James Testa
Foundations of Computer Science
Project Write-Up
5/11/2018

I was faced with the task of creating a program that would work similar to Perl's 'grep' function. To do this, I created a program that would take a regular expression and turn it into a non-deterministic finite automata (NFA). From there, I took the NFA and converted it to a deterministic finite automata (DFA). I then used that DFA to search through files and output matching strings to the screen.

The first task that I confronted was to create a structure to hold and represent the regular expression. With three basic expression attributes (star, or, and concatenation), I had to create a structure that held lists of terms and that knew the attribute the term was. With my custom class, I included two vectors of terms and booleans to identify which attribute the term was holding. With this structure set up, it allowed the regular expression to be parsed and hold a form that would be able to be converted into an NFA.

The definition of a DFA and an NFA are roughly identical; They both require a List of States, Alphabet, Start State, Transition Function, List of Accept States. To represent a state, I used a string. A string can be easily compared and concatenated while also being part of the C++ standard library. It would have been more memory efficient to store the states with an integer but it would have created many more problems when trying to manage and identify states. I used a vector of strings to represent the List of States.

The main storage of the transition function proved to be the hardest to create. To represent a transition function, you need the current state, the input character, and the next state. To represent this you really need a 3D array to store all the information. To accomplish this task, I used a C++ map which works similar to a heap. Because I wanted to use C++ standard library data structures, I represented the transition function as a Map of Maps. This allowed the structure to take in two variables and output one response. Giving the structure a string (current state) and an input (character) made it possible to have the function return a vector of strings. This vector of strings represented a list of states that the current state could travel to. This proved to be vital in representing an NFA. With multiple paths due to empty transitions, there were many different scenarios you could run into that required these many different layers in the transition function.

A DFA is a simpler NFA, and allowed me to use the same structure for both machines. With a few modifications to running the automata, I was able to represent an NFA and a DFA in the same structure. Converting the regular expression to an NFA proved to be reasonably straightforward, by adding states as you traverse the expression tree. Converting the NFA to a DFA was more complex. In order to get all of the possible paths required, I checked all of the empty transitions and created new states. One issue that I did run into with converting to DFA was looping on empty transitions. This stumped me for a while until I created a recursive

function that carried the previously seen states. If I were to go back and update this project, I would try to create a non-recursive function for this process.

With a working regular expression to DFA program, it was straightforward to create a working ‘grep’ function. This really did not give me any issues when I was creating it. Using some of the basic Perl regular expression style guides, I was able to implement key phrases that allowed variables to be written in the regular expressions. Overall, deciding on the transition function was the main issue and converting from NFA to DFA using the same structure took the longest time to implement.

Ultimately, I felt this project was simple enough to implement through the semester. I learned a lot about finite state machines and regular expressions. With the completion of this project, I have better understanding of project planning and execution. I gained knowledge in C++ standard libraries and code flow. I plan on potentially expanding on this project in the future.