Senior Design Requirements Document

Improved Visualization for Formal Languages

Group Members:

Chris Pinto-Font (cpintofont2021@my.fit.edu)

Vincent Borrelli (vborrelli2022@my.fit.edu)

Andrew Bastien (abastien2021@my.fit.edu)

Keegan McNear (kmcnear2022@my.fit.edu)

Faculty Advisor:

Dr. Luginbuhl (dluginbuhl@fit.edu)

Client:

Dr. Luginbuhl

Florida Institute of Technology

September 4, 2025

Table of Contents

1. Introduction

1.1 Purpose
The purpose of this document is to define the requirements for a new application that visualizes Deterministic Finite Automata (DFA). The system will provide educators and students with an intuitive and visually appealing tool to create, animate, and minimize DFAs as a teaching and learning aid.

1.2 Scope
This DFA graphing application will:

- Allow users to manually or automatically create DFAs.
- Plot DFAs graphically, designate initial/final/dead states, and track execution for a given input string.
- Animate the construction and execution of DFAs step-by-step.
- Automatically complete or minimize DFAs using internal algorithms.
- Provide an intuitive, “toolbox”-style GUI with hover-over labeling.
- Offer multiple-symbol transitions (including lambda), readable animations, and built-in user help.
- Format DFA graphs by aligning nodes to improve readability.
- Allow the user to better understand and learn DFAs through step by step animated construction and a “teaching mode”

The application will be able to run on any common desktop computer. It will operate securely, efficiently, reliably, and intuitively.

2. Overall Description

2.1 Product Features

The Better DFA Visualizer will include the following major features:

- Intuitive GUI editor to create and modify DFAs.

- Graphical plotting of DFA states, transitions, and execution paths.

- Step-by-step animation of DFA execution.

- Automatic minimization and completion of incomplete DFAs.

- Multi-symbol transitions between states.

- In-program documentation and tutorials.

2.2 Primary Users

The primary users of the Better DFA Visualizer will include:

- Students: Create and test DFAs; visualize DFA behavior.

- Instructors: Demonstrate DFA construction, execution, and minimization during lectures.
- Researchers/Developers: Experiment with automata concepts.

2.3 Operating Environment

The Better DFA Visualizer will operate in the following environment:

- A computer environment capable of running simple .exe files
- Capable of running without need of WI-FI or external connections.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The application shall provide:

- A graphical “toolbox”-style editor resembling Microsoft Paint.

- Hover-over tooltips explaining each function.

- Tabs for DFA creation, animation, and minimization.
- A learning experience that allows the user to see how DFAs are built
- Animation to show dynamic DFA reduction.

3.1.2 Hardware Interfaces

- Standard desktop/laptop computer.

- Mouse/keyboard based input for drawing and editing DFAs.

3.1.3 Software Interfaces

- Onboard backend logic system utilizing Python libraries

3.1.4 Communications Interfaces

- Not applicable at this stage; the system is standalone.

- Future versions may include online sharing or saving of DFA projects though such a feature is beyond the current scope of the project.

3.2 Functional Requirements

[FR-1] Users shall be able to create and edit DFAs graphically.
[FR-2] Users shall be able to input strings and visualize DFA execution step-by-step.
[FR-3] The system shall allow designation of initial, final, and dead states.
[FR-4] The system shall automatically minimize DFAs.
[FR-5] The system shall automatically complete incomplete DFAs.
[FR-6] Users shall be able to add multiple symbols to a single transition.
[FR-7] The system shall provide in-program documentation/help.

3.3 Performance Requirements

[PR-1] The system shall load and render DFA diagrams with ≤ 2 seconds delay for small DFAs (<50 states).
[PR-2] DFA execution animation shall proceed at a user-adjustable speed with no frame loss.

3.4 Design Constraints

- The application shall be developed in Python.
- Logic shall be handled with backend C++ implementation
- The application will implement Animations and DFA minimization algorithms.

3.5 Software System Attributes

- Reliability: The application shall handle invalid DFA definitions gracefully without crashing.
- Availability: The application shall be usable offline on any supported machine.
- Maintainability: Modular design will allow future feature expansion.
- Portability: Python-based backend ensures cross-platform support.

3.6 Other Requirements

- The program shall include built-in documentation accessible from within the interface.
- The GUI shall use high-contrast symbols and large text for legibility.

4. Non-Functional Requirements

4.1 Performance

[NFR-1] Animations shall run smoothly at 30 FPS or higher.
[NFR-2] Loading DFAs with up to 100 states shall complete in ≤ 3 seconds.

4.2 Reliability

[NFR-3] The application shall automatically save work every semi-regularly to prevent data loss.

4.3 Security

[NFR-4] If online saving/sharing is implemented, files shall be stored securely and access-controlled..

4.4 Usability

[NFR-5] First-time users shall be able to create and run a simple DFA within 5 minutes.
[NFR-6] Tooltips and documentation shall explain all major functions.

8. Conclusion

This document outlines the requirements for the **Improved Visualization for Formal Languages** project. By adhering to these specifications and milestones, the team aims to create a user-friendly, visually engaging application for DFA visualization that meets the educational and functional needs of both instructors and students.