1. (A variation of Sipser's problem 1.32) Real computers perform exciting operations like addition. Because DFAs can only accept or reject inputs, the closest we can come to this with a DFA is to show that some language encoding correct solutions to addition problems is regular. For example, we might consider the language

   $\{\ x + y = z \mid x, y, z \in \{0, 1\}^*$ & the sum of the numbers represented by x and y in binary notation is the number represented by $z\ \}$

   Unfortunately, this very natural language is not regular. On the other hand, if we interleave the digits of x, y, and z appropriately we can describe a regular language that encodes correct binary addition. In particular, consider the language

   $\{\ w \mid w \in \{0, 1\}^*$ &
   $w = x_0\ y_0\ z_0\ x_1\ y_1\ z_1\ ...\ x_k\ y_k\ z_k$ where $x_i, y_i, z_i \in \{0, 1\}$ &
   the sum of the numbers represented by the sequences
   $x_k...x_1x_0$ and $y_k...y_1y_0$ is the number represented by $z_k...z_1z_0\ \}$

   Here we have encoded the addition problem with the digits of x, y, and z interleaved. We have also presented the digits in reverse order (the ones place comes first, then the twos, then the fours, etc.) and we have ensured that the same number of digits are used to represent all three numbers.

   Show that this language is regular.

   You may answer this question either by drawing a state diagram or giving a more formal description by specifying $(Q, \Sigma, \delta, s, F)$. In either case, provide text explaining the intuition behind your approach.

2. In class last week, I asked you to work on designing finite state automata that could recognize binary numbers that were divisible by 3. As you were all working on this task, I was asked one question several times. The question was whether the input bits would be processed from left to right or right to left (or, to put it another way whether the machine would start with the most significant digits or the least significant digits).

   In class that day, I encouraged anyone who asked to assume the digits were processed starting with the most significant digit and ending with the least significant. For this question, I want you to assume the bits arrive in the opposite order. That is, I want you to describe a finite state automaton that reads the digits of a binary number from right to left and accepts exactly those binary sequences that represent numbers that are evenly divisible by three.

   I suspect most of you will find that reversing the digits makes it a bit more difficult to see how to construct such a machine. I ask you to examine whether this really is a harder problem or not in a very simple way. In class, we saw that we could recognize binary numbers divisible by 3 with a machine containing just four states when the digits were processed left to right. While constructing your machine for the version of this problem where the digits are reversed, look for a machine with as few states as possible to see if a machine with the same number of states can still solve this seemingly more difficult problem.

3. In the text, the notion that a FSA may accept a string $s \in \Sigma$ is defined in terms of a sequence of symbols from the alphabet and an associated sequence of states. On the other hand, in class, we gave a recursive definition for a function $\hat{\delta}$ that extends δ from individual symbols of Σ to string of symbols and then stated that M accepts w if $\hat{\delta}(q_0, w) \in F$.

   In class, I claimed that these two approaches to defining acceptance by an FSA were equivalent. The key to this equivalence is the following lemma:

   > **Lemma**: Given a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ and a string $w = w_1 w_2 \ldots w_n$ where $0 \leq n$ and for $1 \leq i \leq n$, $w_i \in \Sigma$, if we define $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ recursively as:
   >
   > - $\hat{\delta}(q, \epsilon) = q$ for all $q \in Q$, and
   > - $\hat{\delta}(q, wx) = \delta(\hat{\delta}(q, w), x)$ for all $q \in Q, w \in \Sigma^*,$ and $x \in \Sigma$.
   >
   > then for any $q_i, q_f \in Q$, $\hat{\delta}(q_i, w) = q_f$ if and only if a sequence of states $r_0, r_1, \ldots, r_n \in Q$ exists such that:
   >
   > - $r_0 = q_i$,
   > - $r_n = q_f$, and
   > - $\delta(r_i, w_{i+1}) = r_{i+1},$ for all i with $0 \leq i < n$.

   Prove this lemma. Your argument should be based on induction over the length of w.

4. The word "dermatoglyphics" is one of the longest words in English in which no letter appears more than once!

   For this problem, I would like you to describe two families of finite automata that recognize the languages, R_Σ, of strings over an alphabet Σ that are not like "dermatoglyphics". That is, the machines you describe should recognize the languages of all strings over their alphabets that include at least one symbol that appears repeatedly. You should not make any assumptions about the size or contents of the alphabet over which these strings are formed. As a result, it will not be possible to simply draw diagrams of these machines. Instead, you should give formal descriptions of the machines.

   (a) Describe a family of deterministic finite automata, D_{R_Σ}, such that D_{R_Σ} recognizes the language of all strings over the alphabet Σ that include at least two copies of some symbol in Σ. In addition, provide a formula expressing the relationship between the size of the alphabet $|\Sigma|$ and the state set Q used by D_{R_Σ}. Try to design your machines so that their state sets are as small as possible.

(b) Describe a family of non-deterministic finite automata, N_{R_Σ}, such that N_{R_Σ} recognizes the language of all strings over the alphabet Σ that include at least two copies of some symbol in Σ. Again, provide a formula expressing the relationship between the size of the alphabet $|\Sigma|$ and the state set Q used by N_{R_Σ}. Try to design your machines so that their state sets are as small as possible.

For both parts of this question you should provide a formal description of the automata (i.e., specify the members of the tuple $(Q, \Sigma, \delta, s, F)$ that describe the automaton) rather than a state diagram. This formal description should be accompanied by a bit of concise prose (and possibly a diagram) that explains the intuition behind your construction.

Hint: For the deterministic version of the machine, you might want to include the power set of the alphabet Σ as part (or all) of the machine's set of states.

5. Consider the family of languages E_n where E_n contains all strings over the alphabet $\{a, b\}$ that have a b at the digit position that is n symbols from the end of the string. That is, E_0 would be the set of all strings that end with a b. E_1 would be the set of all strings that have a b in the next to last position. E_2 would consist of strings with a b followed by any two symbols at the end and so on.

   (a) Describe a deterministic finite automaton that recognizes E_n for a given n using as few states as possible. What is the relationship between the number of states in your automaton and n?

   (b) Describe a non-deterministic finite automaton that recognizes E_n again trying to minimize the number of states. What is the relationship between the number of states in your automaton and n?

   For both parts of this question you should provide a formal description of the automata (i.e., specify the members of the tuple $(Q, \Sigma, \delta, s, F)$ that describe the automaton) rather than a state diagram. This formal description should be accompanied by a bit of concise prose (and possibly a diagram) that explains the intuition behind your construction.