1/2/23

EXPERIMENT 3 : CONVERSION OF NFA TO DFA

AIM: To write a program to convert NFA to DFA

ALGORITHM:

```
1. Start
2. Get input from the user.
3. Set the only state in SDFA to 'unmarked'
4. While SDFA contains an unmarked state do
   (a) Let T be that unmarked state
   (b) For each a in Σ do s = ε- closure (move NFA (T, a))
   (c) If s is not in SDFA already then add s to SDFA
       (as an 'unmarked' state)
5. For each S in SDFA if any s is final state in
   NFA, then mark S as final state in DFA.
6. Print result
7. Stop the program.
```

PROGRAM CODE :

```
#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;

struct DFA {
    char *states;
    int count;
} dfa;
```

```
int last_index = 0;
FILE *fp;
int symbols;

void reset (int ar[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    ar[i] = 0;
  }
}

void check (int ar[], char s[]) {
  int i, j;
  int len = strlen (s);
  for (i = 0; i < len ; i++) {
    j = ( (int) (s[i]) - 65 );
    ar[j]++;
  }
}

void state ( int ar[] , int size, char S[]) {
  int j, k = 0;
  for ( j = 0 ; j < size; j++) {
    if (ar[j] != 0)
       S[k++] = (char) (65 + j);
  }
  S[k] = '\0';
}

int indexing (struct DFA *dfa) {
  int i;
  for (i = 0 ; i < last_index ; i++) {
    if (dfa[i]. count == 0)
      return 1;
  }
```

```
    return -1;
}

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;

    if (strcmp (&NFA_TABLE[i][symbols], "-") !=0) {
      strcpy (buffer, &NFA_TABLE[i][symbols]);
      check (closure_ar, buffer);

    while (z! =100)
    {  if (strcmp (&NFA_TABLE[z][symbols], "-") ! =0) {
       strcpy (buffer, &NFA_TABLE[z][symbols]);
       check (closure_ar, buffer);
       }
       closure_ar[z]++;
       z = closure (closure_ar, states);
     }
   }
   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]);
   }
}
```

```
int new_states (struct DFA *dfa, char S[]) {
    int i;
    for (i=0 ; i< last_index ; i++ ) {
        if (strcmp (&dfa[i]. states , s) ==0)
            return 0;
    }
    Strcpy (&dfa [last_index ++]. states , s);
    dfa[last_index -1]. count =0;
    Return 1;
}

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 ;
    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, "-");
}

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");
  print ("states \t");
}
```

```
int main () {
  int  i, j , states ;
  char T_but [MAX_LEN];
 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++)
  for (i=0 ; i< symbols ; i++)
    printf ("%d" , i );
  printf ("eps") ;
  printf ( " \n\n") ;
  char *DFA_TABLE [MAX_LEN][symbols];
  strcpy (&NFA_TABLE [0][0], "FC");
     "          "       [0][1], "-");
     "          "       [0][2], "BF");
     "          "       [1][0] , "-");
     "          "       [1][1] , "C");
     "          "       [1][2] , "-");
     "          "       [2][0] , "-");
     "          "       [2][1] , "-");
     "          "       [2][2] , "D");
     "          "       [3][0] , "E");
     "          "       [3][1] , "A");
     "          "       [3][2] , "-");
     "          "       [4][0] , "A");
     "          "       [4][1] , "-");
     "          "       [4][2] , "BF");
     "          "       [5][2] , "-");
```

```
for (i=0 ; i < symbols; i++)
   printf("|%d \t", i);
  printf("eps \n");

int closure_ar [states];
char *closure_table [states];

Display_closure (states, closure_ar, closure_table,
  NFA_TABLE , DFA_STABLE);
strcpy (&dfa_states [last_index ++]. states, "-");

dfa_states [last_index - 1]. count = 1;

bzero ((void *) buffer, MAX_LEN);

int sm = 1 , ind = 1
int start_index = 1;

Display_DFA ( last_index , dfa_states,
                    DFA_TABLE );

return 0;
}
```

01/02/2028

RESULT: The NFA is converted to DFA.

```
OUTPUT :

States of NFA : A, B, C, D, E, F
Given symbols for NFA : 0, 1, eps
NFA State transition table :

States     0        1        eps        eps
A          FC                 -          BF
B          -        C         -
C          -        -         D
D          E        A         -
E          A        -         BF
F          -        -         -

e - closure (A) : ABF
e - closure (B) : B
e - closure (C) : CD
e - closure (D) : D
e - closure (E) : BEF
e - closure (F) : F
```