For each of these statements, say if it is true or false. Give a proof or provide a counterexample for your answer.

1. There is a 2-state NFA for the language \((01)^*\).

   **True.** Here it is:

   ![Diagram of a 2-state NFA](image)

2. There is a 2-state DFA for the language \((01)^*\).

   **False.** This is a 3-state DFA for \((01)^*\), and all pairs of states are distinguishable \((q_0, q_1)\) and \((q_0, q_2)\) by \(\varepsilon\), \((q_1, q_2)\) by 1. So the minimal DFA for \((01)^*\) has 3 states.

3. If \(L\) is regular over \(\Sigma = \{0, 1\}\), then \(L'\) is also regular, where

   \[L' = \{x : x \in L \text{ and } x \text{ starts and ends with the same symbol}\} \]

   **True.** We can write \(L'\) as the intersection of \(L\) and the regular language \(0(0+1)^*0+1(0+1)^*1\). Since regular languages are closed under intersection, \(L'\) must be regular.

4. The language \(L = \{wxw^Rx^R : x, w \in \Sigma^*\}\) is context-free over alphabet \(\Sigma = \{a, b\}\).

   **False.** We prove it using the pumping lemma for context-free languages. Let \(n\) be the pumping length and consider the string \(0^n1^n0^n1^n\), which is in \(L\). Notice that every string in \(L\) has the same number of 0s in the first and second half, and the same for 1s. Consider any partition of this string as \(uvwxy\), where \(|vwx| < n\) and \(v\) is not empty. If \(vx\) intersects the initial block \(0^n\), after pumping up we obtain a string that has more 0s in the first half. Similarly, if \(vx\) intersects the final block \(1^n\), after pumping up we obtain a string that has more 1s in the last half.

   If neither of these cases happens, then \(v\) and \(x\) must both come from the middle part \(1^n0^n\). If \(vx\) contains more 0s than 1s, after pumping down we get a string with a shortage of 0s in the second half. If it contains more 1s than 0s, after pumping down we get a string with a shortage of 1s in the first half. If it has the same number of 0s and 1s, after pumping down we get \(0^m1^n0^m1^n\), where \(m < n\), which is not in \(L\).
5. If \( L_1 \) and \( L_2 \) are regular languages, then the following language is context-free:
\[
L = \{ xy : x \in L_1, y \in L_2, \text{ and } |x| = |y| \}.
\]

**True.** Let \( D_1 \) and \( D_2 \) be DFAs for \( L_1 \) and \( L_2 \), respectively. We design a PDA for \( L \). Intuitively, the PDA will first simulate \( D_1 \), then \( D_2 \), using the stack to make sure that both of them are simulated for the same number of steps. The PDA will nondeterministically guess the middle of the input.

More precisely, here is how the PDA works: First, push the bottom marker \( $ \) on the stack. Start reading the input and simulate the DFA \( D_1 \). Every time a symbol is read, push an \( a \) onto the stack. Every time \( D_1 \) reaches an accept state, allow a non-deterministic \( \varepsilon \)-transition to the start state of \( D_2 \). As you continue reading the input, simulate the transitions of \( D_2 \), popping an \( a \) from the stack every time. If the bottom of the stack \( $ \) is reached in an accept state of \( D_2 \), then accept, otherwise reject.

6. The grammar \( S \to aSb \mid a \) is LR(0).

**False.** Consider any input that starts with \( a \). After reading the first \( a \), the valid items are \( S \to a \circ Sb \), \( S \to a \circ \), \( S \to \bullet aSb \), and \( S \to \bullet a \). There is a shift/reduce conflict: Items 1, 3, and 4 are shift items, while item 2 is a reduce item.

7. The following language is decidable:
\[
L = \{ \langle R \rangle : \text{Regular expression } R \text{ generates only strings of even length.} \}
\]

**True.** Let \( \Sigma = \{a_1, \ldots, a_k\} \) be the alphabet for \( R \). The following Turing Machine decides \( L \): On input a regular expression \( \langle R \rangle \), first construct the regular expression \( R' = ((a_1 + \cdots + a_k)(a_1 + \cdots + a_k))^* \). Then convert both \( R + R' \) and \( R' \) into DFA, minimize both DFAs, and accept if the minimized DFAs are the same.

Since \( R' \) represents the strings of even length, \( R' \) and \( R + R' \) will represent the same language if and only if \( R \) contains only strings of even length. If this is the case, the minimal DFAs for \( R' \) and \( R + R' \) will be the same, and the TM will accept; otherwise, they must be different, so the TM will reject.

8. The following language is decidable:
\[
L = \{ \langle G_1, G_2 \rangle : \text{CFGs } G_1 \text{ and } G_2 \text{ generate the same strings.} \}
\]

**False.** We reduce from \( \text{ALL}_{CFG} \). Assume \( L \) is decidable and let \( M \) be a decider for it. We use \( M \) to decide \( \text{ALL}_{CFG} \) as follows: On input \( G \) over alphabet \( \Sigma = \{a_1, \ldots, a_k\} \), construct the following CFG \( G' : S' \to a_1S' \mid a_kS' \mid \varepsilon \). Run \( M \) on input \( \langle G, G' \rangle \) and return the answer.

By construction, \( G' \) generates all possible strings over \( \Sigma \). So if \( G \) generates all strings, \( G \) and \( G' \) will generate the same strings and \( M \) will accept. Otherwise, \( G \) will fail to generate some string, so \( G' \) generates more strings than \( G \) and \( M \) will reject. It follows that \( M \) accepts \( \text{ALL}_{CFG} \), contradicting the undecidability of \( \text{ALL}_{CFG} \). Therefore \( L \) is undecidable.

9. The following language is decidable:
\[
L = \{ \langle M \rangle : \text{TM } M \text{ does not accept any inputs.} \}
\]
False. The complement $\overline{L}$ of $L$ is the language $SOME_{TM}$ of those TMs that accept some input. We showed in class that $SOME_{TM}$ was not decidable. Therefore $L$ cannot be decidable either, because the complement of every decidable language is decidable.

10. Assume $P \neq NP$. The following language is NP-complete:

$$L = \{ \langle \phi, \psi \rangle : \text{Boolean formulas } \phi \text{ and } \psi \text{ share a common satisfying assignment.} \}$$

True. First, $L$ is in NP. The following is a description of a polynomial-time verifier TM for $L$: On input $\langle \phi, \psi, a \rangle$, where $\phi$ and $\psi$ are formulas, and $a$ is a candidate assignment, check that $a$ satisfies both $\phi$ and $\psi$. Because checking if a formula satisfies an assignment can be done in polynomial time, this is a polynomial-time verifier for $L$, and so $L$ is in NP.

We show that $L$ is NP-hard. We show that $SAT$ reduces to $L$. Consider the following transformation that takes an instance $\langle \phi \rangle$ of $SAT$ and produces an instance $\langle \phi', \psi \rangle$ of $L$: Set $\phi' = \phi$ and construct $\psi$ to be a formula over the same variables as $\phi$ that accepts all strings. For example, you can set $\psi = true$ (or if it makes you more comfortable, set $\psi = (x_1 \lor \overline{x}_1) \land \cdots \land (x_n \lor \overline{x}_n)$).

This transformation runs in polynomial-time, as the formula $\psi$ can be constructed very quickly (in linear time). Moreover, if $\psi$ is satisfiable, then $\phi$ and $\psi$ share a satisfying assignment, namely any satisfying assignment of $\phi$. If $\psi$ is not satisfiable, then $\phi$ and $\psi$ cannot share any satisfying assignments.

We showed that $SAT$ reduces to $L$ in polynomial time, therefore $L$ is NP-complete.