Ling 185A Final Project — FSA to RegEx Conversion

Code explanation:

Files:

- finalproject.hs — code for FSA to RegEx conversion
- FiniteStatePart2.hs — code given for Assignment 4. Only the functions needed for NFA → FSA conversion were kept
- RegexToFSA.hs — code submitted for Assignment 4. Only the functions needed for regular expression → FSA were kept. A reToString (from Assignment 2) was also included

The main file *finalproject.hs* has three key functions: *fsa_to_gnfa*, *convert*, and *test.*

fsa_to_gnfa, as the name suggests, takes in an FSA as input and outputs and equivalent GNFA, according to the algorithm given in Sipser page 71. To see the internal structure of the GNFA:

```
$ ghci finalproject.hs
*FinalProject> fsa_to_gnfa fsa_ex1[1]
([Reg 1,Reg 2,Start, Accept],"ab",[Start],[Accept],[(Start, OneRE,Reg
1),(Reg 2,OneRE,Accept),(Reg 1,Lit 'a',Reg 1),(Reg 1,Lit 'b',Reg 2),(Reg
2,Alt (Lit 'a') (Lit 'b'),Reg 2)])
```

On the other hand, the *convert* function, based on the algorithm from Sipser page 73, takes in a GNFA and recursively calculates an equivalent GNFA with one less state, until there are only two states left (start and accepting states). The transition from the last two states will be the equivalent regular expression. To see the regular expression:

```
$ ghci finalproject.hs
*FinalProject> reToString (convert (fsa_to_gnfa fsa_ex1))
"a*b(a|b)*"
*FinalProject> reToString (convert (fsa_to_gnfa fsa_ex2))
"((a(aa|b)*ab|b)((ba|a)(aa|b)*ab|bb)*((ba|a)(aa|b)*|1)|a(aa|b)*)"
```

When writing my code, I checked the outputs above with what was given in the textbook. A better and more general way to check for FSA to RE equivalence is to use the *generates* function on some test inputs. The function *test* in *finalproject.hs* facilitates this:

```
$ ghci finalproject.hs
*FinalProject> test fsa_ex1 "aab"
"FSA and RegEx have the same behavior. Both accept it."
*FinalProject> test fsa_ex1 "aa"
"FSA and RegEx have the same behavior. Both reject it."
*FinalProject> test fsa_oddCs [C,C,C]
"FSA and RegEx have the same behavior. Both accept it."
*FinalProject> test fsa_oddCs [C,C,V]
"FSA and RegEx have the same behavior. Both reject it."
```

[1] fsa_ex1 corresponds to the FSA in figure 1.67 (page 75), fsa_ex2 to the FSA in figure 1.68 (page 76)

The *test* function takes in an FSA and an input, and runs the *generates* function on the given FSA and the FSA constructed on the regular expression computed by the previous two functions. The full process is as follows:

1. given FSA → GNFA (fsa_to_gnfa)
2. GNFA → Regex (convert)
3. Regex → NFA (reToFSA)
4. NFA → new FSA (removeEpsilons)
5. check if given FSA and new FSA accept/reject the input string

As a side note, I've found that running on a bigger FSA as fsa_ex2 ran for a significantly longer time than smaller FSAs, particularly when converting the regular expression back into an FSA. The number of states blew up (which makes sense for an NFA to DFA conversion, which could be exponential in size), which made the calculations a lot slower.

Why I chose this project, and what I learned from it:

I chose this project because in both CS 181 and Ling 185A, we learned about the regular expression to NFA conversion (because it is relatively straightforward), but we did not learn the DFA to regular expression conversion. I was already curious about this topic, so to learn about and also implement it in code seemed like a good choice for me.

I definitely learned more about how this process is done, but also I had to think more about how to simplify regular expressions. The algorithm given in the textbook created a lot of redundancy in the regular expression it outputted (e.g., (b)(a|b)*(1)|0 == b(a|b)*). This couldn't be done in one recursive run, because something (r1|(r2.0)) should reduce to r1, but the recursion would have to go backwards in this case (you don't know that (r2.0) is going to reduce to 0 yet). In addition, coding in Haskell still seems to be a challenge for me, as I never used functional programming before this class, so this project also helped me with developing my Haskell skills as well.

I enjoyed reading about the theory behind FSA to RegEx conversion, and coding it for this project also solidified my learning.