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

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

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

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

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. 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 an \(b\) 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 non-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 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.
3. Produce a DFA equivalent to the NFA shown below using the subset construction described on pages 55-58 of Sipser. Label the states of your DFA with the subset of the letters A, B, and C that corresponds to the subset they represent. Only show the reachable states (including the state corresponding to the empty set of NFA states), but show edges for all transitions.

4. This problem is inspired by a mistake many students make in one of the early labs used in CS 134 that results in missing every other line of a program’s correct output.

If \( L \) is a language over alphabet \( \Sigma \), let \( L_{EO} \) (EO is short for “every other”) be the set of strings obtained by deleting every other character in each string in \( L \). That is

\[
L_{EO} = \{ w_2 w_4 \ldots w_k \mid w_1 w_2 w_3 \ldots w_n \in L \text{ with } 0 \leq n, n - 1 \leq k \leq n, w_i \in \Sigma \}
\]

Similarly, let \( L_{EEO} \) (EEO is short for even every other) be the set of strings obtained by deleting every other character in each string of even length in \( L \). That is,

\[
L_{EEO} = \{ w_2 w_4 w_6 \ldots w_{2n} \mid w_1 w_2 w_3 \ldots w_{2n} \in L \text{ with } 0 \leq n, w_i \in \Sigma \}
\]

(a) The definitions of \( L_{EO} \) and \( L_{EEO} \) are similar but the operations they describe are not identical. To clarify the difference, give an example of a regular language \( R \) such that \( R_{EO} \) and \( R_{EEO} \) are different languages and give at least one example of a string that falls in one but not the other.

(b) Show that if \( R \) is a regular language then \( R_{EEO} \) is also regular. To do this, you should show how to construct a NFA \( N \) such that \( L(N) = R_{EEO} \) given a DFA \( D \) with \( L(D) = R \). You must provide a formal description of \( N \). That is you must precisely describe all the elements of a 5-tuple \( (Q_N, \Sigma, \delta_N, s_N, F_N) \) such that \( N = (Q_N, \Sigma, \delta_N, s_N, F_N) \). You do not, however have to provide a formal proof that the language accepted by \( N \) is \( R_{EEO} \). Instead, you should simply provide a clear intuitive explanation/justification of the machine \( N \) that your formal construction describes.
5. 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 \( \delta \) from individual symbols of \( \Sigma \) 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 \( \hat{\delta} : Q \times \Sigma^* \to Q \) 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 \( w \).