College of Engineering Trivandrum

Compiler Design Lab

Abhishek Manoharan
S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

September 9, 2020

CS431 - Compiler Design Lab · 2020 ·

Exp 5

1 ϵ- NFA to NFA

1.1 Aim

Write program to convert NFA with ϵ transition to NFA without ϵ transition

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 $\Sigma \rightarrow 2^Q$

1.3 Algorithm

Algorithm 1: Algorithm to find Epsilon Closure

```
function CONVERTTONFA ( enfa )
    Initialize an empty object of type NFA with variable name t
    Initialize t. numstates = enfa. numstates ,
    t. numalphabets = enfa .numalphabets and
    t. finalstates = enfa . finalstates
    Iterate through each of the state i in Q
        Initialize l to the closure of state i of e - N F A enfa
        Iterate through each of the input symbol j in Z
            Initialize an empty list of states f
            Iterate through each state k in l
            Add all states of enfa . transitiontable [ k ][ j + 1] to f
        Remove all the duplicates from f
        Compute the e - closure c of f
        Set t . transitiontable [ i ][ j ] = c
    Return t as the NFA without e - transitions corresponding to the
    e - NFA enfa
end function
```

1.4 Code

```cpp
// CPP program to covert epsilon nfa to non epsilon nfa
#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;
}

unordered_set<int> split_Add(string s)
{
    int size = s.size();
    unordered_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;
}
int main()
{
    cout << "Enter the number of states: ";
    int n, m;
    cin >> n;
    cout << "Enter the number of input symbols: ";
    cin >> m;
    vector<vector<unordered_set<int>>> table(n, vector<unordered_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;
            unordered_set<int> to_states;
            if (s != "-")
                to_states = split_Add(s);
            table[i][j] = to_states;
        }
    }

    int start, last, temp, temp1;
    unordered_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);
    }
    int epsilon;
    cout << "Enter the number of epsilon transition: ";
    cin >> epsilon;
    vector<pair<int, int>> eps;
    cout << "From\tTo" << endl;
    for (int i = 0; i < epsilon; ++i)
    {
        cin >> temp >> temp1;
        eps.push_back({temp, temp1});
    }

    queue<int> check;
    vector<unordered_set<int>> closures; // to store the set of states
    // cout << "Epsilon Closures are: \n";
    for (int i = 0; i < n; ++i)
    {
        vector<int> visited(n, 0);
        if (visited[i] != 1)
        {
            unordered_set<int> building;
            building.insert(i);
            //cout << i << ": ";
            visited[i] = 1;
            //cout << "{ " << i;
            for (auto x : eps)
            {
                if (x.first == i)
                {
                    check.push(x.second);
                }
            }
            while (!check.empty())
            {
                int c = check.front();
                check.pop();
                building.insert(c);
                visited[c] = 1;
                //cout << ", " << c;
                for (auto x : eps)
                {
                    if (x.first == c && visited[x.second] != 1)
                    {
                        check.push(x.second);
                    }
                }
            }
            closures.push_back(building);
        }
    }
    cout << endl;
    cout << "New NFA without epsilon transition\n";
    print_header(m);
    for (int i = 0; i < n; ++i)
    {
        cout << i << "\t";
        for (int j = 0; j < m; ++j)
        { // for the new state k where each transitions go
            unordered_set<int> check;
            for (auto x : closures[i])
            {
                for (auto y : table[x][j])
                {
                    if (check.find(y) == check.end())
                    {
                        check.insert(y);
                        for (auto z : closures[y])
                        {
                            if (check.find(z) == check.end())
                                check.insert(z);
                        }
                    }
                }
            }
```

```
            if (check.empty())
            {
                cout << "-";
            }
            else
            {
                for (auto x : check)
                {
                    cout << x << ",";
                }
            }

            cout << "\t";
        }
        cout << endl;
    }
    cout << "Start state is: ";
    for (int i = 0; i < n; ++i)
    {
        for (auto x : closures[i])
        {
            if (s.find(x) != f.end())
            {
                cout << i << " ";
                break;
            }
        }
    }
    cout << endl;
    cout << "Final state is: ";
    for (int i = 0; i < n; ++i)
    {
        for (auto x : closures[i])
        {
            if (f.find(x) != f.end())
            {
                cout << i << " ";
                break;
            }
        }
    }
    cout << endl;
    return 0;
}
```

1.5 Output

```
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       1
1       1,2     2
2       0       1
Enter the number of start states: 1
Enter the start states: 0
Enter the number of final states: 1
Enter the final states: 2
Enter the number of epsilon transition: 1
From    To
0       2

New NFA without epsilon transition
State   a       b
0       2,0,    1,
1       1,2,    2,
2       2,0,    1,
Start state is: 0
Final state is: 0 2
```


```
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       1
1       1,2     2
2       0       1
Enter the number of start states: 1
Enter the start states: 0
Enter the number of final states: 1
Enter the final states: 2
Enter the number of epsilon transition: 1
From    To
0       2

New NFA without epsilon transition
State   a       b
0       2,0,    1,
1       1,2,    2,
2       2,0,    1,
Start state is: 0
Final state is: 0 2
```

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.