UNIT-1

Introduction

Purpose of the Theory of Computation: Develop formal mathematical models of computation that reflect real-world computers. Nowadays, the Theory of Computation can be divided into the following three areas:

- Automata Theory
- Computability Theory
- Complexity Theory,

Automata theory

Automata Theory deals with definitions and properties of different types of "computation models". Examples of such models are:

- Finite Automata. These are used in text processing, compilers, and hardware design.
- Context-Free Grammars. These are used to define programming languages and in Artificial Intelligence.
- Turing Machines. These form a simple abstract model of a "real" computer, such as your PC at home.

Central Question in Automata Theory: Do these models have the same power, or can one model solve more problems than the other?

Computability theory

In the 1930's, G¨odel, Turing, and Church discovered that some of the fundamental mathematical problems cannot be solved by a "computer". (This may sound strange, because computers were invented only in the 1940's).

An example of such a problem is "Is an arbitrary mathematical statement true or false?" To attack such a problem, we need formal definitions of the notions of

- computer,
- algorithm, and
- computation.

The theoretical models that were proposed in order to understand solvable and unsolvable problems led to the development of real computers.

Central Question in Computability Theory: Classify problems as being solvable or unsolvable.

Complexity Theory

The main question asked in this area is "What makes some problems computationally hard and other problems easy?"

Informally, a problem is called "easy", if it is efficiently solvable. Examples of "easy" problems are (i) sorting a sequence of, say, 1,000,000 numbers, (ii) searching for a name in a telephone

directory, and (iii) computing the fastest way to drive from Ottawa to Miami. On the other hand, a problem is called "hard", if it cannot be solved efficiently, or if we don't know whether it can be solved efficiently. Examples of "hard" problems are (i) time table scheduling for all courses at Carleton, (ii) factoring a 300-digit integer into its prime factors, and (iii) computing a layout for chips in VLSI.

Some Terminologies

Alphabets :

An alphabet is a finite, nonempty set of symbols. The alphabet of a language is normally denoted by Σ .

Example :

$\Sigma = \{0, 1\}$

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

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

$\Sigma = \{\#, \nabla, \spadesuit, \beta\}$

Strings or Words over Alphabet :

A string or word over an alphabet Σ is a finite sequence of concatenated symbols of Σ. Denoted by Σ^*

Example : 0110, 11, 001 are three strings over the binary alphabet { 0, 1 } . aab, abcb, b, cc are four strings over the alphabet { a, b, c }.

It is not the case that a string over some alphabet should contain all the symbols from the alphabet. For example, the string cc over the alphabet { a, b, c } does not contain the symbols a and b. Hence, it is true that a string over an alphabet is also a string over any superset of that alphabet.

Length of a string :

The number of symbols in a string w is called its length, denoted by |w|.

Example : | 011 | = 4, |11| = 2, | b | = 1

It is convenient to introduce a notation e for the empty string, which contains no symbols at all. The length of the empty string e is zero, i.e., | e | = 0.

Empty string:

The string of zero length is empty string. It is denoted by e or λ.

Reverse string:

IF $w= w_1w_2\ldots w_n$ where each $w_i \in \Sigma$, the reverse of w is $w_nw_{n-1}\ldots\ldots w_1$

Substring:

Z is a substring of w if z appears consecutively within w .
e.g " deck" is a substring of abcdeckabcjkl.

Concatenation

Two strings over the same alphabet can be combined to form a third by operation of concatenation. The concatenation of strings x and y written xoy or simply xy is the string x followed by the string y.
Formally w=xy if and only if |w| =|x| + |y|, w(j)= x(j) for j=1 … , |x| and w(|x|+j)=y(j) for j=1 , |y|.
e.g 01o 001=01001
woe=eow =w for any string w . concatenation is associative i.e (wx)y= w(xy) for any string w, x,y.

Suffix:

If w=xy for some x, then y is a suffix of w.

Prefix:

If w=xy for some y, then x is a prefix of w.
Note: for each string w and each natural number i, the string w^i is defined as $w^0 = e$
$w^{i+1} = w^i w$
$w^1=w$
do^2 = dodo

Language

Language L over the alphabet (Σ) is the subset of Σ* . Any set of strings over an alphabet Σ i.e any subset of Σ* will be called a language. Thus Σ* , ø and Σ are languages.

Example

For Σ= { 0, 1}

L= { x : x ϵ { 0,1 } * | x has even number of Zero's } 011011100 ϵ L , 1111 ϵ L , 1011011000 ∉ L
The infinite language L are denoted as

L= { w ϵ Σ * : w has property P}

Operation on Languages

1. **Concatenation operation**

   If L_1 and L_2 are languages over Σ , their concatenation is L = L1.L2 or L1L2 where

   L= { w ϵ Σ * : w = xy for some x ϵ L1 and y ϵ L2 }

   e.g Σ= { 0,1}

L_1= { w ϵ Σ * : w has an even number of 0's}

L_2= { w ϵ Σ * : w starts with a 0 and the rest of the symbols are 1's }

then

L_1L_2= { w ϵ Σ * : w has an odd number of 0's }

2. **Kleene star**

Denoted by L* . It is the set of all strings obtained by concatenating zero or more strings from L.

L*= { w ϵ Σ * : w =w_1o w_2 o ……. o w_k for some K ≥ 0 and some w_1o w_2 o ……. o w_k ϵ L }

We write L^+ for the language LL*.

Equivalently L^+ is the language L^+ = { w ϵ Σ * : w =w_1o w_2 o ……. o w_k for some K ≥ 1 and some w_1o w_2 o ……. o w_k ϵ L }

3. **Union**

Finite Automata

Finite Automata

Automata are computational devices to solve language recognition problems. Language recognition problem is to determine whether a word belongs to a language.
Automaton = abstract computing device, or "machine"

An automaton is an abstract model of a digital computer. Finite automata are good models of computers with a extremely limited amount of memory. Example of finite automata is Elevator problem, control unit of computer etc.

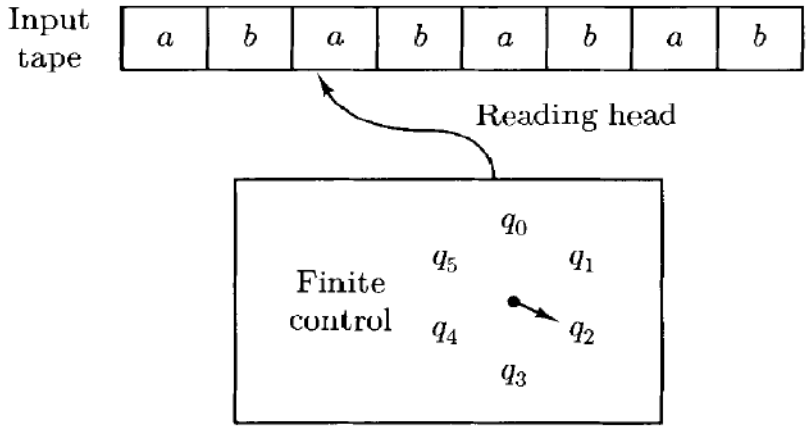


Fig: Finite automata

An automaton has a mechanism to read input ,which is string over a given alphabet. This input is actually written on an *input tape /file* **,**which can be read by automaton but cannot change it.
Input file is divided into cells each of which can hold one symbol. Automaton has a control unit which is said to be in one of finite number of internal states. The automation can change states in a defined way.

A state is the condition with respect to structure ,form, constitution and phase.

Finite automaton is the simplest acceptor or rejecter for language specification

Uses of finite automata

Used in software for verifying all kinds of systems with a finite number of states, such as communication protocols
Used in software for scanning text, to find certain patterns
Used in "Lexical analyzers" of compilers (to turn program text into "tokens", e.g. identifiers, keywords, brackets, punctuation)
Part of Turing machines and abacus machines

Types of Finite Automaton

- Deterministic Finite automata (DFA)
- Non -deterministic finite automata (NDFA/NFA)

Deterministic Finite Automata(DFA)
Deterministic automaton is one in which each move (transition from one state to another) is uniquely determined by the current configuration. If the internal states , input and contents of the storage are known it is possible to predict the next (future) behavior of the automaton . This is said to be deterministic automaton otherwise it is non - deterministic automaton.

Formal Definition
A deterministic finite automaton is a quintuple $M = (K, \sum, \delta, s, F)$ where
K is a finite set of states,
$\sum$ is an alphabet,
$s \in K$ is the initial state,
$F \subseteq K$ is the set of final states, and
δ, the transition function, is a function from $K \times \sum \rightarrow K$.

If M is in state $q \in K$ and the symbol read from the input tape is $a \in \sum$, then $\delta(q, a) \in K$ is the uniquely determined state to which K passes.

The transition from one internal state to another are governed by transition function δ.

If $\delta(q_0, a) \rightarrow q_1$ then if the DFA is in state q_0 and the current input symbol is 'a' the DFA will go into state q_1 .

Configuration of DFA

A configuration is determined by the current state and the unread part of the string being processed.
Let M = (K, ∑, δ, s, F) be a DFA ,we say that a word in K x ∑* is a configuration of M. It represents the current states of M and remaining unread input of M. e.g (q_2, ababab) is one configuration

a	b	a	b	a	b

current scanning symbol

The binary relation $\vdash_M$ holds between two configurations of M if and only if the machine can pass from one to the other as a result of a single move. Thus if (q, w) and (q', w') are two configurations of M, then (q, w) $\vdash_M$ (q', w') if and only if w = aw' for some symbol a ϵ $\sum$, and δ(q, a) = q'. For every configuration except those of the from (q, e) there is a uniquely determined next configuration.
A configuration of the form (q, e) signifies that M has consumed all its input, and hence its operation ceases at this point.

We denote the reflexive, transitive closure of $\vdash_M$ by $\vdash_M^*$; $(q, w) \vdash_M^* (q', w')$ is read, (q, w) yields (q', w') (after some number, possibly zero, of steps).

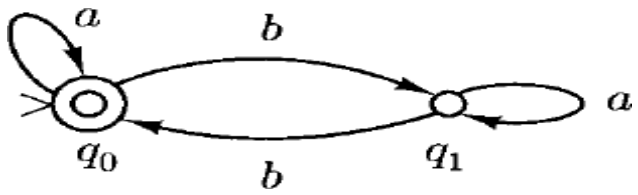
A string w ϵ $\sum$ is said to be accepted by M if and only if there is a state q ϵ F such that* $(s, w) \vdash_M^* (q, e)$. *Finally, the language accepted by M, L(M), is the set of all strings accepted by M.*

Example 1
*Design a DFA that accepts the language L which has set of strings in {a, b } * that have even number of b's .*

Solution
Let required DFA M= (K, $\sum$, δ, s, F)
where
K= {q_0, q_1 }
$\sum$= { a ,b}
s=q_0
F= { q_0 } and δ is the function tabulated below

δ/$\sum$	**a**	**b**
$\rightarrow q_0$	q_0	q_1
q_1	q_1	q_0

State transition diagram is given as



e.g : If M is given the input *aabba*, its initial configuration is (qo, aabba). Then

$$\begin{aligned}(q_0, aabba) &\vdash_M (q_0, abba)\\ &\vdash_M (q_0, bba)\\ &\vdash_M (q_1, ba)\\ &\vdash_M (q_0, a)\\ &\vdash_M (q_0, e)\end{aligned}$$

Therefore $(q_0, aabba) \vdash_M^* (q_0, e)$, and so $aabba$ is accepted by M.

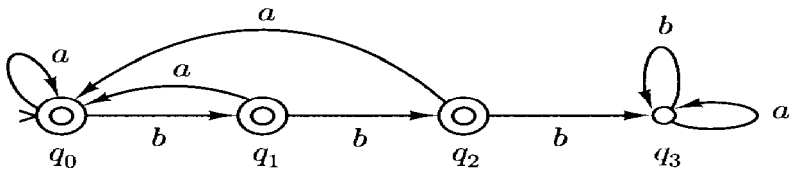
Example 2:

*Design a deterministic finite automaton M that accepts the language L(M) = {w ϵ {a, b} * : w does not contain three consecutive b's}.*

Solution
Let required DFA M= (K,∑, δ, s, F)
where
K= {q_0, q_1 , q_2, q_3 }
∑= { a ,b}
s=q_0
F= { q_0, q_1, q_2 } and δ is the function tabulated below

δ/∑	**a**	**b**
→q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_0	q_3
q_3	q_3	q_3

State diagram is given as

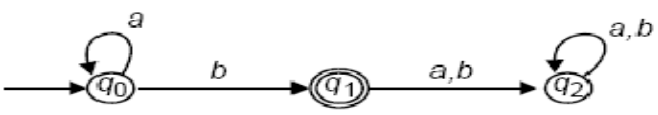


State q_3 is said to be a dead state, and if M reaches state q_3 it is said to be trapped, since no further input can enable it to escape from this state.

Example 3:
Design a DFA, the language recognized by the Automaton being L = {a^n b: n ≥ 0}
Solution
Let required DFA M= (K,∑, δ, s, F)
where
K= {q_0, q_1 , q_2 }
∑= { a ,b}
s=q_0
F= { q_1 } and δ is the function tabulated below

δ/∑	**a**	**b**
→q_0	q_0	q_1
*q_1	q_2	q_2
q_2	q_2	q_2

State diagram is given as

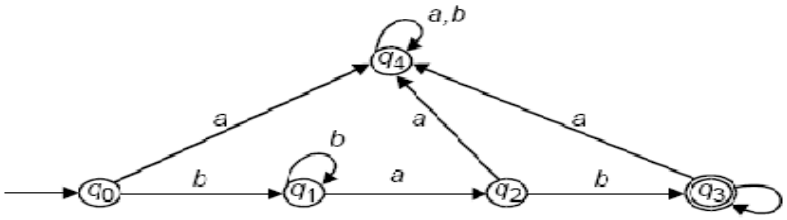


Example 4
Given ∑ = {a, b}, construct a DFA that shall recognize the language L = {$b^m ab^n$:m, n > 0}.

Solution
Let required DFA M= (K,∑, δ, s, F)
where
K= {q_0, q_1 , q_2, q_3 , q_4}
∑= { a ,b}
s=q_0
F= { q_3 } and δ is the function tabulated below

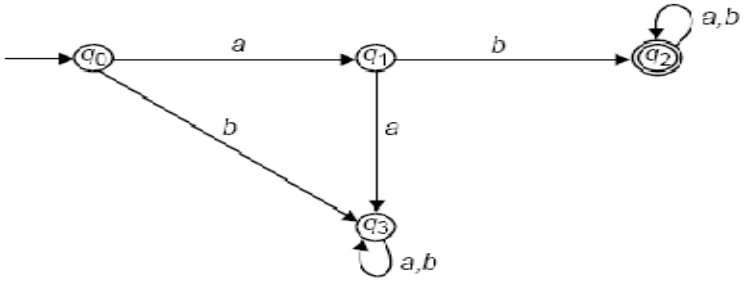
δ/∑	**a**	**b**
→ q_0	q_4	q_1
q_1	q_2	q_1
q_2	q_4	q_3
*q_3	q_4	q_3
q_4	q_4	q_4

State diagram is given as



Example 5:
Construct a DFA which recognizes the set of all strings on ∑ = {a, b} starting with the prefix 'ab'.

Solution



NONDETERMINISTIC FINITE AUTOMATA (NFA)

- NFA has Ability to change states in a way that is only partially determined by the current state and input symbol.
- Permit several possible "next states" for a given combination of current state and input symbol.
- The automaton, as it reads the input string, may choose at each step to go into anyone of these legal next states; the choice is not determined by anything in our model, and is therefore said to be nondeterministic.

- Nondeterministic devices are not meant as realistic models of computers. They are simply a useful notational generalization of finite automata, as they can greatly simplify the description of these automata.
- Nondeterministic finite automaton can be a much more convenient device to design than a deterministic finite automaton,

Formal Definition : A nondeterministic finite automaton is a quintuple M = (K, ∑, Δ, s, F), where
K is a finite set of states,
∑ is an alphabet,
s ϵ K is the initial state,
F K is the set of final states, and
Δ, the transition relation, is a subset of K x (∑ {e}) x K.

NFA is a variant of finite automaton with two capabilities

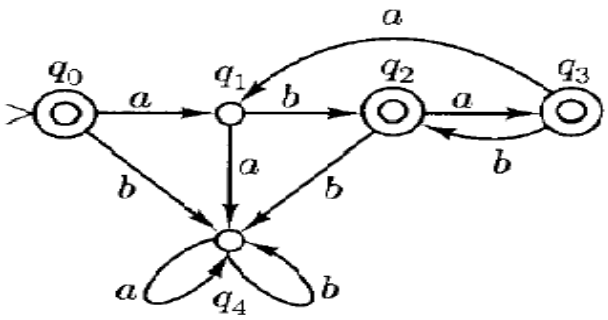
a. e or λ transition : state transition can made without reading a symbol
b. non-determinism : zero or more than one possible value may exist for state transition

For each p ϵ K and each a ϵ ∑{ e} , δ (s, a) =R means
"Upon reading an 'a ' the automaton M may transition from state s to any state in R."
For e in particular, δ (s, e) =R means,
"Without reading the symbol the automaton M may transition from s to any state in R"
Configuration of NFA:

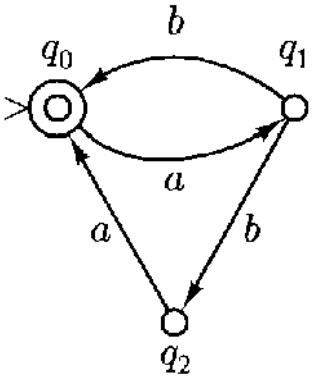
The formal definitions of computations by nondeterministic finite automata are very similar to those for deterministic finite automata. A **configuration** of M is, once again, an element of $K \times \Sigma^*$. The relation $\vdash_M$ between configurations (**yields in one step**) is defined as follows: $(q, w) \vdash_M (q', w')$ if and only if there is a $u \in \Sigma \cup \{e\}$ such that $w = uw'$ and $(q, u, q') \in \Delta$. Note that $\vdash_M$ need not be a function; for some configurations (q, w), there may be several pairs (q', w') —or none at all— such that $(q, w) \vdash_M (q', w')$. As before, $\vdash_M^*$ is the reflexive, transitive closure of $\vdash_M$ and a string $w \in \Sigma^*$ is **accepted** by M if and only if there is a state $q \in F$ such that $(s, w) \vdash_M^* (q, e)$. Finally $L(M)$, the language accepted by M, is the set of all strings accepted by M.

Examples

To see that a nondeterministic finite automaton can be a much more convenient device to design than a deterministic finite automaton, consider the language L = (ab U aba)*, which is accepted by the deterministic finite automaton illustrated in below



L is accepted by the simple nondeterministic device shown in fig below



Fog: NFA for L = (ab U aba)*

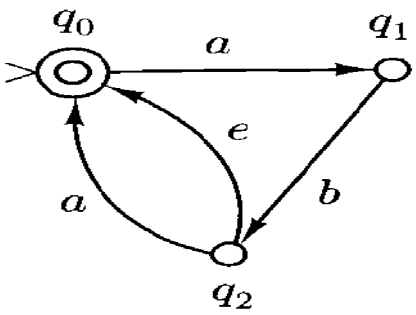


Fig: NFA for L = (ab U aba)* with e-transition

Example 2: Design a nondeterministic finite automata (NFA) that accept the set of all strings containing an occurrence of the pattern bb or of the pattern bab.

Solution:

Let required NFA M= (K, ∑, Δ, s, F), where

K= { q_O , q_l, q_2, q_3, q_4 }

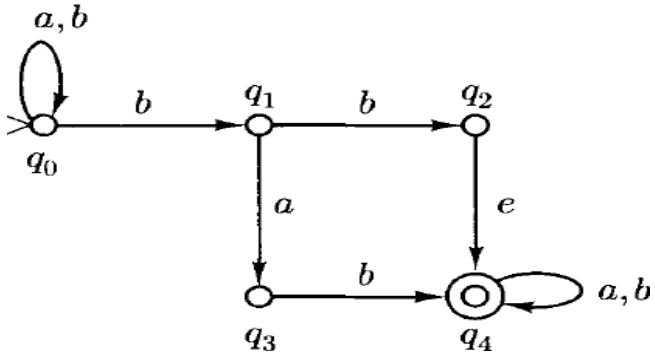
∑= {a, b }

s= q_O

F= { q_4 } and

$$\Delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, b, q_1), (q_1, b, q_2), (q_1, a, q_3), (q_2, e, q_4), (q_3, b, q_4), (q_4, a, q_4), (q_4, b, q_4)\}.$$

and state diagram is given by



When M is given the string *bababab* as input, several different sequences of moves may ensue. For example, M may wind up in the non final state q_o in case the only transitions used are (q_O, a, q_0) and (q_O, b, q_0):

$$
\begin{aligned}
(q_0, bababab) &\vdash_M (q_0, ababab) \\
&\vdash_M (q_0, babab) \\
&\vdash_M (q_0, abab) \\
&\vdots \\
&\vdash_M (q_0, e)
\end{aligned}
$$

The same input string may drive M from state q_0 to the final state q_4, and indeed may do so in three different ways. One of these ways is the following.

$$
\begin{aligned}
(q_0, bababab) &\vdash_M (q_1, ababab) \\
&\vdash_M (q_3, babab) \\
&\vdash_M (q_4, abab) \\
&\vdash_M (q_4, bab) \\
&\vdash_M (q_4, ab) \\
&\vdash_M (q_4, b) \\
&\vdash_M (q_4, e)
\end{aligned}
$$

Since a string is accepted by a nondeterministic finite automaton if and only if there is at least one sequence of moves leading to a final state, it follows that *bababab* ϵ L(M)

E-closure

Epsilon closure (ε- closure) of a state q is the set that contains the state q itself and all other states that can be reached from state q by following ε transitions.
We use the term, ECLOSE(q0) to denote the Epsilon closure of state q0.

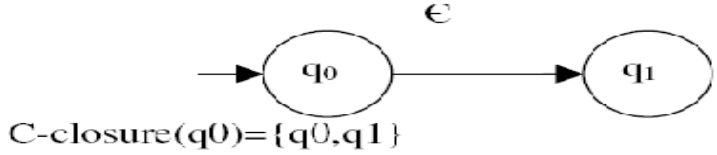
Suppose that the given NFA M= (Q,∑, δ , q_0,F) . For any set s ⊆ Q of states of M, we define the e-closure of S, denoted by E(S) to be the set of states reachable from s via zero or more e transition in a row.
Formally for I ≥ 0 ,we define $E_i(S)$ inductively to be the set of states reachable from s via exactly i many e-transitions, so that $E_0(s)$=s and for I ≥ 0

$E_{i+1}(s) = \{ r \in Q \mid E_q \in E_i(s) : r \in \delta(q, e) \}$

$= \bigcup_{q \in Ei(s)} \delta(q, e)$

$E(s) = \bigcup_{i \geq 0}$ Ei(s)

Namely, the set of all states reachable from s via zero or more e-transitions.
e-c losure(q0) denotes a set of all vertices p such that there is a path from q0 to p labeled . Example



Equivalence of DFA and NFA

A DFA is just a special type of NFA . In a DFA it so happens that the transition relation Δ is in fact a function from K x $\sum$ to K i.e. NFA (K,$\sum$, δ, s, F) is deterministic if and only if there are no transition of the form (q, e, p) in δ and for each q $\in$ K and a $\in \sum$ there is a exactly one p $\in$ K such that (q, a, p) $\in \Delta$

It is there fore evident that the class of languages accepted by DFA is a subset of the class of languages accepted by NFA. Rather surprisingly these classes are in fact equal.

Despite the power and generality enjoyed by NFA, they are no more powerful than the NFA in terms of the language they accept.

NFA can always be converted into an equivalent deterministic one.

Formally, we say that the two finite automata M_1 and M_2 (NFA and DFA) are equivalent if and only if $L(M_1) = L(M_2)$. Thus two automata are considered to be equivalent if they accept the same language, even though they may use different methods to do so.

Theorem : For each NFA , there is equivalent DFA.

Proof:

Let M=(K, $\sum$,Δ, s, F) be a NFA we shall construct a DFA **M'=**(K' , $\sum$, δ', s', F') equivalent to M. View a NFA as a occupying at any moment not a single state but a set of states, namely all the states that can be reached from the initial state by means of the input consumed thus far.

- The set of states of M' will be $K' = 2^K$, the power set of states of M
- The set of final state of M' will consist of all those subset of K that contains at least one final state of M. i.e. F'={ Q$\subseteq$ K : Q $\cap$ F $\neq$ Ø }
- The definition of transition function of M' is slightly complicated . The basic idea is that a move of M' on reading an input symbol a , possibly followed by any number of e-moves of M.

  For every Q $\subseteq$ K and a $\in \sum$

  $\delta'(Q, a) = E\left(\bigcup_{q \in Q} \delta(q, a) \right)$

  To formalize this idea we need a special definition

  For any state q ϵ K , let E(q) be the set of all states of M that are reachable from state q without reading any input (e-moves) i.e.

  $E(q) = \{ p \epsilon K : (q , e) \vdash_M^* (p , e) \}$

  To put it otherwise, E(q) is the closure of the set {q} under the relation
  { (p, r) : there is a transition (p, e, r) ϵ Δ}.

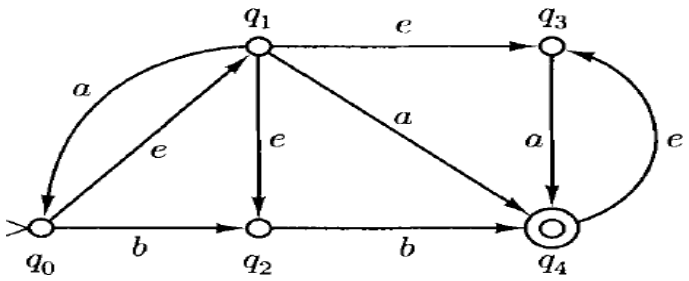
  Thus, E(q) can be computed by the following algorithm:

  Initially set E(q) := {q};

while there is a transition (p, e, r) ∈ Δ. *with p* ∈ *E(q)*

and r ∉ E(q) do: E(q) := E(q) U {r}.

Example: Convert the e-NFA given below to its corresponding DFA



Solution:

Here

Given NFA M= (K,∑,δ, s, F)

$E(q_0)$= { q_0, q_1, q_2, q_3}

$E(q_1)$ = {q_1,q_2,q_3}, and

$E(q_2)$ = {q_2}

We are now ready to define formally the deterministic automaton M'= (K',∑,δ', s', F') that is equivalent to M, where

K' =2^K

S'=E(s)

F'={ Q⊆ K : Q ∩ F ≠ Ø } and

for each Q⊆ K and each symbol a ∈ ∑ ,define

δ'(Q, a) = U{E(p) : p ∈ K and (q, a, p) ∈ Δfor some q ∈ Q }

i. e. δ'(Q, a) is taken to be the set of all states of M to which M can go by reading input a and possibly followed several e-moves.

E(q0) = { q0, q1,q2,q3 }

δ'(s', a)= E(q0) U E(q4) = { q0, q1,q2,q3 , q4}

Similarly,

δ'(s', b)= E(q2) U E(q4)= { q2,q3 , q4}

Again for newly introduced states

δ'({ q0, q1,q2,q3 , q4} , a)= { q0, q1,q2,q3 , q4}

δ'({ q0, q1,q2,q3 , q4} , b)= {q2,q3 , q4}

Again for newly introduced states {q2,q3 , q4}

$\delta'(\{q2, q3 , q4\} , a) = \{q3 , q4\}$

δ'({q2,q3 , q4} , b)= {q3 , q4}

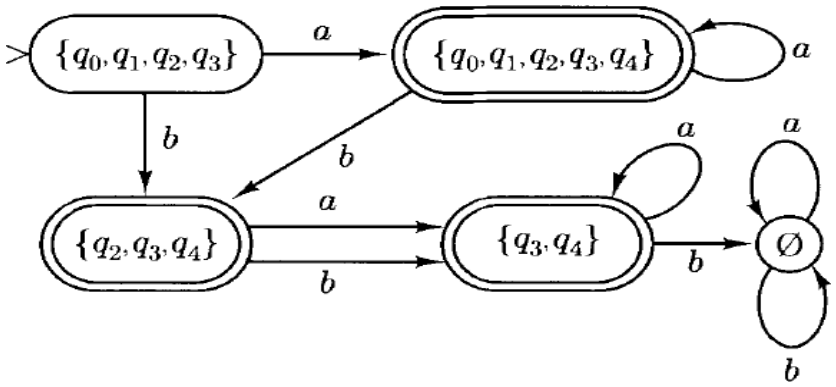
Again,

δ'({q3 , q4} , a)= {q3 , q4}

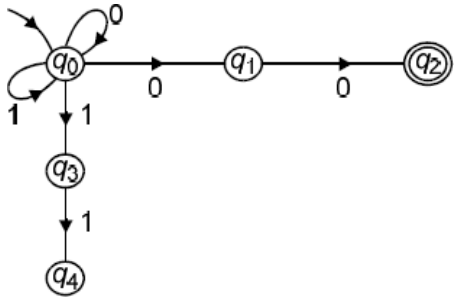
δ'({q3 , q4} , b)= { Ø }

and finally
δ'({Ø , a)= { Ø }
δ'({Ø , b)= { Ø }

F' , the set of final states contains each set of states of which q4 is member , since q4 is the sole final member of F.



Example : Given the NFA as shown in fig. below, determine the equivalent DFA.



Solution

The given NFA has q_2 and q_4 as final states. It accepts strings ending in 00 or 11. The state table is shown below.

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	$\{q_2\}$	$\emptyset$
q_2	$\emptyset$	$\emptyset$
q_3	$\emptyset$	$\{q_4\}$
q_4	$\emptyset$	$\emptyset$

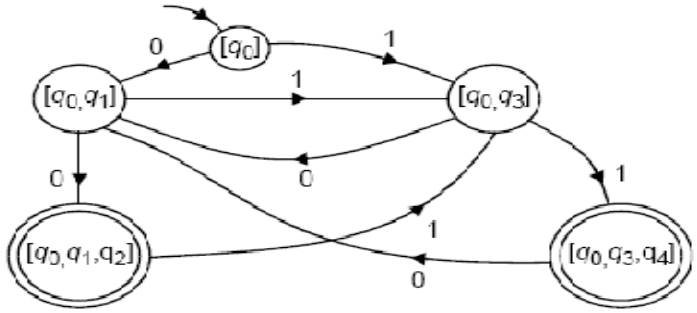
The conversion of NFA to DFA is done through the subset construction.

δ' is given by the following state table.

	0	1
$\rightarrow [q_0]$	$[q_0, q_1]$	$[q_0, q_3]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_3]$
$[q_0, q_3]$	$[q_0, q_1]$	$[q_0, q_3, q_4]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_3]$
$[q_0, q_3, q_4]$	$[q_0, q_1]$	$[q_0, q_3, q_4]$

Any state containing q_2 or q_4 will be a final state.

The DFA is shown below.



1.