Overview

The purpose of this assignment is to broaden your understanding of the relationship between *nondeterministic* and *deterministic* finite automata by implementing the algorithm which converts a given NFA into an equivalent DFA. Recall that a *nondeterministic finite automaton* is defined as 5-tuple of the form $(Q, \Sigma, F, q_0, \delta)$ where:

- Q is a finite set of states;
- Σ is an input alphabet;
- $F \subseteq Q$ is a set of final states;
- $q_0 \in Q$ is the initial state; and
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \longrightarrow \mathcal{P}(Q)$ is the transition function

The key here is to note that the transition function for an NFA differes from that of DFA. As was discussed in class however and as can be learned from the proof of Theorem 1.39 on page 55 of the text, these two objects are equivalent to eachother. Of particular interest for this assignment is the *nature of the proof* itself — i.e., the proof the relevant theorem is *constructive* and outlines an algorithm which given an arbitrary NFA, will produce an equivalent DFA.

Specification

Define a program called `NFAConvert` which given as its input the name of file which contains a textual representation of an NFA, will output a textual representation of the equivalent DFA, obtained by applying the algorithm discussed this semester. The format of the input NFA and output DFA are equivalent and adhere to the same specification as in Project 2. As with the prior project, the same assumptions as to the nature of the NFA's input alphabet apply.