Problem 1
Exercise 1.21 (a) and (b) from Sipser. (Converting finite automata into regular expressions.)

Problem 2
Prove that for every regular expression $R$ there exists another regular expression $R'$ such that the language recognized by $R'$ is the complement of the language recognized by $R$, i.e., $L(R') = \Sigma^* - L(R)$. (You can use any of the results proved in class and/or the textbook to solve this problem.)

Problem 3
In this problem, we use the deterministic finite state transducers (FST), as defined in homework 1 and the textbook (Exercise 1.24-26). (Please refer to the solution of homework 1 posted on piazza for a formal definition of FST.)

(a) Prove that FST are closed under composition, i.e., it is possible to combine any two FSTs $T_2, T_1$ computing the functions $f: \Sigma^* \rightarrow \Gamma^*$ and $g: \Gamma^* \rightarrow \Delta^*$, into a new FST $T$ that computes the function $g \circ f: \Sigma^* \rightarrow \Delta^*$ (defined as usual as $g \circ f(x) = g(f(x))$). [Hint: given FST with sets of states $Q_1, Q_2$, define a new FST with set of states $Q_1 \times Q_2$.]

(b) Apply the transformation described in part (a) to combine the two FST $T_1, T_2$ from exercise 1.24 of the textbook. The automata should be combined in the order corresponding to first running $T_2$ on the input, and then running $T_1$ on the result produced by $T_2$. (We use this order so that the output alphabet of $T_2$ matches the input alphabet of $T_1$. To this end, you may view $T_2$ as an FST with output alphabet $\{0, 1, 2\}$ which never outputs 2.)

Problem 4
Non-deterministic Finite State Transducers (NFST, as studied in the midterm exam) are also closed under composition. See Lecture Notes (to be posted on the webpage by Wednesday) for the definition of NFST and proof of the closure under composition. In this problem we explore the use of NFST to model (a toy version of) a realistic application: the design of an error correcting code (and corresponding decoding scheme) to reliably transmit messages over a noisy channel.

Consider the following setting: two parties, a sender S and a receiver R, want to exchange messages over a communication channel that allows to transmit sequences of bits. The communication channel is not perfectly reliable: for each transmitted bit $b \in \{0, 1\}$, the channel may deliver either the correct bit $b$, or corrupt it and deliver $1 - b$ instead. However, transmission errors are not too frequent: it may be assumed that whenever an error occurs (i.e., a bit gets flipped by the channel), the next two bits will be
transmitted without errors. The sender and the receiver can use this communication channel to achieve errorless communication as follows:

- the sender transmits a redundant encoding of each input bit \( b \in \{0, 1\} \), e.g., obtained by replicating each input bit 3 times, and sending \( bbb \) instead.
- at the other end of the communication channel, the receiver performs error correction and decoding by taking the majority of each group of three received bits \( b_1 b_2 b_3 \in \{0, 1\}^3 \). E.g., when receiving 010, the receiver determines that 000 was sent, and therefore the original input bit was 1.

NFST can be used to model both the encoding and decoding algorithms, and the communication channel. The encoding and decoding algorithms will be deterministic automata, that for every input string, perform exactly one computation, and produce a single output string. (Still, we model them as NFST because the definition of FST from the textbook does not allow to produce strings of different length.) The NFST corresponding to the communication channel makes essential use of nondeterminism to model all possible error patterns that the communication channel may introduce.

You can use jflap to draw your NFST. In jflap, NFST are called “Mealy machines”. Jflap allows to run only NFST that are deterministic. So, for debugging purposes, you will be able to run only the machines corresponding to the encoding and decoding algorithms. You can still use JFLAP to draw, save and print the NFST representing the communication channel, but trying to run it within jflap will result in an error message. For each of the following parts, you should present your solution as the state transition diagram of the required NFST (Mealy machine), possibly drawn and printed using jflap.

(a) Use NFST to model the communication channel. More specifically, give an NFST \( C \) that on input a string \( w \in \{0, 1\}^n \) outputs the set of all possible strings produced by the channel when \( w \) is transmitted. As a reminder, this is the set of all strings \( u \in \{0, 1\}^n \) of length \( n \) such that, for all \( i = 1, \ldots, n-2 \), if \( w[i] \neq u[i] \) (i.e., if the \( i \)th bit gets flipped) then \( w[i+1] = u[i+1] \) and \( w[i+2] = u[i+2] \) (i.e., the following two bits are transmitted without errors. [Hint: you can solve this part using an NFST \( C \) with just 3 states.]

(b) Give an NSFT modeling the encoding algorithm. More specifically, give an NSFT \( E \) that on input \( w \in \{0, 1\}^n \) outputs a string of length \( 3n \) obtained by repeating each input bit 3 times. Your NSFT should be deterministic, i.e., on input a string \( w \) it should produce a set consisting of only one string. [Hint: you can solve this part using an NFST \( C \) with just 3 states.]

(c) Give an NFST corresponding to the decoding algorithm \( D \) that corrects the errors introduced by the channel and recover the original message \( w \). More specifically, \( D \) should be such that if you compose the three NFST together you obtain an NFST such that \( D \circ C \circ E(w) = \{w\} \) for every \( w \in \{0, 1\}^n \). Hint: For the decoder I think 5 states should be enough.

(d) Use the closure of NSFT under composition to combine the NSFT from parts (a), (b) and (c) into an NFST for \( D \circ C \circ E \) and verify that it indeed computes the identity function, i.e., on input \( w \), it outputs \( \{w\} \). Notice, you can verify that an NFST computes the identity function simply by checking that all of its transitions (or at least those reachable from the start state) are labeled either with 0/0 or with 0/1.

You can compose the NSFT in either order \( (D \circ C) \circ E \) or \( D \circ (C \circ E) \), but one of the two ordering is likely to give a simpler solution. When composing the automata, you may start from the start state, and draw only the states that are reachable from it. Unfortunately, jflap does not implement the composition of NFST, so you will have to combine \( E, C \) and \( D \) by hand.