Thiam Kiat Sim CSCE 428 Project Report

For question 1, the problem is rather straightforward, my program parse the dfa file and save them onto vectors. After that, using the data from delta, construct a table with a size of number of states * number of symbols. The content of the table are the "destination" of the states , this can later be used to traverse the input string through the table. After using the input string to traverse through the table, it'll stop at one of the states. Then check whether the states is one of the accepted states or not, if it is, it's accepted, else, it's rejected.

For question 2, the idea of my solving technique is based on this idea, C = (L(M1) ∩ $\underline{L(M2)}$) ∪ ($\underline{L(M1)}$ ∩ L(M2)), where $\underline{L(M1)}$ implies complement of L(M1), then check whether C is empty using breadth first search by checking whether there is a path from starting state to accepting state. The way I made the implementation is that, create a table for each DFA (M1,M2) similar to what I did on question 1. Then, make a cartesian product of the 2 tables. Now, negate the accepting states of M1, so that the accepting state ones become non-accepting states, and non-accepting states becomes accepting states. This will form the definition of ($\underline{L(M1)}$ ∩ L(M2)). Use similar technique to form (L(M1) ∩ $\underline{L(M2)}$). After this is done, I can simply check whether one of them is not empty using breadth first search. If both of them are empty, the two machines are equal, otherwise, they are not equal. If it's a membership relationship, one of them would be empty, and another would not be empty. The counter-example string can be obtained by using the same path the breadth first search traversed to reach the accepting state.

Some potential limitation of my program would be the way I parse my input are such that the first state in the .dfa file is always the starting state, and the delta always start with starting state as well, changing them could result in unintended behaviour.

Some example input the program should accept:

- M1.dfa 0110
- M1.dfa 010101
- M3.dfa 101110
- M4.dfa 010001
- M1.dfa M2.dfa
- M2.dfa M1.dfa

Some example input the program should reject:

- M1.dfa 011000
- M3.dfa 110101
- M1.dfa M3.dfa
- M3.dfa M4.dfa

The only equivalent DFAs are M1.dfa and M2.dfa. M3.dfa, M4.dfa are different.