Matthew O'Brien
CS311
Project 1 Write up

The format I used to read in the DFA to the program was in the form a JSON file. This was because of how simple it was to assign each element of the DFA class to a given list or single item. After some research, the way to organize the transitions in the DFA were really simple using JSON as well. I'm not entirely sure how the JSON file and import works with Python, but I'm assuming it has something to do with indexing. The import function would search out the given character inputs, and assign the elements of those to the corresponding element of the DFA class. This allowed for the different elements of the DFA to be easily assigned within the program, and easily altered assuming the correct format was used when building the DFA. The more basic elements of the DFA were easily represented in text by a list or a single element. These were read in with the JSON import function in Python.

The transitions were represented in a slightly different way. Each transition function had a beginning state that was given, then the transition function could read the corresponding input character and the state that it would move to after the function call. Using the JSON file and python it was much simpler to add the transition functions with this format, and it allowed to easier alter and change the DFA without changing any of the code in the program. Something I found strange while working on the actual input of the DFA was how python seems to define the class. I was trying to create a sort of add_dfa function that would be able to add in any dfa from a given file, so the program could have multiple DFAs running and allow the user to select between them. I could only really get the JSON file to load the way I have it implemented, where the file name is hardcoded into the add, which then loads the elements of the class from the file. Since the project specification didn't mention anything toward “user friendliness”, I decided to go with what I knew would work for this project. Because of that, when changing the DFA, or implementing your own DFA, line 66 “testDFA” will need to be changed depending on which JSON file is being imported into the program. It is also worth noting that the format shown in the example DFAs will need to be followed when implementing another DFA for the program. Any change to that format will cause the program to fail. The program is also not built to test for DFAs that do not work. Any of the transition functions need to be set using real states from the DFA, and only with the alphabet provided. This is only for when building the DFA in the text file. Any DFA that does not work within the text file will cause the program to fail.

The test program will prompt the user to input a string to test. A description of the DFA will be given before the input, as well as the alphabet of the DFA. If by accident a character is input that is not in the alphabet, the program will catch on this input and cancel the test. In order to test the string against the DFA, I used an old habit from previous computer science courses. For some reason, pointers really stuck out to me, and the idea of current, and next pointers. Using that idea and python's ability to take the input string piece by piece, I was able to use that with the transition functions. An initial “current” position was set to the starting state of the DFA. This initial state and the next input of the string were read into the transition functions as a “next” state. The current state changes to the next state, and the for loop is repeated throughout the string. Before the current and next portions of this function are used, there is an error check for invalid inputs. Each individual input is tested against the alphabet described in the JSON file. If at any point in the input there is an invalid character, the function will notify the user, and terminate the function. This will go back to the input selection until the user types in “exit” to quit testing possible strings.

Most of the testing for this program came from making sure the JSON file was properly formatted. Moving from what state to another was very simple using current and next, thanks to Karla's 162 and 163 classes for that habit. After making the if statement for the alphabet against the string, it

was simply to make sure the final state of “current” was recognized as either an accepting state or not. That was done by an if statement to see if the current was an element of the accepting states described in the class. Anything else would be considered none accepting and the appropriate message would be sent to the user. Something I noticed when using the loop for the user input of strings, the input cursor would always begin at the start of the line rather then after the “?”, and I could not find a way to remedy that, but when typing in the input it ends up in the right spot. This could just be from the IDE I used for the program. Like previously stated, most of the testing for this program came down to getting the format of the JSON file. Since this was the first time using a JSON file for me, it took a bit of time to get this right. I had a lot of issues with the transition functions. One thing worth noting, for the test DFA labeled “testDFA” for the binary multiples of 5, I used the DFA from the homework. And since for the homework we were allowed to ignore a starting state that does accept the string, even though the empty set is not an element of the language, it will work for this DFA.

The common tests that were done for this project were short and long inputs that were accepted to be in the language of the DFA or to not be within the language. This was because of the assumption that the JSON file was properly formatted for an accurate DFA. The program was designed in this way to simplify things. Rather then checking to make sure that the DFA worked, it would only test to see if the input was apart of the language of a working DFA. Without the properly formatted DFA, none of this will work as intended. Though with the way the JSON file is formatted, it does allow for many different types of DFAs to be implemented and tested. All of course, working under the assumption that the DFA created is accurately formatted and built with the JSON file. A sort of template is provided in case of addition DFA's to test with the program.