```
/*Program to implement following transition table:
 -------------------
 |      |  a  | b   |
 | -----------------
 |   q0 | q1  | q0  | q0 is initial state
 |   q1 | q2  | q0  | q2 is final state
 |   q2 | q2  | q0  |
 -------------------                    */
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
int s = 0, i, n;
char a[5];
void state0();
void state1();
void state2();
void main(){
     clrscr();
     cout<<"enter the characters in a string:";
     cin>>n;
     for(int j = 0; j < n; j++)
     for(int i = 0; i < n; i++)
     {
     cin>>a[i];
     if(s == 0)
           {
           state0();
           break;
           }
      else
           if(s == 1)
           {state1();
           break;
           }
      else
           if(s == 2)
           {
           state2();
            break;
            }
     }
     {
     if(s == 2)
     {cout<<"\nString accepted !!";
     }
     else
     {cout<<"\nString not accepted !!";
     }
      }
getch();
}
```

```
void state0()
{
     if(a[i]=='a')
     {
            s = 1;
     }
     else
     if(a[i] == 'b')
     {
            s = 0;
     }
     else
     {
     cout<<"invalid character";
     }
}
void state1()
{
     if(a[i]=='a')
     {
            s = 2;
     }
     else
     if(a[i]=='b')
     {
            s = 0;
     }
     else
     {
     cout<<"invalid character";
     }
}
void state2()
{
     if(a[i]=='a')
     {
            s = 2;
     }
     else
     if(a[i]=='b')
     {
            s = 0;
     }
     else
     {
     cout<<"invalid character";
     }
}
```

(a) A DFA equivalent to NDFA is constructed.

(b) A programming implementation is done that simulates the NDFA.

(iii) Writing a program by simulating a DFA is obtained in (ii) step. A combination of two approaches is actually used in the text processing programs like 'egrep' and fgrep' which are advanced forms of Unix grep program.

Example 2.23 Design an FA to recognise the string 'cat' and 'rat'.

Solution The following diagrams represent on FA that recognises the strings 'cat' and 'rat'

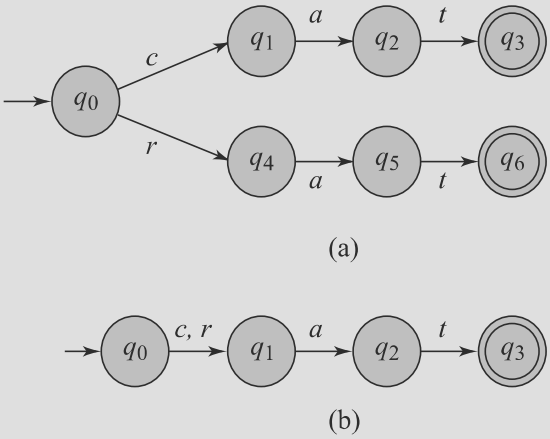

Fig. 2.51 *The DFA to accept strings 'cat' and 'rat'*

In programming implementation we have used the second diagram having four states. The C++ implementation of this FA is given below:

```
/*Programme to implement FSM that accepts string rat or cat */
    #include<iostream.h>
    #include<stdio.h>
    #include<conio.h>
    int s = 0;
    int i, n;
    char a[5];
    void state0();
    void state1();
    void state2();
    void state3();
    void main (){
          clrscr ();
          count <<"enter the characters in a string:";
          cin >>n ;
          for (int j = 0 ; j<n ; j++)
          for (int i = 0 ; i<n ; i++)
          {
          cin>>a [i] ;
```

```
if (s == 0){
        state0( ) ;
        break ;
        }
else if (s == 2) {
        state2( ) ;
        break ;
        }
else if (s == 2) {
        state2( ) ;
        break ;
        }
else if (s == 3) {
        state3() ;
        break ;
        }
}
{
if (s == 3){
        cout <<"\nstring accepted !!";
        }
else {
cout <<"\nstring not accepted !!";
}
}
getch ( ) ;
}
void state0() {
        if ((a [i] == 'r') (a [i] == 'c' )){
s = 1;
else if ((a [i] !='r' (a [i] != 'c'))
{
s = 0;
}
}
void statel() {
if (a [i] == 'a' ) {
s = 2;
}
else if ((a [i] == 'r' (a [i]== 'c')) {
s = 1;
        }
}
void state2() {
if (a [i] == 't' ) {
s = 3;
}
else if (a [i] ! = 't') {
s = 0;
        }
```

```
}
void state3() {
if (a [i] == ' ')
s = 3;
      }
}
// output 1
//enter the characters in a string: 3
//rat
// string accepted !!
//output2
//enter the characters in a string: 3
//mat
//string not accepted !!
```

2.21 LIMITATIONS OF FINITE STATE MACHINES

In this chapter we have seen that finite state machines are very simple. As a consequence, there are limitations to what they can do. For example, it is not possible to design a finite state machine that accepts the language $a^n b^n$, i.e. the set of all strings which consist of a (possible empty) block of a's followed by a (possibly empty) block of b's of exactly the same length.

Basically the language of the form $a^n b^n$ or palindromes is a context free language, which cannot be recognised by a simple computation model like FA. For such types of languages we need a machine more powerful than FA like pushdown automata. FA have certain expressive weaknesses. This also limits their expressive adequacy from the point of view of linguistics, because several linguistic phenomena can only be described by language which cannot be generated by FA. In fact, even the language $a^n b^n$ is linguistically relevant, since several linguistic constructions require 'balanced' structure. Additionally, a finite state machine cannot remember arbitrarily a large amount of information.

Furthermore, Hidden Markov models, which are very common for speech recognition and part of speech tagging, can be seen as a variant of finite state machines, which assign probabilities to their transitions. Thus, finite state machines have their limitations for NLP (Natural Language Processing) purposes.

Summary

- **Automaton** An automaton is defined as a system where information and materials are transformed, transmitted and utilised for performing some processes without direct involvement of human beings.
- **DFA** A finite automaton is DFA if it follows $Q \times \Sigma$ into Q.
- **Representation of DFA** The presentations by which a finite automaton can be represented are: Transition equations, State Transition graph, and Transition table.
- **Transition Graph** A transition diagram (also called transition graph or transition system) is a finite directed labelled graph in which each vertex represents a state and directed edges indicate the transition from one state to another. Edges are labelled with input or input/output.
- **Generalised Transition Graph** Generalised transition graphs are transition graphs where the labels of edges are regular expressions