1. State whether each of the following statements is true. In addition, give a short proof (2-3 lines are sufficient) if the statement is true, and give a counterexample otherwise.

(a) Fix $\Sigma = \{0, 1\}$. If $L$ is regular, then the following language must be regular:
   $$\{w \mid w \in L \text{ and } w \text{ ends in } 10101\}$$

(b) There are infinitely many Turing-recognizable languages.

(c) If two languages $L_1$ and $L_2$ over the same alphabet $\Sigma$ are decidable, then $L_1 \Delta L_2$ is decidable. (Note: the symmetric difference $S \Delta T$ of two sets $S, T$ is defined to be the set of elements belonging to $S$ or $T$ but not both.)

(a) True. The new language is the intersection of two regular languages: $L$ and $\Sigma^*10101$.

(b) True. Any finite language is Turing-recognizable, and there exists infinitely many distinct finite languages.

(c) True. Simulate the Turing machine deciders for $L_1$ and $L_2$ on a given input, and accept iff exactly one of the two machines accept.

2. (Sipser 1.37) Consider the language
   $$F = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and if } i = 1 \text{ then } j = k\}$$
   over alphabet $\Sigma = \{a, b, c\}$.

(a) Show that $F$ is not regular by constructing an infinite set of indistinguishable strings for $F$.

(b) Show that $F$ satisfies the 3 conditions of the pumping lemma. (That is, for every pumping length $p \geq 1$, and for every string $w \in F$ of length at least $p$, show that $w$ can be written in a form that can be pumped and yet remain in $F$.)

(c) Explain why (a), (b) do not contradict the pumping lemma.

(a) Take $ab^j$, for $j = 1, 2, \ldots$. For any $i < j$, we can append $c^i$ to both of $ab^j, ab^j$ to make exactly one of the two in $F$.

(b) Fix $p \geq 1$, and consider any $w = a^i b^j c^k \in F$ of length at least $p$. If $i \geq 1$, write $w = xyz$ with $x = \epsilon$ and $y = a^i$. If $i = 0$, pick $x = \epsilon$, and $y = b^{\min(j,p)}$ if $j \geq 1$, and $y = c^{\min(k,p)}$ otherwise.

(c) Not satisfying the conditions of the pumping lemma is a sufficient but not necessary condition for a language not to be regular.
3. For two languages \( L_1, L_2 \) over some alphabet \( \Sigma \), define \( \text{shuffle}(L_1, L_2) \) to be the set of strings that can be formed by interleaving a string from \( L_1 \) with a string from \( L_2 \). In this interleaving, the symbols from the two strings need not alternate at every step, but their order must be the same as in the two original strings. (So, for example, some valid interleavings of the strings ‘mickey’ and ‘mouse’ are ‘mickeymouse’, ‘mousemickey’ and ‘mimckeousy’.) Show that if \( L_1 \) and \( L_2 \) are regular, then \( \text{shuffle}(L_1, L_2) \) is regular.

We will construct an NFA \( M \) that recognizes \( \text{shuffle}(L_1, L_2) \) using the following idea: \( M \) will simulate \( M_1 \) and \( M_2 \) “in parallel”, at each step choosing non-deterministically whether to make a move in \( M_1 \) or a move in \( M_2 \) (corresponding to choosing whether the next symbol of the input comes from \( L_1 \) or from \( L_2 \)). The parallel simulation can be accomplished using the same pairing trick that we used in class to construct a DFA for \( L_1 \cap L_2 \).

The details are as follows. \( M \) will have states \( Q = Q_1 \times Q_2 \) (i.e., all pairs of states of \( M_1 \) and \( M_2 \)), initial state \( q_0 = (q_{0,1}, q_{0,2}) \), final states \( F = F_1 \times F_2 \), and alphabet \( \Sigma \). Its transition function \( \delta \) is defined by

\[ \delta([p, q], a) = \{ [\delta_1(p, a), q], [p, \delta_2(q, a)] \} \]

for each pair of states \([p, q] \in Q_1 \times Q_2\) and each symbol \( a \in \Sigma\). Thus there are exactly two moves in each configuration of \( M \), one corresponding to a move in \( M_1 \) only, the other to a move in \( M_2 \) only. \( M \) accepts iff the simulated computations of \( M_1 \) and \( M_2 \) both accept (i.e., iff the two interleaved strings belong to \( L_1 \) and \( L_2 \) respectively).

Comments: A common mistake here was to allow \( M \) to make a move in both \( M_1 \) and \( M_2 \) at each step. This is appropriate for the intersection \( L_1 \cap L_2 \), but for the shuffle each symbol in the interleaved string must be in one word or the other (not both). Another common mistake was not to use the pairing construction at all, but instead to try to directly “interleave” the two finite automata \( M_1 \) and \( M_2 \). The people who tried this got into difficulties, usually having to assume that \( M_1 \) and/or \( M_2 \) had no cycles, or generating an “interleaved” machine of infinite size. There doesn’t seem to be any way to make this approach work for this problem.

4. (Sipser 1.31) Consider a new kind of finite automaton called a \( \text{coNFA} \) (also referred to as \( \text{all-paths-NFA} \) in Sipser 1.31). A \( \text{coNFA} \) \( M \) is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) that accepts \( x \in \Sigma^* \) if every possible computation of \( M \) on \( x \) ends in a state from \( F \). Note, in contrast, that an ordinary NFA accepts a string if some computation ends in an accept state.

(a) Prove that the class of languages recognized by \( \text{coNFAs} \) is the class of regular languages.

(b) Give a polynomial time algorithm that on input a \( \text{coNFA} \) \( M \) and a string \( x \in \Sigma^* \), decides if \( M \) accepts \( x \).

(a) We can construct an equivalent DFA as in the proof for NFA, except we accept iff the subset of states (in the \( \text{coNFA} \)) we end up in is non-empty and each of the states in the subset is accepting in the \( \text{coNFA} \).

(b) As in the construction of the DFA from the \( \text{coNFA} \), we simulate the execution of \( M \) on \( x \), but simply keep track of the subset of states that we could be in (corresponding to every possible computation), without explicitly constructing the DFA. Finally, accept if the subset of states (in the \( \text{coNFA} \)) we end up in is non-empty and each of the states in the subset is accepting in the \( \text{coNFA} \), and reject otherwise.
5. Consider the language

$$ADFA_{\text{inf}} = \{ D \mid D \text{ is a DFA that recognizes a language containing infinitely many strings} \}$$

Prove that $ADFA_{\text{inf}}$ is decidable.

Using the pumping lemma, it is easy to see that a DFA $D$ with $n$ states recognizes an infinite language iff it accepts some string of length between $n$ and $2n$ (refer to section notes for a proof sketch). Therefore, to decide if $D$ recognizes an infinite language, it suffices to enumerate over all strings $w$ of length between $n$ and $2n$, and accept if $D$ accepts (at least) one of these strings, and reject otherwise.

6. (Sipser 4.17) Let $C$ be a language. Prove that $C$ is Turing-recognizable (recursively enumerable) iff a decidable language $D$ exists such that $C = \{ x \mid \exists y \text{ such that } \langle x, y \rangle \in D \}$.

Suppose $C$ is Turing-recognizable by some Turing machine $M$. Then, we can take

$$D = \{ \langle x, y \rangle \mid \text{where } x \in C \text{ and } y \text{ is the computation history of } M \text{ on input } x \}$$

(computation history is defined on page 176 of Sipser). To decide the language $D$, we can simply check on input $\langle x, y \rangle$, that $y$ is a valid computation history for $M$ on input $x$, and that the computation ends up in an accept state. If so, accept; and reject otherwise.

To see that the converse holds, given $D$, we can construct a Turing machine $M$ that recognizes $C$ as follows: on input $x$, enumerate over all possible computation histories $y$ in lexicographical order. If $D$ accepts $\langle x, y \rangle$, accept and halt.