Department of Computer Science and Engineering, SVNIT, Surat
System Software
Lab Assignment 1

U20CS005
BANSI MARAKANA

Problem Statement:
1. Write a program to convert a regular expression to DFA.

Solution:
A regular expression can be converted to DFA in two steps:

a. Convert a regular expression to NFA.
b. Convert NFA obtained in above step to DFA.

Program to convert regular expression to NFA

```c
#include <stdio.h>
#include <string.h>
int main()
{
    char reg[20];
    int q[20][3], i = 0, j = 1, len, a, b;
    for (a = 0; a < 20; a++)
        for (b = 0; b < 3; b++)
            q[a][b] = 0;
    printf("Enter regular expression: ");
    scanf("%s", reg);
    len = strlen(reg);
    while (i < len)
    {
        if (reg[i] == 'a' && reg[i + 1] != '|' && reg[i + 1] != '*')
        {
            q[j][0] = j + 1;
            j++;
        }
        if (reg[i] == 'b' && reg[i + 1] != '|' && reg[i + 1] != '*')
        {
            q[j][1] = j + 1;
            j++;
        }
        if (reg[i] == 'e' && reg[i + 1] != '|' && reg[i + 1] != '*')
```

```
        {
            q[j][2] = j + 1;
            j++;
        }
        if (reg[i] == 'a' && reg[i + 1] == '|' && reg[i + 2] == 'b')
        {
            q[j][2] = ((j + 1) * 10) + (j + 3);
            j++;
            q[j][0] = j + 1;
            j++;
            q[j][2] = j + 3;
            j++;
            q[j][1] = j + 1;
            j++;
            q[j][2] = j + 1;
            j++;
            i = i + 2;
        }
        if (reg[i] == 'b' && reg[i + 1] == '|' && reg[i + 2] == 'a')
        {
            q[j][2] = ((j + 1) * 10) + (j + 3);
            j++;
            q[j][1] = j + 1;
            j++;
            q[j][2] = j + 3;
            j++;
            q[j][0] = j + 1;
            j++;
            q[j][2] = j + 1;
            j++;
            i = i + 2;
        }
        if (reg[i] == 'a' && reg[i + 1] == '*')
        {
            q[j][2] = ((j + 1) * 10) + (j + 3);
            j++;
            q[j][0] = j + 1;
            j++;
            q[j][2] = ((j + 1) * 10) + (j - 1);
            j++;
```

```
        }
        if (reg[i] == 'b' && reg[i + 1] == '*')
        {
            q[j][2] = ((j + 1) * 10) + (j + 3);
            j++;
            q[j][1] = j + 1;
            j++;
            q[j][2] = ((j + 1) * 10) + (j - 1);
            j++;
        }
        if (reg[i] == ')' && reg[i + 1] == '*')
        {
            q[0][2] = ((j + 1) * 10) + 1;
            q[j][2] = ((j + 1) * 10) + 1;
            j++;
        }
        i++;
    }
    printf("\n\tTransition Table \n");
    printf("_____________________________________\n");
    printf("Current State |\tInput |\tNext State");
    printf("\n_____________________________________\n");
    for (i = 0; i <= j; i++)
    {
        if (q[i][0] != 0)
            printf("\n  q%d\t      |   a   |  q%d", i, q[i][0]);
        if (q[i][1] != 0)
            printf("\n  q%d\t      |   b   |  q%d", i, q[i][1]);
        if (q[i][2] != 0)
        {
            if (q[i][2] < 10)
                printf("\n  q%d\t      |   e   |  q%d", i, q[i][2]);
            else
                printf("\n  q%d\t      |   e   |  q%d , q%d", i, q[i][2] /
10, q[i][2] % 10);
        }
    }
    printf("\n_____________________________________\n");
    return 0;
}
```

Program to convert NFA to DFA

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

#define STATES 99
#define SYMBOLS 20

int N_symbols;    /* number of input symbols */
int N_NFA_states; /* number of NFA states */
char *NFAtab[STATES][SYMBOLS];
char *NFA_finals; /* NFA final states */

int N_DFA_states; /* number of DFA states */
int DFAtab[STATES][SYMBOLS];
char DFA_finals[STATES + 1]; /* NFA final states */

char StateName[STATES][STATES + 1]; /* state name table */
char Eclosure[STATES][STATES + 1];  /* epsilon closure for each state */

void print_nfa_table(
    char *tab[][SYMBOLS], /* DFA table */
    int nstates,          /* number of states */
    int nsymbols,         /* number of input symbols */
    char *finals)         /* final states */
{
    int i, j;

    puts("\nNFA: STATE TRANSITION TABLE");

    /* input symbols: '0', '1', ... */
    printf("     | ");
    for (i = 0; i < nsymbols; i++)
        printf("  %-6c", '0' + i);
    printf("  e\n"); /* epsilon */

    printf("-----+--");
    for (i = 0; i < nsymbols + 1; i++)
        printf("-------");
    printf("\n");
```

```
    for (i = 0; i < nstates; i++)
    {
        printf("  %c  | ", '0' + i); /* state */
        for (j = 0; j < nsymbols + 1; j++)
            printf("  %-6s", tab[i][j]);
        printf("\n");
    }
    printf("Final states = %s\n", finals);
}

void print_dfa_table(
    int tab[][SYMBOLS], /* DFA table */
    int nstates,        /* number of states */
    int nsymbols,       /* number of input symbols */
    char *finals)       /* final states */
{
    int i, j;

    puts("\nDFA: STATE TRANSITION TABLE");

    /* input symbols: '0', '1', ... */
    printf("     | ");
    for (i = 0; i < nsymbols; i++)
        printf("  %c  ", '0' + i);

    printf("\n-----+--");
    for (i = 0; i < nsymbols; i++)
        printf("-----");
    printf("\n");

    for (i = 0; i < nstates; i++)
    {
        printf("  %c  | ", 'A' + i); /* state */
        for (j = 0; j < nsymbols; j++)
            printf("  %c  ", tab[i][j]);
        printf("\n");
    }
    printf("Final states = %s\n", finals);
}
```

```
void load_NFA_table()
{
    NFAtab[0][0] = "1";
    NFAtab[0][1] = "";
    NFAtab[0][2] = "";
    NFAtab[0][3] = "2";
    NFAtab[1][0] = "";
    NFAtab[1][1] = "3";
    NFAtab[1][2] = "";
    NFAtab[1][3] = "";
    NFAtab[2][0] = "";
    NFAtab[2][1] = "";
    NFAtab[2][2] = "2";
    NFAtab[2][3] = "3";
    NFAtab[3][0] = "";
    NFAtab[3][1] = "";
    NFAtab[3][2] = "";
    NFAtab[3][3] = "";

    N_symbols = 3;
    N_NFA_states = 4;
    NFA_finals = "3";
    N_DFA_states = 0;
}

int string_merge(char *s, char *t)
{
    int n = 0;
    char temp[STATES + 1], *r = temp, *p = s;

    while (*p && *t)
    {
        if (*p == *t)
        {
            *r++ = *p++;
            t++;
        }
        else if (*p < *t)
            *r++ = *p++;
        else
```

```
            *r++ = *t++;
        n++;
    }
    *r = '\0';

    if (*t)
    {
        strcat(r, t);
        n += strlen(t);
    }
    else if (*p)
        strcat(r, p);
    strcpy(s, temp);

    return n;
}

void get_next_state_NFA(char *nextstates, char *cur_states, char
*nfa[STATES][SYMBOLS], int symbol)
{
    int i;
    char temp[STATES + 1];

    temp[0] = '\0';
    for (i = 0; i < strlen(cur_states); i++)
        string_merge(temp, nfa[cur_states[i] - '0'][symbol]);
    strcpy(nextstates, temp);
}

int state_index(char *state, char stnt[][STATES + 1], int *pn)
{
    int i;
    if (!*state)
        return -1; /* no next state */
    for (i = 0; i < *pn; i++)
        if (!strcmp(state, stnt[i]))
            return i;
    strcpy(stnt[i], state); /* new state-name */
    return (*pn)++;
}
```

```c
void get_ep_states(int state, char *epstates,
                   char *nfa[][SYMBOLS], int n_sym)
{
    int i, n = 0;
    strcpy(epstates, nfa[state][n_sym]);
    do
    {
        for (i = 0; i < strlen(epstates); i++)
            n = string_merge(epstates, nfa[epstates[i] - '0'][n_sym]);
    } while (n);
}

void init_Eclosure(char eclosure[][STATES + 1], char *nfa[][SYMBOLS], int
n_nfa, int n_sym)
{
    int i;
    printf("\nEpsilon-accessible states:\n");
    for (i = 0; i < n_nfa; i++)
    {
        get_ep_states(i, eclosure[i], nfa, n_sym);
        printf("    state %d : [%s]\n", i, eclosure[i]);
    }
    printf("\n");
}

void e_closure(char *epstates, char *states, char eclosure[][STATES + 1])
{
    int i;
    strcpy(epstates, states);
    for (i = 0; i < strlen(states); i++)
        string_merge(epstates, eclosure[states[i] - '0']);
}

int nfa_to_dfa(char *nfa[][SYMBOLS], int n_nfa, int n_sym, int
dfa[][SYMBOLS])
{
    int i = 0; /* current index of DFA */
    int n = 1; /* number of DFA states */
    char nextstate[STATES + 1];
```

```
    char temp[STATES + 1]; /* epsilon closure */
    int j;
    init_Eclosure(Eclosure, nfa, n_nfa, n_sym);
    e_closure(temp, "0", Eclosure);
    strcpy(StateName[0], temp); /* initialize start state */

    printf("Epsilon-NFA to DFA conversion\n");
    for (i = 0; i < n; i++)
    { /* for each DFA state */
        for (j = 0; j < n_sym; j++)
        { /* for each input symbol */
            get_next_state_NFA(nextstate, StateName[i], nfa, j);
            e_closure(temp, nextstate, Eclosure);
            dfa[i][j] = state_index(temp, StateName, &n);
            printf("    state %d(%4s) : %d --> state %2d(%4s)\n",
                   i, StateName[i], j, dfa[i][j], temp);
            dfa[i][j] += 'A'; /* 0/1/2/... --> 'A/B/C/...' */
        }
    }
    return n; /* number of DFA states */
}

void get_DFA_finals(
    char *dfinals,           /* DFA final states */
    char *nfinals,           /* NFA final states */
    char stnt[][STATES + 1], /* state-name table */
    int n_dfa)               /* number of DFA states */
{
    int i, j, k = 0, n = strlen(nfinals);
    for (i = 0; i < n_dfa; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (strchr(stnt[i], nfinals[j]))
            {
                dfinals[k++] = i + 'A';
                break;
            }
        }
    }
```

```
    dfinals[k] = '\0';
}

void main()
{
    load_NFA_table();
    print_nfa_table(NFAtab, N_NFA_states, N_symbols, NFA_finals);
    N_DFA_states = nfa_to_dfa(NFAtab, N_NFA_states, N_symbols, DFAtab);
    get_DFA_finals(DFA_finals, NFA_finals, StateName, N_DFA_states);
    print_dfa_table(DFAtab, N_DFA_states, N_symbols, DFA_finals);
}
```

Output:

```
PS D:\C Programs (VS Code)\SS> gcc 1.c
PS D:\C Programs (VS Code)\SS> ./a
Enter regular expression: (a|b)*abbb

        Transition Table

_____________________________________
Current State | Input | Next State
_____________________________________

  q0          |   e   |  q7 , q1
  q1          |   e   |  q2 , q4
  q2          |   a   |  q3
  q3          |   e   |  q6
  q4          |   b   |  q5
  q5          |   e   |  q6
  q6          |   e   |  q7 , q1
  q7          |   a   |  q8
  q8          |   b   |  q9
  q9          |   b   |  q10
  q10         |   b   |  q11

_____________________________________
```

```
PS D:\C Programs (VS Code)\SS> gcc 2.c
PS D:\C Programs (VS Code)\SS> ./a

NFA: STATE TRANSITION TABLE
     |   0       1       2       e
-----+-----------------------------
  0  |   1                       2
  1  |           3
  2  |                   2       3
  3  |
Final states = 3

Epsilon-accessible states:
    state 0 : [23]
    state 1 : []
    state 2 : [3]
    state 3 : []

Epsilon-NFA to DFA conversion
    state 0( 023) : 0 --> state  1(   1)
    state 0( 023) : 1 --> state -1(    )
    state 0( 023) : 2 --> state  2(  23)
    state 1(   1) : 0 --> state -1(    )
    state 1(   1) : 1 --> state  3(   3)
    state 1(   1) : 2 --> state -1(    )
    state 2(  23) : 0 --> state -1(    )
    state 2(  23) : 1 --> state -1(    )
    state 2(  23) : 2 --> state  2(  23)
    state 3(   3) : 0 --> state -1(    )
    state 3(   3) : 1 --> state -1(    )
    state 3(   3) : 2 --> state -1(    )

DFA: STATE TRANSITION TABLE
     |   0    1    2
-----+----------------
  A  |   B    @    C
  B  |   @    D    @
  C  |   @    @    C
  D  |   @    @    @
Final states = ACD
PS D:\C Programs (VS Code)\SS>
```