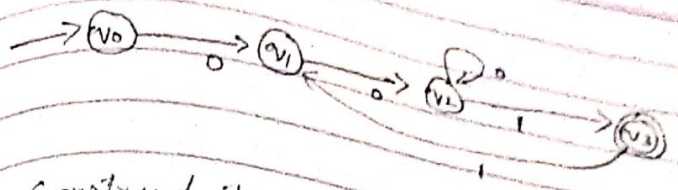
DFA



δ	0	1
q_0	q_0q_f	q_1q_1
q_1	-	-
q_0q_1	q_0q_f	q_1q_1



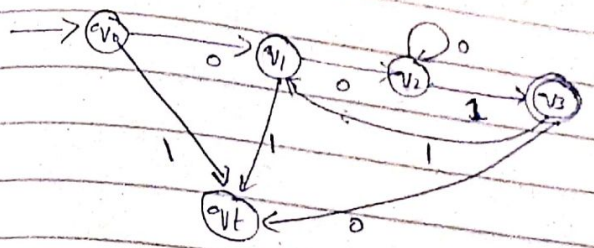
NFA to DFA



Constructed transition table.

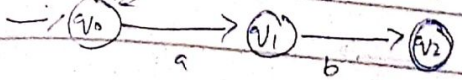
δ	0	1
q_0	q_1	-
q_1	q_2	-
q_2	q_2	q_3
q_3	-	q_1

DFA



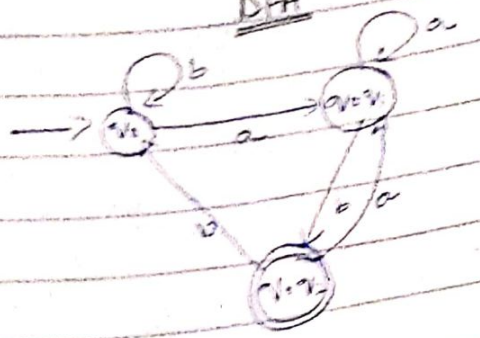
NFA given:-

for language end with ab

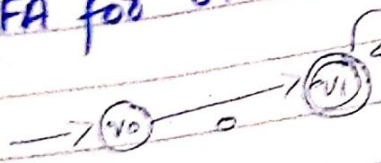


δ	a	b
q_0	$q_0 q_1$	q_0
$q_0 q_1$	$q_0 q_1$	$q_0 q_2$
$q_0 q_2$	$q_0 q_1$	q_0

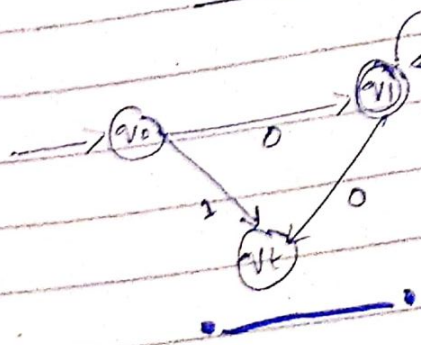
DFA



NFA for 01* convert into DFA.



	0	1
q_0	q_1	-
q_1	-	q_1



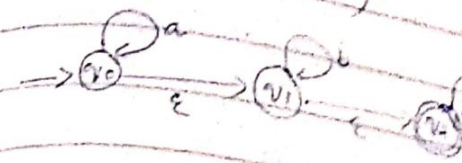
Epsilon-NFA

⇒ NFA that allows to move to multiple states without consuming any input symbol.

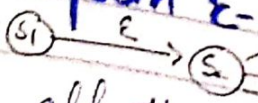
⇒ $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$

Example:

$(a+b+c)^*$

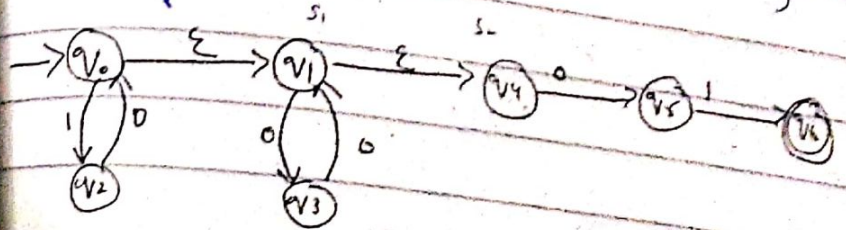


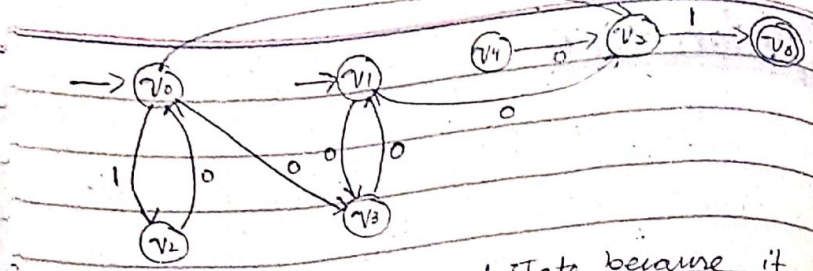
Conversion from ε-NFA to NFA.



1) Find all the edges starting from S_2.
2) Duplicate all edges to S_1 without changing edge labels.
3) if S_1 is initial state, make S_2 initial.
4) if S_2 is final then make it final.
5) Remove the dead state.

Example:





=> v_4 is a dead state because it is unreachable so, we can remove it.

Conversion from Epsilon-NFA to DFA:

=> There are two methods for this conversion.

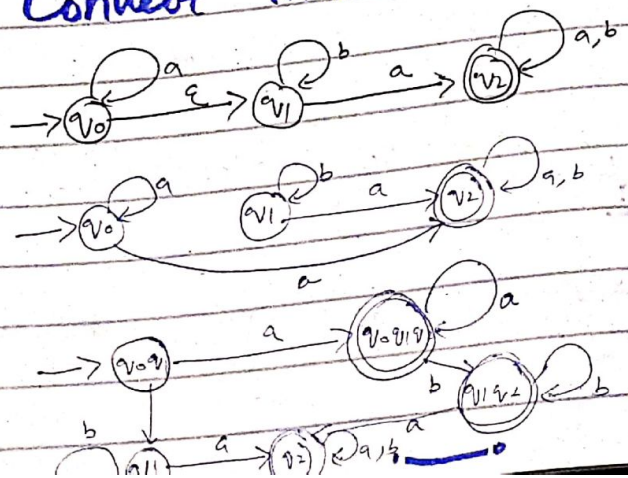
Method 1:

- Convert ε-NFA to NFA
- Then Convert NFA to DFA

Method 2:

- Directly method.

Convert into DFA.



Properties of Regular expression

=> Any terminal symbol i.e symbol ∈ Σ, including ∧ and ϕ are regular expressions.

=> The union of two regular expressions is also a regular expression. $L_1 \cup L_2$ or $L_1 + L_2$

=> The concatenation of two regular expressions is also a regular expression. L_1L_2

=> The iteration (or closure) of a regular is also a regular expression. R^*

=> The intersection of two regular expression is regular. $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

=> The reversal of regular expression is also regular. L^r

=> The complement of a regular expression is also regular.

=> A homomorphism of a regular expression is regular

=> The inverse homomorphism is also a regular.

=> The difference of RL is a regular. $L_1 - L_2 = L_1 \cap \overline{L_2}$

1) Union of RE

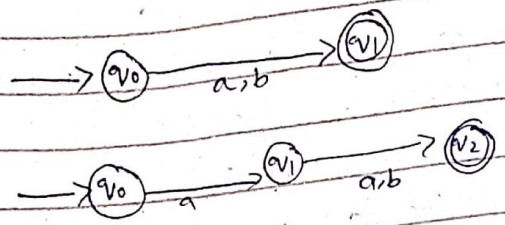
$R_1 \cup R_2$, $R_1 + R_2$, $R_1 | R_2$

RE = $L_1 \cup L_2$

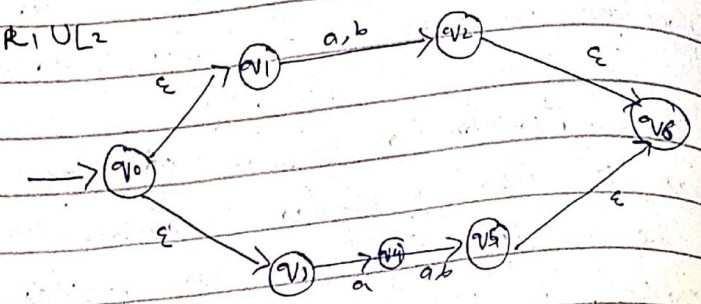
L_1 = NFA1 = a+b

L_2 = NFA2 = a(a+b)

Note: Final State of Resultant FA will be that contains either one or both final state of both FA



$R_1 \cup L_2$

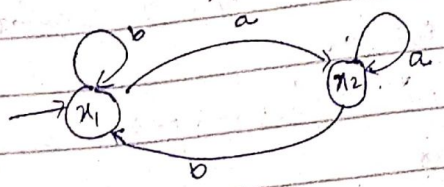


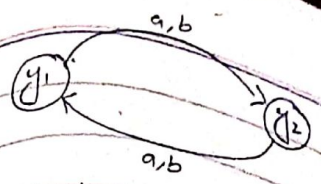
Union of Two DFA

Example 2:

$\gamma_1 = (a+b)^*a$

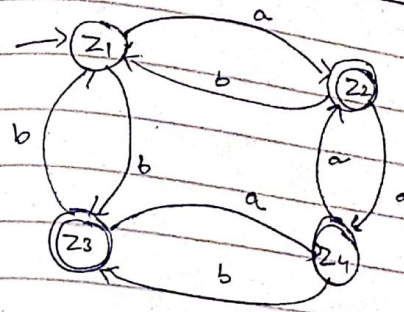
$\gamma_2 = (a+b)((a+b)(a+b))^*$ or $((a+b)(a+b)^*)(a+b)$





old states	New states a	b
$z_1 \equiv (x_1, y_1)$	$(x_2, y_2) \equiv z_2$	$(x_1, y_2) \equiv z_3$
$z_2 \equiv (x_2, y_2)$	$(x_2, y_1) \equiv z_4$	$(x_1, y_1) \equiv z_1$
$z_3 \equiv (x_1, y_2)$	$(x_2, y_1) \equiv z_4$	$(x_1, y_1) \equiv z_1$
$z_4 \equiv (x_2, y_1)$	$(x_2, y_2) \equiv z_2$	$(x_1, y_2) \equiv z_3$

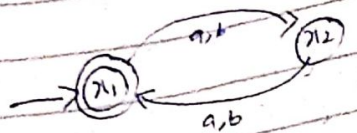
old states	New states after reading a	b
z_1	z_2	z_3
z_2^+	z_4	z_1
z_3^+	z_4	z_1
z_4^+	z_2	z_3

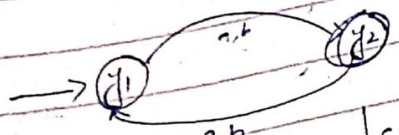


Example 2:

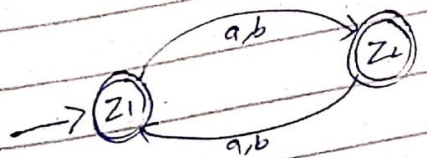
$\gamma 1 = ((a+b)(a+b))^*$

$\gamma 2 = (a+b)((a+b)(a+b))^* . or\ ((a+b)(a+b))^*(a+b)$





Old States	Old a	Old b
$x_1y_1 = z_1$	$x_2y_2 \equiv z_2$	$x_2y_2 \equiv z_2$
$x_2y_2 = z_2$	$x_1y_1 \equiv z_1$	$x_1y_1 \equiv z_1$

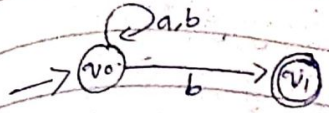


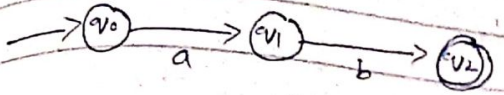
2) Concatenation of two FA

Example:

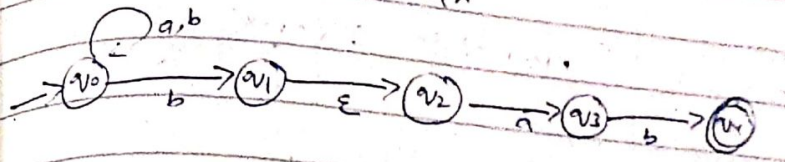
$\gamma_1 = (a+b)^*b$

$\gamma_2 = ab$





$\gamma_1\gamma_2$ - NFA

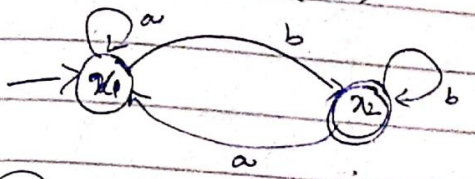


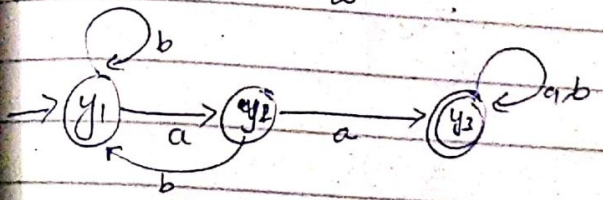
Concatenation of 2 DFA's

Example:-

$\gamma_1\ (a+b)^*b$

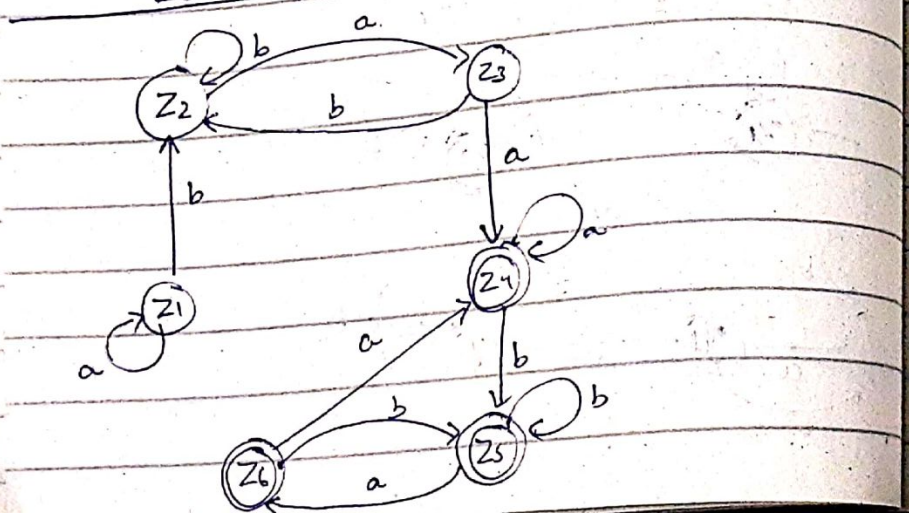
$\gamma_2 = (a+b)^*aa(a+b)^*$





old states	New states after reading a	b
$z_1 \equiv x_1$	$x_1 \equiv z_1$	$(x_1, y_1) \equiv z_2$
$z_2 \equiv (x_2, y_1)$	$(x_1, y_2) \equiv z_3$	$(x_2, y_1) \equiv z_2$
$z_3 \equiv (x_1, y_2)$	$(x_1, y_3) \equiv z_4$	$(x_2, y_1) \equiv z_4$
$z_4^+ \equiv (x_1, y_3)$	$(x_1, y_3) \equiv z_4$	$(x_2, y_1, y_3) \equiv z_5$
$z_5^+ \equiv (x_2, y_1, y_3)$	$(x_1, y_2, y_3) \equiv z_6$	$(x_2, y_1, y_3) \equiv z_6$
$z_6^+ \equiv (x_1, y_2, y_3)$	$(x_1, y_3) \equiv z_4$	$(x_2, y_1, y_3) \equiv z_5$

old states	New states after reading a	b
z_2	z_3	z_2
z_3	z_4	z_2
z_4^+	z_4	z_6
z_5^+	z_6	z_5
z_6^+	z_4	z_5

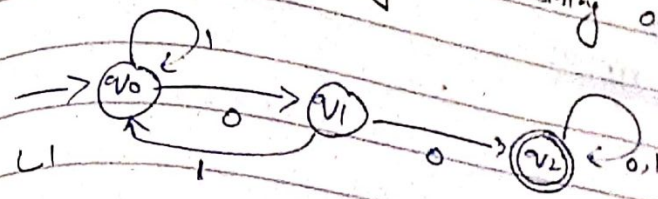


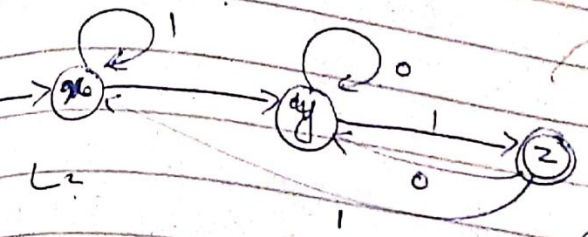
Example #2

L_1 = All strings Containing 00

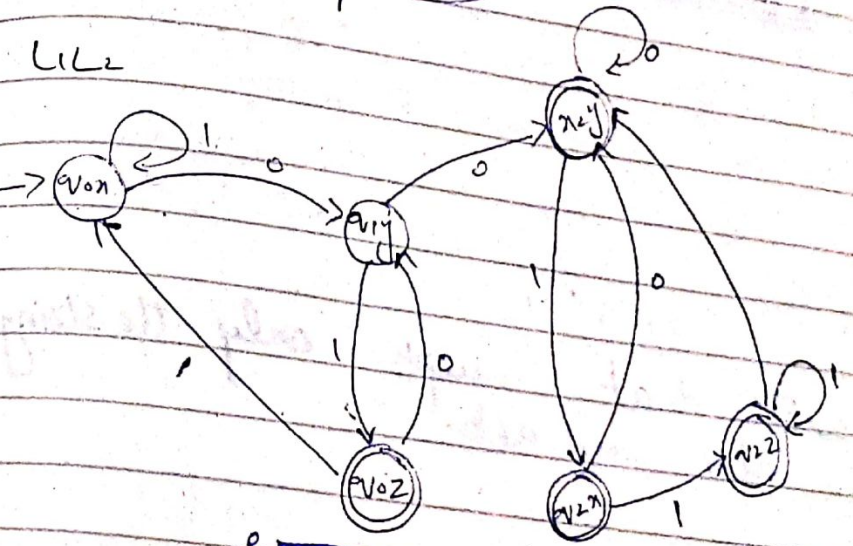
L_2 = All string Containing 01

L_1L_2 = All string Containing 00 or ending with 01





L_1L_2



(3) Complement of RE

- The Complement of regular expression is a negation.
- For Complement, Convert final state to non-final and non-final to final state.

Example:

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

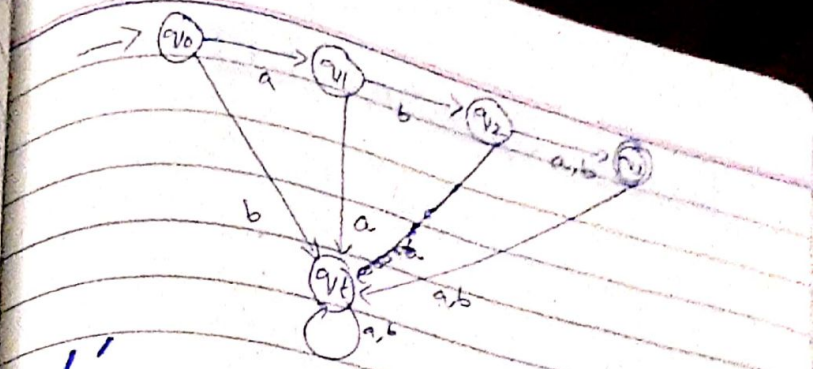
Example:

- if L is the language over the alphabet $\Sigma = \{a, b\}$ of all words that have a 'double a' in them.

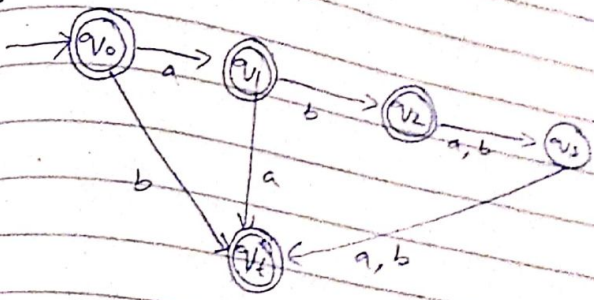
Then L' is the language of words that do not have 'double a'.

$(L')' = L$

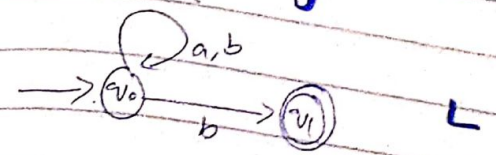
=> FA that accepts only the string aba and abb.



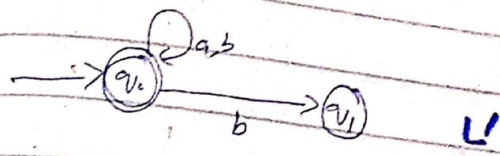
L'



#FA for language that ends with b.



L



L'

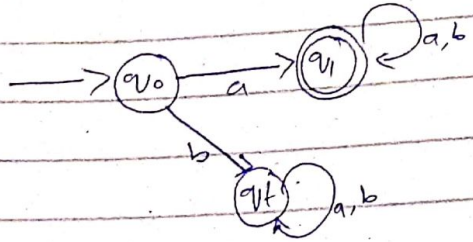
(4) Reverse Operation of RE

=> Regular languages are closed under reversal operation.
=> it is denoted by L^R.
=> The following steps are for this.

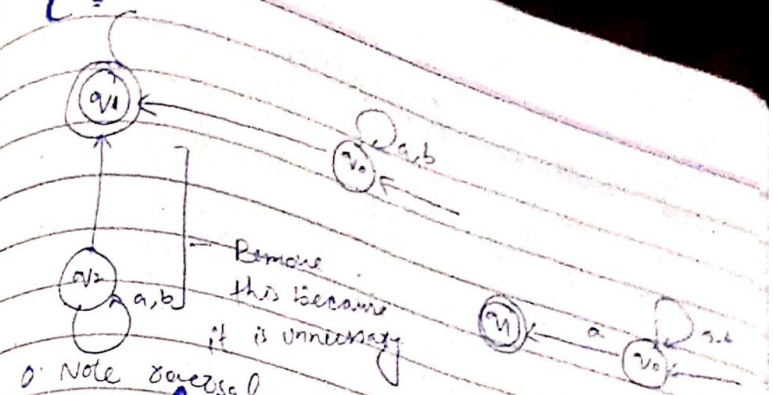
1) Make initial state of FA as final state.
2) And make the final state as initial state.
3) Reverse the direction of edges.
4) No change in Loop and Remove unnecessary conditions

Example:

L = language of words start with a
$L = a(a+b)^*$



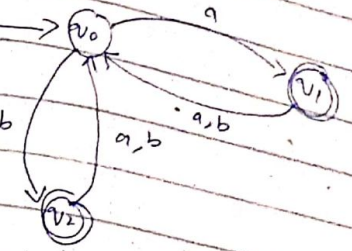
L^R



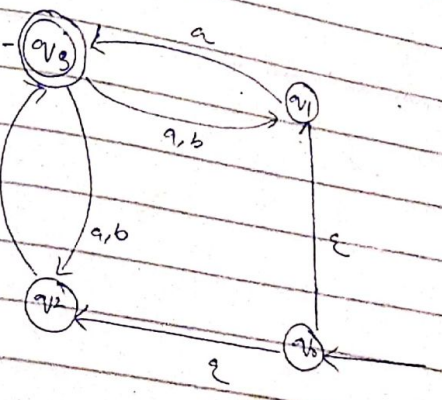
Note reversal of DFA is not always DFA can be NFA

Reversal for more than two final states.

L:

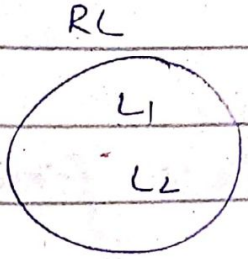


L^R



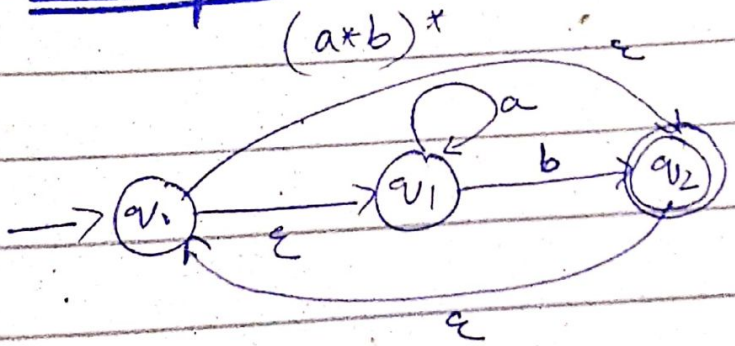
(5) Kleene Star Operation

- Kleene Star of regular language is closed under Kleene Star operation.
- it is denoted by L^*.

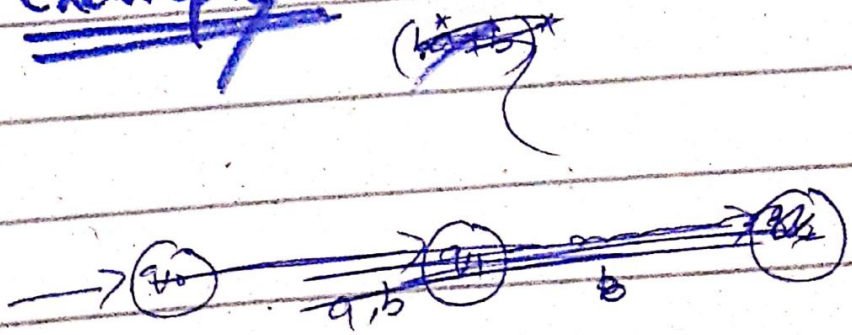


$(L_1 L_2)^*$ it is closed.

Example:



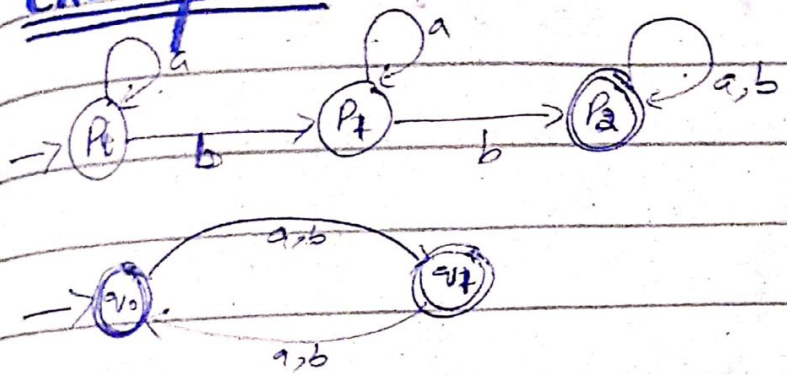
Example:



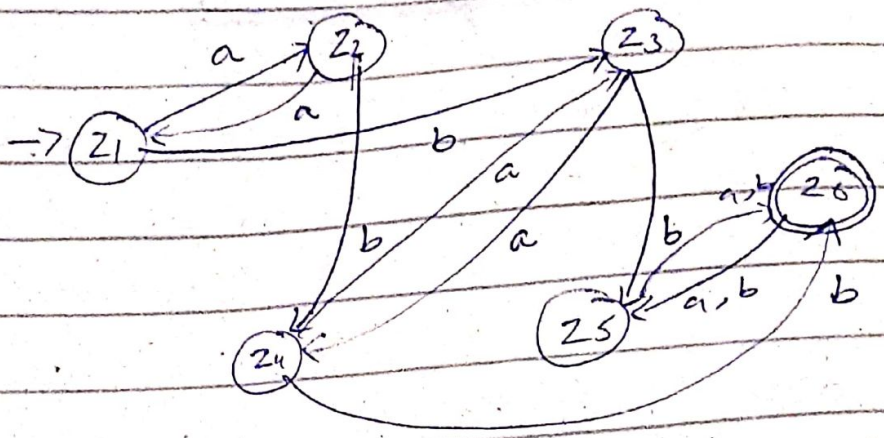
(b) Intersection Operation

- Pass those strings that will be pass by both FA.
- Resultant FA will pass strings that are common.
- Final state will be that state contains final states of both FA.

Example



δ	a.	b
$q_0P_0 = Z_1$	$q_1P_0 = Z_2$	$q_1P_1 = Z_3$
$q_1P_0 = Z_2$	$q_0P_0 = Z_1$	$q_0P_1 = Z_4$
$q_1P_1 = Z_3$	$q_0P_1 = Z_4$	$q_0P_2 = Z_5$
$q_0P_1 = Z_4$	$q_1P_1 = Z_3$	$q_1P_2 = Z_6$
$q_1P_2 = Z_6$	$q_0P_2 = Z_5$	$q_0P_2 = Z_5$
$q_0P_2 = Z_5$	$q_1P_2 = Z_6$	$q_1P_2 = Z_6$



Homomorphism Property:

=> Also called substitution function.

=> Regular languages are closed under homomorphism.

=> Suppose Σ and Γ are alphabet. The a function $h: \Sigma \rightarrow \Gamma^*$ is called homomorphism

1) $h(\varepsilon) = \varepsilon$

2) $h(x,y,z) = h(x) \cdot h(y) \cdot h(z)$

Example: $\Sigma = \{a,b\}$ $\Gamma = \{0,1,2\}$

$h(a) = 010$, $h(b) = 102$

$h(abba) = h(a) \cdot h(b) \cdot h(b) \cdot h(a)$

$= 010 \cdot 102 \cdot 102 \cdot 010$