

**Name:** Naseem Shah

**Roll No:** 058

**Exp. No:** 05

**Date:** 07/09/2021

## DFA Implementation of $(1^*(0 + 01 + 001))^*$

**Aim:** Write a C program to implement a DFA accepting binary strings such that every '00' is immediately followed by 1.

```
/*
C program to implement a DFA accepting binary strings such that every
'00' is immediately followed by 1.
*/

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

const int transition_table[][2] = {
    {1, 0},    // state 0, initial state & final state
    {2, 0},    // state 1
    {3, 0},    // state 2
    {3, 3}     // dead state
};

const int final_state[] = {0};
const int num_final_states =
    sizeof(final_state)/sizeof(final_state[0]);

void main() {
    char s[1000];
    bool valid = false;
    int state = 0;
    printf("input string : ");
    scanf("%s", s);

    for(int i=0; s[i] != '\0' ; i++){
        if(s[i] != '0' && s[i] != '1')
            state = 3;

        if(state == 3)
            break;

        state = transition_table[state][s[i] - '0'];
    }
}
```

```

    }

    for(int i=0; i<num_final_states; i++)
        if(state == final_state[i]){
            valid = true;
            break;
        }

    if(valid)
        printf("Valid string!\n");
    else
        printf("Invalid string!\n");

    return;
}

```

**Result:** Successfully written C program to implement a DFA accepting binary strings such that every '00' is immediately followed by 1.

**Remarks:**(To be filled by faculty)

### Algorithm

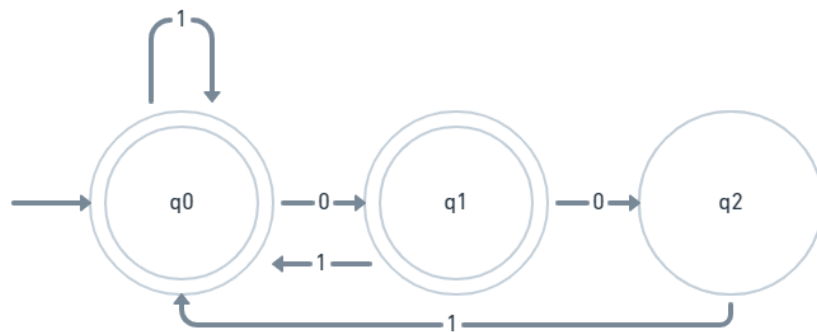
1. Start
2. Create a NFA, and then DFA for the given regular expression,  $(1^*(0 + 01 + 001))^*$ .
3. Create transition table, transition\_table[i][j], for the DFA obtained where each transition\_table[i][j] denotes the current state i, and the next state when input is j.  
transition\_table[4][2]= { {1,0}, {2,0}, {3, 0}, {3,3} }
4. Set final\_states = {0, 1}
5. Read the input string, s
6. Set state = 0, valid = false
7. for each character ch in s, do
  - a. if ch != '0' and ch != '1', then state = 3
  - b. if state = 3, then break
  - c. state = transition\_table[state][ch - '0']
8. for i in final\_states, do
  - a. if i == state, then
    - i. valid = true
    - ii. break
9. if valid == true, then print "Valid string", else print "Invalid string"
10. Stop

### Diagrams & Tables

Table for DFA

State	0	1
*q0	q1	q0
*q1	q2	q0
q2	$\phi$	q0

DFA



Sample output

```
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/EXP4# cd ../exp5
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# ls
exp5.c
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# gcc exp5.c
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# ./a.out
input string : 01
Valid string!
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# 10
10: command not found
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# ./a.out
input string : 10
Invalid string!
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# ./a.out
input string : 111
Valid string!
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# ./a.out
input string : 10101001
Valid string!
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# ./a.out
input string : 1001110011100
Invalid string!
root@Naseem-Laptop:/mnt/d/Coding/LanguageLab/exp5# |
```