Project Report

Guide for running the code:

1. Go to the directory cs461/assignment1
2. NFA2DFA.java has the source code and only this file will be needed to run
3. On terminal, to compile the code, type **javac NFA2DFA.java**
4. To run the code, type **java NFA2DFA <input file name.txt>**
   For example, for input file input1.txt, I would type **java NFA2DFA input1.txt**
5. Put the input file inside the directory cs461/assignment1, where the source code is.

Report:

The problem for this assignment was to read the standard input from a file and then make the graph of corresponding NFA. After constructing the NFA, I needed to convert it to DFA and generate standard output similar to corresponding smaple output.

I used Java to solve the problem. First I created a State class to define a state and save the properties of each state. Using BufferReader I read the file line by line and categorized the input line based on their orientation. For each category I used appropriate condition to parse to tokenize the words of each line and handled them accordingly.

I used List to store all the states, a hashmap to store the edges against each input symbol as key. I then allocated the hashmaps for all input symbol to a list that stores the hashmaps for the corresponding state.

After constructing my NFA, I implented the algorithm for the conversion of NFA to DFA. First I generated the E_Closure of the initial state. Then, for rest of the states, after finding edges for each input symbol, I sent the edges list to E_Closure and also checked if the list already existed in my previously found E_Closure Lists. If it's a new states I pushed it in the queue and kept doing the above mentioned task until the queue became empty.

With this process I found all the states of new DFA and their edges for each input symbol.

Then I printed the outputs.

I debugged the code by printing stuffs that I suspected was causing errors. I also did some tracing at one point. I tested my solutions using sample inputs and matched the outputs with the sample output.

I also ran the code on Hydra server.

The algorithm was pretty straight forward. Keeping tracks of edges with corresponding input symbol was a challenge though. Also matching the outputs with the output sample specially matching with the white spaces was a bit tedious.