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 a 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 the set of final states;
- $q_0 \in Q$ is the initial state; and
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \to \mathcal{P}(Q)$ is the transition function

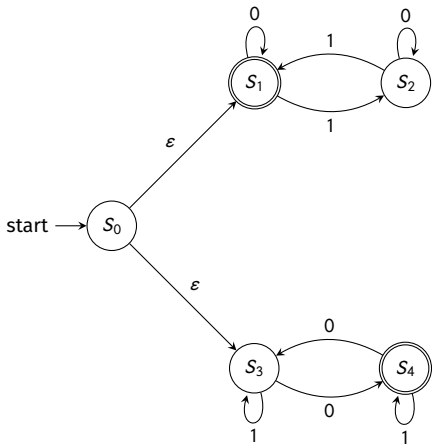
The key here is to note that the transition function differs from that of a DFA. As was discussed in class however, and as can be learned from the proof of Theorem 1.39 on page 55 of the text, these two kinds objects are equivalent. Of particular interest for this assignment is the *nature of the proof*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

Specification

Define a program `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.

NFA Input File Specification

The input format of the NFA will be almost identical to that of the DFA used in the prior project, with the exception being that empty string transitions may be represented by the empty string literal `""` or the letter `E`. Consider for example the following NFA, M_0:



M_0 will be represented in plain text form as follows:

```
% Q
S0
S1
S2
S3
S4
% Sigma
0
1
% F
S1
S4
% Q0
S0
% Delta
S0 E S1
S0 E S3
S1 0 S1
S1 1 S2
S2 0 S2
S2 1 S1
S3 0 S4
S3 1 S3
S4 0 S3
S4 1 S4
```

As before, in this representation the lines preceded by a % sign are to be treated as single line comments in Java - any content on the line after the % sign is to be ignored and is entirely optional. Also note that for this assignment you are guaranteed the following the NFA input file:

- the NFA within it is correctly specified (i.e., is both an NFA and in the correct textual format), and
- that the order of the NFA's constituent parts are as in the example (which by design corresponds with how the definition of a NFA if formulated).

DFA Output Specification

The output of your program should adhere to essentially the same format as the input, with the caveat that here you are outputting a DFA and not an NFA. Per the conversion algorithm, states of the resulting DFA are *sets of states of the input NFA*. With this in mind, for the given example, your program's output should be as follows:

```
% Q
{S0,S1,S3}
{S1,S4}
{S2,S3}
{S1,S3}
{S2,S4}
% Sigma
0
1
```

```
% F
{S0,S1,S3}
{S1,S4}
{S2,S4}
{S1,S3}
% Q0
{S0, S1, S3}
% Delta
{S0,S1,S3} 0 {S1,S4}
{S0,S1,S3} 1 {S2,S3}
{S1,S4} 0 {S1,S3}
{S1,S4} 1 {S2,S4}
{S2,S3} 0 {S2,S4}
{S2,S3} 1 {S1,S3}
{S2,S4} 0 {S2,S3}
{S2,S4} 1 {S1,S4}
{S1,S3} 0 {S1,S4}
{S1,S3} 1 {S2,S3}
```

Keep in mind that as the states of the DFA are *sets of states of the NFA* it is equally correct to have `{S1,S0,S3}` instead of `{S0,S1,S3}` listed as a state.