# BITS Pilani, Hyderabad Campus
# Department of Computer Sc. and Information Systems
# Second Semester, 2024-25
# CS F363 Compiler Construction
# Lab-3: Simulation of DFA and NFA

## 1 Objectives

The objectives of this lab are the following.

1. Understand and implement Deterministic Finite Automata (DFA) using C language to check if the given DFA accepts the given string.
2. Understand and simulate non-deterministic finite automata (NFA), which allows multiple transitions and epsilon transitions, using C programming.
3. Convert an NFA to an equivalent DFA using the generating transition table method.

## 2 Simulation of DFA and NFA

In the following, the C code implementation of a deterministic finite automata (DFA) is given. The code reads the description of a DFA from a text file, `input.txt`. The description of the input format is given below. The code checks if the DFA accepts the given input string (read from the user at the run-time, from terminal).

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STATES 10
#define MAX_SYMBOLS 2

// Structure for DFA
typedef struct {
    int num_states;
    int num_symbols;
    int transition[MAX_STATES][MAX_SYMBOLS];
    bool final_states[MAX_STATES];
} DFA;

// Initialize DFA
void initialize_dfa(DFA *dfa, int num_states, int num_symbols) {
    dfa->num_states = num_states;
    dfa->num_symbols = num_symbols;
    for (int i = 0; i < MAX_STATES; i++) {
        dfa->final_states[i] = false;
        for (int j = 0; j < MAX_SYMBOLS; j++) {
            dfa->transition[i][j] = -1;
        }
    }
}
```

```c
// Read DFA from file
void read_dfa_from_file(DFA *dfa, const char *filename) {


    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("Failed to open file");
        exit(EXIT_FAILURE);
    }

    int num_states;
    fscanf(file, "%d", &num_states);
    initialize_dfa(dfa, num_states, MAX_SYMBOLS);

    char line[256];
    fgets(line, sizeof(line), file); // Skip the line with number of states
    fgets(line, sizeof(line), file); // Read the line containing final states


    //Read final states
    char *token = strtok(line, "{ } , ");
    while (token != NULL) {
        int final_state = atoi(token);
        dfa->final_states[final_state] = true;
        token = strtok(NULL, " ");
    }

   // Read transitions
    for (int i = 0; i < num_states; i++) {
        int state;
        fscanf(file, "%d", &state);
        for (int symbol = 0; symbol < MAX_SYMBOLS; symbol++) {
            int next_state;
            fscanf(file, "%d", &next_state);
            dfa->transition[state][symbol] = next_state;
        }
    }

    fclose(file);
}

// Simulate DFA
bool simulate_dfa(DFA *dfa, const char *input) {
    int current_state = 0; // Assume starting state is 0
    printf("%d", current_state);
    for (int i = 0; input[i] != '\0'; i++) {
        int symbol = input[i] - '0'; // Convert character to integer (assumes input
            is binary)
        if (symbol < 0 || symbol >= dfa->num_symbols) {
            printf("Invalid input symbol detected!\n");
            return false;
        }
        current_state = dfa->transition[current_state][symbol];
        printf(" -> %d", current_state);
    }
    printf("\n");
    return dfa->final_states[current_state]; // Accept if the final state is a final
        state
}
```

```c
int main() {
    DFA dfa;
    read_dfa_from_file(&dfa, "input.txt");
    char input[100];
    printf("Enter input string: ");
    scanf("%s", input);
    if (simulate_dfa(&dfa, input)) {
        printf("Input string is accepted by the DFA.\n");
    } else {
        printf("Input string is rejected by the DFA.\n");
    }
    return 0;
}
```

**Input format:** A `input.txt` file with several lines as follows.

```
5 //number of states and the default state numbers start with 0,i.e., 0, 1, 2, 3, 4
    are the state labels, and 0 is the start state
{2, 4} //The list of final states
0 1 3 // state no. = 0 and delta(0,0)=1 and delta(0,1) =3
1 3 4 // state is 1 and delta(1,0)=3 and delta(1,1) =4
..
```

You can test the above C code with the following input file (`input.txt`)

```
4
{2, 1}
0 0 1
1 2 1
2 3 0
3 0 2
```

Sample test cases are given below:

```
$ ./a.out
Enter input string: 01001
path: 0 -> 0 -> 1 -> 2 -> 3 -> 2
Input string is accepted by the DFA.
$ ./a.out
Enter input string: 00110
path: 0 -> 0 -> 0 -> 1 -> 1 -> 2
Input string is accepted by the DFA.
$ ./a.out
Enter input string: 00011
path: 0 -> 0 -> 0 -> 0 -> 1 -> 1
Input string is accepted by the DFA.
$ ./a.out
Enter input string: 00000
path: 0 -> 0 -> 0 -> 0 -> 0 -> 0
Input string is rejected by the DFA.
```

**Problem 1** Write a C program to check that the given non-deterministic finite automata (NFA) accept the input string $w \in \{0,1\}^*$.

**Input format:** The NFA description will be given in a text file, `input.txt`. The content of the `input.txt` is similar to the case of DFA except that the transition function of the state and the alphabet symbol is a list of states (can also be empty) instead of a single state. A sample test case (`input.txt`) is given below:

```
4
{1, 3}
0 {0, 1} {0}
1 {2} {2}
2 {3} {}
3 {3} {3}
```

You can test your program with the above input file with different strings given below:

```
01001 //Accepted
1001101 //Accepted
10101 //Accepted
11011 // Not Accepted
```

**Problem 2 (Homework)** Extend your C program of Problem 1 to NFA with $\epsilon$-transitions.

**Input format:** The input format is the same as the input format of NFA (given in problem 1), expect each state to have transitions for each symbol in the alphabet $\Sigma$ and $\epsilon$ (the last list of states in a row corresponding to a state defines the transition of the state and $\epsilon$. A sample input is given below.

```
5
{2, 4}
0   {1,4}        {3}          {2}
1   {}          {4, 2}        {4}
2   {1, 3}      {3, 4}         {}
3   {1}           {}          {2}
4   {2, 3, 4}    {4}          {0,1}
```

# 3 Conversion of NFA to DFA (optional)

This section introduces a problem in which you must write a C program that converts a given Nondeterministic Finite Automaton (NFA) into an equivalent Deterministic Finite Automaton (DFA). The task involves simulating the NFA and generating a DFA that has an equivalent behavior using generating the transition table method. The input format is similar to that in previous problems, and the output will display the transition function followed by the list of final states. In the next step, the problem further extends the concept to handle NFAs with $\epsilon$-transitions, requiring you to modify your program to account for these additional transitions while still converting the NFA to a DFA.

**Problem 3** Write a C program to convert the given NFA to an equivalent DFA. The input format is the same as Problem 1, and the output format is also similar, but first the transition function is printed, and then the list of final states is printed. You need to generate only states that are reachable from the start state.

The `input.txt` 's content is similar to the NFA case. A sample `input.txt` is given below.

```
4
{1, 3}
0 {0, 1} {0}
1 {2} {2}
2 {3} {}
3 {3} {3}
```

The output of the above input must be:

```
[0]             [0, 1]          [0]
[0, 1]          [0, 1, 2]       [0, 2]
[0, 1, 2]       [0, 1, 2, 3]    [0, 2]
[0, 2]          [0, 1, 3]        [0]
[0, 1, 2, 3]    [0, 1, 2, 3]    [0, 2, 3]
[0, 1, 3]       [0, 1, 2, 3]    [0, 2, 3]
[0, 2, 3]       [0, 1, 3]       [0, 3]
[0, 3]           [0, 1, 3]     [0, 3]
[0, 1, 2, 3] [0, 1, 3] [0, 2, 3] [0, 3] //final states
```

**Problem 4** Extend your C program written for Problem 3 to NFA with $\epsilon$-transitions.

**Input format:** The input format is the same as the input format of NFA with $\epsilon$-transitions given in problem 2, expect each state to have transitions for each symbol in the alphabet $\Sigma$ and $\epsilon$ (the last list of states in a row corresponding to a state defines the transition of the state and $\epsilon$). A sample input is given below.

```
5
{2, 4}
0   {1,4}       {3}         {2}
1   {}          {4, 2}      {4}
2   {1, 3}      {3, 4}       {}
3   {1}           {}        {2}
4   {2, 3, 4}   {4}         {0,1}
```