College Of Engineering Trivandrum

Compiler Design Lab

CS431

Submitted By:
Neethu S
S6 CSE Roll No:42

TVE18CS043

Department of Computer Science

October 22, 2021

Contents

1 DFA Minimization

1.1 Aim

Write a program to find unique minimal DFA from a given DFA.

1.2 Theory

A deterministic finite automaton M 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 $\delta : Q \times \Sigma \rightarrow Q$
- an initial (or start) state q0 ϵ Q
- a set of states F distinguished as accepting (or final) states $F \subseteq Q$

1.3 Algorithm

```
Step 1. Start
Step 2. Divide Q (set of states) into two sets. One set will contain all final
    states and the other set will contain non-final states. This partition
    is called P0.
Step 3. Initialize k = 1
Step 4. Find Pk by partitioning the different sets of Pk-1. In each set of
    Pk-1, we will take all possible pair of states. If two states of a set are
    distinguishable, we will split the sets into different sets in Pk.
Step 5. Stop when Pk = Pk-1 (No change in partition)
Step 6. All states of one set are merged into one. No. of states in minimized
    DFA will be equal to no. of sets in Pk.
Step 7. Stop
```

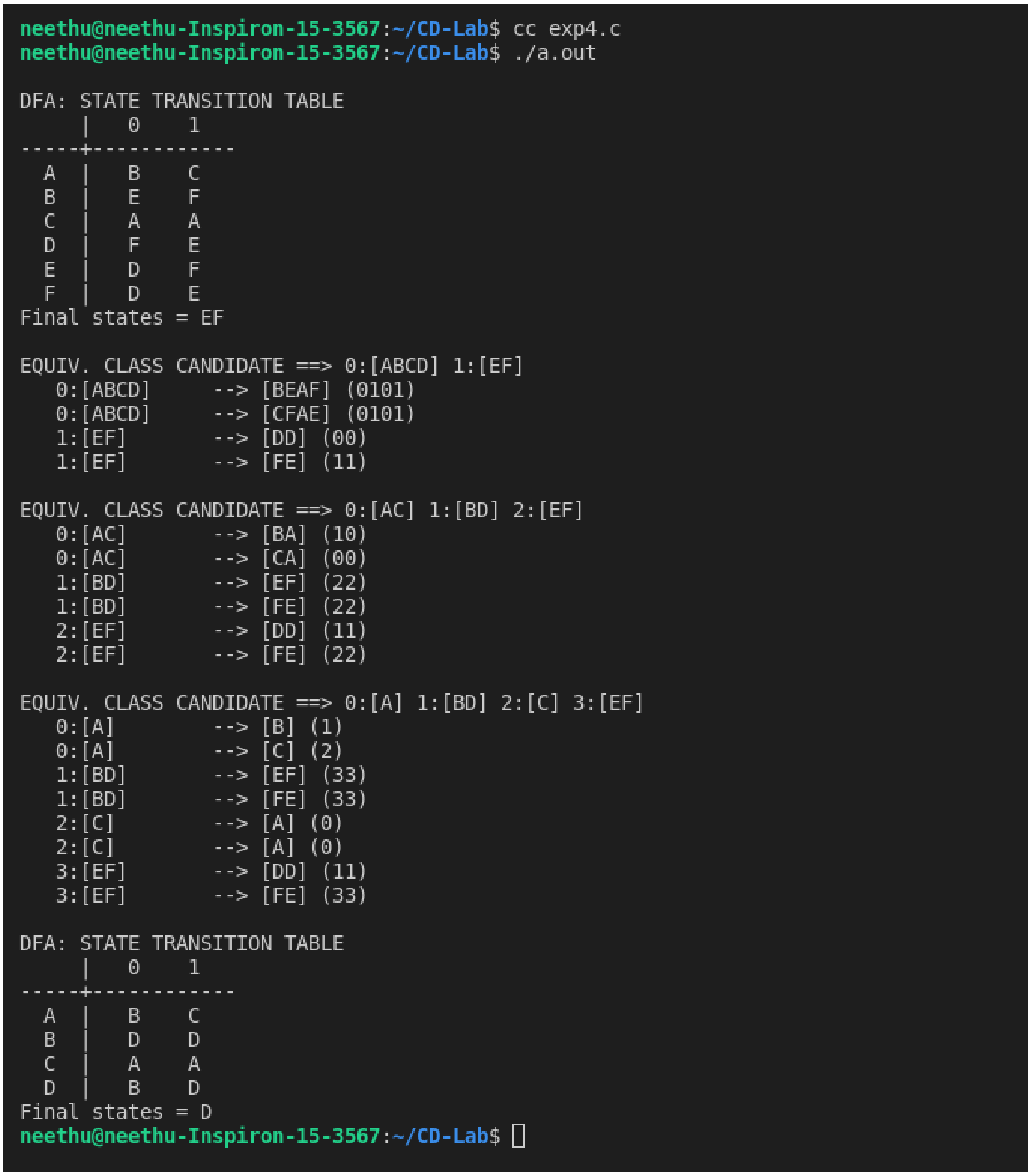
1.4 Code

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

#define STATES 99
#define SYMBOLS 20

int N_symbols;    /* number of input symbols */
int N_DFA_states; /* number of DFA states */
char *DFA_finals; /* final-state string */
int DFAtab[STATES][SYMBOLS];

char StateName[STATES][STATES + 1]; /* state-name table */
```

```
int N_optDFA_states; /* number of optimized DFA states */
int OptDFA[STATES][SYMBOLS];
char NEW_finals[STATES + 1];

/*
    Print state-transition table.
    State names: ’A’, ’B’, ’C’, ...
*/
void print_dfa_table(
    int tab[][SYMBOLS], /* DFA table */
    int nstates,        /* number of states */
    int nsymbols,       /* number of input symbols */
    char *finals)
{
    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]); /* next state */
        printf("\n");
    }
    printf("Final states = %s\n", finals);
}

/*
    Initialize NFA table.
*/
void load_DFA_table()
{

    DFAtab[0][0] = ’B’;
    DFAtab[0][1] = ’C’;
    DFAtab[1][0] = ’E’;
    DFAtab[1][1] = ’F’;
    DFAtab[2][0] = ’A’;
    DFAtab[2][1] = ’A’;
    DFAtab[3][0] = ’F’;
    DFAtab[3][1] = ’E’;
    DFAtab[4][0] = ’D’;
    DFAtab[4][1] = ’F’;
    DFAtab[5][0] = ’D’;
    DFAtab[5][1] = ’E’;

    DFA_finals = "EF";
    N_DFA_states = 6;
    N_symbols = 2;
}

/*
    Get next-state string for current-state string.
*/
void get_next_state(char *nextstates, char *cur_states,
                    int dfa[STATES][SYMBOLS], int symbol)
{
    int i, ch;

    for (i = 0; i < strlen(cur_states); i++)
        *nextstates++ = dfa[cur_states[i] - ’A’][symbol];
    *nextstates = ’\0’;
}

/*
    Get index of the equivalence states for state ’ch’.
    Equiv. class id’s are ’0’, ’1’, ’2’, ...
*/
char equiv_class_ndx(char ch, char stnt[][STATES + 1], int n)
{
    int i;

    for (i = 0; i < n; i++)
        if (strchr(stnt[i], ch))
            return i + ’0’;
    return -1; /* next state is NOT defined */
}

/*
    Check if all the next states belongs to same equivalence class.
    Return value:
        If next state is NOT unique, return 0.
        If next state is unique, return next state --> ’A/B/C/...’
    ’s’ is a ’0/1’ string: state-id’s
*/
char is_one_nextstate(char *s)
{
    char equiv_class; /* first equiv. class */

    while (*s == ’@’)
        s++;
    equiv_class = *s++; /* index of equiv. class */
    while (*s)
    {
        if (*s != ’@’ && *s != equiv_class)
            return 0;
        s++;
    }

    return equiv_class; /* next state: char type */
}

int state_index(char *state, char stnt[][STATES + 1], int n, int *pn,
                int cur) /* ’cur’ is added only for ’printf()’ */
{
    int i;
    char state_flags[STATES + 1]; /* next state info. */

    if (!*state)
        return -1; /* no next state */

    for (i = 0; i < strlen(state); i++)
        state_flags[i] = equiv_class_ndx(state[i], stnt, n);
    state_flags[i] = ’\0’;

    printf("   %d:[%s]\t--> [%s] (%s)\n",
           cur, stnt[cur], state, state_flags);

    if (i = is_one_nextstate(state_flags))
        return i - ’0’; /* deterministic next states */
    else
    {
        strcpy(stnt[*pn], state_flags); /* state-division info */
        return (*pn)++;
    }
}

int init_equiv_class(char statename[][STATES + 1], int n, char *finals)
{
    int i, j;

    if (strlen(finals) == n)
    { /* all states are final states */
        strcpy(statename[0], finals);
        return 1;
    }

    strcpy(statename[1], finals); /* final state group */

    for (i = j = 0; i < n; i++)
    {
        if (i == *finals - ’A’)
        {
            finals++;
        }
        else
            statename[0][j++] = i + 'A';
    }
    statename[0][j] = '\0';

    return 2;
}

int get_optimized_DFA(char stnt[][STATES + 1], int n,
                      int dfa[][SYMBOLS], int n_sym, int newdfa[][SYMBOLS])
{
    int n2 = n; /* 'n' + <num. of state-division info> */
    int i, j;
    char nextstate[STATES + 1];

    for (i = 0; i < n; i++)
    { /* for each pseudo-DFA state */
        for (j = 0; j < n_sym; j++)
        { /* for each input symbol */
            get_next_state(nextstate, stnt[i], dfa, j);
            newdfa[i][j] = state_index(nextstate, stnt, n, &n2, i) + 'A';
        }
    }

    return n2;
}

void chr_append(char *s, char ch)
{
    int n = strlen(s);

    *(s + n) = ch;
    *(s + n + 1) = '\0';
}

void sort(char stnt[][STATES + 1], int n)
{
    int i, j;
    char temp[STATES + 1];

    for (i = 0; i < n - 1; i++)
        for (j = i + 1; j < n; j++)
            if (stnt[i][0] > stnt[j][0])
            {
                strcpy(temp, stnt[i]);
                strcpy(stnt[i], stnt[j]);
                strcpy(stnt[j], temp);
            }
}
```

```
int split_equiv_class(char stnt[][STATES + 1],
                      int i1,    /* index of 'i1'-th equiv. class */
                      int i2,    /* index of equiv. vector for 'i1'-th class */
                      int n,     /* number of entries in 'stnt' */
                      int n_dfa) /* number of source DFA entries */
{
    char *old = stnt[i1], *vec = stnt[i2];
    int i, n2, flag = 0;
    char newstates[STATES][STATES + 1]; /* max. 'n' subclasses */

    for (i = 0; i < STATES; i++)
        newstates[i][0] = '\0';

    for (i = 0; vec[i]; i++)
        chr_append(newstates[vec[i] - '0'], old[i]);

    for (i = 0, n2 = n; i < n_dfa; i++)
    {
        if (newstates[i][0])
        {
            if (!flag)
            { /* stnt[i1] = s1 */
                strcpy(stnt[i1], newstates[i]);
                flag = 1; /* overwrite parent class */
            }
            else /* newstate is appended in 'stnt' */
                strcpy(stnt[n2++], newstates[i]);
        }
    }

    sort(stnt, n2); /* sort equiv. classes */

    return n2; /* number of NEW states(equiv. classes) */
}

/*
    Equiv. classes are segmented and get NEW equiv. classes.
*/
int set_new_equiv_class(char stnt[][STATES + 1], int n,
                        int newdfa[][SYMBOLS], int n_sym, int n_dfa)
{
    int i, j, k;

    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n_sym; j++)
        {
            k = newdfa[i][j] - 'A'; /* index of equiv. vector */
            if (k >= n)             /* equiv. class 'i' should be segmented */
                return split_equiv_class(stnt, i, k, n, n_dfa);
        }
    }
    return n;
}

void print_equiv_classes(char stnt[][STATES + 1], int n)
{
    int i;

    printf("\nEQUIV. CLASS CANDIDATE ==>");
    for (i = 0; i < n; i++)
        printf(" %d:[%s]", i, stnt[i]);
    printf("\n");
}

/*
    State-minimization of DFA: 'dfa' --> 'newdfa'
    Return value: number of DFA states.
*/
int optimize_DFA(
    int dfa[][SYMBOLS],      /* DFA state-transition table */
    int n_dfa,               /* number of DFA states */
    int n_sym,               /* number of input symbols */
    char *finals,            /* final states of DFA */
    char stnt[][STATES + 1], /* state name table */
    int newdfa[][SYMBOLS])   /* reduced DFA table */
{
    char nextstate[STATES + 1];
    int n;  /* number of new DFA states */
    int n2; /* 'n' + <num. of state-dividing info> */

    n = init_equiv_class(stnt, n_dfa, finals);

    while (1)
    {
        print_equiv_classes(stnt, n);
        n2 = get_optimized_DFA(stnt, n, dfa, n_sym, newdfa);
        if (n != n2)
            n = set_new_equiv_class(stnt, n, newdfa, n_sym, n_dfa);
        else
            break; /* equiv. class segmentation ended!!! */
    }

    return n; /* number of DFA states */
}

/*
    Check if 't' is a subset of 's'.
*/
int is_subset(char *s, char *t)
{
    int i;
    for (i = 0; *t; i++)
        if (!strchr(s, *t++))
            return 0;
    return 1;
}

/*
    New finals states of reduced DFA.
*/
void get_NEW_finals(
    char *newfinals,         /* new DFA finals */
    char *oldfinals,         /* source DFA finals */
    char stnt[][STATES + 1], /* state name table */
    int n)                   /* number of states in 'stnt' */
{
    int i;

    for (i = 0; i < n; i++)
        if (is_subset(oldfinals, stnt[i]))
            *newfinals++ = i + 'A';
    *newfinals++ = '\0';
}

void main()
{
    load_DFA_table();
    print_dfa_table(DFAtab, N_DFA_states, N_symbols, DFA_finals);

    N_optDFA_states = optimize_DFA(DFAtab, N_DFA_states,
                                   N_symbols, DFA_finals, StateName, OptDFA);
    get_NEW_finals(NEW_finals, DFA_finals, StateName, N_optDFA_states);

    print_dfa_table(OptDFA, N_optDFA_states, N_symbols, NEW_finals);
}
```

1.5 Output



```
neethu@neethu-Inspiron-15-3567:~/CD-Lab$ cc exp4.c
neethu@neethu-Inspiron-15-3567:~/CD-Lab$ ./a.out

DFA: STATE TRANSITION TABLE
     |   0    1
-----+------------
  A  |   B    C
  B  |   E    F
  C  |   A    A
  D  |   F    E
  E  |   D    F
  F  |   D    E
Final states = EF

EQUIV. CLASS CANDIDATE ==> 0:[ABCD] 1:[EF]
   0:[ABCD]     --> [BEAF] (0101)
   0:[ABCD]     --> [CFAE] (0101)
   1:[EF]       --> [DD] (00)
   1:[EF]       --> [FE] (11)

EQUIV. CLASS CANDIDATE ==> 0:[AC] 1:[BD] 2:[EF]
   0:[AC]       --> [BA] (10)
   0:[AC]       --> [CA] (00)
   1:[BD]       --> [EF] (22)
   1:[BD]       --> [FE] (22)
   2:[EF]       --> [DD] (11)
   2:[EF]       --> [FE] (22)

EQUIV. CLASS CANDIDATE ==> 0:[A] 1:[BD] 2:[C] 3:[EF]
   0:[A]        --> [B] (1)
   0:[A]        --> [C] (2)
   1:[BD]       --> [EF] (33)
   1:[BD]       --> [FE] (33)
   2:[C]        --> [A] (0)
   2:[C]        --> [A] (0)
   3:[EF]       --> [DD] (11)
   3:[EF]       --> [FE] (33)

DFA: STATE TRANSITION TABLE
     |   0    1
-----+------------
  A  |   B    C
  B  |   D    D
  C  |   A    A
  D  |   B    D
Final states = D
neethu@neethu-Inspiron-15-3567:~/CD-Lab$
```

1.6 Result

Implemented the program to find unique minimal DFA from a given DFA using C language in Ubuntu 20.04 and the above outputs were obtained.