CSE 30151 Spring 2014
Monday, February 10th, 2014
Nicholas LaRosa

Project 1 Report

This project was completed in the Python programming language and made executable via a small shell script. All testing was performed on the student00.cse.nd.edu machine with various test input. Debugging was completed using simple print() statements to monitor that state of variables and data structures within member functions and object instances.

This program was approached from an object-oriented standpoint; an NFA is represented by an object (defined in NFA.py) that contains a list of states, a list of symbols (making up the NFA alphabet), a double-dictionary of lists containing transition rules, a string containing the start state, and finally, a list of accept states. These data structures make up the definition of the NFA, and are initialized via the object member functions addStates(), addAlphabet(), addTransition(), addStartState(), and addAcceptStates(). These functions are implemented in such a way that an NFA's alphabet and state list must be established before any transitions, accept states, or the start state can be defined. This is because transitions can only occur from valid states on valid alphabet symbols, and the start state and accept states must be contained within the state list.

Once the user has constructed an NFA, they begin defining the NFA using the functions mentioned above. With the state list, alphabet, and accept states being contained within Python lists, lookup and access is made easy due to the fact that any type of variable can be appended to a Python list. Somewhat more complicated is the data structure required to contain all transition rules. Because a transition occurs from a start state on an alphabet symbol to result in another state, a double dictionary of lists performed well. The first dictionary contains the start states of all defined transitions, and the nested dictionary within each of these dictionary keys contains the alphabet symbols of all defined transitions. Each of these dictionaries likewise contains a nested list, which in turn contains the list of all result states on the corresponding start and input. An array is necessary because the definition of an NFA allows a start state to transition to multiple states on the same input. Additionally, the nested dictionaries allow for states to transition on different alphabet symbols. Ultimately, the transitions can be accessed in a similar fashion to a double array, with the first index containing the start state, and the second index containing the alphabet symbol input. This allows for smooth syntax throughout.

Per project guidelines, the NFA definition is provided via an input file. Thus, the processFile() member function will read an input file and define the corresponding NFA instance using the previously mentioned add…() functions. Once this is complete, the user can have the NFA accept tape input via the getInput() function. Taking the number of tape inputs as its first input from standard out, the getInput() will subsequently call processInputLine() for that provided number. This function keeps track of the state branches by transitioning both on the explicit symbol input and on the empty string. This requires that the recursive transitionOnEmpty() is called at every input on the currently-visited states; this function, in turn, marks states as visited and returns upon looping. Although understanding the best way to recurse proved to be quite difficult, the current implementation does the job well.

Test programs:	**Tape inputs:**
nfa1.txt	nfa-input1.txt
nfa2.txt	nfa-input2.txt

Usage:

```
./nfa nfa-definition.txt < nfa-input.txt > nfa-output.txt
```