College Of Engineering Trivandrum

Compiler Design Lab

CS431

Submitted By:
Neethu S
S6 CSE Roll No:42

TVE18CS043

Department of Computer Science

October 15, 2021

Contents

1 NFA to DFA Conversion

1.1 Aim

Write a program to convert NFA without ϵ transition to DFA transition.

1.2 Theory

An NFA can have zero, one or more than one move from a given state on a given input symbol. An NFA can also have NULL moves (moves without input symbol). On the other hand, DFA has one and only one move from a given state on a given input symbol.

An NFA is represented formally by a 5-tuple, (Q, Σ, Δ, q0, F), consisting of

- a finite set of states Q
- a finite set of input symbols Σ
- a transition function $\Delta : Q \times \Sigma \rightarrow P(Q)$
- an initial (or start) state q0 ϵ Q
- a set of states F distinguished as accepting (or final) states $F \subseteq Q$

Here, P(Q) denotes power set of Q.

1.3 Algorithm

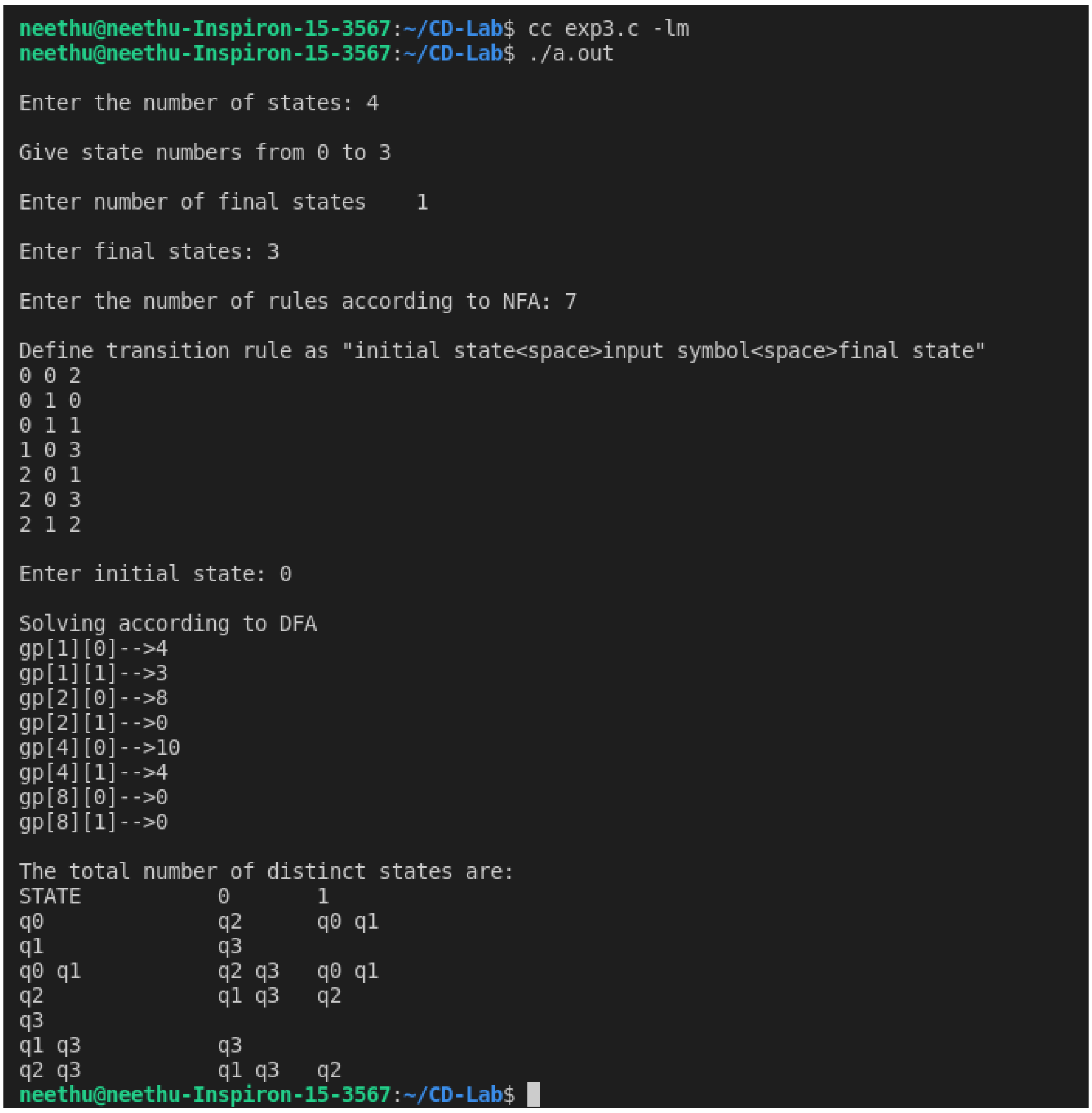
Suppose there is an NFA N (Q, Σ, Δ, q0, F) which recognizes a language L.
Then the DFA D (Q', Σ', Δ', q0, F') can be constructed for language L as:

```
Step 1: Initially Q' = null
Step 2: Add q0 to Q'
Step 3: For each state in Q', find the possible set of states for each input
        symbol using transition function of NFA. If this set of states is
        not in Q', add it to Q'
Step 4: Final state of DFA will be all states with contain F(final states of NFA)
```

1.4 Code

```
#include <stdio.h>
#include <string.h>
#include <math.h>

int ninputs;
int dfa[100][2][100] = {0};
int state[10000] = {0};
char ch[10], str[1000];
int go[10000][2] = {0};
int arr[10000] = {0};
int main()
{
    int st, fin, in;
    int f[10];
    int i, j = 3, s = 0, final = 0, flag = 0, curr1, curr2, k, l;
    int c;

    printf("\nEnter the number of states: ");
    scanf("%d", &st);

    printf("\nGive state numbers from 0 to %d\n", st - 1);

    for (i = 0; i < st; i++)
        state[(int)(pow(2, i))] = 1;

    printf("\nEnter number of final states\t");
    scanf("%d", &fin);

    printf("\nEnter final states: ");
    for (i = 0; i < fin; i++)
    {
        scanf("%d", &f[i]);
    }

    int p, q, r, rel;

    printf("\nEnter the number of rules according to NFA: ");
    scanf("%d", &rel);

    printf("\nDefine transition rule as
    \"initial state<space>input symbol<space>final state\"\n");

    for (i = 0; i < rel; i++)
    {
        scanf("%d %d %d", &p, &q, &r);
        dfa[p][q][r] = 1;
    }

    printf("\nEnter initial state: ");
    scanf("%d", &in);

    in = pow(2, in);

    i = 0;

    printf("\nSolving according to DFA\n");

    int x = 0;
    for (i = 0; i < st; i++)
    {
        for (j = 0; j < 2; j++)
        {
            int stf = 0;
            for (k = 0; k < st; k++)
            {
                if (dfa[i][j][k] == 1)
                    stf = stf + pow(2, k);
            }

            go[(int)(pow(2, i))][j] = stf;
            printf("gp[%d][%d]-->%d\n", (int)(pow(2, i)), j, stf);
            if (state[stf] == 0)
                arr[x++] = stf;
            state[stf] = 1;
        }
    }

    for (i = 0; i < x; i++)
    {
        for (j = 0; j < 2; j++)
        {
            int new = 0;
            for (k = 0; k < st; k++)
            {
                if (arr[i] & (1 << k))
                {
                    int h = pow(2, k);

                    if (new == 0)
                        new = go[h][j];
                    new = new | (go[h][j]);

                    go[arr[i]][j] = new;
                }
            }
            if (state[new] == 0)
            {
                arr[x++] = new;
                state[new] = 1;
            }
        }
    }

    printf("\nThe total number of distinct states are:\n");

    printf("STATE\t\t0\t1\n");

    for (i = 0; i < 10000; i++)
    {
        int x = 0;
        if (state[i] == 1)
        {
            int y = 0;
            if (i == 0)
                continue;
            else
                for (j = 0; j < st; j++)
                {
                    x = 1 << j;
                    if (i & x)
                    {
                        printf("q%d ", j);
                        y = y + pow(2, j);
                    }
                }
            printf("\t\t");
            for (j = 0; j < st; j++)
            {
                x = 1 << j;
                if (x & (go[y][0]))
                {
                    printf("q%d ", j);
                }
            }
            printf("\t");
            for (j = 0; j < st; j++)
            {
                x = 1 << j;
                if (x & (go[y][1]))
                {
                    printf("q%d ", j);
                }
            };
            printf("\n");
        }
    }
    return 0;
}
```

1.5 Output



```
neethu@neethu-Inspiron-15-3567:~/CD-Lab$ cc exp3.c -lm
neethu@neethu-Inspiron-15-3567:~/CD-Lab$ ./a.out

Enter the number of states: 4

Give state numbers from 0 to 3

Enter number of final states    1

Enter final states: 3

Enter the number of rules according to NFA: 7

Define transition rule as "initial state<space>input symbol<space>final state"
0 0 2
0 1 0
0 1 1
1 0 3
2 0 1
2 0 3
2 1 2

Enter initial state: 0

Solving according to DFA
gp[1][0]-->4
gp[1][1]-->3
gp[2][0]-->8
gp[2][1]-->0
gp[4][0]-->10
gp[4][1]-->4
gp[8][0]-->0
gp[8][1]-->0

The total number of distinct states are:
STATE           0       1
q0              q2      q0 q1
q1              q3
q0 q1           q2 q3   q0 q1
q2              q1 q3   q2
q3
q1 q3           q3
q2 q3           q1 q3   q2
neethu@neethu-Inspiron-15-3567:~/CD-Lab$ 
```

1.6 Result

Implemented the program to convert NFA without epsilon-transition to DFA using C language in Ubuntu 20.04 and the above outputs were obtained.