CSE432: Automata and Computability

Assignment Report

Anas Salah Abdelrazek

ID : 19P9033

Contents

Assignment Requirements

Assignment

Each student should write a java – C++ or C# program to simulate

1- Deterministic finite automata that accept even number of 0's and even number of 1's.
2- Deterministic finite automata that accept the set of all strings with three consecutive 0's followed by any number of 1's using
3- push down automata that accept palindrome

Note:

- The source code files should be uploaded
- A report that documents the code should be included
- Screen shots from the output should also be included in the report

Github Link

anassalah24/Automata-Assignment (github.com)

Question 1

Code

```cpp
#include <iostream>
#include <string>

using namespace std;

// DFA states
enum class State {
    EVEN_ZEROS_EVEN_ONES, // Even number of 0s and even number of 1s seen so far
    ODD_ZEROS_EVEN_ONES,  // Odd number of 0s and even number of 1s seen so far
    EVEN_ZEROS_ODD_ONES,  // Even number of 0s and odd number of 1s seen so far
    ODD_ZEROS_ODD_ONES    // Odd number of 0s and odd number of 1s seen so far
};

```

```
14    // DFA transition function
15    State transition(State current_state, char input) {
16        switch (current_state) {
17            case State::EVEN_ZEROS_EVEN_ONES:
18                if (input == '0') {
19                    return State::ODD_ZEROS_EVEN_ONES;
20                } else {
21                    return State::EVEN_ZEROS_ODD_ONES;
22                }
23            case State::ODD_ZEROS_EVEN_ONES:
24                if (input == '0') {
25                    return State::EVEN_ZEROS_EVEN_ONES;
26                } else {
27                    return State::ODD_ZEROS_ODD_ONES;
28                }
29            case State::EVEN_ZEROS_ODD_ONES:
30                if (input == '0') {
31                    return State::ODD_ZEROS_ODD_ONES;
32                } else {
33                    return State::EVEN_ZEROS_EVEN_ONES;
34                }
35            case State::ODD_ZEROS_ODD_ONES:
36                if (input == '0') {
37                    return State::EVEN_ZEROS_ODD_ONES;
38                } else {
39                    return State::ODD_ZEROS_EVEN_ONES;
40                }
41        }
42    }
```

```
43
44  // DFA accepts if it ends in EVEN_ZEROS_EVEN_ONES and number of 0s and 1s seen so far is even
45  bool accepts(string input) {
46      State current_state = State::EVEN_ZEROS_EVEN_ONES; // Start in the initial state
47      int num_zeros = 0, num_ones = 0; // Keep track of number of 0s and 1s seen so far
48      for (char c : input) {
49          current_state = transition(current_state, c); // Update state based on input
50          if (c == '0') {
51              num_zeros++; // Increment num_zeros if input is '0'
52          } else {
53              num_ones++; // Increment num_ones if input is '1'
54          }
55      }
56      return current_state == State::EVEN_ZEROS_EVEN_ONES && num_zeros % 2 == 0 && num_ones % 2 == 0;
57      // Return true only if the DFA ends in the accepting state AND number of 0s and 1s seen so far is even
58  }
59
60  int main() {
61      string input;
62      cout << "Enter a string of 0s and 1s: ";
63      cin >> input;
64      if (accepts(input)) {
65          cout << "Accepted!" << endl;
66      } else {
67          cout << "Rejected!" << endl;
68      }
69      return 0;
70  }
71
```

Description

1. The program begins by defining four possible states for the DFA, which are represented by an enumeration class `State`. The four possible states are:

- `EVEN_ZEROS_EVEN_ONES`: The DFA has seen an even number of 0s and an even number of 1s so far.

- `ODD_ZEROS_EVEN_ONES`: The DFA has seen an odd number of 0s and an even number of 1s so far.

- `EVEN_ZEROS_ODD_ONES`: The DFA has seen an even number of 0s and an odd number of 1s so far.

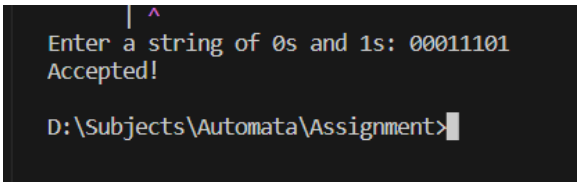
- `ODD_ZEROS_ODD_ONES`: The DFA has seen an odd number of 0s and an odd number of 1s so far.

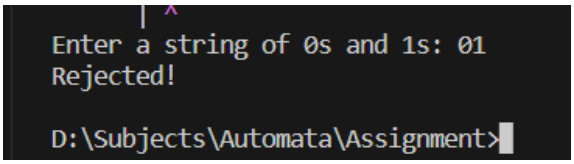
2. The program defines a transition function `transition()` that takes the current state of the DFA and the next input character as parameters, and returns the next state of the DFA based on the current state and the input character. The transition function uses a `switch` statement to implement the transitions for each possible state.

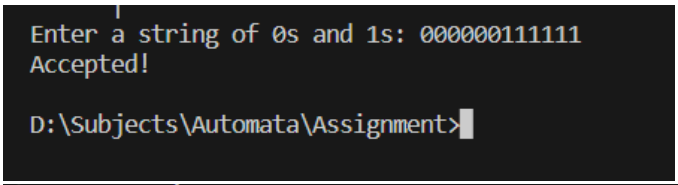
3. The program defines an `accepts()` function that takes a string as input and determines whether the DFA accepts the string. The `accepts()` function initializes the current state of the DFA to `EVEN_ZEROS_EVEN_ONES`, and keeps track of the number of 0s and 1s seen so far. It then iterates through each character of the input string, updating the current state of the DFA and the number of 0s and 1s seen so far based on the transition function. Finally, the `accepts()` function returns `true` if the DFA ends in the `EVEN_ZEROS_EVEN_ONES` state and the number of 0s and 1s seen so far is even, indicating that the input string has an even number of 0s and an even number of 1s.

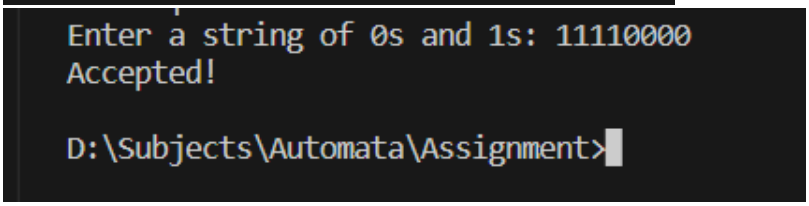
4. The program's `main()` function prompts the user to enter a string of 0s and 1s, reads the input string from standard input using `cin`, and passes the input string to the `accepts()` function. If the `accepts()` function returns `true`, indicating that the DFA accepts the input string, the program outputs "Accepted!". Otherwise, it outputs "Rejected!".

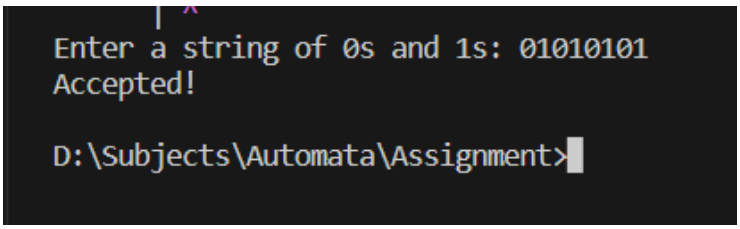
Output Screenshots

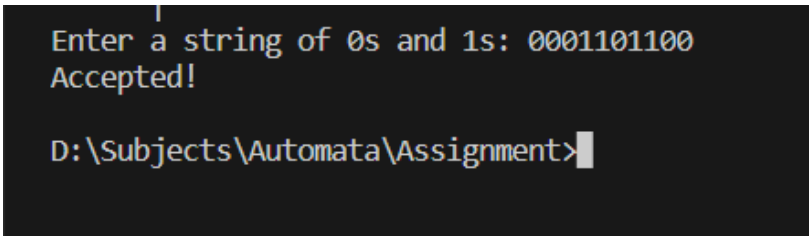


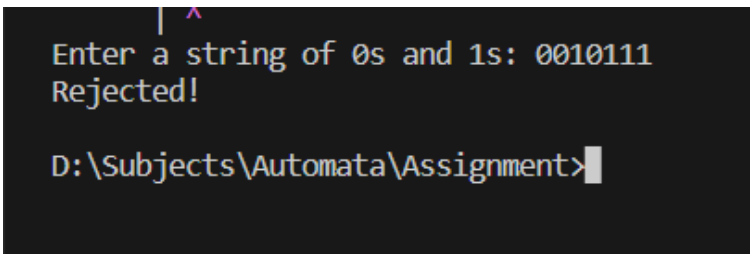












Question 2

Code

C++ Question2.cpp > ...

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
    // Define the DFA's states
    enum State { START, FIRST_ZERO, SECOND_ZERO, THIRD_ZERO, ACCEPT };
    State currentState = START;

    // Read in the input string
    string input;
    cout << "Enter a string of 0s and 1s: ";
    cin >> input;
```

```cpp
    // Process each character in the input string
    for (char c : input) {
        switch (currentState) {
            case START:
                // If the current state is START and the current character is 0,
                // transition to the FIRST_ZERO state. Otherwise, stay in the START state.
                if (c == '0') {
                    currentState = FIRST_ZERO;
                }
                break;
            case FIRST_ZERO:
                // If the current state is FIRST_ZERO and the current character is 0,
                // transition to the SECOND_ZERO state. Otherwise, go back to the START state.
                if (c == '0') {
                    currentState = SECOND_ZERO;
                } else {
                    currentState = START;
                }
                break;
            case SECOND_ZERO:
                // If the current state is SECOND_ZERO and the current character is 0,
                // transition to the THIRD_ZERO state. Otherwise, go back to the START state.
                if (c == '0') {
                    currentState = THIRD_ZERO;
                } else {
                    currentState = START;
                }
                break;
            case THIRD_ZERO:
                // If the current state is THIRD_ZERO and the current character is 0,
                // stay in the THIRD_ZERO state. If the current character is 1,
                // transition to the ACCEPT state. Otherwise, go back to the START state.
                if (c == '0') {
                    currentState = THIRD_ZERO;
                } else if (c == '1') {
                    currentState = ACCEPT;
                } else {
                    currentState = START;
```

```
            case THIRD_ZERO:
                // If the current state is THIRD_ZERO and the current character is 0,
                // stay in the THIRD_ZERO state. If the current character is 1,
                // transition to the ACCEPT state. Otherwise, go back to the START state.
                if (c == '0') {
                    currentState = THIRD_ZERO;
                } else if (c == '1') {
                    currentState = ACCEPT;
                } else {
                    currentState = START;
                }
                break;
            case ACCEPT:
                // If the current state is ACCEPT and the current character is not 1,
                // go back to the START state. Otherwise, stay in the ACCEPT state.
                if (c != '1') {
                    currentState = START;
                }
                break;
        }
    }

    // Output whether the input string was accepted or rejected
    if (currentState == ACCEPT) {
        cout << "Input string accepted" << endl;
    } else {
        cout << "Input string rejected" << endl;
    }

    return 0;
}

```

Description

This C++ program uses a Deterministic Finite Automaton (DFA) to accept strings that have three consecutive 0s followed by any number of 1s. Here's a description of how the program works:

1. The DFA has five states: START (the initial state), FIRST_ZERO (the state after reading the first 0), SECOND_ZERO (the state after reading the first two 0s), THIRD_ZERO (the state after reading three consecutive 0s), and ACCEPT (the accepting state after reading three consecutive 0s followed by any number of 1s).

2. The program reads in a string of 0s and 1s from the user.

3. The program processes each character in the input string using a for loop and a switch statement.

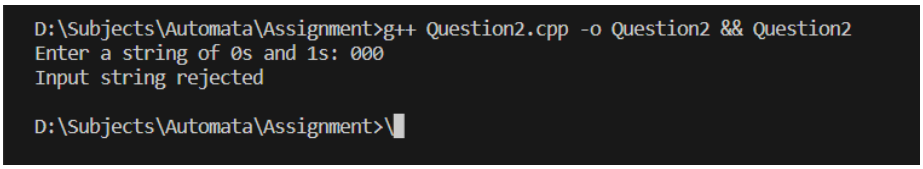
4. For each character in the input string, the switch statement checks the current state of the DFA and the current character to determine the next state of the DFA.

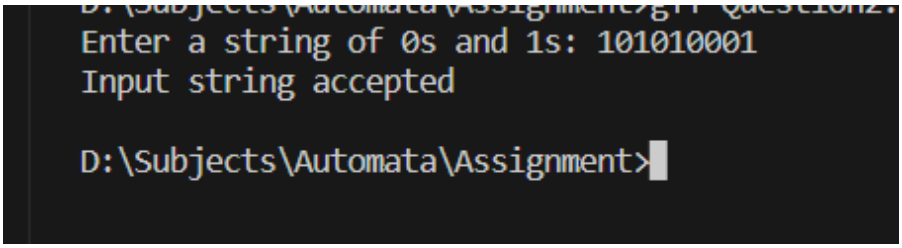
5. If the next state is the ACCEPT state, it means the DFA has accepted the input string. Otherwise, the DFA has rejected the input string.

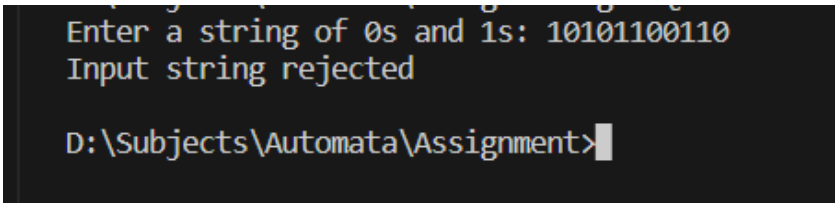
6. The program outputs either "Input string accepted" or "Input string rejected", depending on whether the DFA has accepted or rejected the input string.

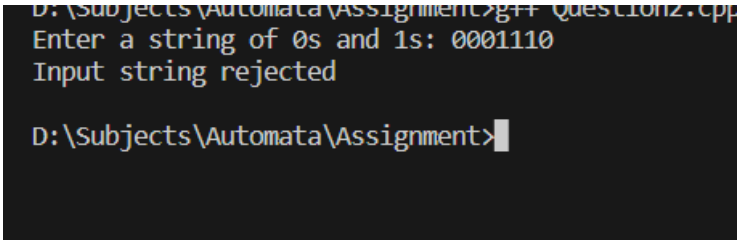
7. The program terminates.

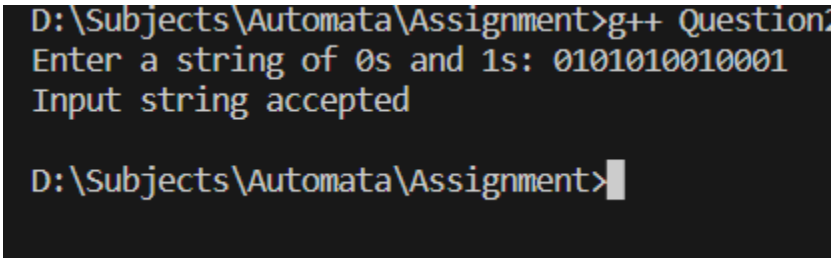
Output Screenshots











Question 3

Code

Question3.cpp > ...

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main() {
    // Ask the user to enter a string
    string str;
    cout << "Enter a string: ";
    cin >> str;

    // Create an empty stack to hold the first half of the string
    stack<char> stk;

    // Get the length of the string
    int n = str.length();

    // Push the first half of the string onto the stack
    for (int i = 0; i < n/2; i++) {
        stk.push(str[i]);
    }

    // Iterate over the second half of the string and compare each character with the popped character
    for (int i = (n+1)/2; i < n; i++) {
        // Pop a character from the stack
        char ch = stk.top();
        stk.pop();

        // Compare the popped character with the corresponding character in the second half of the string
        if (str[i] != ch) {
            // If the characters are not equal, the string is not a palindrome
            cout << "Not a palindrome" << endl;
            return 0;
        }
    }

    // If we have successfully popped all the characters from the stack and compared them with the corresponding characters
    //in the second half of the string, the string is a palindrome
    cout << "Palindrome" << endl;
    return 0;
}
```

Description

The code takes a string as input from the user and uses a stack-based algorithm to check whether the string is a palindrome or not.

First, the code prompts the user to enter a string. Then, it creates an empty stack that will hold the first half of the string. It gets the length of the string and iterates over the first half of the string, pushing each character onto the stack.

Next, the code iterates over the second half of the string and pops each character from the stack. It compares the popped character with the corresponding character in the second half of the string. If the characters are not equal, the code concludes that the string is not a palindrome and exits the loop. Otherwise, if all characters are compared successfully and they match, the code concludes that the string is a palindrome.

Finally, the code outputs the result of the palindrome test to the user. If the string is a palindrome, it outputs "Palindrome". Otherwise, it outputs "Not a palindrome".

In essence, the code uses a pushdown automaton (in the form of a stack) to check whether the string is a palindrome or not, by comparing characters from opposite ends of the string.

Output Screenshots