Project Summary

Overview

Problem Statement: Although tools exist that visualize regular expressions and automata, most are outdated or limited in functionality. Students often learn the equivalence between regular expressions and finite automata only in theory, without a modern, hands-on way to observe how one transforms into the other. This project addresses that gap by developing a transparent, from-scratch system that reveals each step of the conversion process and provides real-time interaction with the resulting automaton.

Project Description: The proposed project, titled **Regular Expression Visualizer**, implements a Rust-based application that parses regular expressions and constructs their equivalent finite automata representations. It demonstrates the theoretical equivalence between regular expressions, nondeterministic finite automata (NFAs), and deterministic finite automata (DFAs)—a central topic in Automata Theory. Users will input a regular expression, view its ε-NFA, optionally convert it to a DFA, and test string membership interactively. The goal is to modernize and extend prior visualization concepts with improved clarity, interactivity, and maintainability.

Goals and Purpose

- Implement a manual lexer, parser, and automata generator for regular expressions without built-in regex or parser libraries.
- Apply Thompson's construction for $RE \rightarrow NFA$ and subset construction for optional $NFA \rightarrow DFA$ conversion.
- Create an interactive GUI using the **iced** framework for visualizing automata and testing strings in real time.

This project functions as both an educational demonstration of formal language theory and a usable teaching aid for exploring automata behavior.

Implementation Details

Developed entirely in **Rust**, the project emphasizes performance and safety. It consists of three core components:

1. **Core Engine:** Tokenizer and recursive-descent parser for regular expressions, constructing ε-NFAs and optionally DFAs.
2. **Simulation:** Implements ε-closure and transition functions to test if input strings are accepted.
3. **Visualization:** GUI for entering expressions, rendering state diagrams, and showing acceptance results through iced's canvas system.

Evaluation and Expected Results

Evaluation will focus on correctness of automata construction, clarity of visualization, and accuracy across benchmark inputs such as $(a|b)^*abb$ and $(ab)^*|c$. Metrics including state counts, transition accuracy, and acceptance verification will support analysis. The final submission will include full Rust source code, performance discussion, and a live in-class demonstration.