

UNIT-2

PART-1

Regular Expressions : Finite Automata and Regular Exp

- Applications of Regular Expressions,
- Algebraic laws for Regular Expressions.
- Conversion of finite automata to Regular Expressions.

PART-2

Pumping lemma for Regular language - Statements of the pumping lemma

- Applications of the pumping lemma.

PART-3

- Closure properties of Regular languages - Closure properties of Regular languages
- Decision properties of Regular languages
 - Equivalence and minimization of automata

Regular Expressions.

Regular sets - A special class of sets of words over S , called regular sets, is defined recursively as follows (Kleene proves that any set recognized by an FSM is regular. Conversely, every regular set can be recognized by an FSM)

1. Every finite set of words over S (including ϵ , the empty set) is a regular set.
2. If A and B are regular set over S , then $A \cup B$ and AB are also regular
3. If S is a regular set over S , then so is its closure S^*
4. No set is regular unless it is obtained by a finite no. of applications of definitions (1) to (3).
i.e. the class of regular sets over S is the smallest class containing all finite sets of words over S and closed under union, Concatenation and Star operation

Examples

1. Let $\Sigma = \{a, b\}$ then the set of strings that contains both odd no. of a's and b's is a regular set.
2. Let $\Sigma = \{0\}$ then the set of strings $\{0, 00, 000, \dots\}$ is a regular set.
3. Let $\Sigma = \{0, 1\}$ then the set of strings $\{01, 10, \dots\}$ is a regular set.

Regular Expressions

→ The language accepted by finite automata can be easily described by simple expressions called Regular Expressions. It is the most effective way to represent any language.

+ The languages accepted by some regular expression are referred to as Regular languages.

→ A regular expression can also be described as a sequence of patterns that defines a string.

→ A regular expression is used to match character combinations in strings. String searching algorithm uses this pattern to find the operations on a string.

For instance:

- In a regular expression x^* means zero or more occurrence of x . It can generate $\{\epsilon, x, xx, xxx, \dots\}$

- In a regular expressions x^+ means one or more occurrence of x . It can generate $\{x, xx, xxx, \dots\}$

Operations on Regular Language

The various operations on regular languages are.

Union: If L and M are two regular languages then their union $L \cup M$ is also a union.

$$* \quad L \cup M = \{s/s \text{ is in } L \text{ or } s \text{ is in } M\}$$

Intersection: If L and M are two regular languages then their intersection is also an intersection

$$L \cap M = \{st / s \text{ is in } L \text{ and } t \text{ is in } M\}$$

Kleen closure: If L is a regular language then its kleen closure L^* will also be a regular language

L^* = Zero or more occurrence of language L .

Complementation: Let $M = (Q, \Sigma, \delta, q, F)$ be a DFA. Then complementary language in $\Sigma^* - L(M)$ is accepted by a DFA $M' = (Q, \Sigma, \delta, q, Q - F)$. M' is identical of M except that final and non final states are interchanged

Example

1 Write the regular expression for the language accepting all combinations of a's over the set $\Sigma = \{a\}$

Sol

All combinations of a's means a may be zero, one, double and so on...

If a is appearing zero times, that means a null string. That is we expect the set of

$\{\epsilon, a, aa, \dots\}$ so we give a regular expression for this as $R = a^*$ $\Rightarrow$ That is Kleene closure of a

2 Write the regular expression for the language accepting all combinations of a's except the null string over the set $\Sigma = \{a\}$

Sol The regular expression has to be built for the language

$$L = \{a, aa, aaa, \dots\}$$

This set indicates that there is no null string so we can denote regular expression as

$$R = a^+$$

3 Write the regular expression for the language accepting all string containing any no. of a's and b's

Sol: The regular expression will be

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

This will give the set as $L = \{\epsilon, a, aa, b, bb, ab, ba, aba, bab, \dots\}$ any combination of a and b

The $(a+b)^*$ shows any combination with a and b even a null string.

4 Write the regular expression for the language accepting all the string which are starting with 1 and ending with 0 over $\Sigma = \{0, 1\}$

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

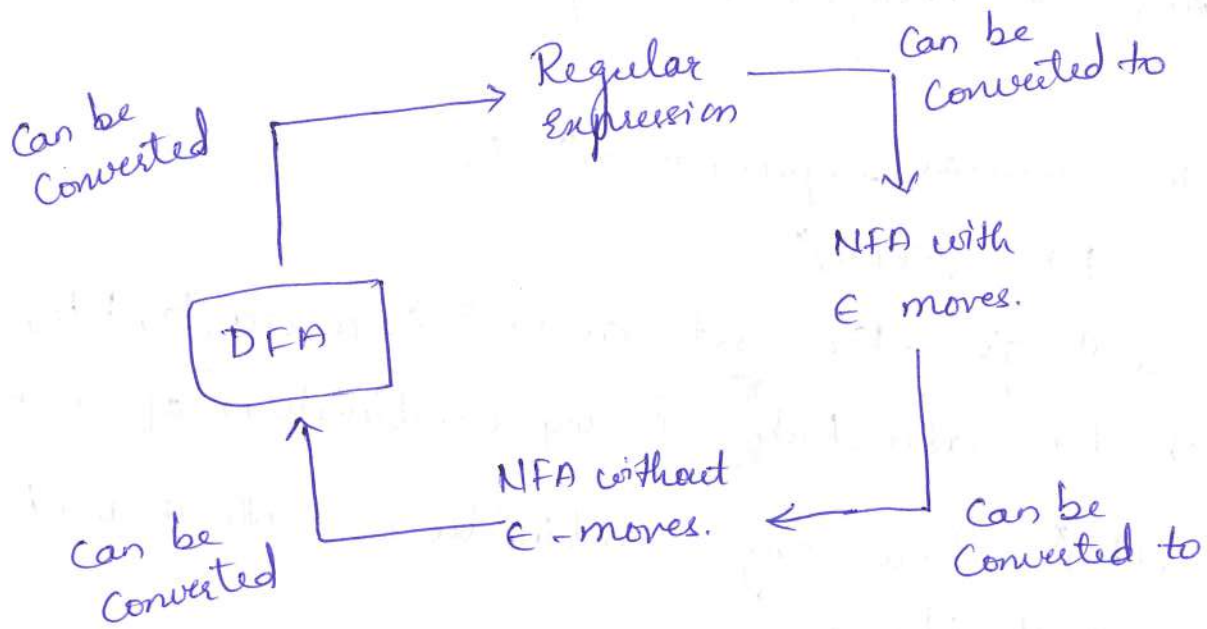
Example for Union $\Rightarrow$ if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 0, 11\}$ then $L \cup M = \{\epsilon, 0, 11, 001, 10, 111\}$

Eg Concatenation $\Rightarrow$ if $L = \{001, 10, 111\}$ and $M = \{\epsilon, 0, 11\}$ then $LM = \{001, 10, 11, 001011, 10011, 111011\}$

Eg: Kleene (star) closure $\Rightarrow$ if $L = \{0, 11\}$ then $L^* = \{011, 11110, \epsilon, \dots\}$

Examples for operations are continued in next page.

The relationship between FA and RE is as follows



The above figure explains that it is easy to Convert

- RE to NFA with ϵ moves.
- NFA with ϵ to without ϵ
- NFA without ϵ to DFA
- DFA can be converted easily to R.E

6800.

6774.
13542

~~2-7~~

2-9

Intersection Example

L_1 = All words that begin with an 'a'

$$\therefore r_1 = a(a+b)^*$$

L_2 = All words that end with an 'a'

$$\therefore r_2 = a(a+b)^*a$$

$$\therefore r_1 \cap r_2 = a(a+b)^*a + a$$

Complement Example

L_1 = All strings with a double 'a' then
 L_1^c = A strings that do not contain the substring
aa.

Identity Rules for Regular Expressions / Algebraic Laws

- Two regular exp's P and q are equivalent ($P=q$) if P and q represent the same set of strings
- We now give the identities for regular exp, these are useful for simplifying regular exp.

$$I_1 : \phi + r = r$$

$$I_2 : \phi r = r \phi = \phi$$

$$I_3 : \epsilon r = r \epsilon$$

$$I_4 : \epsilon^* = \epsilon \text{ and } \phi^* = \epsilon$$

$$I_5 : r + r = r$$

$$I_6 : r^* r^* = r^*$$

$$I_7 : r r^* = r^* r$$

$$I_8 : (r^*)^* = r^*$$

$$I_9 : \epsilon + r r^* = r^* = \epsilon + r^* r$$

$$I_{10} : (pq)^* p = p(qp)^*$$

$$I_{11} : (p+q)^* = (p^* q^*)^* \\ = (p^* + q^*)^*$$

$$I_{12} : (p+q)r = pr + qr \text{ and}$$

$$r(p+q) = rp + rq$$

The following table shows some examples of reg. exp and language corresponding to these reg. exp.

Regular exp	Meaning
$(a+b)^*$	Set of strings of a's and b's of any length including the null string.

$(a+b)^*abb \rightarrow$ set of strings of a's and b's ending with the string abb

$ab(a+b)^* \rightarrow$ set of strings of a's and b's starting with the string ab .

$(a+b)^*aa(a+b)^* \rightarrow$ set of strings of a's and b's having a sub string aa .

$a^*b^*c^* \rightarrow$ set of strings consisting of any no. of a's (may be empty string also) followed by any no. of b's (may be empty string) followed by any no. of c's (include empty string)

$a^+b^+c^+ \rightarrow$ set of strings consisting of atleast one 'a' followed by string consisting of atleast one 'b' followed by string consisting of atleast one 'c'

$aa^*bb^*cc^* \rightarrow$ A explanation is as above $(a^+b^+c^+)$

$(a+b)^*(a+bb) \rightarrow$ set of strings of a's and b's ending with either a (or) bb

$(aa)^*(bb)^*b \rightarrow$ set of strings consisting of even no. of a's followed by add no. of b's

$(0+1)^*000 \rightarrow$ set of strings of 0's and 1's ending with three consecutive zero's (or) ending with 000

$(11)^* \rightarrow$ set consisting of even no. of 1's

$a^*+b^* \rightarrow$ set of strings of no. of a's (or) b's or included empty strings.

Applications of Regular Expressions

① Regular Expressions in UNIX

There are various UNIX notations used for regular expressions

The range of characters can be specified using square brackets

Eg: $[a-z]$ denotes set of lower case letters.

② Lexical analyzer:

Compilers use this program of lexical analyzer in the process of compilation. The task of lexical analyzer is to scan the i/p program and separate out the tokens.

Eg:- if identifier is a category of token in the source language and it can be identified by regular exp as $(\text{letter})(\text{letter} + \text{digit})^*$

③ Finding patterns in text.

are useful notations used for finding the pattern from given text. A large class of pattern can be identified by regular expressions

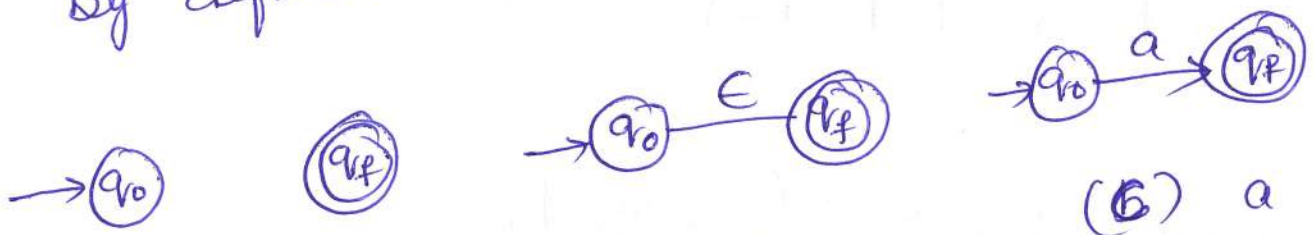
④ GREP utility in unix

In unix there is a GREP utility for searching the designed pattern in the file. For searching the pattern from the Grep makes use of regular expressions. When particular string is present in the file, then using grep we get the line containing that string from file.

Constructing Finite Automata for given regular expression:

Theorem: Let R be a regular expression. Then there exists a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ which accepts $L(R)$

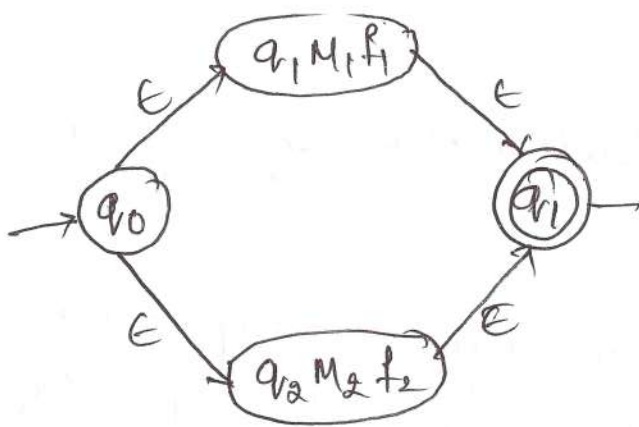
By definition ϕ , ϵ and a are reg. exp



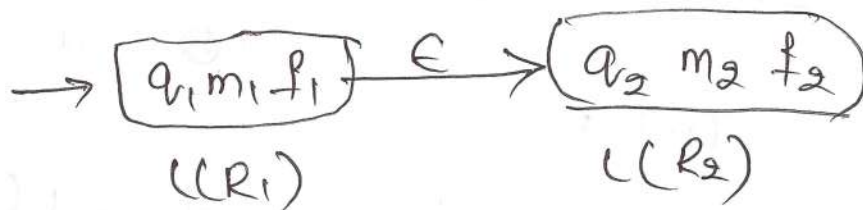
Schematic representation of FA accepting $L(R)$
 a regular expression R to accept language $L(R)$
 where q is the ^{start} state and f is the final state of machine M .



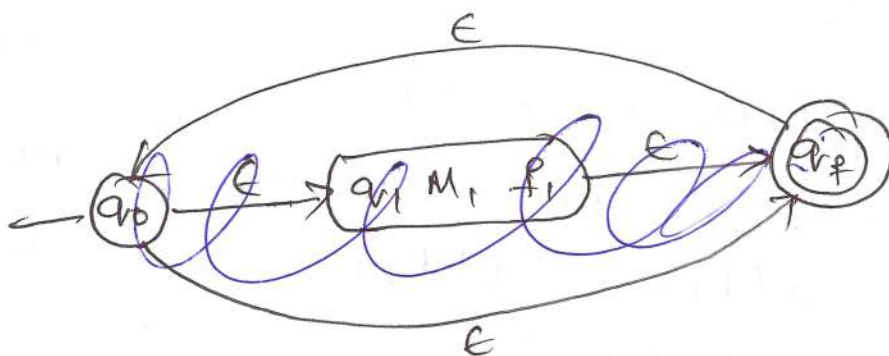
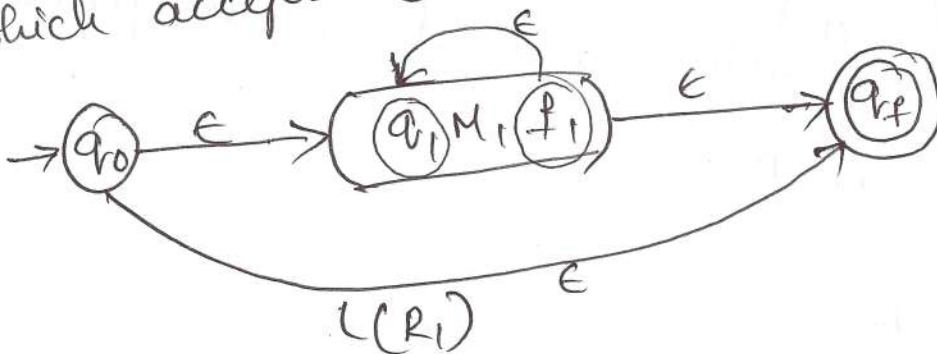
Schematic Representation of FA accepting $L(R)$
 Case 4:- $R = R_1 + R_2$ we can construct an NFA which accepts $L(R_1)$ or $L(R_2)$ which can be represented as $L(R_1 + R_2)$



Case 2: $R = R_1 \cdot R_2$ we can construct an NFA which accepts $L(R_1)$ followed by $L(R_2)$, which can be represented as $L(R_1 \cdot R_2)$



Case 3:- $R = (R_1)^*$ we can construct an NFA which accepts either $L(R_1)^*$



To accept the lang $L(R_1)^*$
 2-(ii)

Example 1:- Construct an equivalent NFA with ϵ transition for the following regular expression

$$r = 0^* + 11$$

Sol Given $r = 0^* + 11$

$$r = r_1^* + r_2 r_2$$

$$r_1 = 0 \quad r_2 = 1$$

$$r = r_1^* + r_3$$

Construct NFA for r.e $r_1 = 0$ is indicated as.

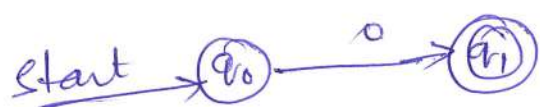


fig: NFA for $r_1 = 0$

To make 0^* we have to consider one new state as starting state and one new state as final state

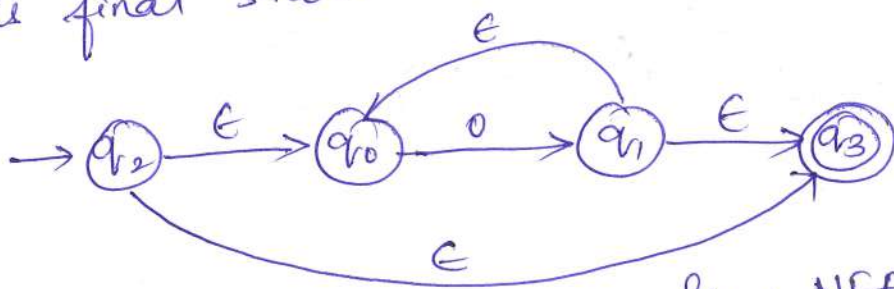


fig: NFA for $r_1 = 0^*$

for $r_3 = r_2 r_2$ ($\because r_2 = 1$)

→ First $r_2 = 1$ r.e indicate as,



NFA for $r_2 = 1$

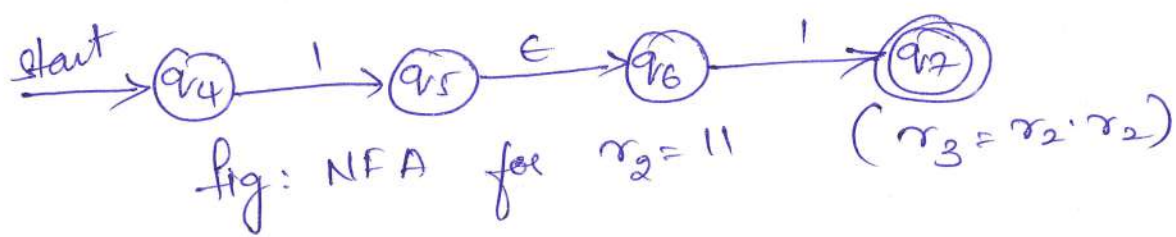
Second $r_2 = 1$ r.e indicate as



NFA for $r_2 = 1$

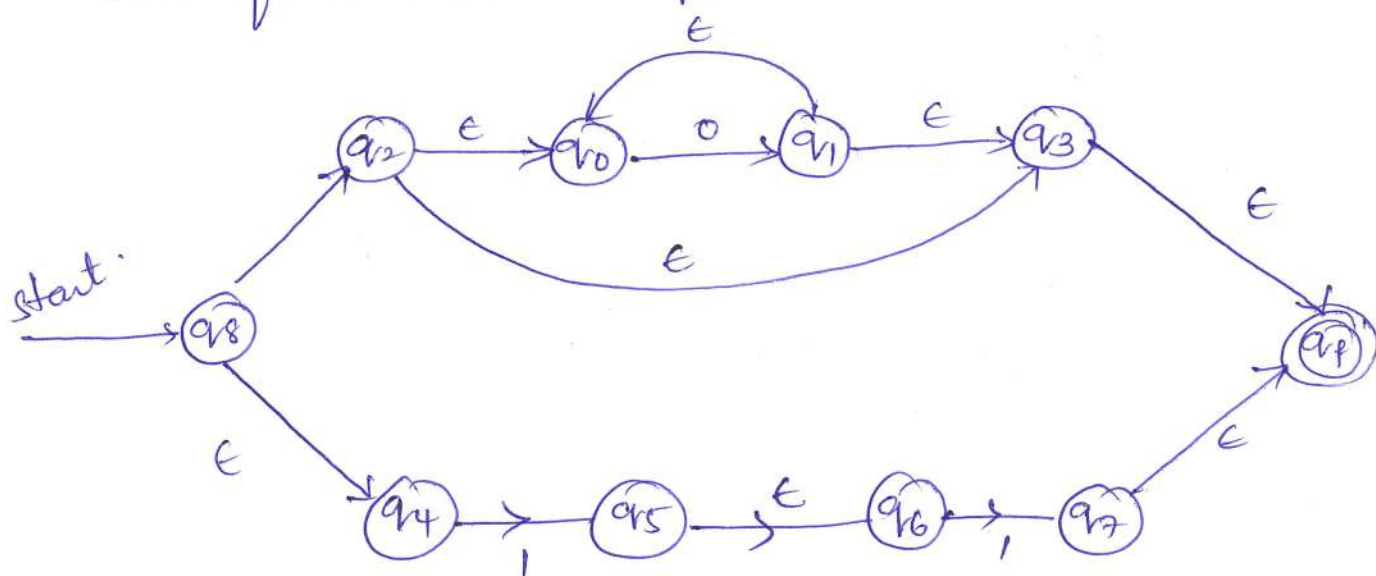
2 - (12)

- To make 1 concatenated with 1



Then $r = r_1 + r_3$

- The above two are added by choosing new starting and final state as follows



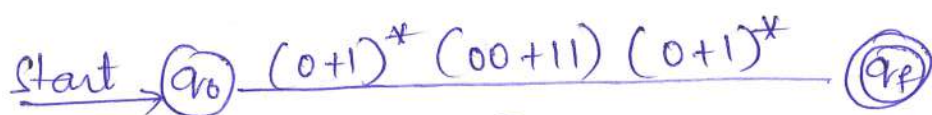
NFA for $r = r_1 + r_3$

Example 2 Convert the following regular expression into equivalent NFA with ϵ -transition $r = (10^*)^*$

Example 3:- Construct finite automata to accept the regular Exp $r = (0+1)^* (00+11) (0+1)^*$

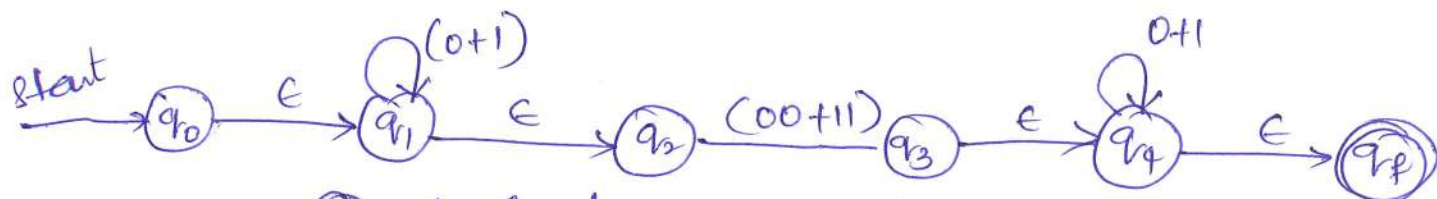
Method 2

$$r = (0+1)^* (00+11) (0+1)^*$$

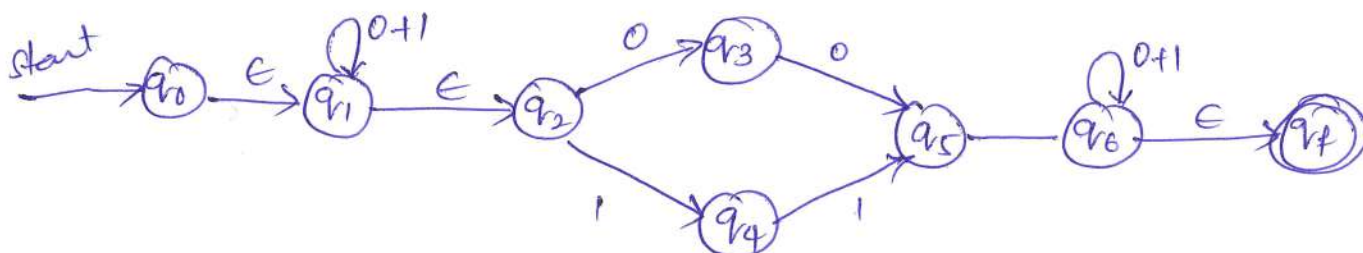




(b) eliminate concatenation



(c) eliminate $\times$ operation



(d) eliminate concatenation and $+$

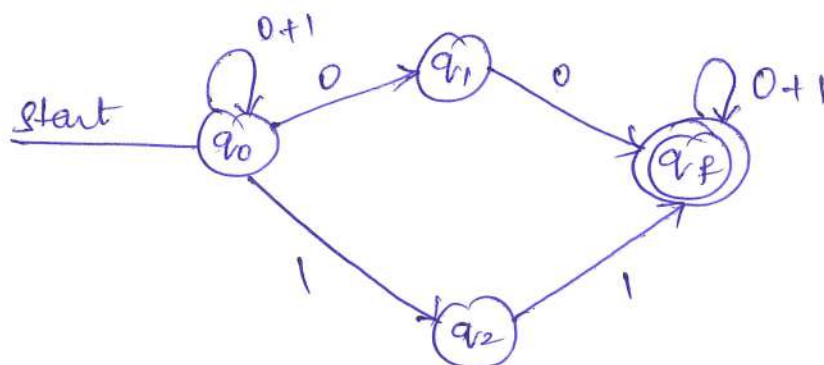


Fig: NFA for $r = (0+1)^* (00+11) (0+1)^*$

Example

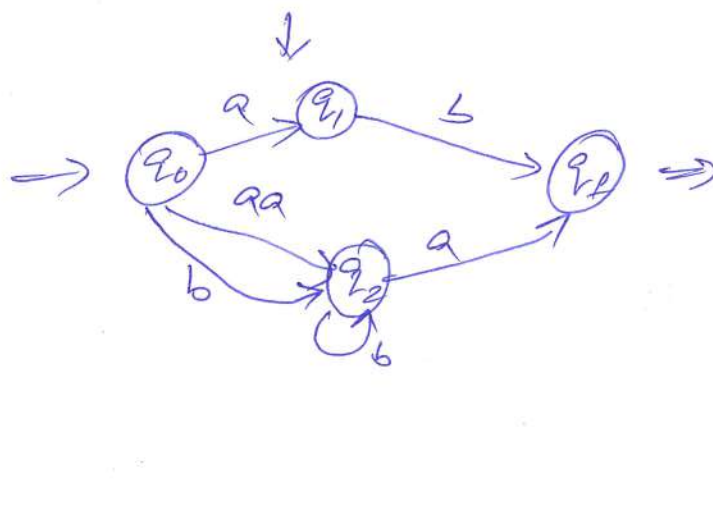
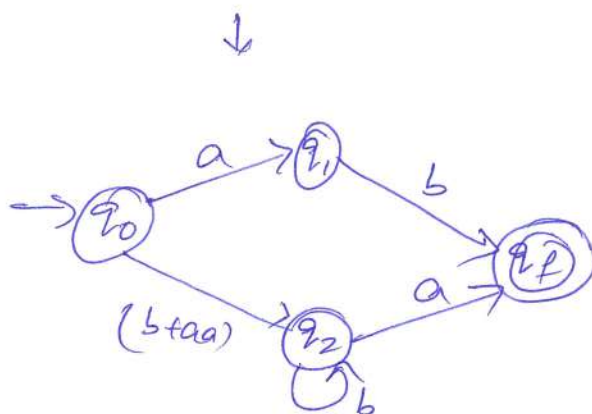
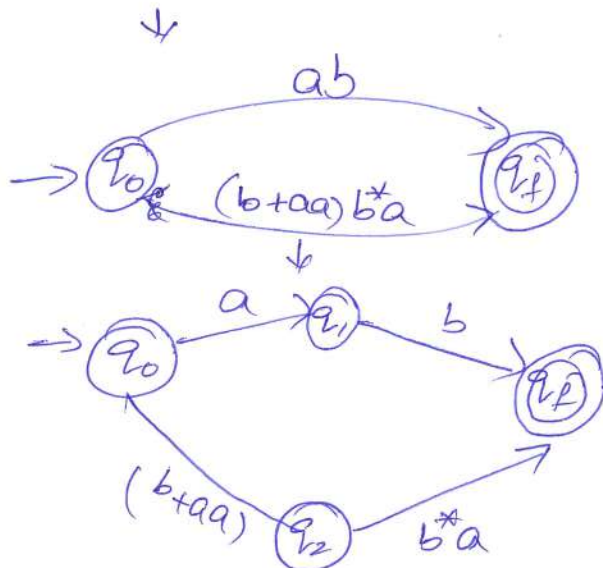
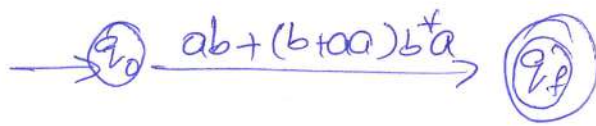
1, Construct FA Equivalent to r.e - $0^* 10^* 10^*$

2, Construct a DFA with reduced states equivalent to Regular Exp $r = 10 + (0+11) 0^* 1$

Regular and Non Regular Languages.

Design a FA from given RE $[ab + (b+aa)b^*a]$

sol:- Design transition diagram for given exp

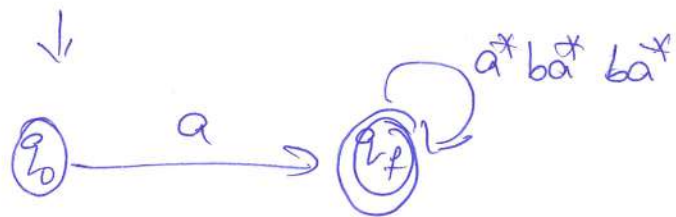
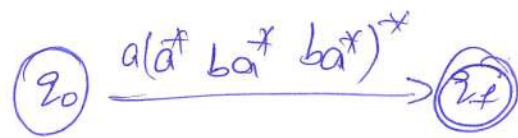


Transition Table for NFA

state	<u>a</u>	<u>b</u>
q_0	$\{q_1, q_2\}$	q_3
q_1	$\emptyset$	q_f
q_2	q_3	$\emptyset$
q_3	q_f	q_3
q_f	$\emptyset$	$\emptyset$

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

Soln



NFA T.T

	a	b
q_0	q_f	$\emptyset$
q_f	q_f	q_2
q_2	q_f	q_2

2- (16)

$$\delta(q_0, a) = [q_1, q_2] \text{ (new state)}$$

$$\delta(q_0, b) = q_3$$

$$\delta(q_2, a) = q_3 \quad \delta(q_2, b) = \emptyset$$

$$\delta(q_3, a) = q_4 \quad \delta(q_3, b) = q_3$$

$$\delta(q_4, a) = \emptyset \quad \delta(q_4, b) = \emptyset$$

$$[q_1, q_2] = \delta(q_1, a) \cup \delta(q_2, a)$$

$$= \emptyset \cup q_3$$

$$= q_3$$

$$\delta(q_1, q_2, b) = \delta(q_1, b) \cup \delta(q_2, b)$$

$$= q_4 \cup \emptyset$$

$$= q_4$$

DFA to Regular Expression

The two popular methods for converting a DFA to its regular expression are

Converting DFA to Reg. Exp

Arderi's method

State Elimination method

State Elimination Method

This method involves the following steps in finding the regular expression for any given

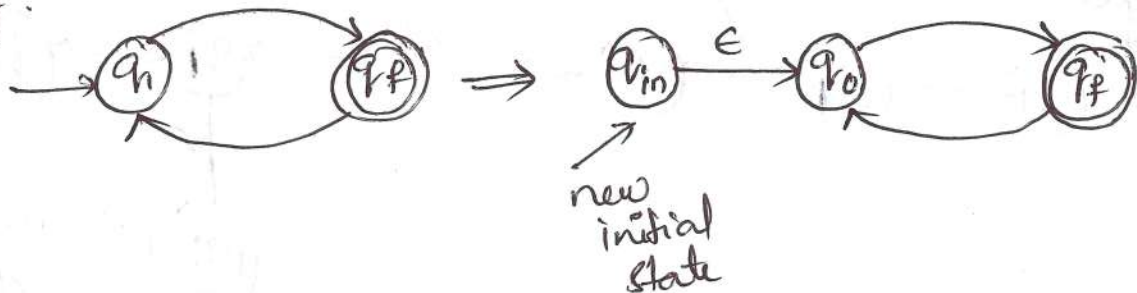
DFA

Step 1:

→ Thumb rule:- The initial state of the DFA must not have any incoming edge.

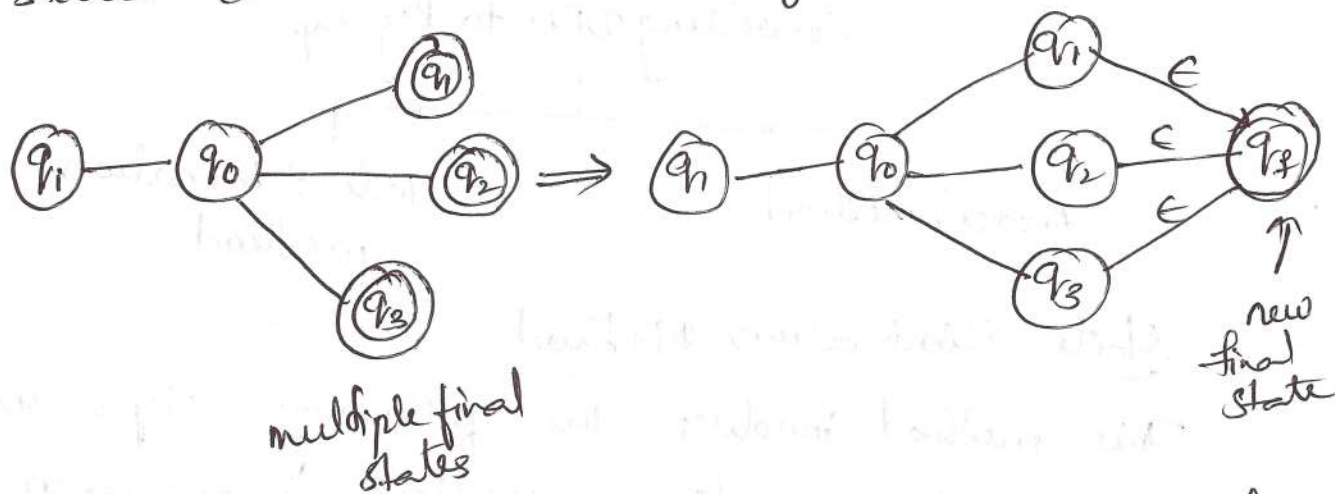
If there exists any incoming edge to the initial state, then create a new initial state ^{or} having no incoming edge to it

Eg:



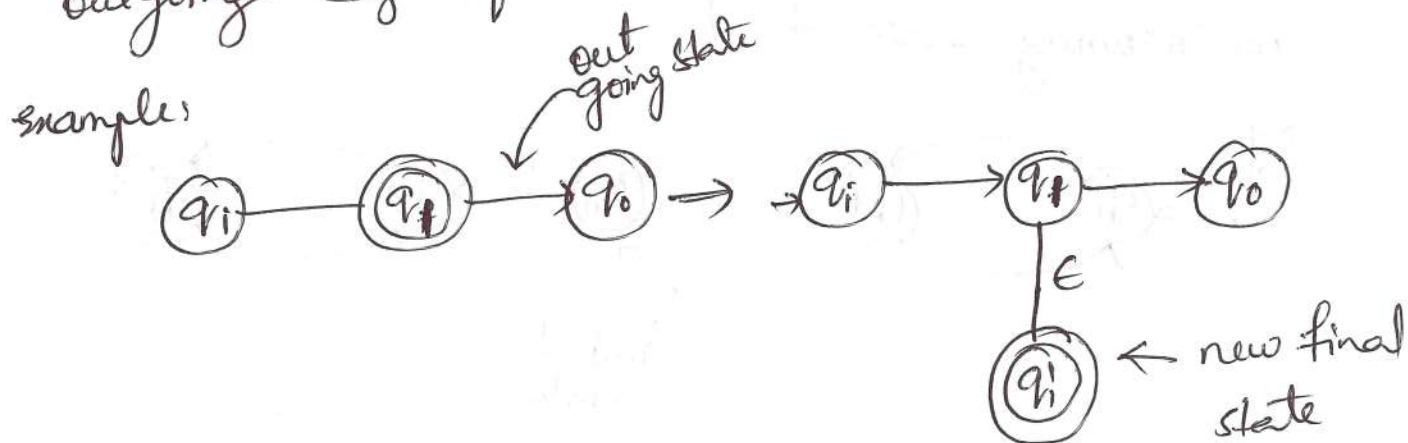
Step 2: There must exist only one final state in the DFA.

if there exists multiple final states in the DFA then convert all the final states into non-final states and create a new single final state



Step 3: The final state of the DFA must not have any outgoing edge.

If there exists any outgoing edge from the final state, then create a new final state having no outgoing edge from it.



Step 4 - eliminate all the intermediate state one by one. These states may be eliminated in any order

In the end

only an initial state going to the final state will be left. The cost of these transitions is the required regular expressions.

Example : Find regular expressions for the following DFA

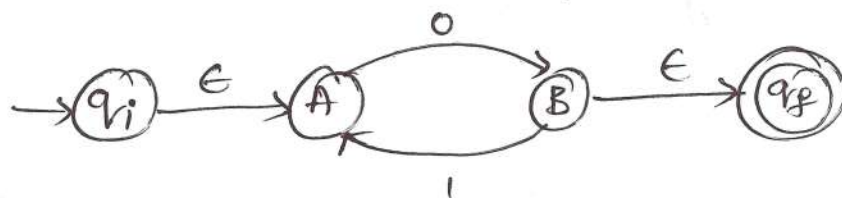


Solⁿ :-

Step 1 :- Initial state A has an incoming edge. So we create a new initial state q_i the resulting DFA is



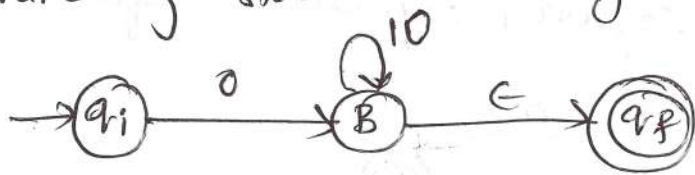
Step 2 :- Final state B has an outgoing edge so we create a new final state q_f



Step 3:-

Now start eliminating intermediate states
first let us eliminate state A

There is a path going from state q_i to state B via state A. So, after eliminating state A, we put a direct path from state q_i to state B having cost $\epsilon \cdot 0 = 0$. There is a loop on state B using state A. So, after eliminating state A, we put a direct loop on state B having cost $1 \cdot 0 = 10$.
eliminating state A we get

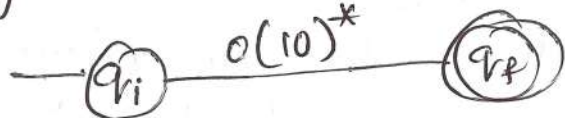


Step: 4

Now eliminate state B

There is a path going from state q_i to state q_f via state B. So after eliminating state B, we put a direct path from state q_i to state q_f having cost $0 \cdot (10)^* \epsilon = 0(10)^*$.

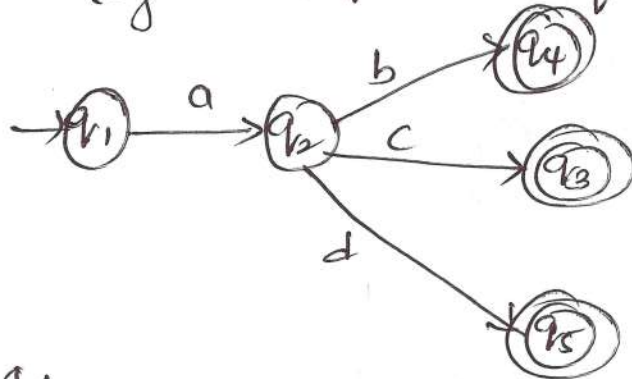
eliminating state B we get



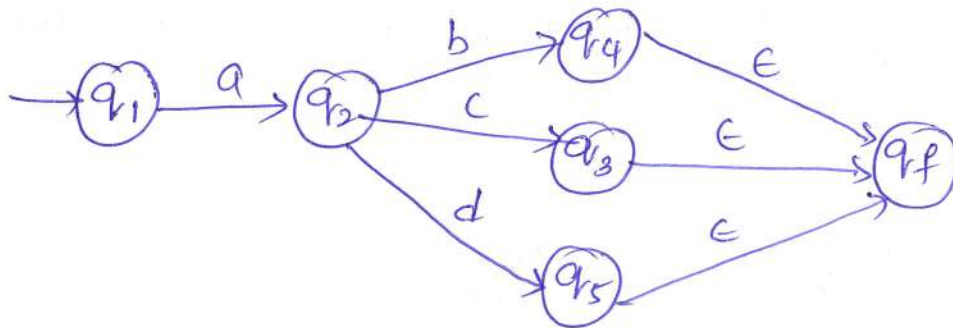
from here $\rightarrow$ Regular Expression $= 0(10)^*$

Example 2

Find regular expression for the following DFA

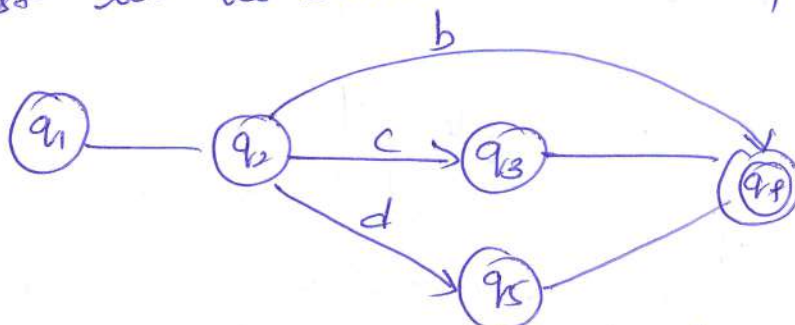


Step 1: There exist multiple final states, so we convert them into a single final state



Step 2 :- Now we start eliminating the intermediate states

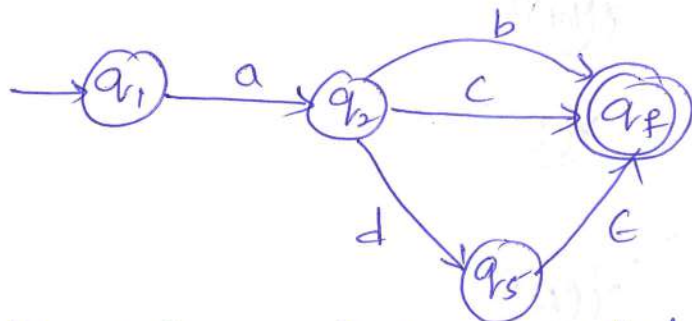
first let us eliminate state q_4



Steps :- There is a path going from state q_2 to state q_f via state q_4 . So after eliminating state q_4 we put a direct path from state q_2 to q_f having $b \cdot \epsilon = b$

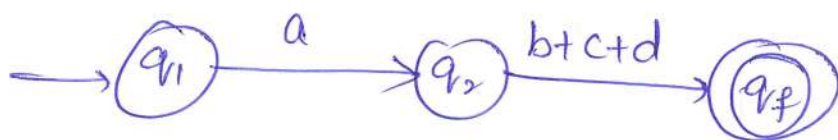
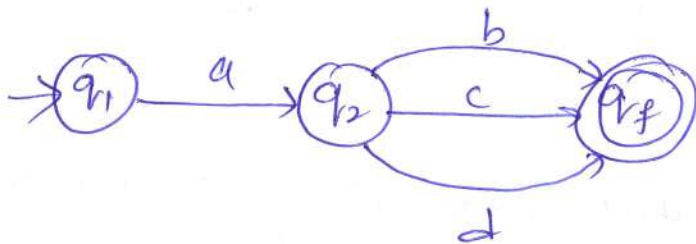
Step-3 let us eliminate state q_3

There is a path going from state q_2 to q_f via state q_3 . So after eliminating state q_3 , we put a direct path from state q_2 to state q_f having cost $c \cdot e = c$



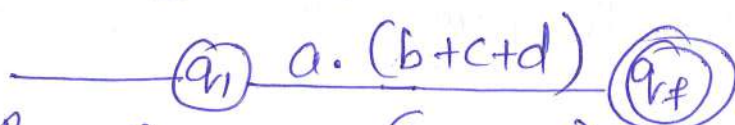
Step 4 let us eliminate state q_5

There is a path going from state q_2 to state q_f via state q_5 . So after eliminating state q_5 , we put a direct path from start q_2 to state q_f having cost $d \cdot e = d$



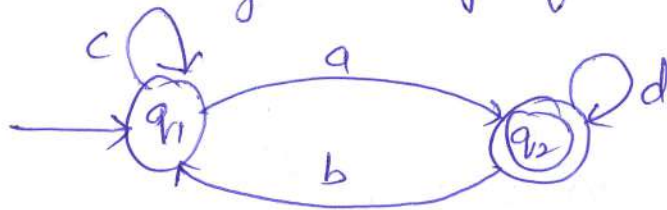
Step-5 : Now let us eliminate state q_2

There is a path going from state q_1 to state q_f via state q_2 . So after eliminating state q_2 we put a direct path from state q_1 to state q_f having cost $a \cdot (b+c+d)$

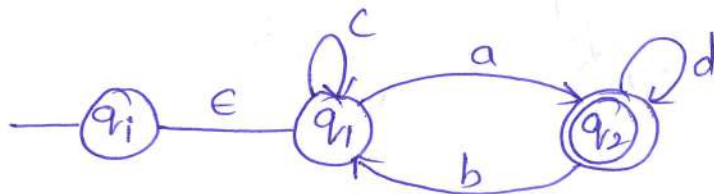


$\therefore$ Reg. Exp = $a \cdot (b+c+d)$

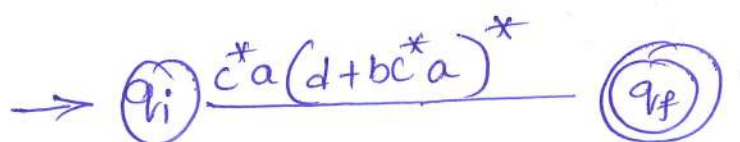
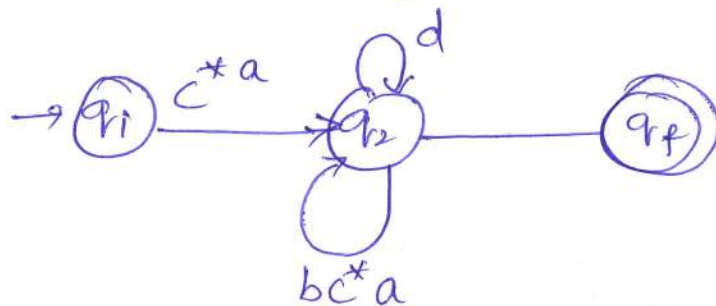
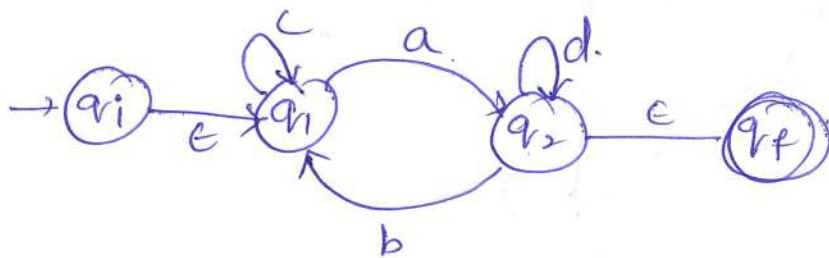
3 Find regular exp for DFA given



Sol

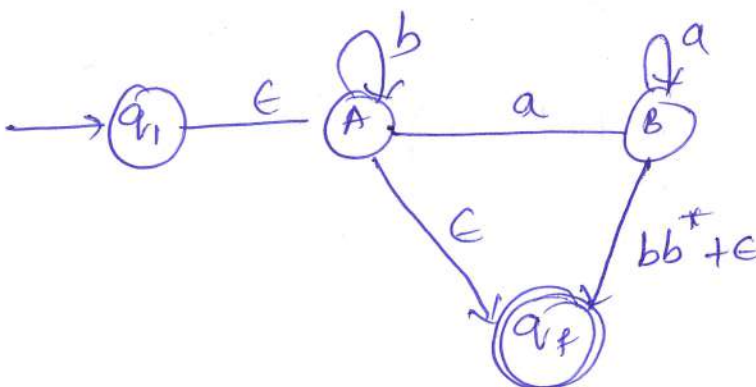
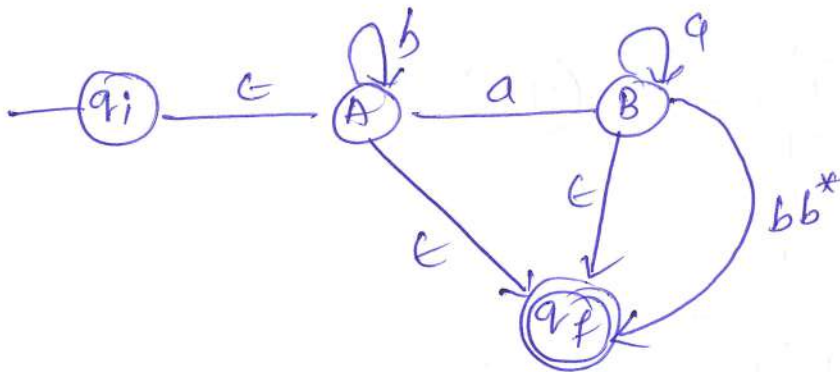
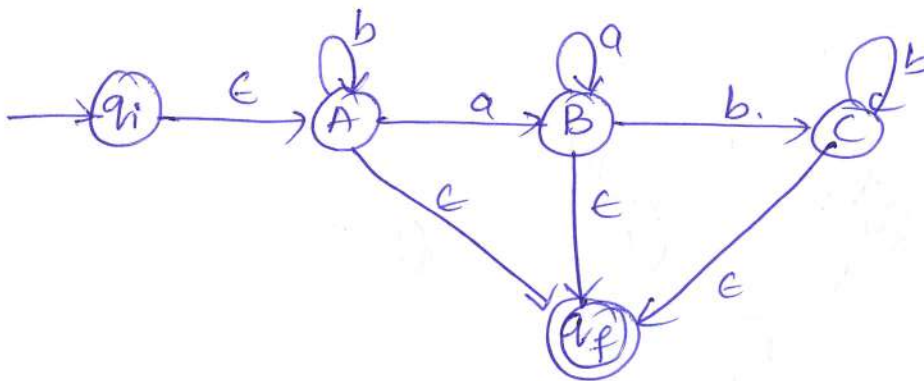
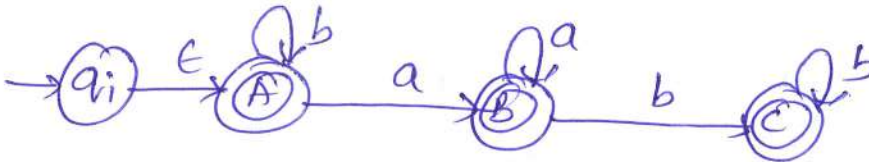
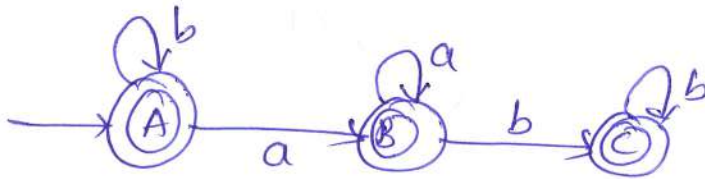
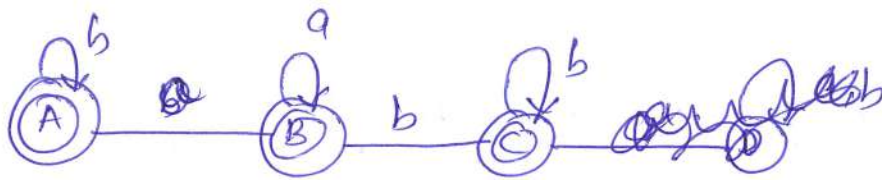


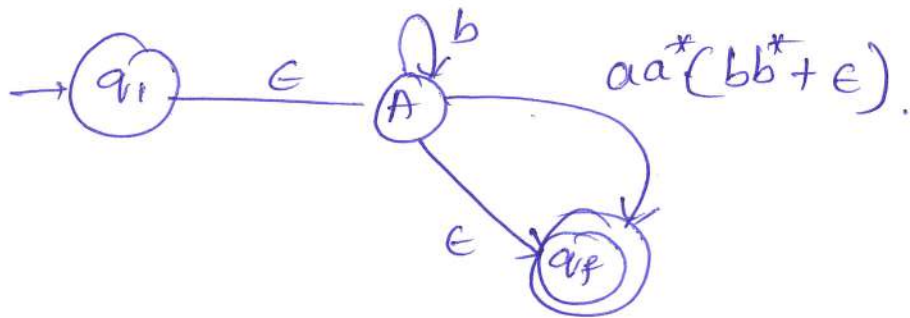
⇓



$$\therefore \text{Reg exp} = c^* a (d + bc^* a)^*$$

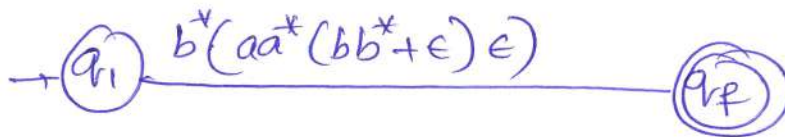
Solve





$$\Rightarrow \epsilon \cdot b^* \cdot (aa^*(bb^*+e)+e)$$

$$= b^*(aa^*(bb^*+e)+e)$$



ARDEN'S Theorem

→ If P and Q are two R.E over Σ , and if P doesn't contain ϵ , then the following equation in R given by $R = Q + RP$ has a unique solⁿ i.e. $R = QP^*$

$$R = Q + RP \quad \text{--- (1)}$$

Replace $R = QP^*$ in Eq (1)

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

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

$\boxed{\because \epsilon + R^*R = R^*}$ are Identity

$$R = QP^*$$

$\therefore$ proved $R = QP^*$

Consider $R = Q + RP$

replace $R = Q + RP$ in (1).

$$= Q + (Q + RP)P$$

$$= Q + [QP + RP^2] \quad \text{replace } R = Q + RP$$

$$= Q + [QP + [Q + RP]P^2]$$

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

$$= Q + QP + QP^2 + \dots + QP^n + RP^{n+1}$$

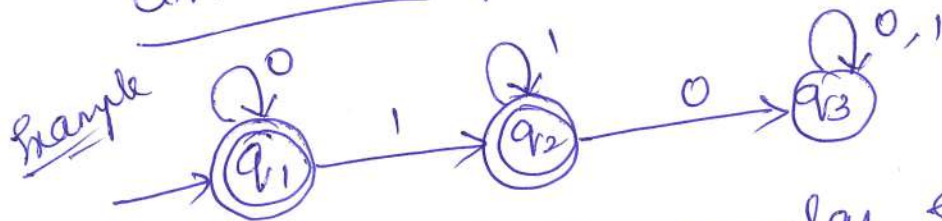
$$\therefore \text{let } R = QP^*$$

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

$$= Q \left[\epsilon + P + P^2 + \dots + P^n + P^*P^{n+1} \right]$$

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

Conversion of F.A to Reg. Exp.



Sol let us build the regular expression for each state

$$q_1 = q_1 0 + \epsilon \quad \text{--- (1)}$$

$$q_2 = q_1 1 + q_2 1 \quad \text{--- (2)}$$

$$q_3 = q_2 0 + q_3 0 + q_3 1 \quad \text{--- (3)}$$

$$= q_2 0 + q_3 (0+1)$$

Since q_1 and q_2 are final states let us solve q_1

$$q_1 = \epsilon + q_1 0$$

which is $R = Q + RP$ equivalent so we can write QP^*

$$q_1 = \epsilon \cdot (0)^* = 0^*$$

$$\therefore R = Q + RP \Rightarrow QP^*$$

Substitute q_1 in ~~Eq 1~~ Eq 2

$$q_2 = 0^* 1 + q_2 1$$

$\downarrow \quad \downarrow \quad \downarrow$
 $Q \quad R \quad P$

$$\therefore R = Q + RP \Rightarrow QP^*$$

$$q_2 = 0^* 1 (1)^*$$

The regular Expression is given by

$$r = q_1 + q_2$$

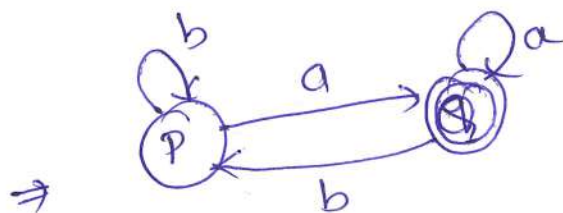
$$= 0^* + 0^* 1 \cdot 1^*$$

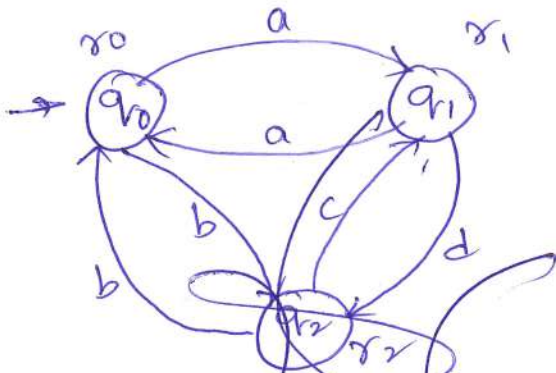
$$\therefore \cancel{0^*} 1 \cdot 1^* = 1^+$$

$$r = 0^* + 0^* 1^+$$

Example Write a Regular Expression for the following DFA

	a	b
$\rightarrow P$	Q	P
Q	Q	P





⇒ Arden's theorem we will write down 1Q equations

$$r_0 \rightarrow q_0, \quad r_1 = q_1, \quad r_2 = q_2$$

$$r_0 = \epsilon + r_1 a + r_2 b$$

$$r_1 = r_0 a + r_2 c$$

$$r_2 = r_0 b + r_1 d$$

So let us write the equations

$$P = Pb + Qb + \epsilon \quad \text{--- (1)}$$

$$Q = Pa + Qa \quad \text{--- (2)}$$

Since q is the final state

consider eq (1).

$$P = \underset{\substack{\downarrow \\ R}}{P}b + \underset{\substack{\downarrow \\ R}}{Q}\underset{\substack{\downarrow \\ P}}{b} + \underbrace{\epsilon}_Q$$

$$\therefore R = Q + RP \quad QP^*$$

$$= (Qb + \epsilon)b^*$$

$$= Qbb^* + \epsilon \cdot b^*$$

$$= Qbb^* + b^*$$

$$P = Qbb^* + b^* \quad \text{--- (3)}$$

Substitute P in (2)

$$Q = \underbrace{(Qbb^* + b^*)}_Q a + \underset{\substack{\downarrow \\ R}}{Q}\underset{\substack{\downarrow \\ P}}{a} \quad \therefore R = Q + RP \quad QP^*$$

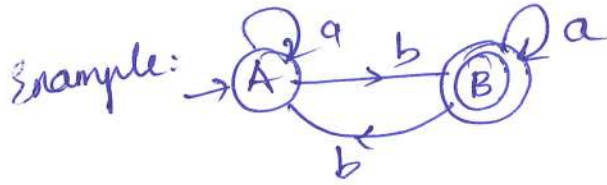
$$= (Qbb^*a + b^*a) + Qa$$

$$= \underbrace{Qbb^*a + b^*a + Qa}$$

$$\underset{\substack{\downarrow \\ R}}{Q} = \underset{\substack{\downarrow \\ R}}{Q} \underbrace{(bb^*a + a)}_P + \underbrace{b^*a}_Q$$

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

acc to ardent.
 $\therefore QP^*$



$$A = E + Aa + Bb.$$

$$B = Ab + Ba \quad \text{--- (2)}$$

$$R = Q + RP$$

$$R = QP^*$$

Consider (2)

$$\underset{R}{B} = \underset{Q}{Ab} + \underset{R}{B}\underset{P}{a} \Rightarrow B = \cancel{Ab}a^* \quad \text{--- (3)}$$

Substitute (3) in (1).

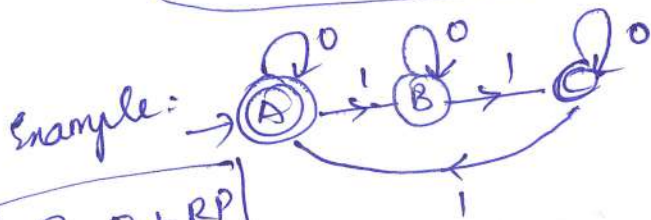
$$A = E + Aa + A\cancel{b}a^*b$$

$$\underset{R'}{A} = E + \underset{Q}{A}\underset{R}{a} + \underset{P}{A}ba^*b$$

$$A = E \cdot (a + ba^*b)^* \quad \text{--- (4)}$$

final state is B so. substitute A in B. (4) into (2)

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



$$R = Q + RP$$

$$= QP^*$$

$$A = E + A0 + C1 \quad \text{--- (1)}$$

$$B = A1 + B0 \quad \text{--- (2)}$$

$$C = B1 + C0 \quad \text{--- (3)}$$

observe that eq (3) is in form of $Q + RP$.

$$\underset{R'}{C} = \underset{Q}{B}1 + \underset{R}{C}0 \Rightarrow B10^* \quad \text{--- (4)}$$

$$\underset{R'}{B} = \underset{Q}{A}1 + \underset{R}{B}0 \Rightarrow A10^* \quad \text{--- (5)}$$

Substitute (4) in (5).

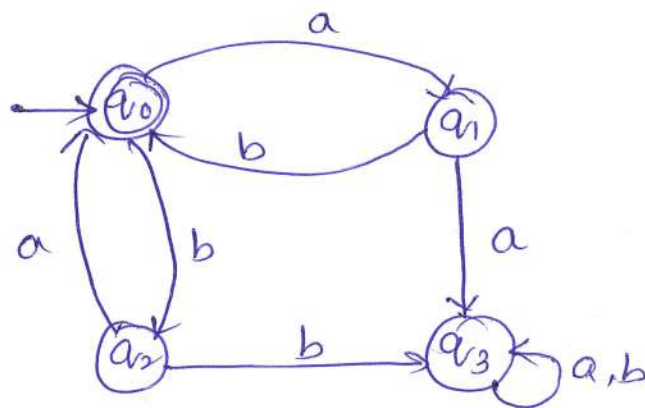
$$C = A10^*10^* \quad \text{now substitute C in (1)}$$

$$A = E + A0 + A10^*10^*1$$

$$\underset{R'}{A} = E + \underset{Q}{A}\underset{R}{0} + \underset{P}{A}(0 + 10^*10^*1) \Rightarrow E \cdot (0 + 10^*10^*1)^*$$

$$\Rightarrow 0 + 10^*10^*1$$

Example 3



Sol

write equations for ~~eq~~ each state

$$q_0 = q_1b + q_2a + \epsilon \quad \text{--- (1)}$$

$$q_1 = q_0a \quad \text{--- (2)}$$

$$q_2 = q_0b \quad \text{--- (3)}$$

$$q_3 = q_1a + q_2b + q_3(a+b)$$

$$\text{(or)} \\ = q_1a + q_2b + q_3a + q_3b \quad \text{--- (4)}$$

Now q_0 is the final state and it contains values of q_1 and q_2 only we will put equations 2 and 3 in ①

$$q_0 = q_0a \cdot b + q_0b \cdot a + \epsilon \\ = q_0ab + q_0ba + \epsilon$$

$$= q_0 \underbrace{(ab+ba)}_P + \underbrace{\epsilon}_Q$$

by arden's theorem $R = RP + Q \Rightarrow QP^*$

$$R = \epsilon \cdot (ab+ba)^* \Rightarrow (ab+ba)^*$$

Q1 let us write the equations

$$P = P_b + Q_b + \epsilon \quad \text{--- (1)}$$

$$Q = P_a + Q_b \quad \text{--- (2)}$$

Since Q is the final state it will give regular expression.

Consider eq (1).

$$P = P_b + \underbrace{Q_b + \epsilon}_Q$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$R \quad P \quad Q$$

according to arden's theorem
 $R = Q + RP$ then $R = QP^*$

$$P = (Qb + \epsilon)b^*$$

$$= Qbb^* + \epsilon \cdot b^*$$

$$P = Qbb^* + b^* \quad \text{--- (3)}$$

Substitute (3) in (2).

~~$$Q = (Qbb^*)a + Qb$$~~

$$Q = (Qbb^* + b^*)a + Qb$$

$$Q = Qbb^*a + b^*a + Qb$$

$$= Qb + Qbb^*a + b^*a$$

$$Q = \underbrace{Q}_{R}(\underbrace{b}_{P} + \underbrace{bb^*a}_{P}) + \underbrace{b^*a}_{Q} \Rightarrow \underbrace{b^*a}_{(Q)} + \underbrace{b^*ab^*}_{(Q)} + \underbrace{b^*ab^*ba^*}_{(Q)}$$

2- (30)

$$R \in b^*ab^* + b^*a(bb^*a)^*$$

Regular and Non-Regular Languages

- The languages accepted by finite automata are described by regular expressions. So to prove a language is accepted by finite automata it is sufficient to prove the regular expression of that language is accepted by finite automata.

→ The languages which are accepted by some finite automata are called "regular languages." Here it means that the FA accepts only the word of this language and doesn't accept any word outside it.

1) Some of the words of the languages are not accepted * by FA
(ol)

2) All the words of the language are accepted in the addition to that some extra strings are also accepted

* All languages are either regular or non regular none of the language are both.

- By looking at some of the lang^s we can say whether they are regular or not.

i. The languages whose words need some sort of comparison can be never regular

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

Here the no. of a's must be equal to no. of b's
for each 'a' we check the existence of b.
which cannot be done using FA.

The lang's whose words are in arithmetic progression
and need no comparison will be regular.

Example (1) $L = \{a^{2n}, n \geq 1\}$

The words of this language are, aa, aaaa,
aaaaaa, ..., a^{2n} . which are in arithmetic Hence
it is a regular language progression with period 2.
Hence it is a regular language.

Example (2) $L = \{a^p, p \text{ is prime}\}$.

The words of this language are
 $\{a, aa, aaaa, \dots, a^p\}$

- we can see these word are not in arithmetic
progression Hence it is not regular.

Unit - 2 Part - 2

Pumping lemma for Regular languages.

→ Pumping lemma is used to prove that a language is not regular.

→ Regular languages are the languages that can be represented or designed using a FSM. All regular languages are designed by FSM only.

→ The languages which cannot be designed using FSM those languages are called as non regular languages.

→ A pumping lemma is used to prove that a language is not regular.

* It cannot be used to prove that a language is regular.

Statement for pumping lemma :-

If A is a regular language then A has a pumping length ' p ' such that any string ' s ' where $|s| \geq p$ may be divided into 3 parts

$s = xyz$ such that the following conditions must be true

$|s| \rightarrow$ length of string s

(1) $xy^iz \in A$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq p$

2 - (34)

→ increasing γ should $\in$ to A is condition 1.

To prove that a language is not regular using pumping lemma, follow the below steps:-

(We prove using contradiction)

→ Assume that A is Regular.

- It has to have a pumping length (say P)
- All strings longer than P can be pumped $|s| \geq P$
- Now find a string ' s ' in A such that $|s| \geq P$
- Divide s to $x y z$
- show that $x y^i z \notin A$ for some i
- Then consider all ways that s can be divided into $x y z$.
- show that none of these can satisfy all the 3 pumping lemma conditions at the same time
- s cannot be pumped == Contradiction.

Example 1 :

using pumping lemma prove that the language

$A = \{ a^n b^n \mid n \geq 0 \}$ is not regular

proof :-

Assume that A is regular

let pumping length = $P = 7$

$s = a^P b^P \Rightarrow s = aaaaaaa bbbbbb$
 $\swarrow \quad \downarrow \quad \searrow$
 $x \quad y \quad z$

Case 1: The y is in the 'a' part

$\underbrace{aaaaaa}_x \underbrace{a}_y \underbrace{bbbbbb}_z$

Case 2: The y is in the 'b' part

$\underbrace{aaaaaa}_x \underbrace{bbb}_y \underbrace{bbb}_z$

Case 3: The y is in the 'a' and 'b' part

$\underbrace{aaaaa}_x \underbrace{aa}_y \underbrace{bbb}_z$

$x y^i z \Rightarrow x y^2 z$

for case 1,

aa aaaa aaaa abbbbbbb

no. of a = 11

$11 \neq 7$

no. of b = 7

for case 2.

$$xy^iz \rightarrow xy^2z$$

aaaaaaabbb bbbb bbbb b.

$$\text{no. of } a = 7 \quad \text{no. of } b's = 11$$
$$7 \neq 11$$

for case 3.

$$xy^iz = xy^2z.$$

aaaaa aabb aabb bbbbbb.

$$\text{no. of } a = 9 \quad \text{no. of } b's = 9$$

$a = b$ but it is not following condition

$$a^n b^n.$$

check length of $|xy| \leq p$ for each case.

Case 1 $6 \leq 7$

Case 2. $9 \leq 7$ (failed)

Case 3. $9 \leq 7$ (failed)

Hence A is not regular.

Example 2:-

using pumping lemma prove that the language

$A = \{xyx \mid x \in \{0,1\}^*\}$ is not regular.

Proof

01|01
4 4

- Assume A is regular then it must have a pumping length

let pumping length = p

let the string $s = 0^p 1 0^p$

let $p = 5$

Case (i),
 $s = \underbrace{00000}_x \underbrace{1}_y \underbrace{00000}_z$

$$xy^2z = xy^2z$$

$$s = 0 \underbrace{0000 \ 0000}_{\neq} \underbrace{100000}_1$$

$$\therefore s \notin A$$

~~Case (ii)~~ $|y| > 0$ (True)

$$|xy| \leq p$$

$$\Rightarrow 5 \leq p \Rightarrow 5 \leq 5$$

$\therefore$'s' cannot be pumped which is a contradiction

to our assumption

Hence proved that $A = \{yy \mid y \in \{0,1\}^+\}$ is not a regular.

Theoretically:-

Given $A = \{yy \mid y \in \{0,1\}^+\}$

i.e. y can be any combination of 0 or 1 - set.

$\rightarrow \frac{y}{1} \frac{y}{2} \rightarrow$ should be same the FSM need to store (1) in the memory to get next half.

But FSM doesn't have much memory to store data i.e. limited memory.
Eg: $\frac{01}{4} \frac{01}{4}$ and hence FSM can't store this and hence this lang can't be designed using FSM. If it is not able to design then it is not regular.

Applications of pumping lemma

→ pumping lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

→ If L is regular, it satisfies pumping lemma

→ if L doesn't satisfy pumping lemma, it's not regular

UNIT - 2 part - 5

* Closure properties of Regular Languages :-

→ A set is closed under an operation ~~if~~ ^{by} applying that operation to any members of the set always yields a member of the set.

Eg: the ⁺integers are closed under addition and multiplication but not division.

Language properties: Decision properties & closure properties
Regular languages are closed under union, intersection, concatenation

→ If both R_1 and R_2 are regular languages then any operation ~~we~~ ^{if} the operation is done then the resultant language is also Regular, then only that it is called as closed under that operation

1 Union - If L_1 and L_2 are two regular languages then their union $L_1 \cup L_2$ will also be regular

Example:

$$L_1 = \{an \mid n > 0\} \text{ and } L_2 = \{bn \mid n > 0\}$$

$$L_3 = L_1 \cup L_2 = \{an \cup bn \mid n > 0\} \text{ is also regular}$$

2 Intersection: If L_1 and L_2 are two regular languages then their intersection $L_1 \cap L_2$ will also be regular

Example

$$L_1 = \{am \, bn \mid n > 0 \text{ and } m > 0\} \text{ and}$$

$$L_2 = \{am \, bn \cup bn \, am \mid n > 0 \text{ and } m > 0\}$$

$L_3 = L_1 \cap L_2 = \{a^m b^n \mid n > 0 \text{ and } m > 0\}$ are also regular.

3 Concatenation

If L_1 and L_2 are two regular languages, their concatenation $L_1 \cdot L_2$ will also be regular.

Example: -

$$L_1 = \{a^m \mid m > 0\} \text{ and } L_2 = \{b^n \mid n > 0\}$$

$L_3 = L_1 \cdot L_2 = \{a^m \cdot b^n \mid m > 0 \text{ and } n > 0\}$ is also regular

4 Kleen Closure :-

If L_1 is a regular language, its kleene closure L_1^* will also be regular

Example :- $L_1 = (a \cup b)$

$$L_1^* = (a \cup b)^*$$

5 Complement :

If $L(G)$ is a regular language its complement $L'(G)$ will also be regular. Complement of a language can be found by subtracting strings which are in $L(G)$ from all possible strings

Example :

$$L(G) = \{a^n \mid n > 3\} \quad L'(G) = \{a^n \mid n \leq 3\}$$

6. Homomorphism :

A homomorphism on an alphabet is a function that gives a string for each symbol in that alphabet.

If L is a regular language and h is a homomorphism on its alphabet, then $h(L) = \{h(w) \mid w \text{ is in } L\}$ is also a regular language.

7. Inverse Homomorphism :

Let h be a homomorphism and L a language whose alphabet is the output language of h . $h^{-1}(L) = \{w \mid h(w) \text{ is in } L\}$

Decision properties.

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

Eg: Is given language L empty.

Problem types.

Decidable problems: We can always construct a corresponding algorithm, that ^{can} answer the problem correctly i.e. it may 'Yes' or 'no' Eg: prime or not

Undecidable Problem: The problems for which we can't construct an algorithm that can answer the problem correctly in finite time are termed as undecidable problems

Eg: When a CFG generates all the strings or not

Semi Decidable Problem: Eg: Turing Recognisable problem

Decision properties

1. Emptiness and Non Emptiness
2. Finiteness and Infiniteness
3. Membership
- ④ Equivalence

(1) Emptiness and Non Emptiness

- To check whether a given FA accepts empty language or non-empty language.

Steps to check emptiness

1. Select the state that cannot be reached from the initial states & delete them (remove unreachable states)

2. If the resulting machine contains at least one final state, so then the finite automata accepts the non-empty languages.

3. If the resulting machine is free final state then finite automata accepts empty languages

Finiteness and Infiniteness

- It states that the language accepted by a given FA is finite or not

Steps

1. Select the state that cannot be reached from the initial state & delete them (remove unreachable state)

2. Select the state from which we cannot reach the final state & delete them (remove dead states)
3. If the resulting machine contains loops or cycles then the finite automata accepts infinite language
4. If the resulting machine do not contains loops or cycles then the finite automata accepts ~~in~~finite language.

Membership

is a property to verify an arbitrary string is accepted by a finite automata or not

- It is decidable, because by reading the string if it reaches the final state, then it is a member. otherwise, if the string halts in a non-final state then it is said to be not a member of FA.

Equivalence:

- Two finite automata over S are equivalent, if they accept the same set of string over S .
- FA over S are not equivalent if for some string w over S one automation reaches a final state and other automation reaches a non final state

Procedure to Test the equivalence of two FSM over Σ .

Let M and M' be two FSM over Σ

Step 1 :- Construct comparison table consisting of $n+1$ columns where 'n' is the no. of i/p symbols.

- The first column consists of pairs of vertices of the (q, q') where $q \in M$ and $q' \in M'$

→ If (q, q') appears in some row of the first column then the corresponding entry in the a -column ($a \in \Sigma$) is (q_a, q'_a) where $\delta(q, a) = q_a$

$$\delta'(q', a) = q'_a$$

Step 2 : Comparison table is constructed by starting with the pair of initial vertices q_0, q'_0 of M and M' in first column.

→ The first element in the subsequent columns are (q_n, q'_n) where $\delta(q_n, a) = q_n^*$ $\delta'(q'_n, a) = q'_n$

Step 3 :- Repeat the construction by considering the pairs in the second and subsequent columns which are not in the first column.

- The row wise construction is repeated. There are two cases.

Case 1: If we reach a pair (q, q') such that q is a final state of M and q' is a non final state of M' and viceversa, we terminate the construction and conclude the M and M' are not equivalent

Case 2: Here the construction of table is terminated when no new element appears in the second and subsequent columns which are not in the first column.

In such a case we conclude that M and M' are equivalent

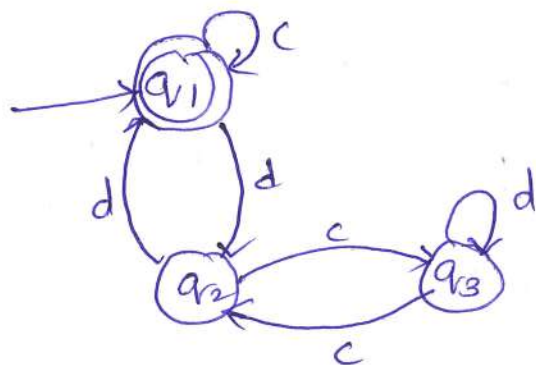
Simple steps to identify Equivalence.

1) For any pair of states (q_i, q_j) the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta(q_i, a) = q_a$ and $\delta(q_j, a) = q_b$

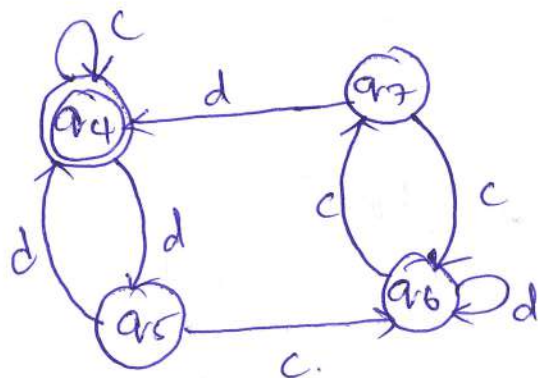
The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is intermediate state and the other is final state

2) If initial state is final state of one automaton, then in second automata also initial state must be final state for them to be equivalent.

Example Consider the following two DFA's M and M' over $\{c, d\}$



(A) Automaton for M



(B) Automata for M'

Sol
 Step 1: The initial states in M and M' are q_1 and q_4 respectively, Hence the first element of the first column in the comparison table must be (q_1, q_4) .

states	c	d
$\rightarrow (q_1, q_4)$	(q_1, q_4) $\downarrow$ $F_S \quad F_S$	(q_2, q_5) $\downarrow \quad \downarrow$ $I_S \quad I_S$
(q_2, q_5)	(q_3, q_6) $\downarrow \quad \downarrow$ $I_S \quad I_S$	(q_1, q_4) $\downarrow \quad \downarrow$ $F_S \quad F_S$
(q_3, q_6)	(q_2, q_7) $\downarrow \quad \downarrow$ $I_S \quad I_S$	(q_3, q_6) $\downarrow \quad \downarrow$ $I_S \quad I_S$
(q_2, q_7)	(q_3, q_6)	(q_1, q_4)

2-98

$= q_7$ is left over state if we observe q_7 is in combination with q_2 so we consider (q_2, q_7)

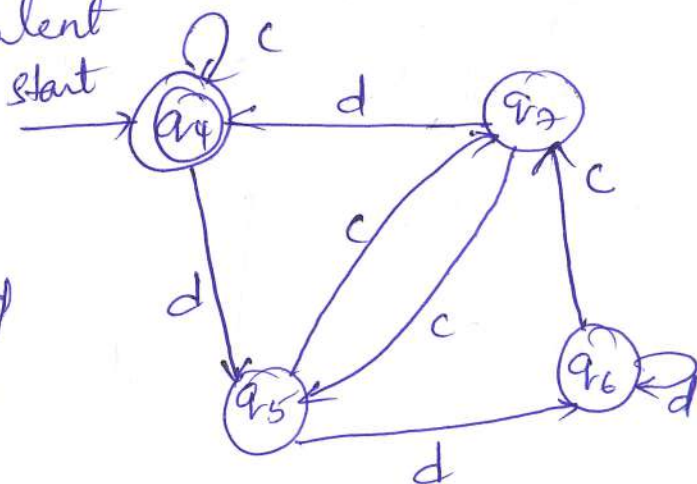
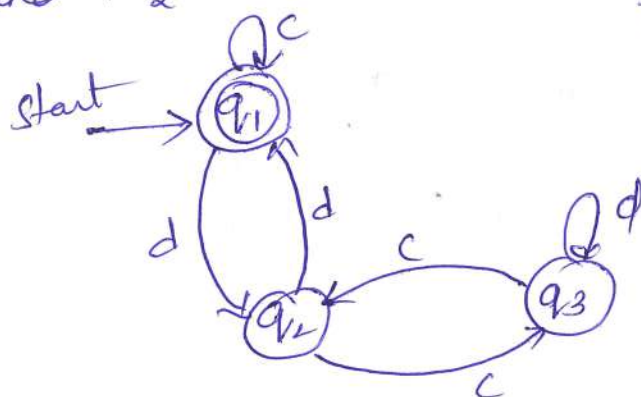
Step 2: The first element in second column is (q_1, q_4) since both q_1, q_4 are reachable from respective initial state. The first element in third column is (q_2, q_5) since $\delta(q_1, d) = q_2$ $\delta(q_4, d) = q_5$

Step 3: Repeat the construction by taking new pair (q_2, q_5) in first row and so on.

Here now new elements appear in the second and subsequent columns which are not in the first column

$\therefore M, M'$ are equivalent

Example 2 Show that the following automata M_1 and M_2 are not equivalent



States

c

d

q_1, q_4

q_1, q_4

q_2, q_5

q_2, q_5

q_3, q_7

q_1, q_6

$\delta \quad \delta$

$\delta \quad \delta$

Here q_1, q_4 is final states where as q_6 is Intermediate state where it doesn't satisfy the rules of Equivalence and hence M_1 and M_2 are not Equivalent.

Minimization of FSM

Testing Equivalence States

Indistinguishable states: Two states P and q of a DFA are called indistinguishable if $\delta(P, w) \in F$ implies $\delta(q, w) \in F$ and $\delta(P, w) \notin F$ implies $\delta(q, w) \notin F$ for all $w \in \Sigma^*$.

Distinguishable states: There exists some string $w \in \Sigma^*$ such that $\delta(P, w) \in F$ and $\delta(q, w) \notin F$ or vice versa then states P and q are said to be distinguishable by a string w .

Equivalence: Two states q_1 and q_2 are equivalent ($q_1 = q_2$) if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or non-final states for all strings x of length k or less.

k -equivalence : Two states q_1 and q_2 are k -equivalent $k \geq 0$ if both $\delta(q_1, x)$ and $\delta(q_2, x)$ are final states or non-final states for all strings x of length k or less.

Minimization

→ For a given language DFA can be represented in many ways differing with no. of states. Always less numbered DFA is preferable. So, it is desirable to reduce the no. of states as far as possible.

- One method for reducing the states of a DFA is based on finding and combining indistinguishable (equivalent) states.

The following procedures describe finding pairs distinguishable (non-equivalent) states.

Procedure for construction of minimum Automata.

Step 1 :- Partition the states of given automaton as set of final states and set of non final states By definition of 0-equivalence.

Q_1^0 - the set of all final states

Q_2^0 - $(Q - Q_1^0)$ the set of non-final states.

Step 2: (Construction of $\pi_1, \pi_2, \dots$)

The subset of π_1 are obtained by partitioning subset of π_0 further.

→ If $q_1, q_2 \in Q$; consider the states in each 'a' where $a \in \Sigma$, corresponding to q_1 and q_2 . If those states are in the same subset of π_0 , q_1 and q_2 are 1-equivalent.

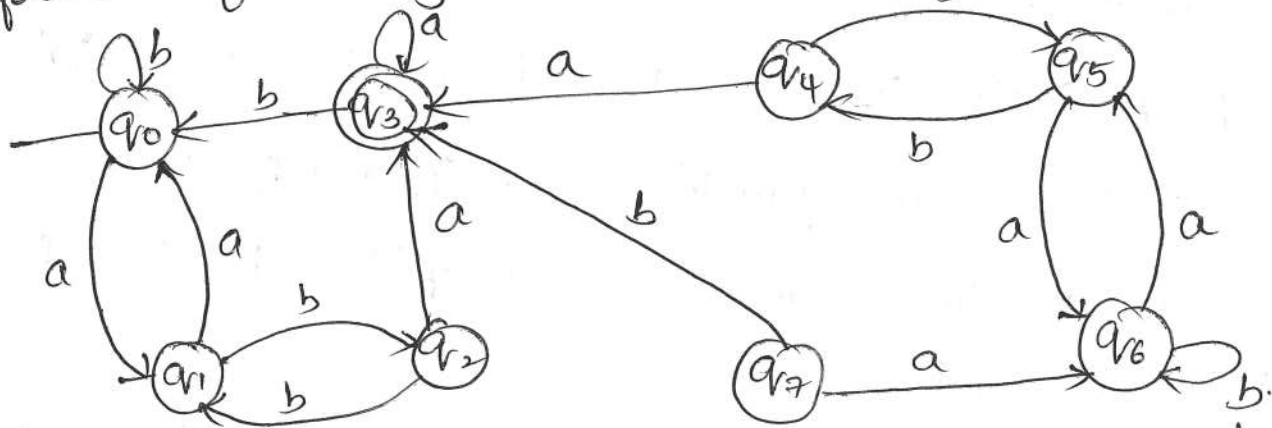
- If the states under some 'a' are in different subsets of π_0 , then q_1 and q_2 are not $\pi_1, \pi_2, \dots$ 1-equivalent of π_0 . ~~then q_1 and q_2 are not~~
By proceeding in this way subsets are constructed.

Step 3: Construct π_n for $n=1, 2, \dots$ upto $\pi_n = \pi_{n+1}$

Step 4: Construction of minimum automaton

- for the required minimum state automata
the states are the equivalence classes obtained in step 3

Example: Construct the minimum state automata for the following transition diagram



Solⁿ we will first construct a transition table for the given DFA. We will start equivalence classes.

States	a	b
→ q ₀	q ₁	q ₀
q ₁	q ₀	q ₂
q ₂	q ₃	q ₁
⊙ q ₃	q ₃	q ₀
q ₄	q ₃	q ₅
q ₅	q ₆	q ₄
q ₆	q ₅	q ₆
q ₇	q ₆	q ₃

Step 1: There is one final state i.e. q₃ Hence.

Q can be partitioned as Q₁⁰ and Q₂⁰

$$Q_1^0 = \{\text{final states}\} = \{q_3\}$$

$$Q_2^0 = Q - Q_1^0 \Rightarrow \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\} - \{q_3\}$$

$$Q_2^0 = \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$$

Step 2: The $Q_1^0 = \{q_3\}$ cannot be partitioned further
 — We will concentrate on $Q_2^0 = \{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$
 In this step we will first compare q_0 with all remain
 states i.e. $\{q_1, q_2, q_4, q_5, q_6, q_7\}$ by observing their
 input transitions.

ip state	a	b
q_0	q_1	q_0
q_1	q_0	q_2

both lie in same
set of Q_2^0

both lie in same
set of Q_2^0

Hence q_0 and q_1 are 1-equivalent

ip state	a	b
q_0	q_1	q_0
q_2	q_3	q_1

both lie in same set of
 Q_2^0

both doesn't lie in
same set and hence q_0 and q_2 are not 1-equivalent.

	a	b
q_0	q_1	q_0
q_4	q_3	q_0

lies in same set of Q_2^0

doesn't lie in
same set of and hence q_0 & q_4 are not 1-equivalent

	a	b
q_0	q_1	q_0
q_5	q_6	q_4

both lie in same set Q_2^0 and hence q_0, q_5 are 1-equivalent

	a	b
q_0	q_1	q_0
q_6	q_5	q_6

both lie in same set of Q_2^0

Hence q_0 and q_6 are 1-equivalent.

	a	b
q_0	q_1	q_0
q_7	q_6	q_3

both lie in the same set Q_2^0 — doesn't lie in same set

Hence q_0 and q_7 are not 1-equivalent.

Thus we obtain as 1-equivalent to q_1, q_5 and q_6

$$\therefore Q_3^0 = \{q_0, q_1, q_5, q_6\}$$

Then the remaining elements are q_2, q_4 and q_7

$\therefore$ We will check equivalence of q_2 with q_4 & q_7

	a	b
q_2	q_3	q_1
q_4	q_5	q_6

both lie in same set Q_3^0

both lie in same set Q_3^0

Hence q_2 and q_4 are in 1-equivalent

	a	b
q_2	q_3	q_1
q_7	q_6	q_5

$q_3 - Q_1^0$
 $q_6 - Q_3^0$ doesn't lie in same set

doesn't lie in the same set

Hence q_2 and q_7 are not in 1-equivalent

Thus we obtained $Q_3' = \{q_2, q_4\}$ $Q_4' = \{q_7\}$
 $\Pi_1 = \{Q_1', Q_2', Q_3', Q_4'\}$
 $= \{\{q_3\}, \{q_0, q_1, q_5, q_6\}, \{q_2, q_4\}, \{q_7\}\}$

Step 3 :- Now $Q_1' = \{q_3\}$ which cannot be partitioned further

Now, Consider the set $Q_2' = \{q_0, q_1, q_5, q_6\}$ for constructing Q_2'' . For that purpose we will compare q_0 with q_1, q_5, q_6

	a	b
q_0	q_1	q_0
q_1	q_6	q_2

q_0 lies in Q_2' and q_2 lies in Q_2'

q_1 lies in Q_2' and q_0 lies in Q_2'

2 - (56)

Hence q_0 and q_1 are not 2-equivalent

state \ i/p	a	b
q_0	q_1	q_0
q_5	q_6	q_4

q_0 lies in Q_2^1 and
 q_4 lies in Q_3^1

q_1 & q_6 lies
in same set Q_2^1

Hence q_0 and q_5 are not 2-equivalent

state \ i/p	a	b
q_0	q_1	q_0
q_6	q_5	q_6

both lies in Q_2^1

both lies in Q_2^1

Hence q_0 and q_6 are 2-equivalent

$$\therefore Q_3^2 = \{q_1, q_5\}$$

$$Q_4^2 = \{q_2, q_4\}$$

$$Q_5^2 = \{q_7\}$$

$$\therefore \Pi_2 = \{Q_2^1, Q_3^2, Q_4^2, Q_5^2\}$$

$$= \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$$

Step 4 : $Q_1^2 = \{q_3\}$

For Q_2^3 we will consider q_0 and q_6 states

	a	b
q_0	q_1	q_0
q_6	q_5	q_6

both lies in Q_2^2

both lies in Q_3^2

Hence q_0 and q_6 are 3-equivalent

Q - (57)

Similarly q_1 and q_5 are 3-equivalent

$$\therefore Q_1^3 = \{q_3\}$$

$$Q_2^3 = \{q_0, q_6\}$$

$$Q_3^3 = \{q_1, q_5\}$$

$$Q_4^3 = \{q_2, q_4\}$$

$$Q_5^3 = \{q_7\}$$

$$\Pi_3 = \{Q_1^3, Q_2^3, Q_3^3, Q_4^3, Q_5^3\}$$

$$\Pi_3 = \{\{q_3\}, \{q_0, q_6\}, \{q_1, q_5\}, \{q_2, q_4\}, \{q_7\}\}$$

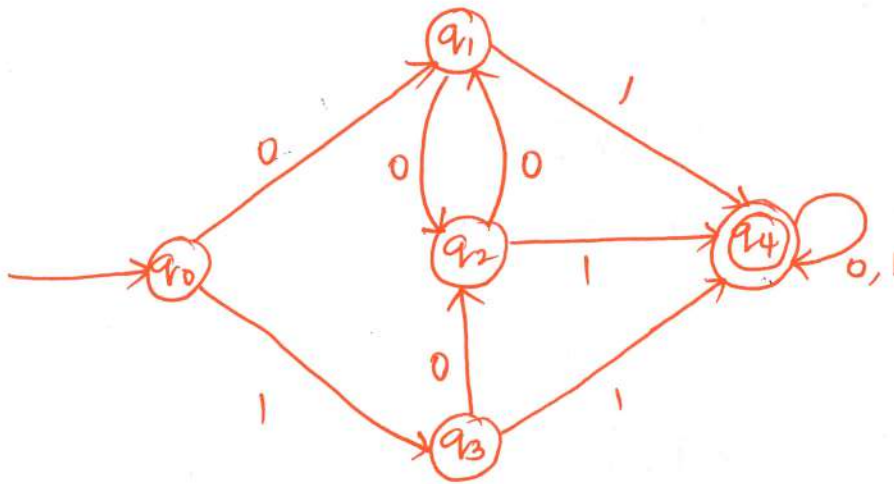
As $\Pi_2 = \Pi_3$ we stop construction of Π_k .

$\therefore$ Thus $q_0 = q_6$, $q_1 = q_5$, $q_2 = q_4$

Hence minimize FA by replacing q_6 by q_0 , $q_5 = q_1$ and $q_4 = q_2$. Then eliminating q_5, q_6 and q_4 states

State \ i/p	a	b
q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
$\textcircled{q_3}$	q_3	q_0
q_7	q_0	q_3

Example 2 Construct the minimum state equivalent DFA



Sol

state \ <u>inp</u>	0	1
→ q ₀	q ₁	q ₃
q ₁	q ₂	q ₄
q ₂	q ₁	q ₄
q ₃	q ₂	q ₄
q ₄	q ₄	q ₄

Step 1:-

0-Equivalence - $\{q_0, q_1, q_2, q_3\}$ $\{q_4\}$

compare q_0 with other states

	0	1
$\{q_0\}$	$\{q_1\}$	$\{q_3\}$
q_1	$\{q_2\}$	$\{q_4\}$

not same

(q_0, q_1) are not 1-equivalence

Same set

	0	1
q_0	$\{q_1\}$	$\{q_3\}$
q_2	$\{q_1\}$	$\{q_4\}$

(q_0, q_2) are not 1-equivalence

	0	1
q_0	$\{q_1\}$	$\{q_3\}$
q_3	$\{q_2\}$	$\{q_4\}$

q_0, q_3 are not 1-equivalent

1-equival → $\{q_0\}$ $\{q_1, q_2, q_3\}$ $\{q_4\}$

Now compare q_1, q_2 & q_3

	0	1
q_1	q_2	q_4
q_2	q_1	q_4

(q_1, q_2) are ² equivalent

	0	1
q_1	q_2	q_4
q_3	q_2	q_4

(q_1, q_3) are ² equivalent

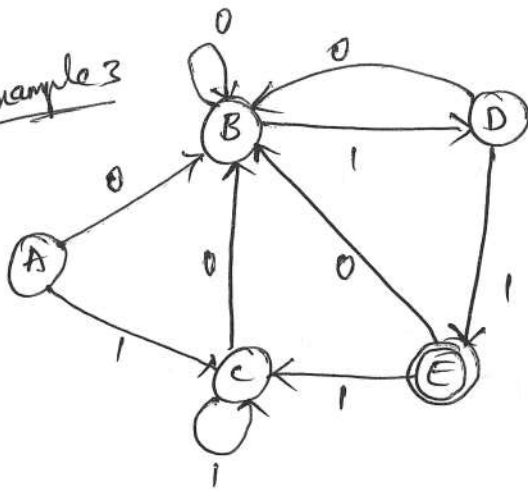
① - $\{q_0, q_1, q_2, q_3\}, \{q_4\}$

② - Equivalent $\rightarrow \{q_0\}, \{q_1, q_2, q_3\}, \{q_4\}$

③ - Equivalent $\rightarrow \{q_0\}, \{q_1, q_2, q_3\}, \{q_4\}$

State \ Input	0	1
$\{q_0\}$	q_1	q_3
$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$	$\{q_4\}$
$\{q_4\}$	q_4	q_4

Example 3



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

0-Equivalence = {A, B, C, D} {E}

1-Equivalence = {A, B, C} {D} {E}

2-Equivalence = {A, C} {B} {D} {E}

3-Equivalence = {A} {C} {B} {D} {E}

4

	0	1
q ₀	q ₁	q ₅
q ₁	q ₆	q ₂
q ₂	q ₀	q ₂
q ₃	q ₂	q ₆
q ₄	q ₇	q ₅
q ₅	q ₁	q ₆
q ₆	q ₆	q ₄
q ₇	q ₆	q ₂

0-Equivalence

= {q₀, q₁, q₃, q₄, q₅, q₆, q₇} {q₂}

1-Equivalence

= {q₀, q₄, q₆, q₇}

{q₁, q₃}

{q₅, q₂}

~~q₅~~

~~q₂~~

2-Equivalence

= {q₀, q₄} {q₆} {q₁, q₃} {q₅, q₂}

3-Equivalence

= {q₀, q₄} {q₆} {q₁, q₃} {q₅, q₂}

The row of 2-Equivalence & 3-Equivalence is equal.

2-(61)

	0	1
$\rightarrow \{q_0, q_4\}$	$\{q_1, q_7\}$	$\{q_3, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_0, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
$- \{q_2\}$	$\{q_0, q_4\}$	$\{q_2\}$

2nd method.

Compare q_0 with other

O-eg- $\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\} \setminus \{q_2\}$.

	0	1
q_0	q_1	q_5
q_1	q_6	q_2

not in same set so
 $q_0 \neq q_1$

	0	1
q_0	q_1	q_5
q_3	q_2	q_6

not in same set so
 $q_0 \neq q_3$

	0	1
q_0	q_1	q_5
q_4	q_7	q_5

are in same set so
 $q_0 = q_4$

	0	1
q_0	q_1	q_5
q_5	q_2	q_6

not equal $q_0 \neq q_5$

	0	1
q_0	q_1	q_5
q_6	q_6	q_4

$\Rightarrow q_0 = q_6$

2-69

$$\begin{array}{c|c|c} q_0 & q_1 & q_5 \\ \hline q_2 & q_6 & q_7 \end{array} \quad q_0 \neq q_7$$

1- Equivalence - $\{q_0, q_4, q_6\} \quad \{q_1, q_3, q_5, q_7\} \quad \{q_2\}$

$$\begin{array}{c|c|c} q_1 & q_6 & q_2 \\ \hline q_3 & q_2 & q_6 \end{array} \quad q_1 \neq q_3$$

$$\begin{array}{c|c|c} q_1 & q_6 & q_2 \\ \hline q_5 & q_2 & q_6 \end{array} = q_1 \neq q_5$$

$$\begin{array}{c|c|c} q_1 & q_6 & q_2 \\ \hline q_7 & q_6 & q_2 \end{array} = q_1 = q_7$$

2- Equin - $\{q_0, q_4, q_6\} \quad \{q_1, q_7\} \quad \{q_3, q_5\} \quad \{q_2\}$

$$\begin{array}{c|c|c} q_3 & q_2 & q_6 \\ \hline q_5 & q_2 & q_6 \end{array} \rightarrow \{q_3 = q_5\}$$

3- Equivalence $\{q_0, q_4, q_6\} \quad \{q_1, q_7\} \quad \{q_3, q_5\} \quad \{q_2\}$

	0	1
$\{q_0, q_4, q_6\}$	$\{q_1, q_7, q_6\}$	$\{q_5, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
$\{q_2\}$	$\{q_6\}$	$\{q_2\}$