Converting an NFA to a DFA with programming C++

M. Davoudi-Monfared\textsuperscript{1}, R. shafiezadehgarousi\textsuperscript{2}, E. S. Haghi\textsuperscript{1}, S. Zeinali\textsuperscript{1} and S. Mohebali\textsuperscript{1}

Department of Computer Science, Islamic Azad University, Professor Hesabi of Tafresh Branch, Tafresh, Iran\textsuperscript{1}
Department of Management, Islamic Azad University, Professor Hesabi of Tafresh Branch, Tafresh, Iran\textsuperscript{2}

Received: 19-May-2015; Revised: 03-October-2015; Accepted: 12-October-2015
©2015 ACCENTS

Abstract

In Automata Theory, if a language is recognized by a Non-deterministic finite automaton (NFA), then we must show the existence of deterministic finite automaton (DFA) that also recognizes it. There are many idea and algorithms to convert an NFA in to an equivalent DFA that simulates the NFA. In this paper, we present an algorithm to convert an NFA to a DFA with programming by C++. This approach contains several classes and is so universal that cover converting from NFA to DFA completely.

Keywords

Automata, Convert, NFA, DFA, C++.

1. Introduction

In theory of Automata, deterministic (DFA) and nondeterministic (NFA) finite automaton are useful and important concepts. DFA is the easiest structure for recognizing a formal language. NFA is a generalization of DFA that in NFA several choices may exist for the next state and any point. So every DFA is automatically a NFA and Every NFA has an equivalent DFA \cite{5}. We convert ordinary NFA’s into DFA’s by a process known as the Subset Construction. There are many articles about this concept. In \cite{1} and \cite{4}, Authors showed that for each n-state NFA accepting a finite language, there exists an equivalent DFA. In large NFA’s, the subset construction is very long process end boring. So we need an algorithm to convert every NFA to DFA easily. In this paper we obtain an algorithms with C++ can convert every NFA to a DFA.

These algorithms divided too many classes to be useful and universal for every large NFA. All notations and symbols are standard and the reader referred to \cite{3} for computer sciences and to \cite{6} for programming.

2. Basic concepts and definition

In the following, we describe the definition of DFA and NFA.

**Definition 1** A DFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

1. \(Q\) is a finite set called states.
2. \(\Sigma\) is a finite set called alphabet.
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the transition function.
4. \(q_0 \in Q\) is the start state and
5. \(F \subseteq Q\) is the set of accept states.

**Definition 2** A NFA is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\)

1. \(Q\) is a finite set of states.
2. \(\Sigma\) is a finite alphabet.
3. \(\delta: Q \times \Sigma \rightarrow P(Q)\) is a transition function.
4. \(q_0 \in Q\) is the start state and
5. \(F \subseteq Q\) is the set of accept states.

It is easy to see that NFA is stronger than DFA for reorganization of languages; But DFA has the simple and intelligible structure.

3. Main Algorithm

In this case, an algorithm is given to convert a NFA to the equivalent DFA. One DFA can move from a state to other states while a DFA move from a state to one other state. In fact, with a iteration symbol, NFA can move to next states. However don’t add to the device any more power and yet series of language serve as regular language accept. NFA can be in several states at the same time. We can simulate it with a DFA which has a set of foundation NFA states.

In this program, we used engrafted lists dynamically and define some functions in different classes and grafted list is used in classes. In grafter list, there is
description about the structure and the nodes type. Then the ceases before and after nodes are evaluated. We make a circle for all of the outputs in empty situation. For inserting and omitting the array is used with 100 length to do this work. To do so, at first, the list should be regulate and then array elements copy from the list and its length add to its elements. we have used the important functions in these classes. also, symbols are used such as ‘==’ for the equality of functions name and ‘==’ to allocate and ‘<<’ to oppose in function and Null returns unequal in list. To do the process, it has used two FOR circles and two functions A, B. the final result is shown in array. Then we consider the most input cases and define counters and cases. Then put the empty state for state can’t move to another state with a symbol. Note that showing λ (empty), can move from one state to another without seeing the input. We start from 0 state and make a new state for array output. But we distinguish the out puts for all of the states except empty. the other cases are done as before. for understanding the regulation of program, at the beginning of each state, there are explanations about them. All notations in C" are standard and the reader can refer to [2] in unknown subprogram. In the following, the program is written by C++.

#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
#include<string.h>
#include<iomanip.h>
#define MAX_STATES 500
#define MAX_INPUTS 50

/* I have used Linked Lists to dynamically allocate the states */
/* I found Linked lists are very useful as after transormation states */
/* may be more than one. I have written functions in LinkedList class */
/* and their function easily known by seeing their name */
/* GetAllInfo function returns the length of the LinkedList and copies*/
/* all elements into an Array. I used Templates to input the states */
/* and Input symbols. I preferred to input integers as input symbols */
/* as with integers we can make the program more easier */
/* I overloaded ‘<<’ operator to output the states as q followed by */
/* the number of that particular state. */

// NODE STRUCTURE OF LINKED LIST
template<class T>
struct LinkedListNodeType
{
    LinkedListNodeType<T> *Next;
    T NodeInfo;
    LinkedListNodeType<T> *Prev;
};

---------

#include<iostream.h>
#include<fstream.h>
#include<stdlib.h>
#include<string.h>
#include<iomanip.h>
#define MAX_STATES 500
#define MAX_INPUTS 50

/* I have used Linked Lists to dynamically allocate the states */
/* I found Linked lists are very useful as after transormation states */
/* may be more than one. I have written functions in LinkedList class */
/* and their function easily known by seeing their name */
/* GetAllInfo function returns the length of the LinkedList and copies*/
/* all elements into an Array. I used Templates to input the states */
/* and Input symbols. I preferred to input integers as input symbols */
/* as with integers we can make the program more easier */
/* I overloaded ‘<<’ operator to output the states as q followed by */
/* the number of that particular state. */

// NODE STRUCTURE OF LINKED LIST

#include<iostream.h>
#include<fstream.h>
#include<string.h>
#include<iomanip.h>
#define MAX_STATES 500
#define MAX_INPUTS 50

/* I have used Linked Lists to dynamically allocate the states */
/* I found Linked lists are very useful as after transormation states */
/* may be more than one. I have written functions in LinkedList class */
/* and their function easily known by seeing their name */
/* GetAllInfo function returns the length of the LinkedList and copies*/
/* all elements into an Array. I used Templates to input the states */
/* and Input symbols. I preferred to input integers as input symbols */
/* as with integers we can make the program more easier */
/* I overloaded ‘<<’ operator to output the states as q followed by */
/* the number of that particular state. */
// DEFAULT CONSTRUCTOR WITH NO ARGUMENTS

template<class T>
LinkedList<T>::LinkedList()
{
    // INITIALIZING HEADER [DUMMY NODE]
    Head = new LinkedListNodeType<T>;
    Head->Next = Head;
    Head->Prev = Head;
}

/////////////////////////////////////////////////////////////////////////
////////
// TO FIND AN ELEMENT IN THE LIST IF IT IS NOT THERE RETURN THE APPROPRIATE INSERTING POSITION

template<class T>
void LinkedList<T>::Find(T NodeInfo, bool& found, LinkedListNodeType<T>*& curr)
{
    curr = Head->Next;
    // TRAVERSE UNTIL HEAD OR NODE LESS THE CURRENT NODE
    while ((curr != Head) && (curr->NodeInfo < NodeInfo))
    {
        curr = curr->Next;
    }
    // Set FOUND. If list is empty or value is not located, then found will return FALSE. TRUE is only returned if the value is located.
    if (curr == Head)
    {
        found = false;
    } else
    {
        found = (curr->NodeInfo == NodeInfo);
    }

    // TO INSERT AN ELEMENT IN TO LINKED LIST

template<class T>
void LinkedList<T>::Insert(T NodeInfo)
{
    bool found;
    LinkedListNodeType<T> *p,*noo,*n;
    // REARRANGING THE POINTERS TO POINT TO NEXT NODE OF CURRENT NODE
    cur->Next = n;
    p->Next = noo;
    Head->Next = curr->Next;
    delete curr;
    p = n = curr;
    n = noo;
    return
}

/////////////////////////////////////////////////////////////////////////
////////
// TO TEST WHETHER THE LIST IS EMPTY OR NOT

template<class T>
bool LinkedList<T>::IsEmpty()
{
    LinkedListNodeType<T> *curr = Head->Next;
    int count = 0;
    while (curr != Head)
    {
        count++;
        curr = curr->Next;
    }
    // TESTING COUNT
    if (count > 0)
    {
        return false;
    } else
    {
        return true;
    }

    // TO REMOVE FIRST ELEMENT FROM THE LIST

template<class T>
T LinkedList<T>::RemoveFirst()
{
    T ReturningInfo; // RETURNING NODE
    LinkedListNodeType<T> *curr = Head->Next;
    ReturningInfo = curr->NodeInfo;
    cur->Next = curr->Next;
    delete curr; // RELEASING THE MEMORY
    return ReturningInfo;
}

/////////////////////////////////////////////////////////////////////////
////////
// ADJUST POINTERS IN 'NOO' TO POINT TO 'P' AND 'N'. ALSO PUT VALUE IN NODE

noo->Prev = p;
noo->NodeInfo = NodeInfo;
noo->Next = n;

// ADJUST POINTER IN PREVIOUS AND NEXT NODES TO POINT TO NEW NODE
p->Next = noo;
n->Prev = noo;
}

/////////////////////////////////////////////////////////////////////////
////////
// TO RETURN ALL NODE ELEMENTS IN A
template<class T>
void LinkedList<T>::GetAllInfo(T Array[1000], int& Length)
{
    Length = 0;
    LinkedListNodeType<T> *curr = Head->Next;

    // TRAVERSING ALL NODES AND COPYING ELEMENTS INTO ARRAY
    while (curr != Head)
    {
        Array[Length] = curr->NodeInfo;
        Length++;
        curr = curr->Next;
    }
}

ARRAY WHICH ARE COPIED FROM LINKEDLIST

template<class T>
void LinkedList<T>::GetAllInfo(T Array[1000], int& Length)
{
    Length = 0;
    LinkedListNodeType<T> *curr = Head->Next;

    // TRAVERSING ALL NODES AND COPYING ELEMENTS INTO ARRAY
    while (curr != Head)
    {
        Array[Length] = curr->NodeInfo;
        Length++;
        curr = curr->Next;
    }
}

else
    os << "(NULL)";
return os;
}
private:

int NumberOfElements;// TO COUNT THE
NUMBER OF ELEMENTS OF THE SET
LinkedList<T>ElementsList;// TO STORE THE
ELEMENTS OF THE SET
public:

Set();
void Insert(T Element);
T Remove();
bool IsElementPresent(T Element);
bool IsEmpty();
void GetAllInfo(T Array[1000], int& Length);
Set& Union(Set A, Set B);
Set& operator=(Set A);
bool operator==(Set A);
void SetToNull();
};

;// IMPLEMENTATION SECTION
OF SET

// DEFAULT CONSTRUCTOR FOR SET CLASS
template<class T>
Set<T>::Set() {
    NumberOfElements = 0;
}

// INSERING A ELEMENT INTO SET
template<class T>
void Set<T>::Insert(T Element) {
    if(!IsElementPresent(Element)) {
        NumberOfElements++;
        // INCREASING THE COUNT
        ElementsList.Insert(Element);// INSERTING INTO LINKEDLIST
    }
}

// REMOVING LEAST COST ELEMENT FROM
template<class T>
Set<T>::Remove()
{
template<class T> T Set<T>::IsElementPresent(T Element) {
    bool found;
    LinkedListNodeType<T>* curr;
    ElementsList.Find(Element, found, curr);
    return found;
}

template<class T> void Set<T>::GetAllInfo(T Array[1000], int&Length) {
    ElementsList.GetAllInfo(Array, Length);
}

template<class T> Set<T>& Set<T>::Union(Set A, Set B) {
    T Array[1000];
    intLength,i;
    A.GetAllInfo(Array, Length);
    for(i=0;i<Length;i++)
        Insert(Array[i]);
    B.GetAllInfo(Array, Length);
    for(i=0;i<Length;i++)
        if(!IsElementPresent(Array[i]))
            Insert(Array[i]);
    return *this;
}

template<class T> bool Set<T>::operator==(Set A) {
    T Array[1000], Array1[1000];
    intLength, Length1,i;
    A.GetAllInfo(Array, Length);
    A.GetAllInfo(Array1, Length1);
    if(Length==Length1)
        for(i=0;i<Length;i++)
            if(Array[i]!=Array1[i])
                flag=false;
    return *this;
}

template<class T> Set<T>& Set<T>::operator=(Set A) {
    T Array[1000];
    intLength,i;
    A.GetAllInfo(Array, Length);
    SetToNull();
    for(i=0;i<Length;i++)
        Insert(Array[i]);
    return *this;
}
return flag;
}
else
return false;

template<class T>
void Set<T>::SetToNull()
{
  // CALLING REMOVE MEMBER FUNCTION OF LINKED LIST UNTIL LINKED LIST IS EMPTY
  while(!IsEmpty())
    Remove();
}

int GetNumberOfState(Set<int> ALLStates[MAX_STATES], Set<int> temp, int Length)
{
  for(int i = 0; i<Length; i++)
    if(ALLStates[i] == temp)
      return i;
  return -1;
}

int main()
{
  Set<int> INFA[MAX_STATES][MAX_INPUTS], NFA[MAX_STATES][MAX_INPUTS], DFA[MAX_STATES][MAX_INPUTS], MinDFA[MAX_STATES][MAX_INPUTS], MDFAStatesNo[MAX_STATES];
  int i, j, k, MarkCount = 0, UnMarkCount = 0, States, NoInputs, Temp, Length, Input, NoFinalStates, Final[MAX_STATES], MDFAStatesNo[MAX_STATES];
  bool flag, InputStates[MAX_STATES], FinalStates[MAX_STATES], MinDFAInputStates[MAX_STATES], MinDFAFinalStates[MAX_STATES];

  cout << "Enter Number of states : ";
  cin >> States;
  cout << "Enter Number of Inputs(without counting lambda) : ";
  cin >> NoInputs;
  cout << "If there is Lambda transactions[1 for true, 0 for false] ";
  cin >> lambda;

  for(i = 0; i < States; i++)
  {
    if(lambda == 1)
    {
      cout << "Enter the No of Transitions for State " << i << " With Lambda ";
      cin >> Temp;
      for(k = 0; k < Temp; k++)
      {
        cout << "Enter transformed State " << k + 1 << " for state " << i << " with Lambda ";
        cin >> NextState;
        LambdaT[i].Insert(NextState);
      }
    }
    for(j = 0; j < NoInputs; j++)
    {
      cout << "Enter the No of Transitions for State " << i << " With Input " << j << " ";
      cin >> Temp;
      for(k = 0; k < Temp; k++)
      {
        cout << "Enter transformed state " << k + 1 << " for state " << i << " with input " << j << " ";
        cin >> NextState;
        INFA[i][j].Insert(NextState);
      }
    }
  }

  cout << "Enter the Input State Number :";
  cin >> Input;

  cout << "Enter the No of Final States :";
  cin >> NoFinalStates;

  for(k = 0; k < NoFinalStates; k++)
  {
    cout << "Enter " << k + 1 << " Final state ";
    cin >> Final[k];
  }

  DFAStates = States;
  cout << "Here first state(first column) represents a state" << endl << "After lambda transition and from 2nd to last" << endl << "column represents a state after input symbols" << endl << "from 0 to 'numberofinputsymbols'-1' and "
}
<<endl<<"each of the row represents transformations"<<endl<<"for state start from '0' to 'numberofstates-1"<<"un";cout<<"Input NFA is as follows:"<<endl;for(i=0;i<States;i++){
cout<<LambdaT[i];for(j=0;j<NoInputs;j++){
cout<<setw(-20)<<INFA[i][j];
}cout<<endl;
}
{char c;cout<<"Input char for prompt":cin>>c;if(lambda==1){for(i=0;i<States;i++){
temp.SetToNull();temp=LambdaT[i];temp.Insert(i);do{
temp.GetAllInfo(Array,Length);for(k=0;k<Length;k++)
temp.Union(temp,LambdaT[Array[k]]);temp.GetAllInfo(Array,Length1);if((Length1-Length)==0)break;}while(1);for(j=0;j<NoInputs;j++){
temp.GetAllInfo(Array,Length);temp1.SetToNull();//cout<<endl<<temp<<endl;{char c;cin>>c;}for(k=0;k<Length;k++){
if(!(INFA[Array[k]][j].IsEmpty()))temp1.Union(temp1,INFA[Array[k]][j]);//cout<<endl<<temp1<<endl;{char c;cin>>c;}for(k=0;k<Length;k++)
}
do{
temp1.GetAllInfo(Array,Length);for(k=0;k<Length;k++)temp1.Union(temp1,LambdaT[Array[k]]);temp1.GetAllInfo(Array,Length1); //cout<<endl<<"After lambdatrns"<<endl<<temp1<<endl;{char c;cin>>c;}if((Length1-Length)==0)break;}
}while(1);
DFA[i][j]=temp1;
NFA[i][j]=temp1;//cout<<endl<<"Fianllay"<<endl<<temp1<<endl;{char c;cin>>c;}
}
}
else{
for(i=0;i<States;i++){
for(j=0;j<NoInputs;j++){
DFA[i][j]=NFA[i][j];
}
}
}


cout<<"*************** AFter removing lambda transitions ***************"<<endl;
cout<<"first row represents transformations for state 0 and later rows"<<endl<<"represents states incresing by '1'. Columns represents"<<endl<<"transformations for each of the input symbol from 0 to"<<endl<<"numberofinputsymbols-1 Lambda transformations are removed"<<endl;
for(i=0;i<States;i++){
for(j=0;j<NoInputs;j++){
}
}


cout<<endl;
{}
cout<<"*************** //After removing lambda transitions ......Now converting into DFA ************/***************//***************
ALLStates[0].Insert(0);
temp.SetToNull();
DFAstates=1;//States;
do{
for(i=0;i<DFAstates;i++){
for(j=0;j<NoInputs;j++){
}
}
}
flag=false;
for(k=0;k<DFAstates;k++)
{
    if(DFA[i][j]==ALLStates[k])
        flag=true;
    }
if(flag==false)
{
    //cout<<"Found "<<DFA[i][j]<<endl;char;c;cin>>c;
    break;
}
if(flag==false)
break;
if(flag==false)
{
    ALLStates[DFAstates]=DFA[i][j];
    DFA[i][j].GetAllInfo(Array,Length);
    for(i=0;i<NoInputs;i++)
    {
        temp.SetToNull();
        for(j=0;j<Length;j++)
        {
            temp.Union(temp,NFA[Array[j]][i]);
            //cout<<"After union with"<<Array[j]<<endl;char;c;cin>>c;
        }
        //cout<<"Finally"<<DFAstates<<""<<i<<""<<temp<<endl;
        {char c;cin>>c;}
    }
    DFAstates++;
cout<<"q"<<i;
}
}
cout<<endl;
{char c;cout<<"Input char for Prompt":cin>>c;}

/******* minimization of a Dfa
**********/
//MarkCount = 0;UnMarkCount = 0;

for(i=0;i<DFAsates-1;i++)
{
for(j=i+1;j<DFAsates;j++)
{
if( ((FinalStates[i]==true)&&(FinalStates[j]==false))
 || ((FinalStates[i]==false)&&(FinalStates[j]==true)) )
{
Mark[MarkCount].Insert(i);
Mark[MarkCount].Insert(j);
MarkCount++;
}
else
{
if(i != j)
{
UnMark[UnMarkCount].Insert(i);
UnMark[UnMarkCount].Insert(j);
UnMarkCount++;
}
}
}
/*cout<<endl<<"Marked are";
for(i=0;i<MarkCount;i++)
{cout<<endl<<Mark[i];
}
*/
/*cout<<endl<<"un marked are";
for(i=0;i<UnMarkCount;i++)
{if(!UnMark[i].IsEmpty())
{cout<<endl<<UnMark[i];
}
}
*/

/******* reduce algorithm
**********/

/* Using mark procedure to find all pairs of Indistinguishable states */
for(i=0;i<UnMarkCount-1;i++)
{
if(!UnMark[i].IsEmpty())
for(j=0;j<UnMarkCount;j++)
{
if(!UnMark[j].IsEmpty())
{
UnMark[i].GetAllInfo(Array,Length);
for(k=0;k<Length;k++)
{if(UnMark[i].IsElementPresent(Array[k])==true)
break;
if(UnMark[i].IsElementPresent(Array[k])==true)
{
temp.SetToNull();
temp.Union(UnMark[i],UnMark[j]);
UnMark[j].SetToNull();
UnMark[i]=temp;
}
}
}
}

for(k=0;k<MarkCount;k++)
{
if(MinTemp == Mark[k])
{
//cout<<endl<<"Finally"<<DFAsates<<"*"<<i<<"*
	<temp_p<<endl<<temp<<endl;{char c;cin>>c;}
Mark[MarkCount]=UnMark[i];
MarkCount++;
UnMark[i].SetToNull();
j=NoInputs;
break;
}
}
*/

/* Finding Sets of Indistinguishable states that are
Not Marked */
cout<<endl<<"Finally ,After Merging UnMarked ...."
<<endl<<"i.e. finding distinguishable states:"<<endl;
cout<<endl<<"sets of distinguishable states which
may contain more"
<<endl<<"more than one indistinguishable
state:"<<endl;
for(i=0;i<UnMarkCount;i++)
{
if(!UnMark[i].IsEmpty())
{
cout<<endl<<UnMark[i];
}
}
for(i=0;i<DFAstates;i++)
{
ALLStates[i].SetToNull();
ALLStates[i].Insert(i);
ALLStates[i].GetAllInfo(Array,Length);
for(k=0;k<UnMarkCount;k++)
{
if(!UnMark[k].IsEmpty())
{
if(UnMark[k].IsElementPresent(Array[0])==true)
break;
}
}
if(UnMark[k].IsElementPresent(Array[0])==true)
DFA[i][j]=UnMark[k];
cout<<DFA[i][j];
}
}

for(i=0;i<DFAstates;i++)
{
char c;cout<<"Input char for Prompt:";cin>>c;
}

for(i=0;i<DFAstates;i++)
{
if(!DFA[i].IsEmpty())
{
cout<<endl<<"For state "<ALLStates[i]" Trans are:";
for(j=0;j<NoInputs;j++)
{
DFA[i][j].GetAllInfo(Array,Length);
for(k=0;k<UnMarkCount;k++)
{
if(!UnMark[k].IsEmpty())
{
if(UnMark[k].IsElementPresent(Array[0])==true)
break;
}
}
if(UnMark[k].IsElementPresent(Array[0])==true)
DFA[i][j]=UnMark[k];
cout<<DFA[i][j];
}
cout<<endl;
}

for(i=0;i<DFAstates;i++)
{
if(MDFAStatesNo[i]==i)
{
for(j=0;j<NoInputs;j++)
{
DFA[i][j].GetAllInfo(Array,Length);
for(k=0;k<UnMarkCount;k++)
{
if(!UnMark[k].IsEmpty())
{
if(UnMark[k].IsElementPresent(Array[0])==true)
break;
}
}
if(UnMark[k].IsElementPresent(Array[0])==true)
DFA[i][j]=UnMark[k];
cout<<DFA[i][j];
}
cout<<endl;
}
}
{ MinDFA[Temp][j]=DFA[i][j];
cout<<MinDFA[Temp][j];
} Temp++;
cout<<endl;
}
{ char c;cout<<"Input char for Prompt:";cin>>c;
cout<<"After reordering numbers of states"<<endl;
cout<<"some states may contain more than one state"
<<endl<<"making those states number into one state
number:"<<endl;
cout<<endl<<"Final minimized DFA is ":"<<endl;
for(i=0;i<DFAsates;i++)
{
cout<<"For q"<<i<<" States are ";
for(j=0;j<NoInputs;j++)
{
 Temp=GetNumberOfState(MDFAStates,MinDFA[i][
 j],DFAsates);
MinDFA[i][j].SetToNull();
MinDFA[i][j].Insert(Temp);
cout<<setw(-15)<<MinDFA[i][j];
}
cout<<endl;
}
{ char c;cout<<"Input char for Prompt:";cin>>c;
for(i=0;i<DFAsates;i++)
{
 flag=false;
MDFAStates[i].GetAllInfo(Array,Length);
for(j=0;j<Length;j++)
{
 if(InputStates[Array[j]]==true)
{
 flag=true;
break;
}
MinDFAInputStates[i]=flag;
flag=false;
for(j=0;j<Length;j++)
{
 if(FinalStates[Array[j]]==true)
{
 flag=true;
break;
}
MinDFAFinalStates[i]=flag;
}
}
cout<<"starting States of minimized DFA are ";
for(i=0;i<DFAsates;i++)
{
 if(MinDFAInputStates[i]==true)
{
 q"<<i;
}
cout<<endl;
cout<<"Final States of minimized DFA are ";
for(i=0;i<DFAsates;i++)
{
 if(MinDFAFinalStates[i]==true)
{
 q"<<i;
}
cout<<endl;
}
cout<<"Input char for Prompt:";cin>>c;
return 0;
}

4. Conclusion

We see that the program is so long. But it is a universal program for converting a NFA to a DFA. In this program, we used engrafted lists dynamically and define some functions in different classes and grafted list is used in classes. In grafter list, there is description about the structure and the nodes type. Then ceases before and after nodes are evaluated. We make a circle for all of the outputs in empty situation. For inserting and omitting the array is used with 100 lengths to do this work. To do so, at first ,the list should be regulate and then array elements copy from the list and its length add to its elements we have used the important functions in these classes also ,symbols are used such as ‘==’ for the equality of functions name and ‘==’ to allocate and ‘<<’ to oppose in function and Null returns unequal in list . To do the process, it has used two FOR circles and two functions A, B. the final result is shown in array .Then we consider the most input cases and define counters and cases. Then put the empty state for state can’t move to another state with a symbol. Note that showing λ (empty), can move from one state to another without seeing the input. We start from 0 state and make a new state for array output. But we distinguish the out puts for all of the states except empty .the other cases are done as before.

For converting large NFA such that contains empty alphabet as a transfer function, we force to define different classes. If the empty alphabet is not in transfer functions of NFA, we can design a simple and routine program for converting. This program
answers for any NFA included large NFA, simple NFA without empty alphabet and etc.

For future work we try to simulate Turing machine in C++ programming. The Turing machine works as a computer and can solve many computational algorithms. DFA and NFA are weak than the Turing machine. If it is possible, then any DFA and NFA simulate simply.

**Acknowledgment**

The authors would like to thank the referee for the valuable suggestions and comments. Also they wish to thank Islamic Azad University Tafresh branch for financial support.

**References**