1. Miscellaneous questions:
   - How does NFA differ from a DFA?
   - Why would you choose one over the other?

2. What language is accepted by the following DFA?

   ![DFA Diagram]

   **Answer:** Binary strings of even length.

3. What language is accepted by the following NFA?

   ![NFA Diagram]

   **Answer:** Binary strings ending in 01 or 010

4. Find the simplest DFA you can that accepts binary strings of any length except 2.

   **Answer:**

   ![DFA Diagram]

5. Find the simplest NFA you can that accepts:
   
   (a) The language denoted by $x(yy^*|yz^*)$.

   **Answer:**
(b) The language denoted by $(01)^*0^*10^*1$.

**Answer:**

6. Can you design an NFA/DFA to implement backreferences? (Reminder: Backreferences allow you to check for a repeat of an earlier captured string; for example $(ab)*\backslash 1$ matches $abab$ but not $abba$).

**Answer:** A finite state machine must have a finite number of states, by definition. However, the string matched by the group $\backslash 1$ could be arbitrarily large, so we can’t construct an NFA for it.

Side note: It turns out that you can also use backreferences to solve 3SAT, an NP complete problem, such that an algorithm that runs in polynomial time with respect to the input string would also solve 3SAT in polynomial time. For more, see [http://perl.plover.com/NPC/NPC-3SAT.html](http://perl.plover.com/NPC/NPC-3SAT.html).

7. Convert the NFA in Exercise 4(b) to a DFA

**Answer:**
8. (Challenge):
   
   (a) Construct NFA for decimal numbers divisible by 3. (Leading zeroes are okay.)

   Answer:

   (b) Construct NFA for binary numbers divisible by 5. (Leading zeros are okay.)

   Answer:
(c) Write a regular expression for the language in each of Exercise (a) and (b).

Answer: Very complex. See http://www.andrew.cmu.edu/user/ko/pdfs/lecture-5.pdf for one algorithm to convert NFAs to regular expressions.

Further Readings

- Finite automata
  - Dragon Book (Compilers: Principles, Techniques, and Tools), Ch. 3.6-3.7
  - Regular Expression Matching Can Be Simple and Fast (http://swtch.com/~rsc/regexp/regexp1.html)
  - Visualizing Regular Expressions (http://hackingoff.com/compilers/regular-expression-to-nfa-dfa)