CMPT440

Formal Languages + Computability

Alexander Goncalves
Professor Rivas
5/7/2017

1. Project Premise

The main idea behind my project was to mimic a Linux/Unix Shell, to execute simple commands. This would be done to illustrate that such a program could be built using a state machine as the driving force behind the program. My initial concept was to have a small UI identical to a Bash Shell, where a user could write commands, which then would get sent to a DFA. When the data was sent there, it would be processed using the switch case approach to a DFA, which would then either execute some command or result in an error state. Afterwards I went into the specifics to see which technologies were available to make this happen.

2. Technologies Used

After looking into my options, I decided to use Java as my programming language for this project. The reason for this is because Java's strong object-oriented syntax would be useful for the code structure I had in mind. For example, I wanted each bash command to exist as it's own entity separate from the actual DFA, Java Objects are ideal for such an abstraction. In addition to this, Java has mature UI frameworks which were necessary for this project. The one I ultimately chose was the Swing framework, because I already had exposure to it in the past. In addition to this, I found it easier to use than the main alternative, JavaFX. In the end Java and Swing were a good match for this project.

3. Code Structure

The point of entry for my code is the main method inside of the UI class. The UI class contains a lot of behavioral methods/listeners which tell the program how to deal with mouse clicks and certain keyboard keys. The user can enter text input and upon hitting enter, the data is processed in the DFA object. This object does most of the heavy lifting, and goes through the string of data word by word to determine if the input is in a valid state. The DFA class contains an instance of all of the individual commands. Each individual command knows how to process the data that has been sent to it, which then returns output as to whether the operation was successful.

4. Commands

The initial command I implemented was Whoami, which in a UNIX system returns the name of the user that's logged in. The second command I made was Mkdir, which creates new directories, when a path is specified. Following this is Echo, which returns a given input, as output on the screen. Next, the Touch command creates new files in the Desktop directory, where file names are specified in the command. Rm gets rid of files, when the name of the file is specified with the full path. Cp copies one file into a given directory. Man will return how to use the commands, when a given command is specified. Lastly, Head returns the first 10 or less lines of a file when the path of the file is given.

5. Command List

Below is all of the valid commands that can be made in this DFA.

whoami

whoami -help

touch <Filenames>

rm -help <Full File Path>

rm <Full File Path>

mkdir <Full Directory Path>

man <Any of these commands>

head <-1 to -10> <Full File Path>

echo <Text Characters>

echo -help <Text Characters>

echo -up <Text Characters>

echo -down <Text Characters>

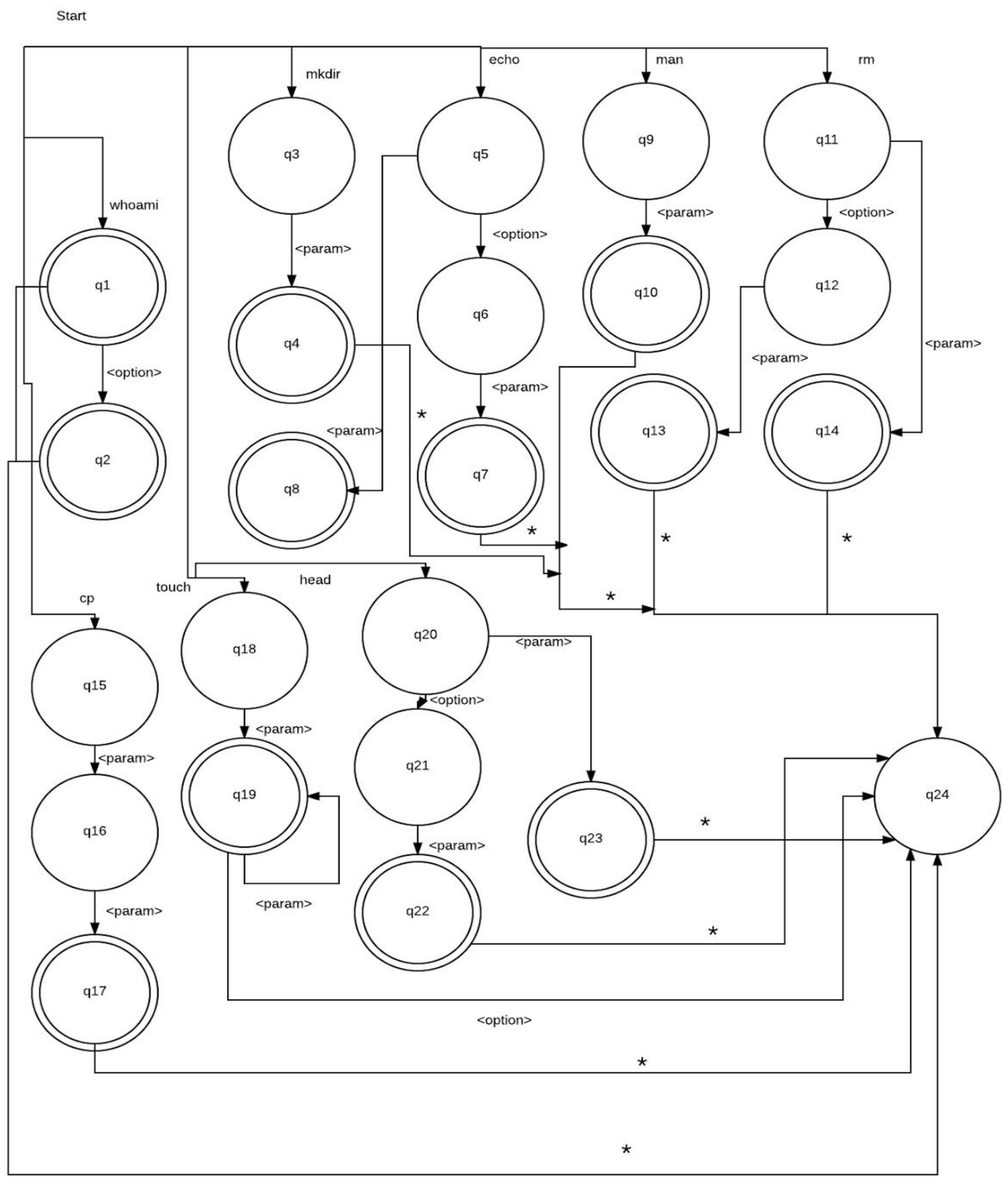
cp <Full File Path> <Full Directory Path>

6. Error Handling

Error handling occurs in twice through each individual text processing. The first time it occurs is during the actual text processing in the DFA. This ensures that the content of the input is valid, this checks if an option (such as -help or -up) has been set appropriately. If the DFA goes to an accepting state, a second error check happens within the execution of the command's function. This checks for a correct file path, or to ensure that IOExceptions don't happen. The reason this second check is necessary is because something may be valid as DFA text input, but then may not actually be possible on the system, such as a reference to a non-existing file.

7. DFA

Below is the DFA of this program.



8. What I Learned

From this application I learned how powerful state machines truly are. With the necessary research I discovered how much software actually uses DFAs and NFAs. In particular, I now understand where a state machine could be useful in a compiler or an interpreter. If I had to redo this project, I believe I would pre-process the user input into tokens. This would have made it easier to process the input and would have made the DFA implementable using the table approach. However, the current program is satisfactory and shows the usage and implementation of state machines.