UNIT-I

Introduction to finite Automata

→ Finite Automata are useful Model for many important kinds of hardware and software.

→ some of the important uses of finite Automata are.

(i) Software for designing and checking the behaviour of digital circuits.

(ii) The "lexical Analyzer" of a typical compiler, i.e, the compiler Component that breaks the input text into logical units, such as identifiers, keywords, and punctuation.

(iii) Software for Scanning large bodies of text, such as collection of web pages, to find occurences of words, phrases, or other patterns

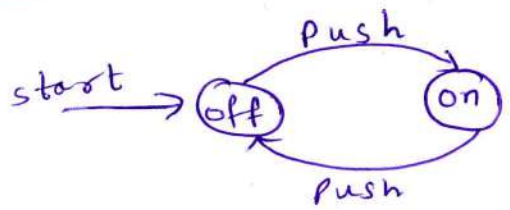
(iv) software for verifying systems of all types that have a finite number of distinct states, such as communication protocols or protocols for secure exchange of information.

→ There are many systems that can be viewed as being at all times in one of a finite number of "states".

→ The purpose of state is to remember the relevant portion of the system's History.

→ Since there are only a finite no of states, the entire history cannot be remembered, so that system must be designed carefully to remember what is important and forget what is not.

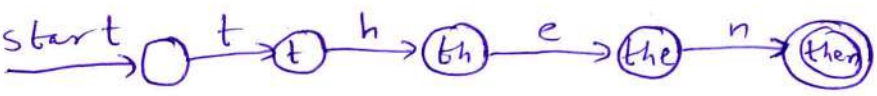
Eg: Nontrivial Finite Automata is an "on/off switch"



States - "on" & "off"
Arcs b/w states - "inputs"

→ It is often necessary to indicate one (or) more states as "final" (or) "accepting" states.

Eg: Finite automaton that could be part of lexical Analyzer



Structural Representation

→ These are two important notations that are not automaton-like, but play important role in the study of automata and their applications.

Grammars are useful models when designing software that processes data with a recursive structure. Eg: parser

Regular Expressions also denote the structure of data, especially text strings.

Eg: [A-Z] [a-z]* [ ] [A-Z] [A-Z]

Automata and Complexity

Automata are essential for the study of the limits of computation.

→ There are two important issues:

(i) what can a computer do at all? This study is called "decidability" and the problems that can be solved by computer are called "decidable"

(ii) What can a computer do efficiently?

This study is called "intractability", and the problems that can be solved by a computer using no more time than some slowly growing function of the size of the input are called "tractable".

The Central Concepts of Automata Theory

Alphabets

→ An "Alphabet" is finite, nonempty set of symbols.

→ we use the symbol Σ for an alphabet.

Eg: $\Sigma = \{0,1\}$, binary alphabet

$\Sigma = \{a, b, \ldots z\}$, the set of all lower-case letters

Strings

→ A string (or sometimes word) is a finite sequence of symbols chosen from some alphabet.

Eg: 01101 is a string from the binary alphabet $\Sigma = \{0,1\}$

Empty String: The Empty string is the string with zero occurrences of symbols.

→ This string is denoted by ∈

Length of string

→ It is often useful to classify strings by their "length", i.e, the no. of positions for symbols in the string.

Eg:- 01101 has length '5'.

→ The length of string is "no. of symbols" in the string

→ The standard notation for the length of a string w is $|w|$.

Eg:- $|011| = 3$ and $|\epsilon| = 0$

Powers of an Alphabet

→ If Σ is an Alphabet, we can express the set of all strings of a certain length from that alphabet by using an exponential notation.

→ "Σ^k" is the set of strings of length k, each of whose symbols is in Σ.

Eg:- $\Sigma^0 = \{\epsilon\}$, ϵ is only string ~~with~~ whose length is 0.

→ If $\Sigma = \{0, 1\}$, then $\Sigma^1 = \{0, 1\}$

$\Sigma^2 = \{00, 01, 10, 11\}$, $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

→ The set of all strings over the alphabet Σ is conventionally denoted Σ^*.

$\{0,1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000 \ldots\}$

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

→ The set of nonempty strings from alphabet Σ is denoted Σ^+.

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

Concatenation of strings

→ let x and y be strings

→ xy denotes the Concatenation of x and y.

Eg:- $x = 1101$ and $y = 0011$

$xy = 11010011$ and $yx = 00111101$

→ For any string w, the equations $\epsilon w = w\epsilon = w$ hold.

Languages

→ A set of strings all of which are chosen from some Σ^*, where Σ is a particular alphabet, is called Language.

→ If Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a Language over Σ.

Note that Language over Σ need not include strings with all the symbols of Σ

Eg:- English, collection of English legal words is set of strings over the alphabet that consists of all the letters.

Eg:- C, whose legal programs are subset of the possible strings that can be formed from the alphabet of the Language. Alphabet is subset of ASCII characters.

Examples of strings

→ The Language of all strings consisting of n 0's followed by n 1's for some $n \geq 0$

$L = \{ \epsilon, 01, 0011, 000111, \text{---} \}$

→ The set of strings of 0's and 1's with an equal number of each:

$L = \{\in, 01, 10, 0011, 0101, 1001, \ldots\}$

→ The set of binary numbers whose value is a prime.

$L = \{10, 11, 101, 111, 1011, \ldots\}$

→ Σ^* is a language for any alphabet Σ

→ ϕ, the empty Language, is a language over any alphabet.

→ $\{\in\}$, the language consisting of only the empty string, is also a language over any alphabet.

→ $\phi \neq \{\in\}$ ϕ - No strings, $\{\in\}$ - has one string

→ Language formed from alphabet can be either finite or Infinite.

Problems

→ In Automata Theory, a problem is the question of deciding whether a given string is a member of some particular Language.

→ "problem" can be expressed as membership

in a Language.

→ if Σ is an alphabet, and L is a Language over Σ, then the problem L is:

- Given a string ω in Σ^*, decide whether or not ω is in L.

Eg:- In 'C' we use these symbols → Finite set

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

[program in C] = [string in To C]

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

'C' programming Language = set of all "Valid" programs

[Infinite] $= \{P_1, P_2 \ldots\}$

L is finite

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

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

↑ $s = aaa$

check if s is in 'L_1' then comparision Linearly easy

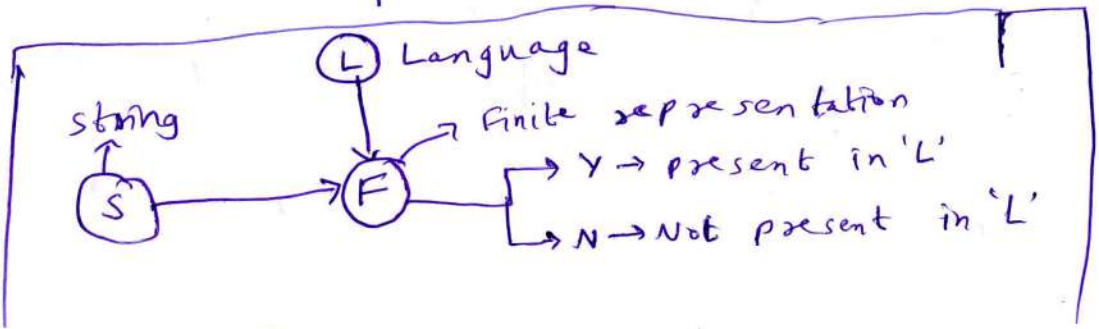
L is infinite

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

$L_2 = \{a, aa, aaa, ab, \ldots\}$

↑ $s = baba$

check if s in in 'L_2' then comparision is not possible as L_2 is infinite



input $a_1 a_2 \cdots a_n$ is "accepted", and if not then it is "rejected"

Eg:- D.F.A that accepts all and only the strings of 0's and 1's that have the sequence "01" somewhere in the string.

$L = \{ w \mid w$ is of the form $x01y$ for some strings x and y consisting of 0's & 1's only $\}$

$= \{ x01y \mid x$ and y are strings of 0's and 1's $\}$

. Examples of strings in the language include 01, 11010 and 10011

Examples of strings not in the language include ϵ, 0 and 111000

Simpler Notations for DFA's

(i) A Transition diagram, which is a graph ~~such as the ones~~.

(ii) A Transition Table, which is a tabular

listing of the δ function, which by implication tells us the set of states and the input alphabet

Transition Diagram

→ DFA $A = (Q, \Sigma, \delta, q_0, F)$ is a graph

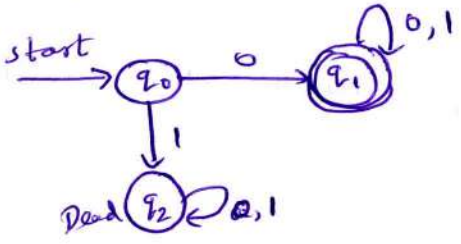
a) For each state in Q there is a node.

b) For each state q in Q and each input symbol a in Σ, let $\delta(q,a) = p$, then the transition diagram has an arc from "node q" to "node p", labeled a.

(c) There is an arrow into the start state q_0, labeled start. The arrow does not originate at any node.

(d) Nodes corresponding to accepting states (in F) are marked by a double circle. states not in F have a single circle.

Eg:- DFA accepting all strings starting with '0'



Transition Tables

→ A transition Table is a conventional, tabular representation of a function like 'δ' that takes "two arguments" and returns a "value".

→ The "rows" of the table correspond to the "states".

→ The "columns" of the table correspond to the "inputs."

→ The entry for the row corresponding to the "state 'q'" and the column representing the "input 'b'" is the state δ(q, b).

Eg:

	0	1
(start) → q_0	q_1	q_2 (D)
(final) * q_1	q_1	q_1
(Dead) * q_2	q_2	q_2

Transition Table

The Language of D-FA

→ Now, we can define the "language" of a D-FA $A = (Q, \Sigma, \delta, q_0, F)$.

→ The Language is denoted by $L(A)$

**

$$L(A) = \{ \omega \mid \hat{\delta}(q_0, \omega) \text{ is in } F \}$$

→ The Language of A is set of strings 'w' that take the start state q_0 to one of the accepting states.

→ If L is L(A) for some DFA A, then we say "L" is a "Regular Language"

** → In DFA, for every state, for every input there will be exactly one transition

&

** → In DFA, the transition function will be

$$\delta - Q \times \Sigma \rightarrow Q$$

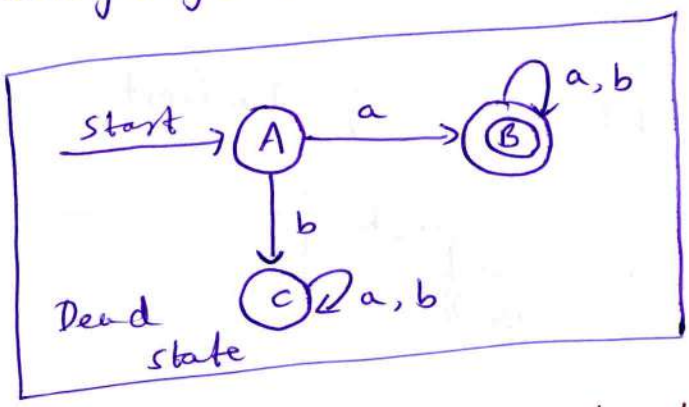
Examples of D.F.A

① Construct DFA that accepts set of all strings over {a,b} which "strats with 'a'"

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

Infinite

Language (L) = { a, ab, aa, aba, aaa, ------ }

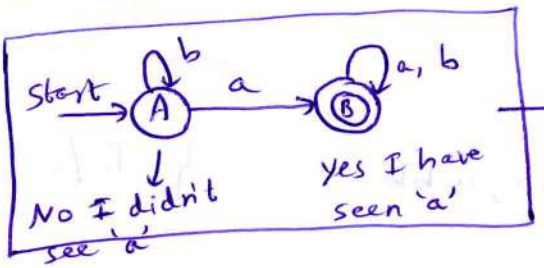


② Construct DFA that accepts set of all strings over {a,b} "containing 'a'"

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

Infinite

L = { a, ba, ab, aa, ------ }

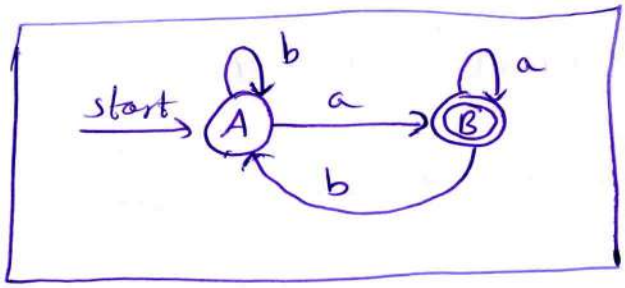


→ No Dead state

③ Construct D.F.A that accepts set of all strings over {a,b} "ending with a"

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

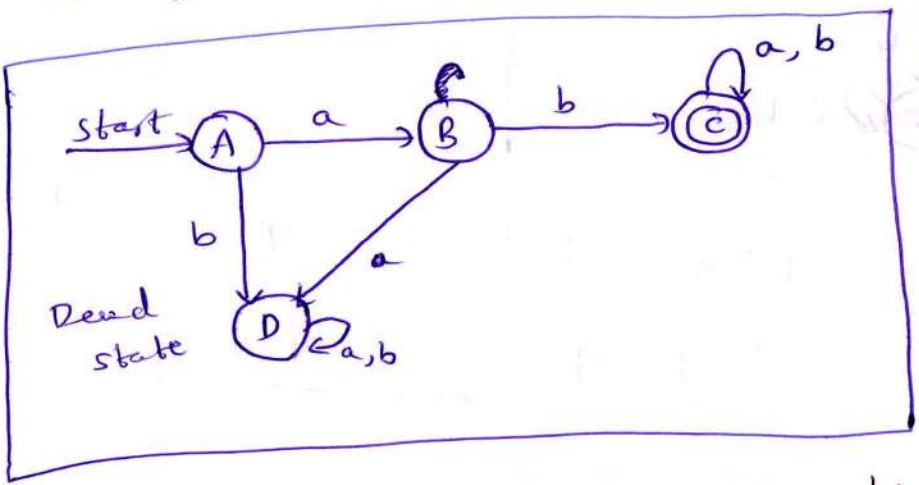
$L = \{a, ba, aa, aaa, aba, \ldots\}$ Infinite



(4) Construct DFA that accepts set of all strings over $\Sigma = \{a, b\}$ "starting with "ab""

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

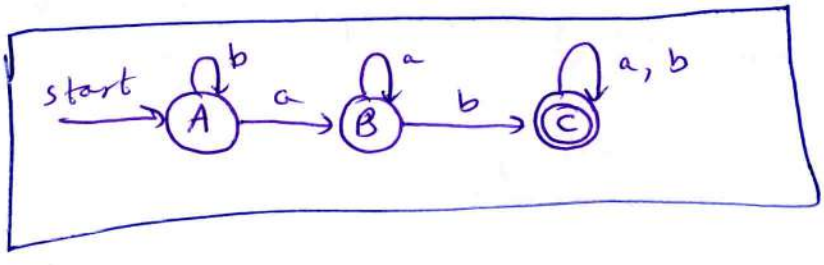
$L = \{ab, aba, abb, \ldots\}$ Infinite



(5) Construct DFA that accepts set of all strings over $\Sigma = \{a, b\}$ "containing ab" or "substring "ab""

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

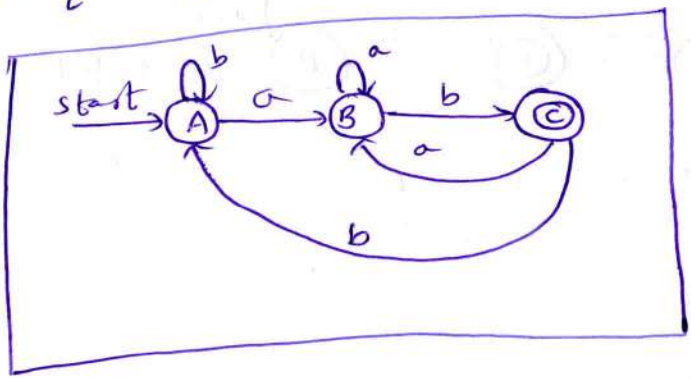
$L = \{ab, aab, bab, aba, abb, \ldots\}$ Infinite



(6) Construct a DFA that accepts set of all strings over $\Sigma = \{a, b\}$ "ending with ab"

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

$L = \{ ab, aab, bab, \ldots\ldots \}$ Infinite

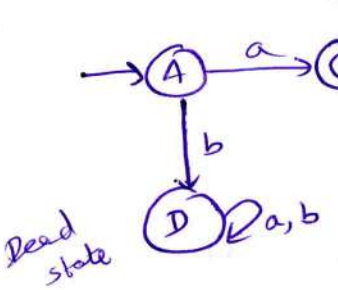


(7) Construct a DFA that accepts set of all strings over $\Sigma = \{a, b\}$ which "starts with 'a'" and "ends with 'b'"

$L = \{ ab, aab, abb, \ldots\ldots \}$ Infinite

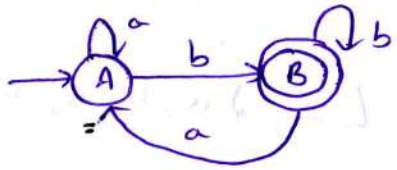
starts with 'a'

$L_1 = \{a, ab, aa, \ldots aaa, \ldots\}$

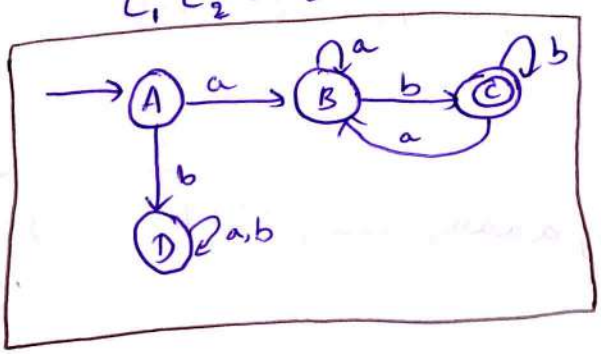


ends with 'b'

$L_2 = \{ b, ab, bb, aab, \ldots \}$



$L_1 L_2 = \{ ab, aab, abb, \ldots\ldots \}$



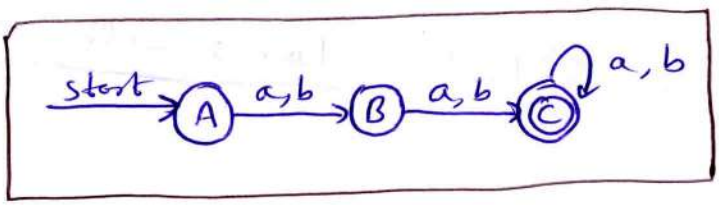
(8) Construct a D-F-A that accepts set of all strings over $\Sigma = \{a, b\}$ whose length is 2?

$L = \{aa, ab, ba, bb\}$ $\Sigma = \{a, b\}$

(9) Construct D-FA that accepts set of all strings over $\Sigma = \{a, b\}$ whose length is greater than (or) equal to 2. $|w| \geq 2$

$L = \{aa, ab, ba, bb, aaa, \ldots\}$

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



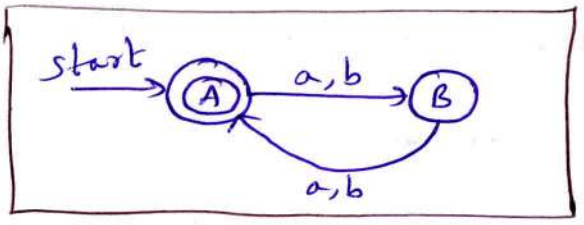
(10) Construct DFA that accepts set of all strings over $\Sigma = \{a, b\}$ whose length is less than (or) equal to 2 $|w| \leq 2$

$L = \{a, b, \epsilon, aa, ab, ba, bb\}$ → Finite

(11) Construct DFA that accepts set of all strings over $\Sigma = \{a, b\}$ whose length is even (Even length strings)

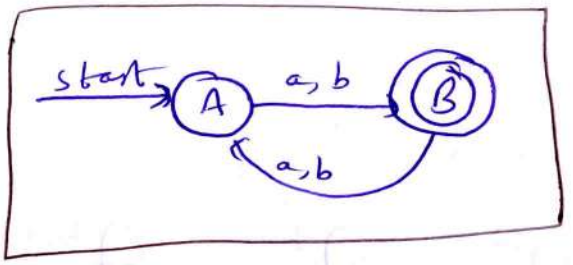
$|w| \bmod 2 = 0$

$L = \{\epsilon, aa, ab, ba, bb, aaaa, \ldots, bbbb, \ldots\}$ → Infinite



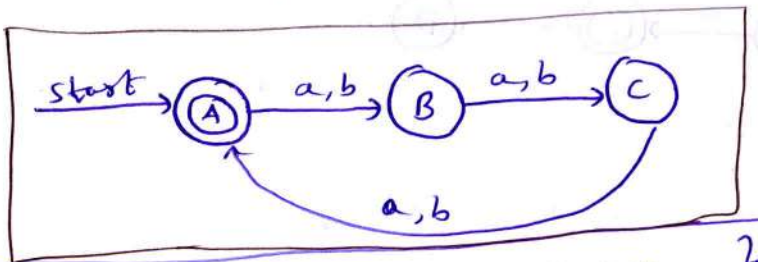
(12) Construct DFA that accepts all strings over $\Sigma = \{a, b\}$ whose length is odd (odd length strings)

$|w| \bmod 2 = 1$ (odd length)



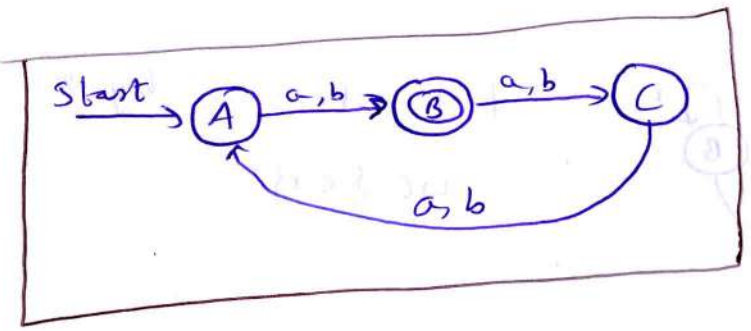
(13) Construct DFA that accepts all strings over $\Sigma = \{a, b\}$ whose length is multiples of 3

$|w| \bmod 3 = 0$



$L = \{\epsilon, aaa, bbb, aba, \ldots\}$ Infinite

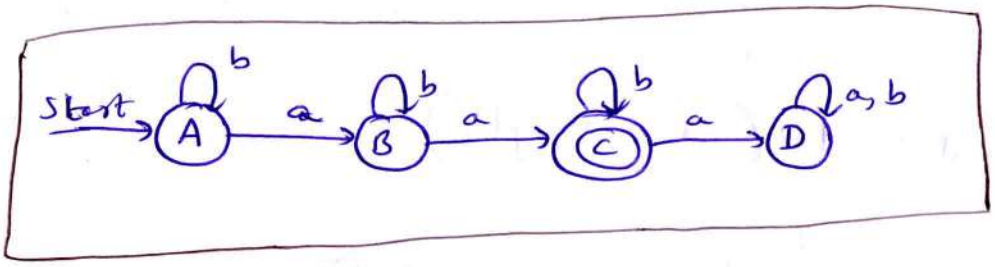
$|w| \bmod 3 = 1$



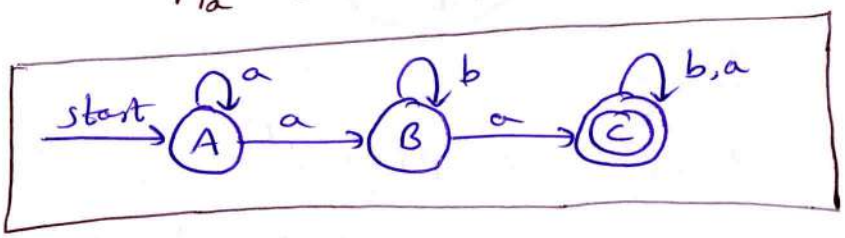
(14) Construct DFA that accepts all strings

where no. of a's = 2 $n_a(w) = 2$

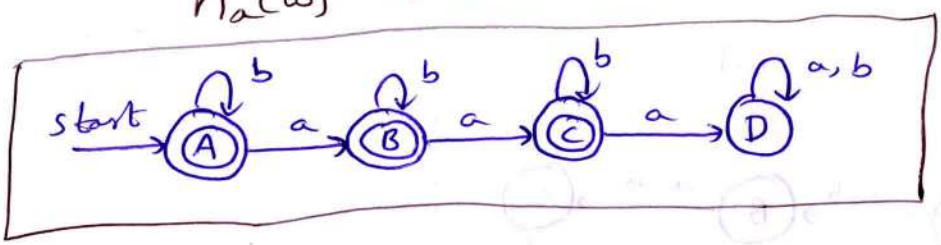
$L = \{aa, baa, aba, aab, \ldots\}$ → Infinite



$n_a(w) \geq 2$

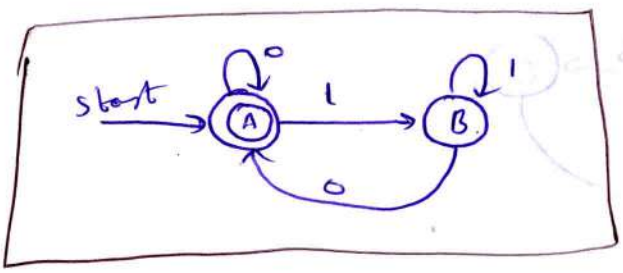


$n_a(w) \leq 2$



(15) Construct DFA that accepts all strings over $\Sigma = \{0,1\}$ Binary number which is divisible by 2

$L = \{10, 100, 1000, 1010, 1100 \ldots\}$

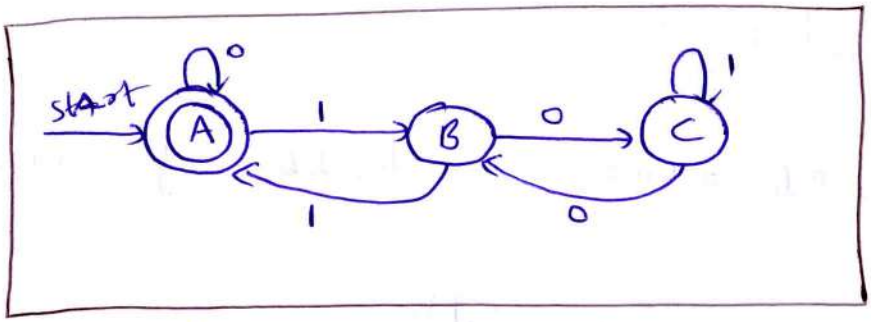


$|w|$ = divisible by 2

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

Binary number divisible by 3.

$|w| =$ is divisible by 3



Shortcut

	0	1
A	A	B
B	C	A
C	B	C

Binary number divisible by 4

	0	1
A	A	B
B	C	D
C	A	B
D	C	D

→ (A)

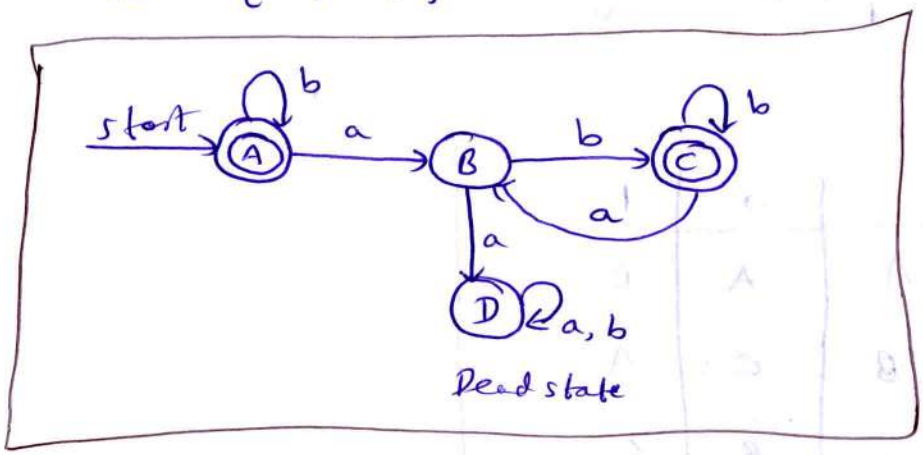
Binary number divisible by 4 & $\Sigma = \{0,1,2\}$

	0	1	2
A	A	B	C
B	D	A	B
C	C	D	A
D	B	C	D

(16) Construct D·FA that accepts set of all strings over $\Sigma = \{a, b\}$ where every 'a' should be followed by 'b'.

$L = \{ \epsilon, ab, abab, \ldots, b, bb, \ldots \}$ → Infinite

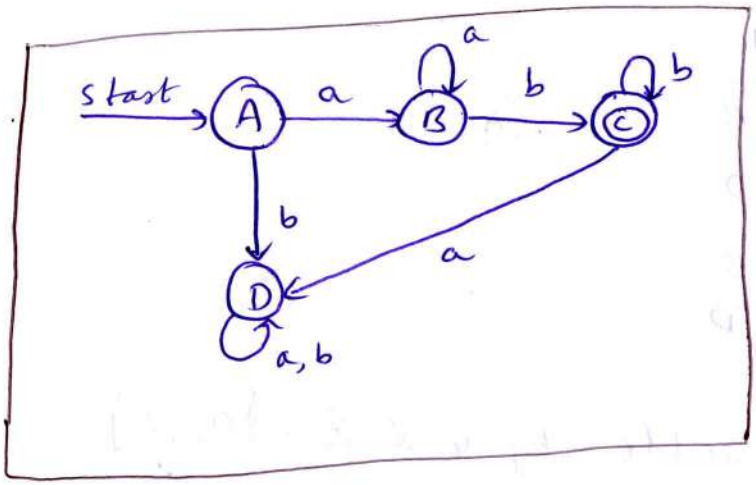
=



(17) Construct D·FA where $L = \{ a^n b^m / n, m \geq 1 \}$

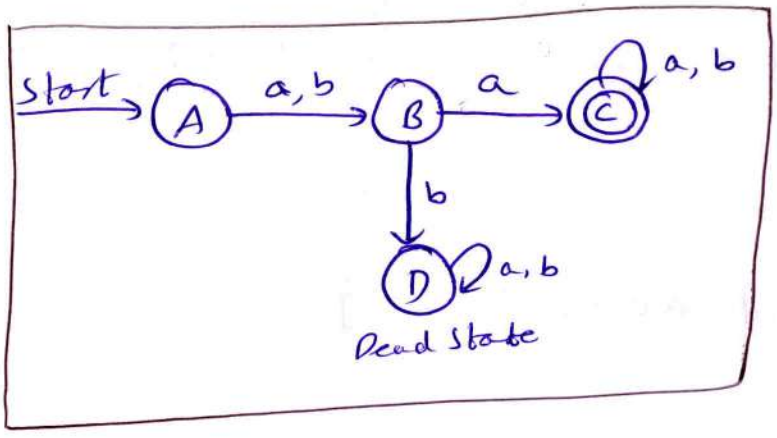
$L = \{ ab, abb, aab, aabb, aaabb \ldots \}$ → Infinite

=



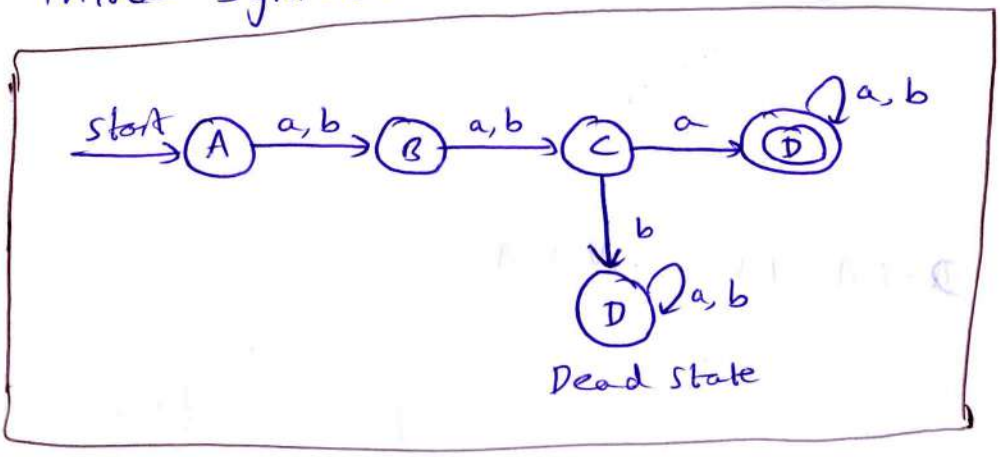
(18) Construct D·FA which accepts set of all strings over $w \in \{a, b\}^*$ such that 2nd string in 'w' from LHS should be 'a'

$L = \{ aaa, aa, baa, aab, baab, \ldots \}$ Infinite



=

Third symbol from L-H's is 'a'



Non Deterministic Finite Automata

→ A "Nondeterministic" finite automata (NFA) has the power to be in several states at once.

Definition of Non Deterministic Finite Automata

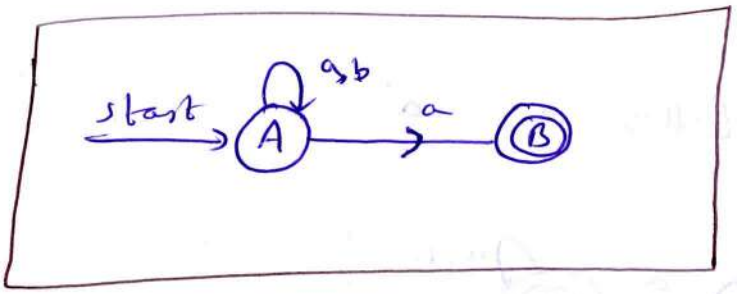
$$A = (Q, \Sigma, \delta, q_0, F)$$

δ : Transition Function $\boxed{Q \times \Sigma \longrightarrow 2^Q}$

→ NFA is important in Text Search.

① Construct NFA over $\Sigma = \{a, b\}$ that ends with a.

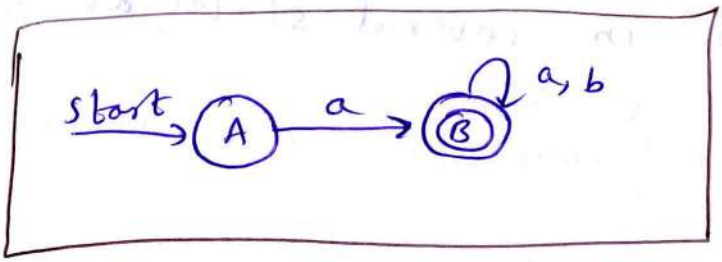
$L = \{ a, ba, aa, aaa, \ldots \}$



** Every D-FA is N-F-A

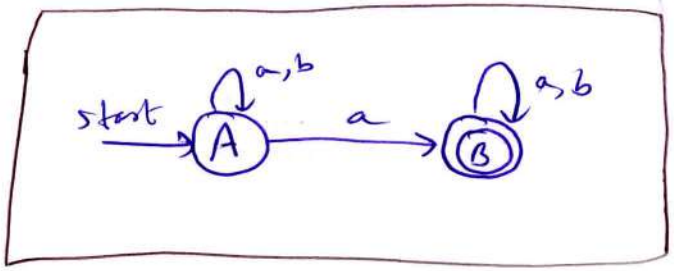
② Construct NFA over $\Sigma = \{a, b\}$ that starts with a

$L = \{ a, ab, aba, aab \ldots \}$



③ Construct NFA over $\Sigma = \{a, b\}$ that contain 'a'

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



** → Both N.F.A and D.F.A are equally Powerful

$$N.F.A \cong D.FA$$

→ As every DFA is NFA no need to convert DFA to NFA.

** → The NFA can be converted DFA.

Subset Construct Method

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

NFA which starts with 'a'

$L = \{a, aa, ab, abb, aab, aba, \ldots\}$

N.F.A

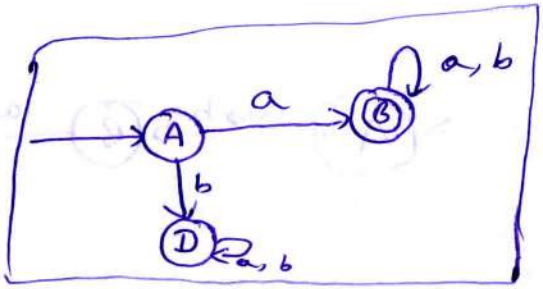
NFA T-T

	a	b
→ A	B	ϕ
★ B	B	B

D.FA T-T

	a	b
→ A	{B}	{D}
★ B	{B}	{B}
D	{D}	{D}

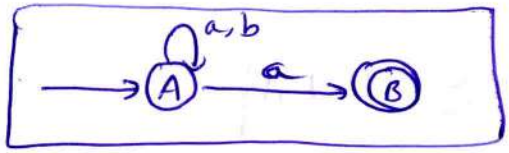
So Final D.F.A



Eg 2:- NFA ends with 'a'

$L_1 = \{a, ba, aa, aba, aaa, \ldots\}$

NFA



N·FA T·T

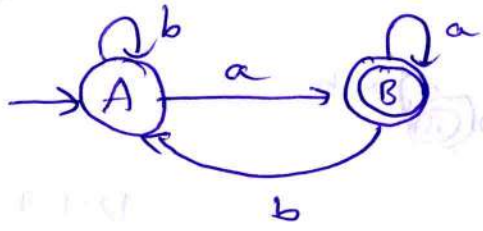
	a	b
→ A	{A,B}	{A}
*B	∅	∅

D·F·A T·T

	a	b
→ [A]	[AB]	[A]
*[AB]	[AB]	[A]

** → start with initial state and draw on each state that are reachable from start state

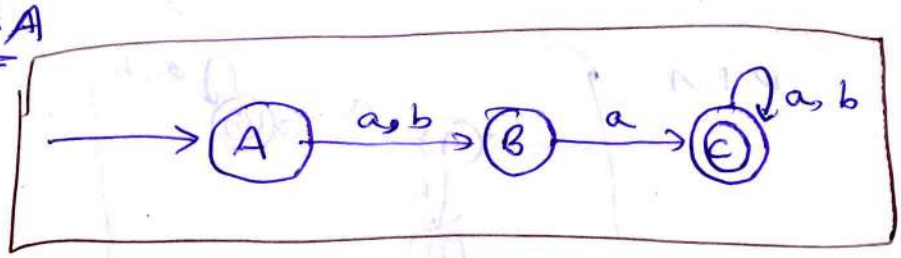
DFA



Eg 3:- N.F.A second symbol from L.H.S is 'a'

$L = \{aa, ba, aab, bab, baa, \ldots\}$

N.F.A



NFA TO DFA CONVERSION :-

NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N) \longrightarrow$ DFA $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$

$L(D) = L(N)$

→ Q_D will have 2^n states where Q_N have n states.

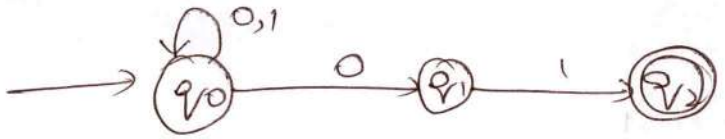
→ F_D is subset of $Q_N \Rightarrow S \cap F_N \neq \phi$. one of state of NFA should be included.

→ for each set $S \subseteq Q_N$ & for each input a in Σ,

$$\delta_D(S, a) = \bigcup_{p \text{ in } S} \delta_N(p, a)$$

To compute $\delta_D(S, a)$ we look at all the states p in S, what state N goes to from p on input a & takes the union of all those state.

Example :- Consider the NFA of accepting all strings ending with "01".



Subset Construction :-

	0	1
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	ϕ	$\{q_2\}$
$*\{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	ϕ	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

$\delta((q_0, q_1, q_2), 0) = (q_0 q_1) \cup \phi \phi = (q_0 q_1)$

$\delta(q_0 q_1 q_2), 1) = q_0 \cup q_2 \cup \phi = q_0 q_2$

$\delta((q_0, q_1), 0) = (q_0 q_1) \cup \phi = \{q_0 q_1\}$

$\delta((q_0, q_1), 1)) = q_0 \cup q_2 = \{q_0 q_2\}$

$\delta((q_0, q_2), 0) = (q_0, q_1) \cup \phi = (q_0, q_1)$

$\delta(q_0, q_2), 1) = (q_0) \cup \phi = q_0$

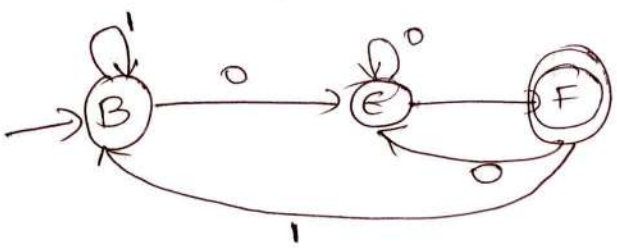
$\delta((q_1, q_2), 0) = \phi \cup \phi = \phi$

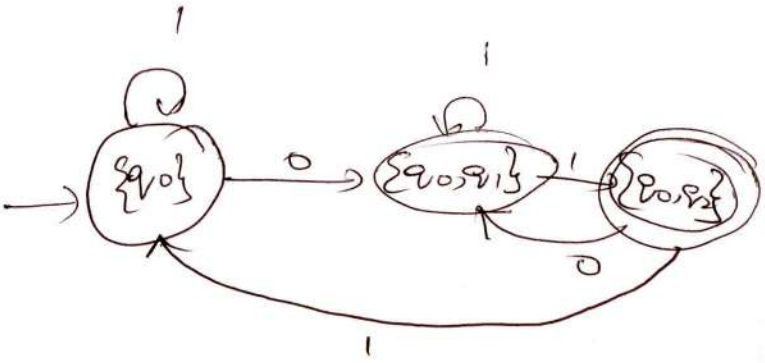
$\delta((q_1, q_2), 1) = q_2 \cup \phi = q_2$

	0	1
A	A	A — X
→ B	E	B
C	A	D — X
* D	A	A — X
E	E	F
* F	E	B
* G	A	D — X
* H	E	F — X

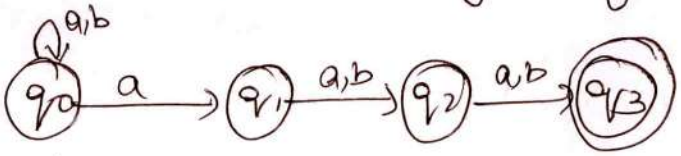
ϕ q_0 q_1 q_2 q_0q_1 q_0q_2 q_1q_2 $q_0q_1q_2$ $q_1q_2q_3$

A B C D E G H





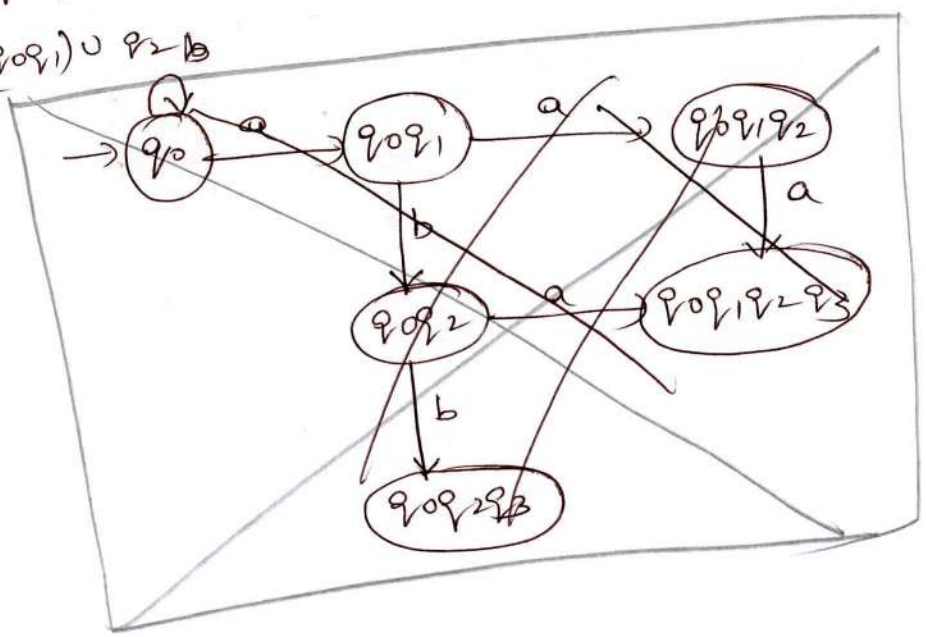
Example :- NFA for the 3rd symbol from RHS is "a".

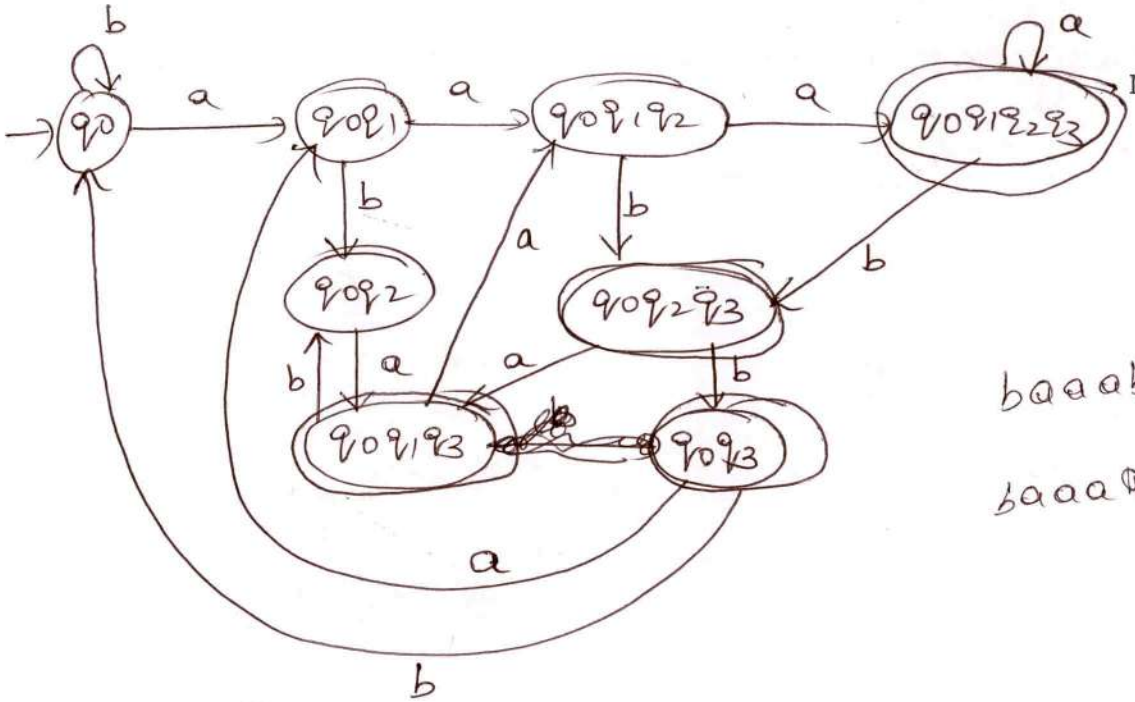


	a	b
→ q_0	$\{q_0q_1\}$	$\{q_0\}$
q_1	q_2	q_2 — X
q_2	q_3	q_3 — X
* q_3	ϕ	ϕ — X
(q_0q_1)	$\{q_0q_1q_2\}$	$\{q_0q_2\}$
$(q_0q_1q_2)$	$\{q_0q_1q_2q_3\}$	$\{q_0q_2q_3\}$
(q_0q_2)	$\{q_0q_1q_3\}$	$\{q_0q_3\}$
* $(q_0q_1q_2q_3)$	$\{q_0q_1q_2q_3\}$	$\{q_0q_2q_3\}$
* $(q_0q_2q_3)$	$\{q_0q_1q_3\}$	$\{q_0q_3\}$
* $q_0q_1q_3$	$\{q_0q_1q_2\}$	$\{q_0q_2\}$
* (q_0q_3)	$\{q_0q_1\}$	$\{q_0\}$

$\delta(q_0,a) \cup \delta(q_0,a)$

$(q_0q_1) \cup q_2$ b



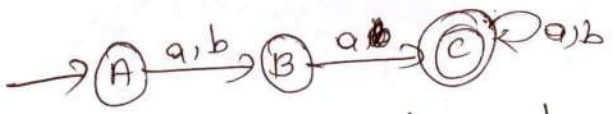


baaabab — ✗

baaaabb — ✓

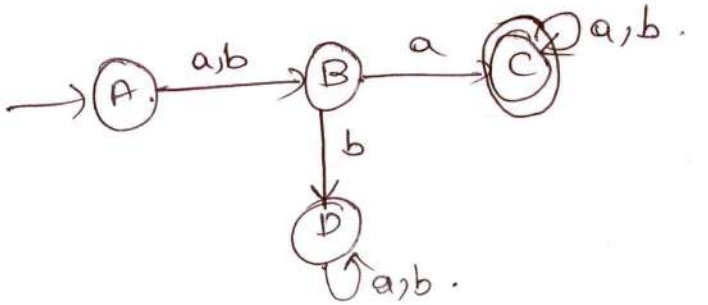
Example-III

NFA :- second symbol from LHS is 'a'.



	a	b
A	B	B
B	C	ϕ
C	C	C

	a	b
→A	{B}	{B}
B	{C}	{D}
*C	C	C
D	D	D

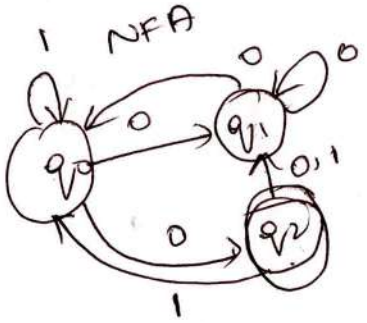


aab — ✓

aba — ✗

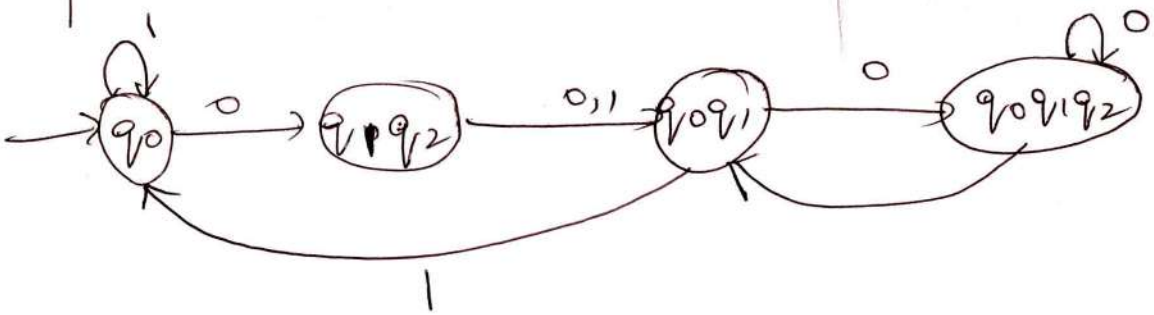
Convert the following NFA to DFA

δ	0	1
$\rightarrow q_0$	$\{q_1, q_2\}$	$\{q_0\}$
q_1	$\{q_0, q_1\}$	ϕ
$*q_2$	$\{q_1\}$	$\{q_0, q_1\}$



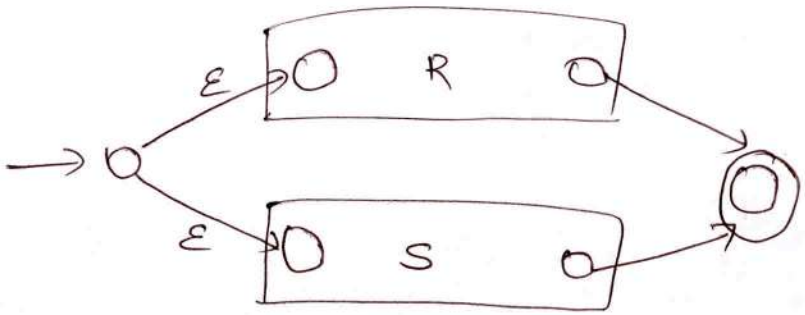
Subset construction:-

δ	0	1
ϕ	ϕ	ϕ
$\rightarrow q_0$	$\{q_1, q_2\}$	$\{q_0\}$
q_1	$\{q_0, q_1\}$	ϕ
$*q_2$	$\{q_1\}$	$\{q_0, q_1\}$
$*\{q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_0\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$

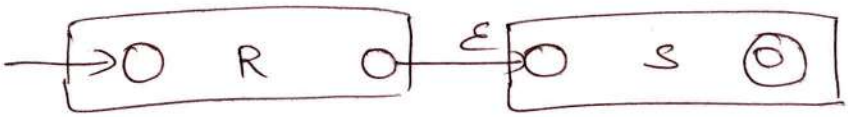


Inductive step in the Regular Expression to ε-NFA Conversion

1) The expression is R+S for some smaller expressions R & S. Starting at the new state, we can go to the start state of either automaton for R or S, following a path labeled by string in L(R) or L(S). Once we reach the accepting state of the automaton for R or S we can follow one of the ε-arcs to the accepting state of the new automaton.



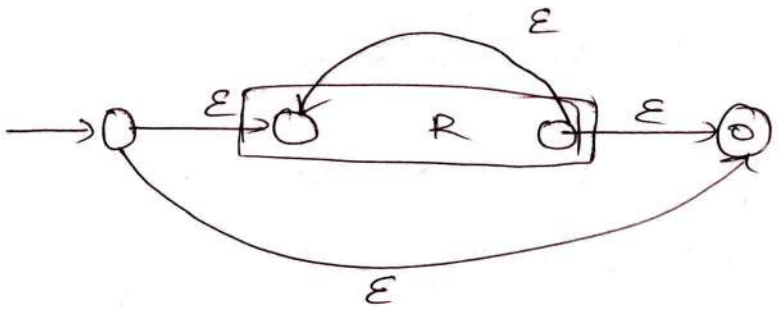
2) The expression is RS for some smaller expression R & S. The idea is that the only paths from start to accepting state go first through the automaton for R where it must follow a path labeled by string in L(R) & then through S.



3) The expression is R^* for some smaller expression R. The automaton allows us either:-

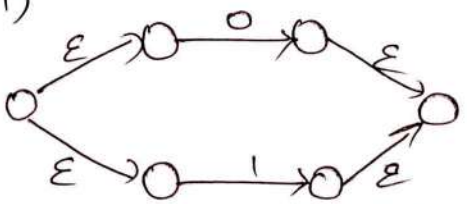
a) Directly from start state to accepting state along path "ε".

b) To the start state of R, through that automaton 1 or more times & then to the accepting state.

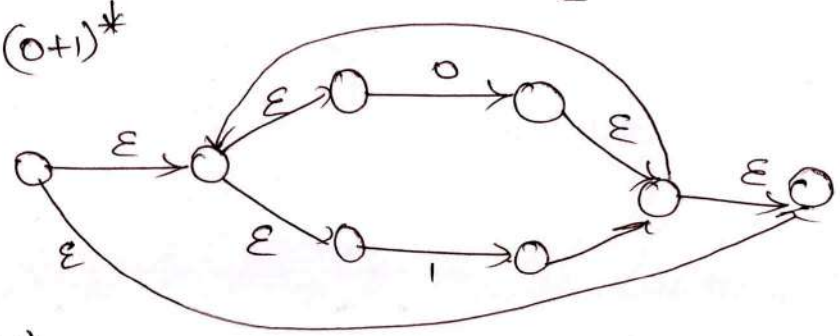


Eg :- Convert the Regular expression $(0+1)^*1(0+1)$ to ε-NFA.

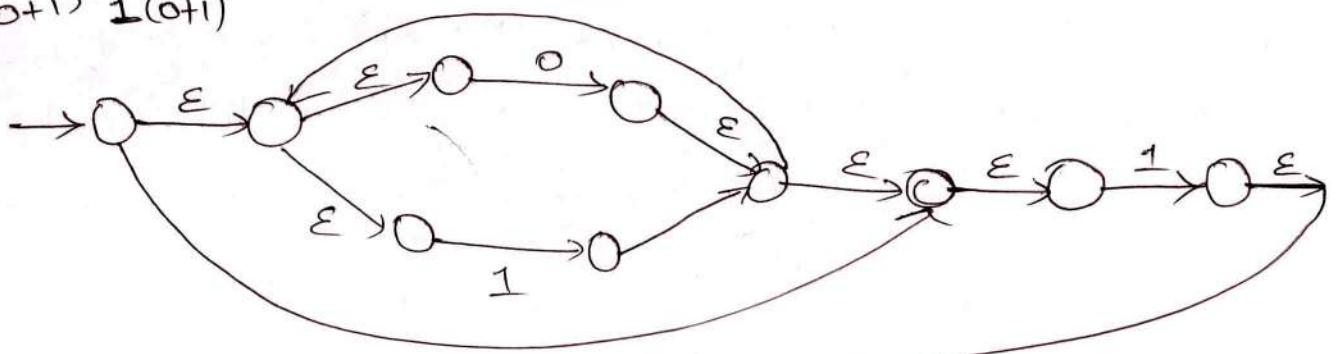
1st step :- $(0+1)$

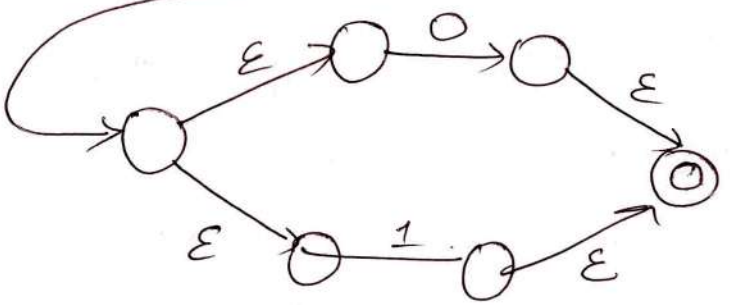


$(0+1)^*$

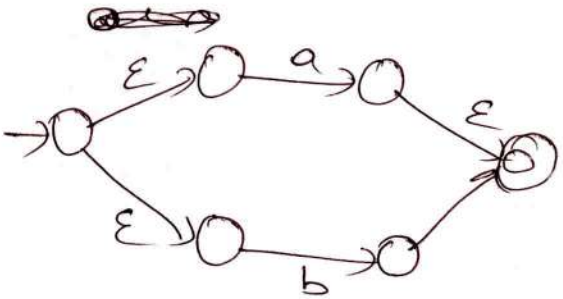


$(0+1)^*1(0+1)$



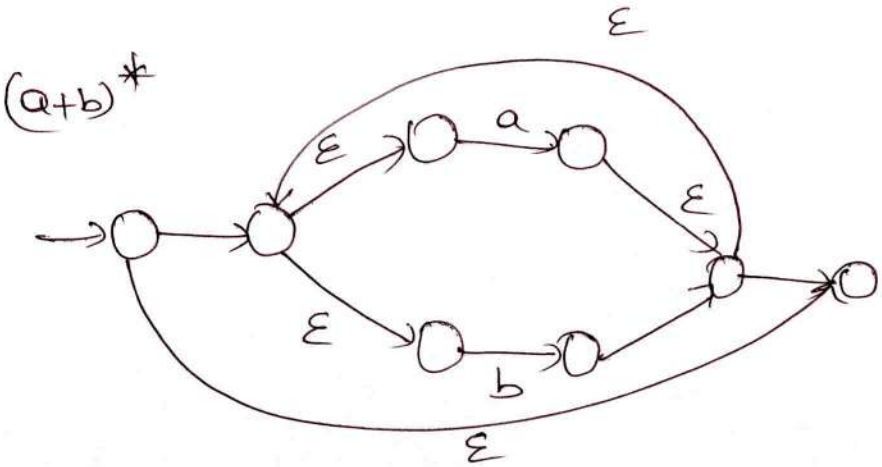


(a+b)

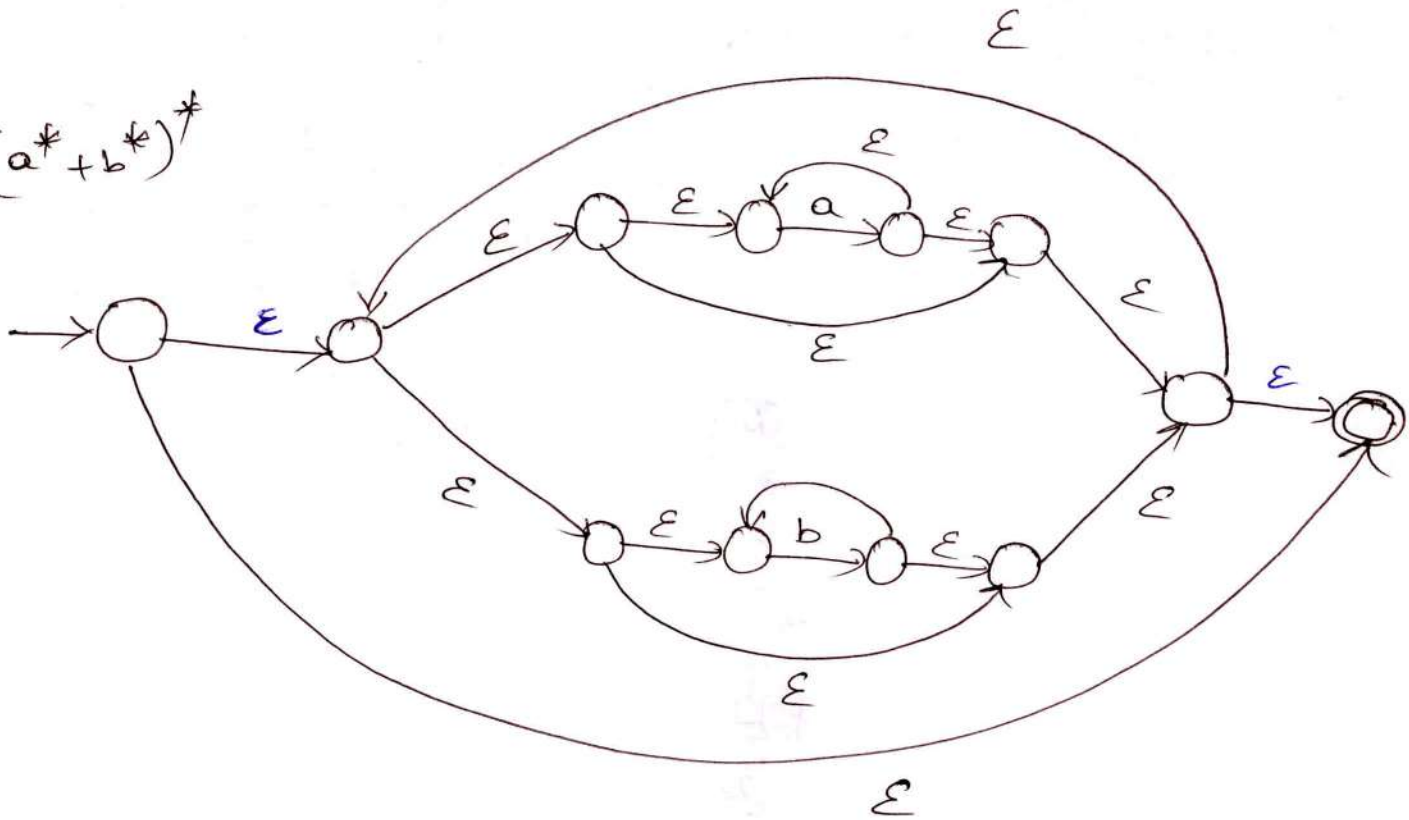


(a.b)

$(a+b)^*$



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



Epsilon closure (ε-closure) :-

→ Epsilon closure of a state is simply the set of all states that we can reach by ε input.

→ Denoted as ε-closure (q)

Epsilon closures of considered example is :-

ε-closure$(q_0) = \{q_0, q_1, q_2\}$

ε-closure$(q_1) = \{q_1, q_2\}$

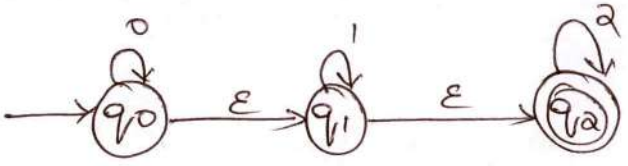
ε-closure$(q_2) = \{q_2\}$

ε-transitions are used for convinience but they do not increase the power of NFA.

To eliminate them, we can convert NFA with ε into an equivalent NFA without ε, by eliminating the ε-edges & replacing them with the edges labelled with symbols present in Σ.

Converting NFA with ε-transition to NFA without ε-Transition :-

Ex :-



Transition table :-

	0	1	2	ε
→q_0	q_0	ϕ	ϕ	q_1
q_1	ϕ	q_1	ϕ	q_2
*q_2	ϕ	ϕ	q_2	ϕ

NFA with epsilon - (ε) transitions :-

→ We can extend an NFA by introducing 'ε-moves' that allows us to make a transition on the empty string.

→ This new capability does not expand the class of languages that can be added by finite automata but it does gives us some added "programming convenience".

→ There would be an edge labelled 'ε' between two states & this edge allows transition from one state to another even without receiving an input symbol.

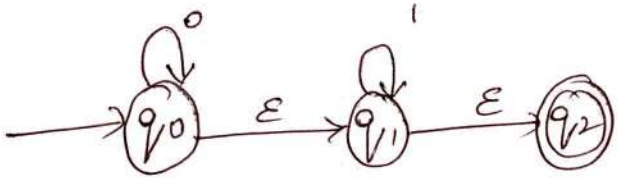
→ Constructing such NFA is easy but the constructed NFA is not that powerful.

→ We can represent ε-NFA exactly as we do an NFA with one exception i.e., the transition function must include information about transitions on 'ε'.

$$\varepsilon\text{-NFA} = (Q, \Sigma, \delta, q_0, F)$$

where δ is defined as $Q \times \Sigma \cup \{\varepsilon\} \longrightarrow 2^Q$.

Example :- εNFA that accepts strings of form $\{0^n 1^m 2^p\}$ i.e., any number of 0's followed by any number of 1's followed by any number of 2's.



** Every state on 'ε' goes to itself.

Step-I :- Find ε-closure of each state

ε-closure $(q_0) = \{q_0, q_1, q_2\}$

ε-closure $(q_1) = \{q_1, q_2\}$

ε-closure $(q_2) = \{q_2\}$

Step-II :- Find the transition on each state for each element.

$\hat{\delta}(q_0, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_0, 0)))$

$= \varepsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, 0)))$

$= \varepsilon\text{-closure}(q_0 \cup \phi \cup \phi)$

$= \{q_0, q_1, q_2\}$

$\hat{\delta}(q_0, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_0, 1)))$

$= \varepsilon\text{-closure}(\delta(q_0, q_1, q_2), 1)) \rightarrow \varepsilon\text{-closure}(q_1)$

$= \{q_1, q_2\}$

$\hat{\delta}(q_0, 2) = \varepsilon\text{-closure}(\delta(q_0, q_1, q_2), 2)) \Rightarrow \{q_2\}$.

$\hat{\delta}(q_1, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_1, 0)) = \varepsilon\text{-closure}(\delta(q_1, q_2), 0) \Rightarrow \phi$

$\hat{\delta}(q_1, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_1, 1)) = \varepsilon\text{-closure}(\delta(q_1, q_2), 1) = \{q_1, q_2\}$

$\hat{\delta}(q_1, 2) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_1, 2)) = \varepsilon\text{-closure}(\delta(q_1, q_2), 2) = \{q_2\}$

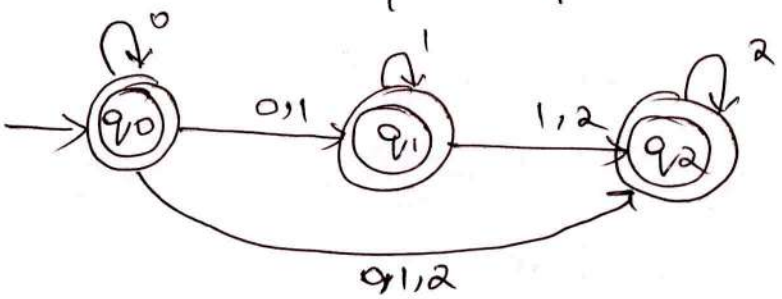
$\hat{\delta}(q_2, 0) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_2, 0)) = \varepsilon\text{-closure}(\delta(q_2, 0)) = \phi$

$\hat{\delta}(q_2, 1) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_2, 1)) = \varepsilon\text{-closure}(\delta(q_2, 1)) = \phi$

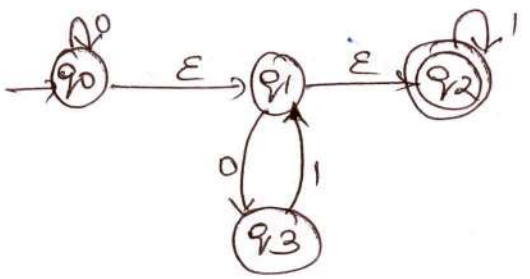
$\hat{\delta}(q_2, 2) = \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(q_2, 2)) = \varepsilon\text{-closure}(\delta(q_2, 2)) = \{q_2\}$

NFA without ε-transitions :-

	0	1	2
→ * q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
* q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
* q_2	ϕ	ϕ	$\{q_2\}$



Example 2:- ε-NFA to NFA



	0	1	ε
→ q_0	q_0	ϕ	q_1
q_1	q_3	ϕ	q_2
* q_2	ϕ	q_2	ϕ
q_3	ϕ	q_1	ϕ

ε-closure(q_0) ⇒ $\{q_1, q_2, q_0\}$

ε-closure(q_1) ⇒ $\{q_1, q_2\}$

ε-closure(q_2) ⇒ $\{q_2\}$

ε-closure(q_3) ⇒ $\{q_3\}$

$\hat{\delta}(q_0,0)$ = ε-closure(δ(ε-closure(q_0),0)) = ε-closure($\delta(q_0,q_1,q_2),0$))

= ε-closure($q_0 q_3$)

= $\{q_0 q_1 q_2 q_3\}$

$\hat{\delta}(q_0,1)$ = ε-closure($\delta(q_0,q_1,q_2),1$) = ε-closure(q_2) = $\{q_2\}$

$\hat{\delta}(q_1,0)$ = ε-closure($\delta(q_1,q_2),0$)) = ε-closure(q_3) = $\{q_3\}$

$\hat{\delta}(q_1,1)$ = ε-closure($\delta(q_1,q_2),1$) = ε-closure(q_2) = $\{q_2\}$

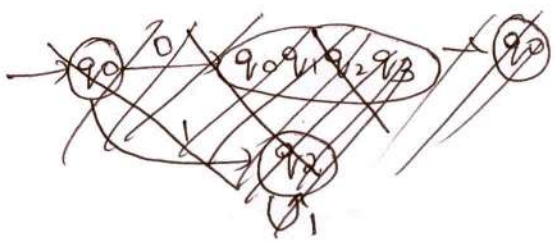
$\hat{\delta}(q_2,0)$ = ε-closure($\delta(q_2),0$) = ε-closure(ϕ) = ϕ

$\hat{\delta}(q_2,1)$ = ε-closure($\delta(q_2),1$) = ε-closure(q_2) = $\{q_2\}$

$\hat{\delta}(q_3,0)$ = ε-closure($\delta(q_3),0$) = ε-closure(ϕ) = ϕ

$\hat{\delta}(q_3,1)$ = ε-closure($\delta(q_3),1$) = ε-closure(q_1) = $\{q_1 q_2\}$

	0	1
* → q_0	$\{q_0 q_1 q_2 q_3\}$	$\{q_2\}$
* q_1	$\{q_3\}$	$\{q_2\}$
* q_2	ϕ	$\{q_2\}$
q_3	ϕ	$\{q_1 q_2\}$



Regular Expressions :-

→ Regular expressions are useful for representing certain sets of strings in an algebraic fashion.

→ We can find the regular expressions that can exactly define same languages that the various forms of automata defines i.e the regular languages.

Basic operations :-

→ Any terminal symbol/element of Σ is Regular Expression (RE).

Ex :- ϕ is an RE & denotes the empty set.

ϵ is an RE & denotes the set $\{\epsilon\}$

a is an RE & denotes the set $\{a\}$

→ Union of 2 languages :-

Denoted as "$L \cup M$" - the set of strings that are in either L or M or both.

Ex :- $L = \{001, 10, 111\}$ $M = \{\epsilon, 001\}$

$L \cup M = \{\epsilon, 001, 10, 111\}$

→ Concatenation of 2 languages :-

Set of All strings that can be formed by taking any string in L & concatenating it with any string in M.

Denoted as $L.M$ or LM.

Ex :- $L = \{001, 10, 111\}$ & $M = \{\epsilon, 001\}$

LM or $L \cdot M \Rightarrow \{001, 10, 111, 001001, 10001, 111001\}$

(001, 10, 111 : concatenating with ϵ; 001001, 10001, 111001 : concatenating with 001)

→ closure or star or Kleene closure :-

→ Denoted as L^* for language L.

→ Represents the set of those strings that can be formed by taking any no of strings from L, possibly with repetitions & concatenating all of them.

Ex :- $L = \{0,1\}$ then L^* is all strings of 0's & 1's.

$L = \{0,11\}$ then L^* is those strings of 0's & 1's such that the 1's come in pairs e.g :- 011, 11110 $\in$ but not 01011 or 101.

$L^1 = L$

$L^2 = \{00, 011, 110, 1111\}$

$L^3 = \{000, 0011, 0110, 1100, 01111, 11011, 11110, 111111\}$

Identity Rules for Regular expressions :-

P & Q are two equivalent regular expressions (i.e., P & Q represent the same set of strings) then to simplify the regular expressions, the following identity rules can be used :-

1) $\phi + R = R$, $\varepsilon + R = R + \varepsilon$

2) $\varepsilon R = R\varepsilon = R$

3) $R + R = R$

4) $RR^* = R^*R = R^+$

5) $\varepsilon + RR^* = R^* = \varepsilon + R^*R$

6) $\phi R = R\phi = \phi$

7) $\varepsilon^* = \varepsilon$, $\phi^* = \varepsilon$

8) $(R^*)^* = R^*$

9) $(PQ)^*P = P(QP)^*$

10) $(P+Q)R = PR + QR$

11) $R(P+Q) = RP + RQ$

12) $(P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^*$

Examples :-

1) All strings of 0's & 1's

A) Language ⇒ {ε, 0, 1, 00, 01, 10, 11, 000 ----}

R.E ⇒ (0+1)*

2) Set of all strings of 0's & 1's ending in 00.

A) Language ⇒ {00, 000, 100, 0000, 0100, 1000, 1100, 00000--}

This can be written as {ε, 0, 1, 00, 01, 10, 11, 000, ---} 00

R.E ⇒ (0+1)*00

3) Set of all strings of 0's & 1's beginning with 0 & ending with 1.

A) Language ⇒ {01, 001, 0001, 011, 0011, 0101, 0110, 00001---}

This can be written as {ε, 0, 1, 00, 01, 10, 11, 000 ---} 1

R.E ⇒ 0(0+1)*1

4) Set of all strings having even no of 1's.

A) Language ⇒ {ε, 11, 1111, 111111 ----}

R.E ⇒ (11)*

5) Set of all strings having odd no of 1's

A) Language ⇒ {1, 111, 11111 - ---}

This can be written as {ε, 11, 1111, 111111 --- ---} 1

(11)*1 or 1(11)*

6) Set of all strings that consists of alternating 0s & 1's.

A) All strings begin '0' & end with 1 $\longrightarrow (01)^*$

All strings begin '1' & end with 0 $\rightarrow (10)^*$

All strings begin & end with 0 $\longrightarrow 0(10)^*$

All strings begin & end with 1 $\longrightarrow 1(01)^*$

$$(01)^* + (10)^* + 0(10)^* + 1(01)^*$$

Applications of Regular Expressions

→ Operating systems (Unix)

→ Compilation (Lexical Analysis)

→ Programming Languages (Java)

Regular Expressions in UNIX

→ Basically Regular Expressions are used in search tools

→ Text file search in Unix (tool: egrep)

→ egrep :- This command searches for a text pattern in the file and list the words containing that pattern.

Eg:- File: ABC

Sam
Dexter
John
Raman

[a-z]

[a...z]

+ - one or more occurences

* - zero or more occurences

Command : % egrep "n" ABC

search words in file ABC with 'n'

John
Raman

Regular Expressions in Lexical Analysis

→ Interaction of lexical Analyzer with parser.

→ The lexical Analyzer (lexer) reads source code and generates a stream of tokens.

→ • Examples of Tokens : Identifier, Keyword, Constant, operator, special symbol.

→ Tokens can be described using Regular Expressions.

→ The rule describing how token can be formed.

Eg: Identifier - RE $([a-z] \mid [A-z])\,([a-z] \mid [A-z] \mid [0-9])^*$

Regular Expressions in Java:

→ Regular Expressions are a language of string Patterns built in to most modern programming Languages, including Java 1.4 onwards

→ Regular Expressions used for searching, extracting and modifying text.

→ java.util.regex contains classes for working with regular expressions in Java.

Regular Expressions in finding patterns in text:

→ Regular Expressions are useful notations used for finding patterns from given text.

Eg: The strings ending with digits is a pattern in the given text which can be recognized by following R-E:

$$[a-z^{*}A-Z]^{+}[0-9]^{+}$$

Algebraic Laws & Identity Rules of Regular Expressions

→ we have algebriac laws for Regular Expressions regarding following operations:

1) Union operation on Regular Expressions
2) Concatenation operation on Regular Expressions
3) Concatenation over union operation on Regular Expressions-

Algebraic Laws for Regular Expression :-

i) Commutative Law for union :-

→ L+M = M+L - says that we may take the union of 2 languages in either order.

ii) Associative Law for union :-

→ (L+M)+N = L+(M+N) - says that we may take union of 3 languages either by taking the union of first two initially or taking the union of the last two initially.

iii) Associative Law for Concatenation :-

→ (LM)N = L(MN) - Says that we can concatenate 3 languages by concatenating either the first two or the last two initially.

iv) Identity for union :-

→ $\phi + L = L + \phi = L$

v) Identity for Concatenation :-

→ $\epsilon L = L\epsilon = L$

vi) Left distributive Law of concatenation over union :-

→ L(M+N) = LM+LN.

vii) Right distributive Law of Concatenation over union :-

→ (M+N)L = ML+ NL.

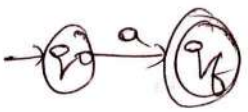
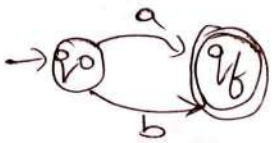
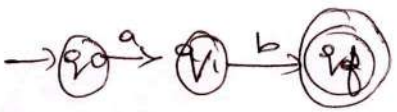
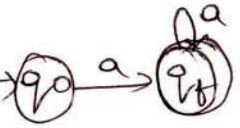
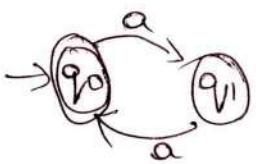
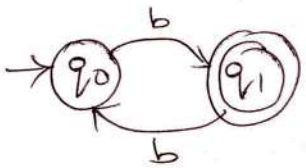
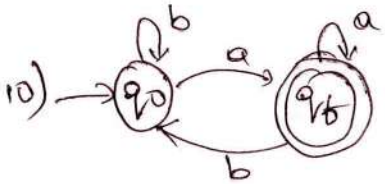
ix) Idempotent Law :-

→ L+L = L. - If we take union of 2 identical expressions, we can replace them by one copy of the expression.

Equivalence of Finite Automata with Regular Expression:-

→ For any given regular expression we can construct its equivalent finite automaton that recognizes the language it describes & vice versa.

Relation b/w FA, RS & RE :-

	Finite Automata	Regular set	Regular Expression
1)		ϕ	ϕ
2)		$\{\varepsilon\}$	ε
3)		$\{a\}$	a
4)		$\{a, b\}$	$a+b$ $a\|b$
5)		$\{ab\}$	$a \cdot b$
6)		$\{\varepsilon, a, aa \text{----}\}$	a^*
7)		$\{a, aa \text{----}\}$	a^+
8)		$\{\varepsilon, aa, (aa)^2 \text{---}\}$	$(aa)^*$
9)		$\{b, bbb \text{----}\}$	$b(bb)^*$
10)		$\{a, aa, aaa \text{----}, ba, bba \text{---}, baba \text{---}\}$	$b^*a(a^*bb^*a)^*$ or $(b^*aa^*b)^*b^*a$

ARDEN'S THEOREM

** If P and Q are two Regular Expressions over Σ, and if P does not contain ϵ, then the following equation in R given by $R = Q + RP$ has a unique solution i.e $R = QP^*$

$$R = Q + RP \longrightarrow (1) \qquad R = QP^*$$

$$= Q + QP^*P$$

$$= Q(\epsilon + P^*P) \qquad (\epsilon + R^*R = R^*)$$

$$\boxed{R = QP^*}$$

→ we have proved QP^* is solution.

→ $R = Q + RP$

$= Q + (Q + RP)P$

$= Q + QP + RP^2$

$= Q + QP + (Q + RP)P^2$

$= Q + QP + QP^2 + RP^3$

$\vdots$

$= Q + QP + QP^2 + \cdots\cdots + QP^n + RP^{n+1}$ Replace $[R = QP^*]$

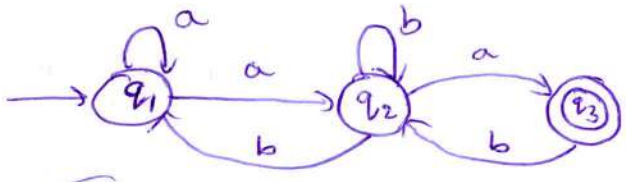
$= Q + QP + QP^2 + \cdots\cdots + QP^n + QP^*P^{n+1}$

$= Q(\epsilon + P + P^2 + \cdots\cdots + P^n + P^*P^{n+1})$

$\boxed{R = QP^*}$ Unique solution

Designing Regular Expressions - Examples

Q Find the Regular Expression for the following NFA



$q_3 = q_2 a \rightarrow (1)$

$q_2 = q_2 b + q_3 b + q_1 a \rightarrow (2)$

$q_1 = \epsilon + q_1 a + q_2 b \rightarrow (3)$

Substitute 2 in 1

(1) $q_3 = q_2 a$

$= (q_1 a + q_2 b + q_3 b) a$

$\boxed{q_3 = q_1 aa + q_2 ba + q_3 ba} \rightarrow (4)$

(2) $q_2 = q_1 a + q_2 b + q_3 b$ [substitute q_3 from (1)]

$= q_1 a + q_2 b + q_2 ab$

$\underbrace{q_2}_{R} = \underbrace{q_1 a}_{Q} + \underbrace{q_2}_{R}\underbrace{(b+ab)}_{P}$ $\quad (R = Q + RP)$

$\Downarrow$

$R = QP^*$

$\boxed{q_2 = q_1 a (b+ab)^*} \rightarrow (5)$

(3) $q_1 = \epsilon + q_1 a + q_2 b$ [substitute q_2 from (5)]

$= \epsilon + q_1 a + (q_1 a (b+ab)^*) b$

$\underbrace{q_1}_{R} = \underbrace{\epsilon}_{Q} + \underbrace{q_1}_{R}\underbrace{(a + a(b+ab)^*) b}_{P}$ $\quad R = Q + RP$

$\Downarrow$

$R = QP^*$

$q_1 = \epsilon \underbrace{(a + a(b+ab)^*) b}_{R}{}^*$ $\quad \epsilon R = R$

$\boxed{q_1 = ((a + a(b+ab)^*) b)^*} \rightarrow (6)$

$q_3 = q_2 a$ [substitute q_2 from ⑤]

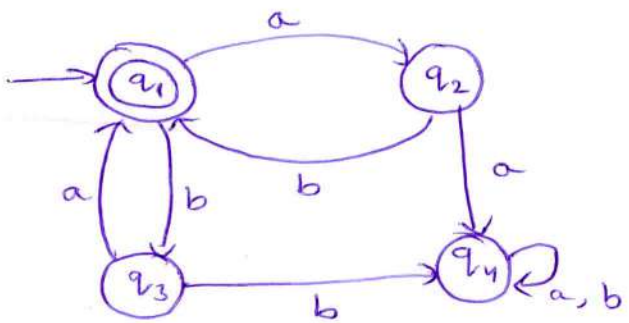
$= q_1 a (b+ab)^* a$ [substitute q_1 from ⑥]

$q_3 = \left(\left(a + a(b+ab)^*\right) b\right)^* a (b+ab)^* a$

Final Regular Expression is

$$\left(\left(a + a(b+ab)^*\right) b\right)^* a (b+ab)^* a$$

(Q) Finding the Regular Expression for the following DFA



$q_1 = \epsilon + q_2 b + q_3 a$ ①

$q_2 = q_1 a$ ②

$q_3 = q_1 b$ ③

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$ ④

$q_1 = \epsilon + q_2 b + q_3 a$ [substitute q_2 from ② q_3 from ③]

$q_1 = \epsilon + q_1 a b + q_1 b a$

$\underbrace{q_1}_{R} = \underbrace{\epsilon}_{Q} + \underbrace{q_1}_{R}\underbrace{(ab+ba)}_{P}$

$R = Q + RP$

$R = QP^*$

$\epsilon \cdot R = R$

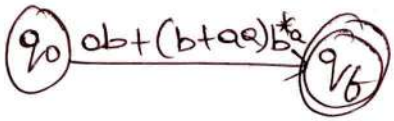
$q_1 = \epsilon (ab+ba)^*$

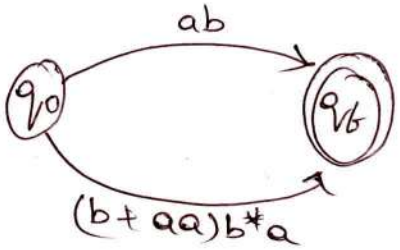
$q_1 = (ab+ba)^*$

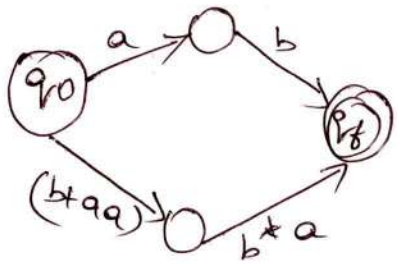
Regular Expression $(ab+ba)^*$

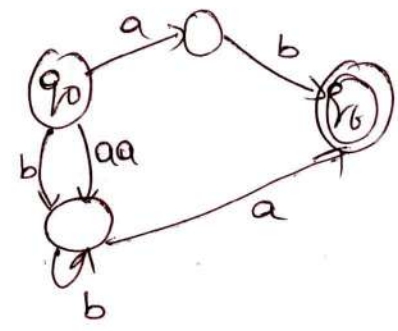
R.E TO NFA

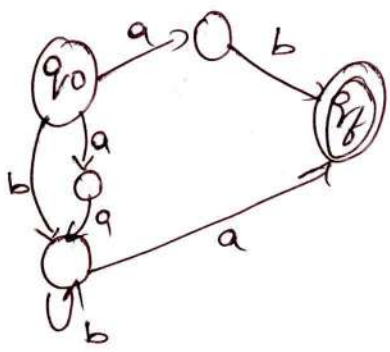
1) $[ab+(b+aa)b^*a]$



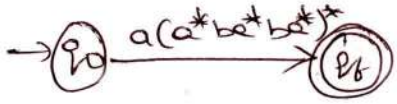


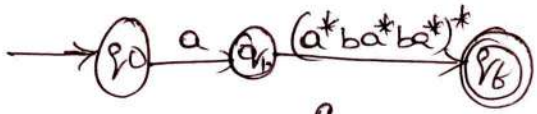


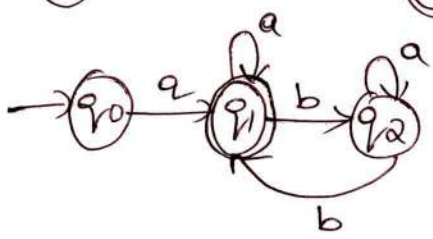




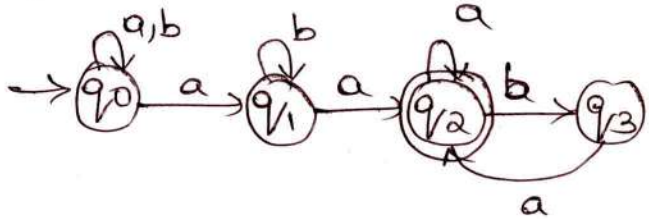
2) $[a(a^*ba^*ba^*)^*]$



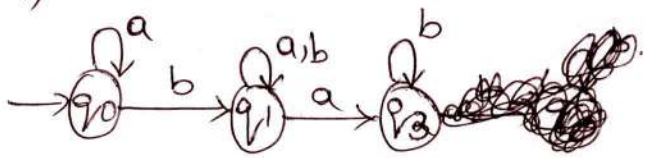




3) $(a+b)^*ab^*a(a+ba)^*$

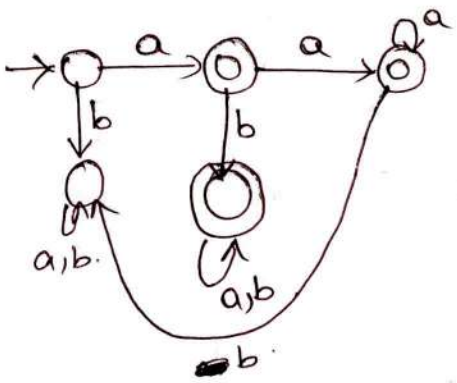


4) $a^*b(a+b)^*ab^*$



R.E TO DFA

1) $(aa^* + aba^*b^*)$

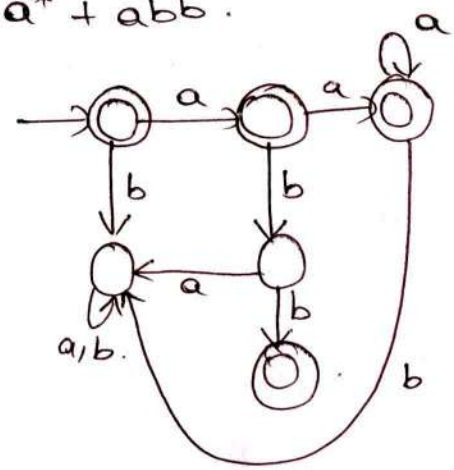


2) $(aa^*)^* + abb$.

$a.a^* \rightarrow a^+$

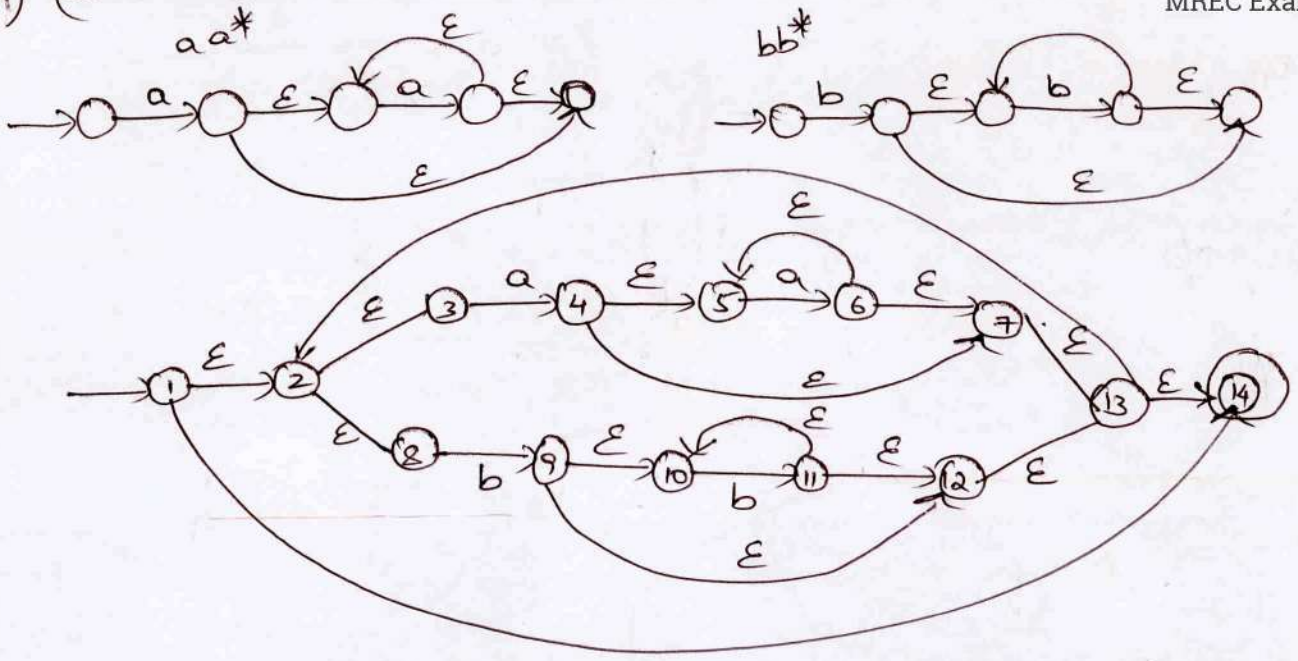
$(a^+)^* \rightarrow a^*$

$a^* + abb$.



R.E TO ε-NFA

1) $(aa^* + bb^*)^*$



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

$a^* + b^* \Rightarrow$