EECS 510 Final Project

Lillian Brooks-Kanost and Eyassu Mongalo
Dr. Jennifer Lohoefener

EECS 510

November 25, 2025

1 Design a Formal Language

Create a non-trivial language of your choice. Your language can be based on patterns, constraints, hobbies, programming languages, or purely imaginative constructs. Get creative and have some fun here. Examples include (but are not limited to): arithmetic expressions, colloquial language (e.g., slang), music, and sports. Your language must include an alphabet (set of symbols) and a clear set of rules or patterns for valid strings in the language (semantics). A short written explanation (1-2 pages) describing the intent, structure, and purpose of the language must be provided.

2 Grammar

Using formal grammar rules (regular, context-free, unrestricted, depending on the complexity of the language), write a precise grammar that generates all valid strings of the language. The grammar should be specified using production rules that clearly define how strings in the language are formed.

3 Automaton

Based on your designed language and grammar, implement an automaton to recognize whether a given string belongs to the language. If your language takes into account memory or state, be sure to use a PDA or Turing machine. For this section of the project I am expecting a visual diagram of your automaton. No hand-drawn visuals will be accepted, you must generate some form of digital automaton (e.g., LaTeX or Graphviz).

4 Data Structure

Design and implement a data structure that represents the automaton in memory. The data structure should capture the essential components of the automaton such as states, transitions, start state, accepting states, etc. You must include a written description of your data structure, clearly defining each component and how they work. For example, the below NFA could be represented as a file containing 11 lines. Each line within the file serves a specific purpose: Line 1: A whitespace-separated list of the states Line 2: A whitespace-separated list of input symbols Line 3: The start state Line 4: A whitespace-separated list of accept states Lines 5-11: Each line represents one transition with each transition containing exactly three whitespace-separated elements: 1) the state the transition leaves from, 2) the symbol the transition reads, and 3) the state the transition goes to. Automata File specification: File specification If your automata is a PDA or Turing machine you will need to include representations in your data structure for the stack alphabet, pushing and popping from the stack on a transition, tape symbols, and directional moves respectively.

5 Testing

Write a function (in code) that tests whether your automaton (part 3) represented as a data structure (part 4) accepts a string, w. Your function should return 'accept' along with an accepting path if the automaton recognizes the string and 'reject' otherwise. For example:

accept(A, w)

Where the argument, A, is the automaton to use and w is the string to test. Note: if you are defining your data structure as a file you will need a helper function to read your data structure into memory.

Using the same example from part 4 as our automaton, the output of accept(A, "110") would be:

Accepted output

While the output of accept(A, "11") would be:

Rejected output