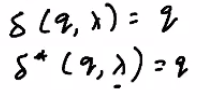
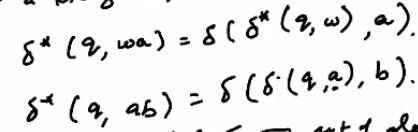
Finite State Automata - Contd

Delta(q0, lambda) = q0 (lambda = empty)

In general, for any state q:





**DFA (Acceptor automaton)**

Output -> Accepted or Not Accepted

Accepted when input string w is part of the language of the automaton, else not accepted. In other words, it gives the definition of the language of the machine, so can be used as a model of a grammar.

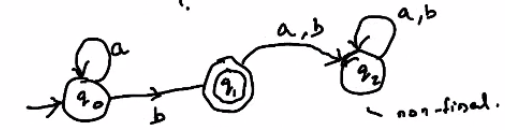
Eg

1)

L = {(a^n)b: n >= 0}

L = {b, ab, aab, aaab ....}

Finite state automaton for this:



The non-final state q2 is called a trap state

Test with w = aba

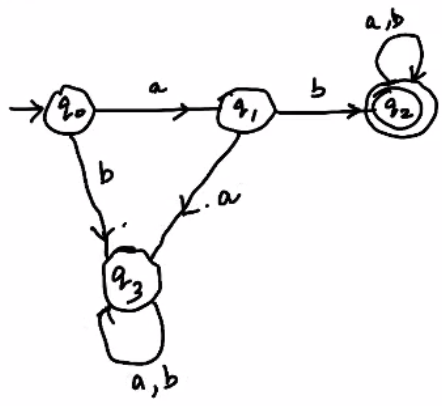
Machine does not reach a final state so not accepted

2)

Sigma = {a,b}

L = {strings with prefix ab}

So L = {ab, aba, abaa, abb ...}



**Regular Language**

A Language L is said to be regular iff. There exists a finite acceptor automaton M such that L = L(M), i.e. a language that can be defined using a finite acceptor automaton. To show that a language L is regular, draw a deterministic finite state automaton for that language.

Eg.

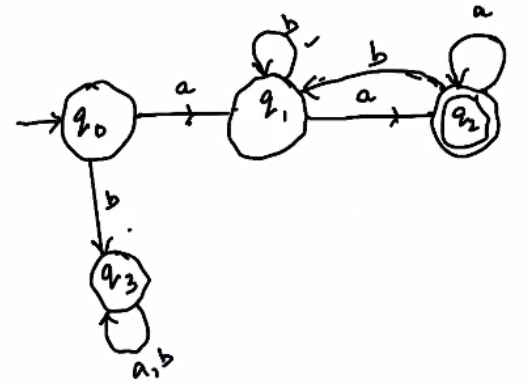
1)

L = {awa: w belongs to {a,b}\*} where w = {lambda, a, b, ab, aa, ...}



However, as can be seen in q1, there are 2 possible transitions when input is a. So this is not a deterministic automaton. This is an NFA.

We can construct a DFA for this language as follows



Test:

Q0 = aab

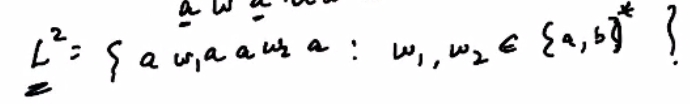
Then as can be seen, it doesn’t reach a final state, so it is not accepted.

This shows that L is regular.

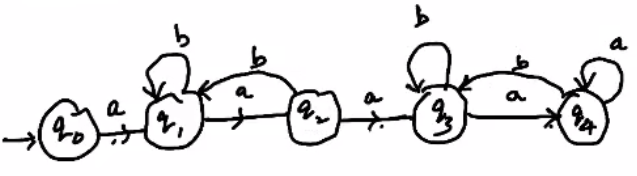
2)

Is L^2 regular?

L^2 = LL



DFA:

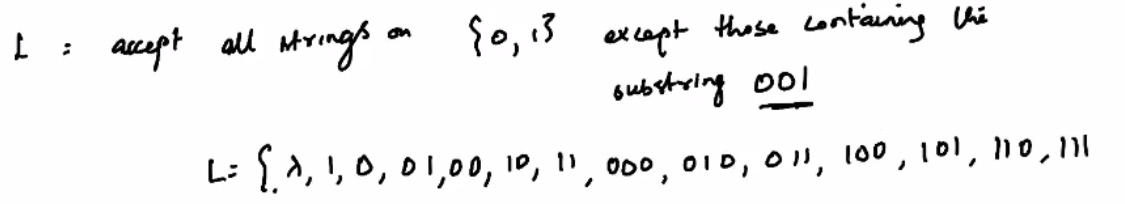


Test:

Q0 = abaaaaba

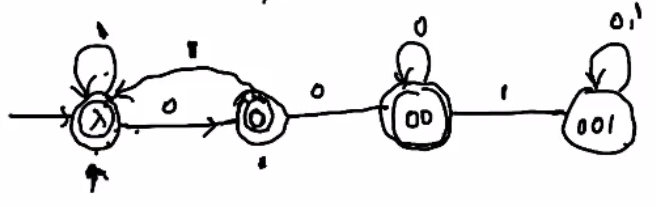
Gets accepted

3)



So we need 001 to be a non final state, i.e we can create a trap sequence for it.

For lambda, 0 and 00 we need to be in a final state.



Test:

Q0 = 1010100

Reaches final state

Q0 = 100100

Reaches non final state

**NOTE:**  
DFA only remembers only the current state, and has no memory of previous states. It performs state transitions based on the input string.