PROPOSITION.

Mathematical Logic – It is the foundation upon which proofs and arguments rest.

Proposition –
→ The statement used in mathematical Logic.
→ Can be either true or false, can't be both.
→ Classified as declarative statement to which only one of the truth values can be assigned.

Connectives – ① Conjunction ② Disjunction ③ Negation
④ Implication ⑤ Bidirectional (iff).

– Truth Table –

P	Q	P ⇒ Q	P ⇔ Q
F	F	T	T
F	T	T	F
T	F	F	F
T	T	T	T

– Theory of Computation (Branch of mathematics)

- Automata Theory
- Computability Theory
- Computational Theory

Model of Computation
→ Production system
→ λ (Lambda) calculus
→ Recursive function
→ Turing Machine

Automata

- Deterministic → ① finite ② Infinite
- Non-deterministic

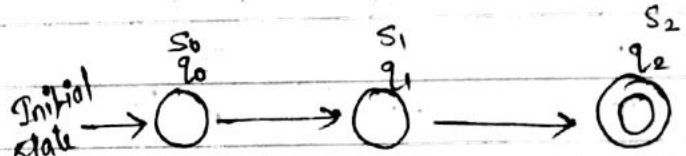
* Automata

→ Automation means machine

→ It is denoted by M

→ There are five arguments with automata

$M(\underbrace{Q}_{\text{set of states}}, \Sigma, \underbrace{q_0}_{\text{initial state}}, \underbrace{\delta}_{\text{Transition fn}}, \underbrace{F}_{\text{set of final states}})$



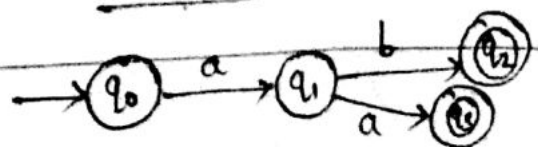
This is called Transition graph.

→ Transition fn requires two state current state and Input state

→ Q = Set of states
Σ = Alphabet (set of input symbols)
q_0 = Initial state
δ = Transition function
F = set of final states

* Finite Automation

Transition Graph



final state is also known as acceptor state.

Transition Table.

Current state	Input a	b	c
q_0	q_1	-	-
q_1	-	q_2	q_3
q_2	-	-	-
q_3	-	-	-

Transition function

$\delta(q_0, a) \rightarrow q_1$

$\delta(q_1, b) \rightarrow q_2$

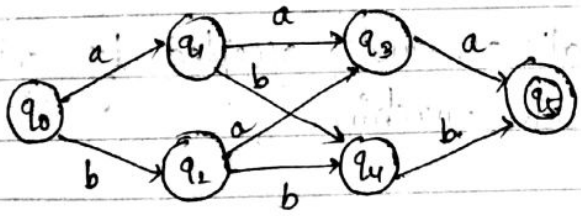
$\delta(q_1, c) \rightarrow q_3$

$Q = \{q_0, q_1, q_2, q_3\}$

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

$F = \{q_2, q_3\}$.

Ques



Current state	Input a	b
q_0	q_1	q_2
q_1	q_3	q_4
q_2	q_3	q_4
q_3	q_5	-
q_4	-	q_5
q_5	-	-

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

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

$F = \{q_5\}$

Function → $\delta(q_0, a) \to q_1$ $\quad \delta(q_2, a) \to q_3$

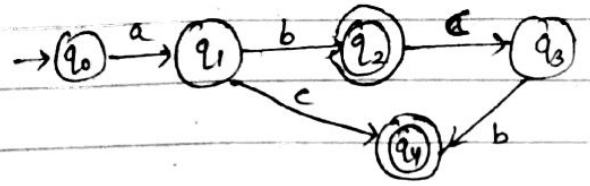
$\delta(q_0, b) \to q_2$ $\quad \delta(q_2, b) \to q_4$

$\delta(q_1, a) \to q_3$ $\quad \delta(q_3, a) \to (q_5)$

$\delta(q_1, b) \to q_4$ $\quad \delta(q_4, b) \to (q_5)$

* Conversion of Deterministic to Non Deterministic

String Processing by finite Automation



Transition Graph.

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

* $Q = \{q_0, q_1, q_2, q_3, q_4\}$

$F = \{q_2, q_4\}$

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

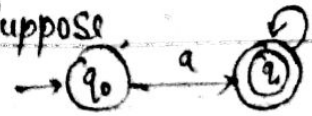
abb
abcc
abca
aba
} These all are not accepted by finite automata machine.

acb

cca

abc → It is also not accepted because it leads to the state which is not the member of final state.

Suppose



a^* accepted $\quad$ * (kleene star)

* is used when we are not sure about the number of occurence of string.

Language of finite Automation
→ Language is termed as set of valid strings.
→ In human language, the validity of string is tested by using the graph.
→ In automata theory, the process of validation is done using machine or tool and finite automation is one such tool.
→ 'w' is string that belongs to a language say L1 then we can have a finite automation M1 iff at the end of the string w, M1 reaches to an acceptor state.

Limitation of finite Automation
→ Limited capacity machine
→ The span of the language accepted by finite automata is very small.

$$\left.\begin{array}{l} \delta(q_0, a) \to q_1 \\ \delta(q_1, b) \to (q_2) \end{array}\right] \Rightarrow \delta^*(q_0, ab) \to q_2 \in F$$

Regular Expression
→ Recursive
→ Let Σ be the character set of language L then the regular expression is defined as per following rules
(a) Every character or alphabet belongs to character set Σ is a regular expression.
(b) Null String ϵ is a Regular Expression.
(c) If R_1 and R_2 are two regular expression then $R_1 + R_2$ is also a regular expression.
(d) If R is a regular expression then R^* is also a regular expression.

Let, for any language L
$\Sigma = \{a, b, c\}$ then
We have,

(a) $a \in \Sigma$

(b) $\{a\} \subseteq \Sigma$

(c) If R_1 & R_2 is regular expression then $R_1.R_2$ and R_2R_1 is also regular expression

* Types of Finite Automation

1. Deterministic
2. Non-deterministic

Deterministic Finite Automation

→ A finite automation is termed as deterministic, if for some given state and an input symbol, the transition $f^n(\delta)$ may lead to a set of states that is a subset of Q., where the set of states can have multiple independent paths.

→ A non deterministic finite automation M is a five tuple structure. where,

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

(a) Q is a finite set of states in which the NFA can exist.

(b) Σ is the set of input symbols that can be fade to NFA.

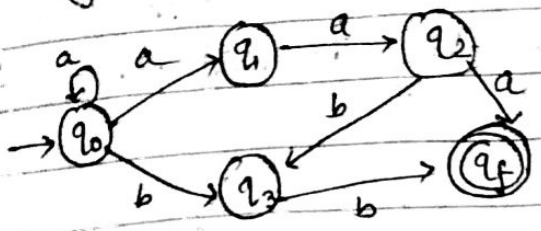
(c) q_0 belongs to Q is the starting state of NFA

(d) $F \subseteq Q$ is the set of final stated

(e) δ is the transition function that maps to the next possible set of states for the given current state of NFA and the current input symbol.

The set of output states may be =

1. Empty state
2. May contain exactly one state.
3. May contain more than one state.



Current	Σ (Input symbol)	
	a	b
→ q_0	(q_0, q_1)	q_3
q_1	q_2	-
q_2	q_f	q_3
q_3	-	q_f
q_f	-	-

→ Reason of Non deterministic

Accepted - bb, a^*, b^* Non-accepted - ab.

* String acceptance by NFA

→ In DFA, a string w is said to be accepted iff the transition system stops at a final state after the processing of entire string w.

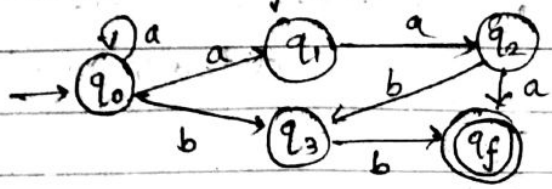
→ Since, there exists one and only one possible path for a string in DFA, we can definitely conclude whether the given DFA accepts or rejects the given string, but with NFA this is not the case.

→ A given string w submitted to an NFA may have multiple simultaneous possibilities when it follows different alternative paths.

→ The string may be accepted in one path rejected in other path and not properly consumed in some other path. so it is difficult to decide the acceptibility of the string but if among the various alternatives paths, there is atleast one path that leads to final state then the string is said to be accepted by NFA.

Equivalence - Equal in some respect/aspect.

* Conversion from NFA to DFA.



$\delta \rightarrow Q \times \Sigma = 2^Q$ states

If Q is the set of states and Σ is the ~~set~~ set of inputs then we get, 2^Q states of transition function.

Transition Table (NFA)

	a	b	...
→ q_0	$\{q_0, q_1\}$	q_3	
q_1	q_2	-	
q_2	q_f	q_3	
q_3	-	q_f	
q_f	-	-	

$Q = \{q_0, q_1, q_2, q_3, q_f\}$

$P(Q) = \{\phi, \{q_0\}, \{q_1\} \ldots$

$\ldots \{q_0, q_1, q_2, q_3, q_f\}$

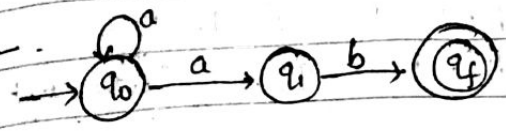
Transition table for DFA

	a	b
→ $\{q_0\}$	$\{q_0, q_1\}$	$\{q_3\}$
$\{q_1\}$	$\{q_2\}$	ϕ
$\{q_2\}$	$\{q_f\}$	$\{q_3\}$
$\{q_3\}$	$\{q_f\}$	
$\{q_f\}$		

Note Eliminate that Input state which is not present in NFA table.

$\{q_2\}$	ϕ	$\{q_f\}$
$\{q_f\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$	$\{q_3\}$

Ques.



Transition Table (NFA)

Current state	Input state a	b
→ q_0	$\{q_0, q_1\}$	-
q_1	-	q_f
q_f	-	-

$Q = \{q_0, q_1, q_f\}$

$P(Q) = \{\phi, \{q_0\}, \{q_1\} \ldots \{q_0, q_1, q_2, q_f\}\}$

$= 2^3 = 8$ states

// This is the input states for DFA.

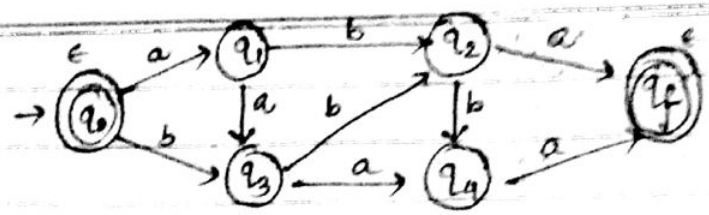
Transition Table (DFA).

Current state	Input symbol a	b
→ $\{q_0\}$	$\{q_0, q_1\}$	ϕ
$\{q_1\}$	ϕ	$\{q_f\}$
----	----	----
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_f\}$
* ϕ	ϕ	ϕ

* Equivalence of Two DFAs

Two DFAs are equivalence if they accept the string (same) in same manner. (It doesnot only depend on length but also in i/s sequence).

zero level equivalence - length of string is 0 (null string).
One level equivalence - length of string 1 is accepted for each ...
so on.



0 level Equivalence ⇒ str length = 0 = ϵ i.e. q_0, q_f
1 level equivalence ⇒ str length = 1 i.e. q_2, q_4 (both accept a)

Transducers / Finite State Machine

→ A finite state machine is ~~nothing but~~ similar to finite automata except that it has (additional) capability of producing output. i.e. FSM = FA + output.

Types of Finite state Machine :-

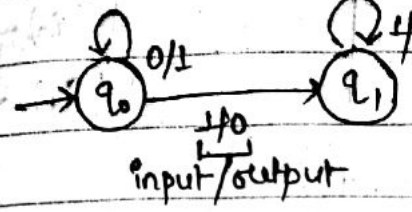
1. ~~Mealy~~ machine
2. Moore machine

① **Mealy Machine** :- → If the output of the FSM is depending on the present state and the present input only then this model of finite state machine is known as Mealy machine.

→ A mealy machine can be described by 6 tuples i.e. $(Q, \Sigma, \Delta, \delta, \lambda, S)$ where

Q = Finite and non-empty set of states
Σ = Set of input symbols ie. input alphabets
Δ = Output alphabet
δ = Transition function that maps present state and present input symbol onto next state. ie $Q \times \Sigma \to$
λ = Output transition / fn that maps present state and present input onto output $Q \times \Sigma \to \Delta$
S = ~~the~~ q_0 (initial or starting state) and $S \in Q$

eg- $(\{q_0, q_1\}, \{0,1\}, \{0,1\}, \delta, \lambda, q_0)$



Present state	Next state		Output	
	0	1	0	1
→ q_0	q_0	q_1	1	0
q_1	—	q_1	—	0

$\delta(q_0, 0) \rightarrow q_0$ ... $\lambda(q_0, 0) \rightarrow 1$

④ Moore Machine :- ⇒ If the output of the FSM is depending on the present state only this model of FSM is known as Moore machine.

⇒ A moore machine can be described by 6 tuples ie. $(Q, \Sigma, \Delta, \delta, \lambda, S)$ where

Q = finite and non-empty set of states

Σ = Input alphabet

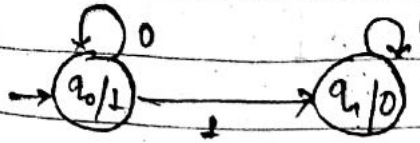
Δ = Output alphabet

δ = Transition fn that maps present state and present input symbol onto next step ie. $Q \times \Sigma \rightarrow Q$

λ = Output fn that maps present state onto output ie. $Q \rightarrow \Delta$

S = q_0 (initial or starting state) and $S \in Q$

eg- $(\{q_0, q_1\}, \{0,1\}, \{0,1\}, \delta, \lambda, q_0)$



Present state	Next state		Output
	0	1	
$\rightarrow q_0$	q_0	q_1	1
q_1	q_1	—	0

$\lambda(q_0) \rightarrow 1$
$\lambda(q_1) \rightarrow 0$

* Conversions in Finite Automation
  1. NFA to DFA
  2. Mealy to Moore
  3. Moore to Mealy

2. Mealy to Moore ⇒ In Mealy machine the output depends upon the present input & present state, whereas in moore machine the output depends upon the only present state.

→ At first stage, we develop a procedure so that all the states of mealy machine which are associated with different options and find them. In case of a state having different output, split the states into some states.

→ This way, we have all the states that are associated with single output.

Considering the following transition table :-

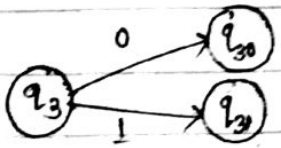
Present state	Next state			
	state a=0	o/p a=0	state a=1	o/p a=1
$\rightarrow q_1$	q_3	1	q_2	1
q_2	q_4	0	q_4	0
q_3	q_2	0	q_1	1
q_4	q_1	1	q_3	1

◆ In the given transition table we have to look into next

state columns for each state and find out the present state associated with more than one output.

→ for q_1 at q_3 for $(q_3, 0) \to 1$ and $(q_2, 1) \to 1$ ie we get same output 1. So, it is not required to split q_1.
→ for q_2 at $(q_1, 0) \to 0$ and $(q_4, 1) \to 0$ (same output 0) So, it is not required to split q_2.
→ for q_3, at $(q_2, 0) \to 0$ and $(q_4, 1) \to 1$ (different output) So, it is required to split q_3.
→ for q_4, at $(q_1, 0) \to 1$ and $(q_3, 1) \to 1$ (same output) So, it is not required to split q_4.

→ Splitting q_3,



→ Thus, the moore machine have 5 states and 5 outputs. The states are $q_1, q_2, q_{30}, q_{31}, q_4$ and the corresponding outputs are 1, 0, 0, 1, 1

→ Transition Graph

Present state	Next state		output
	a=0	a=1	
→ q_1	q_{30}	q_2	1
q_2	q_1	q_4	0
q_{30}	q_2	q_1	0
q_{31}	q_2	q_1	1
q_4	q_1	q_{31}	1

→ In the above transition table, we can find that the initial state q_1 is associated with output 1. This states

that when we apply null input we get an output if moore machine starts at state q_1.

→ Thus, this machine accepts an empty state which is not accepted by mealy machine. To overcome this situation either we must add a new state starting at q_0 or we must neglect the response of moore machine to input null.

→ q_0 q_{30} q_2 . 0

③ Moore to Mealy

Limitations

→ ① Restricted Model
② Input tape is read only memory and there are finite number of states, if is FAM is finite.
③ FSM has only string pattern recognising power.
④ It is not possible to write a FSM that generates the language A^*, B^*.

Application

→ ① Lexical analyzer
② Switching circuit diagram.
③ Text editors :
④ Spelling checker:

Conversion of NFA to DFA

→ ① NFA says that its transition function maps the input queue into summation to a set of all possible states
② The possible states can be empty, contain only one state or more than one.

③ for the output of NFA there are 2^n possibilities where n is the cardinality of Q. ie. each of these possibilities is member of P(Q)

④ Let the NFA with states $Q = \{A, B, C\}$ then the output is limited to the following possibilities -
$\{\}, \{A\}, \{B\}, \{C\}, \{A,B\}, \{B,C\}, \{A,C\}$.

⑤ To convert an NFA to an equivalent DFA, we take each element of P(Q) as input one by one and then find the corresponding output. The outputs will also belong to P(Q).

⑥ Considering the following transition table -

Current state	a	b
$\rightarrow q_0$	q_2	q_0, q_1
q_1	q_2	q_1
q_2	q_1, q_3	q_0
q_3	-	-

$Q = \{q_0, q_1, q_2, q_3\}$
$P(Q) = \{\{\}, \{q_0\}, \{q_1\}, \{q_2\}, \{q_3\}, \{q_0, q_1\}, \{q_0, q_2\}$
$\ldots \{q_0, q_1, q_2, q_3\}\}$

The process of finding the output -

① DFA doesnot support null move.

② for an input state having only one member state, the output will be directly from that of NFA.

③ for an input state having more than one input states, the union of outputs of each state will be taken into consideration. for eg- If the input set is $\{q_0, q_1\}$ and for input symbol 'a' the transition table of NFA says that when current state is q_0 and input is 'a', new state is q_2 and similarly for q_1, it is q_2.

Take union of both outputs is q_0 union q_2 that will give you q_2 ie. $q_0 \cup q_2 \Rightarrow q_2$

So, in DFA, when the current state is set $\{q_0, q_1\}$ input is 'a'. the next state will be q_2.

(4) Now design the transition table/graph for the corresponding DFA.

(5) The transition table of corresponding DFA has the entries in the input that donot occur at all in the output of NFA. ie. these entries will never be generated and has to be eliminated.

* Regular Language

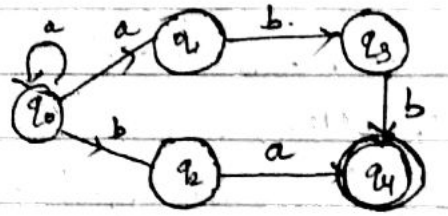
→ The language accepted by finite automata

Regular Expression

→ (1) for a given character set Σ over a language L, regular expression is defined as follows.

(i) Every character or alphabet belonging to Σ is a regular expression. (Previously written).

Ques



P.S.	Next state a	Next state b
q_0	$\{q_0, q_1\}$	q_2
q_1	–	q_3
q_2	(q_4)	–
q_3	–	(q_4)
q_4	–	–

P.S.	Next state a	Next state b
$\{q_0\}$	$\{q_0, q_1\}$	$\{q_2\}$
$\{q_1\}$	ϕ	$\{q_3\}$
$\{q_2\}$	$(\{q_4\})$	ϕ
$\{q_3\}$	ϕ	$(\{q_4\})$
$\{q_4\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2\}$

Date | Page No.

Regular Expression

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

ie. a, b, c are regular expressions.

aa, ab, ac, bb, ba, bc, cc, ca, cb are also regular exp.

then from above; abc, acc, ... are also regular exp.

* Rule for the generation of regular set from regular expression

① Regular expression a^* means any no. of repeatitions of a ie. from 0 to ∞. The corresponding regular set is $\{\epsilon, a, aa, \ldots\}$.

② A regular expression (a+b) means the string can be either a or b. The corresponding regular set is $\{a, b\}$.

③ The regular expression (a+b)c means the string will start with either a or b and will be followed by c. The corresponding regular set is $\{ac, bc\}$.

④ The regular expression (a+b)c(d+e) means that the string will start with either a or b, will be followed by c and will ends with either d or e. The regular set is $\{acd, ace, bcd, bce\}$.

⑤ The regular set corresponding to regular expression null is $\{\epsilon\}$ ie. null set.

Ques. Write a regular expression for the language L = $\{aa, aaaa, abaaaa, \ldots\}$.

Sol⇒ L = $\{aa, aaaa, aaaaaa, \ldots\}$

Here, 'aa' repeats any no. of times starting from 1. As $(aa)^*$ means the no. of repeatition from 0 to ∞. As 1 occurence of aa has to be appended to $(aa)^*$ to make its repeatition atleast once, therefore, the required expression is $R = (aa)^*aa$ or $R = aa(aa)^*$

Ques Write a regular expression for the language {and 0,

Sol⇒ R = 0+1+2

Ques Write a regular expression for the language L = {0,1} such that every string in L starts with 00 and ends with 11.

Sol⇒ R = 00(0+1)*11.

Ques Write a regular expression for the language = {0,1} s.t every string in L contains alternate 0s and 1s.

Sol⇒ If a string ω in L starts with a 0, then it must follow some finite number of sequences of 10 and if the string starts with 1 then it must follow some finite no. of sequences of 01. Therefore, the required regular expression is R = 0(10)* + 1(01)* + 0(10)*1 + 1(01)

Ques Write a regular exp. for the language L over {0,1} such that every string in L ends with {11}.

Sol⇒ (0+1)*11

Ques Write a regular exp. for the language L over {a,b,c} such that every string in L contains a substring bac.

Sol⇒ (a+b+c)*abac(a+b+c)*

Ques Write a regular exp. for the following regular sets -

① Set of all strings over {a,b} containing exactly 2 a's and Two bs.

→ R = {aabb, abab, bbaa, baba, baab, abba}

② Set of all strings over {a,b,c} beginning with c and ending with cc.

→ c(a+b+c)*cc

Date
Page No.

* Identities related to regular expressions

1. $\phi + R = R + \phi = R$

→ $L(\phi + R) = L(\phi) \cup L(R) = \{\} \cup L(R) = L(R) \cup \{\} = L(R + \phi)$

2. $\epsilon R = R\epsilon = R$

→ $L(\epsilon R) = L(R\epsilon) = L(R)$

The language corresponding to $L(\epsilon R)$ is the set obtained by prefixing ϵ to every element of the set $L(R)$, that keeps the set $L(R)$ unchanged. Hence,

$L(\epsilon R) = L(R)$ and $L(R\epsilon) = L(R)$

$\Rightarrow \epsilon R = R$ $\Rightarrow R\epsilon = R$.

3. $R + R = R$

→ Since; $L(R+R) = L(R) \cup L(R) = L(R)$. $\Rightarrow R + R = R$.

4. $\epsilon^* = \epsilon$

→ $L(\epsilon^*) = \{\epsilon, \epsilon\epsilon, \epsilon\epsilon\epsilon, \ldots\} = \{\epsilon, \epsilon, \epsilon, \epsilon \ldots\} = \{\epsilon\}$

So, $L(\epsilon^*) = L(\epsilon) \Rightarrow \epsilon^* = \epsilon$

5. $(PQ)^* P = P(QP)^*$

→ LHS = $(PQ)^* P = P + (PQ)P + (PQPQ)P + \ldots$

$= P + PQP + PQPQP + \ldots$

$= P + P(QP) + P(QPQP) + \ldots = P(QP)^*$

$R = \epsilon$ (null string)
$R = \phi$ (null set, no element)

Date
Page No.

* Finite automation corresponding to Regular expression.

1. 'ϵ' is the primitive level of regular expression.

$R = \epsilon$ → (Sf)

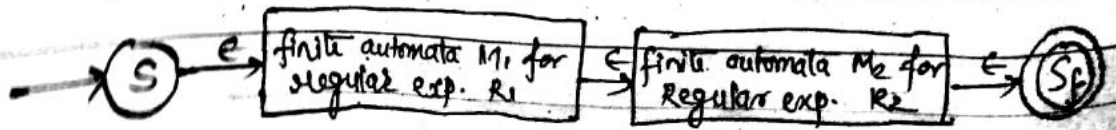
2. Alphabet level is the primitive level regular expression ie. every alphabet of the character set Σ is a regular expre. Let $\Sigma = \{a, b, c\}$ then a, b and c are regular exp. Therefore, finite automation that corresponds to regular exp. i.e. $R = a$ is given by → (S) —a→ (Sf)

3. The last primitive level regular exp. is $R = \phi$. In this case no string is generated by the regular exp.

* Recursive definition of Regular expression and their corresponding regular expression

Ques Let R_1 and R_2 are regular exp. then $R_1 . R_2$ are also regular exp.

Proof - Let R_1 & R_2 be two primitive regular exp. and let M_1, M_2 be their corresponding finite automata. Then, the finite automata M for the regular expression $R_1 . R_2$ is as follows



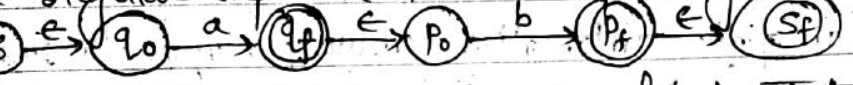
Date
Page No.

i.e. R_1R_2

② As we have seen that the finite automation M for R_1 and R_2 can be obtained by connecting M_1 and M_2 through serial path. ⑥ The null string indicates that the starting and ending states are to be superimposed carefully with the corresponding states in M_1 and M_2. The null strings are to be removed before final automation.

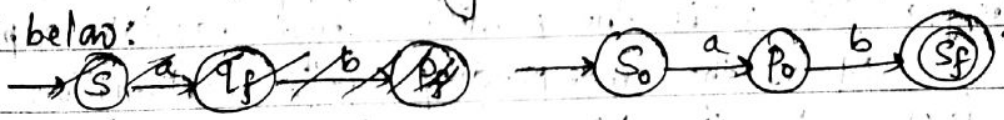
⑥ Let $R_1=a$ and $R_2=b$ The finite automation for $R_1=a$ is given by $\rightarrow (q_0) \xrightarrow{a} ((q_f))$: similarly, the final automation for regular exp. $R_2=b$ is given by $\rightarrow (p_0) \xrightarrow{b} ((p_f))$

④ The construction step for finite automation corresponding to the regular exp. $R=R_1R_2=ab$ is given by.



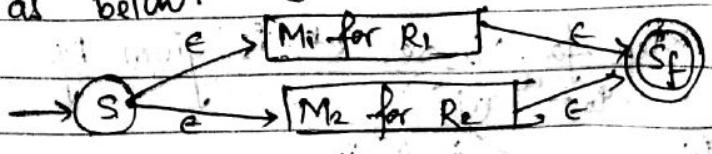
The states q_f and p_f are not final states because strings to be accepted after completing R_1 and R_2.

⑤ The final finite automation corresponding to regular exp. $R=R_1R_2=ab$ after removing the ϵ transition is given below:



Ques If R_1 and R_2 are regular exp. then R_1+R_2 is also regular exp.

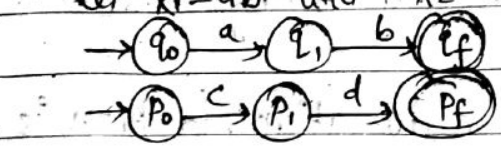
Proof → Let R_1 & R_2 be two primitive level regular exp. having as their M_1 & M_2 corresponding finite automata. Then the finite automation M for the regular exp. R_1+R_2 can be represented as below.



The above figure says that the finite automation M for R_1+R_2 can be obtained by connecting M_1 and M_2 through a

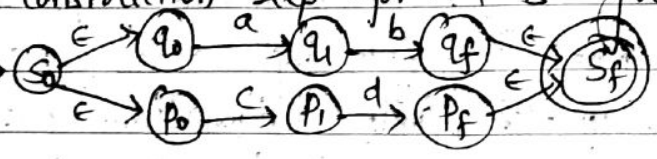
parallel path. ϵ indicate that the starting & ending states are to be superimposed suitably with corresponding states in M_1 and M_2. ϵ are to be removed before final automation is created.

Let $R_1 = ab$ and $R_2 = cd$. Then M_1 is given by

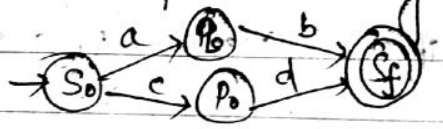


Similarly, M_2 is given by

The construction step for M is given by

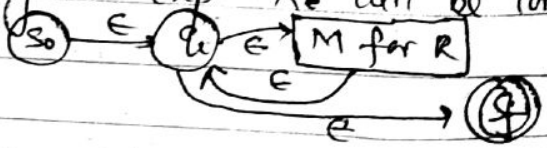


The final finite automation corresponding to regular exp. $R = R_1 + R_2$ after removing ϵ transition is



Ques If R is a regular exp. then R^* is also a regular exp.

Proof → Let R be the primitive level regular exp. having M as the corresponding finite automation. Then finite automation M_1 for regular exp. R^* can be constructed as below:

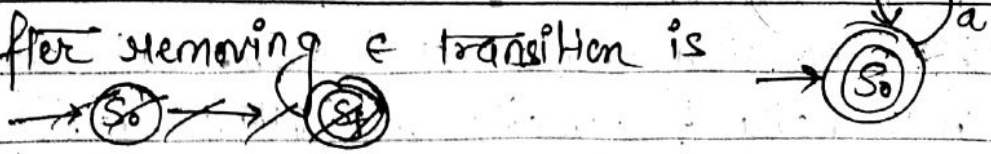


Let $R = a$, therefore the finite automation M for $R = a$ is given by

The construction step for M_2 over R^* is given by

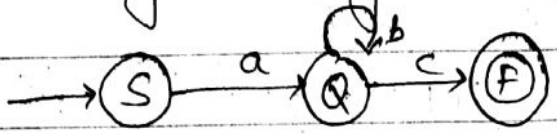
The final finite automation corresponding to regular exp.

$R = R^*$ after removing ϵ transition is



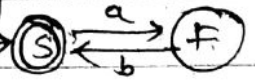
Ques Construct a finite automation for the regular language represented by the regular exp. ab^*c.

Sol:



Ques Construct a finite automation for the regular exp. $(ab)^*$

Sol:



* Arden's Theorem

Let P, Q and r be the regular expression over the character set Σ, if P is not null string then $r = q + rp$ has a unique solution qp^*.

First, we will try to find whether qp^* is a soln of the eqn $r = q + rp$ substituting r by qp^* on both sides of the eqn $r = q + rp$.

$$\Rightarrow qp^* = q + (qp^*)p = q(\epsilon + pp^*) = qpp^* = qp^*$$

Now, we will try to find whether it is the only soln or not and for this replace r by $q + rp$ on the RHS of the eqn $r = q + rp$.

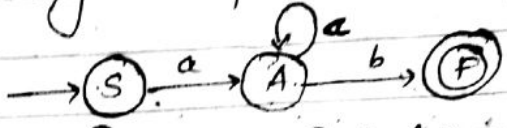
$$r = q + rp$$
$$= q + (q + rp)p = q + qp + rpp \ldots$$
$$= q + qp + (q + rp)pp = q + qp + qpp + rppp \ldots$$
$$= q + qp + qp^2 + \cdots$$
$$= q(\epsilon + p + p^2 + \ldots) = qp^*$$

Hence, $r = q + rp$ has unique soln qp^*.

Ques Where to use Arden's theorem to a finite automation.

→ ① Finite automation should not contain null string move
② There should be exactly one initial state.

Ques Find the regular exp. corresponding to the finite autom. -ation.



Sol→ $S = \epsilon$ —① , $A = Sa + Ac$ —② , $F = Ab$ —③

Substituting ① in ②

$A = Sa + Ae = \epsilon a + Ac = a + Ac$

Using Arden's theorem $A = ac^*$ —④

Putting ④ in ③, $F = Ab$ ie, $F = ac^*b = ab^*$

This is the final result of the corresponding ques.

Ques Find the regular exp. corresponding to the finite automation

Sol→ $S = \epsilon + Sc$ —① $A = Sa + Ac + Fa$ —②

$F = Ab$ —③

Substituting ③ in ②. Applying Arden's thm in ①, we get

$S = c^*$ —④

Substituting eqn ③ & ④ in eqn ②, we get

$A = c^*a + Ac + Aba = c^*a + A(c + ba)$ —⑤

Applying Arden's thm on eqn 5.

$A = c^*a(c+ba)^*$ —⑥

Putting ⑥ in ③, we get, $F = c^*a(c+ba)^*b$

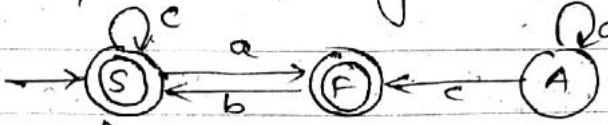
* Algorithm to find the regular exp. corresponding to a finite automation :—

→ ① Simplify M by eliminating the unreachable states.
② If all the states are eliminated, then the regular exp. is null ie ϕ.

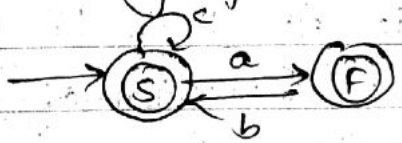
③ for each final state Q_F in M, let M_F be the DFA then M and M_F are same except that Q_F is the only finite state. Simplify M_F by eliminating the unreachable states (if any) and then reduce the transition diagram. Let R_F be the regular exp. for M_F.

④ The final regular exp. for M are $R_1 + R_2 + R_3 + \cdots + R_n$ where n is the no. of final states in M.

Ques: Find the regular exp. corresponding to the finite automation M given by



Sol:→ ① There is no path from the state S to state A. Hence, state A is unreachable and has no role to the required regular exp. So, state A can be eliminated. Thus, the remaining finite automation is



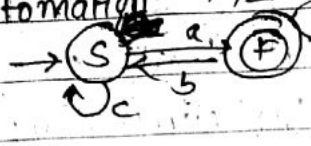
② As there are two final states S and F, first make F as non-final state and find the regular exp. for the new finite automation M_1 given below:

③ Now convert M_1 to a suitable form by eliminating the state F and the corresponding transition diagram is

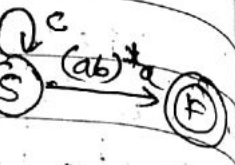
and the corresponding regular exp is $R_1 = (c + ab)^*$

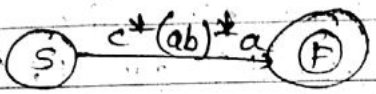
④ Making S as non final state, find the regular exp. for the

new finite automation M_2 given by



Now reduce M_2 to the transition graph





and the corresponding regular exp. is $c^*(ab)^*a$.

④ The final regular exp. $R = R_1 + R_2$ ie $(c+ab)^* + c^*(ab)^*a$

* Closure properties of regular sets

→ ① Regular exp(s) are closed ~~and~~ over union operations.

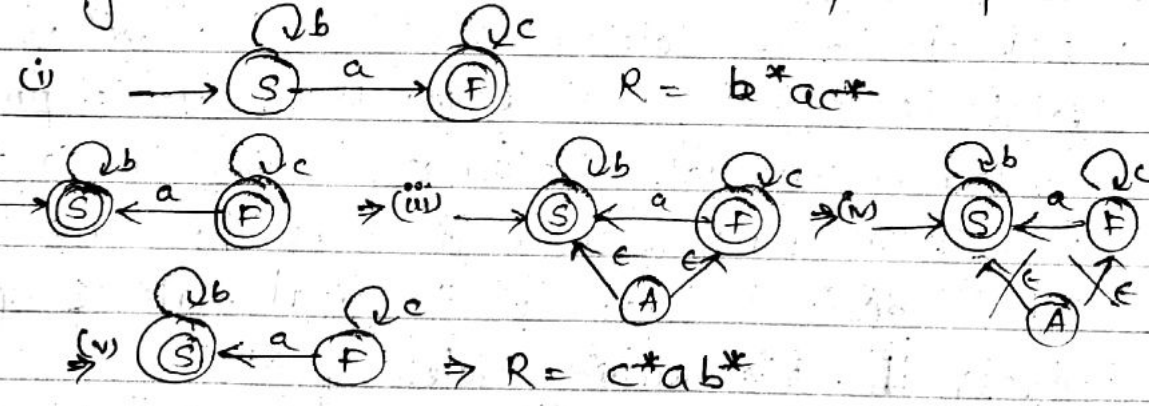
:- Let S_1 and S_2 are two regular sets having regular exp(s) R_1 and R_2 respectively. Let $S = S_1 \cup S_2$ then every element of S can be expressed either by exp. R_1 or R_2 or both. Thus set S can be generated by regular exp. R_1+R_2. Since, the union of two regular sets has generated a regular set, the regular sets are closed under the union operation.

② Regular sets are closed over concatenation operation.

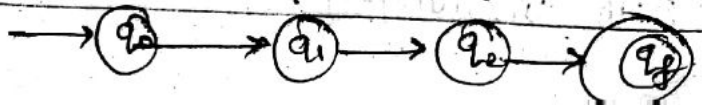
:- Let S_1 and S_2 are two regular sets having corresponding regular exp(s) R_1 & R_2. Let S is a set created by the concatenation of the elements of S_1 and S_2 in order. Now, each of element of S can be broken into two parts with 1st part belongs to S_1 and 2nd part belongs to S_2. ie 1st part corresponds to R_1 and 2nd to R_2. It means every element of S can be generated by the regular exp. R_1R_2. Thus, S is a regular set. Hence, regular sets are closed over concatenation.

(3) Regular sets are closed over the transpose operation
:- Let S_1 be a regular set associated with regular exp. R_1 and M_1 be the corresponding finite automation. Now, for every string w belongs to S_1, there is a path in M_1 from initial state to final state. Now, we can perform following operations on M_1 to produce M_2

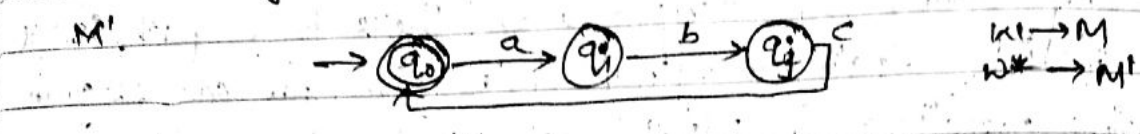
(a) Reverse the dirn of each subpath.
(b) Change the initial state of M_1 as final state of M_2. Create a new start state and connect it to all acceptor state of M_1 using null string transition.:
(c) Remove all null transitions.
(d) Now, let S_2 be the set of all strings accepted by M_2. Now, every string z belongs to S_2 is the reverse of $w \in S_1$. It means S_2 is the set of reverse strings of S_1. As S_2 is associated with M_2 and M_2 is associated with a regular exp. then S_2 is a regular set. Hence, Regular sets are closed over transpose operation.



(4) Regular sets are closed over kleen star.
:- The closure of a regular set over kleen star means if R is a regular exp then R^* is also a regular exp.
Let R is a regular exp. with corresponding finite automation as given below.



Wrap around the finite automation M and superimpose the final state q_f with initial state q_0 to make the initial state as final state and create the new automation say M'.



* Pumping Lemma for Regular language

→ ① Every language is not regular

② In order to prove that a language is not regular, pumping lema is used.

➡ Let L is a regular language then there exists a const. n such that for every string $w \in L$ and $|w| \geq n$, w can be broken into three parts x, y, z such that $w = xyz$, $y \neq \epsilon$ and $|xy| \leq n$. Then if $i \geq 0$ the string $xy^iz \in L$

→ Let M be the DFA that corresponds to the language L and n is the no. of states in it. ie. n is desired constant. Let w be a string that belongs to L. Let length of w be m such $m > n$. Let $w = a_1, a_2, \ldots a_{m-1} a_m$ and let q_0 be the starting state of M and δ be the transition fn. Then $\delta(q_0, w)$ leads to the final state q_f by passing through the intermediate states of M.

→ Let set of states $S = (q_0, q_1, \ldots, q_{m-1}, q_m)$ be the sequence of states. then as $m > n$ means all these states are not distinct states ie. repetition of states is bound to occur.

⇒ Let the 1st repetition occurs at $q_j = q_k$. Then the sequence S can be divided into following three parts

$x = q_0 \ldots q_j$, $y = q_{j+1} \ldots q_k$, $z = q_{k+1} \ldots q_n$.

As $k \leq n$ and $|xy| \leq n$, Means every repetition of y keeps the acceptance path unchanged.

Page No.

Ques Use pumping lemma to prove that the language $L = \{0^s \mid s \text{ is a perfect square}\}$ is not regular.

Sol^n $L = \{\epsilon, 0^1, 0^4, 0^9, \ldots\}$

Let L be a regular language and M be the corresponding finite automation with n distinct states. Let w be the string belong to L i.e. $w \in L$ $|w| = n^2$. Since, n is +ve integer then $n^2 \geq n$, it means w can be broken into 3 parts xyz with $|xy| \leq n$ and $|y| > 0$. Since, $w \in L$ therefore $xyz \in L \Rightarrow xy^2z$ should belong to L

$|xy^2z| = |x| + 2|y| + |z| \Rightarrow |xy^2z| > n^2 \Rightarrow |xy^2z| > |w|$