Name: Adarsh Dhakar

Roll No.: 22CS01040

Repository Link: Github Link

Deployed Link of React Simulation: React App

****The CPP code is attached in the submission. This is a pdf report.**

****More detailed README is on the Github Link.**

****React App has the simulation using Javascript, React, and few styling libraries, the code is in the Github repo.**

Regular Expression to DFA Converter

This C++ program provides a command-line tool to convert a given regular expression into an equivalent Deterministic Finite Automaton (DFA). The conversion process involves several key stages of compilation theory:

1. **Parsing and Validation**
2. **Infix to Postfix Conversion**
3. **NFA Construction**
4. **DFA Conversion**

Supported Operators and Alphabet

- **Alphabet**: The user defines the alphabet (e.g., a,b,c) at the start of the program. Any character not in the alphabet or the operator list is considered invalid.
- **Union (+)**: Represents the logical 'OR' operation. For example, a+b matches either 'a' or 'b'.
- **Concatenation (.)**: Represents sequencing. For example, a.b matches 'a' followed by 'b'.
- **Kleene Star (*)**: Represents zero or more occurrences of the preceding element. For example, a* matches ε (empty string), a, aa, etc.
- **Parentheses (())**: Used for grouping to control the order of operations.

How to Compile and Run

The program is written in standard C++ and can be compiled with g++.

1. **Save the Code**

Save the provided C++ code into a file named regularExpressionToDFA.cpp.

2. **Compile the Code**

Open your terminal or command prompt and run the following command. The -std=c++17 flag is recommended.

g++ -std=c++17 regularExpressionToDFA.cpp -o re_to_dfa

3. Run the Executable

Execute the compiled program from your terminal.

./re_to_dfa

The program will then prompt you to enter the grammar and the regular expression.

Usage Example 1

```
This program converts a Regular Expression to a DFA.
The following characters are reserved operators: ( ) * + . E

Please enter a grammar (e.g., a,b): a,b,c
Please enter a regular expression (e.g., (a+b)*.a.b.b): a.b.c

Concatenation: a.b.c
Postfix Expression: ab.c.

--- NFA Construction ---
Start Node: 0
End Node: 5

Node no: 0 (Start)
  0 --a--> 1
Node no: 1
  1 --b--> 3
Node no: 3
  3 --c--> 5
Node no: 5 (Final)
  No outgoing edge.

--- DFA Transition Table ---
Start State: 0
Final States: { 3 }

State   |       a       |       b       |       c       |
--------|---------------|---------------|---------------|
 0      |       1       |       -       |       -       |
 1      |       -       |       2       |       -       |
 2      |       -       |       -       |       3       |
*3      |       -       |       -       |       -       |

(* denotes final state)
```

Usage Example 2

```
This program converts a Regular Expression to a DFA.
The following characters are reserved operators: ( ) * + . E

Please enter a grammar (e.g., a,b): a,b
Please enter a regular expression (e.g., (a+b)*.a.b.b): (a+b)*

Concatenation: (a+b)*
Postfix Expression: ab+*

--- NFA Construction ---
Start Node: 6
End Node: 7

Node no: 0
  0 --a--> 1
Node no: 1
  1 --E--> 5
Node no: 2
  2 --b--> 3
Node no: 3
  3 --E--> 5
Node no: 4
  4 --E--> 2
  4 --E--> 0
Node no: 5
  5 --E--> 4
  5 --E--> 7
Node no: 6 (Start)
  6 --E--> 4
  6 --E--> 7
Node no: 7 (Final)
  No outgoing edge.

--- DFA Transition Table ---
Start State: 0
Final States: { 1, 2, 0 }

State   |       a       |       b       |
--------|---------------|---------------|
*0      |       1       |       2       |
*1      |       1       |       2       |
*2      |       1       |       2       |

(* denotes final state)
```