Problem 1: Nondeterministic Finite Automata. (Average: 8.3/10) Answer these questions for the NFA $N_1$ in Sipser’s Example 1.38. The extended transition function, $\delta^*$, is as we defined it in class.

a. What is the set of possible states of $N_1$ after processing input 1?
   \textbf{Answer:} $\{q_1, q_2, q_3\}$. We start in state $q_1$. On input 1, there are two possible transitions: the self-edge, back to state $q_1$, and the edge to state $q_2$. We also need to include $\epsilon$-transitions. There is one from $q_2$ to $q_3$, but no $\epsilon$-transitions out of state $q_3$.

b. What is the set of possible states of $N_1$ after processing input 010?
   \textbf{Answer:} $\{q_1, q_3\}$

c. Is there a string $w$ such that $\delta^*(q_1, w)$ does not include $q_1$? (If so, give the string $w$. If not, argue why no such $w$ exists.)
   \textbf{Answer:} No such string exists. Since $q_1$ has a self-edge for inputs 0 and 1, for any string in $\{0, 1\}^*$ one possible state of the NFA is $q_1$ since it can always follow the self-edges and stay in state $q_1$.

d. Is there a string $w$ such that $\delta^*(q_1, w) = \{q_1, q_2, q_3, q_4\}$? (If so, give the string $w$. If not, argue why no such $w$ exists.)
   \textbf{Answer:} The shortest such string is 11. We know there is no shorter string since to reach state $q_4$, we need to process at least two 1s (for the transitions between $q_1 \rightarrow q_2$ and $q_3 \rightarrow q_4$. On input 11, we can reach all four states: $q_1 \rightarrow q_1 \rightarrow q_1; q_1 \rightarrow q_1 \rightarrow q_2 \rightarrow q_2; q_1 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3; q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4$. (We use the notation $q \rightarrow x t$ to mean we are consuming input $x \in \Sigma$ and transitioning from state $q$ to state $t$.)

\textbf{Bonus question:} (20 points for good answer) Precisely describe the language of all input strings for which $N_1$ ends in all four states.

Problem 2: Constructing NFAs. (Average: 12.8/15) For each of the following languages, draw an NFA that recognizes the language using fewest possible number of states. For all languages, assume $\Sigma = \{0, 1\}$.

a. $\{w|w$ starts and ends with different symbols. $\}$

PS2C-1
b. \( \{w | w \text{ does not contain two consecutive 0s.} \} \)

c. The language described by the regular expression \( \{0, 1\}^* 01^* \).

**Problem 3: Eliminating Epsilon.** (Average: 8.4/10) Prove that for every NFA \( N \), there is an NFA \( N' \) that recognizes the same language as \( N \) but does not use any \( \epsilon \)-transitions. (A very short, very convincing proof is possible, but you will receive some credit for a longer proof.)

**Answer:** The easy way to prove this is to use the equivalence of NFAs and DFAs (as proven in class and in Theorem 1.39). Since we can use this construction to produce a DFA \( N_D \) that
recognizes the same language as \( N \), it is trivial to construct \( N' \) from \( N_D \) (just modify the transition rules by replacing the output state with a singleton set containing that state).

**Problem 4: Language Splitting.** (Average: 6.7/10) (Based on Sipser’s 1.63a) Prove that any infinite regular language (that is, a regular language with an infinite number of strings) can be split into three infinite disjoint regular subsets.

**Answer:** Remember that a language is a set of strings. We need to prove that any infinite regular language, \( A \), can be divided into \( A_1, A_2, \) and \( A_3 \) where:

- \( A_1 \cup A_2 \cup A_3 = A \) (the union of the shares is equal to the original language)
- \( A_1, A_2, \) and \( A_3 \) each contain an infinite number of strings (for any choice of \( A \), an infinite regular language)
- \( A_1, A_2, \) and \( A_3 \) are disjoint (that is, \( A_1 \cap A_2 = A_1 \cap A_3 = A_2 \cap A_3 = \emptyset \))
- \( A_1, A_2, \) and \( A_3 \) are regular sets (they can be recognized by some DFA)

Since \( A \) is regular we know it can be recognized by some DFA. Using the same reasoning that led to the pumping lemma, there is some pumping length \( p \) (limited by the number of states in \( A \)), after which we must have gone through a cycle, returning to the same state twice. By the pumping lemma, any string \( w \in A \) with length greater than or equal to \( p \) can be subdivided into \( w = xyz \) such that \( |y| > 0 \) and all strings \( xy^iz \) for \( i \geq 0 \) are in \( A \). We can split \( A \) into three languages satisfying the needed properties by subdividing the integers into 3 disjoint sets, and making the three language shares correspond to \( xy^iz \) for each of the sets. One way to divide the integers into infinite sets is using divisibility:

- \( A_1 \) contains the strings \( xy^{3i}z \) for \( i \geq 0 \).
- \( A_2 \) contains the strings \( xy^{3i+1}z \) for \( i \geq 0 \).
- \( A_3 \) contains the strings \( xy^{3i+2}z \) for \( i \geq 0 \).

Note that the pumping lemma says that we can do this for any string in \( A \) with \( |w| \geq p \), so this means we have three, disjoint, infinite sets. We do not yet know if we cover the full language \( A \), since there may be other strings in \( A \) that use different loops or have length less than \( p \). All of those strings can be put in one of the subsets, say \( A_1 \) (which makes most sense since it includes \( i = 0 \)).

So, we have satisfied the first three properties. Finally, we need to argue that \( A_1, A_2 \) and \( A_3 \) are regular sets. We can ignore the finite strings that were added to \( A_1 \), since they are finite and must be regular, so only need to consider the strings \( xy^{3i}z \) in \( A_1 \). We can construct a DFA that recognizes \( A_1 \) by taking the states corresponding to \( y \) in the DFA that recognizes \( A \), and replacing the cycle with a new set of states (all are non-accepting) that correspond to going around the cycle three times. For example, if \( y \) is \( q_{y1} \rightarrow 0 q_{y2} \rightarrow 1 q_{y3} \rightarrow 1 q_{y1} \) we would replace the transition from the state before \( q_{y1} \) with a transition to a new state, \( q_{y1a} \).
which transitions to $q_{y2a}$ on 0, which transitions to $q_{y3a}$ on 1, which transitions to $q_{yy1a}$ on 1 (instead of returning to $q_{y1}$, and so on, repeating the states, until the end of the third repetition where state $q_{yy3a}$ will transition back to state $q_{y1}$. To construct the machine to recognize $A_2$, we use the same idea, but go through a sequence of states that consume $y$ before entering the loop. Similarly, $A_3$ can be recognizes by a machine that has a sequence of states that consume $yy$ before entering the loop.

**Problem 5: Regularity.** (Average: 12.2/20) For each part, include a convincing proof supporting your answer.

a. Is $\{w|w$ describes a valid Sudoko puzzle $\}$ a regular language? (A Sudoko puzzle is a 9x9 grid of squares, some of which contain digits. A puzzle is valid if there is some way to fill in all the empty squares such that in the final grid every row, column, and 3x3 square contains exactly the digits 1-9.)

**Answer: Yes.** Since the language is finite, we know it is regular. We know the language is finite, since all strings in the language have bounded length — 81 squares, each of which is either blank or a digit 1-9. This means an upper bound on the size of the language is $10^{81}$. In fact, there are many fewer valid puzzles, but we don’t need to know the exact number. To know the language is regular, it is enough to know that is it finite.

b. Define a new operation on languages, $D$, as:

$$D(L) = \{w|w \in \Sigma^* \text{ and } ww^R \in L\}$$

(where $w^R$ denotes the reverse of $w$). Does $D$ preserve regularity?

**Answer: Yes.** This is a tricky question, since our intuition might mislead us into thinking $D(L)$ is irregular since it seems similar to the language $ww^R$, which we know is irregular. In fact, however, we can construct a DFA $M_D$ that recognizes $D(L)$.

If we could manipulate the input, we could construct the $M_D$ machine by changing the input from $w$ to $ww^R$, and running $M$ on the new input (as shown in the top part of the figure). If it accepts, then the string $w$ is in $D(L)$. But, we can’t do that! A machine that could splice a copy of the input in reverse to the end of the input is much beyond the capabilities of a DFA, which can only consume the input one symbol at a time.

Instead, we can simulate processing $w$ and $w^R$ simultaneously. The basic idea is to combine $M$, the DFA that recognizes $L$, with $M^R$, the DFA that recognizes $L^R$ (which we know exists since the class of regular languages is closed under reversal), into a single machine that simulates both simultaneously on the same input. Recall that the $M^R$ machine will start in a state that is labeled with $F$, the set of accepting states of $M$. On the first input symbol, it goes to the state representing the set of states in $M$ which would transition to an accepting state on that input symbol. At each point in processing the input substring $z$, it is in a set of states such that running $M$ on $z$
starting from any of those states would end in an accepting state. Thus, at the end of processing the input \( w \), the machine \( M \) is in the state of \( M \) after processing \( w \), and \( M^R \) is in the state representing the set of states of \( M \) that would reach an accepting state if they processed \( w^R \). If the state of the \( M \) machine matches any of the states represented by the state of \( M^R \), then there is a path to an accepting state that goes forward the \( w \), and then backward through \( w \) (that is, through \( w^R \)), so the string \( w^R \) is in \( L \). This means the combined machine recognizes the language \( D(L) \), showing it preserves regularity. Note the similarity between this question and Problem 7. The difference is that the reverse machine processes the string in reverse in Problem 7, whereas in this question it processes the string forward so it is recognizing \( w^R \).

c. Define a new operation on languages, \( \mathcal{X} \), as:
\[
\mathcal{X}(L) = \{ w | \exists z \in \Sigma^* \text{ such that } wz \in L \}
\]
Does \( \mathcal{X} \) preserve regularity?

**Answer: Yes.** Note that \( \mathcal{X}(L) \) accepts a string if there is any sequence of inputs that could follow it leading to an accepting state. So, we can construct a DFA \( A^X \) that recognizes \( \mathcal{X}(L) \) from the DFA \( A = (Q, \Sigma, \delta, q_0, F) \) that recognizes \( L \) by adding all states for which there is any path to an accepting state to \( F \): \( A^X = (Q, \Sigma, \delta, q_0, F^X) \) where \( F^X = F \text{CanReach}(F) \) and \( \text{CanReach} : Q^* \rightarrow Q^* \) is defined recursively by:
\[
\text{CanReach}(X) = q \bigcup_{s \in Q \delta(s,a) \in X \text{ for some } a \in \Sigma} \text{CanReach}(s)
\]
Note that this looks like a circular definition since the base case is not obvious. But, it is guaranteed to be non-circular since eventually there are no more values to union.

d. Define an deterministic infinite automaton similarly to a deterministic finite automaton, except that \( Q \) is no longer required to be a finite set. Prove that DIAs can recognize non-regular languages.

**Answer:** To prove that a DIA can recognize non-regular languages, we can explain how to construct a DIA that recognizes some non-regular language. Let’s use \( L = \{0^n1^n | n \geq 0\} \) which we know is non-regular. A DIA to recognize \( L \) could be defined by using infinitely many states to count the number of 0s, and then edges back from each of those states on 1 inputs, counting the number of 1s:

\[
Q = N^+\text{zeros} \cup N^+\text{ones} \cup \text{Reject} — the states are the infinite set of the natural numbers twice and an extra “Reject” state.
\]

\( \Sigma = \{0, 1\} \)

\( q_0 = q_0^\text{zeros} \) - initially we are in a state that is counting 0s and we have seen zero 0s \( F = \{q_0^\text{zeros}\} \)

\( \delta: Q \times \Sigma Q \) is defined by:

\[
\delta(q_i^\text{zeros}, 0) = q_{i+1}^\text{zeros} \quad \text{(on a 0, go to the next numbered zero-counting state)}
\]

\[
\delta(q_i^\text{zeros}, 1) = q_{i-1}^\text{ones} \quad \text{for } i \geq 1 \quad \text{(on a 1, switch to the one-counting states, starting at } i - 1 \text{ since we just saw the first 1)}
\]

\[
\delta(q_i^\text{ones}, 1) = q_{i-1}^\text{ones} \quad \text{for } i \geq 1 \quad \text{(count the ones)}
\]

\[
\delta(q_0^\text{ones}, 1) = q_{\text{Reject}} \quad \text{(too many 1’s, permanent reject)}
\]

\[
\delta(q_i^\text{ones}, 0) = q_{\text{Reject}} \quad \text{(saw a 0 after the first 1, permanent reject)}
\]

**Problem 6: Proving Irregularity.** (Average: 11.6/15) (Based on Sipser 1.46, but different)

Prove that the following languages are not regular. You may use any technique you want, including the pumping lemma, and the closure properties we have established for regular languages in the book, class, and other problems.

a. \( \{0^n1^n|n \geq 0\} \)

**Answer:** Assume \( A = \{0^n1^n|n \geq 0\} \) is regular and use the pumping lemma to get a contradiction. Choose \( w = 0^p1^p \). From the pumping lemma, \( w = xyz \) where \( |xy| \leq p \) and \( xy^iz \in A \). Because of the length constraint on \( xy \), we know \( y \) must be within the first \( p \) symbols of \( w \), which is \( 0^p \). Hence, pumping \( y \) increases the number of 0s before the 1, but does not change the second half of the string. This produces a string, \( 0^{p+|y|}1^{2p} \), which is not in \( A \). Hence, we have a contradiction and know \( A \) is not regular.

b. \( \{0^n1^m|m = 2n\} \)

**Answer:** Assume \( A = \{0^n1^m|m = 2n\} \) is regular and use the pumping lemma to get a contradiction. Choose \( w = 0^p1^{2p} \) which is in \( A \). From the pumping lemma, \( w = xyz \)
where $|xy| \leq p$ and $xy^iz \in A$. Because of the length constraint on $xy$, we know $y$ must be within the first $p$ symbols of $w$, which is $0^p$. Hence, pumping $y$ increases the number of $0$s, but not the number of $1$s. This produces a string, $0^{p+|y|}1^2p$, which is not in $A$. Hence, we have a contradiction and know $A$ is not regular.

c. $\{w|w \in \{0,1\}^* \text{ is not a palindrome}\}$ ($w$ is a palindrome iff $w = w^R$)

**Answer:** We already proved that the language

$$\text{Palindromes} = \{w|w \in \{0,1\}^* \text{ is a palindrome}\}$$

is non-regular, and that the class of regular languages is closed under complement. Since the complement of this language is non-regular, we know the non-Palindromes language is also non-regular.

**Problem 7: Dual Finite Automata.** (Average: 5.9/10) (The idea behind this question is from Pei-Chi Wu, Fend-Jian Wang, and Kai-Ru Young, *Scanning Regular Languages by Dual Finite Automata*, ACM SIGPLAN Notices, Vol 27, No 4, April 1992. It is not necessary to read this paper to answer this question, but you are welcome to read it if you are interested.)

Consider the problem of testing whether a long string is in some regular language: given $A$, a DFA recognizing some language, and $w$ a string, determine if $w$ is in $L(A)$. The straightforward solution is to process the string left-to-right through the DFA. This requires $n$ steps where $n$ is the length of the input string, and each step involves following one transition of the $\delta$ function for the DFA. Is there a way to speed up language recognition for an arbitrary regular language if we have multiple processors that can run in parallel?

The paper mentioned above proposes a method where language scanning is done using two processors: one processes the string from left-to-right using $A$, the other processes the string from right-to-left using $A^R$, a DFA that recognizes the reverse language of $L(A)$. $A^R$ is constructed using similar methods to what we saw in class: first, an NFA is constructed by reversing the edges in $A$ and then the NFA is converted to a DFA using the subset construction. The resulting DFA, $A^R = (\mathcal{P}(Q), \Sigma, \delta', q_0', F')$ where $q_0' = F$, $F' = \{q'|q' \in \mathcal{P}(Q), q_0 \in q'\}$, and:

$$\delta'(q', a) = \{q|q \in Q \text{ such that } \delta(q, a) = q_x, q_x \in q'\}$$

When the scanners meet, we can divide $w$ into $x$ and $y$ such that $w = xy$ and $A$ has processed $x$ and $A^R$ has processed $y^R$. At this point, $A$ is in some state in $Q$, $s_A = \delta^*(q_0, x)$, and $A^R$ is in some state in $\mathcal{P}(Q)$, $s_R = \delta'^*(q_0', y^R)$.

What condition on $s_A$ and $s_R$ can be used to determine if $w$ is in $L(A)$?

**Answer:** Accept $w$ if the set labeling the state of $A^R$ includes the state of $A$ at the point where they meet.

At the point the machines meet, $s_A = \delta^*(q_0, x)$, the state reached after processing the string $x$ in the forward direction. The state of $A^R$ represents a set of states in $A$, it is labeled with
a member of $\mathcal{P}(Q)$. $A^R$ starts in a state representing all the accepting states of $A$ (that is, $F$). As it processes the string $y$ backwards, it is in a state representing the set of states in $A$ which would reach an accepting state processing $y$ forwards. That is, for all states $q_i$ represented by the state of $A^R$, $\delta^*(q_i, y) \in F$. So, if the state of $A^R$ after processing $y$ contains $s_A$, that means $\delta^*(\delta^*(q_0, x), y) \in F$ so $w = xy \in F$. 

\[ y \quad w \quad z \] 

\[ A \quad A^R \] 

\[ q_0 \quad q_f \quad \{ \text{states reaching accept on } z \} \quad F \]