1. For the \( \epsilon \)-NFA of textbook Exercise 2.5.2, find an equivalent NFA (without \( \epsilon \)-moves) using the method explained in class.

2. Do Exercise 2.5.2 of the text. (The solution to Exercise 2.5.1 is on the book’s website.)

3. Do Exercise 2.5.3(a).

4. (Optional) Do Exercise 2.5.3(b,c). Part (c) is somewhat ambiguous if the string has length less than ten. Do Part (c) twice: first, where every string of length less than ten is rejected; second, where a string of length less than ten is accepted iff it contains a 1. [Hint: There is a 20-state solution to the first (with no \( \epsilon \)-moves) and an 11-state solution to the second.]

5. Do Problem 2.3 (pp. 81–82).

6. (Optional; involves a new concept) Let \( N = (Q, \Sigma, \delta, q_0, F) \) be some \( \epsilon \)-NFA. We say that a set \( S \subseteq Q \) of states is \( \epsilon \)-closed iff there are no \( \epsilon \)-transitions from inside \( S \) to outside \( S \). More formally, \( S \) is \( \epsilon \)-closed iff, for all \( q \in S \) we have \( \delta(q, \epsilon) \subseteq S \). Notice that \( Q \) itself is \( \epsilon \)-closed. Now consider the following algorithm to compute the \( \epsilon \)-closure ECLOSE\((q)\) of a state \( q \in Q \), adapted from the inductive definition on page 74:

   (a) Set \( C := \{ q \} \)
   (b) WHILE \( C \) is not \( \epsilon \)-closed, DO
      i. Pick some \( r \in C \) such that \( \delta(r, \epsilon) \not\subseteq C \)
      ii. Set \( C := C \cup \delta(r, \epsilon) \)
   (c) Return \( C \)

You are to show that ECLOSE\((q)\) (returned by this algorithm) is the least \( \epsilon \)-closed set containing \( q \). That is, show that the algorithm finishes after a finite number of steps, and the returned set \( C \) satisfies

   (a) \( C \) is \( \epsilon \)-closed and \( q \in C \), and
   (b) For any \( \epsilon \)-closed set \( D \) containing \( q \), we have \( C \subseteq D \).

Part (a) has a direct argument. Prove part (b) by showing that it is a loop invariant for the WHILE-loop, that is, it is true just when the loop starts and remains true from one iteration to the next. (This is a type of inductive argument—induction on the number of loop iterations performed so far.)
7. Let $\Sigma = \{., +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Give an $\epsilon$-NFA with input language $\Sigma$ that accepts all strings that represent numerical constants that start with an optional sign (+ or −), contain at least one decimal digit, and contain an optional decimal point. The sign may be absent, but if it appears, it can only be the first symbol in the string. The decimal point may be absent, but if it appears, it may appear anywhere in the string subject to the previous rule, and it may appear no more than once. Take advantage of nondeterminism and $\epsilon$-transitions as much as possible.

8. Do Exercise 3.1.1(b).

9. (Optional) Do Exercises 3.1.2(b,c) and 3.1.3(a,b,c).

10. Do Exercise 3.1.4(c).

11. Write a regular expression for the language of strings over $\{a, b, c\}$ where no $a$ appears after any $b$ or $c$.

12. Do Exercise 3.2.3.

13. Do Exercise 3.2.4(b).

14. Recall the DFA $D$ we constructed that accepts a binary string iff it has an odd number of 1’s:

$$
\begin{array}{c|cc}
   & 0 & 1 \\
\hline
\rightarrow & A & A \\
\ast B & B & A \\
\end{array}
$$

(a) Convert $D$ into an equivalent clean $\epsilon$-NFA using the clean-up procedure in class (add a new start state, a new final state, and some $\epsilon$-transitions).

(b) Use the state elimination method to convert $D$ to a regular expression. Eliminate state $A$ first, then $B$.

15. Same exercise as before, except make $A$ the final state (so that $D$ accepts a string iff it has an even number of 1’s).

16. (Optional) Recall the product DFA $P$ that counts an even number of zeros and an odd number of ones:

$$
\begin{array}{c|cc}
   & 0 & 1 \\
\hline
\rightarrow & EE & OE \\
OE & EE & OO \\
\ast EO & OO & EE \\
OO & EO & OE \\
\end{array}
$$

Use the state elimination method to convert $P$ to a regular expression. (To control the complexity, you may wish to define names for intermediate regexes.)