MODULE 2
INTRODUCTION TO AUTOMATA THEORY

An Automata is a construct that possess all the indispensable features of a computer.It accepts an input,produces output, may have some temporary storage and can make decisions in transforming the input into the output.

Finite Automata:

It is a mathematical model of acomputer.A Finite Automata consists of a finite set of states and a set of transitions from state to state that occur on input symbols chosen from an alphabet $\sum$

The pictorial representation of FA consists of

a)Input tape – It is divided into number of cells with one symbol in each cell. Input is a string on alphabet set $\sum$.

b)Finite Control –

c)Movable reading head

Initially the head is placed at the leftmost square of the tape and the finite control is set to an initial state.q_0,q_1,q_2….. are the statesin the finite control systems.x,y,z are the input symbols.At regular intervals, the automata reads one symbol from the input and then enters in a new state that depends only and then enters in a new state that depends on the current state and current symbol.After reading an input symbol, reading head move one square to the right on the input tape, so that on the next move,it will read the symbol in the next tape square.This process is repeated again and again.The automata then indicates its acceptability of a string by the state it is at the end ie if it winds up in one of a set of final states,the input string is considered to be accepted.The language accepted by the machine is the set of strings it accepts.

Notations for DFA

There are two preferred notations for describing automata.

1.Transition diagram

2.Transition table

Transition table:

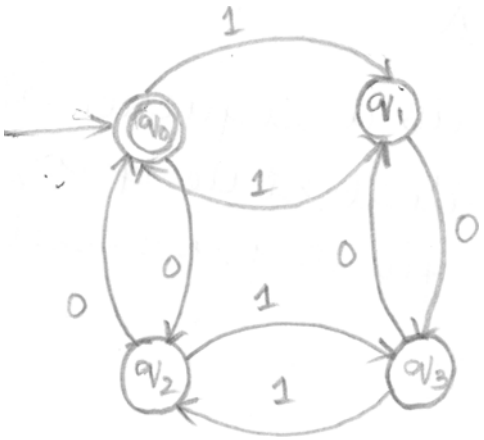
A transition table is a conventional table representation of a function like δ that takes 2 arguments and return states.The rows of the table corresponds to states and the columns corresponds to the input.

Transition Diagram:

A directed graph called a transition diagram is associated with a FA. Here vertices of a graph corresponds to the states of the FA and the edges represents transitions.The FA accepts a string ‘x’ if the sequence of transitions corresponding to the symbols of ‘x’ leads from start state to an accepting state.In transition graph, the initial state ‘q_0’ is indicated by the arrow labeled ‘start’.The states are represented as circles and the final states are indicated by the double circles.

For eg:

A transition diagram of a FA accepting all strings of 0’s and 1’s in which both the number of 0’s and 1’s are even.



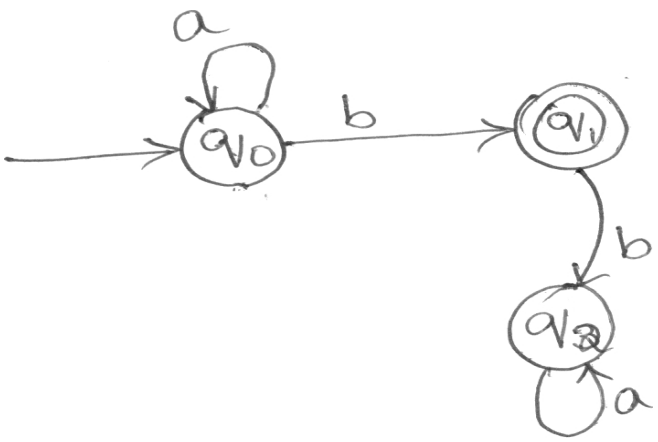
Deterministic Finite Automata(DFA)

A DFA is a FA which allows only one transition from a state on the same input symbol.A DFA is defined using 5 tuples M=(Q, Σ, δ, q_0, F) where

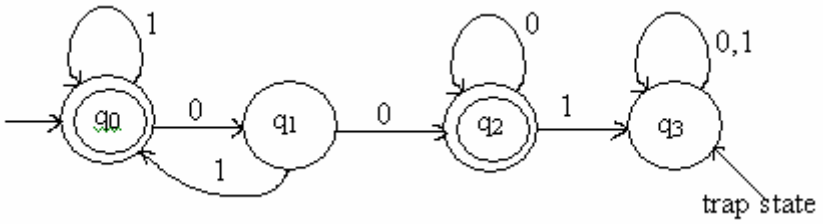
Q – set of states
Σ - i/p alphabet
δ - Q x Σ $\rightarrow$ Q
q_0 – initial state
F – set of final states

1.Construct a DFA accepts all strings consisting of arbitrary number of a's followed by a single 'b'.

L={a^n b ;n>=0}
HereM=(Q, Σ, δ, q_0, F); where Q={ q_0,q_1,q_2 } ;
Σ= {a,b }; F={ q_1 }



2. Construct a DFA accepts all strings defined on {0,1}, except those containing substring 001

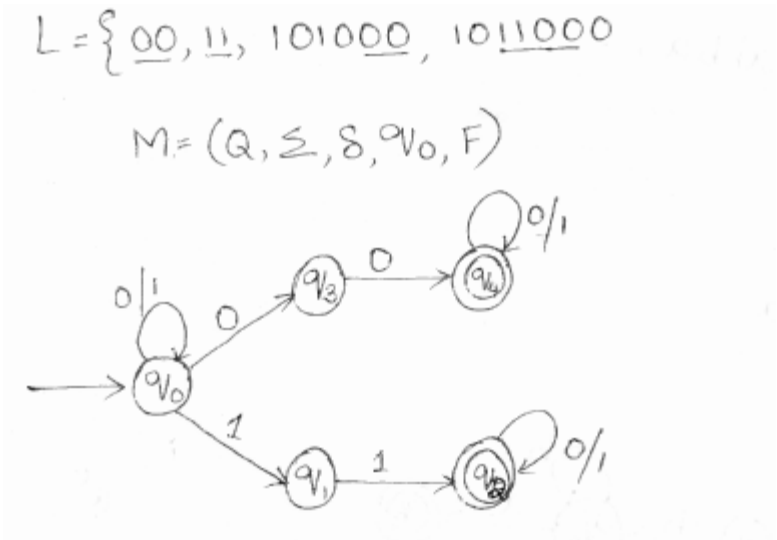


Nondeterministic Finite Automata:

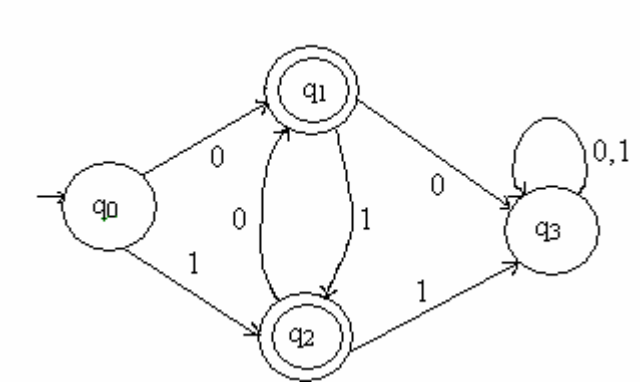
Nondeterminism means a choice of moves for an automon.NFA has more than one possible moves for a single state and the same input symbol. A NFA is defined by a 5 tuple M=(Q, Σ, δ, q_0, F) where

Q – set of states
Σ - i/p alphabet
δ - Q x (∑ x {∈}) → 2^Q
q_0 – initial state
F – set of final states

1) Design a NFA for the lang L = all strings over {0,1} that have atleast two consecutive 0's or 1's.



2. Design a NFA that accepts all strings defined on {0,1}, except those containing two consecutive zero's or one's



Differences between DFA & NFA

- For DFA, the outcome is a single state of Q whereas for NFA the outcome is a subset of Q
- NFA can make a transition without consuming an input symbol ie $\in$ transitions are possible.

A string is accepted by a NFA if there is some sequence of possible moves that will put the machine in a final state at the end of the string

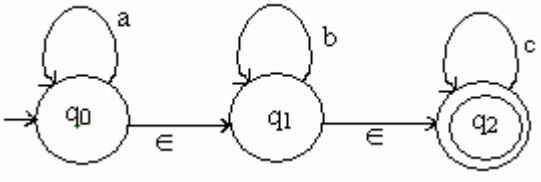
NFA with $\in$ transitions

$\in$- closure of a particular state is a set of all those states of automata which can be reached from that state on a path labeled by '$\in$'.

Rules:

- q0 is added to $\in$- closure of q0
- If q1 is in $\in$- closure of q0 and there is an edge labeled '$\in$' from q1 to q2, then q2 is also added to $\in$- closure of q0 if q2 is not already there.
- If 'T' is a set of states then $\in$- closure (T) is the union of $\in$- closure of single states.

For the figure given below, check whether input 'ab' is accepted or not



$\in$ - closure (q_0) = {q_0, q_1, q_2}

$\delta^*(q_0, \in)$ = $\in$ - closure (q_0) = {q_0, q_1, q_2}

$\delta^*(q_0, a)$ = $\in$ - closure ($\delta(\delta^*(q_0, \in)), a$)
= $\in$ - closure ($\delta(\{q_0, q_1, q_2\}), a$)
= $\in$ - closure ($\delta(q_0,a), \delta(q_1,a), \delta(q_2,a)$)
= $\in$ - closure ($\{q_0\}, \phi, \phi$)
= $\in$ - closure ($\{q_0\}$)
= {q_0, q_1, q_2}

$\delta^*(q_0, ab)$ = $\in$ - closure ($\delta(\delta^*(q_0, a)), b$)
= $\in$ - closure ($\delta(q_0,b), \delta(q_1,b), \delta(q_2,b)$)
= $\in$ - closure ($\phi, \{q_1\}, \phi$)
= $\in$ - closure (q_1)
= { q_1, q_2}

q_2 is the final state , hence ‘ab’ is accepted by the machine.

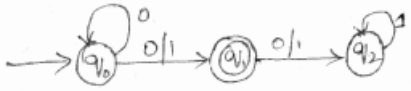
Theorm

If there is a NFA which accepts a language ‘L’ then there exists a DFA that accepts the same language ‘L’

Conversion of NFA to DFA

Step 1:Find all the states of NFA without $\in$-transitions including initial and final states
a)Initial state will be $\in$ - closure of the initial state of NFA with $\in$ - transitions
b)Final state of NFA without $\in$ - transitions are all those new states which contains the final state of NFA with $\in$ - transitions
c)Find the $\in$ - closure of rest of the states

2) Construct a DFA from following NFA.



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

$M = (\{q_0, q_1, q_2\}, \{0,1\}, \delta, q_0, \{q_1\})$

	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	ϕ	$\{q_2\}$

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

Transition table for DFA.

	0	1
$[q_0]$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	$[q_2]$	$[q_2]$
$[q_2]$	ϕ	$[q_2]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_2]$	$[q_2]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_1, q_2]$

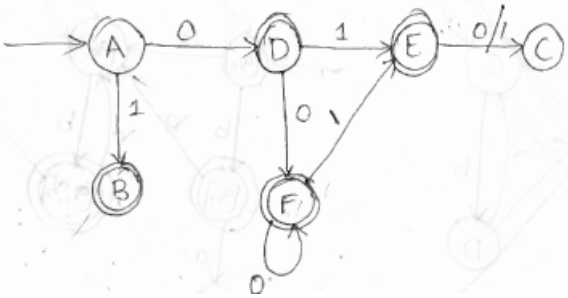
We can name each state of DFA as

$[q_0] = A$; $[q_1] = B$; $[q_2] = C$;

$[q_0, q_1] = D$; $[q_1, q_2] = E$; $[q_0, q_1, q_2] = F$

Here $q_0' = [q_0] = A$

$F' = \{B, D, E, F\}$



Minimization of DFA

Step 1:Remove all inaccessible or unreachable states.
Step 2:Draw the transition table for rest of the states
Step 3:Now divide rows of transition table into 2 sets as:
a)One set contains only those rows which start from non-final states.
b)Other set contains only those rows which starts from final states.
Step 4:Apply step 4 on both sides individually .Slip out the same states from the table,
Step 5:Repeat the step 4 for set2
Step 6: Combine set1 and set 2 .Now it is the transition table of minimized DFA

Finite Automata with output

Moore machine

A finite state machine that produces an output for each state. Output depends on present state, it is independent of current input. It can be represented as
$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$
where Δ - output alphabet
λ - mapping function from Q to Δ

Mealy Machine

A mealy machine produces an output for each transition (state/input pair). . It can be represented as
$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$
where Δ - output alphabet
λ - mapping function which maps Q and Σ to Δ

A moore machine can be transformed into an equivalent mealy machine by associating the output of each state with every transition that leads to that state. The languages accepted are the same (although the mealy machine doesn't recognize E).

To transform a mealy machine into a more machine, create a state for each state/output pair. In this example there are two possible outputs, 0 and 1. If state q on A goes to r, producing 1 in the mealy machine, then q_0 and q_1 on A go to r_1, and the state r_1 produces the output 1. Verify that this moore machine produces the same output as the original meally machine, for any given input string.

Regular Expressions

Regular expressions are algebraic description or notations used to describe language accepted by the FA.The operations on languages are used as the operations on r.e

1.Union
2.Concatenation
3.Star Closure
4.+ve closure

Regular expression involves a combination of strings of symbols from alphabet set Σ, parenthesis, the operators such as '+','*' and '.'.Union is denoted by using '+' operator and concatenation is denoted by using '.'operator

The set of regular expression is defined by the following rules

- Every letter of Σ can be made into a r.e
- '∈' is a regular expression
- If 'r1' and 'r2' are regular expressions then
  1.(r1) (2)r1.r2 (3)r1+r2 (4) $r1^*$ (5)$r1^+$ are also r.e
- A string is a r.e if and only if it can be derived from primitive r.e by a finite no: of applications of the rules in (2).

Pumping lemma for regular expressions

Pumping lemma is based on Pigeon hole principle. It states that

- If an infinite language is regular, it can be defined by a dfa.
- The dfa has some finite number of states (say, *n*).
- Since the language is infinite, some strings of the language must have length > *n*.
- For a string of length > *n* accepted by the dfa, the walk through the dfa must contain a cycle.
- Repeating the cycle an arbitrary number of times must yield another string accepted by the dfa.

The *pumping lemma* for regular languages is a way of proving that a given (infinite) language is not regular. It can be briefed as below.

- Assume the language L is regular.
- By the pigeonhole principle, any sufficiently long string in L must repeat some state in the dfa; thus, the walk contains a cycle.
- Show that repeating the cycle some number of times ("pumping" the cycle) yields a string that is not in L.
- Conclude that L is not regular.

If L is an infinite regular language, then there exists some positive integer m such that any string w ∈L whose length is m or greater can be decomposed into three parts, xyz, where

- |xy| is less than or equal to m,
- |y| > 0,
- $w_i = xy^iz$ is also in L for all i = 0, 1, 2, 3, ...

Applications of Finite Automata

1.Lexical Analyzer

The tokens of programming language can be expressed as regular sets.A no: of lexical analyzer generators takes as input a sequence of regular expression describing the tokens and produce a single finite automata recognizing any token.This lexical analyzer may be used as a module in a compiler.

2.Text Editors

Certain text editors and similar programs permit the substitution of a string for any string matching a given regular expression