B) Construct an initial state, and construct states that correspond to each symbol in the SLG’s alphabet. Draw transitions from the initial node to each node associated with an allowed initial symbol, with the transition meaning concatenating the corresponding initial symbol. Then, check the list of possible bigrams; draw a transition from the state corresponding to the first symbol of the bigram to the state corresponding to the second symbol of the bigram. Loop the state onto itself if the bigram consists of repeated characters. Label any states corresponding to allowed final symbols as exit states.

J) I used reToFSA recursively. The helper union, concat, and star functions that were written were used with reToFSA recursively so that reToFSA first converted their constituent RegExps; mapStates flatten was used to avoid state naming collisions. I considered the simplest cases of literals, and the zero/one regexs and manually outputted those FSAs; reToFSA simply built upon this framework recursively to build full FSAs. Variable names should be relatively self-explanatory. *name*_i refers to the set of initial states, and _f refers to final states. Just overall, the best way to describe my methodology is that I followed the graphical FSA representations from the class handout. For converting REs to FSAs, I simply considered that REs recursing over themselves can be covered by recursively calling reToFSA; It was trivial to implement the Lit, ZeroRE, and OneRE cases.