Jesse Alltop

Asg 2

jalltop

/home/ADILSTU/jalltop/Public/IT328myWork/isutech2016

**Summary:**

First and foremost, I did find this assignment to be rather challenging, I felt that I dedicated a pretty significant amount of time to the assignment and still needed extra time to complete it to my liking. This resulted in three different occasions where I threw out what I had built to start over fresh.

For the classes that I used through this assignment I have State, which represents a state in the NFA composed of a name and the transitions it can make. Then I also have a class called dfaState, which of course is a state in the DFA machine that results from the conversion of the NFA, it also contains a name, a list of NFA states that compose this state, and a hash map that has a input character as a key and a list of the NFA states that can reached given that key. Those are the only two classes that I implemented for this assignment. It is also worth noting that all of the parameters are public, I know this is poor practice, but with running behind schedule it made for cleaner and quicker coding.

So, with the actual conversion of the NFA to DFA, I first begin with building up the NFA machine. This is done within the createNfa method which takes in a file scanner. Within the createNfa method we set the number of states, create the states, find all available input tokens, add the available transitions for each state to the appropriate state, and finally save the initial and accepting states for the machine. As far as data structures go I used ArrayList, so I could dynamically grow with the number of NFA states as I needed to. Since the input characters are static I used a simple array for those.

Next, I move on to converting/building the DFA from this now existing NFA. For the conversion I followed the subset construction of the NFA. So, this method begins with creating the starting state for the DFA which is simply the lambda transitions of the starting state of the given NFA machine. This new state is then added to a set of DFA States, I chose a set because I only wanted unique values to be added to the set of states, a set seemed like an easier way to achieve this instead of validating every new entry to a list. Next I moved from this function to a new function called createDfaState which takes a parameter of a dfaState, the first call of this method is with the starting DFA state. The method createDfaState is the method that will do most of the algorithm I used for this assignment. The way this algorithm works is it takes the state given and will find any states that can be reached consuming one input, which may be a set of NFA States and then form that set it will find their lambda transitions and add any new states to the set. If this collection of states is not in the dfaStates set it will be added to the set. Within this createDfaState method it will also check that given an input symbol there is a state for it to transition to, and if that state has not been created yet it will create it and add it to the set of DFA states; if all inputs have a transition then it will mark the state and is complete and I have a fully built state in the DFA machine. This algorithm will be repeated for any new states added to the set until all states have been checked.

Finally, once all states have been checked we have the DFA converted from the given NFA. The only thing left to do is check the file of input strings to see if they are valid within the given machine. This is as simple as moving one character at a time within the string until we reach the end and checking to see if the state we finish in is in the collection of accepting states if we are then we accept if not we reject.

Final notes: Initially I had trouble deciding how I wanted to implement the NFA states, in my mind I wanted to use a HashSet, but there might be multiple moves available for a single input, once I started fresh it seemed easiest to just insert a string with all possible moves for that state and to parse it as I needed, there is more than likely a more efficient method where it is only parsed initially, but this method seemed the most straightforward to me. I also ran into some problems where finding transitions, either lambda or from input where giving me the wrong states, which after countless tries I finally realized was from my states not being in the order I expected them to be, which is why I wrote a method to sort the DFA states based on name that way I could have a standard way of finding a particular state. Overall, I did find this to be a challenging assignment and at times I will admit I was more frustrated than I would like to admit, but upon completing this assignment, although late, I feel a great sense of accomplishment.

**Output:**

Sigma: a b

------

0: {1} {4} {2}

1: {5} {2} {}

2: {5} {} {}

3: {} {} {0}

4: {} {1} {3}

5: {} {} {4}

------

s: 0

A: {5}

To DFA: [2: {5} {} {} , 0: {1} {4} {2} ] [3: {} {} {0} , 5: {} {} {4} , 4: {} {1} {3} , 2: {5} {} {} , 1: {5} {2} {} , 0: {1} {4} {2} ] [3: {} {} {0} , 4: {} {1} {3} , 2: {5} {} {} , 0: {1} {4} {2} ] [3: {} {} {0} , 4: {} {1} {3} , 2: {5} {} {} , 1: {5} {2} {} , 0: {1} {4} {2} ]

Sigma: a b

------------------

0: 1 2

1: 1 3

2: 1 3

3: 1 3

------------------

s: 0

A: {1}

The following strings are accepted:

aabaa

aaaaa

aaaaaaaaaa

aaaaaaaa

aaa

aaaaaaaaaaaaaaa

bbaabba

ba