

CHAPTER 1

INTRODUCTION

The aim of the project is to build a code that implements a DFA. The idea is to implement it using client and server program. The program takes string as an input , and it should print the states the string goes through and prints whether the string is accepted by the DFA or not accepted by the DFA.

The client-server is implemented using socket programming. The input is accepted at the client end , this input is further sent to the server. The server accepts the input from the socket streams, and it runs the DFA module, here the string is checked for the validity according to the given DFA.

It shows all the states passed by the string and displays whether the string is accepted or not accepted by the given DFA.

CHAPTER 2

DETERMINISTIC FINITE AUTOMATION (DFA)

2.1 INTRODUCTION

In the theory of computation, a branch of theoretical computer science, a deterministic finite automation(DFA) also known as deterministic finite acceptor(DFA) and deterministic finite state machine is a finite machine that accepts/rejects finite strings of symbols and only produces a unique computation (or run) of the automation for each input string, ‘Deterministic’ refers to the uniqueness of the computation. In search of the simplest models to capture the finite state machines, McCulloch and Pitts were among the first researcher to introduce a concept of Finite Automation in 1943. A deterministic finite automaton (DFA) consists of:

1. a finite set of states (often denoted Q)
2. a finite set Σ of *symbols* (alphabet)
3. a *transition function* that takes as argument a state and a symbol and returns a state (often denoted δ)
4. a start state often denoted q_0
5. a set of final or accepting states (often denoted F)

We have $q_0 \in Q$ and $F \subseteq Q$

So a DFA is mathematically represented as a 5 tuple

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

The transition function δ is a function in

$Q \times \Sigma \rightarrow Q$

$Q \times \Sigma$ is the set of 2-tuples (q, α) with $q \in Q$ and $\alpha \in \Sigma$

2.2 DFA WITH TRANSITION TABLE

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Figure 1 Transition table

The \rightarrow indicates the start state: here q_0

The $*$ indicates the final state(s) (here only one final state q_1)

This defines the following transition diagram:

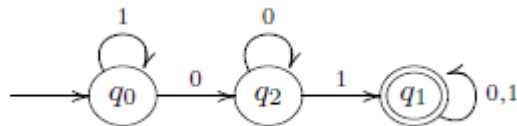


Figure 2 Transition diagram of DFA

For this example

$$Q = \{q_0, q_1, q_2\}$$

start state q_0

$$F = \{q_1\}$$

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

δ is a function from $Q \times \Sigma$ to Q

$$\delta: Q \times \Sigma \rightarrow Q$$

$$\delta(q_0, 1) = q_0$$

$$\delta(q_0, 0) = q_2$$

Example: password

When does the automaton accept a word??

It reads the word and accepts it if it stops in an accepting state

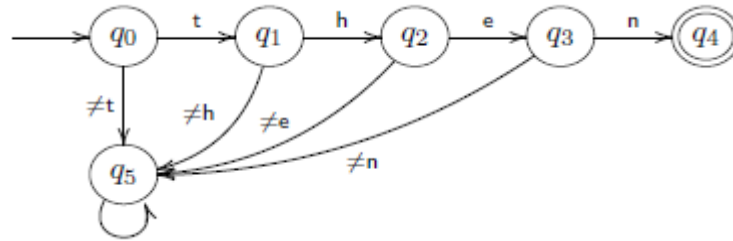


Figure 3 Transition diagram of DFA to read a string

Only the word then is accepted

Here $Q = \{q0, q1, q2, q3, q4\}$

Σ is the set of all characters

$F = \{q4\}$

We have a “stop” or “dead” state $q5$, not accepting

2.3 HOW A DFA PROCESSES A STRING

Let us build an automaton that accepts the words that contain 01 as a subword

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

$L = \{x01y \mid x, y \in \Sigma^*\}$

We use the following states

A: start

B: the most recent input was 1 (but not 01 yet)

C: the most recent input was 0 (so if we get a 1 next we should go to the accepting state D)

D: we have encountered 01 (accepting state)

We get the following automaton

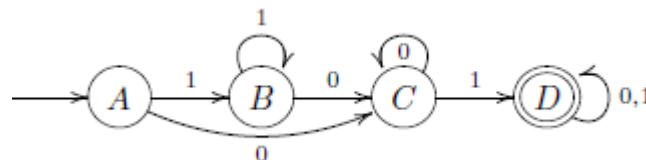


Fig 3: Transition diagram to process a string

Transition table

	0	1
→A	C	B
B	C	B
C	C	D
*D	D	D

Figure 4 Transition table 2

$Q = \{A, B, C, D\}$, $\Sigma = \{0,1\}$, start state A, final state(s) $\{D\}$

2.4 APPLICATION OF DFA

Deterministic Finite Automata, or DFAs, have a rich background in terms of the mathematical theory underlying their development and use. This theoretical foundation is the main emphasis of ECS 120's coverage of DFAs. However, this handout will focus on examining real-world applications of DFAs to gain an appreciation of the usefulness of this theoretical concept. DFA uses include protocol analysis, text parsing, video game character behavior, security analysis, CPU control units, natural language processing, and speech recognition. Additionally, many simple (and not so simple) mechanical devices are frequently designed and implemented using DFAs,

such as elevators, vending machines, and traffic-sensitive traffic lights.

2.4.1 Vending Machines

Figure 1 presents a DFA that describes the behavior of a vending machine which accepts dollars and quarters, and charges \$1.25 per soda. Once the machine receives at least \$1.25, corresponding to the blue-colored states in the diagram, it will allow the user to select a soda. Self-loops represent ignored input: the machine will not dispense a soda until at least \$1.25 has been deposited, and it will not accept more money once it has already received greater than or equal to \$1.25.

To express the DFA as a 5-tuple, the components are defined as follows:

1. $Q = \{ \$0:00; \$0:25; \$0:50; \$0:75; \$1:00; \$1:25; \$1:50; \$1:75; \$2:00 \}$ are the states
2. $\Sigma = \{ \$0:25; \$1:00; \text{select} \}$ is the alphabet
3. δ , the transition function, is described by the state diagram.
4. $q_0 = \$0:00$ is the start state
5. $F = \emptyset$; is the set of accept states

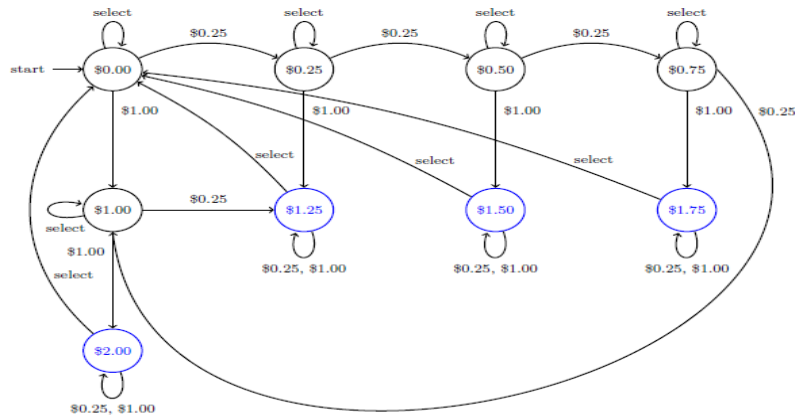


Figure 5 Vending machine state diagram

2.4.2 AI in Video Games: Pac-Man's Ghosts

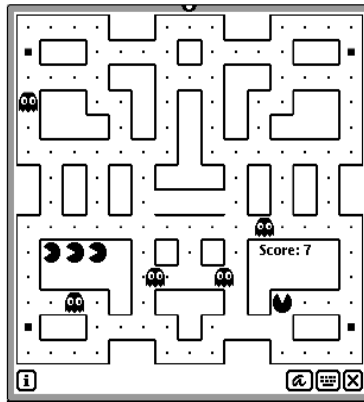


Figure 6 Screenshot of Pacman's game

Finite state machines lend themselves to representing the behavior of computer controller characters in video games. The states of the machine correspond to the character's behaviors, which change according to various events. These changes are modeled by transitions in the state diagram. State machines are certainly not the most sophisticated means of implementing artificially intelligent agents in games, but many games include characters with simple, state-based behaviors that are easily and effectively modeled using state machines.

Here we consider the classic game, Pac-Man. For those unfamiliar with the game- play, Pac-Man requires the player to navigate through a maze, eating pellets and avoiding the ghosts who chase him through the maze. Occasionally, Pac-Man can turn the tables on his pursuers by eating a power pellet, which temporarily grants him the power to eat the ghosts. When this occurs, the ghosts' behavior changes, and instead of chasing Pac-Man they try to avoid him.

The ghosts in Pac-Man have four behaviors:

1. Randomly wander the maze
2. Chase Pac-Man, when he is within line of sight
3. Flee Pac-Man, after Pac-Man has consumed a power pellet
4. Return to the central base to regenerate

These four behaviors correspond directly to a four-state DFA. Transitions are dictated by the situation in the game. For instance, a ghost DFA in state 2 (Chase Pac-Man) will transition to state 3 (Flee) when Pac-Man consumes a power pellet.

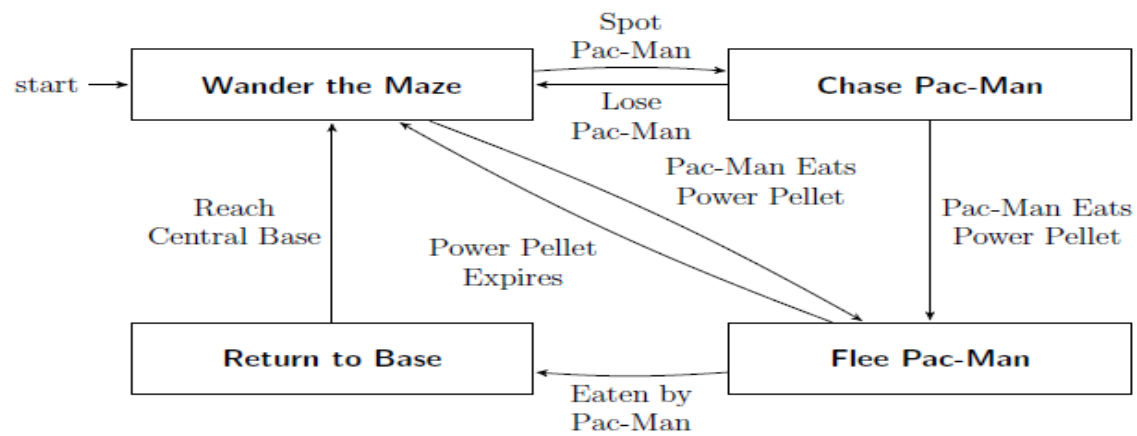


Figure 7 Behavior of a Pac Man ghost

CHAPTER 3

SOCKET PROGRAMMING

3.1 INTRODUCTION

Internet and WWW have emerged as global ubiquitous media for communication and changed the way we conduct science, engineering, and commerce. They are also changing the way we learn, live, enjoy, communicate, interact, engage, etc. The modern life activities are getting completely centered around or driven by the Internet. To take advantage of opportunities presented by the Internet, businesses are continuously seeking new and innovative ways and means for offering their services via the Internet. This created a huge demand for software designers and engineers with skills in creating new Internet-enabled applications or porting existing/legacy applications to the Internet platform. The key elements for developing Internet-enabled applications are a good understanding of the issues involved in implementing distributed applications and sound knowledge of the fundamental network programming models.

4.2 CLIENT/SERVER COMMUNICATION

At a basic level, network-based systems consist of a server, client, and a media for communication

as shown in Fig. 13.1. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network.

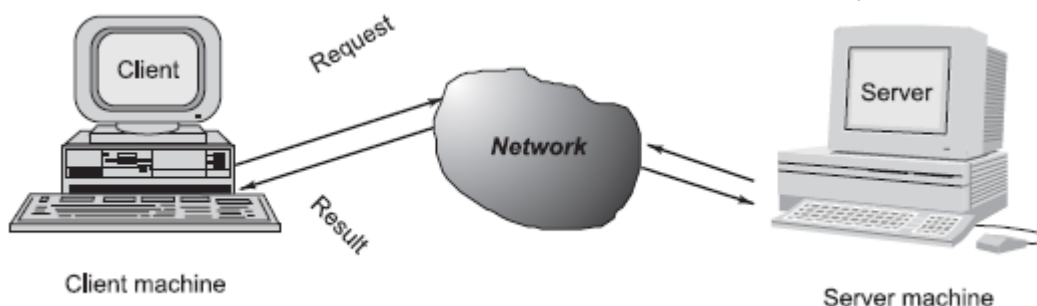


Figure 8 Client-Server communication

Generally, programs running on client machines make requests to a program (often called as server program) running on a server machine. They involve networking services provided by the transport layer, which is part of the Internet software stack, often called *TCP/IP (Transport Control Protocol/Internet Protocol) stack*, shown in Fig. 13.2. The transport layer comprises two types of protocols, *TCP (Transport Control Protocol)* and *UDP (User Datagram Protocol)*. The most widely used programming interfaces for these protocols are sockets. TCP is a connection-oriented protocol that provides a reliable flow of data between two computers. Example applications that

use such services are HTTP, FTP, and Telnet. UDP is a protocol that sends independent packets of data, called *datagrams*, from one computer to another with no guarantees about arrival and sequencing. Example applications that use such services include Clock server and Ping. The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer. Port is represented by a positive (16-bit) integer value. Some ports have been reserved to support common/well known services:

- ftp 21/tcp
- telnet 23/tcp
- smtp 25/tcp
- login 513/tcp
- http 80/tcp,udp
- https 443/tcp,udp

User-level process/services generally use port number value ≥ 1024 .

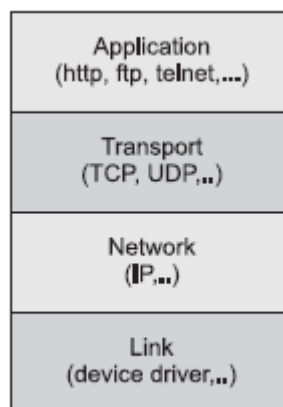


Figure 9 TCP/IP software stack

3.3 OBJECT ORIENTED PROGRAMMING WITH JAVA

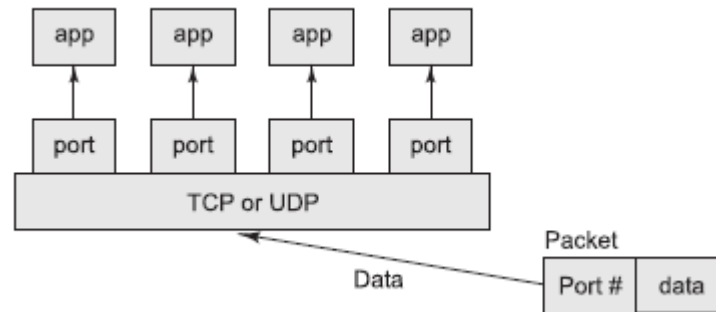


Figure 10 TCP/UDP mapping of incoming packets to appropriate port/process

Object-oriented Java technologies—Sockets, threads, RMI, clustering, Web services—have emerged as leading solutions for creating portable, efficient, and maintainable large and complex Internet applications.

3.4 HOSTS IDENTIFICATION AND SERVICE PORTS

Every computer on the Internet is identified by a unique, 4-byte *IP address*. This is typically written in dotted quad format like 128.250.25.158 where each byte is an unsigned value between 0 and 255. This representation is clearly not user-friendly because it does not tell us anything about the content and then it is difficult to remember. Hence, IP addresses are mapped to names like *www.buyya.com* or www.google.com, which are easier to remember. Internet supports name servers that translate these names to IP addresses. In general, each computer only has one Internet address. However, computers often need to communicate and provide more than one type of service or to talk to multiple hosts/computers at a time. For example, there may be multiple ftp sessions, web connections, and chat programs all running at the same time. To distinguish these services, a concept of *ports*, a logical access point, represented by a 16-bit integer number is used. That means, each service offered by a computer is uniquely identified by a port number. Each Internet packet contains both the destination host address and the port number on that host to which the message/request has to be delivered. The host computer

dispatches the packets it receives to programs by looking at the port numbers specified within the packets. That is, IP address can be thought of as a house address when a letter is sent via post/snail mail and port number as the name of a specific individual to whom the letter has to be delivered.

3.5 SOCKETS AND SOCKET BASED COMMUNICATION

Sockets provide an interface for programming networks at the transport layer. Network communication using Sockets is very much similar to performing file I/O. In fact, socket handle is treated like file handle. The streams used in file I/O operation are also applicable to socket-based I/O. Socket-based communication is independent of a programming language used for implementing it. That means, a socket program written in Java language can communicate to a program written in non-Java (say C or C++) socket program. A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server listens to the socket for a client to make a connection request (see Fig. 13.4a). If everything goes well, the server accepts the connection (see Fig. 13.4b). Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.

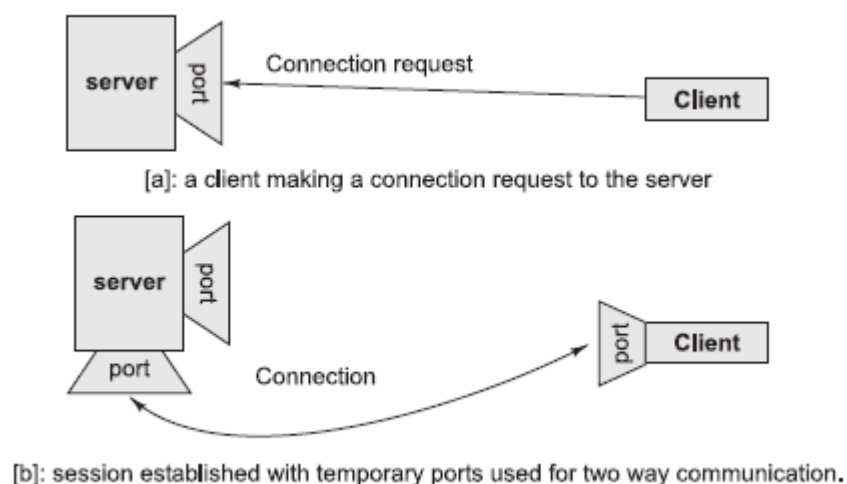


Figure 11 Establishment of path for two-way communication between a client and server

3.6 SOCKET PROGRAMMING AND JAVA.NET CLASS

A socket is an endpoint of a two-way communication link between two programs running on the network. Socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Java provides a set of classes, defined in a package called java.net, to enable the rapid development of network applications. Key classes, interfaces, and exceptions in java.net package simplifying the complexity involved in creating client and server programs are:

The Classes

- ContentHandler
- DatagramPacket
- DatagramSocket
- DatagramSocketImpl
- HttpURLConnection
- InetAddress
- MulticastSocket
- ServerSocket
- Socket
- SocketImpl
- URL
- URLConnection
- URLEncoder
- URLStreamHandler

The Interfaces

- ContentHandlerFactory
- FileNameMap
- SocketImplFactory
- URLStreamHandlerFactory

Exceptions

- BindException
- ConnectException
- MalformedURLException
- NoRouteToHostException
- ProtocolException
- SocketException
- UnknownHostException
- UnknownServiceException

3.7 TCP/IP SOCKET PROGRAMMING

The two key classes from the java.net package used in creation of server and client programs are:

- ServerSocket
- Socket

A server program creates a specific type of socket that is used to listen for client requests (server socket). In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept: developers have to open a socket, perform I/O, and close it. Figure 13.5 illustrates key steps involved in creating socket-based server and client programs.

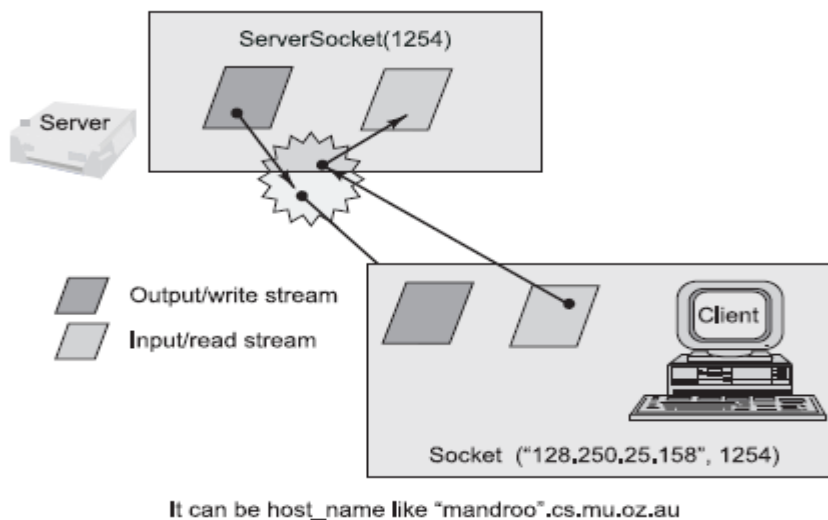


Figure 12 Socket-based client and server programming

3.8 RUNNING SOCKET PROGRAM

Compile both server and client programs and then deploy server program code on a machine which is going to act as a server and client program, which is going to act as a client. If required, both client and server programs can run on the same machine. To illustrate execution of server and client programs, let us assume that a machine called mundroo.csse.unimelb.edu.au on which we want to run a server program. The client program can run on any computer in the network (LAN, WAN, or Internet) as long as there is no firewall between them that blocks communication. Let us say we want to run our client program on a machine called gridbus.csse.unimelb.edu.au. The client program is just establishing a connection with the server and then waits for a message. On receiving a response message, it prints the same to the console. The output in this case is which is sent by the server program in response to a client connection request. It should be noted that once the server program execution is started, it is not possible for any other server program to run on the same port until the first program which is successful using it is terminated. Port numbers are a mutually exclusive resource. They cannot be shared among different processes at the same time.

CHAPTER 4

IMPLEMENTATION

// A Java program for a Client

```
import java.net.*;
import java.io.*;

public class Client
{
    // initialize socket and input output streams
    private Socket socket      = null;
    private DataInputStream input = null;
    private DataOutputStream out  = null;

    // constructor to put ip address and port
    public Client(String address, int port)
    {
        // establish a connection
        try
        {
            socket = new Socket(address, port);
            System.out.println("Connected");

            // takes input from terminal
            input = new DataInputStream(System.in);

            // sends output to the socket
            out = new DataOutputStream(socket.getOutputStream());
        }
        catch(UnknownHostException u)
        {
            System.out.println(u);
        }
        catch(IOException i)
        {
            System.out.println(i);
        }

        // string to read message from input
        String line = "";

        try
        {
```



```
        line = input.readLine();
        out.writeUTF(line);
    }
    catch(IOException i)
    {
        System.out.println(i);
    }

    // close the connection
    try
    {
        input.close();
        out.close();
        socket.close();
    }
    catch(IOException i)
    {
        System.out.println(i);
    }
}

public static void main(String args[])
{
    Client client = new Client("127.0.0.1", 5000);
}
}
```

// A Java program for a Server

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
import java.lang.*;
class dfa{
    static int flag=0;
    public static int Start(char c)
    {
        System.out.println("state: start");
        if(c=='g') return 1;
        else return 0;
    }
}
```

```
public static int Q0(char c)
{
    System.out.println("state: Q0");
    if(c=='g') return 1;
    else if(c=='o') return 2;
    else return 0;
}

public static int Q1(char c)
{
    System.out.println("state: Q1");
    if(c=='g') return 1;
    else if(c=='o') return 3;
    else return 0;
}

public static int Q2(char c,int i)
{
    System.out.println("state: Q2");
    if(c=='g') return 1;
    else if(c=='d'){
        i=i-3;
        System.out.println("state: final");
        System.out.println("string found at index "+i);
        return 4;
    }
    else return 0;
}

}

public static int Qf(char c)
{
    flag=1;
    int state=0;
    return state;
}

}

class pro extends dfa {
```

```
pro(String a)
{
    //String a;
    int state=0;
    Scanner sc=new Scanner(System.in) ;
    //System.out.println("enter the string\n");
    //a=sc.nextLine();
    char c;
    for(int i=0;i<a.length();i++)
    {
        c=a.charAt(i);
        if(state==0)
            state=Start(c);
        else if(state==1)
            state=Q0(c);
        else if(state==2)
            state=Q1(c);
        else if(state==3)
            state=Q2(c,i);
        else if(state==4)
        {
            state=Qf(c);
        }

    }
    if(flag==0)
        System.out.println("string is not accepted");

}
}
```

```
public class Server
{
    //initialize socket and input stream
    private Socket      socket  = null;
    private ServerSocket server = null;
    private DataInputStream in   = null;

    // constructor with port
    public Server(int port)
    {
        // starts server and waits for a connection
        try
        {
            server = new ServerSocket(port);
            System.out.println("Server started");


            System.out.println("Waiting for a client ...");

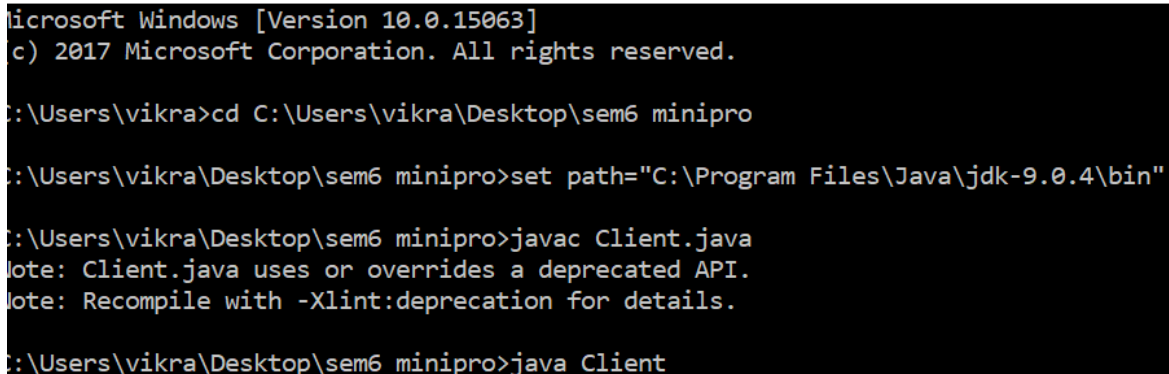
            socket = server.accept();
            System.out.println("Client accepted");

            // takes input from the client socket
            in = new DataInputStream(
                new BufferedInputStream(socket.getInputStream()));

            String line = "";
            try
            {
                line = in.readUTF();
                System.out.println(line);
            }
            catch(IOException i)
```

```
        {  
            System.out.println(i);  
        }  
  
        pro p=new pro(line);  
        System.out.println("Closing connection");  
  
        // close connection  
        socket.close();  
        in.close();  
    }  
    catch(IOException i)  
    {  
        System.out.println(i);  
    }  
}  
  
public static void main(String args[])  
{  
    Server server = new Server(5000);  
}
```

 Command Prompt



```
Microsoft Windows [Version 10.0.15063]  
(c) 2017 Microsoft Corporation. All rights reserved.  
  
C:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro  
  
C:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"  
  
C:\Users\vikra\Desktop\sem6 minipro>javac Client.java  
Note: Client.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
  
C:\Users\vikra\Desktop\sem6 minipro>java Client
```

Figure 13 1st Screenshot of Client Program

Implementation of DFA using Socket Programming

Command Prompt - java Client

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

C:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"

C:\Users\vikra\Desktop\sem6 minipro>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\vikra\Desktop\sem6 minipro>java Client
Connected
```

Figure 14 2nd Screenshot of Client Program

Command Prompt - java Client

```
Exception in thread "main" java.lang.NullPointerException
    at Client.<init>(Client.java:43)
    at Client.main(Client.java:67)

C:\Users\vikra\Desktop\sem6 minipro>javac Client.java
Note: Client.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\vikra\Desktop\sem6 minipro>java Client
Connected
enter the string
good should not be accepted
```

Figure 15 3rd Screenshot of Client Program

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

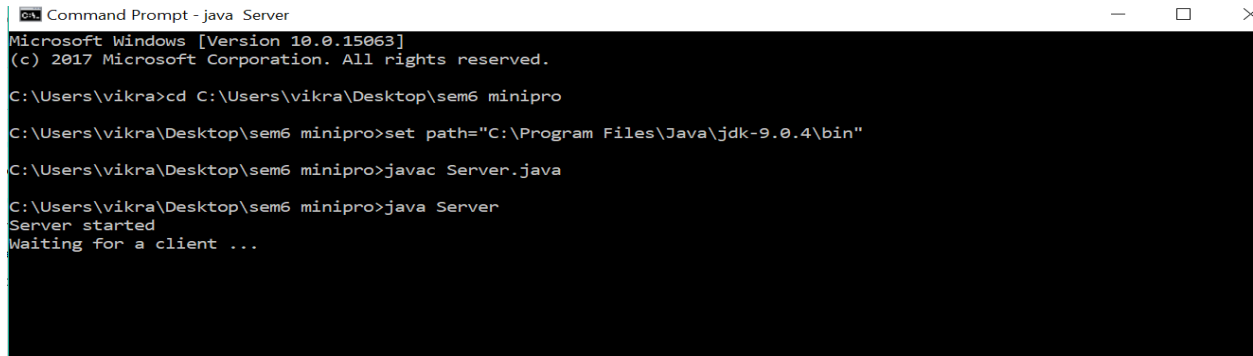
C:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"

C:\Users\vikra\Desktop\sem6 minipro>javac Server.java

C:\Users\vikra\Desktop\sem6 minipro>java Server
```

Figure 16 1st Screenshot of Server Program

Implementation of DFA using Socket Programming



```
Command Prompt - java Server
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

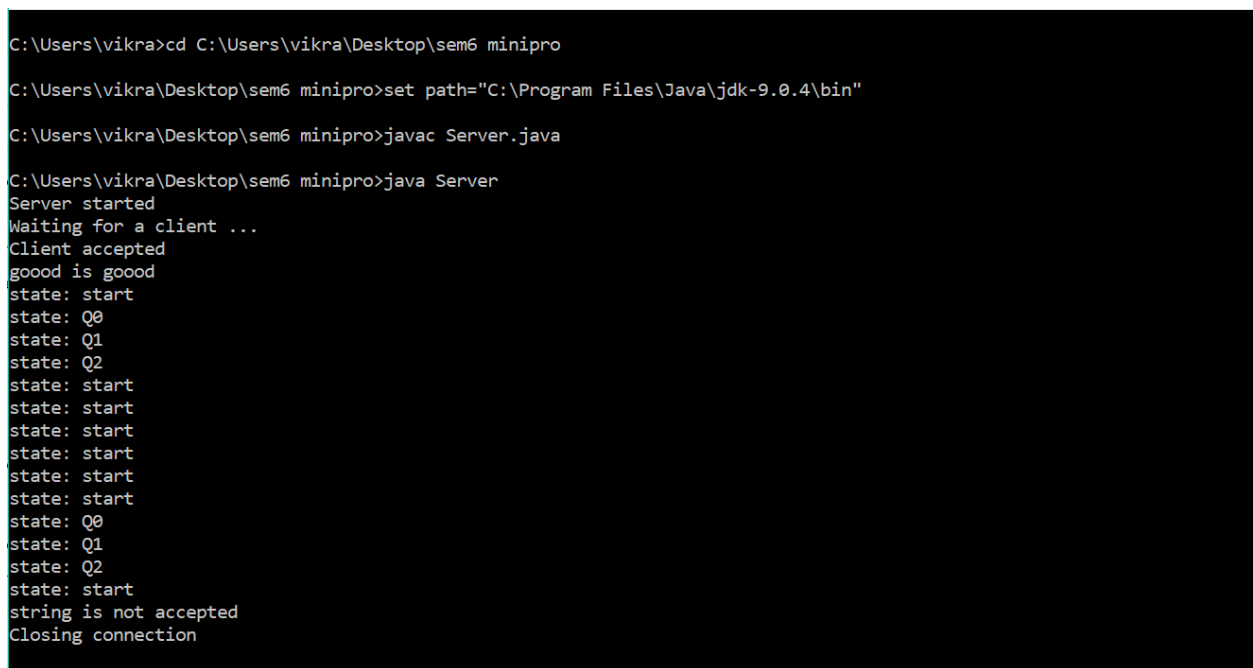
C:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

C:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"

C:\Users\vikra\Desktop\sem6 minipro>javac Server.java

C:\Users\vikra\Desktop\sem6 minipro>java Server
Server started
Waiting for a client ...
```

Figure 17 2nd Screenshot of Server Program



```
C:\Users\vikra>cd C:\Users\vikra\Desktop\sem6 minipro

C:\Users\vikra\Desktop\sem6 minipro>set path="C:\Program Files\Java\jdk-9.0.4\bin"

C:\Users\vikra\Desktop\sem6 minipro>javac Server.java

C:\Users\vikra\Desktop\sem6 minipro>java Server
Server started
Waiting for a client ...
Client accepted
good is good
state: start
state: Q0
state: Q1
state: Q2
state: start
state: start
state: start
state: start
state: start
state: start
state: start
state: Q0
state: Q1
state: Q2
state: start
string is not accepted
Closing connection
```

Figure 18 3rd Screenshot of Server Program

CHAPTER 5

REQUIREMENT SPECIFICATION

5.1 HARDWARE SPECIFICATION

Processor : Pentium-III

Memory : 128MB

Hard Disk : 20GB

5.2 SOFTWARE SPECIFICATION

Operating System : WINDOWS, LINUX, FEDORA

Editor : Emacs, Vim, Sublime

Compiler : JAVA JDK 1. 8.0

CHAPTER 6

CONCLUSION

The objective of the project is achieved that was to build a Client- Server program to implement a DFA. In order to achieve this, I have used socket programming. The Program display each and every state of the strings goes through in a given DFA. And, it displays whether the given string is accepted by the DFA.

REFERENCES

1. www.scribd.com/implementationofdfa
2. www.tutorialspoint.com/deterministicfiniteautomata
3. www.geeksforgeeks.org/toc-finiteautomata
4. Eric Gribkoff, UC Davis “Application of Deterministic Finite Automata” Spring 2013