Non-Deterministic Finite Automata (NFA)

* An NFA is a TG with a unique start state with the property that each of its edge labels is a single alphabet letter.

Definition

An NFA is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where

Q: is a set of finite states

Σ: is a finite set of symbols called (Alphabet)

$q_0 \in Q$ a distinguished state the start state

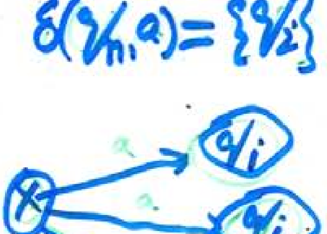
F: a subset of Q called the final or accepting states

and

δ: is a total function from $Q \times \Sigma$ to $P(Q)$ known as transition function such that an input symbol may cause more than one next state i.e. to one state out of a set of possible next states.

$P(Q)$ is the power set of Q, 2^Q

$\delta(q_n, a) = \{q_i\}$



$\delta(q_n, a) = \{q_i, q_j, q_k\}$

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

(12) * A state may have more than one outgoing edges labelled with the same symbol.

*

* NFA extend the Model of DFA.

Definition

The language of an NFA M, denoted $L(M)$, is the set of strings accepted by the M. That is,

$L(M) = \{ w \mid \text{there is a computation } [q_0, w] \overset{*}{\vdash} [q_i, \lambda] \text{ with } q_i \in F \}$

$\Rightarrow$ That is, as there is possibility of multiple paths for a given string of input, If at least one path exists that leads from the start state to a final state, the String is accepted as from the $L(M)$.

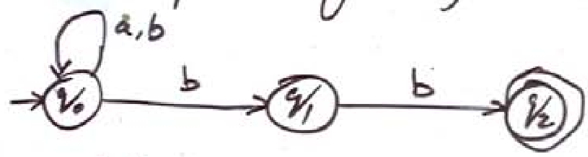
EX An NFA M is given, show computations for string ababb.

$M: \; Q = \{q_0 \; q_1 \; q_2\}$

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

$F = \{q_2\}$

input = ababb



δ	a	b
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	ϕ	$\{q_2\}$
q_2	ϕ	ϕ

$[q_0, ababb]$
$\vdash [q_0, babb]$
$\vdash [q_0, abb]$
$\vdash [q_0, bb]$
$\vdash [q_0, b]$
$\vdash [q_0, \lambda]$

not accepted

$[q_0, ababb]$
$\vdash [q_0, babb]$
$\vdash [q_1, abb]$

machine Crashes

$[q_0, ababb]$
$\vdash [q_0, babb]$
$\vdash [q_0, abb]$
$\vdash [q_0, bb]$
$\vdash [q_1, b]$
$\vdash [q_2, \lambda]$

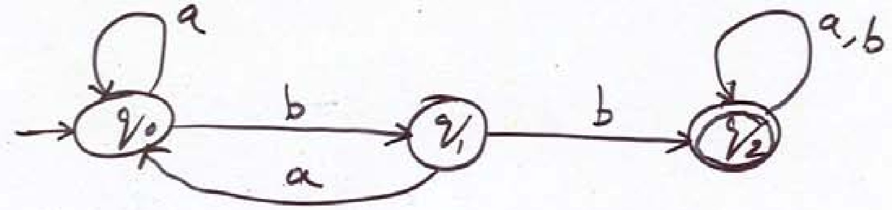
Accepted

$(a+b)^* bb$

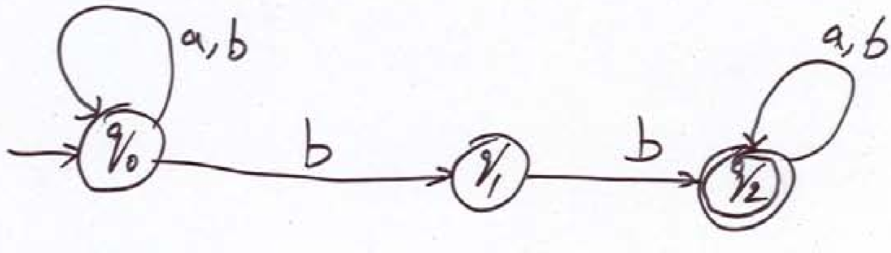
NFA and DFA

F.As M_1 and M_2 accept $(a+b)^*bb(a+b)^*$

M_1 :



M_2 :



M_1 is a DFA and M_2 is an NFA

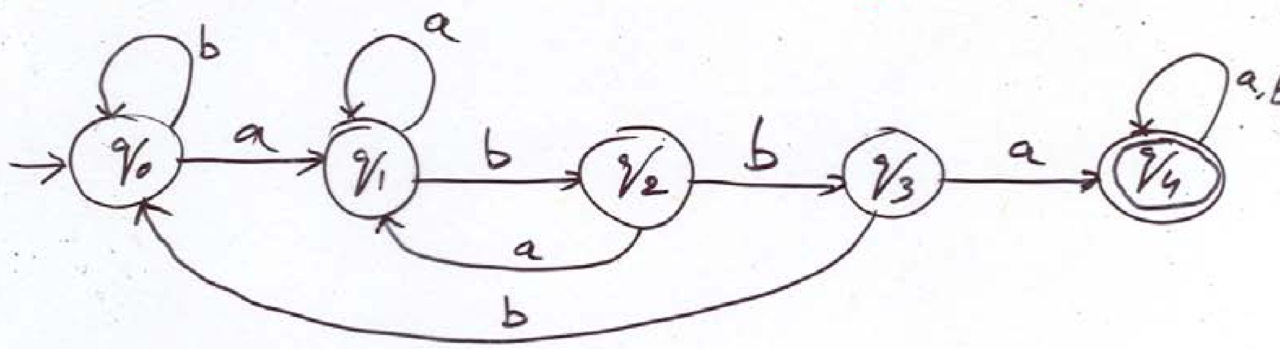
M_1 enters acceptance state q_2 on first occurance of bb.

M_2 can enter the acceptance state upon processing any occurance of bb.

Machines to accept strings containing substring abba

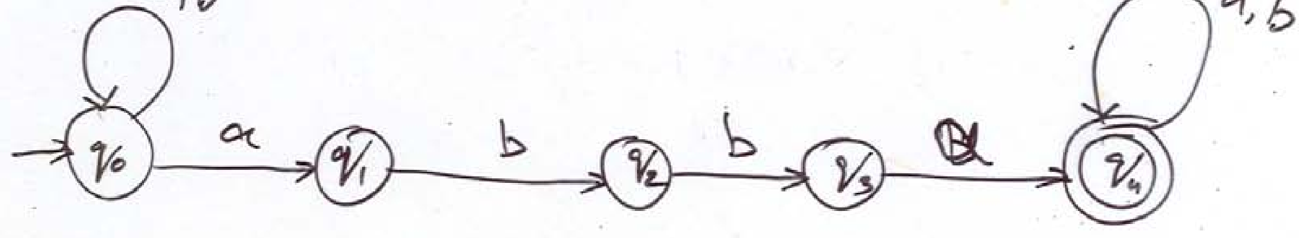
M_1:

DFA



M_2:

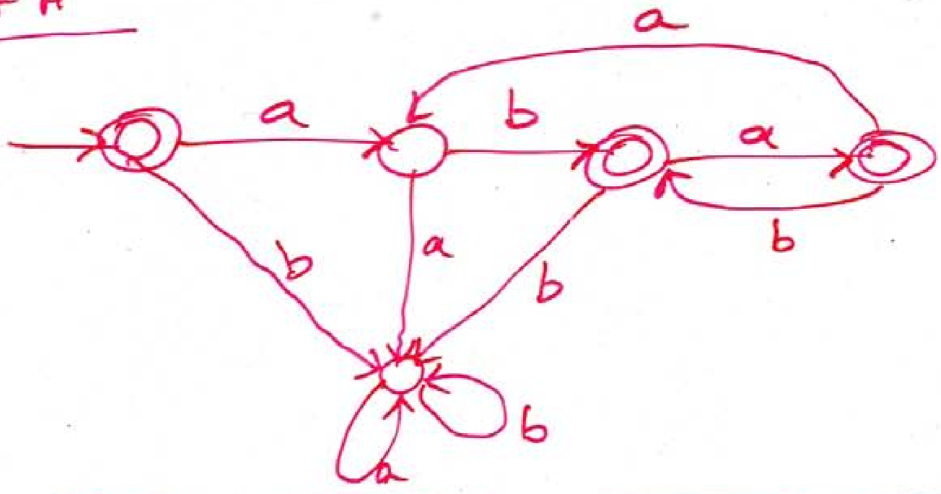
NFA



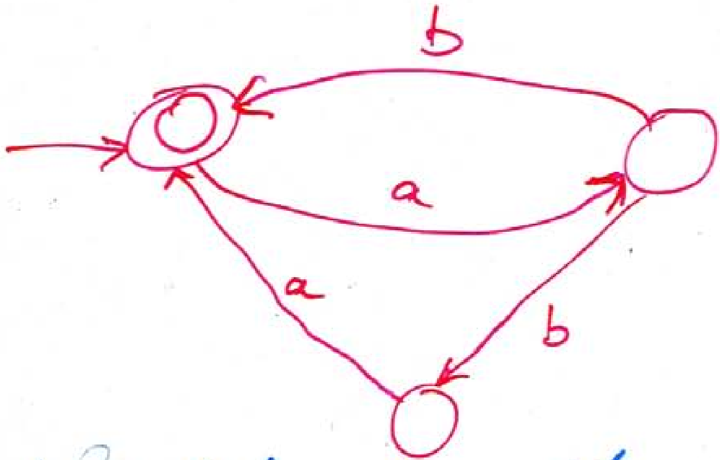
Some kind of guidence (human) is required to select the right moment to make a move along path abba..

$L = (ab + aba)^*$

DFA

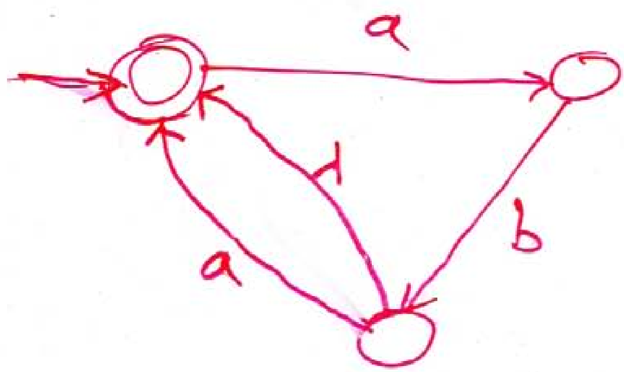


NFA



A Simple Automata, it is more natural.

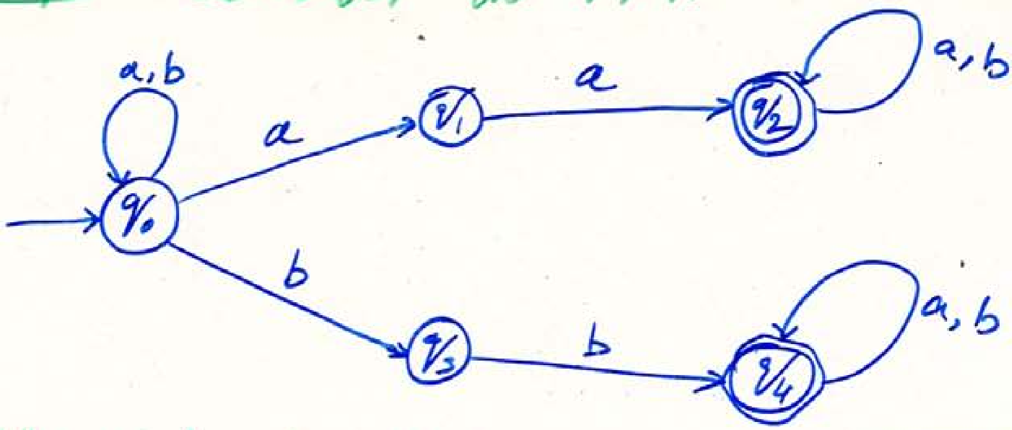
λ-NFA



Even much Simpler Automata.

→ NFA is more powerful notation, useful generalization of FA, as is greatly simplify the description of FA.

Example Consider the NFA



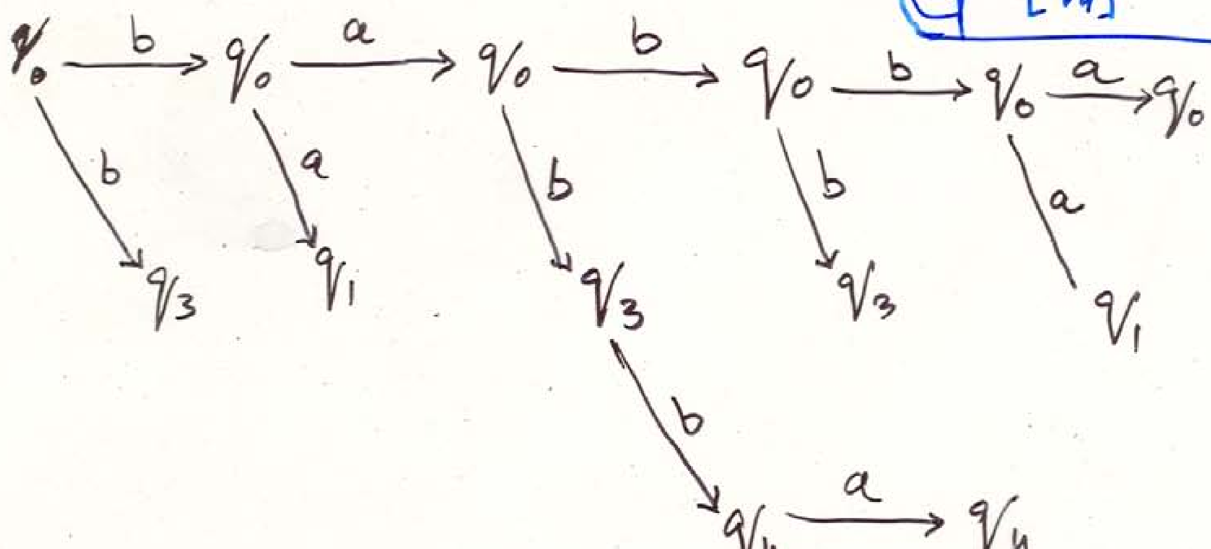
Let the input : babba

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

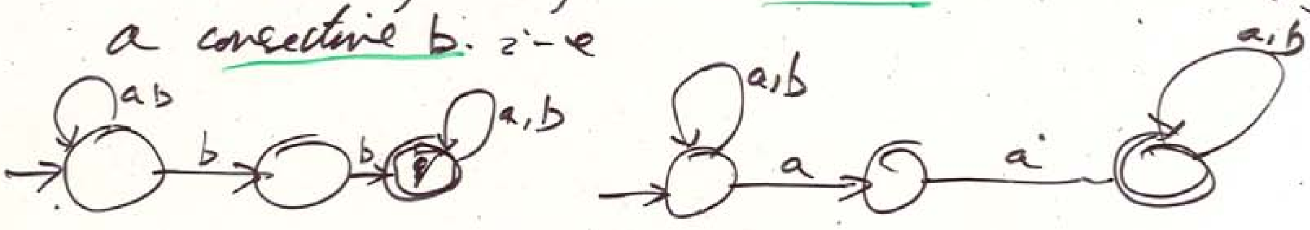
Hence the sequence of states for input babba is

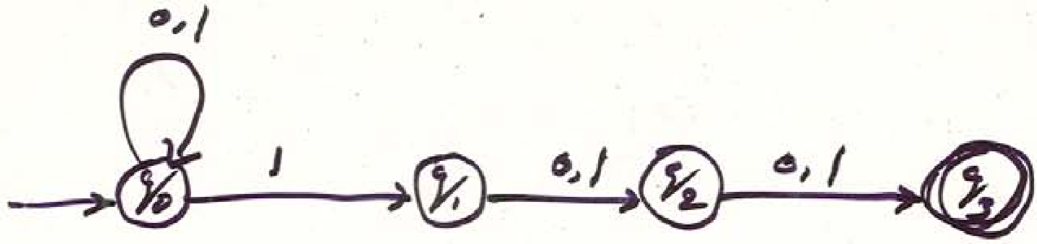
$$\delta(q_0, babba) = \{q_0, q_3, q_4\}$$

Try: abbaab

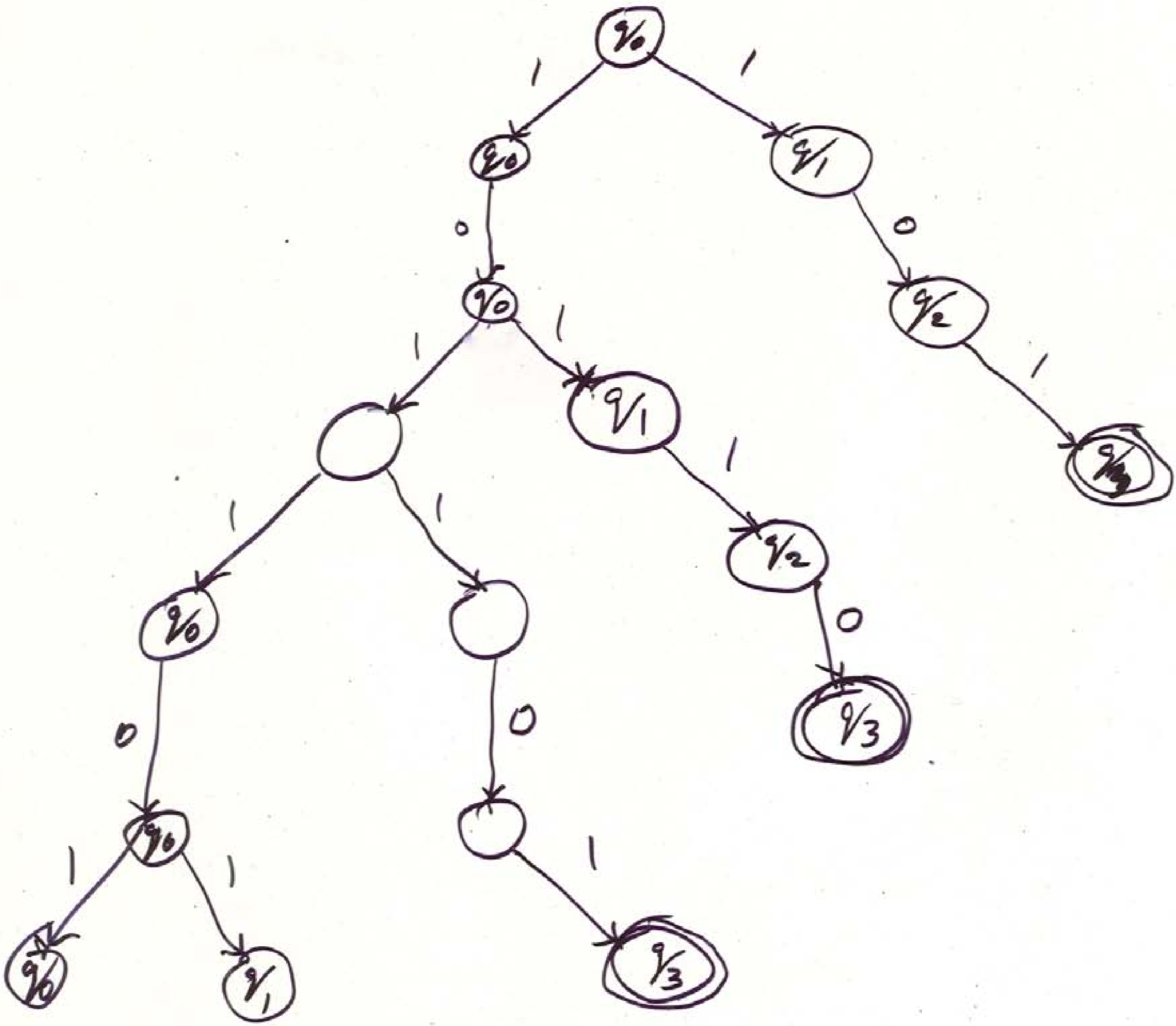


Remarks :
- NFA accepts all string with either two consecutive a's or two consecutive b's.
- Above machine is a union of two machines one recognizing string with double a and other recognizing a consecutive b. i-e





input string 101101



NFAs with Λ-moves / lambda transitions
NFA-λ empty moves / null string transitions

An NFA with lambda transitions is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q, δ, q_0 and F are same as in an NFA. The transition function is a function from $Q \times (\Sigma \cup \{\lambda\})$ to $P(Q)$, that is,

⇒ NFA may contain emty moves from a state to other states. No input symbol is consumed on an emty move.

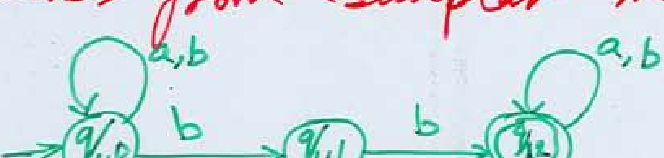
⇒ The definition of machine halting must be extended to include the possibility that a computation may continue using lambda moves after the input string has been completely processed. Ex

a^*b^*

⇒ Lambda-transitions can be used to build complex machines from simplar machines.

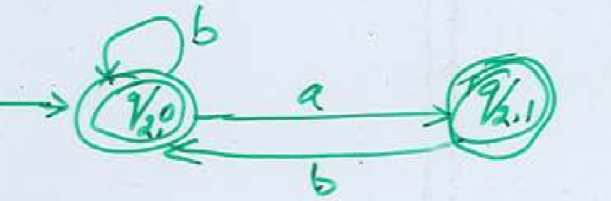
Let

M_1:



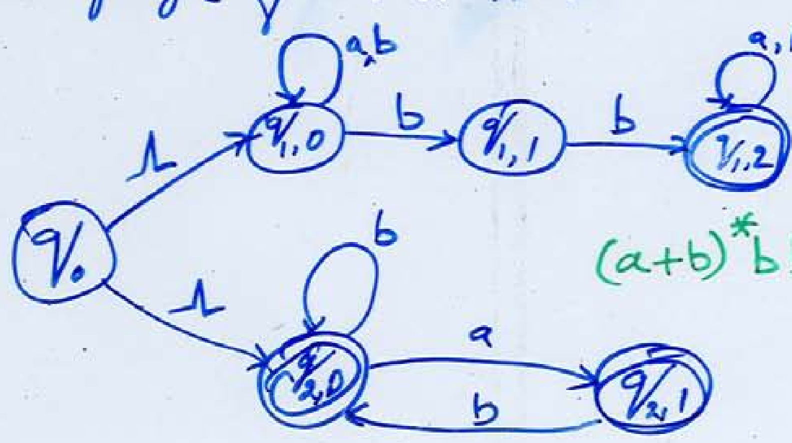
$(a+b)^* bb (a+b)^*$

M_2:



$(b+ab)^*(a+\Lambda)$

The language of NFA-λ M is $L(M_1) \cup L(M_2)$.



$(a+b)^* bb (a+b)^* + (b+ab)^*(a+\Lambda)$

(16)

National University
Of Computer & Emerging Sciences
Telecom Engineering Department-Islamabad Campus

Inter Office Memo

From Telecom Engineering Department Ref. No TE/DIR/08-08/053
To Director, Islamabad Campus Date 28/08/2008
Subject: Attendant for Graduate Block

Department of Telecom Engineering has established labs in graduate block. To properly look after the graduate block the department requires the services of fulltime attendant.

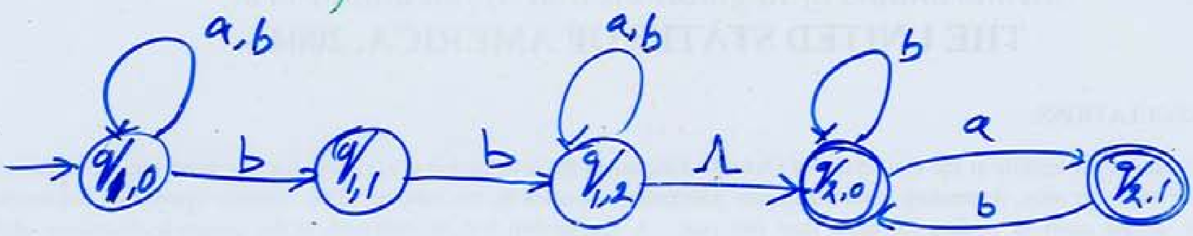
Thank You.

Head of Department

Dept. Sec

Example Continued.

The language of the following machine M is concatenation of the $L(M_1)$ and $L(M_2)$, it is constructed by joining the two machines with a λ move.

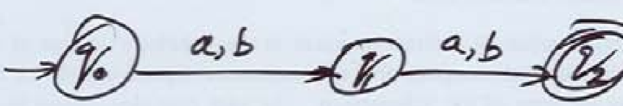


$$(a+b)^* bb(a+b)^*(b+ab)^*(a+\lambda)$$

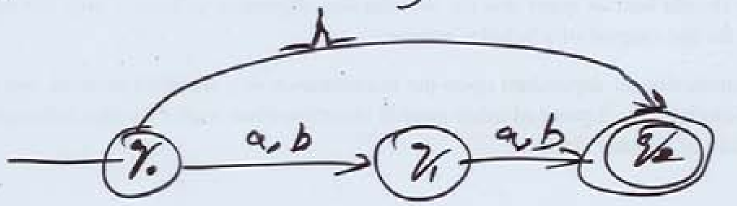
Example:

Lambda transition are used to construct an NFA-λ that accepts all strings of even length over $\{a, b\}$

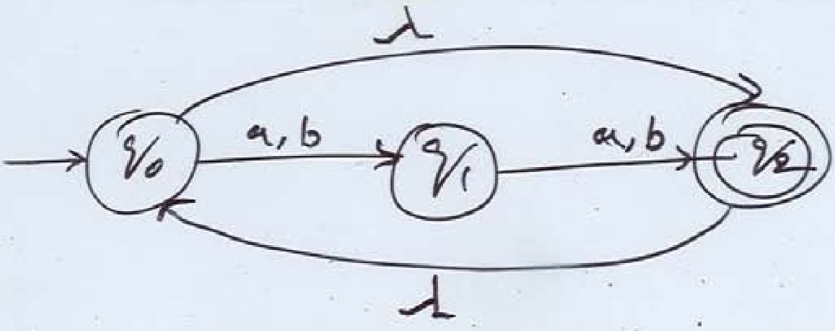
- First we construct a machine that accepts strings of length two.



- To accept the null string, a lambda arc is added from q_0 to q_2:



- strings of any positive even length are accepted by following a lambda arc from q_2 to q_0 to repeat the sequence q_0, q_1, q_2



Theorem generalizing the previous Examples.

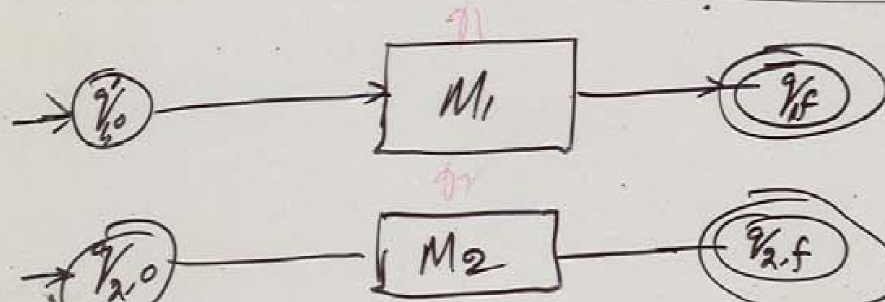
Let M_1 and M_2 be two NFA-λ's. The are NFA-λ's that accept $L(M_1) \cup L(M_2)$, $L(M_1)L(M_2)$ and $L(M_1)^*$.

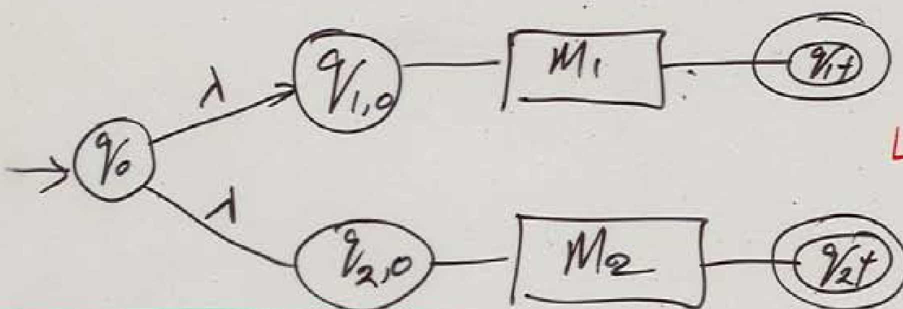
$\Rightarrow$ where for M_1, M_2 have unique q_0' start and final state q_f (which may be added) such that.

(i) The in-degree of the start state q_0' is zero.

(ii) The only accepting state is q_f.

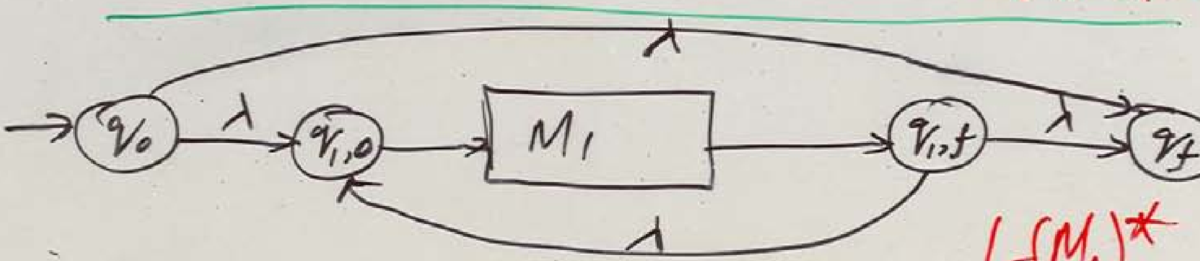
(iii) The out degree of q_f is zero.





$L(M_1) \cup L(M_2)$

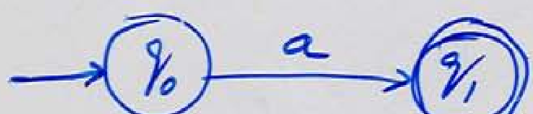
$L(M_1)L(M_2)$



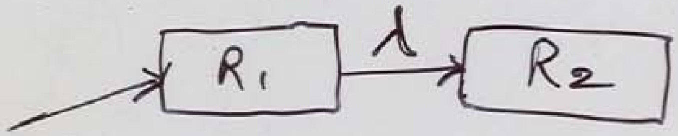
$L(M_1)^*$

Conversion of Regular Expressions to an NFA

- The regular Expression 'a' (a single token) is represented by the NFA

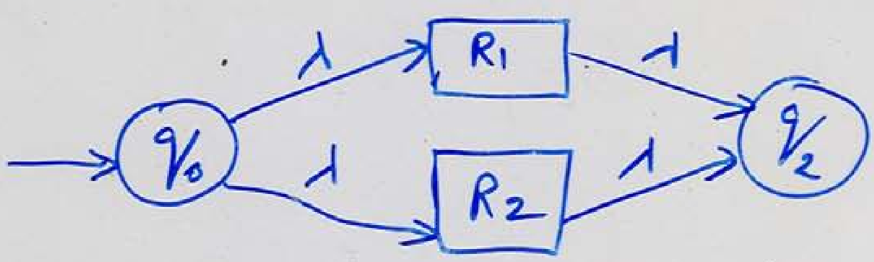


- The regular expression $R_1 R_2$ (the regular exp R_1 followed by R_2) is represented by diagram as:

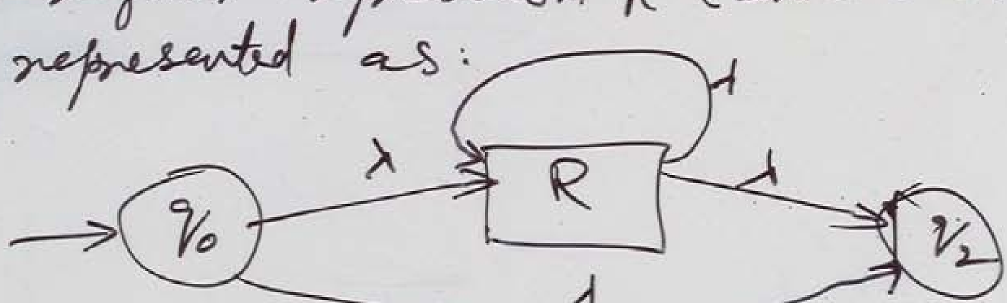


The box enclosing R_1 represents NFA that recognizes the regular exp R_1. Final state of R_2 is overall final state.

- The regular expression $R_1 + R_2$ (alternation) is represented as:

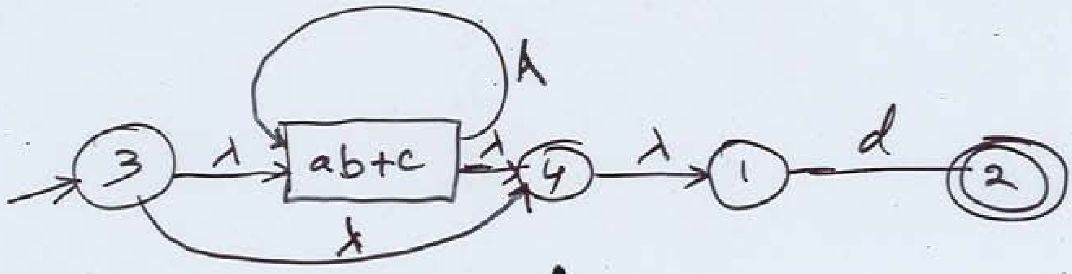


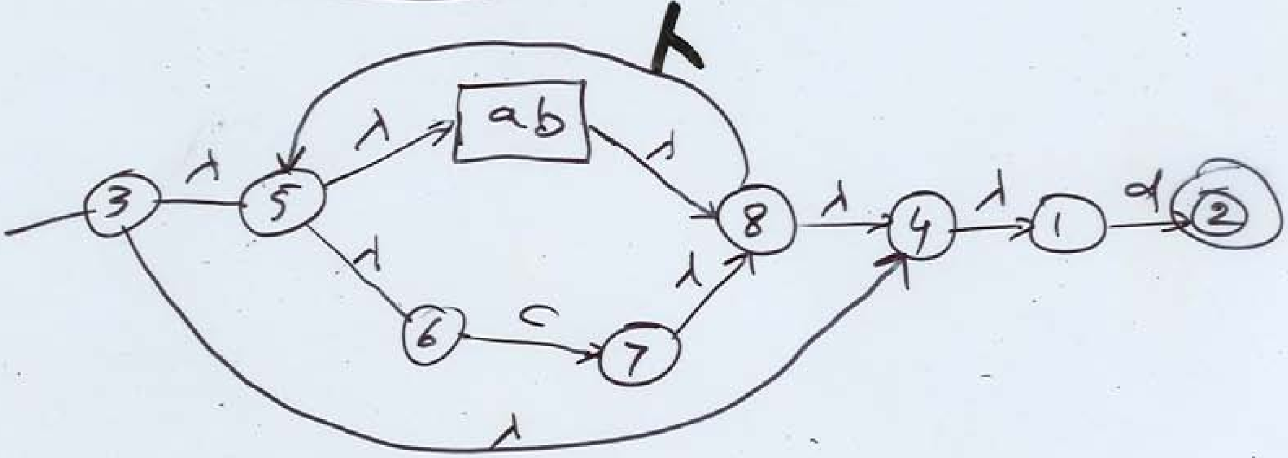
- The regular expression R^* (zero or more repetitions) is represented as:

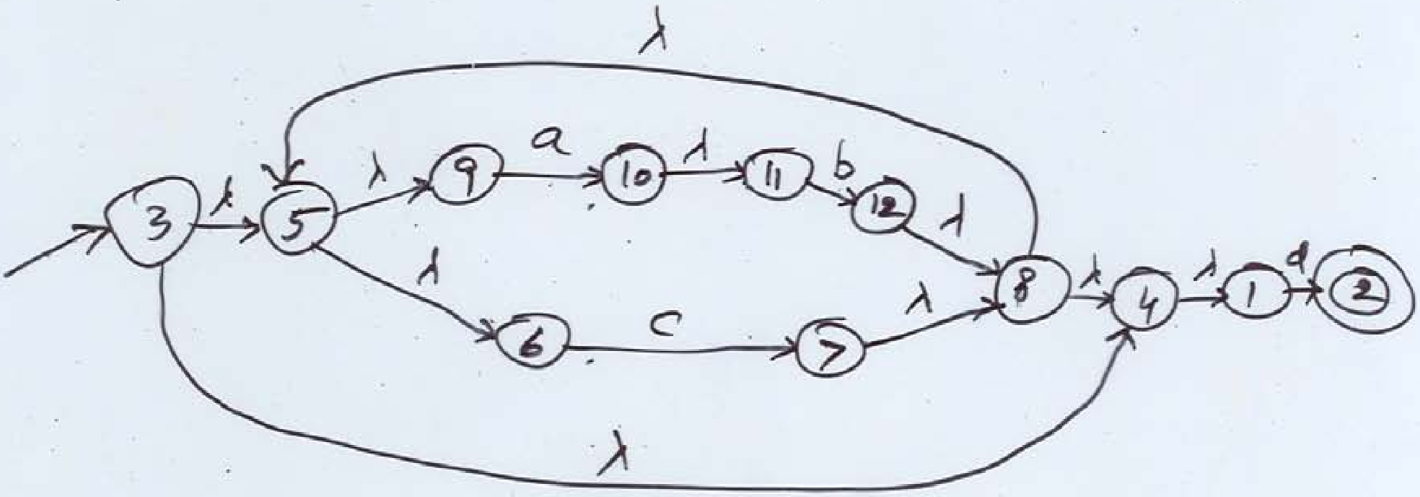


Example : (Reguler expression ⟶ NFA)

Let Regular expression : $(ab+c)^*d$







Conversion of NFA to DFA

Definition λ-closure

- The operation λ-closure of a state q of an NFA is the set of states, in that NFA, that can be reached from q on λ-transitions only.
- The λ-closure of a set of states T is the set of states that can be reached via λ-transitions from states that are member of T.
- It also follows that λ-closure of q must include q and λclosure of set T must include the members of T.

$$\lambda(q) = \{ p \in Q : (q, \lambda) \vdash_M^* (p, \lambda) \}$$

Algorithm

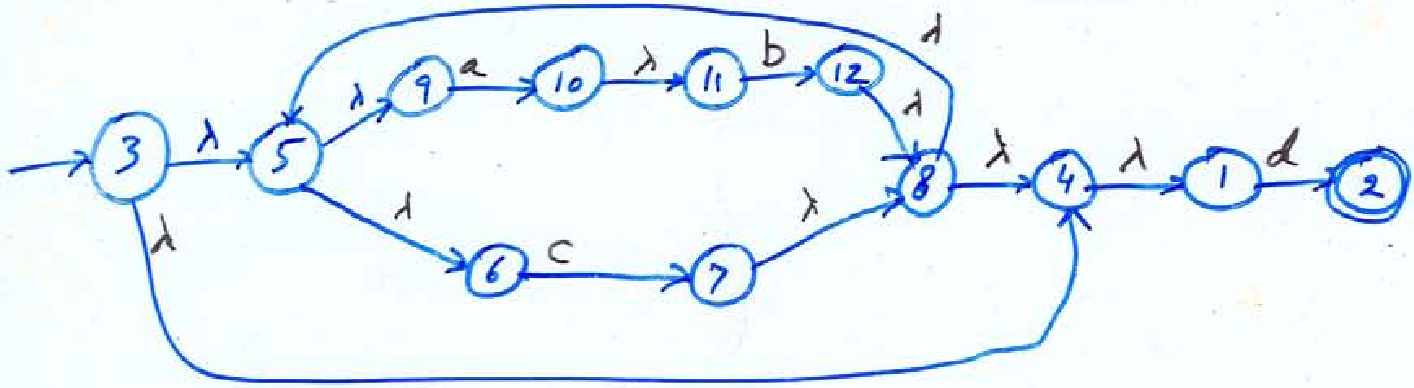
- Define the starting state of the DFA as being equivalent to the λ-closure of the start state of NFA.
- For each possible input symbol, a new set of states $\{t_1, t_2, \cdots t_m\}$ of the NFA is calculated as the λ-closure of all the states reachable from any of the states $\{q_0, q_1 \cdots q_n\}$ when that input symbol is received.

goto Set

- Each of the new set of states become the states of DFA and may be labled as B, C, D, and so on.
- The process is repeated until all λ-closure generated sets of states from NFA have been labeld.

(21) GotoSet $= \delta'(Q' \subseteq Q, a) = \bigcup \{ \lambda(p) : p \in Q \text{ and } (q, a, p) \in \delta \text{ for some } q \in Q' \}$

Example: consider the NFA generated for regular exp. $(ab+c)^*d$



(1) Starting state of NFA is 3.

λ-closure (3) = {3,5,4,9,6, 1} — label this state A in the Corresponding DFA

(2) States reachable from A when input is <u>a</u> = {10} ← goto set for a

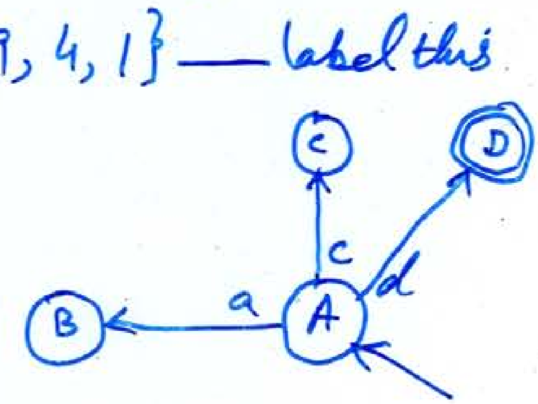
λ-closure of this state = {10, 11} — Label this state B.

(3) states reachable from A when input b = {φ} — the empty set can be ignored.

goto set for b

(4) states reachable from A when input is c = {7} ← goto set for c

λ-closure (7) = {7, 8, 5, 6, 9, 4, 1} — label this state as (C).



(5) States reachable from A on input d = {2}

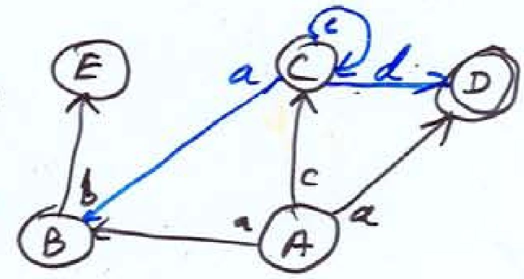
λ-closure of (2) = {2} — Label this state D, which is the final state of DFA

(6) The process now starts again with state B. $B = \{10, 11\}$

states reachable from B when a input $= \{\phi\}$

$\{12\}$ λ closure

(7) states reachable from B when b input $= \{12, 8, 5, 9, 6, 4, 1\}$
— label this state E.



(8) states reachable from B when C input $= \{\phi\}$

(9) states reachable from B when d input $= \{\phi\}$

(10) For state C: $C = \{7, 5, 8, 6, 9, 4, 1\}$

states reachable from C when a input $= \{10\}$

λ - closure of this set of states $= \{10, 11\}$ — this has already been labeled as B.

(11) states reachable from C when b input $= \{\phi\}$.

(12) states reachable from C when c input $= \{7\}$

λ - closure of (7) $= \{7, 8, 5, 6, 9, 4, 1\}$ — this has already been labeled as state C.

(13) state reachable from C when d input $= \{2\}$

λ - closure of this set of states $= \{2\}$ — this has already been labeled as D.

(14) For state D:

states reachable from D when a input $= \{\phi\}$.

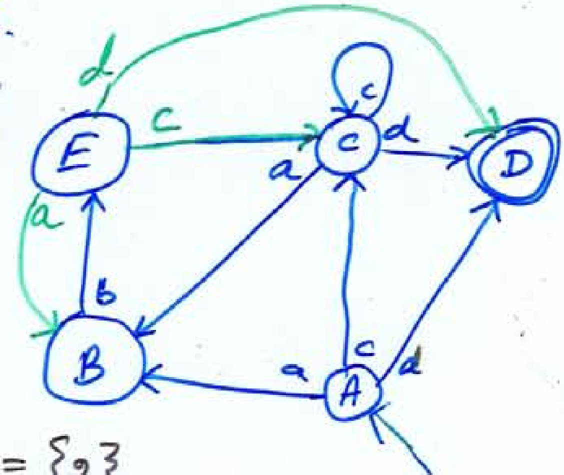
(15) state reachable from D when b input $= \{\phi\}$.

(16) state reachable from D when c input $= \{\phi\}$.

(17) state reachable from D when d input $= \{\phi\}$.

For state E: E = {12, 8, 5, 9, 6, 4, 1}

(18) States reachable from E when a input = {10}.
λ-closure of this set of states = B.

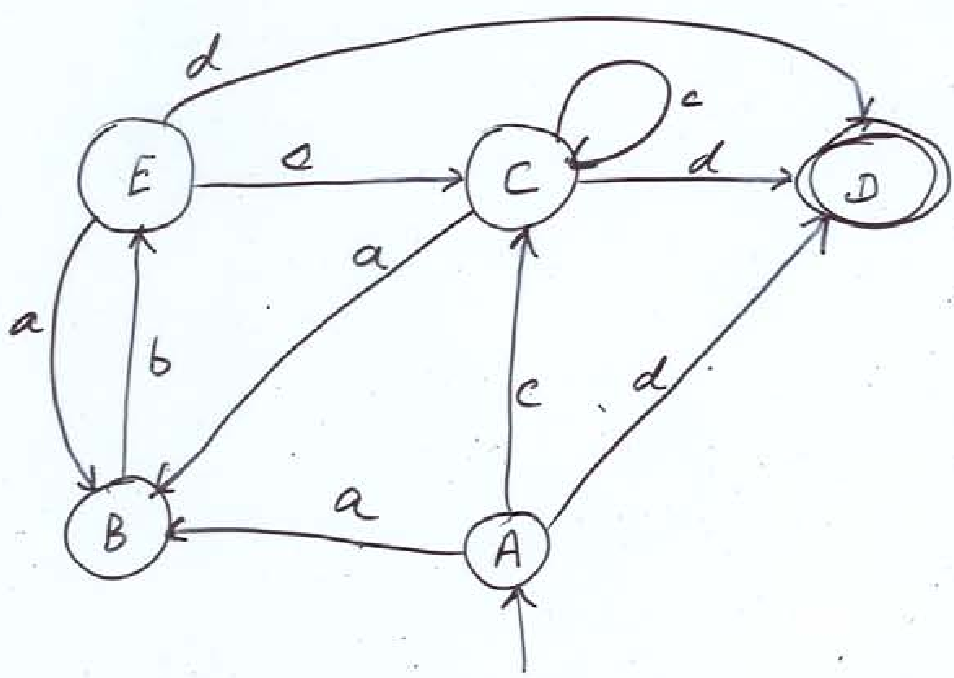


(19) states reachable from E when b input = {φ}.

(20) States reachable from E when c input = {7}
λ-closure of this set = C.

(21) states reachable from E when d input = {2}
λ-closure of this set = D.

The algorithm terminate here.



State	a	b	c	d
A	B	–	C	D
B	–	E	–	–
C	B	–	C	D
D	–	–	–	–
E	B	–	C	D

State minimization of a DFA

- To reduce the storage requirements for implementation of a DFA one must ensure that the DFA is the simplest possible for the given regular expression.

- The essential part of the algorithm operates repeatedly partitioning the set of states of original machine.

Algorithm:

1. The first step is to partition the set of states so that the finishing state(s) are in one group and all the other states are in another group. e.g. consider the DFA for $(ab+c)^*d$, original set (ABCDE) is partitioned as (ABCE) (D).

2. The next step is to apply partitioning procedure to each of these groups of states to produce new partition. This partitioning is repeated until no new partitions are produced and the final partition define the structure of the reduced machine.

3. Partitioning Procedure:
   - If a group of states consist of just one state, it cannot be subdivid
   - Each other group is taken in turn. Each group is partitioned into subgroups so that two states q_0 and q_1 are in the same subgroup, if, and only if, for all possible inputs symbols, q_0 and q_1 both have transitions on each input symbol to states in the same group of current partitioning

Example $(ab+c)^*d$

	a	b	c	d
A	B	–	C	D
B	–	E	–	–
C	B	–	C	D
D	–	–	–	–
E	B	–	C	D

(ABCDE)

(ABCE) (D)

$$P = \begin{bmatrix} A & B & C & E \\ B\text{-}CD & \text{-}E\text{--} & B\text{-}CD & B\text{-}CD \end{bmatrix} \quad Q = \begin{bmatrix} D \\ \text{----} \end{bmatrix}$$

The group the destination in each belong to:

$$\begin{bmatrix} A & B & C & E \\ P\text{-}PQ & \text{-}P\text{--} & P\text{-}PQ & P\text{-}PQ \end{bmatrix} \quad \begin{bmatrix} D \\ \text{----} \end{bmatrix}$$

By matching patterns of destination another partition is created.

$$\begin{bmatrix} A & C & E \\ P\text{-}PQ & P\text{-}PQ & P\text{-}PQ \end{bmatrix} \quad \begin{bmatrix} B \\ \text{-}P\text{--} \end{bmatrix} \quad \begin{bmatrix} D \\ \text{----} \end{bmatrix}$$

(ACE)(B)(D)

These groups can be labeled again as:

$$R = \begin{bmatrix} A & C & E \\ B\text{-}CD & B\text{-}CD & B\text{-}CD \end{bmatrix} \quad S = \begin{bmatrix} B \\ \text{-}E\text{--} \end{bmatrix} \quad T = \begin{bmatrix} D \\ \text{----} \end{bmatrix}$$

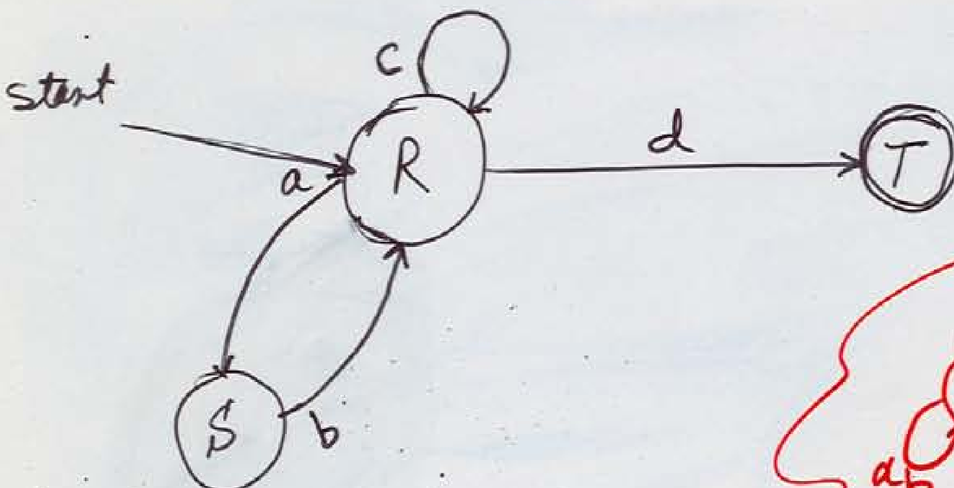
Partitioning Process reapplied.

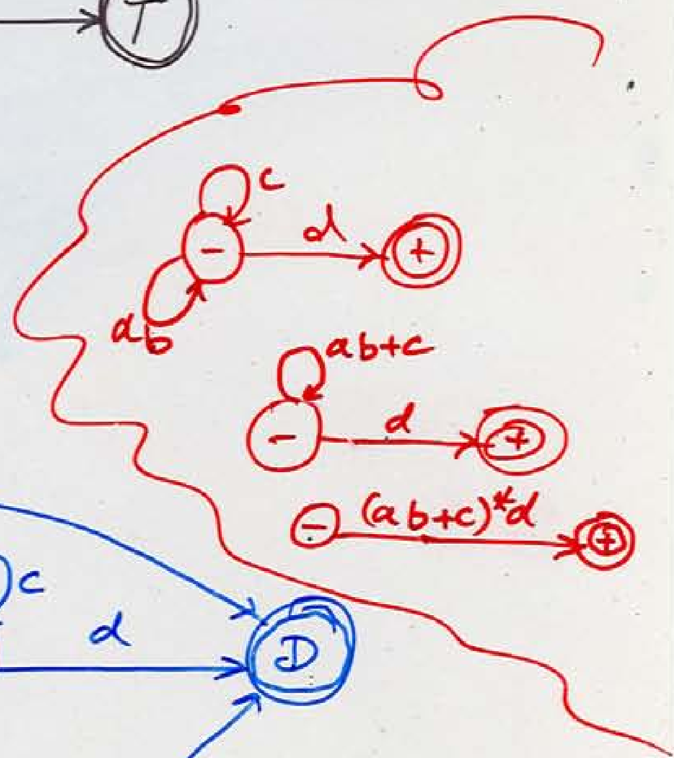
$$\begin{bmatrix} A & C & E \\ S\text{-}RT & S\text{-}RT & S\text{-}RT \end{bmatrix} \quad \begin{bmatrix} B \\ \text{-}R\text{--} \end{bmatrix} \quad \begin{bmatrix} D \\ \text{----} \end{bmatrix}$$

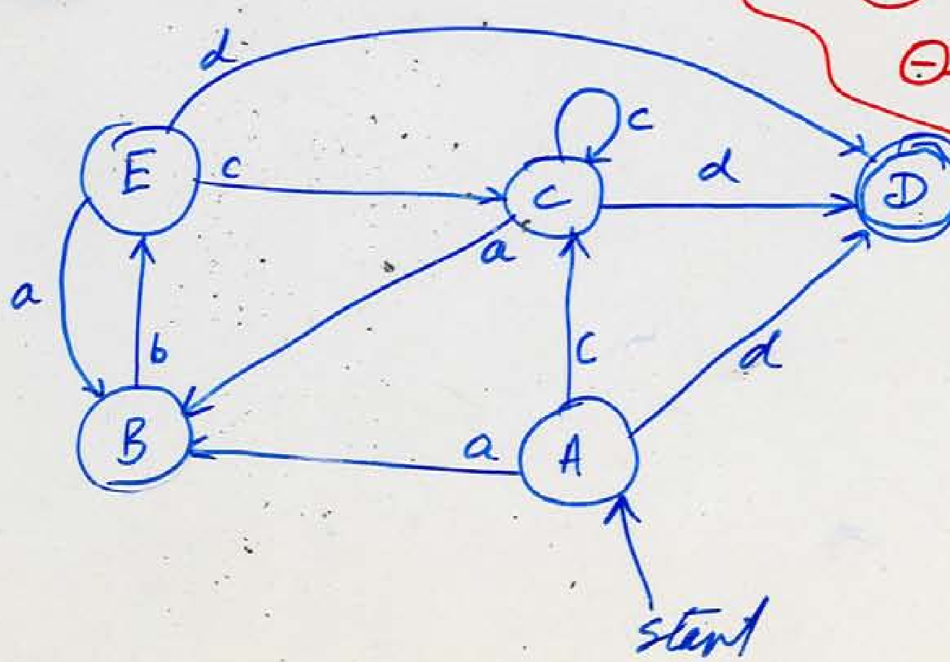
⇒ yielding the same partition as before, so the process
⇒ halts. This means the the states A, C, E of the original machine can be grouped together as a single state in the new machine while B. D remain same.

If the states in the new machine are labeled as R, S and T, corresponding to states (A,C,E) B and D respectively in old machine, The transition table for the new machine is as:

	a	b	c	d
R	S	–	R	T
S	–	R	–	–
T	–	–	–	–

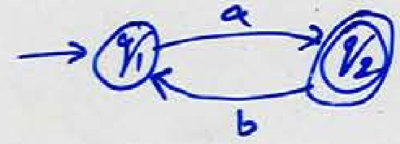






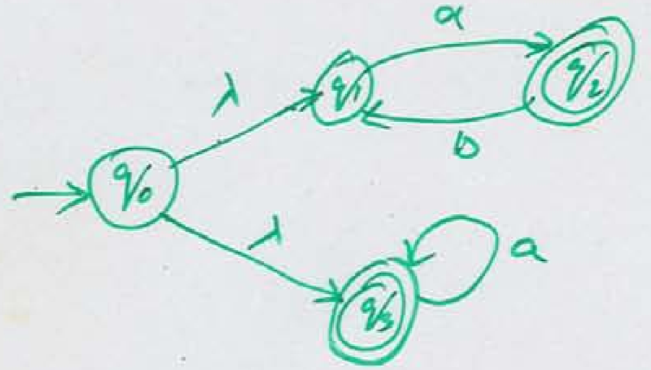
R.E.1 $\underline{a(ba)^*}$

R.E.2 $\underline{a^*}$

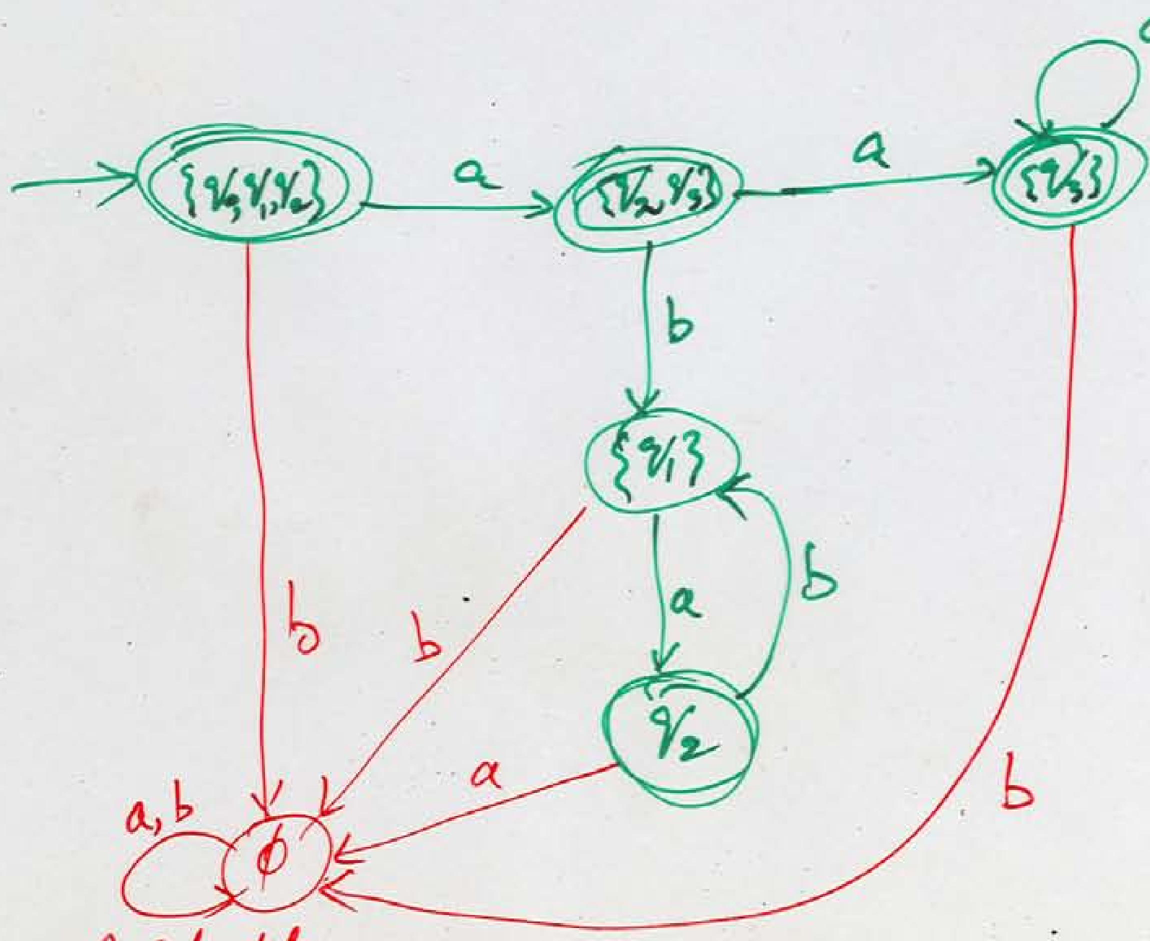
$M_1 =$ 

$M_2 =$ 

R.E. 3 $\underline{a(ba)^* + a^*}$

M_3: 

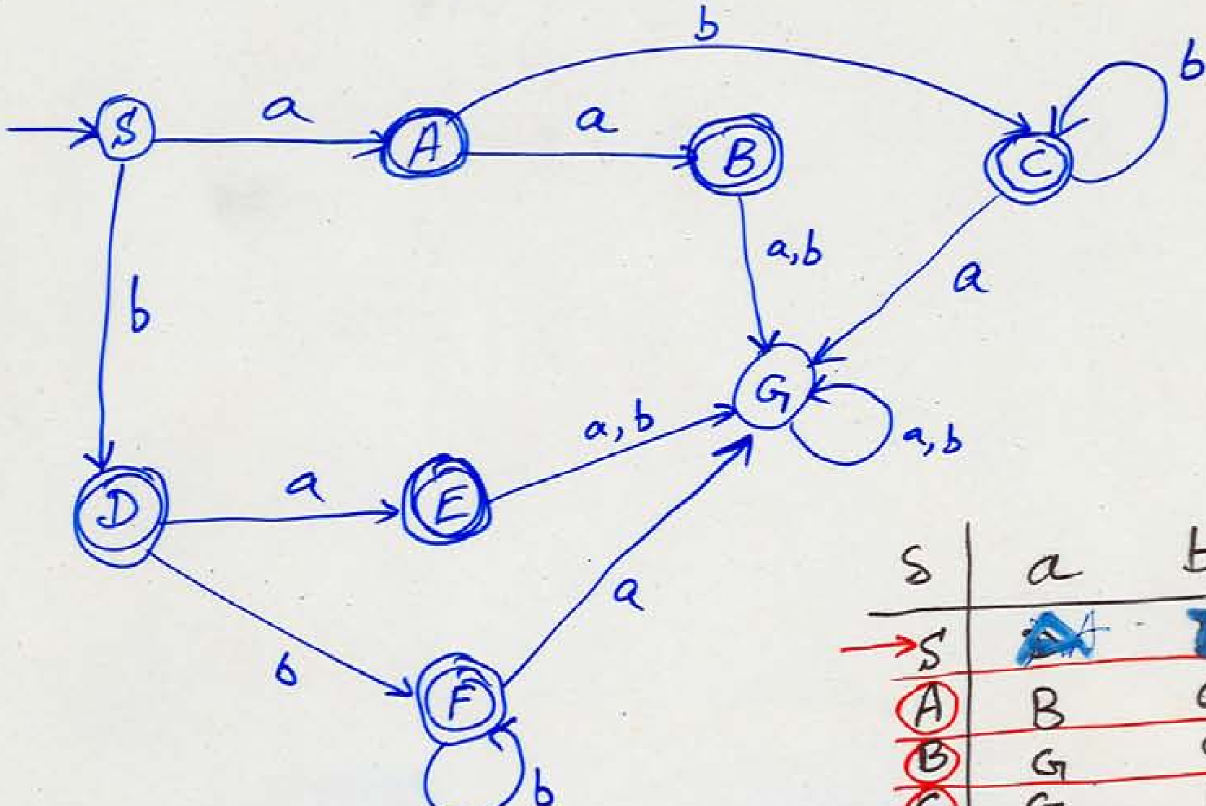
δ	a	b
q_0	$\{q_2, q_3\}$	$\emptyset$
q_1	$\{q_2\}$	$\emptyset$
q_2	$\emptyset$	$\{q_1\}$
q_3	$\{q_3\}$	$\emptyset$



Reject state

$(ba+bb)^* + (ab+aa)^*$

Construct an NFA for above R.E.



δ	a	b
→S	A	D
A	B	C
B	G	G
C	G	C
D	E	F
E	G	G
F	G	F
G	G	G

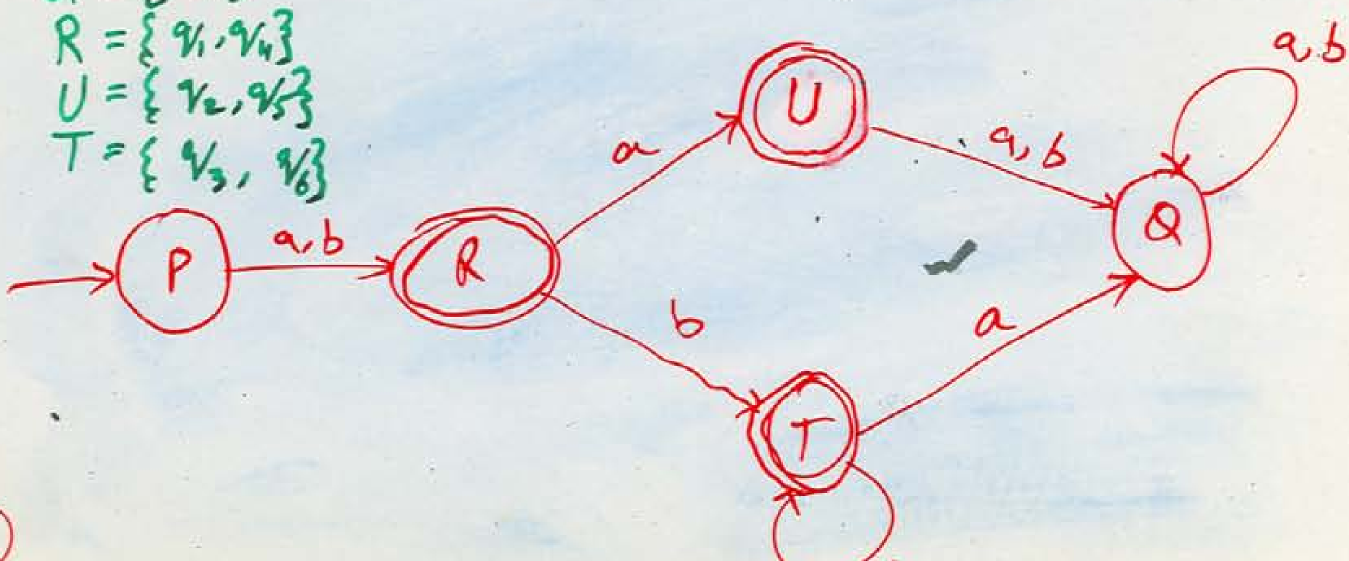
$P = \{S\}$

$Q = \{q_7\}$

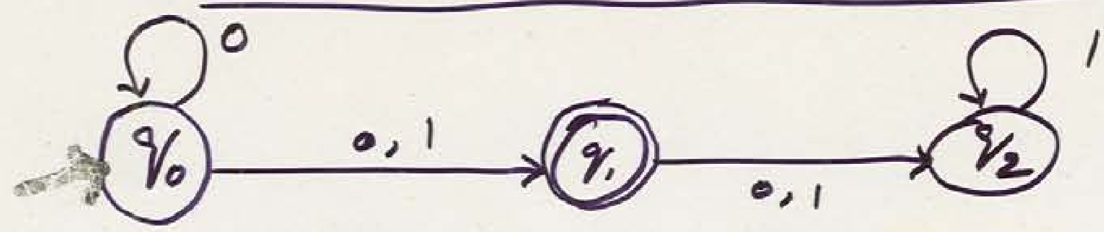
$R = \{q_1, q_4\}$

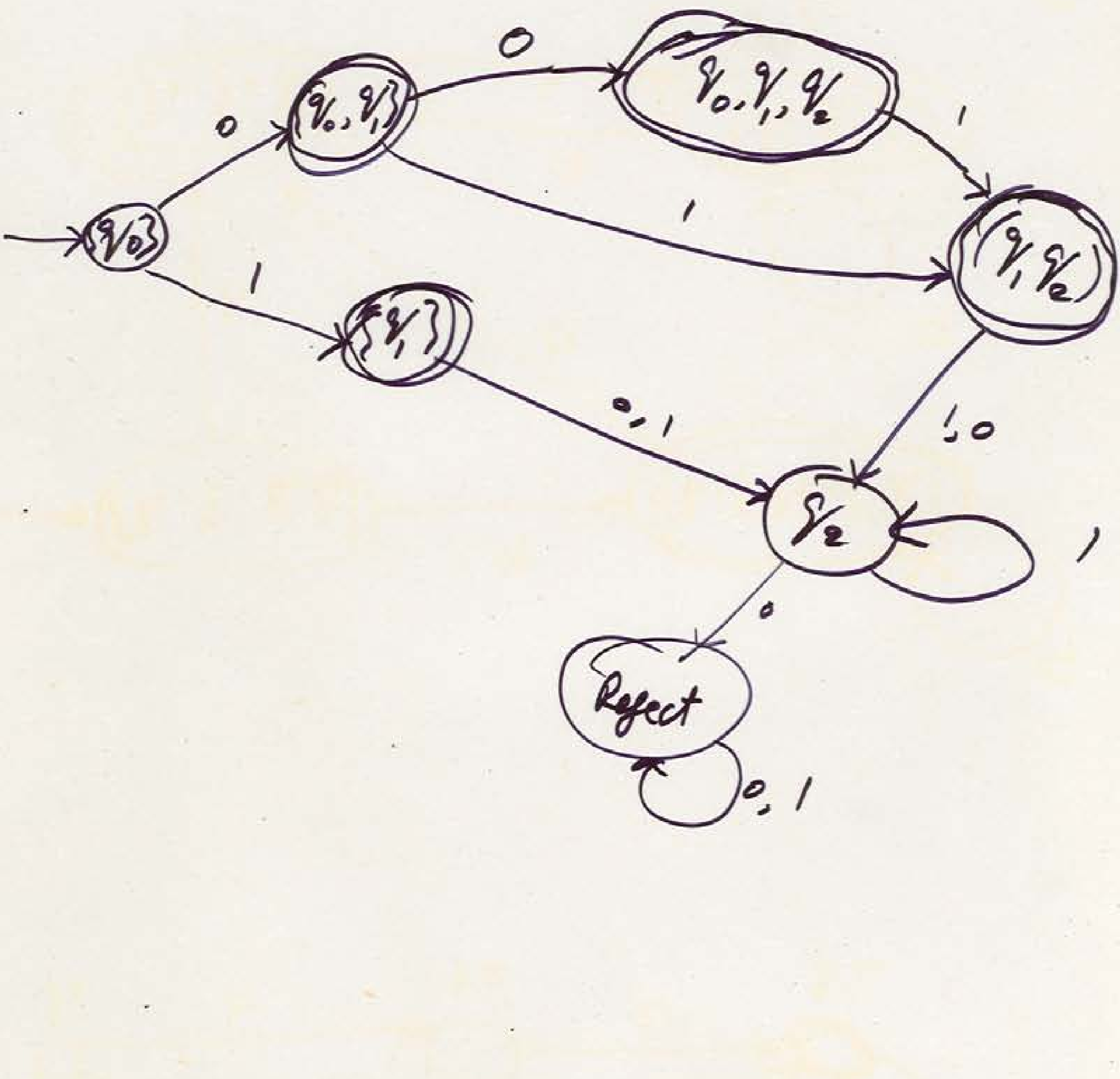
$U = \{q_2, q_5\}$

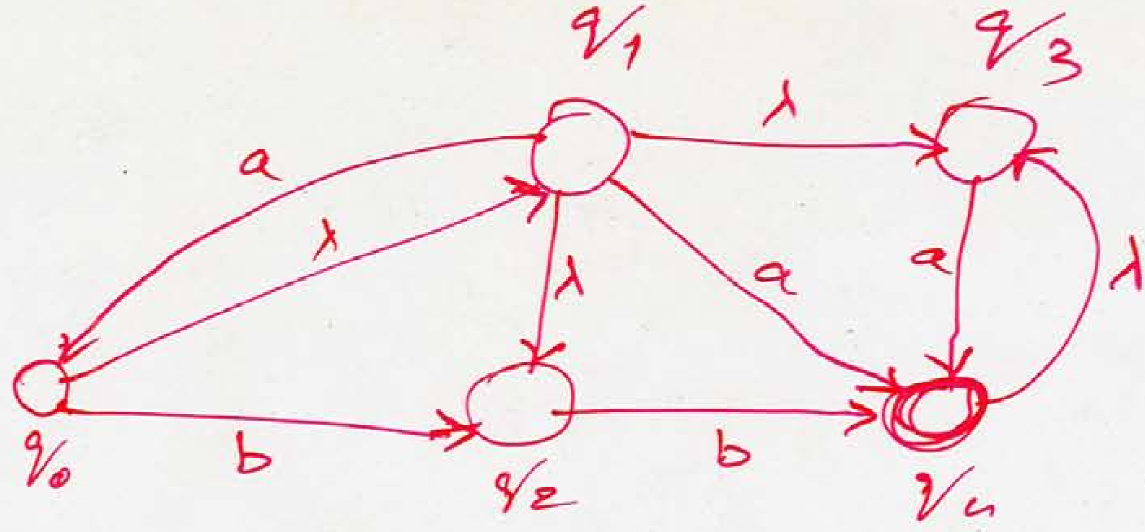
$T = \{q_3, q_6\}$



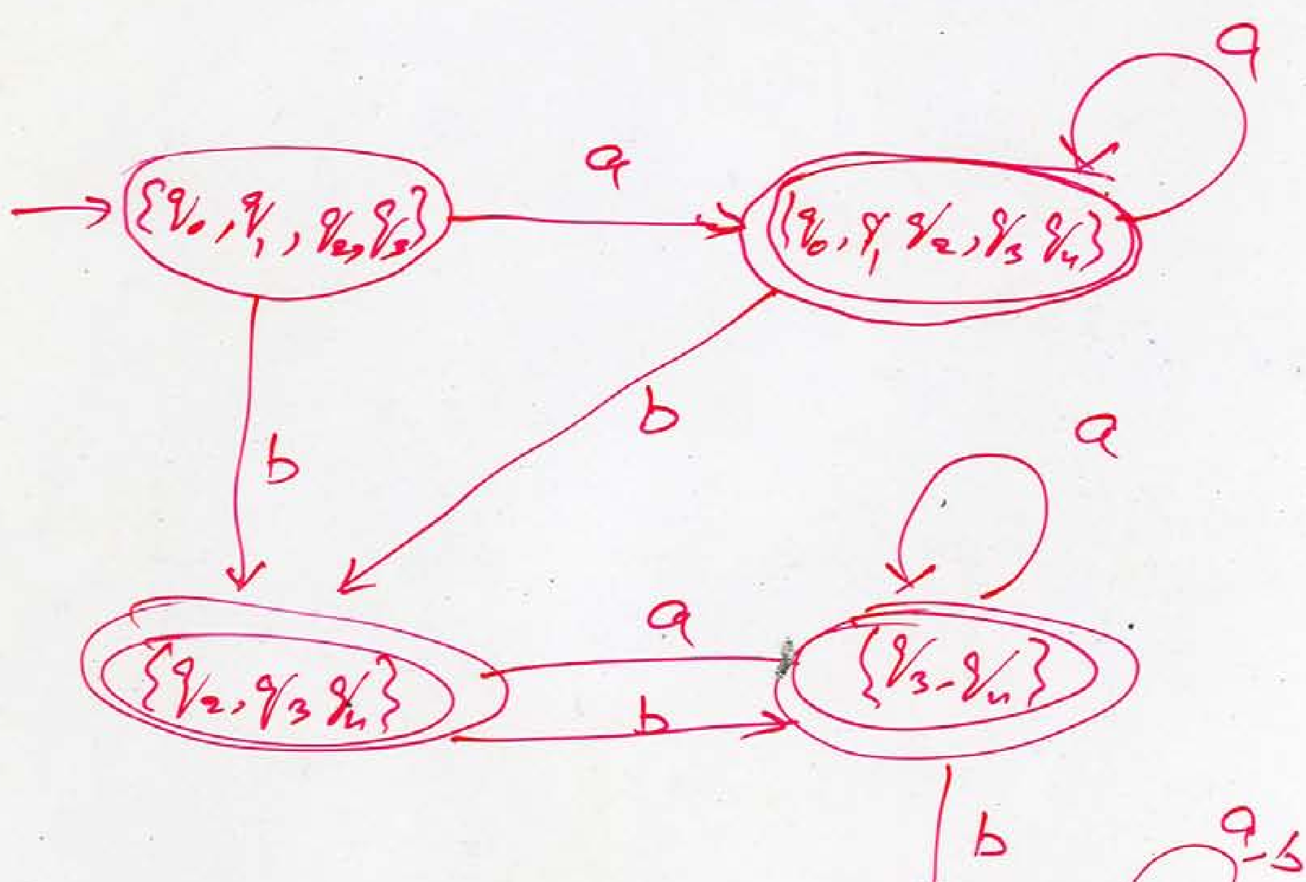
Convert NFA → DFA







q1
q3
λ
a
λ
λ
a
a
λ
q0
b
q2
b
q4



a
a
{q0, q1, q2, q3}
{q0, q1, q2, q3, q4}
b
b
a
a
{q2, q3, q4}
{q3, q4}
b
b
a, b
Sink
∅