College of Engineering Trivandrum

Compiler Design Lab

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

September 16, 2020

CS431 - Compiler Design Lab · 2020 ·

Exp 6

1 NFA to DFA

1.1 Aim

Write program to convert NFA to DFA.

1.2 Theory

NFA

- NFA stands for non-deterministic finite automata. It is easy to construct an NFA than DFA for a given regular language.
- The finite automata are called NFA when there exist many paths for specific input from the current state to the next state.
- Every NFA is not DFA, but each NFA can be translated into DFA.
- NFA is defined in the same way as DFA but with the following two exceptions, it contains multiple next states, and it contains ϵ transition.
  - Q: finite set of states
  - Σ: finite set of the input symbol
  - q0: initial state
  - F: final state
  - δ: Transition function
  - δ : Q x $\Sigma \rightarrow 2^Q$

DFA

- In a DFA, for a particular input character, the machine goes to one state only.
- A transition function is defined on every state for every input symbol.
- Also in DFA null (or ϵ) move is not allowed,
- i.e., DFA cannot change state without any input character.
- DFA consists of 5 tuples Q, Σ , q, F, δ.
  - Q : set of all states.
  - Σ : set of input symbols. (Symbols which machine takes as input)
  - q : Initial state. (Starting state of a machine)
  - F : set of final state.
  - δ : Transition Function, defined as δ : Q X $\Sigma \rightarrow$ Q.

1.3 Algorithm

Algorithm 1: Algorithm to convert NFA to DFA

```
function CONVERTTODFA ( nfa )
    Initialize an empty object of type DFA with variable name dfa
    Initialize dfa . num_alphabets = nfa . num _alphabets , i = 0
    Intialize a set lazySet which stores subsets of Q and store {0} in it Create a new row of
    size dfa . num_alphabets and insert into dfa .
    table and initialize all values to -1.
    While i < lazySet . size ()
        Iterate through each of the input symbol j in E
            Initialize an empty set of states reachable and a variable next = -1
            Iterate i through each element in lazySet [ i ] and push
            into reachable the set nfa . table [ i ][ j ]
            Check if reachable is already in lazySet . If yes ,
            the get the value of next from     lazySet . If not , then
                Insert into lazySet , the set reachable and
                set next = lazySet . size ()
                Insert next into dfa . finalStates if any element
                in reachable is a final state of the original nfa
                Create a new row of size dfa . num_alphabets and
                insert into dfa . table and
                initialize all values to -1.
                dfa . table [ i ][ j ] = next
        Increment i
    Return dfa as the DFA .
end function
```

1.4 Code

```cpp
//CPP program to convert NFA into DFA
#include <bits/stdc++.h>
using namespace std;
void print_header(int m)
{
    cout << "State";
    for (int j = 0; j < m; ++j)
    {
        char ch = char(int('a') + j);
        cout << "\t" << ch;
    }
    cout << endl;
}
set<int> split_Add(string s)
{
    int size = s.size();
    set<int> res;
    int start = 0;
    for (int i = 0; i < size; ++i)
    {
        if (s[i] == ',')
        {
            string sub = s.substr(start, i - start);
            start = i + 1;
            int temp = stoi(sub);
            res.insert(temp);
        }
    }
    string sub = s.substr(start, size - start);
    int temp = stoi(sub);
    res.insert(temp);
    return res;
}
void print(vector<vector<set<int>>> table)
{
    for (auto x : table)
    {
        for (auto y : x)
        {
            for (auto z : y)
            {
                cout << z << ",";
            }
            cout << "\t";
        }
        cout << endl;
    }
}
int main()
{
    cout << "Enter the number of states: ";
    int n, m;
    cin >> n;
    cout << "Enter the number of input symbols: ";
    cin >> m;
    vector<vector<set<int>>> table(n, vector<set<int>>(m));
    print_header(m);
    for (int i = 0; i < n; ++i)
    {
        cout << i << "\t";
        for (int j = 0; j < m; ++j)
        {
            string s;
            cin >> s;
            set<int> to_states;
            if (s != "-")
                to_states = split_Add(s);
            table[i][j] = to_states;
        }
    }
    //print(table);
    int start, last, temp, temp1;
    set<int> s, f;
    cout << "Enter the number of start states: ";
```

```
    cin >> start;
    cout << "Enter the start states: ";
    for (int i = 0; i < start; ++i)
    {
        cin >> temp;
        s.insert(temp);
    }
    cout << "Enter the number of final states: ";
    cin >> last;
    cout << "Enter the final states: ";
    for (int i = 0; i < last; ++i)
    {
        cin >> temp;
        f.insert(temp);
    }
    vector<vector<set<int>>> dfa;
    set<set<int>>
        present;
    map<set<int>, int> state_num;
    queue<set<int>> elem;

    int dfa_state = 0;
    elem.push(s);
    present.insert(s);
    while (!elem.empty())
    {

        vector<set<int>> dfa_row;

        set<int> current = elem.front();
        state_num[current] = dfa_state;
        dfa_state++;
        dfa_row.push_back(current);
        elem.pop();
        for (int i = 0; i < m; ++i)
        {
            set<int> new_state;
            for (auto x : current)
            {
                for (auto y : table[x][i])
                {
                    new_state.insert(y);
                }
            }
            dfa_row.push_back(new_state);
            if (present.find(new_state) == present.end())
            {
                present.insert(new_state);
                elem.push(new_state);
            }
        }
        dfa.push_back(dfa_row);
    }
    print_header(m);
    for (int i = 0; i < dfa_state; ++i)
    {
        for (int j = 0; j < m + 1; ++j)
        {
            cout << state_num[dfa[i][j]] << "\t";
        }
        cout << endl;
    }
    cout << "Final states are: ";
    for (auto x : present)
    {
        for (auto y : x)
        {
            if (f.find(y) != f.end())
            {
                cout << state_num[x] << " ";
                break;
            }
        }
    }
    cout << endl;
    return 0;
```

```
}
```

1.5 Output

```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ nfa-dfa.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 3
Enter the number of input symbols: 2
State   a       b
0       0,2     1
1       1,2     2
2       0,2     1
Enter the number of start states: 1
Enter the start states: 0
Enter the number of final states: 2
Enter the final states: 0 2
State   a       b
0       1       2
1       1       2
2       3       4
3       5       3
4       1       2
5       5       3
Final states are: 0 5 1 3 4
```


```
abhishek@hephaestus:~/Desktop/S7/CD LAB$ g++ nfa-dfa.cpp
abhishek@hephaestus:~/Desktop/S7/CD LAB$ ./a.out
Enter the number of states: 3
Enter the number of input symbols: 2
State   a       b
0       0,2     1
1       1,2     2
2       0,2     1
Enter the number of start states: 1
Enter the start states: 0
Enter the number of final states: 2
Enter the final states: 0 2
State   a       b
0       1       2
1       1       2
2       3       4
3       5       3
4       1       2
5       5       3
Final states are: 0 5 1 3 4
```


1.6 Result

Implemented the program toconvert ϵ-NFA to a NFA in CPP. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.