NFA2DFA

Synopsis

The NFA2DFA converter lab is an assignment where we have to use the subset construction algorithm to convert an NFA into a DFA. The problem offers great flexibility in the methods that the programmer can take to solve it.

Approach

Because converting this without knowing how to do it by-hand is non-trivial, I tried to solve these examples by hand before actually attempting to code it. When I got how it worked, I then had to implement it in a language. At first, I was using C (GNU89), but the memory management that I had to deal with when using my libraries was too much of a hassle whenever C++ data structures handle their own memory. Therefore, I opted to use C++ instead. The part that took the most time (both in C and in C++) was parsing the input.

Because I switched to C++, it allowed me to abuse the functionality of C++ that sets it apart from C, such as classes and templates. I made an NFA class and passed it by reference into every function I needed it for. The states of the NFA machine are stored as strings as opposed to integers because I thought ahead and knew the DFA nodes will be named with strings. The NFA class includes multiple data structures for the same data to make it easier to access states. For instance, I can access the first state of the machine by going into a vector at index 0. But I can also go into a map at the index of that state's name and go straight to it. Since they are pointers, they are literally pointing to the exact same data.

In terms of the algorithm, it's essentially how the diagrams in class were given. However, it was implemented with a bit more thought into what data is already available to save time and make lookups faster. For instance, instead of having to keep track of what states are being marked in e-closure, there is a set of strings that we can use to check if a state is already in there. This makes most lookups be in $O(N \log N)$ time as opposed to $O(N)$ time. My "move" function has the "e-closure" algorithm implemented into it. Therefore, the function for e-closure is only called on the very first node in order to get the first DFA state. Then it is pushed to a vector. A for loop is used and iterates through that vector (which is incrementing in size as we iterate through it). While iterating through the vector of DFA states, "move" is called on every state for every

alphabet character that isn't the "E". Any DFA nodes found via "move" is then pushed to the same vector. This adds the property that when the for loop ends, the lookups for all nodes has ended. It also saves the hassle of having to use a queue.

Debugging Solution

There actually wasn't much to debug in my attempt to do this assignment. Whenever I implemented e-closure and move functions, it kind of just worked. However, when I was implementing in C, I had multiple issues with pointer data and even had memory corruption. These issues actually passed on to my C++ build, where I initially used a queue for handling the construction of the DFA, and found out that memory was not being copied when I pushed to the queue. "gdb" was extremely valuable in debugging my code as it literally told me what was wrong with my code on most occasions. There were a few occasions where even gdb couldn't help, such as memory corruption in data structures, and pointers to objects that were freed prior. Thankfully, it worked out in the end.

Issues

Aside from memory issues stated above, there weren't many issues. The way I had the NFA setup allowed for so much flexibility that it reduced the number of issues I could have had down before I even had them.