Sarah Huang
COSC 461: Intro to Compilers
September 6, 2022

NFA to DFA Project Report

1. The problem I attempted to solve is to take a text file representing a nondeterministic finite automaton (NFA) and translate it into a deterministic finite automaton (DFA). You need to code the subset construction algorithm that utilizes the move and e-closure algorithms.

2. First, I looked at the textbook Compilers: Principles, Techniques, & Tools to get ideas on converting NFA to DFA because I was unfamiliar with the topic. In the text, Figure 3.31 through Figure 3.33 outlined pseudo-code which was useful in implementing the three algorithms. I coded the program in C++ because I have more experience in it. For my NFA data structure, I ended up with a map keyed with an integer (the state) with a value of a nested map keyed with a character (input alphabet) and a vector of integers (moves).

```
map <int, map <char, vector <int> > > nfaTable;
```

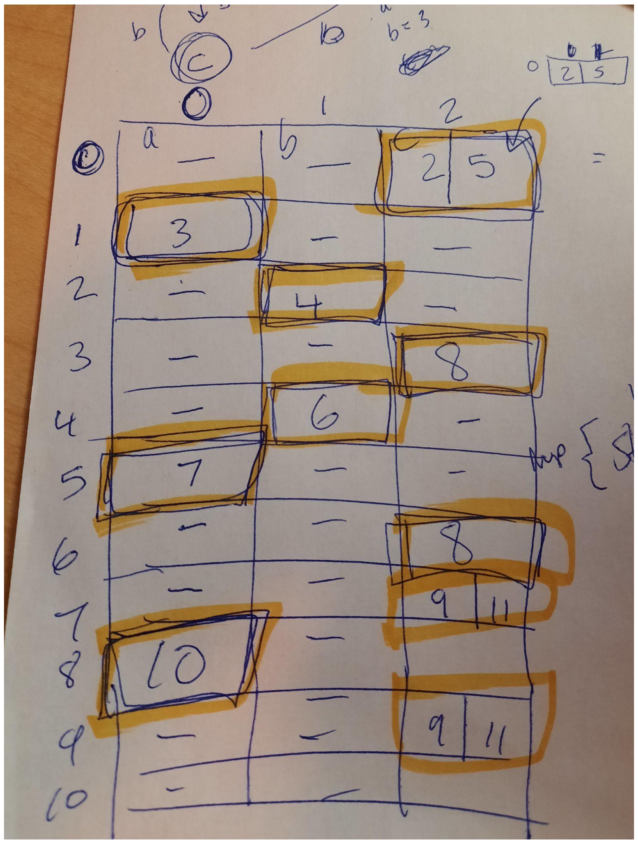
   Unfortunately, I could not reuse the same data structure for the DFA because there is a "marked" property and one DFA state could be used for multiple NFAs. I implemented a struct with a boolean (taken/marked), a vector of integers (multiple states), and a map keyed with a character and a value of an integer (input alphabet and moves). In practice, I use it with a map keyed with an integer (state) and value of the DFA struct.

```
struct DFA{
      map <char, int> moves;
      vector <int> states;
      bool taken;
};
map <int, DFA> dfaTable;
```

3. It was important to debug the NFA transition table once I finished reading in the input file. I printed it out and also tried to mimic the format of the input file, which came in handy later with print formatting at the end. I also tried using this visual NFA to DFA converter (http://alexklibisz.github.io/FSA-Animate/#/) to walk through the steps of the conversion. Other than that, I did my usual debugging technique of writing many print statements to check values or find where a seg fault is. I tested my solution by comparing the three given output files with my outputs.

4. The data structures I used, in the end, were not my first ones. I tried to use a 3D vector at first like in the picture I drew. To be honest, I did not think it through early on in the assignment. I wasted a lot of time trying to figure out how to use this vector system, and I did not realize how limiting it was, especially with the marked attribute. I think the transition table made me think of vectors. Also, I had to constantly remind myself what indices were what, hence the picture. I was hoping I could figure it out eventually, but it was a good decision to start over and rethink my data structures. Another problem I tackled was the specific formatting on the output file. I am not very good with `setw()`, so it took many attempts to get it just right. One last realization, it is incredibly important to git commit as much as possible. I finished the code but it looked really rough, so I took the time to neaten the code and eliminate unnecessary lines. After an hour later of many changes, I decided to commit again. While I was making these changes, I only ran the first output.txt which was a bad idea. I decided to run all of them, and the other two were wrong. I didn't know what exactly I changed to cause this, so I went back to the rough code and incrementally added each new big change and tested it immediately afterward. Wasted some time backtracking.

```
vector <vector <vector <int> > > nfa;
nfa [state][alphabet][move] (ex. nfa[0][2][0] = 2)
```



	0 a	1 b	2
0	—	—	2 | 5
1	3	—	—
2	—	4	—
3	—	—	8
4	—	6	—
5	7	—	—
6	—	—	8
7	—	—	9 | 11
8	10	—	—
9	—	—	9 | 11
10	—	—	