Module-2

Regular Expression

Definition of a Regular Expression

A regular expression is a string that can be formed according to the following rules:

* $\emptyset$ is a regular expression denoting an empty language
* ε is a regular expression indicates language containing empty string.
* Every element in Σ is a regular expression
* Given two regular expression α & β $\alpha\beta$ is a regular expression.
* Given two regular expression α & β, $\alpha \cup \beta$ is a regular expression.
* Given a regular expression α, α^* is a regular expression.
* Given a regular expression α, α^+ is a regular expression.
* Given a regular expression α, (α) is a regular expression.

So, if we let $\Sigma = \{a, b\}$, the following strings are regular expressions:

$$\emptyset, \varepsilon, a, b, (a \cup b)^*, abba \cup \varepsilon.$$

The expression obtained by applying any of rules above are regular expression.

Eg Obtain a regular expression (R.E) representing strings of a's & b's having length 2.

Sol:
$$R.E = aa + ab + ba + bb$$
or
$$R.E = (a+b)(a+b)$$

Eg Obtain a R.E representing strings of a's & b's having odd length.

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

Eg Obtain a R.E to accept strings of a's & b's of length ≥ 2

Sol:
$$\varepsilon + a + b + aa + ab + ba + bb$$
or
$$(\varepsilon + a + b)(\varepsilon + a + b)$$
or
$$(\varepsilon + a + b)^2$$

Eg Obtain a R.E to accept strings of a's & b's whose tenth symbol from the right end is a.

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

(Eg) Obtain a R.E to accept strings of a's & b's for the following:

(i) Starting with a & ending with b

(ii) with atleast three consecutive a's.

Sol:) (i) $a(a+b)^*b$

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

(Eg) Obtain a R.E to accept the ~~words~~ strings of a's & b's with 2 (or) more letters but beginning & ending with the same letter.

Sol:)

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

(Eg) obtain a R.E to accept strings of a's & b's whose length is either even or multiples of 3 or both.

Sol:)

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

(Eg) Obtain a R.E to accept strings of a's & b's such that every block of 4 consecutive symbols contains least 2 a's.

Sol:)

$$[a(a+b)a(a+b) + (a+b)a(a+b)a + aa(a+b)(a+b) + (a+b)(a+b)aa + (a+b)aa(a+b) + a(a+b)(a+b)a]^+$$

Eg Obtain a R.E for the language

(i) $L = \{a^n b^m \mid m+n \text{ is even}\}$

(ii) $L = \{a^n b^m \mid m \geq 1, n \geq 1, nm \geq 3\}$

(iii) $L = \{a^{2n} b^{2m} \mid n \geq 0, m \geq 0\}$

Sol?

(i) $(aa)^*(bb)^* + a(aa)^* b(bb)^*$

(ii)

Case (i): Since $nm \geq 3$

if $m=1$ then $n \geq 3$

R.E = $aaaa^* b$.

Case (ii): Since $nm \geq 3$

if $n=1$ then $m \geq 3$

R.E = $bbbb^* a$

Case (iii): Since $nm \geq 3$

if $m \geq 2$ then $n \geq 2$

R.E = $aaa^* bbb^*$.

R.E = $aaaa^* b + bbbb^* a + aaa^* bbb^*$.

(iii) $(aa)^*(bb)^*$

Eg Obtain the R.E for strings of a's & b's containing not more than 3 a's.

Sol? $b^*(\varepsilon + a) b^* (\varepsilon + a) b^* (\varepsilon + a) b^*$.

Eg Obtain a R.E for the language $L = \{a^n b^m \mid n \geq 4, m \leq 3\}$

Sol?

$aaaaa^* (\varepsilon + b) (\varepsilon + b) (\varepsilon + b)$

(Eg) Obtain the R.E for the $L=\{w : n_a(w) \bmod 3 = 0$ where $w \in (a,b)^*\}$

Sol? $(b^*ab^*ab^*ab^*)^*$

(Eg) Obtain the R.E for the set of all strings that do not end with 01 over $\{0,1\}^*$.

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

(Eg) Obtain a R.E for $L=\{w$: strings ends with ab(or) ba where $w \in (a,b)^*\}$

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

or

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

Analysing a Simple Regular Expression

(i) $L((a \cup b)^*b) = L((a \cup b)^*)\,L(b)$

$= (L((a \cup b)))^*\,L(b)$

$= (L(a) \cup L(b))^*\,L(b)$

$= (\{a\} \cup \{b\})^*\{b\}$

$= \{a,b\}^*\{b\}$

(ii) $L(((a \cup b)(a \cup b))a(a \cup b)^*) = L(((a \cup b)(a \cup b)))\,L(a)\,L((a \cup b)^*)$

$= L((a \cup b)(a \cup b))\{a\}(L((a \cup b)))^*$

$= L((a \cup b))\,L((a \cup b))\{a\}\{a,b\}^*$

$= \{a,b\}\{a,b\}\{a\}\{a,b\}^*$

Kleene's Theorem

Theorem: Any language that can be defined with a regular expression can be accepted by some FSM & so is regular.

(or)

Let 'R' be a regular expression then their exists finite automata $M = (Q, \Sigma, \delta, q_0, F)$ which accepts $L(R)$

(or)

prove that their exists a finite automata to accept the language $L(R)$ corresponding to the regular expression R.

Proof: By definition ϕ, ε, a are regular expression, so that corresponding machines to recognize the language for the respective expressions are shown in figure below:

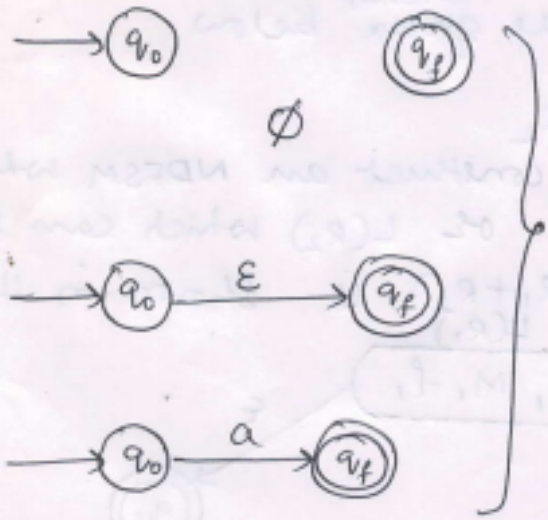


Fig: NDFSM to accept ϕ, ε, a.

We must show how to build FSMs to accept languages that are defined by regular expression that exploit the operations of concatenation, union and Kleene star.

The schematic representation of R.E are to accept the language L(R) is shown in the figure where 'q' is the start state & 'f' is the final state of machine M.

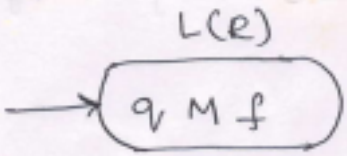


fig: Schematic representation of FSM accepting L(R).

In the definition of a R.E, it is clear that if R&S are R.E then R+S, R^*, R.S are R.E which clearly uses 3 operators +, *, .

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ be a machine which accepts the language $L(R_1)$ corresponding to the R.E R_1.

Let $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ be a machine which accepts the language $L(R_2)$ corresponding to the R.E R_2.

The various machines corresponding to the R.E R_1+R_2, $R_1 \cdot R_2$, R^* are shown below.

Case (i) :- $R = R_1 + R_2$

We can construct an NDFSM which accepts either $L(R_1)$ or $L(R_2)$ which can be represented as $L(R_1+R_2)$ is shown in the figure below:

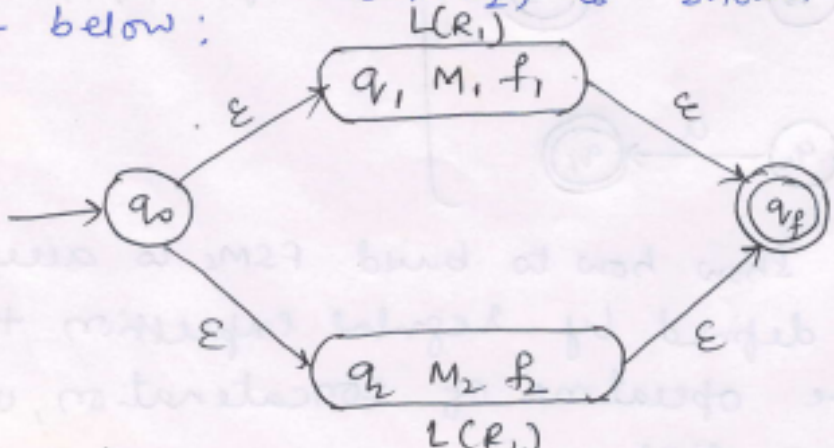


Fig: To accept the language $L(R_1+R_2)$

Case (ii) : $R = R_1 \cdot R_2$

We can construct an NDFSM which accepts $L(R_1)$ followed by $L(R_2)$ which can be represented as $L(R_1 \cdot R_2)$ as shown in figure below:

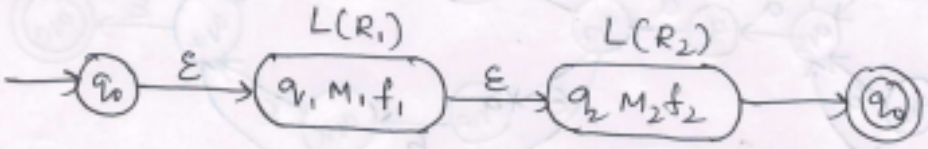


Fig: To accept the language $L(R_1 \cdot R_2)$

Case (iii) : $R_1 = (R_1)^*$

We can construct an NDFSM which accepts the language $L(R_1)^*$ as shown in figure below:

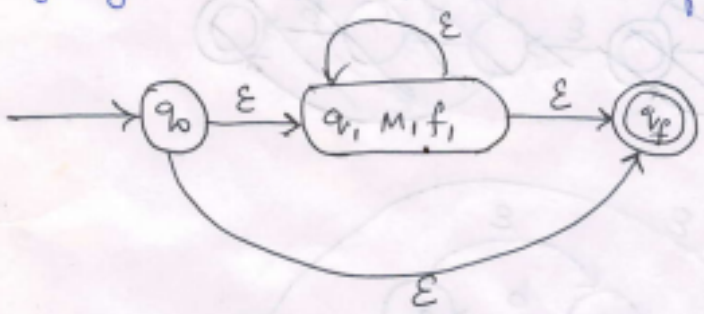


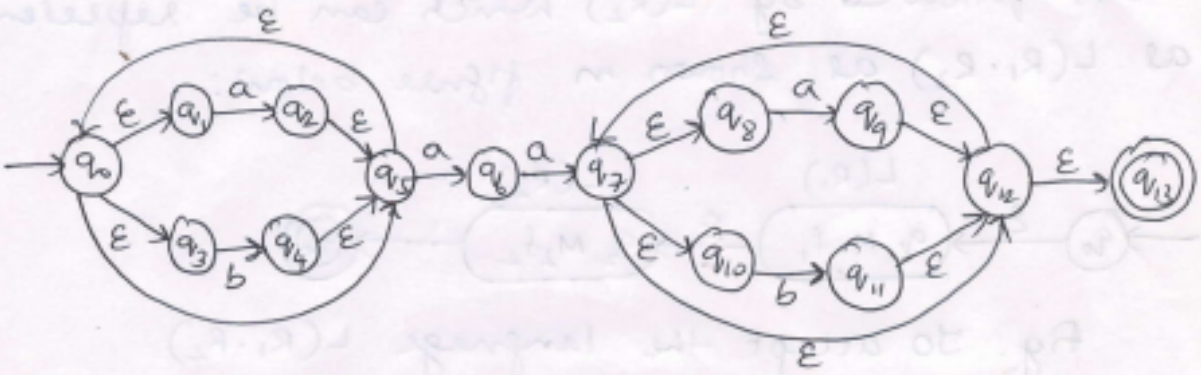
Fig: To accept the language $L(R)^*$

Hence for given any R.E it is proved that there exists a finite state machine to accept the language $L(R)$.

Building an FSM from a Regular expression

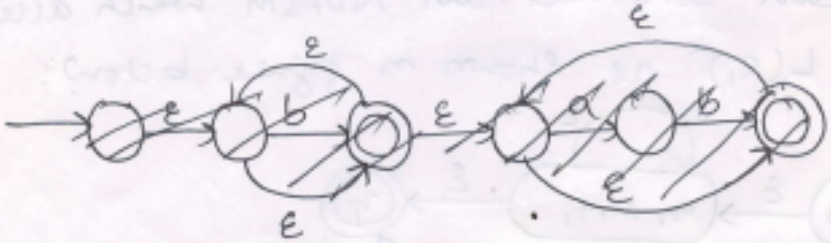
(Eg) Obtain an FSM for R.E $(a+b)^* aa(a+b)^*$

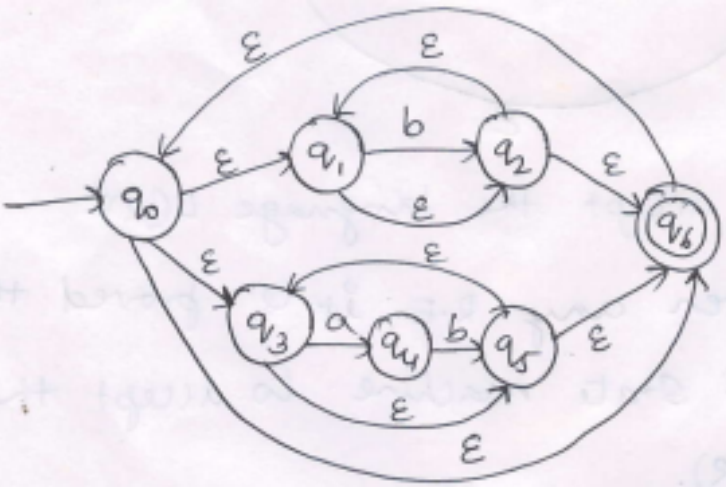
Soln:



(Eg) Obtain an FSM for $(b+ab)^*$ or $(b \cup ab)^*$

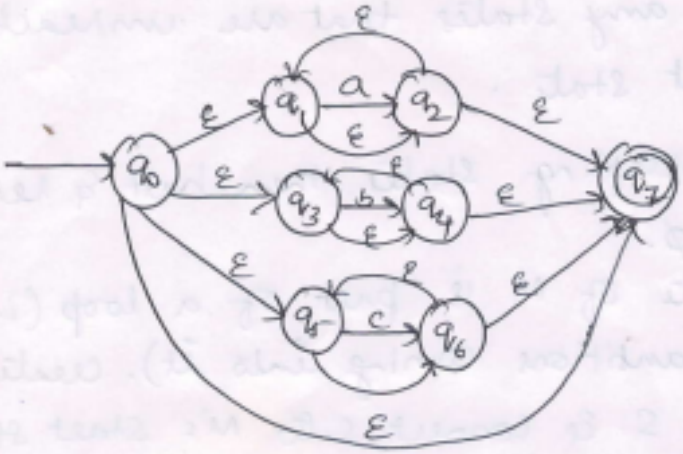
Soln:





(Eg) Obtain an FSM for regular expression $a^* + b^* + c^*$

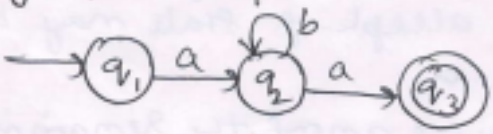
Soln



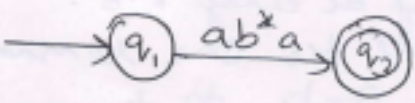
Building a Regular expression from an FSM

We can build a regular expression from an FSM. The goal of the algorithm is to construct, from ~~a~~ an input FSM M, an output machine M' such that M & M' are equivalent & M' has only two states, a start state & a single accepting state. It will also have just one transition, which will go from its start state to its accepting state. The label on that transition will be a R.E that describes all the strings that could have driven the original machine M from its start state to some accepting state.

(Eg) Building an equivalent Machine M



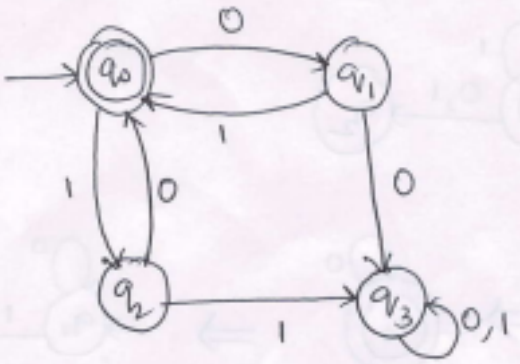
Soln



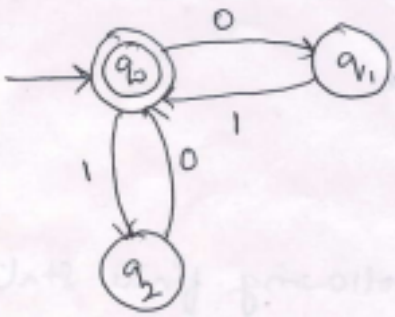
fsmtoregexheuristic(M: FSM) =

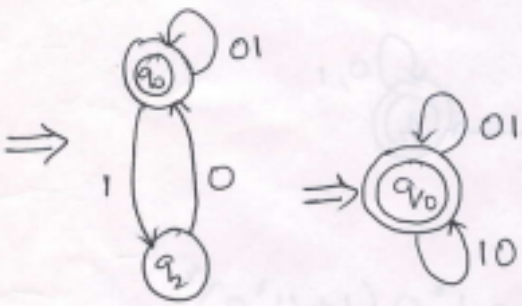
1) Remove from M any states that are unreachable from the start state.
2) If M has no accepting states then halt & return the simple R.E ∅.
3) If the start state of M is part of a loop (i.e it has any transitions coming into it). Create a new start state S & connect S to M's start state via an ε-transition. This new start state S will have no transitions into it.
4) If there is more than one accepting state of M or if there is just one but there are any transitions out of it, create a new accepting state & connect each of M's accepting states to it via an ε-transition. Remove the old accepting states from the set of accepting states. Note that the new accepting state will have no transitions out from it.
5) If, at this point, M has only one state, then that state is both the start state & the accepting state & M has no transitions. So L(M) = {ε}. Halt & return the simple R.E ε.
6. Until only the start state & the accepting state remain do
   6.1: Select some state rip of M. Any state except the start state or the accepting state may be chosen.
   6.2: Remove rip from M.
   6.3: Modify the transitions among the remaining states so that M accepts the same strings. The labels on the rewritten transitions may be any R.E.
7. Return the R.E that labels the one remaining transition from the start state to the accepting

(Eg) Obtain the R.E for the following finite state machine automata



Sol: q_3 state is not reachable from start state & Eliminate it:

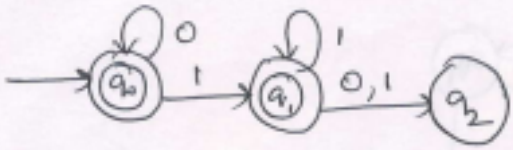




It is clear from the transition diagram that the machine accepts.

$$R.E = (01+10)^*$$

(Eg) Obtain a R.E for the following finite state machine automata.



Sol?

$R.E_1 = 0^*$

$R.E_2 = 0^*11^*$

$R.E = R.E_1 + R.E_2$

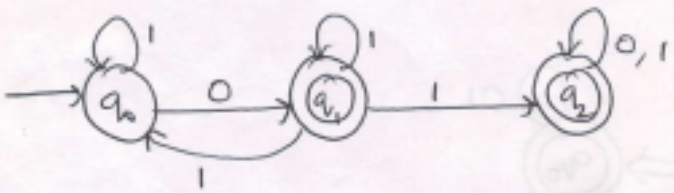
$= 0^* + 0^*11^*$

$= 0^*(\varepsilon + 11)^*$

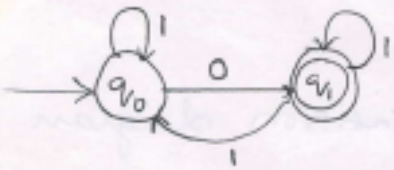
$R.E \Rightarrow 0^*(\varepsilon+1)^*$

(Eg) Obtain a R.E for the following finite state machine.

Sol?



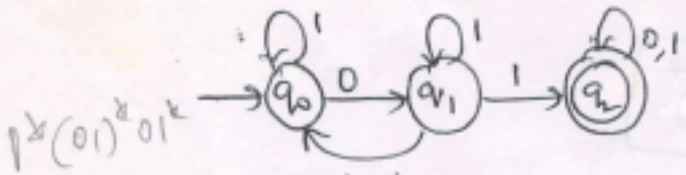
Sol?

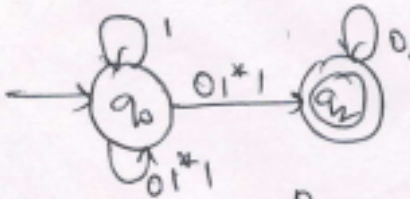


$R.E_1 = 1^*0(1+11^*0)^*$

or

$1^*01^*(\varepsilon + 11^*01^*)$





$R.E_2 = (1+01^*1)^*01^*1(0+1)^*$

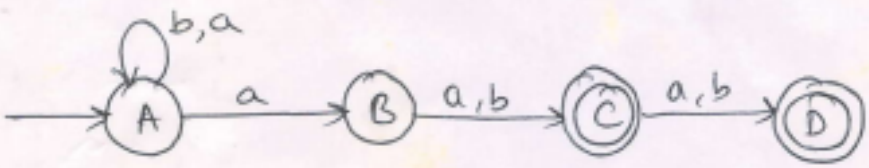
$R.E = R.E_1 + R.E_2$

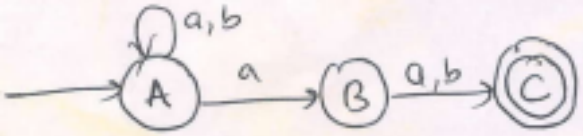
$R.E = 1^*0(1+11^*0)^* + (1+01^*1)^*01^*1(0+1)^*$

(Eg) Obtain an regular expression for a finite state machine given below:

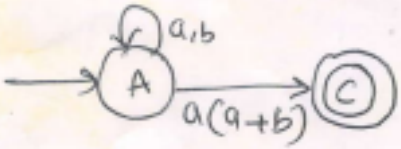
	a	b
→ A	A,B	A
B	C	C
* C	D	D
* D	∅	∅

Sol:

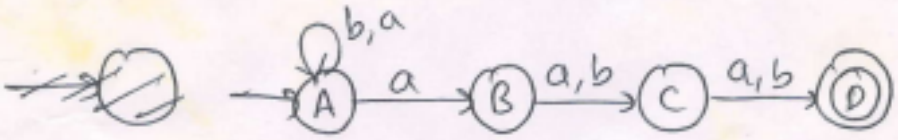


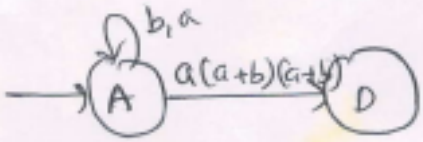


R.E



$R.E_1 = (a+b)^* a(a+b)$





$R.E_2 = (a+b)^* a(a+b)(a+b)$

$$R.E. = \left[(a+b)^* a(a+b)\right] + \left[(a+b)^* a(a+b)(a+b)\right]$$

Regular Grammar

Definition: A regular grammar G is a quadruple (V, T, P, S)

i.e $G = (V, T, P, S)$

where:

V : the set of variables or nonterminals
T : the set of terminals
P : Productions
S : Start Symbol.

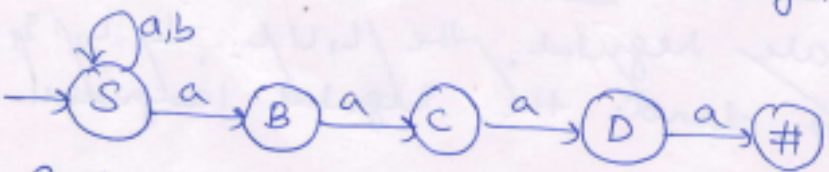
In regular grammar, all rules in R must:

* have a left-hand side that is a single non terminal and
* have a right hand side that is ϵ or a single terminal or a single terminal followed by a single non terminal.

(Eg) Let $L = \{w \in (a,b)^* : w$ ends with pattern aaaa$\}$

$S \to aS$
$S \to bS$
$S \to aB$
$B \to aC$
$C \to aD$
$D \to a$

Applying grammar to fsm to this grammar we get



$G = (V, T, P, S)$

$V = \{S, B, C, D\}$

$T = \{a, b\}$

$P = \{ S \to aS$

$S \to bs$

$S \to aB$

$B \to aC$

$C \to aD$

$D \to a \}$

S is a start symbol.

Some Important closure properties of Regular languages

Closure under

* Union
* concatenation
* Kleene star
* Complement
* Intersection
* difference
* reverse
* letter substitution.

Theorem: The regular languages are closed under Union, concatenation & Kleene star.

Proof: closure under union, concatenation & star

If L_1 & L_2 are regular, the $L_1 \cup L_2$, $L_1 . L_2$ & L_1^* are also denote the regular languages

Theorem: The regular languages are closed under Union, concatenation & Kleene star.

Proof: It is given that L_1 & L_2 are regular languages. So there exists regular expression R_1 & R_2 such that:

$$L_1 = L(R_1)$$
$$L_2 = L(R_2)$$

By the definition of regular expressions, we have

* $R_1 + R_2$ is a regular expression denoting the language $L_1 \cup L_2$.
* $R_1 \cdot R_2$ is a regular expression denoting the language $L_1 \cdot L_2$.
* R_1^* is a regular expression denoting the language L_1^*.

So, the regular languages are closed under Union, concatenation & star operation.

Theorem: If L & M are regular languages, then so is $L \cap M$.

Proof: Let L & M be the languages of automata $A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$ & $A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$ & we are assuming that the alphabets of both automata are same.

For $L \cap M$ we shall construct an automata A that simulates both A_L & A_M. The states in A

are pairs of states, the first from A_L & the second from A_m. To design the transitions of A, Suppose A is in state (p, q) where p is the state of A_L & q is the state of A_m. If a is the input symbol, we see what A_L does on input a, say it goes to state s. We also see what A_m does on input a, say it makes a transition to state t. Then the next state of A will be (s, t). In that manner A has simulated the effect of both A_L & A_m

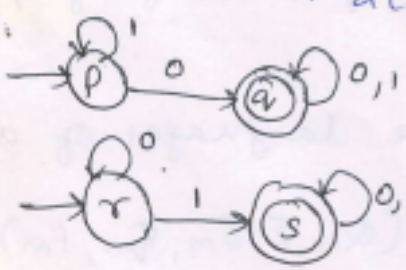
The start state of A is the pair of start states of A_L & A_m. Since we want to accept if & only if both automata accept, we select as the accepting states of A all those pairs (p, q) such that p is an accepting state of A_L & q is an accepting state of A_m.

Formally we define

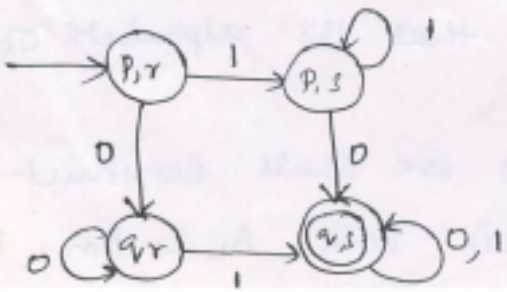
$$A = (Q_L \times Q_m, \Sigma, \delta, (q_L, q_m), F_L \times F_m)$$

The w is accepted by A if & only if both A_L & A_m accept w. Thus A accepts the intersection of L & M

Ex



Soln.



Problems on proving languages not regular

Eg Prove that $a^n b^n$ is not regular

Sol: Assume that given language is regular

Let length of $w = 2n$

$\therefore |w| = 2n > n$

We can split the string w into xyz such $|xy| \leq n$ & $|y| \geq 1$ as shown below.

$w = a^{n-1} a\ b^n$

$x\ \ y\ \ z$

By pumping lemma $xy^k z \in L$ for $k \geq 0$

i.e $a^{n-1} a^{(k)} b^n \in L$ for $k \geq 0$

If we select $k=0$ the language generated will have $(n-1)$ a's followed by n b's which is contradiction to the assumption that language is regular. So the given language is not regular.

Eg Show that $L = \{a^i b^j \mid i > j\}$ is not regular.

Sol^n: Consider the string $w = a^{n+1} b^n$

$|w| = 2n+1 \geq n$

$w = a^{n+1} b^n = a^n a\ b^n = a^{n-1} a\ a b^n$

$x\ \ y\ \ z$

$|xy| \leq n$

$n \leq n$

$k=0$

$w = a^{n-1}a^k ab^n$

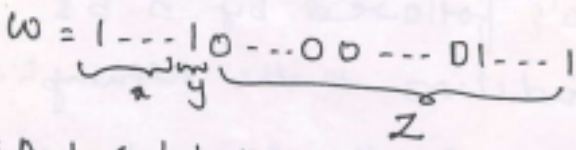
$= a^{n-1}ab^n \notin L$

If we select $k=0$ the language generated will have Equal no of a's & b's which is contradiction to the assumption that language is Regular. So the given language is not Regular.

(Eg) Show that $L=\{ww^R / w\in(0+1)^*$ is not regular$\}$

Sol:

$w = 1....10...0 \quad w^R = 0...01...1$



$|x| = n-1$ & $|y|=1$ so that $|xy| = |x|+|y| = n-1+1 = n$

If $k=0$, y does not appear & so the number of 1's on the left of w will be less than the no of 1's on the right of w & so the string is not of the form ww^R. So $xy^kz \notin L$.

So the language $L=\{ww^R / w\in(0+1)^*\}$ is not Regular.

(Eg) Prove that the following languages are not regular

(i) $L=\{a^{i^2} / i\geq 1\}$ (ii) $L=\{a^p / p$ is a prime$\}$

(iii) $L=\{0^i 1^i / i\geq 1\}$.

Proving languages not to be regular

A regular language is accepted by finite state machine, i.e for any regular language we can construct a finite state machine. But, for non regular languages it is not possible to have an equivalent Finite state machine which accepts only one regular languages. Using pumping lemma which uses pigeon hole principle one can easily check whether a language is regular or not.

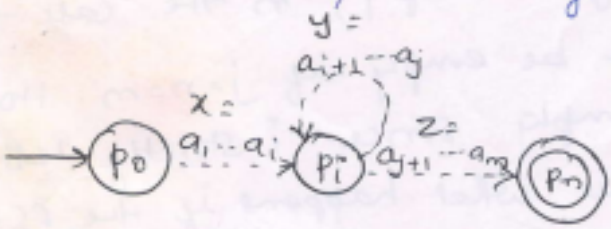
Theorem: (The pumping lemma for regular language) Let L be a regular language then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into three strings, $w = xyz$ such that

(1) $y \neq \varepsilon$

(2) $|xy| \leq n$

(3) For all $k \geq 0$, the string $xy^k z$ is also in L.

Proof:



Suppose L is regular. Then $L = L(A)$ for some DFA A. Suppose A has n states. Now consider any string w of length n or more, say $w = a_1 a_2 \cdots a_m$, where $m \geq n$ & each a_i is an input symbol. For $i = 0, 1, \cdots n$ define state p_i to be $\hat{\delta}(q_0, a_1, a_2 \cdots a_i)$, where δ is the transition fnc of A, & q_0 is the start state of A. That is, p_i is the state A is in

after reading the first i symbols of w

Note that $p_0 = q_0$.

By the pigeon hole principle, it is not possible for the $n+1$ different p_i's for $i = 0, 1, \ldots, n$ to be distinct, since there are only n different states. Thus we can find two different integers i & j with $0 \le i < j \le n$, such that $p_i = p_j$. Now we can break $w = xyz$ as follows.

(1) $x = a_1 a_2 \cdots a_i$

(2) $y = a_{i+1} a_{i+2} \cdots a_j$

(3) $z = a_{j+1}, a_{j+2} \cdots a_m$

That is, x takes us to p_i once, y takes us from p_i back to p_i & z is the balance of w. The relationship among the strings & states as suggested by figure above.

Note that x may be empty, in the case that $i = 0$. Also z may be empty if $j = n = m$. However y can not be empty, since i strictly less than j.

Now consider what happens if the FSM A receives the input $xy^k z$ for any $k \ge 0$. If $k = 0$ then the automaton goes from the start state q_0 to p_i on input x. Since p_i is also p_j, it must be that A goes from p_i to the accepting state shown in figure above on input z. Thus A accepts xz.

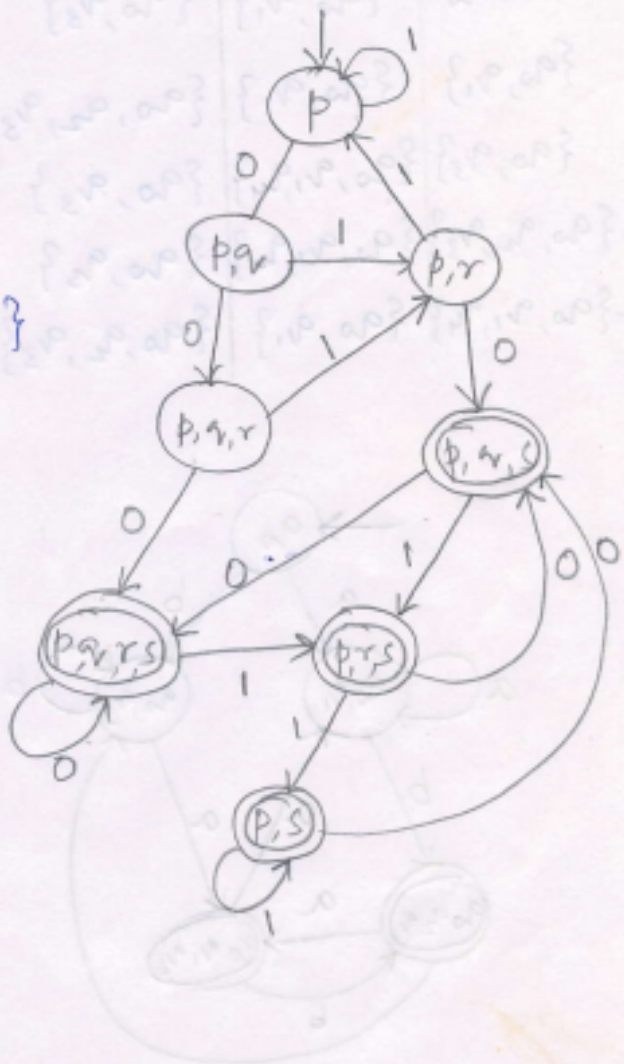
If $k > 0$, then A goes from q_0 to p_i on input x, circles from p_i to p_i k times on input y^k, & then goes to the accepting state on input z. Thus for any $k \ge 0$, $xy^k z$ is also accepted by A; that is $x y^k z$ is in

Eg Convert to a DFSM the following NDFSM using Lazy evaluation method.

δ	0	1
→ p	{p, q}	{p}
q	{r}	{r}
r	{s}	{∅}
* s	{s}	{s}

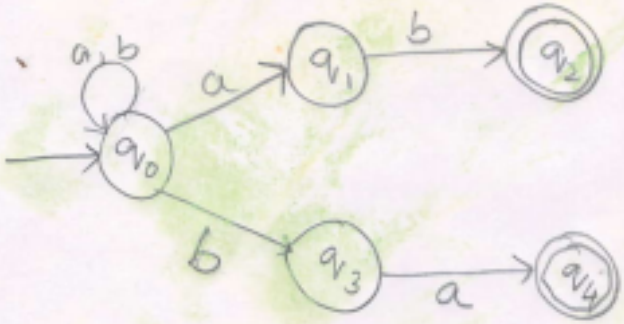
Soln

δ	0	1
→ {p}	{p, q}	{p}
{p, q}	{p, q, r}	{p, r}
{p, q, r}	{p, q, r, s}	{p, r}
* {p, q, r, s}	{p, q, r, s}	{p, r, s}
{p, r}	{p, q, s}	{p}
* {p, q, s}	{p, q, r, s}	{p, r, s}
* {p, r, s}	{p, q, s}	{p, s}
* {p, s}	{p, q, s}	{p, s}

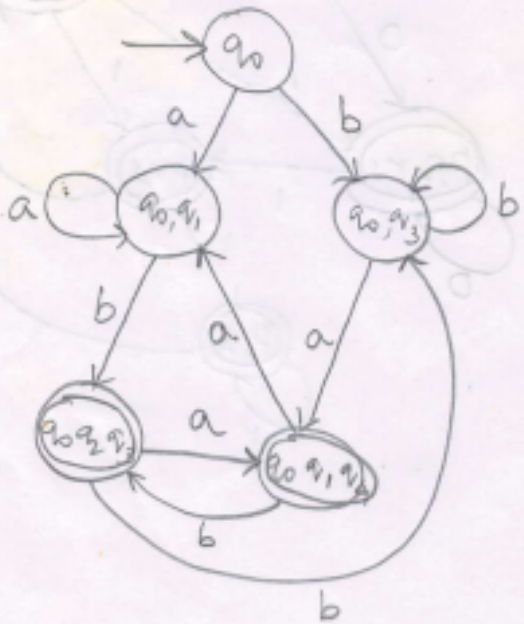


(Eg) Obtain an NDFSM to accept strings of a's & b's ending with ab or ba from this obtain an equivalent DFSM using lazy evaluation method.

Soln:



δ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_3\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2, q_3\}$
$\{q_0, q_3\}$	$\{q_0, q_1, q_4\}$	$\{q_0, q_3\}$
$*\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_4\}$	$\{q_0, q_3\}$
$*\{q_0, q_1, q_4\}$	$\{q_0, q_1\}$	$\{q_0, q_2, q_3\}$



Non Deterministic finite state Machine with Epsilon transition (ε-NDFSM)

The Non Deterministic finite state machine is 5 tuple or quin tuple indicating 5 components

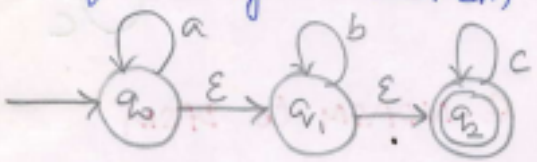
$$M = (Q, \Sigma, \delta, q_0, F)$$

where:

Q is finite set of states
Σ is the input alphabet.
$q_0 \in Q$ is the start state
$F \subseteq Q$ is the set of accepting states
δ is transition function. It maps from

$$Q \times \Sigma \cup \varepsilon \text{ to } 2^Q$$

Eg Convert the following ε-NDFSM to DFSM



Soln: Start State

$$\varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\} \text{ —— A}$$

Consider the state A

$\delta(A, a) = q_0 \cup \phi \cup \phi = q_0 = \varepsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\}$ — A
$\delta(A, b) = \phi \cup q_1 \cup \phi = q_1 = \varepsilon\text{-closure}(q_1) = \{q_1, q_2\}$ — B
$\delta(A, c) = \phi \cup \phi \cup q_2 = q_2 = \varepsilon\text{-closure}(q_2) = \{q_2\}$ — C

Consider the state B

$\delta(B, a) = \phi \cup \phi = \phi$
$\delta(B, b) = q_1 \cup \phi = q_1 = \varepsilon\text{-closure}(q_1) = \{q_1, q_2\}$ — B
$\delta(B, c) = \phi \cup q_2 = q_2 = \varepsilon\text{-closure}(q_2) = \{q_2\}$ — C

Finite Automata with Finite

Consider the state C

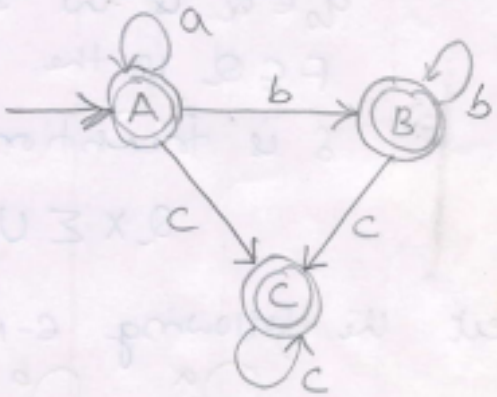
$\delta(C, a) = \phi$

$\delta(C, b) = \phi$

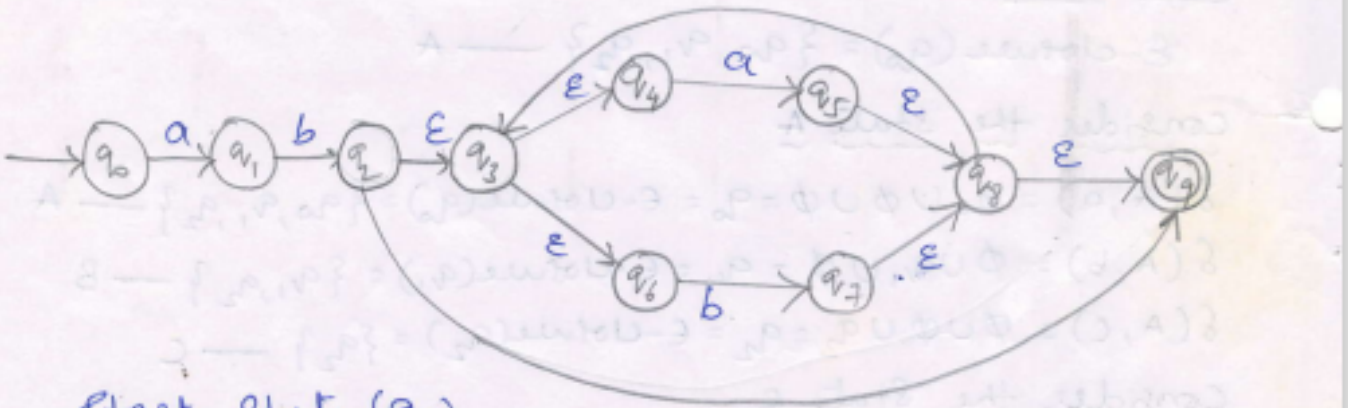
$\delta(C, c) = q_2 = \epsilon\text{-closure}(q_2) = q_2$ —— C

No more new states stops, construct transition table & DFSM

δ	a	b	c
→ * A	A	B	C
* B	ϕ	B	C
* C	ϕ	ϕ	C



(Eg) Convert the following ϵ-NDFSM to DFSM



Soln Start state (q_0)

$\epsilon\text{-closure}(q_0) = q_0$ —— A

Consider the state A

$\delta(A, a) = q_1 = \epsilon\text{-closure}(q_1) = q_1$ —— B

$\delta(A, b) = \phi$

Consider the state B:

$\delta(B, a) = \emptyset$

$\delta(B, b) = q_2 = \epsilon\text{-closure}(q_2) = \{q_2, q_3, q_4, q_6, q_9\}$ — C

Consider the state C:

$\delta(C, a) = q_5 = \epsilon\text{-closure}(q_5) = \{q_5, q_8, q_3, q_4, q_6, q_9\}$ —

$\delta(C, b) = q_7 = \epsilon\text{-closure}(q_7) = \{q_7, q_8, q_9, q_3, q_4, q_6\}$ —

Consider the state D

$\delta(D, a) = q_5 = \epsilon\text{-closure}(q_5) = \{q_3, q_4, q_5, q_6, q_8, q_9\}$ —

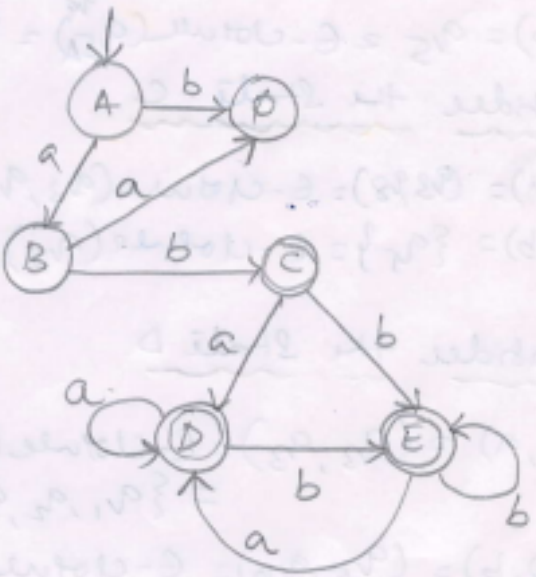
$\delta(D, b) = q_7 = \epsilon\text{-closure}(q_7) = \{q_3, q_4, q_6, q_7, q_8, q_9\}$ —

Consider the state E

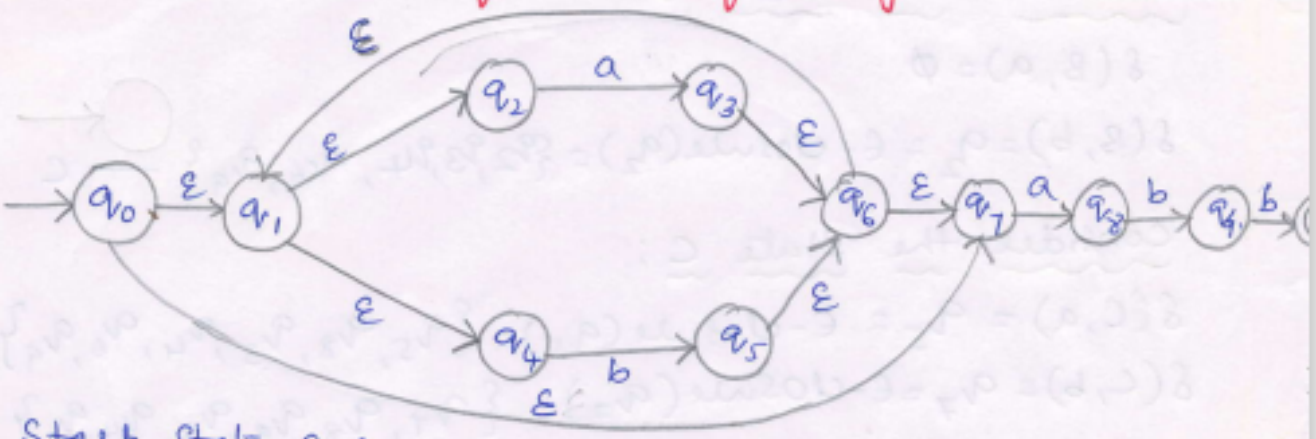
$\delta(E, a) = q_5 = \epsilon\text{-closure}(q_5) = \{q_3, q_4, q_5, q_6, q_8, q_9\}$ — D

$\delta(E, b) = q_7 = \epsilon\text{-closure}(q_7) = \{q_3, q_4, q_6, q_7, q_8, q_9\}$ — E

δ	a	b
→ A	B	$\emptyset$
B	$\emptyset$	C
* C	D	E
* D	D	E
* E	D	E



(Eg) Obtain the DFSM from the following ϵ-NDFSM



Soln Start State (q_0)

ϵ-closure(q_0) = {q_0, q_1, q_2, q_4, q_7} — A

Consider the State A

$\delta(A, a) = \{q_3 q_8\}$ = ϵ-closure($q_3 q_8$) = {$q_1, q_2, q_3, q_4, q_6, q_7, q_8$} — B

$\delta(A, b) = q_5$ = ϵ-closure(q_5) = {$q_5, q_6, q_7, q_1, q_2, q_4$} — C

Consider the State B

$\delta(B, a) = \{q_3, q_8\}$ = ϵ-closure(q_3, q_8)

= {q_1, q_2, q_3, q_4, q_6} — B

$\delta(B, b) = q_5$ = ϵ-closure(q_5) = {$q_5, q_6, q_7, q_1, q_2, q_4, q_9$} — D

Consider the State C

$\delta(C, a) = (q_3 q_8)$ = ϵ-closure(q_3, q_8) = {$q_1, q_2, q_3, q_4, q_6, q_7, q_8$} — B

$\delta(C, b) = \{q_5\}$ = ϵ-closure(q_5) = {$q_1, q_2, q_4, q_5, q_6, q_7$} — C

Consider the State D

$\delta(D, a) = (q_3, q_8)$ = ϵ-closure(q_3, q_8)

= {$q_1, q_2, q_3, q_4, q_6, q_7, q_8$} — B

$\delta(D, b) = (q_5, q_{10})$ = ϵ-closure(q_5, q_{10})

= {$q_1, q_2, q_4, q_5, q_6, q_7, q_{10}$} — E

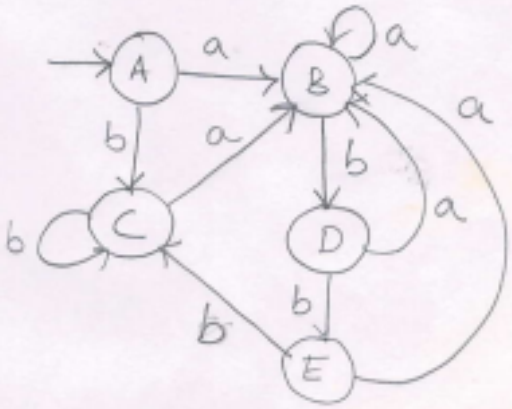
Consider the state E :

$\delta(E, a) = \{q_3, q_8\} = \epsilon\text{-closure}(q_3, q_8)$

$= \{q_1, q_2, q_3, q_4, q_7, q_8\}$ —— B

$\delta(E, b) = \{q_5\} = \epsilon\text{-closure}(q_5) = \{q_1, q_2, q_4, q_5, q_6, q_7\}$ —

δ	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



Machine based hierarchy of language classes not attached