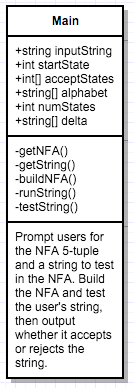
Nick Sohm

CS357A

12/2/15

Project Summary

1. Nick Sohm
2. Open NFA\_Final.py in IDLE and run the module.
   1. NFA\_Final.py
      1. Note: Incorrect input form is not checked for and could crash the program. If this happens, restart the module.
      2. Enter a number of states > 0.
      3. Enter 1 or more symbols in the NFA’s alphabet. Leave the prompt blank and hit enter to stop inputting symbols. (This input isn’t actually used, feel free to leave it blank)
      4. Input transitions between states. These follow the same rules as for inputting the NFA’s alphabet. To input an epsilon transition, use ‘e’. Note that you can input multiple transitions between the same two states to account for more than 1 symbol transitions.
      5. Enter the start state. (e.g. If the start state is q0, enter 0.)
      6. Enter the accept states. These follow the same rules as for inputting the NFA’s alphabet, and the start state.
      7. Enter the string you want to test on the NFA.
      8. If “The String was ACCEPTED” is present in the output, the NFA accepts the input string. It rejects otherwise.
      9. Input a 0 to stop execution, a 1 to enter a new string, and a 2 to enter a new NFA.
3. The original UML diagram for the project:   
   I didn’t have much recent experience with Python before making the UML diagram, so the program turned out a bit different. I didn’t use arrays of a specific type, but rather lists. The algorithm that would have essentially been used in the runString() method was implemented as a recursive function. This function starts at the input start state and builds the NFA, keeping track of the state that it is on, and the current symbol it will be using from the input string. Once it hits the end of a path available on the NFA, it checks if the current state is an accept state, and if the end of the string has been reached. If any path accepts, the NFA accepts, and if they all reject, it rejects. If it ever hits a branching path, it starts another instance of the function with the proper input. The buildNFA() method is basically covered after the definition of the algorithm function in the code, handling the input and building of the NFA 5-tuple and string. The data structure I used detailed all the transitions in the form [[startState, endState, symbolUsed], […], […]], where the […] are other sets of transitions like the first.
4. The attached file NFA\_TestCases contains the two main tests run on NFA\_Final as detailed in the proposal (also included). These two tests cover many of the different ways in which an NFA can be built (epsilon transitions, more than one of a certain symbol-transition from one state, etc.), and alongside more ridiculous tests I’ve run myself, I’m pretty confident that this program will work for any proper NFA.