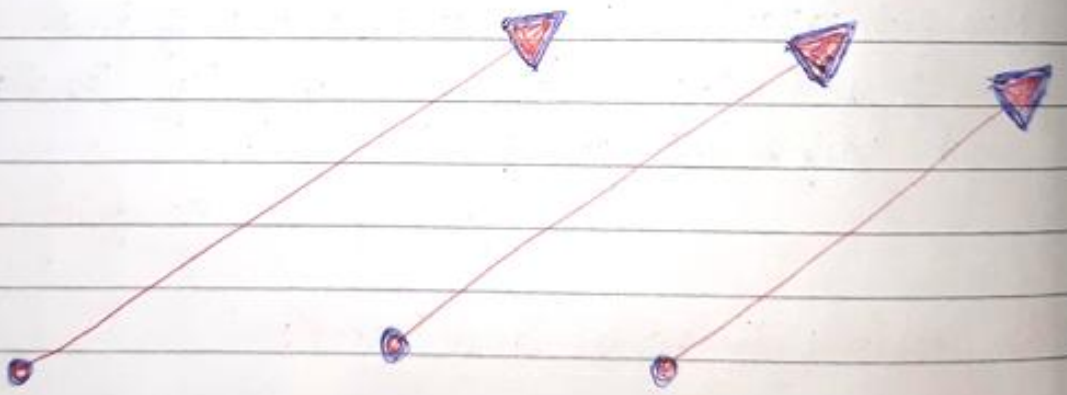
"THEORY"

"OF"

"COMPUTATION"



"GATE"

"THEORY OF COMPUTATION"

→ Introduction to Theory of Computation & DFA (Deterministic Finite-Automata).

↳ "TOC" → "Theory of Computation" It is something about "computation".

↳ "Computation" :→ Computation is any task that can be performed by a calculator or a computer..

So, we are going to mathematically model a "computer" or any "general machine" & we are going to study about the theory related to this machine.

↳ Which means what are the capabilities of this machine, what are all the "problems" that can be "solved by this machine" & so-on.

↳ What are the limitations of such a machine are?

→ "Basics of TOC"

1> Symbol :→ It is the basic building blocks of this subject. A symbol can be any letter, or digit or anything, either pictures.

Eg :→ "a, b, c, 0, 1, ☺ &."

2> Alphabet :→ From symbols we can form alphabets. Alphabet is generally represented by "Σ". "Alphabet" is nothing but some collection of these "symbols".

Eg :→ $\Sigma = \{a, b\}$ Once, we define the alphabet all the strings, will be based on this predefined set called alphabet.

P.T.O

Note :- There can be any number of 'symbols' present in the 'alphabet'.

Eg:- $\{a, b, c, \}$, $\{0, 1, 2, 9\}$

"It is a finite set"

2) String :- A 'string' is a sequence of symbols present in the alphabet

Eg:- if $\Sigma = \{a, b\}$

then string = "a, b, aa, bb, ab, aba, ---"

Now possible questions can be, given an alphabet eg $\Sigma = \{a, b\}$

then find out how many strings of length '2' are possible on the alphabet.

= 1 2

a a
a b
b a
b b

} 4 strings

Now find out how many strings of length 'n' are possible on this alphabet $\Sigma = \{a, b\}$

$\{a,b\}$ $\{a,b\}$ ---------- $\{a,b\}$ $\}$ n spaces

Each blanks can be filled in two way

∴ No. of string = $2 * 2 * 2 * 2 - - - - - n$

= 2^n → 'length'

"No. of symbols"

General formula is :- Suppose the no. of symbols in a "Σ" is represented by "$|\Sigma|$" then no. strings of length n.

$= |\Sigma|^n$

→ Language :→ In "ToC" a language is nothing but a "collection of strings"

Eg:- If $\Sigma = \{a, b\}$

L_1 = "Set of all strings of length '2'"

$= \{aa, ab, ba, bb\}$

"L_1 is a finite language"

L_2 = "Set of all strings of length '3'"

$= \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$ "8 strings"

"L_2 is also a finite language".

L_3 = "Set of all strings where each string starts with 'a'"

$= \{a, aa, ab, aaa, aab, aba, abb, ------\}$

"L_3 is an "infinite language"

Note :→ A "language" which can be formed over an alphabet 'Σ' can be either "finite" or "infinite".

→ Powers of Σ :→

Let us say we have defined Σ as $\Sigma = \{a, b\}$

This alphabet consists of two symbols 'a' & 'b' only.

P.T.O

Σ^1 = Set of all strings which can be formed over $\Sigma\{a,b\}$ which are of length 1 exactly.

$= \{a, b\}$

$\Sigma^2 = \Sigma.\Sigma = \{a,b\}.\{a,b\}$ → "Concatenation"

$= \{aa, ab, ba, bb\}$

Set of all strings which can be formed on Σ of length 2

$\Sigma^3 = \Sigma.\Sigma.\Sigma = \{a,b\}.\{a,b\}.\{a,b\}$

∴ Cardinality of this set is $|\Sigma| = 8$

:

: so-on

:

$\Sigma^n = \Sigma.\Sigma.\Sigma \ldots \Sigma$ (n times). Set of all strings over Σ of length n

Σ^0 = Set of all strings over Σ of length "zero".

Smallest string of $\Sigma^0 = \{G\}$

- "epsilon" or "null string".
- string of length "zero"

∴ $|G| = 0$

length of 'G' = '0' strings

→ Σ^* :⇒ { In place of '*' we can substitute any number starting from '0' }.

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

$\therefore \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$

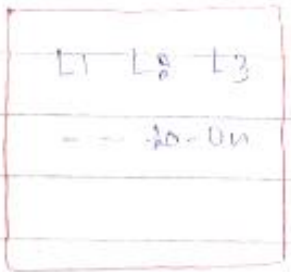
$= \{\epsilon\} \cup \{a, b\} \cup \{aa, ab, ba, bb\} \dots$

"It is infinite"

'Note' :⇒ Σ^* is nothing but set of all strings possible over $\Sigma = \{a, b\}$ here. Can also be called as mother of all lang. over Σ. So we also be called as a universal set.

Once defined "Σ" & "Σ^*" we can say that, the languages that have been defined over Σ are actually parts of this Σ^*

Σ^*



$\therefore \{L_1 \subseteq \Sigma^*, L_2 \subseteq \Sigma^*, L_3 \subseteq \Sigma^*\}$

A language is nothing but any subset, it can be "finite" or "infinite". which we should be able to define precisely

Note:⇒ No. of strings formed over "Σ" is infinite, similarly no of languages formed over "Σ" is also infinite.

There are many practical applications, where we are going to use these "strings & languages".

"P.T.O"

Let us see one practical application, where "strings" & "languages" are actually used :->

So, we can consider the "C" programming language.

Let us assume that we are having a language called "C-programming language".

The symbols that we type while writing a "C-program" are :->

$\Sigma = \{ a, b, \ldots, z, A, B, \ldots, Z, 0, 1, \ldots, 9, +, *, \ldots \}$

So, this is a finite set & we are always going to use a finite "no. of symbols" in order to type a C-program.

∴ Alphabet for any C-program is going to be ["Σ"] defined above.

Now using these "symbols" we are going to form "strings"

So, if we write like :->

```
void main()
{
    int a, b;
    ;
}
```

↳ It is nothing but a "C-program" but compared to "TOC" it is nothing but a "string"

& a program is nothing but a "string"

Now an interesting question to be asked over here is, What is "C-programming language" ?

↳ It is nothing but set of all "valid programs".

Here the important word is "valid", but there can still be some invalid programs which can give errors when we try to compile it or run it.

So, the interesting question here is, "What are the valid programs" and given any program, how can we say that, the program is "valid" or "invalid".

First Question is :⇒ "How many programs are possible in C-programming language"?

Answer is = "Infinite" $\{P_1, P_2, P_3, P_4, \ldots \ldots \ldots P_n\}$

"infinite no. of programs"

Next Question is :⇒ Given any program "Pn" How can we say that it is valid?

So, a program is "valid" only if, it is present in this "set".

One of the naive solutions is :⇒ store all these programs $\{P_1, P_2, P_3 \ldots P_n\}$ in Computer and given any program "Pn", scan it with the list. to identify whether it is valid or not.

But the problem with this solution is that, any computer will have a finite memory, but the no. of progs. possible is 'infinite'; so this is just not possible.

So, we are generally interested in, given a language "L" & a string "S".

Now the interesting question that can be asked is, whether this string "S" is present in the language "L" or not.

if the lang. is infinite, obviously we cannot store the entire language on the Computer due to "memory constraints".

So, the entire explanation boils down to one point :⇒

if 'L' is finite

Eg:- If $\Sigma = \{a, b\}$ & L_1 is set of all strings defined over the alphabet 'Σ' with length '2'.

then 'L_1' = $\{aa, ab, ba, bb\}$

Now given any string like "S" = "aaa". Now if we ask a question whether this string 'S' is present in language "L_1" or not.

then we can verify it by comparing "S" with each element of "L_1" & we can say that it is not present in language "L_1".

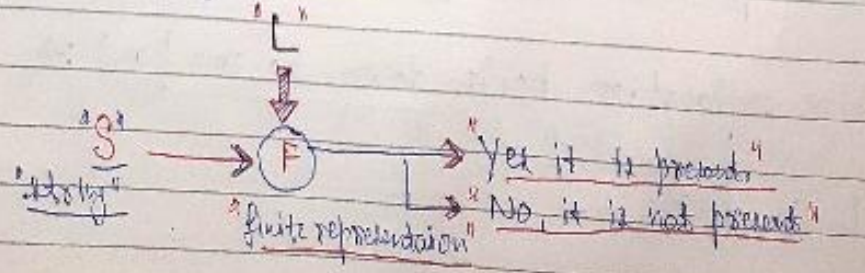
Here the language "L_1" is "finite" we can do this.

Now let us consider a language which is infinite. Let "L_2" be a language consisting of all strings defined over $\Sigma = \{a, b\}$ starting with 'a'. Clearly we can say that the language is infinite.

then 'L_2' = $\{a, aa, ab, aab \text{ ---- }\}$ 'infinite'

Now suppose string given is $S = \{baba\}$ & we have to find out whether this string is present in 'L_2' or not. then the problem is how long we should be comparing it. If we are linearly search for this string & matching it against the lang. then the machine will take forever time to do this.

So, the simple thing is how can you, come up with a finite representation for a "language". which can be stored in the memory, using which if we give a "string" we should be able to say whether the string is present there in the "language or not".



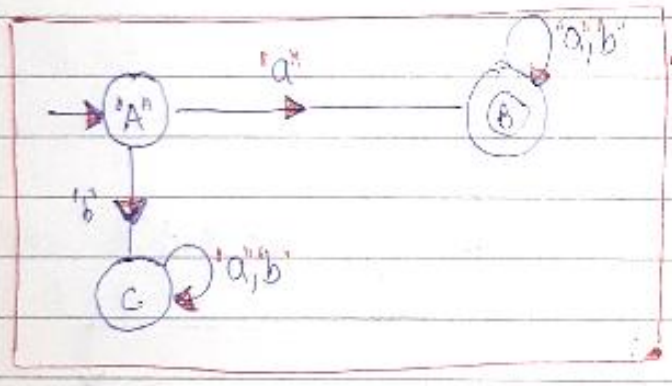
Let us now take an example to see, "what this finite representation" could be :→

The finite representation of a language is called "finite-Automata" & why this "finite-automata" if we give a string to this finite-Automata, It will say "yes" it has accepted the string or "No" it has not accepted the string.

Let us try to draw a "finite representation" for a language :-

"L_1" = "Set of all strings which starts with 'a' & $\Sigma = \{a, b\}$"

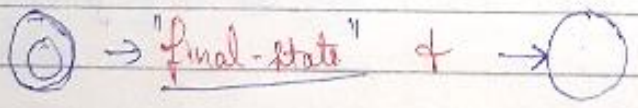
$= \{a, aa, ab, aaa, \text{-----}\}$



"Finite representation for this language "L_1".

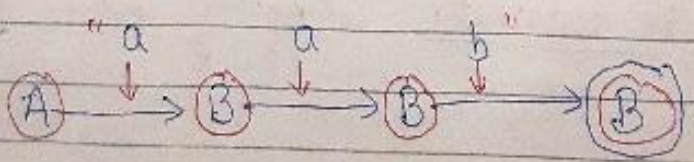
It is also known as a "finite automata" or "DFA"

All these circles are "states" Let are having three states 'A', 'B' & 'C' here



So, initial state is 'A' & final state is "B"

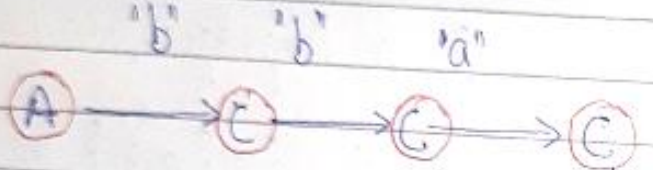
Now given a string = "aab" we need to find out whether it is present in the language or not.



A "string" is said to be "accepted" by "finite automata" if upon scanning a string, we reach from initial state to the final state.

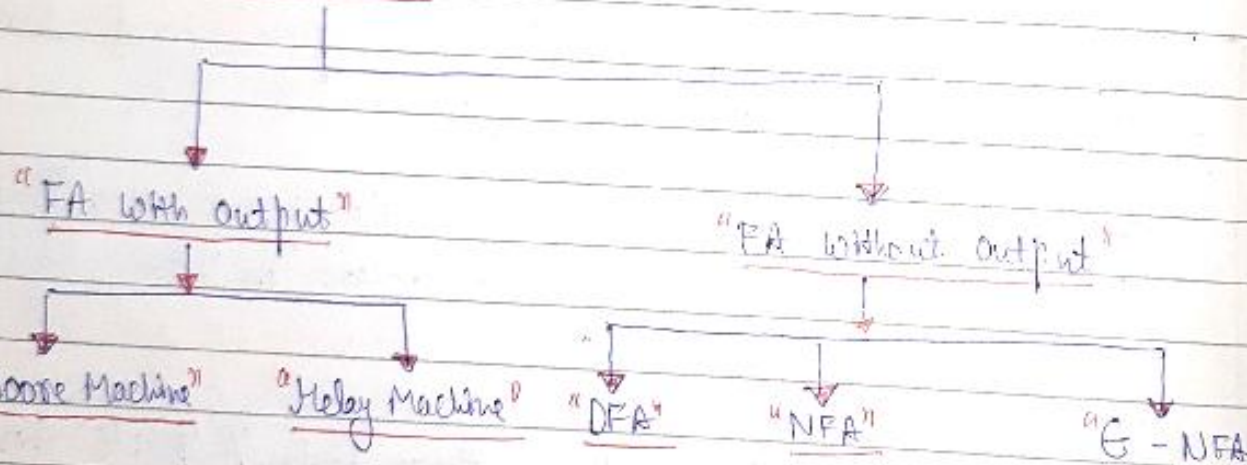
∴ "aab" is accepted

Now let us take the string "bba".



→ since "C" is not a final state hence string "bba" is rejected.

→ "Finite - Automata" :→



1> "DFA (Deterministic Finite Automata)" :→ It is a finite automata which contains

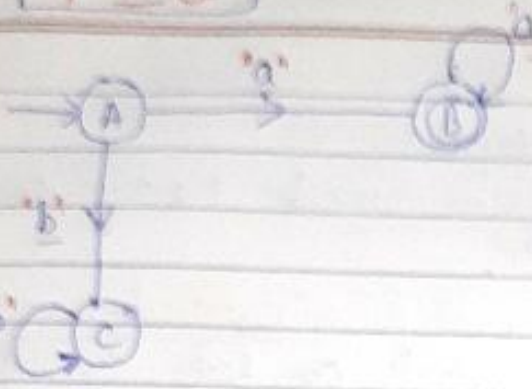
$(Q, \Sigma, \delta, q_0, F)$

Generally the "DFA", "NFA", or "∈ NFA" are all represented by this "quintuple".

Let us take a "finite automata" accepting some "infinite language".

Note :- "DFA" can have more than one "final state"

["F ⊂ Q"]



This is also known as a "transition diagram"

Now in this.

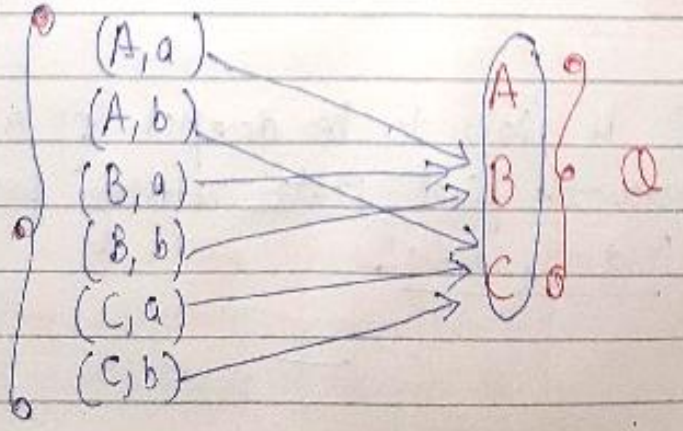
i. "Q" is set of all states.	$Q = \{A, B \& C\}$
ii. "Σ" is the alphabet.	$\Sigma = \{a, b\}$
iii. "q_0" is the starting state	$q_0 = \{B\}$
iv. "F" is set of final states	$F = \{B\}$

Till now, this "Q", "q_0", "F" & "Σ" are common for all "DFA" "NFA" & "ε-NFA" only difference is with "δ" which is transition function.

Let us now define 'δ' delta

So, 'δ' is a transition function from "$Q \times \Sigma \rightarrow Q$"

$\{A, B, C\} \times \{a, b\}$



This is the 'transition function'
('state of every input')

Note :> In 'DFA' for every input there will be exactly one transition & the transition is exactly "1".

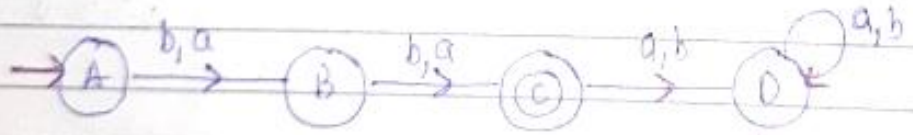
It is called 'DFA' because if we see any state, on seeing an input it is exactly going to only one "other state" whereas in NFA on seeing an input we might go to many states.

Now, let us take some examples to construct "DFA's" :->

Q:> Construct a 'DFA' that accepts set of all strings over {a,b} of length 2.

Sol:> $\Sigma = \{a, b\}$ & $L = \{aa, ab, ba, bb\}$

Once you identify the language, then take smallest possible "string" & construct a skeleton & then reform it.



"Dead state"

'D' is acting like a dead state, because when we reach 'D' we are never going to reach the "final state" or any other state.

Note :> A "string" is said to be accepted by a "final automata" if we are able to reach the "final state" starting from the "initial state" when scanning the "entire string".

Note :- If the language is finite, we will definately be able to give "finite automata" but if it is infinite we don't know whether we would be able to "give FA" or not!

→ "String Acceptance" :- A string is accepted by an FA, if upon scanning the entire string if we reach [a] final state from the initial state.

→ "Language-Acceptance" :- A language is said to be accepted, if we accept all the string of the language as well as the string which are not present there in the language should be rejected by the automata.

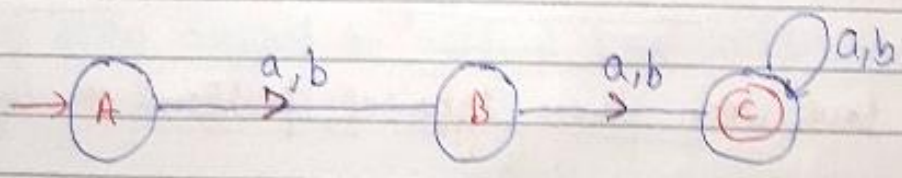
"CONSTRUCTION OF MINIMAL DFA" :-

We have constructed a DFA for set of all strings over {a,b} with length '2' Now what if we have to construct a 'DFA' for set of all strings {a,b} with length atleast '2".

Q:- Construct a DFA such that "$w \in \{a,b\}^*$ & $|w| \geq 2$".

Sol:- Here $\Sigma = \{a,b\}$. Now $L = \{aa, ab, ba, bb, aaa, \ldots bbb \ldots\}$
↳ This language is infinite".

Take the "smallest string" & give the direction & then try to "reform it".



"State 'A'" represents set of all strings of length '0'.
"state 'B'" represents set of all strings of length '1'
& "State 'C'" represents set of all strings of length '2' or greater than or equal "2"

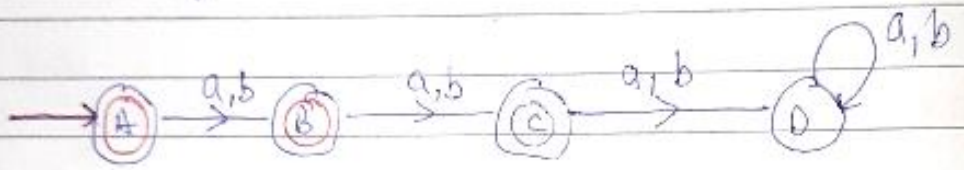
Note:> Whenever we have to make "ε" to be accepted by "DFA". Make "initial_state" = final_state!

Q:> Construct a DFA over $w \in \{a, b\}$ such that $|w| \leq 2$.

Sol:> This Lang. will consists of strings of length = [0 '0s' | '0s' '2']

$L = \{\epsilon, a, b, aa, ab, ba, bb\}$ It is a finite lang.

Note:> Whenever the language is "finite", draw the skeleton of the "Biggest-string".



[State 'A' represents all strings of length '0'.
State 'B' represents all strings of length '1'
State 'C' represents all strings of length '2'
State 'D' represents dead-state.]

Note:> Given a language we can draw many "DFA's". But then the language will always contain only one "minimal DFA".

Note:> "Minimum no. of states" for the language is ["4"] only.

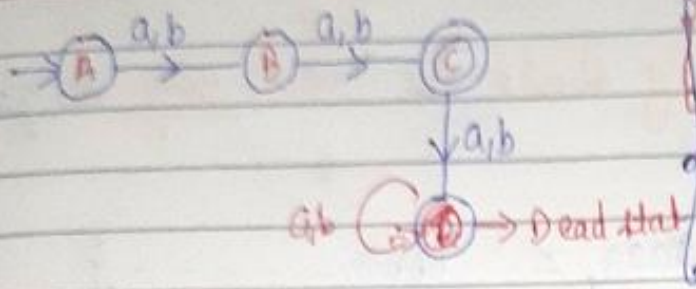
There is a procedure to "Construct a DFA" & then "minimize it".

Till now we have seen three type of problems :>

1> with length exactly [2] $|w| == 2$
2> with length atleast [2] $|w| \geq 2$
3> with length atmost [2] $|w| \leq 2$

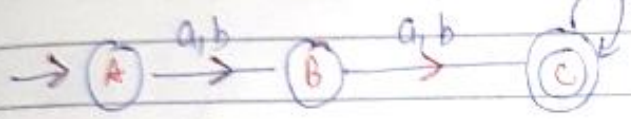
"Length of strings"

1) With length exactly [2] for $\Sigma = \{a, b\}$:⇒ "$|w| = 2$"



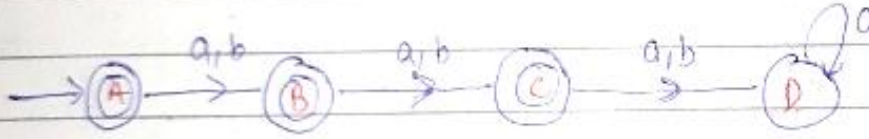
"3 states" are required to accept strings of length = 2, & "1 more state" is required to reject all other strings of length greater than "2".

2) With length atleast [2] for $\Sigma = \{a, b\}$:⇒ "$|w| \geq 2$"



3 states are required to accept any string greater than length = 2

3) With length atmost [2] over $\Sigma = \{a, b\}$:⇒ "$|w| \leq 2$"



Similarly here

Now if we look at these, the no. of states in strings of [length 2" = n" no of states in strings of length [$|w| \geq 2 = 3$] & string of length [$|w| \leq 2$ = "4".]

"Most - Important"

∴ in general, if we have to accept strings of length [$|w| = n$"]

then no. of states required in "minimal DFA" which accepts all the strings of "length = "n"" is $(n+2)$.

if $|w| \geq n$ No of minimal states required is $= (n+1)$.

if $|w| \leq n$ No. of minimal states required is $= (n+2)$.

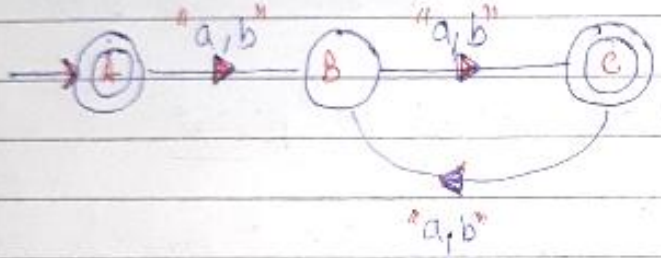
Note :- "Finite Automata" is called "finite", because no. of states in "finite-automata" is "finite".

Q:- Construct a minimal DFA which accepts all strings over $\Sigma = \{a, b\}$ such that "$|w| \bmod 2 = 0$".

Sol:- Here the length of the string should be "even" only, to get accepted by this minimal DFA.

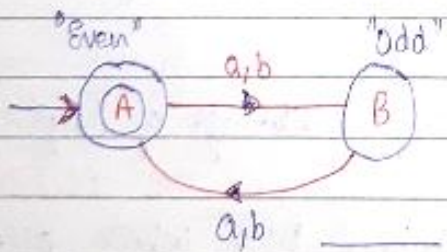
$L = \{ \in, aa, ab, ba, bb, aaaa, \dots bbbb \dots \}$

The language is "infinite here" :-



This can further be minimized as :-

we require "two states only" in which "one state" will accept all the "even length strings" & other "will reject it".



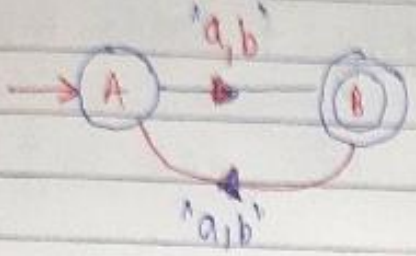
"Minimal DFA", for set of all strings whose length is divisible by "2".

Q:- DFA (minimal DFA) which accepts all strings over $\Sigma = \{a, b\}$ such that "$|w| \bmod 2 \neq 0$" ?

Sol:- Here the length of the strings should be "odd" only.

$L = \{ a, b, aaa, bbb, aba \dots \}$ odd length strings

The lang. is "Infinite here also":→



Note:→ If string length has to be divisible by '2' then remainder can be either '0' or '1' ∴ "two states"

if string length has to be divided by '3", then three remainders are possible:→ '0', '1' or '2' ∴ "three states"

Q:→ DFA (minimal) which accepts all strings over $\Sigma = \{a,b\}$ such that "$|w| \bmod 3 = 0$".

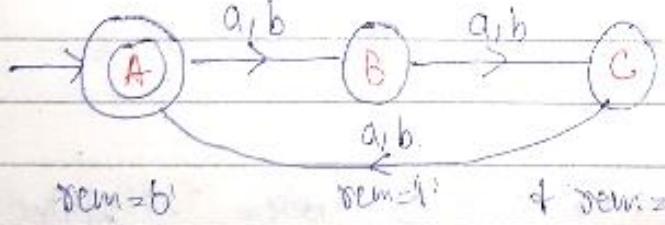
Sol:→ $L = \{\in, aaa, aab, \ldots bbb, aaaaaa \ldots\}$

↓

"0/3 = 0"

"Infinite Language"

When we divide any no. by '3' the possible remainders are ["0,1" or "2"].



These three states with distinguish b/w the remainders.

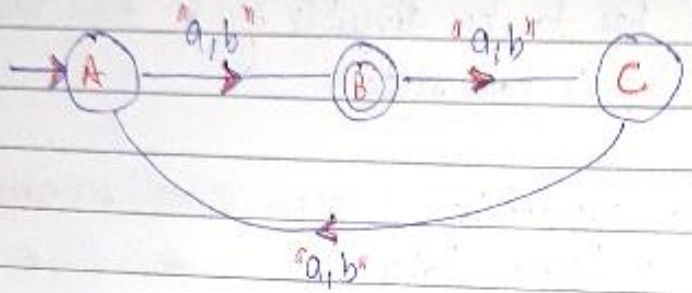
State "A" denotes all strings whose length when divided by "3" gives remainder = '0'.

State "B" denotes all strings whose length when divided by "3" gives a remainder = '1' & State 'C' when remainder = '2'.

Q:> $|w| \equiv 1 \bmod 3$

It means if we take length of a string & divide it by '3' the remainder should be '1'.

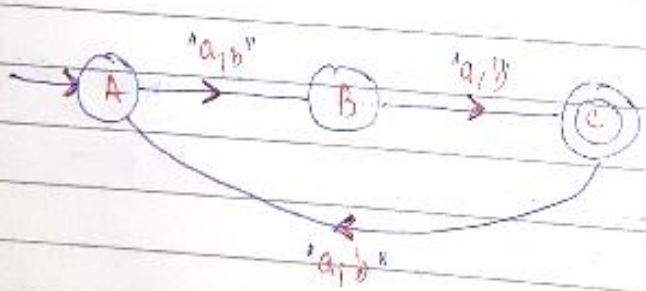
then DFA =>



Q:> $|w| \equiv 2 \bmod 3$.

It means if we take "length of a string" & divide it by "3" the the remainder should be "2".

the DFA is :=>



Note:> In general if $|w| \bmod n = 0$ then "n" states will be there in minimal DFA

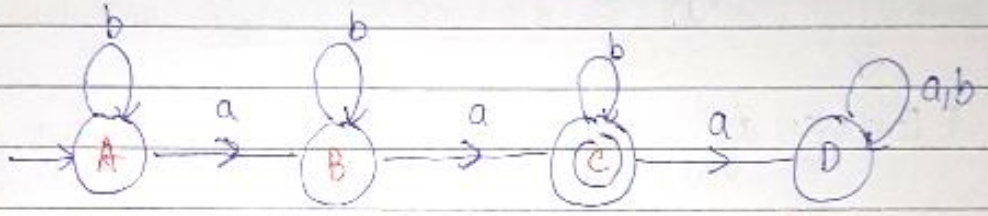
Questions can also be framed like no. of a's or no of b's in a string for that if we have to construct a minimal DFA the

Q:⇒ Construct minimal DFA over $\Sigma = \{a, b\}$ which accepts all the strings $w \in \{a, b\}^*$ where no. of a's = "2" i.e. $[n_a(w) = 2]$., that can be any no. of b's in the strings. (No restriction on that)

Sol:⇒ $L = \{aa, aab, aba, \text{---} \; aab \; \text{---} \; \}$

↳ "Infinite".

First thing is, we need to construct DFA for [2 a's]



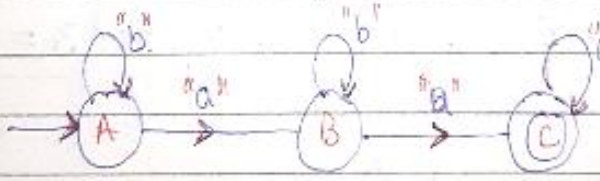
{Dead state}

State "A" → 0 a's
State "B" → 1 'a'
State "C" → 2 'a's

"Dead state" is a "state" from which there is "no path" to the "final state".

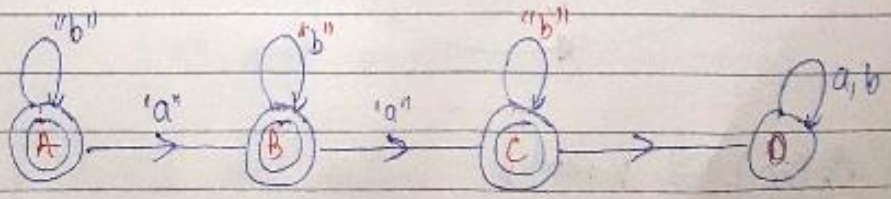
Q:⇒ DFA over $\Sigma = \{a, b\}^*$ such that "$n_a(w) \geq 2$".

Sol:⇒



Q:⇒ DFA over $\Sigma = \{a, b\}^*$ such that '$n_a(w) \leq 2$' at most '2"

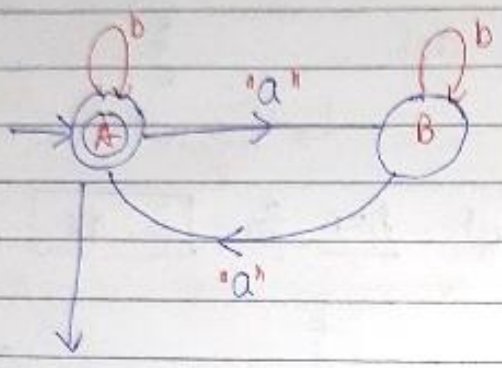
Sol:⇒



"P.T.O"

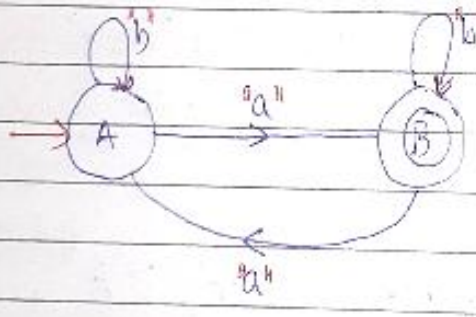
Q:> Construct a minimal "DFA" over $\Sigma = \{a, b\}^*$ such that length/no. of a's in string should be even? [$n_a(w) \bmod 2 = 0$]

sol:>



Q:> if "$n_a(w)$ = Odd".

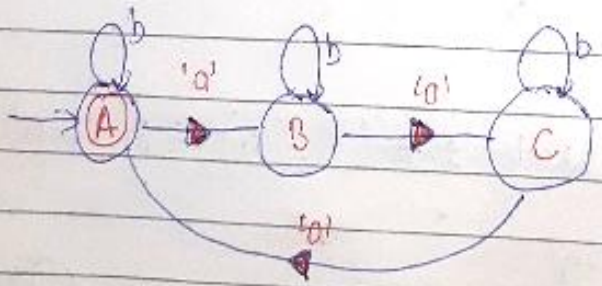
sol:>



"No. of a's is odd"

Q:> if [$n_a(w)$ = 'multiple of 3'.] or "$n_a(w) \bmod 3 = 0$"

sol:>



if $n_a(w) \cong K \bmod n$

No. of states would be ["n"] bcoz if we divide any

numbers with 'n', then we will get 'n' remainder ("0" to "n-1")

Till now, we have seen questions on string length of "no. of a's" Now we want to see questions, where we have some restrictions on number of "a's" as well as number of "b's".

Q:→ Construct a minimal "DFA" which accepts set of all strings over $\Sigma = \{a, b\}$ such that $n_a(w) \equiv 0 \bmod 2$ & $n_b(w) \equiv 0 \bmod 2$

It means each string should contain even no. of "a's" & "b's".

Note:→ [Every "even length" string need not contain "even no of a's & b's".]

Sol:→ $L = \{\epsilon, aa, bb, abab, baba, bbaa, \dots\}$ → "infinite language"

Note:→ Always the no. of states in a "minimal DFA" depends on the "no. of ways or no. of groups" depending on question.

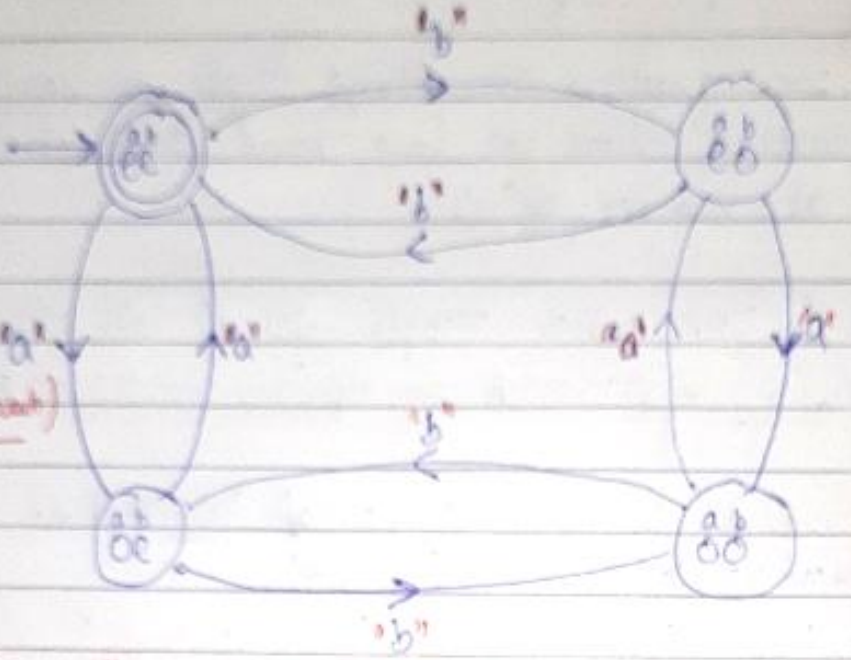
$n_a(w)$	$n_b(w)$	Eg:
even	even →	ϵ, aa, bb
even	odd →	aabb
odd	even →	baab
odd	odd →	ab

∴ We will have [one state representing (even-even).
2nd state representing (even-odd).
3rd state representing (odd-even)
4th state representing (odd-odd)]

So, we have found out that atleast we have "four states" to distinguish the "entire language" into "four classes".

P.T.O

the four states are :-

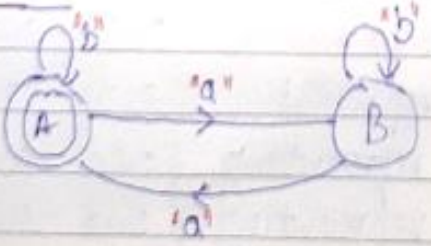


This is "one method" to construct the "DFA" for this question.

Q:→ Construct a "DFA" that accepts set of all strings over {a,b} in which number of "a's" and number of "b's" both are even.

Sol:→ "Second method" This method is also known as "Cross product method".

1) "Counting a's"



State "A" will count even no. of A's & State "B" will count odd no. of A's } [Here we are not counting no. of b's]

3 "Counting b's" :→

State "C" will count even no. of b's & State "D" will count odd no. of b's [Here we are not counting no. of "a's".]

Now, we need to combine both of them using "Cross-product method".

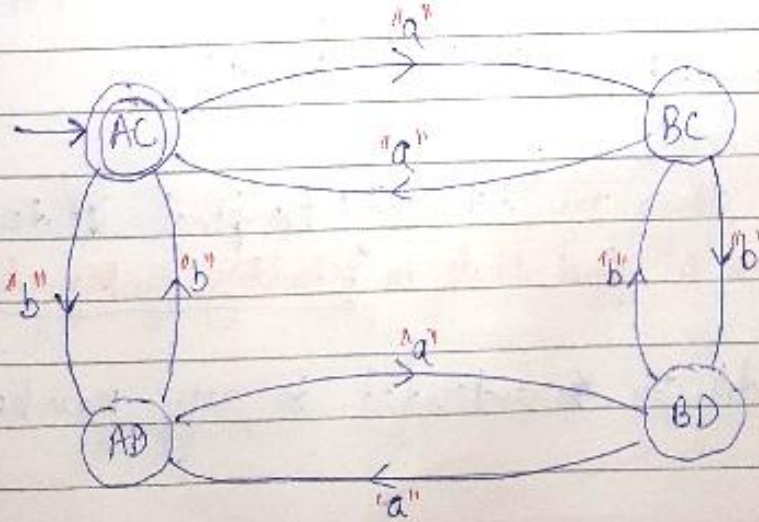
"Cross-Product Method" :→

Step ① First write no. of states in it.

$$\{A, B\} \times \{C, D\} = \{AC, AD, BC, BD\}$$

"first state writes" "Sub-first state"

Step ② Draw the states after Cross-product :→ ["AC" is final state & "AC" is the initial state]



P.T.O

"Step 3" :- After this we need to identify the transitions.

1> A, C —"a"→ B, C ∴ (AC) —"a"→ (BC)

2> A, C —"b"→ A, D ∴ (AC) —"b"→ (AD)

3> B, C —"a"→ A, C ∴ (BC) —"a"→ (AC)

4> B, C —"b"→ B, D ∴ (BC) —"b"→ (BD)

5> A, D —"a"→ B, D ∴ (AD) —"a"→ (BD)

6> A, D —"b"→ A, C ∴ (AD) —"b"→ (AC)

7> B, D —"a"→ A, D ∴ (BD) —"a"→ (AD)

8> B, D —"b"→ B, C ∴ (BD) —"b"→ (BC)

Since in individual diagrams "A & C" are the final states so combination "AC" would be a final state in "resulting automata".

Note:- Now, this model can be extended to any number of questions.

[Count no. of a's horizontally & no. of b's vertically.]

Q:⇒ Construct a "DFA" over $\Sigma = \{a,b\}$ such that number of a's is even or number of b's is even

"no a's" or "no b's"
↓ ↓
e e {e,e}

So, here [(e,e)] is the final state
moreover [(e,o)] can also be "final state" because here no. of a's is even & further [(o,e)] can also be a "final state" because no. of b's is even.

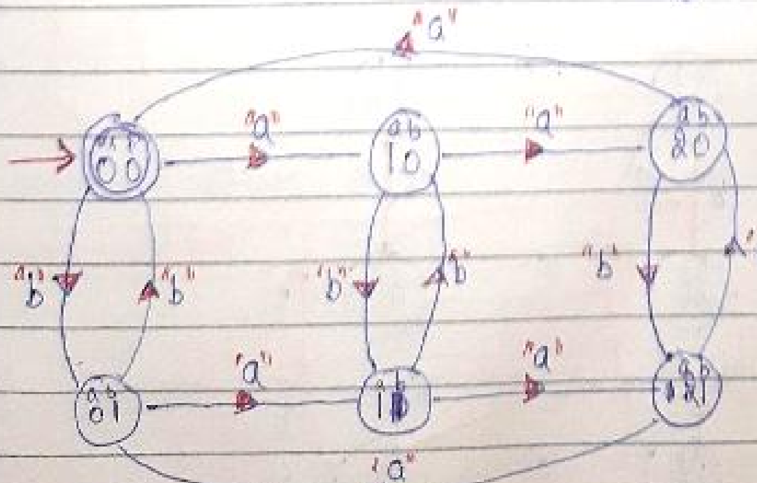
Only state which cannot be final in this case (o o)

Note:⇒ If one "DFA" contains ['n'] states & other "DFA" contains ['m'] states then cross-product of these two will contain ["n * m"] states.

Q:⇒ Construct a "DFA" over $\Sigma = \{a,b\}$, which accepts all the strings in which no. of a's is divisible by '3' i.e. $n_a(\omega) \cong 0 \bmod 3$ & no. of b's is divisible by '2' i.e. $n_b(\omega) \cong 0 \bmod 2$.

Sol:⇒ So, here first 'DFA' contains ['3' states] & 2nd 'DFA' contains ['2' states] & hence the final automata will contain (3 * 2) i.e. [6 states].

So it can be constructed directly as :⇒



P.T.O

[illegible] $n_a(w) \bmod 3 \geq n_b(w) \bmod 2$

$\{10, 20, 21, 11\}$ so these are the final states.

ii) $n_a(w) \bmod 3 = n_b(w) \bmod 2$

$\{00, 11\}$

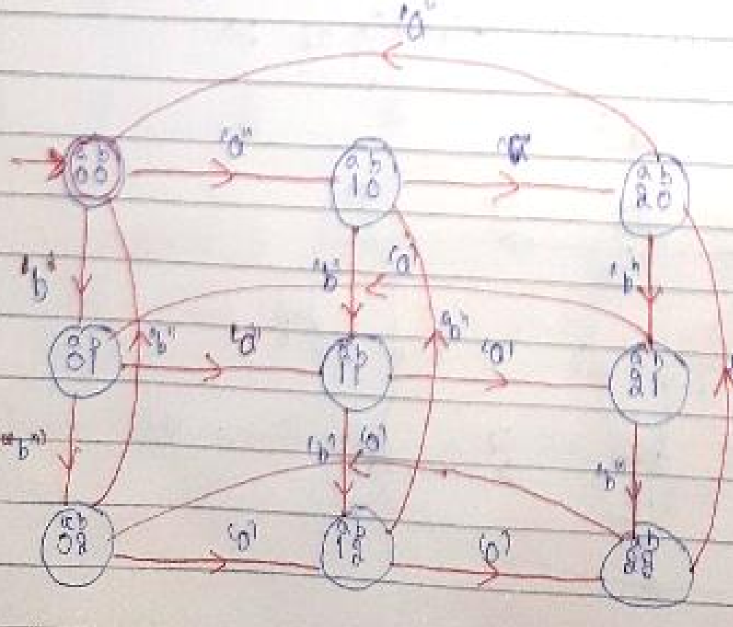
Q.) Construct a minimal 'DFA' which accepts set of all strings over $\{a, b\}$ in which number of 'a's' are divisible by 3 & number of 'b's' are divisible by 3.

Sol.) $n_a(w) \cong 0 \bmod 3$ & $n_b(w) \cong 0 \bmod 3$

No. of states in final automata = '9'

→ final state is "00"

So, using the same method of counting no. of 'a's' horizontally & no. of 'b's' vertically we get.



These numbers are actually numbers divided with '3'

Now suppose the question is:- $[n_a(w) \bmod 3 = 1]$ & $[n_a(w) \bmod 3 = 2]$

Here final states = "1"

Now suppose the question is:- $[n_a(w) \bmod 3 > n_b(w) \bmod 3]$

Final states are = {"10", "20" & "21"}

The above question can also be asked like $[(n_a(w) > n_b(w)) \bmod 3]$

Now in general if $n_a(w) \neq 0 \bmod(n)$ & $n_b(w) \neq 0 \bmod(m)$ then no. of states (minimum no of states in FA) = $[m \times n]$.

Q:- Construct a minimal DFA which accepts set of all strings over {0,1} in which the alphabet $\Sigma = \{0,1\}$ when interpreted as binary numbers is divisible by "8".

Sol:- $\Sigma = \{0,1\}$ $\quad w \in \{0,1\}^*$

Now if we take any string say "110" & if we interpret this this string to be a "binary number"

$(110)_2 = (6)_{10}$ then it should be divisible by "2".

"Base"	"Nomenclature"	"Symbols"
1-	Unary	0
2-	Binary	0, 1
3-	Ternary	0, 1, 2
10-	Decimal	0, 1, 2, 3 - - - 9
16-	Hexadecimal	0, 1, 2, 3 - - 9, A, - F

P.T.O

Go Page

mycompanion

Some Interesting points about "Binary Number System" :-

1> In binary number system, how are numbers formed & how are they validated.

Eg:- suppose we are having a digit '1' & we appen '2' after it, it becomes 12,

"12" (for decimal)

↳ 1 * 10 + 2 = "12"

Base

Now suppose No is "12" & we append "3" to it then it becomes "123"

↳ 12 * 10 + 3 = 120 + 3 = "123"

↓ ↓ ↓

"Existing no." | "base" | "new no to be added"

1456 (in decimal).

1 * 10 + 4 = "14"

14 * 10 + 5 = "145"

145 * 10 + 6 = "1456"

Now let us extend this method of "base multiplication" to binary

Let the number be $(10)_2$ Now let us try to find its "decimal value"

1 * 2 + 0 = 2 + 0 = "2"

$(101)_2 = (5)_{10}$

1*2+0 = 2

2*2+1 = 5

Similarly for others

Now, let us get back to the 'question':

The no. system is binary, so here in question, we need to construct a "minimal-DFA" which accepts all strings over {0,1}, as binary nos divisible by "2".

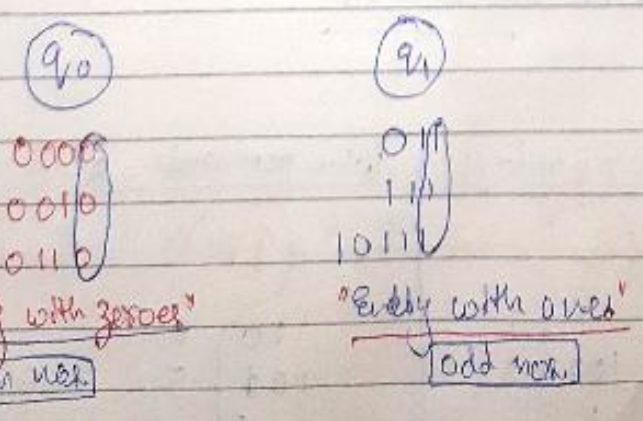
Note:⇒ When we divide any no by "2" we will get only '2 remainders'

[1> '0' ; 2> '1'] ⇒ we get two remainders, if we divide any number by "2"

So we can classify all the "binary nos" into two classes.

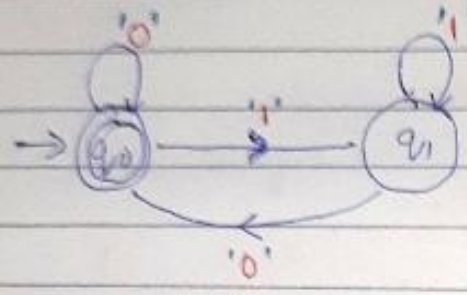
1> when divided by "2", they will generate "0" as remainder.

2> when divided by "2", they will generate "1" as remainder.



P.T.O

Note:- If we get '0' & on dividing it with "2" we will get a remainder "0" so we need to remain in state ["q_0"].
& If we get '1' & on dividing it with "2" we will get a remainder "1" so we need to move to state ["q_1"].



["q_0" state indicates → remainder is ['0'] when divided by "2"
&
"q_1" state indicates → remainder is ['1'] when divided by "2".]

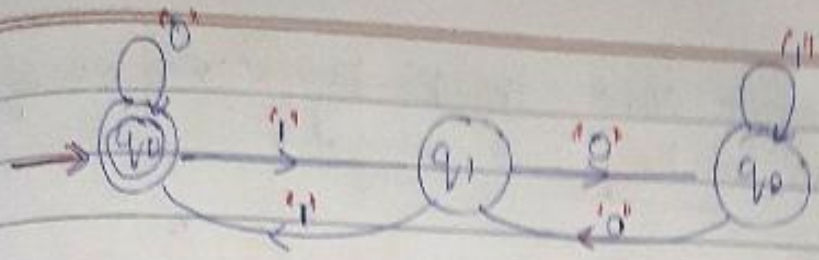
Q:- Construct a minimal "DFA" which accepts set of all strings over $\{0,1\}$ which when interpreted as a binary number is divisible by "3".

Sol:- $\Sigma = \{0,1\}$ $\quad w \in (0,1)^*$

Now when we divide any no. by "3", the possible remainder are three actually.

1> → "0"
2> → "1"
3> → "2"
} Possible remainders.

"When remainder=0"	"When remainder=1"	"When remainder=2"
q_0	q_1	q_2
0 — 0	1 — 1	10 — 2
11 — 3	100 — 4	100 — 5
110 — 6	⋮	⋮
⋮		



When we see "0" & when we divide it by "3" we get "0" as remainder, so on seeing "0" at "q_0" we are going to remain there only. Now if we see "1" at "q_0" the string is going to be '1' in decimal & when we divide it by '3' we get remainder as '1' so, we need to move to state "q_1"

Now let us say that, we are in state 'q_1' & we have reached 'q_1" after seeing a "1". Now at "q_1" if we see 0 then string become (10) which is '2' in ~~binary~~ decimal & when 2 is divided by '3' remainder is '2' which is state "q_2"

& similarly we can do for "other".

Now, let us draw a "transition table" for this automata :⇒

	"0"	"1"	← "inputs"
→ q_0	"q_0"	"q_1"	
q_1	"q_2"	"q_0"	
q_2	"q_1"	"q_2"	

Note:⇒ initial state is represented by "→"
final state is represented by "*"

"Hats"

Question can also be like when divided by [3] remainder is [1], then in that case, the final-state is "q_1". instead of "q_0"

P.T.O

Note :- If the question is all about divisible by (n) then no. of states in "DFA" is 'n' itself.

Q:- Construct a minimal "DFA" which accepts set of all strings over {0,1} which when interpreted as a binary number is divisible by 4.

Sol:- $\Sigma = \{0,1\}$ & $\omega \in (0,1)^*$

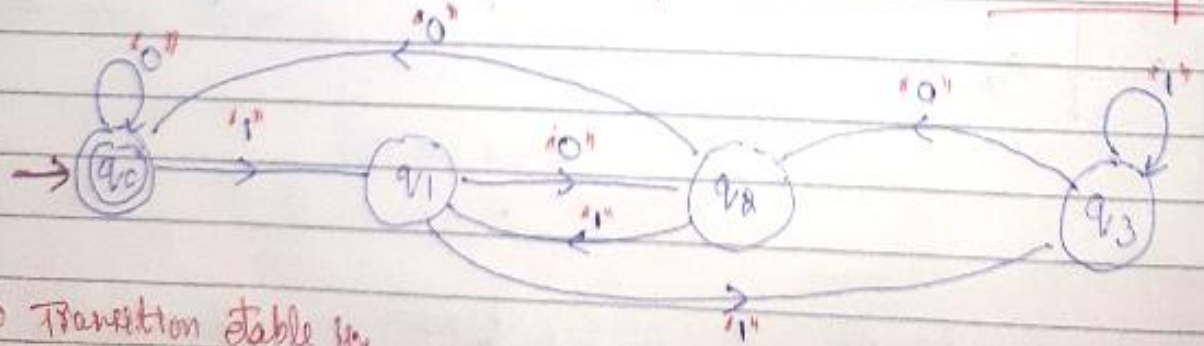
i.e a no. when divided by 4, can have four remainders 0, 1, 2 & 3

So, we can divide the strings into "four classes" on the basis of the remainder they will generate when divided by "4".

(q_0), (q_1) (q_2) & (q_3).

0	0	1 ← 10	10 → 2	11 → 3
4	100	5 ← 101	110 → 6	111 → 7
8	1000	9 ← 1001	1010 → 10	1011 → 11
⋮	⋮	⋮	⋮	⋮

Now Automata is like:-



Now Transition table is:-

	'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

shortcut :- simply write (q_0, q_1, q_2, q_3) & repeat it "

Ex- Construct a minimal "DFA" which accepts set of all strings over {0,1,2} which when interpreted as a ternary number is divisible by 5.

Sol:- Here $\Sigma = \{0,1,2\}$ & $[w \in (0,1,2)^*]$

No. of states is "4" = (q_0, q_1, q_2, & q_3) Here 'q_0' is the final state.

follow shortcut method :-

	'0'	'1'	'2'
→ q_0	q_0	q_1	q_2
q_1	q_3	q_0	q_1
q_2	q_2	q_3	q_0
q_3	q_1	q_2	q_3

"Transition diagram".

In general, if base 'k' no. is given {0, 1, 2, - - - k-1} & we are asked to divide it by 'm' complexity. then

No. of inputs = 'k-1' & No. of states = 'm'.

⇒ "DFA" of binary numbers divisible by 'n' & epsilon is not present in Automata :-

Note :- suppose we are dividing a string complexity by '3', then we know that no. of states will be 3, "q_0, q_1 & q_2"

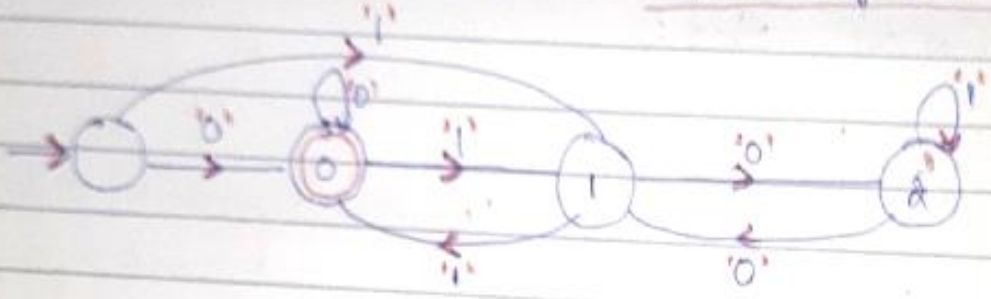
"q_0" → When remainder = "0"
"q_1" → When remainder = "1"
"q_2" → when remainder = "2"

Here "q_0" is both "initial & final state" for this 'automata'.

Note :- whenever "initial and final states" are "same", then 'ϵ' will also be there in the automata.

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

Now suppose it is explicitly mentioned that 'ϵ' should not be present in "Σ". Now Let's do the same question of divisible by [3].

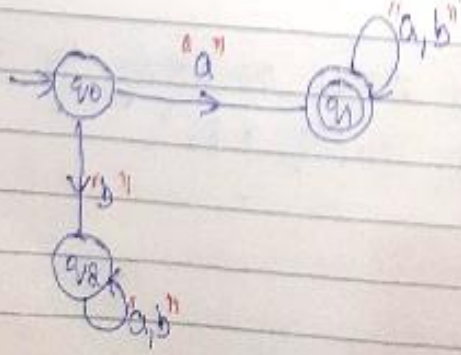


Now add one more state just to remove ['ϵ'] transition as shown above.

Q:- Construct a minimal "DFA" which accepts set of all strings over $\{a, b\}$ where each string starts with an "a" ?

Sol:- $\Sigma = \{a, b\}$ $\quad w \in (a, b)^*$

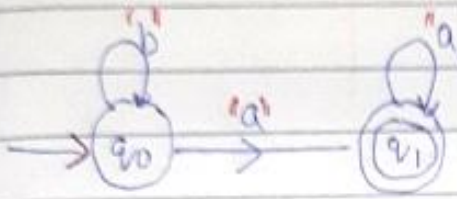
$L = \{a, aa, aaa, ab, abb, aabb \ldots\}$ 'infinite lang.'



Q:- Construct a minimal "DFA" which accepts set of all strings over $\{a,b\}$ where each string contains 'a'.

Sol:- $\Sigma = \{a,b\}$, $w \in (a,b)^*$

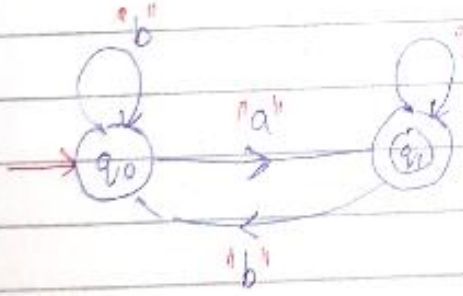
$L = \{a, aa, ab, ba, baa \ldots\}$ "Infinite language"



Q:- Construct a minimal "DFA" which accepts set of all strings over $\{a,b\}$ where each string ends with an 'a'.

Sol:- $\Sigma = \{a,b\}$ $w \in (a,b)^*$

$L = \{a, aa, baa, baba, bbba, babbaa \ldots\}$ Infinite.



If we are getting 'b' at state 'q_1', we would be waiting for an 'a' to appear.

Q:- Construct a minimal "DFA" which accepts set of all strings over $\{a,b\}$ where each string starts with 'a', contains 'a' & ends with 'a'.

Sol:- $\Sigma = \{a,b\}$ & $w \in (a,b)^*$

$L = \{a, aa, aba, abbaa \ldots\}$ infinite

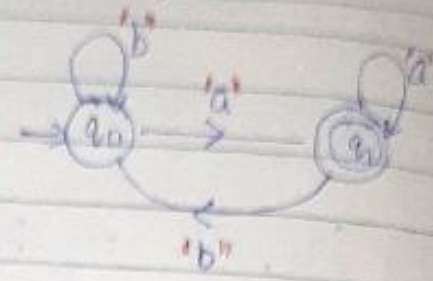
P.T.O

"Conversion"

(Starting with 'a')

(Containing 'b')

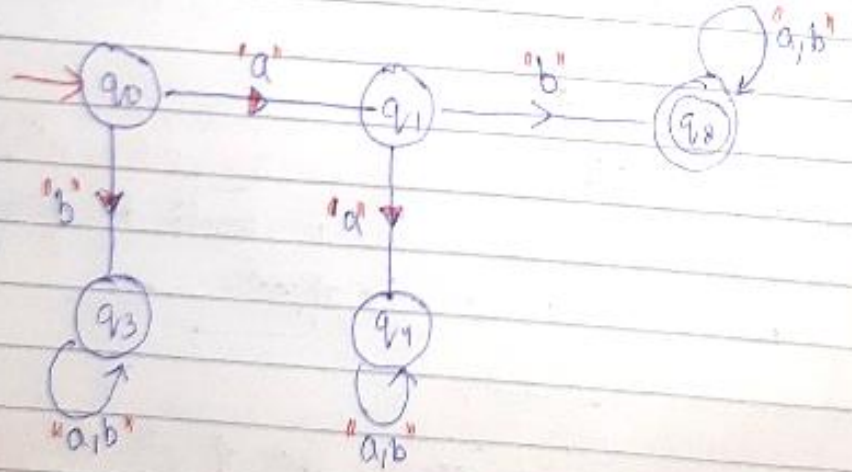
(Ending with 'a')



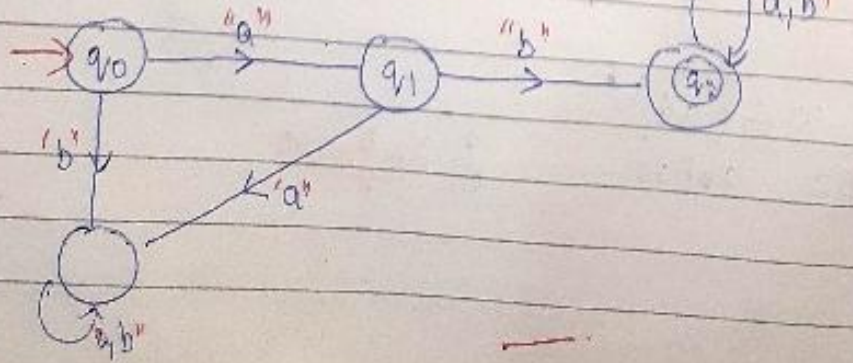
Q:→ Construct a DFA which accepts set of all strings over $\{a,b\}$, which starts with "ab".

Sol:→ $\Sigma = \{a,b\}$ $\omega \in (a,b)^*$

Now $L = \{ab, aba, abb, abab.....\}$ infinite.



"OR"



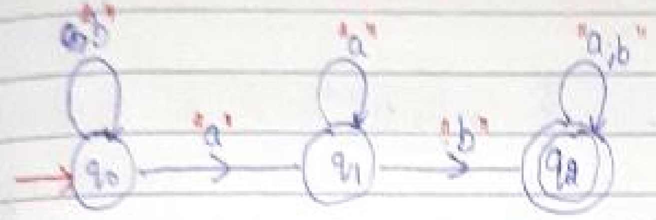
Eg:⇒ (Given a string "abc" then substring can be :⇒ ab, bc, abc, ∈ but not "ac")

Substring means, there should not be any 'gap'

Q:⇒ Construct a "DFA" which accepts set of all strings over {a,b} which contains 'ab'.

Sol:⇒ $\Sigma = \{a, b\}$ $w \in (a, b)^*$

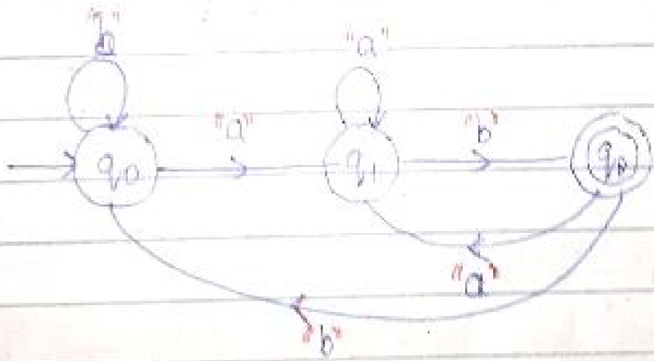
L = { ab, aab, bab, aba, ... babba } "Infinite language"



Q:⇒ Construct a "DFA" which accepts set of all strings over {a,b} which ends with "ab".

Sol:⇒ $\Sigma = \{a, b\}$ $w \in (a, b)^*$

L = { ab, abab, bab, aab, abbab ... } "Infinite-language"



⇒ DFA problem & Concatenation of DFA :⇒

Eg Q:⇒ Construct a "DFA" which accepts set of all strings over {a,b} which starts with 'a' & ends with 'b'.

let us try to concatenate two problems.

P.T.O

$\Sigma = \{a, b\}$ & $w \in (a,b)^*$

(1) Starting with "a"

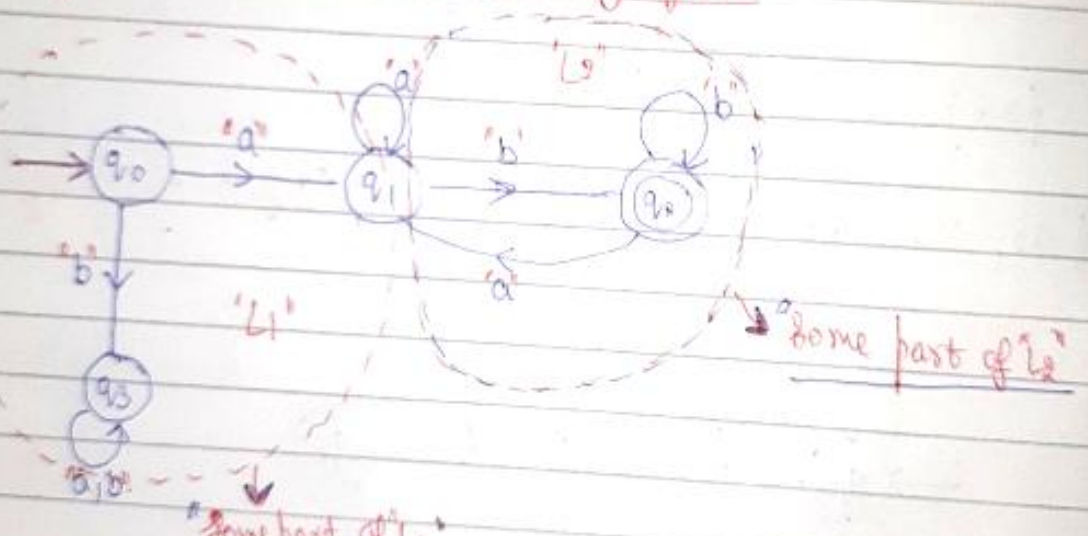
$L_1 = \{a, aa, ab, aabb, \ldots\}$ infinite

(2) Ending with "b"

$L_2 = \{b, ab, aabb, abbab \ldots\}$ "infinite"

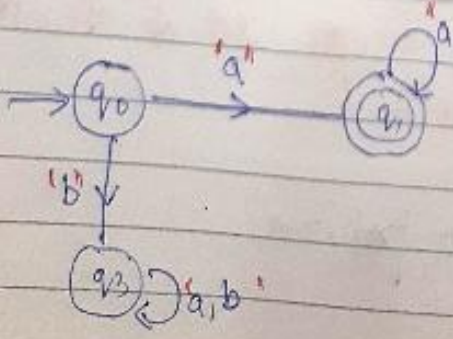
$L_1 \cdot L_2 = \{ab, aaab, abaabb, aabbabbab \ldots\}$ inf.

Let us try to construct "DFA" directly first :>

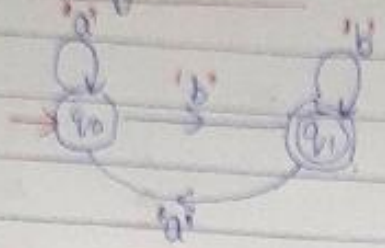


Let us now draw "DFA's" for both these languages "L_1" & "L_2" independently

DFA for "L_1" "Starting with a"



DFA for "L₂" "Ending with b"



So, here we have concatenated two "automata's".

Q:→ Construct a "DFA" which accepts set of all strings over {a,b} which starts and ends with different symbols.

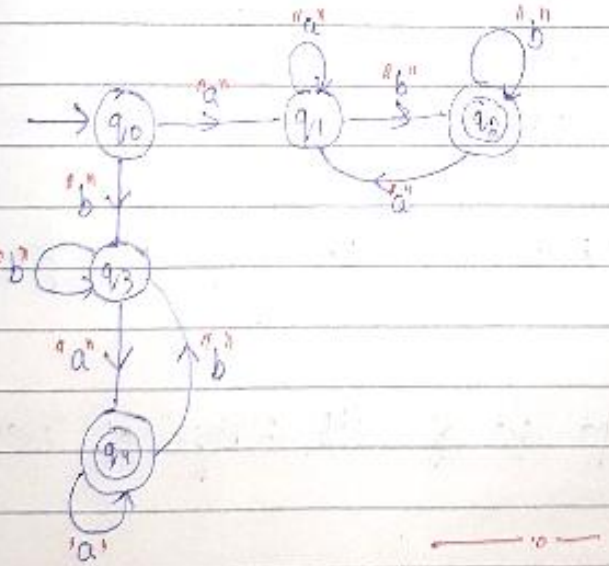
Sol:→ $\Sigma = \{a, b\}$ Starts & ends with different symbols

"a" - - "b"

"b" - - - "a"

"Start symbol" "End symbol"

$L = \{ab, ba, abb, \cdots\}$ "infinite"



Q:→ Construct a "DFA" which accepts set of all strings over {a,b} with starts & ends with the same symbol.

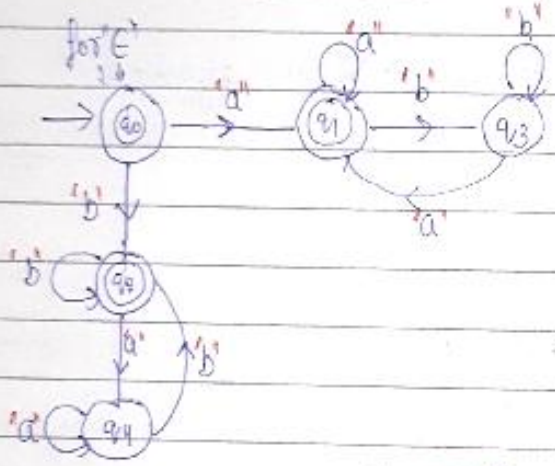
P.T.O

Ex:- $\Sigma = \{a, b\}$ & $G(a,b)^*$

Strings of ends with same symbol :- a - - - a
b - - - b
↑ 'start' ↑ 'end'

$L = \{\epsilon, a, b, aba, bab, abba, aa, bb \text{ ---- }\}$ "Infinite lang".

Note :- Whenever we have to accept "ϵ", we are going to make "initial-state" same as "final-state".

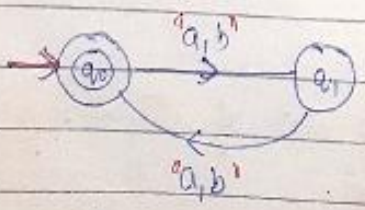


"COMPLEMENTATION OF DFA" :-

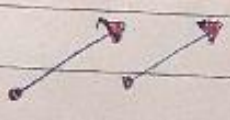
Note :- Complementing a "DEFINITE-AUTOMATA" works only for "DFA" & not for "NFA's".

Eg:- Let us say we want to accept "set of all strings" of "even length" over $\Sigma = \{a, b\}$ 'a' is even

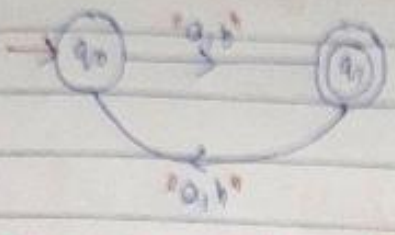
'L'



"EVEN-LENGTH-STRINGS"



for "odd-length strings"



"ODD-LENGTH-STRINGS"

So, here we can analyse, that both these "automatas" are same except for the "states" "final states" are converted to "non-final states" & "vice versa".

$$"L1" = \overline{"L2"}$$

"Second-Example" :⇒

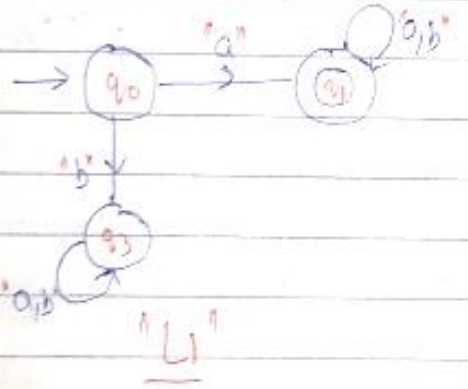
$L = \{\epsilon, b, ba \text{ ---} \}$

↑

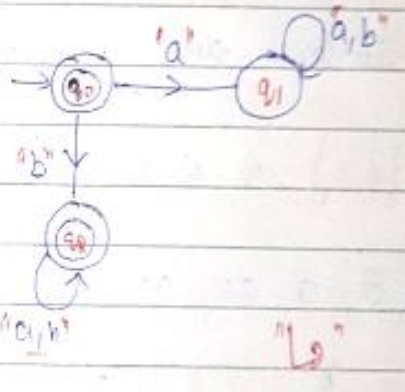
not starting with 'a'

or

"Starting with 'a' over $\Sigma = \{a, b\}$"



"Starting with 'b" over $\Sigma = \{a, b\}$"



$\therefore L1 = \overline{L2}$

So, In order to find the Complement of a language "L", Construct "DFA" for "L" & then change final state to non-final states & vice-versa.

Complementation :⇒

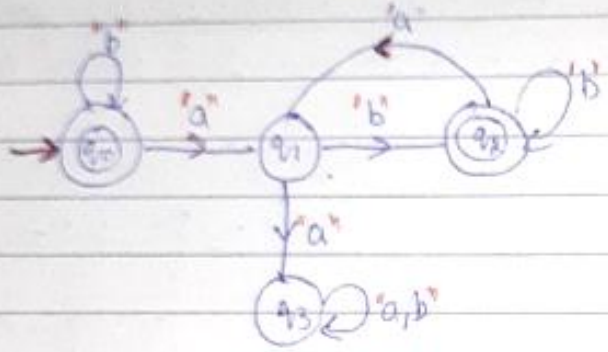
1> It is applied only for DFA's & not for NFA's.

2> $(Q, \Sigma, \delta, q_0, F) \xrightarrow{\text{"Complement"}} (Q, \Sigma, \delta, q_0, (Q - F))$.

Q:> Construct a minimal DFA, which accepts set of all strings over {a,b} in which every 'a' is followed by a 'b'.

Sol:> $\Sigma = \{a,b\}$ & $w \in (a,b)^*$

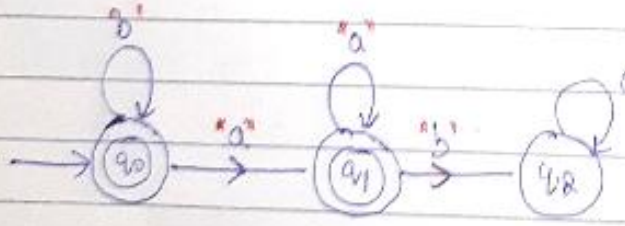
$L = \{ab, abb, abbab, bb, bb \ldots\}$ "Infinite"



Q:> Construct a minimal DFA, which accepts set of all strings over {a,b} in which every 'a' should never followed by a 'b'.

Sol:> $\Sigma = \{a,b\}$ & $w \in (a,b)^*$ [Here it is not a "complement" because there are some "intersections".]

$L = \{\in, a, aa, aaa, \ldots b, bb, \ldots babba\}$

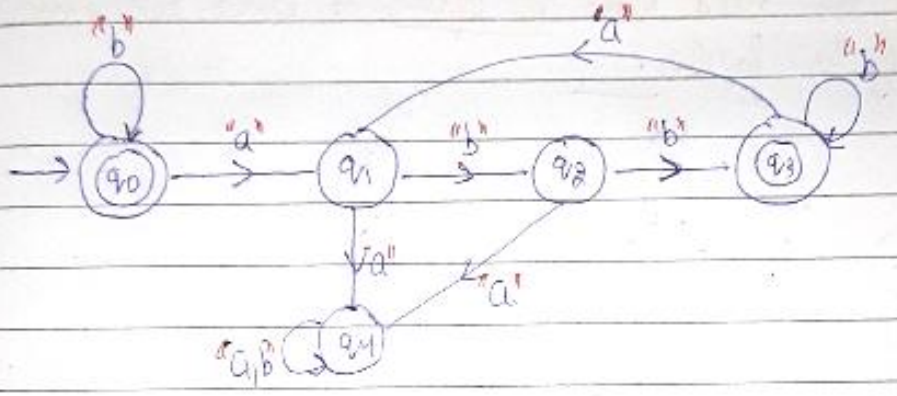


q_0 acts like, it accepts all the strings which doesn't contain 'a'. q_1 is going to remember that it has accepted 1 'a' or any no. b's followed by one 'a' followed by any no. of a's

Q83:> Construct a minimal DFA which accepts set of all strings over {a,b} in which every 'a' should be followed by a 'bb'.

Sol:> $\Sigma = \{a, b\}$ & $w \in (a,b)^*$

$L = \{ \in, abb, abbabb, bb, bbb - \}$ Infinite



a – bb ✓ Accepted

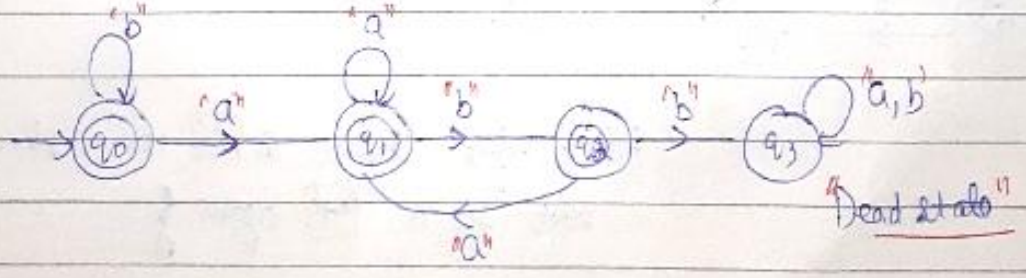
a – aa	×
a – ab	×
a – ba	×

Rejected

Q84:> Construct a minimal DFA which accepts set of all strings over {a,b} in which every 'a' should never be followed by a "bb".

Sol:> $\Sigma = \{a, b\}$ & $w \in (a,b)^*$

$L = \{ \in, a, aa, aba, b, bb, bbb \dots bba - \}$ "Infinite language"



Q85:> Construct a minimal DFA which accepts $L = \{ a^n b^m / n, m >= 1 \}$

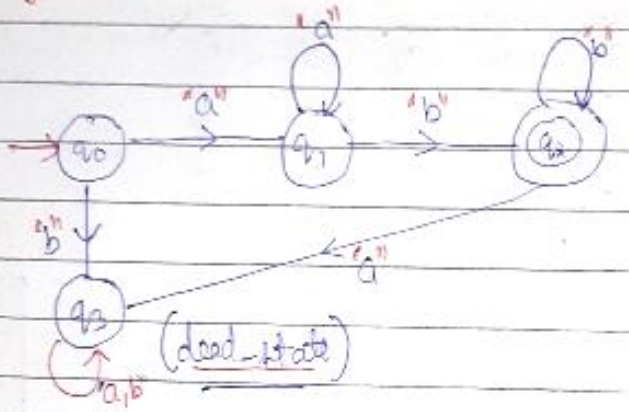
P.T.O

Sol:> $\Sigma = \{a, b\}$ $L = \{a^n b^m / n, m \geq 1\}$

$L = \{ab, aab, abb, aabb, aaabb - - - -\}$

only restriction is we should see 'a's then 'b's & then no 'a's.

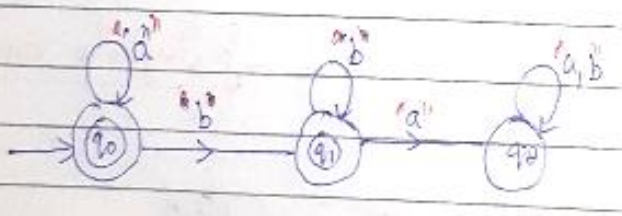
& No. of 'a's need not be equal. & once we saw any 'b' will not be seeing any 'a' further.



Q:> $\Sigma = \{a, b\}$ $L = \{a^n b^m / n, m \geq 0\}$

$L = \{\in, a, aa, aaa - ab, - , b, bb, bbb - - - -\}$ 'Infinite Lang'

Sol:>

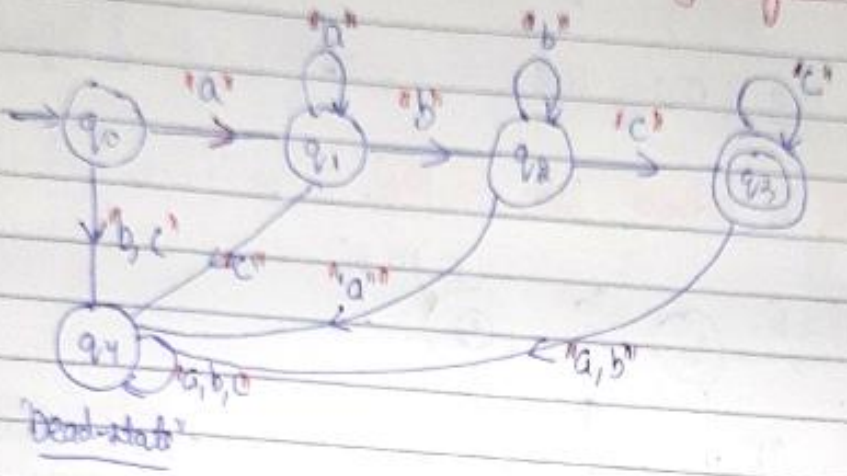


& Here after seeing a 'b' we should not see a 'a' again &

Q:> Construct a minimal "DFA" which accepts $L = \{a^n b^m c^l \mid n, m, l \geq 1\}$ &

Sol:> $L = \{a^n b^m c^l \mid n,m,l >= 1\}$ $w \in \{a,b,c\}^*$

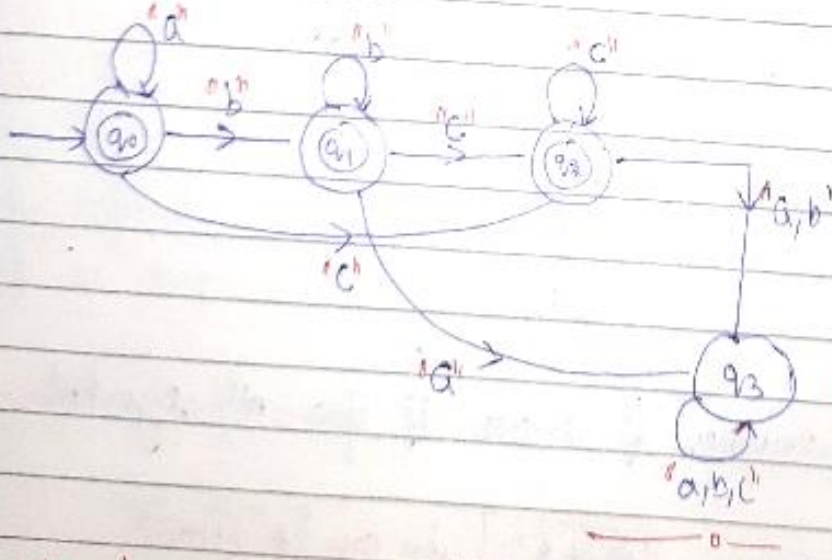
$L = \{abc, aabc, aabbc, aabbcc, \ldots\}$ Infinite.



Q:> Construct a minimal "DFA" which accepts $L = \{a^n b^m c^l \mid n,m,l >= 0\}$

Sol:> $L = \{a^n b^m c^l \mid n,m,l >= 0\}$ $w \in \{a,b,c\}^*$

$L = \{\in, a, b, c, aa, bb, cc, abc, \ldots\}$ "Infinite-Language"



Q:> Construct a minimal "DFA" which accepts set of all strings over $\{a,b\}$ such that second symbol from L.H.S is "a"

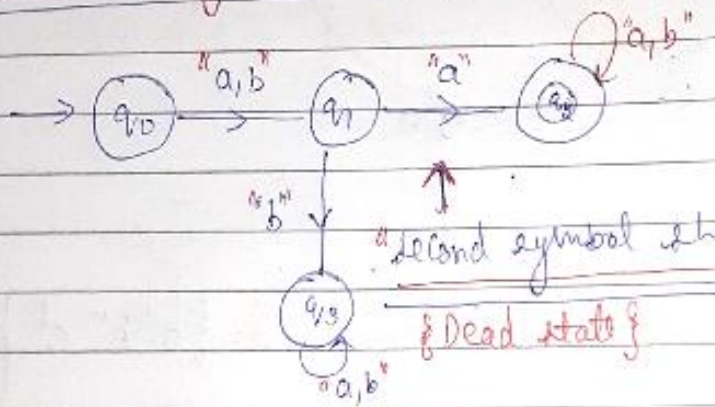
PTO

Sol:> $\Sigma = \{a, b\}$ $\quad w \in \{a,b\}^*$

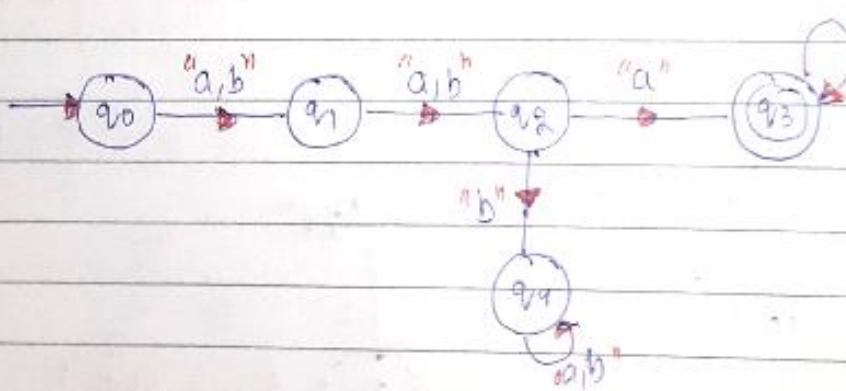
2nd symbol in the string "w" from L.H.S should be "a"

$L = \{aaa, aa, ba, a, a, \ldots ba \ldots\}$ {"Infinite"}

Smallest string is either "aa" or "ba".



Now let us construct "DFA" for a lang. where third symbol from L.H.S is "a".



'Note':> What is the number of states if the nth symbol from L.H.S is 'a',

then it is $\boxed{n+2}$ in case of "L.H.S"

Q:> Construct a "DFA" which accepts set of all strings over $\{a,b\}$ where strings are of the form $a^3 b w a^3$, where 'w' is any string

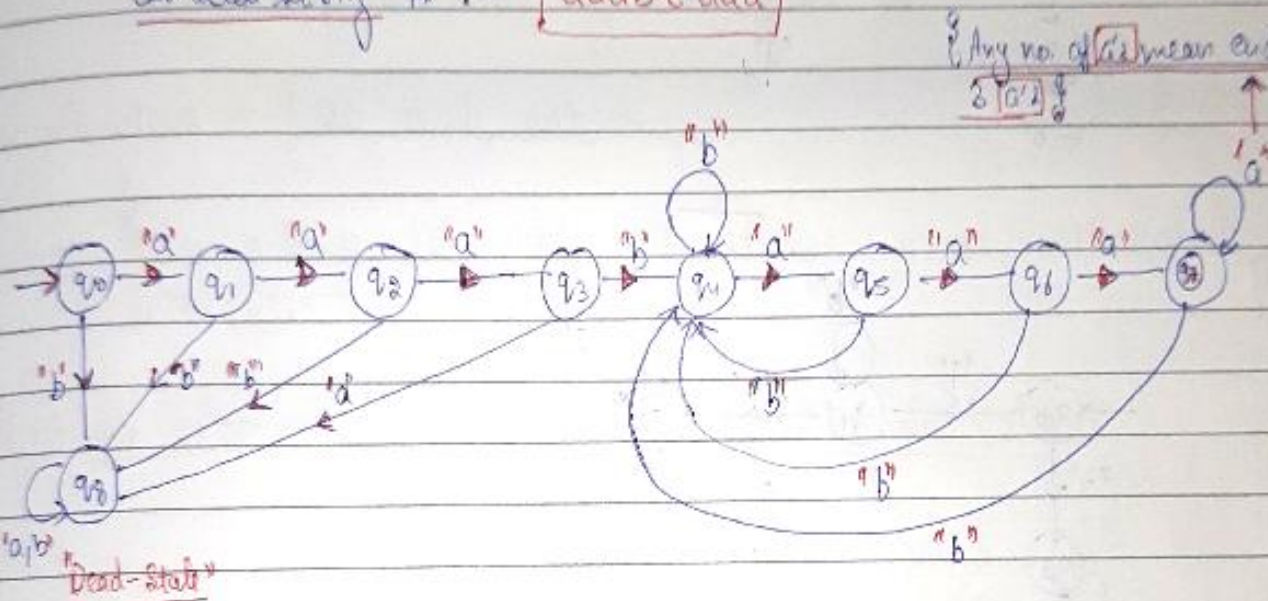
$\lambda \rightarrow$ 'ε' is always part

over $\{a, b\}$.

sol:- $L = \{a^3 b\, w\, a^3\}$ the 'w' can be anything i.e $\boxed{w \in \{a,b\}^*}$.

$L = \{a^3 b \in a^3,\ a^3 b a a^3,\ a^3 b b a^3,\ a^3 b a b a^3 \ldots\}$ Infinite long

"Smallest string" is :- $\boxed{aaab \in aaa}$



⇒ OPERATIONS on "DFA" like "Union", "Concatenation", "cross-product", "Complementation", & "Reversal" :⇒

We can take two "DFA's" & perform "Union" or "Concatenation" or "cross-product" or "Complement" or "reversal" :⇒

▷ "Union" – Eg:- (strings starting and ending with different symbols)

$\Sigma = \{a, b\}$ $L_1 = \{a\cdot b,\ aab,\ abb \ldots\}$ } Both are infinite.

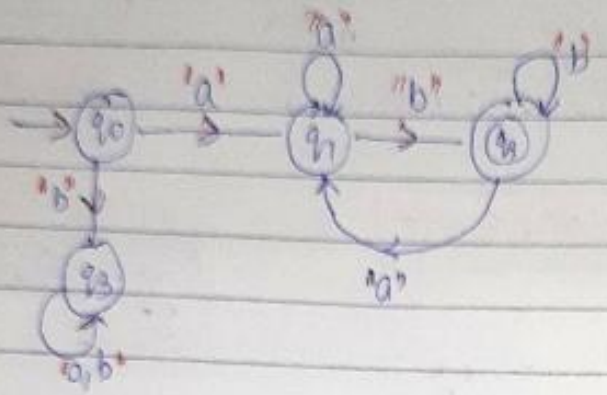
$L_2 = \{ba,\ baa,\ bba \ldots\}$ }

So, our required language which "starts" & "ends" with different symbols consists of two parts

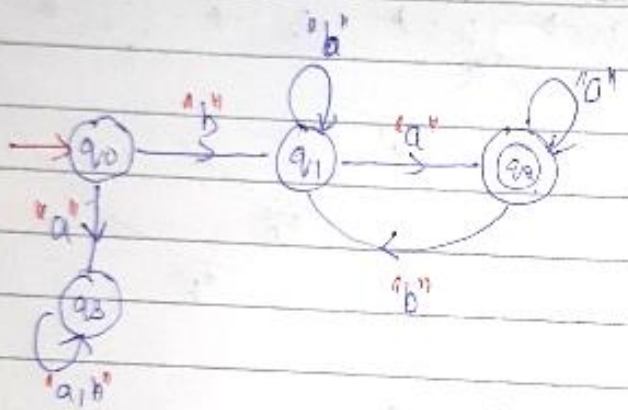
Part 1 :- { starts with 'a' & ends with 'b' } 'L1'

Part 2 :- { starts with 'b' & ends with 'a' } 'L2'

"L1"

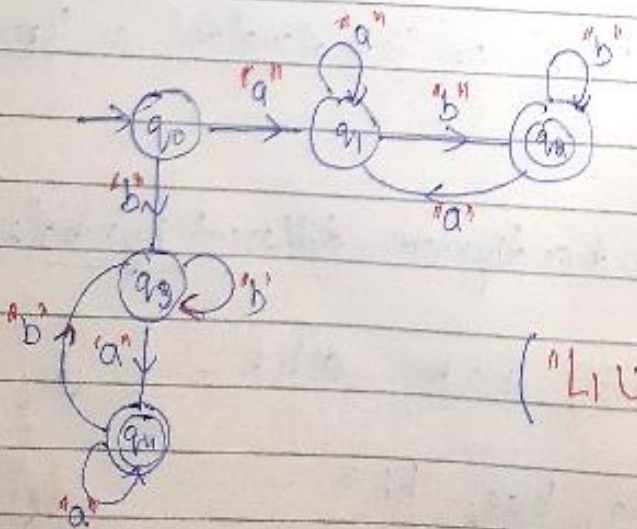


"L2"



Now, $L = "L_1 \cup L_2"$

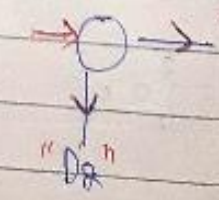
So, we need to merge these two automata.



L_1 — D_1

L_2 — D_2

$"L_1 \cup L_2"$



$("L_1 \cup L_2")$

"Concatenation" :→

L_1 = { set of all strings starting with ['a'] } { a, aa, ab, aaa - - - }
L_2 = { set of all strings ending with ['b'] } { b, ab, bb, bbb - - - }

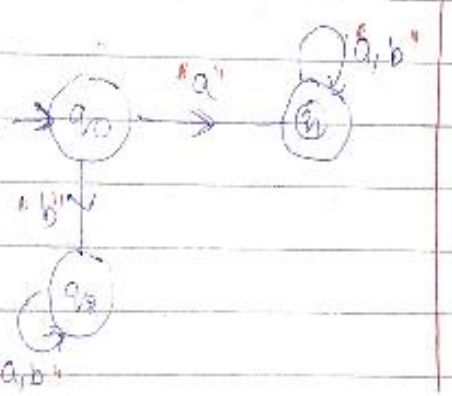
"Concatenation" of two languages mean we can take any string from "L_1" & append it with any string from "L_2".

∴ $L_1.L_2$ = { ab, aab, aab, - - - - - - - - - - - - - - - - - }

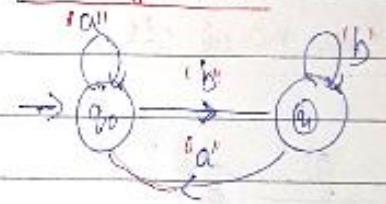
↓

It is a lang. in which all strings start with ['a'] & ends with ['b'].

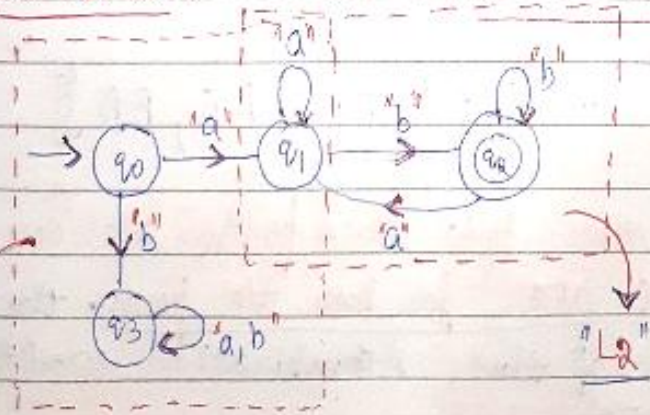
"DFA for 'L_1'" :→



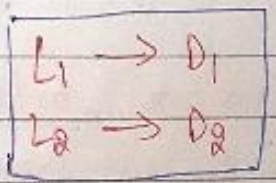
DFA for "L_2" :→



DFA for '$L_1.L_2$' :→ starts with 'a' & ends with 'b' :→



$L_1.L_2$ = "L"



"L_2" then "$L_1.L_2$" ≡ "$D_1.D_2$"

The final state of first "DFA" is going to be initial state of 2nd "DFA".

Note:> The "final state" of first "DFA" & "initial state" of 2nd "DFA" should be merged & transitions should be managed accordingly.

3> "Cross-Product" :> (whenever question is like "&&", "||" !) etc

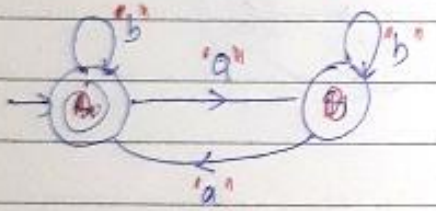
"L_1" = { Even no. of a's } = { $\in$, aa, baa, aba, ---- }

"L_2" = { Even no. of b's } = { $\in$, bb, bba, bab, ---- }

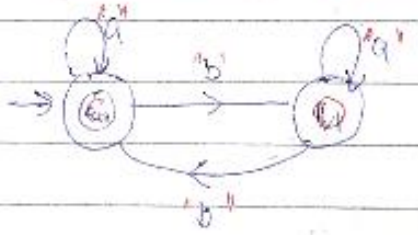
L = { strings which contains even no. of [a's] as well as even no. of [b's] }

Now we can construct DFA's (individual DFA's) for both these languages & can combine them into a single "DFA".

L_1 = { Even no. of a's }



L_2 = { Even no. of b's }



We have already seen the method of "combining these two DFA's" by taking "cross-product".

{ A, B } x { C, D } = { (A C), AD, BC, BD }

Whenever conditions are given b/c two parts, we can draw "DFA" by taking cross-product of DFA's for both the parts. then what ever be the final states in each of them, A combination would become the final state for the "final DFA".

4) "Complementation" :> Eg:- [Does not contains 'a']

Eg:> let "L_1" is a language which contains all strings containing 'a'

$L_1 = \{ a, aa, ba, aaa, bab, \dots \}$

Then

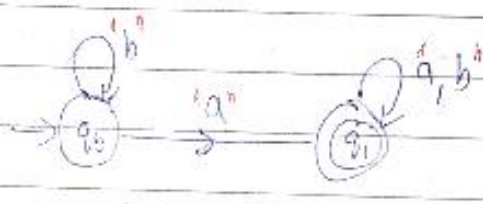
$\overline{L_1} = \{ \in, b, bb, bbb \dots \}$

↳ {set of all strings which does not contain 'a'}

let us say already we have finished drawing a "DFA" for one language 'L_1'

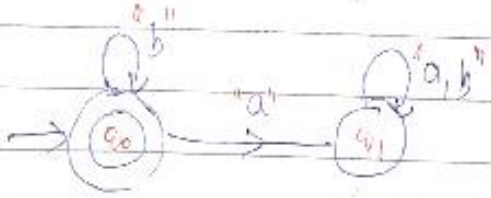
Now if the question is, it does not contain 'a', then find the language in which string contains 'a' & then take complement of 'a'

Now DFA for L_1 =



Now in order to find $\overline{L_1}$ make all "final states" as "non-final states" & "all non-final" states as "final states"

$\overline{L_1}$ =



"L_1" = {Q, Σ, δ, F, q_0} then "$\overline{L_1}$" = {Q, Σ, δ, q_0, "Q − F"}

5) "Reversal" :> Eg:> a w b → b w a

"L_1" → "Reverse of 'L_1'"

It is nothing but, if we have a language & if we have to find the reverse of this language, then take each string of the language and

simply reverse it.

Eg:- Let 'L' be a language in which all strings starts with 'a'.

L_1 = { strings with 'a' } = { a, aa, ab, aaa, aab ... }

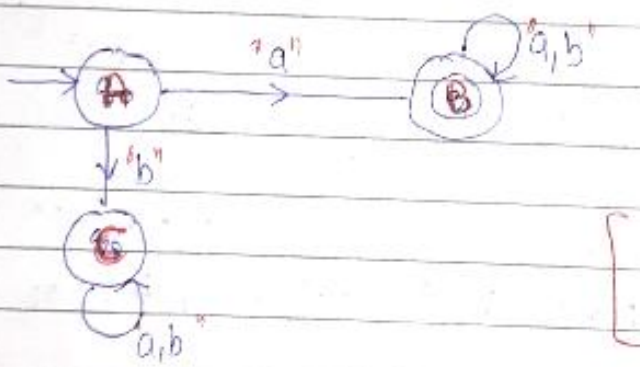
Now L_1^R = { a, aa, ba, aaa, baa ... }

ie, if initial language 'L_1' is starting with "a", then reversal of language 'L_1^R' will be ending with "a"

Reversal of a lang. can be obtained by:-

First draw the "DFA" for the original language.

"L_1" =



[Resultant can be NFA or DFA]

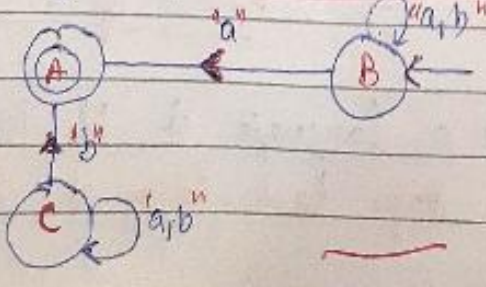
After this follow the steps given below :->

1> Draw the states as it is.

2> Make final state as initial state & initial state as final state.

3> If there are any "edges" simply "reverse them".

"L_1^R" =



[But it is not DFA but actually NFA]

→ Problems of DFA {Deterministic Finite Automata}

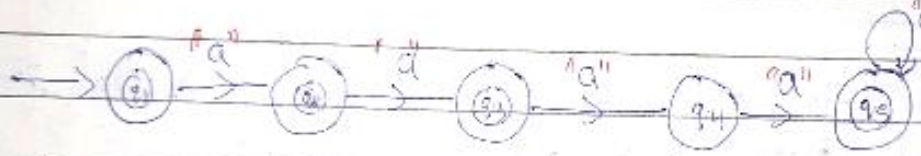
Q:→ Construct a minimal "DFA" over {a}

1> For $\{a^n \mid n>=0,\ n!=3\}$
2> For $\{a^n \mid n>=0,\ n!=2,\ n!=4\}$

{ There are some cases where we have to accept everything & reject some few cases }

Sol:→ 1> $L = \{a^n \mid n \geq 0,\ n \neq 3\}$.

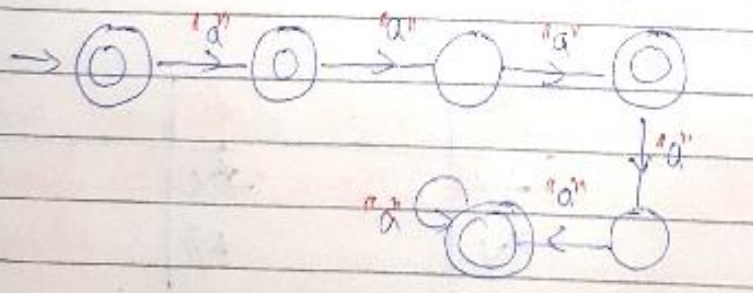
$L = \{\in, a, aa, aaaa, aaaaa \text{ ----}\}$ Here we have to exactly reject one string. 'aaa'



"$\in$, a, aa, & aaaa & so on are accepted" except [aaa]

2> $L = \{a^n \mid n \geq 0,\ n \neq 2 \text{ & } n \neq 4\}$.

"$L = \{\in, a, aaa, aaaaa, \text{ ---}\}$



Q:→ How many two state DFAs with a designated initial state & designated final state can be constructed over the alphabet $\Sigma = \{a, b\}$?

PTO

Sol:- Given that DFA is going to have only "two states".

	a	b
→ X	X/Y	X/Y
Y*	Y/Y	Y/Y

So, there are "2 × 2 × 2 × 2" possibilities

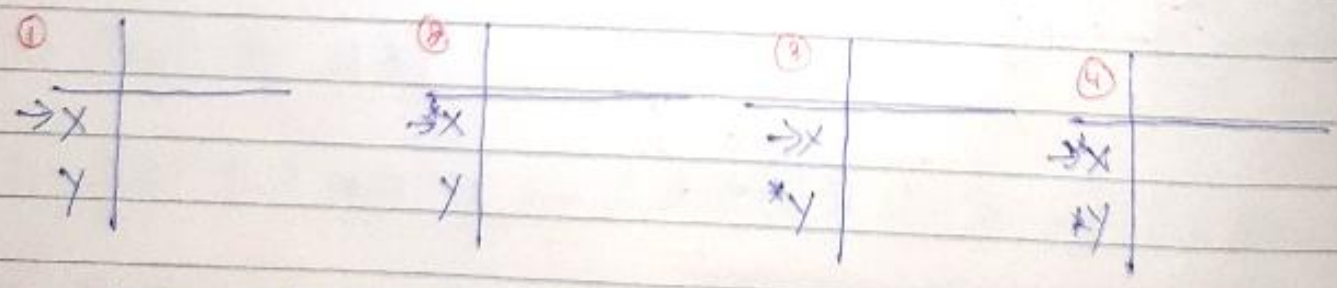
Note :- Once we designate 'X' as "initial state", we cannot change it to "final state" & once we designate 'Y' as "final state", it cannot be changed to "initial state".

Now suppose the question is :-

How many two state DFA's with a designated initial state can be constructed over the alphabet $\Sigma = \{a, b\}$.?

Sol:- Possibilities are :-

1) "There is no final state in DFA".
2) "X itself is the final state."
3) "Y is final state in DFA"
4) "X & Y both are final states"



Now each of these cases will give rise to 16 DFA's

∴ Total DFA's possible are :- $2^4 * 4$

$= 2^4 * 2^2 = 2^6 =$ {64 DFA's}

Q:> How many two 'state DFA's' can be constructed with a designated initial state that accept the empty language over the alphabet $\Sigma = \{a, b\}$?

Sol:> Given, only "2 states" are present in the "DFA"

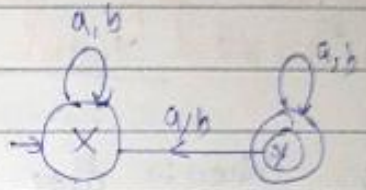
Empty language is $\{\}$ or ϕ

"ϵ" is not an empty language but infact it is an "empty-string"

	a	b
→X	X/Y	X/Y
Y	X/Y	X/Y

16 DFA's

	a	b
→X	X	X
*Y	X/Y	X/Y



Here "X" cannot be made as final state, because if done so, then "ϵ" would also be accepted by the DFA & we don't want that.

1> lang. accepted by DFA will always be empty. "16" DFA's are possible with 1st layout.

2> for second layout, only "4 DFA's" are possible

Q:> How many two states DFA can be constructed with a designated initial state that accepts the universal language over the alphabet $\Sigma = \{a, b\}$?

Sol:> Given only "2 states" are present in the "DFA" & "$\Sigma = \{a, b\}$"

"L" = $\{\epsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$

PTO

	'a'	'b'
→x	x/y	x/y
*y	xy	xy

"16 DFA's are possible"

	'a'	'b'
→x	x	x
y	x/y	x/y

"4 DFA's are possible"

∴ "Total DFA's possible" = "16 + 4 = 20"

Q:→ How many three state DFA's can be constructed with a "designated initial state" over the alphabet $\Sigma = \{a, b\}$?

Sol:→ Given:→ there are "three states" in "DFA".

& one of the state is designated as "initial-state".

	a	b
→x		
y		
z		

$\{$"x yz"$\}$: $\{$0 final states$\}$ ①

(x) y z
x (y) z
x y (z)

$\left.\begin{array}{l}\text{x can be final}\\ \text{or}\\ \text{y can be final}\\ \text{or}\\ \text{z can be final}\end{array}\right\}$ ③

(x) (y) z
(x) y (z)
x (y) (z)

$\left.\begin{array}{l}\text{x \& y can be final}\\ \text{x \& z can be final}\\ \text{y \& z can be final}\end{array}\right\}$ ③

or All of them can be final states

(x) (y) (z) $\}$ All are final ①

	a	b
←		
→X	X/Y	X/Y
Y	X/Y	X/Y
Z	X/Y	X/Y

$3^3 = 27$ possibilities

Total possibilities are $3+3+1+1 = 8 * 3^6$ total DFA's

Q:> How many ["n"] state DFA's can be constructed with a designated initial state over the alphabet [Σ] containing ["m"] symbols?

Sol:> No. of States = 'n' & No of alphabet/symbols = "m"

There is one "designated initial state".

	1	2	3	4	- - -	n
→1						
2						
3						
4						
⋮						
n						

size of table is "m * n"

& No. of ways in which we can fill these table entries is :>

Each entry in the table is having 'n' ways of getting filled.

$= n \times n \times n \times \cdots$ (m*n) times

$= \boxed{n^{(m*n)}}$

if we don't choose anyone among them as "final state" we can get "one structure" $= {}^nC_0$ [Not choosing anyone of these states as the final state.]

$\{ {}^nC_0 + {}^nC_1 + {}^nC_2 + {}^nC_3 + \cdots {}^nC_n \}$

↓ "One fin ↓ two fin ↓ 3 final ↓ all are final"

$\boxed{\left({}^nC_0 + {}^nC_1 + {}^nC_2 + {}^nC_3 + \cdots {}^nC_n \right) n^{(m*n)}} = \boxed{2^n \times n^{m \times n}}$

$\{ (1+1)^n = {}^nC_0 1^0 * 1^n + {}^nC_1 1^1 1^{n-1} + \cdots \}$
