// C Program to illustrate how to convert e-nfa to DFA

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_LEN 100

char NFA\_FILE[MAX\_LEN];

char buffer[MAX\_LEN];

int zz = 0;

// Structure to store DFA states and their

// status ( i.e new entry or already present)

struct DFA {

char \*states;

int count;

} dfa;

int last\_index = 0;

FILE \*fp;

int symbols;

/\* reset the hash map\*/

void reset(int ar[], int size) {

int i;

// reset all the values of

// the mapping array to zero

for (i = 0; i < size; i++) {

ar[i] = 0;

}

}

// Check which States are present in the e-closure

/\* map the states of NFA to a hash set\*/

void check(int ar[], char S[]) {

int i, j;

// To parse the individual states of NFA

int len = strlen(S);

for (i = 0; i < len; i++) {

// Set hash map for the position

// of the states which is found

j = ((int)(S[i]) - 65);

ar[j]++;

}

}

// To find new Closure States

void state(int ar[], int size, char S[]) {

int j, k = 0;

// Combine multiple states of NFA

// to create new states of DFA

for (j = 0; j < size; j++) {

if (ar[j] != 0)

S[k++] = (char)(65 + j);

}

// mark the end of the state

S[k] = '\0';

}

// To pick the next closure from closure set

int closure(int ar[], int size) {

int i;

// check new closure is present or not

for (i = 0; i < size; i++) {

if (ar[i] == 1)

return i;

}

return (100);

}

// Check new DFA states can be

// entered in DFA table or not

int indexing(struct DFA \*dfa) {

int i;

for (i = 0; i < last\_index; i++) {

if (dfa[i].count == 0)

return 1;

}

return -1;

}

/\* To Display epsilon closure\*/

void Display\_closure(int states, int closure\_ar[],

char \*closure\_table[],

char \*NFA\_TABLE[][symbols + 1],

char \*DFA\_TABLE[][symbols]) {

int i;

for (i = 0; i < states; i++) {

reset(closure\_ar, states);

closure\_ar[i] = 2;

// to neglect blank entry

if (strcmp(&NFA\_TABLE[i][symbols], "-") != 0) {

// copy the NFA transition state to buffer

strcpy(buffer, &NFA\_TABLE[i][symbols]);

check(closure\_ar, buffer);

int z = closure(closure\_ar, states);

// till closure get completely saturated

while (z != 100)

{

if (strcmp(&NFA\_TABLE[z][symbols], "-") != 0) {

strcpy(buffer, &NFA\_TABLE[z][symbols]);

// call the check function

check(closure\_ar, buffer);

}

closure\_ar[z]++;

z = closure(closure\_ar, states);

}

}

// print the e closure for every states of NFA

printf("\n e-Closure (%c) :\t", (char)(65 + i));

bzero((void \*)buffer, MAX\_LEN);

state(closure\_ar, states, buffer);

strcpy(&closure\_table[i], buffer);

printf("%s\n", &closure\_table[i]);

}

}

/\* To check New States in DFA \*/

int new\_states(struct DFA \*dfa, char S[]) {

int i;

// To check the current state is already

// being used as a DFA state or not in

// DFA transition table

for (i = 0; i < last\_index; i++) {

if (strcmp(&dfa[i].states, S) == 0)

return 0;

}

// push the new

strcpy(&dfa[last\_index++].states, S);

// set the count for new states entered

// to zero

dfa[last\_index - 1].count = 0;

return 1;

}

// Transition function from NFA to DFA

// (generally union of closure operation )

void trans(char S[], int M, char \*clsr\_t[], int st,

char \*NFT[][symbols + 1], char TB[]) {

int len = strlen(S);

int i, j, k, g;

int arr[st];

int sz;

reset(arr, st);

char temp[MAX\_LEN], temp2[MAX\_LEN];

char \*buff;

// Transition function from NFA to DFA

for (i = 0; i < len; i++) {

j = ((int)(S[i] - 65));

strcpy(temp, &NFT[j][M]);

if (strcmp(temp, "-") != 0) {

sz = strlen(temp);

g = 0;

while (g < sz) {

k = ((int)(temp[g] - 65));

strcpy(temp2, &clsr\_t[k]);

check(arr, temp2);

g++;

}

}

}

bzero((void \*)temp, MAX\_LEN);

state(arr, st, temp);

if (temp[0] != '\0') {

strcpy(TB, temp);

} else

strcpy(TB, "-");

}

/\* Display DFA transition state table\*/

void Display\_DFA(int last\_index, struct DFA \*dfa\_states,

char \*DFA\_TABLE[][symbols]) {

int i, j;

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n");

printf("\t\t DFA TRANSITION STATE TABLE \t\t \n\n");

printf("\n STATES OF DFA :\t\t");

for (i = 1; i < last\_index; i++)

printf("%s, ", &dfa\_states[i].states);

printf("\n");

printf("\n GIVEN SYMBOLS FOR DFA: \t");

for (i = 0; i < symbols; i++)

printf("%d, ", i);

printf("\n\n");

printf("STATES\t");

for (i = 0; i < symbols; i++)

printf("|%d\t", i);

printf("\n");

// display the DFA transition state table

printf("--------+-----------------------\n");

for (i = 0; i < zz; i++) {

printf("%s\t", &dfa\_states[i + 1].states);

for (j = 0; j < symbols; j++) {

printf("|%s \t", &DFA\_TABLE[i][j]);

}

printf("\n");

}

}

// Driver Code

int main() {

int i, j, states;

char T\_buf[MAX\_LEN];

// creating an array dfa structures

struct DFA \*dfa\_states = malloc(MAX\_LEN \* (sizeof(dfa)));

states = 6, symbols = 2;

printf("\n STATES OF NFA :\t\t");

for (i = 0; i < states; i++)

printf("%c, ", (char)(65 + i));

printf("\n");

printf("\n GIVEN SYMBOLS FOR NFA: \t");

for (i = 0; i < symbols; i++)

printf("%d, ", i);

printf("eps");

printf("\n\n");

char \*NFA\_TABLE[states][symbols + 1];

// Hard coded input for NFA table

char \*DFA\_TABLE[MAX\_LEN][symbols];

strcpy(&NFA\_TABLE[0][0], "FC");

strcpy(&NFA\_TABLE[0][1], "-");

strcpy(&NFA\_TABLE[0][2], "BF");

strcpy(&NFA\_TABLE[1][0], "-");

strcpy(&NFA\_TABLE[1][1], "C");

strcpy(&NFA\_TABLE[1][2], "-");

strcpy(&NFA\_TABLE[2][0], "-");

strcpy(&NFA\_TABLE[2][1], "-");

strcpy(&NFA\_TABLE[2][2], "D");

strcpy(&NFA\_TABLE[3][0], "E");

strcpy(&NFA\_TABLE[3][1], "A");

strcpy(&NFA\_TABLE[3][2], "-");

strcpy(&NFA\_TABLE[4][0], "A");

strcpy(&NFA\_TABLE[4][1], "-");

strcpy(&NFA\_TABLE[4][2], "BF");

strcpy(&NFA\_TABLE[5][0], "-");

strcpy(&NFA\_TABLE[5][1], "-");

strcpy(&NFA\_TABLE[5][2], "-");

printf("\n NFA STATE TRANSITION TABLE \n\n\n");

printf("STATES\t");

for (i = 0; i < symbols; i++)

printf("|%d\t", i);

printf("eps\n");

// Displaying the matrix of NFA transition table

printf("--------+------------------------------------\n");

for (i = 0; i < states; i++) {

printf("%c\t", (char)(65 + i));

for (j = 0; j <= symbols; j++) {

printf("|%s \t", &NFA\_TABLE[i][j]);

}

printf("\n");

}

int closure\_ar[states];

char \*closure\_table[states];

Display\_closure(states, closure\_ar, closure\_table, NFA\_TABLE, DFA\_TABLE);

strcpy(&dfa\_states[last\_index++].states, "-");

dfa\_states[last\_index - 1].count = 1;

bzero((void \*)buffer, MAX\_LEN);

strcpy(buffer, &closure\_table[0]);

strcpy(&dfa\_states[last\_index++].states, buffer);

int Sm = 1, ind = 1;

int start\_index = 1;

// Filling up the DFA table with transition values

// Till new states can be entered in DFA table

while (ind != -1) {

dfa\_states[start\_index].count = 1;

Sm = 0;

for (i = 0; i < symbols; i++) {

trans(buffer, i, closure\_table, states, NFA\_TABLE, T\_buf);

// storing the new DFA state in buffer

strcpy(&DFA\_TABLE[zz][i], T\_buf);

// parameter to control new states

Sm = Sm + new\_states(dfa\_states, T\_buf);

}

ind = indexing(dfa\_states);

if (ind != -1)

strcpy(buffer, &dfa\_states[++start\_index].states);

zz++;

}

// display the DFA TABLE

Display\_DFA(last\_index, dfa\_states, DFA\_TABLE);

return 0;

}



