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

1. The following language is regular over alphabet $\Sigma = \{0, 1, 2\}$:

   \[ L = \{x: x \text{ contains at least one 0, at least one 1, and at least one 2}\}. \]

   True. The following regular expression represents $L$:

   \[
   \Sigma^*0\Sigma^*1\Sigma^*2\Sigma^* + \Sigma^*0\Sigma^*2\Sigma^*1\Sigma^* + \Sigma^*1\Sigma^*0\Sigma^*2\Sigma^* \\
   + \Sigma^*1\Sigma^*2\Sigma^*0\Sigma^* + \Sigma^*2\Sigma^*0\Sigma^*1\Sigma^* + \Sigma^*2\Sigma^*1\Sigma^*0\Sigma^*.
   \]

2. For every regular $L$, the minimal DFA for $L^*$ has fewer states than the minimal DFA for $L^*$.

   False. For example let $\Sigma = \{0, 1\}$ and $L = \{\varepsilon, 1\}$. Then $L^* = 1^*$ has a two-state DFA:

   However, the following DFA, which is minimal for $L$, has three states:

   This DFA is minimal because all pairs of states are distinguishable: $(q_0, q_2)$ and $(q_1, q_2)$ by $\varepsilon$, $(q_0, q_1)$ by $1$.

3. If $L$ is regular over $\Sigma = \{0, 1\}$, then $L' = \{uxv: x \in L, u, v \in \Sigma^*\}$ is also regular.

   True. If $R$ is a regular expression for $L$, then $(0 + 1)^*R(0 + 1)^*$ is a regular expression for $L'$.

4. The CFG $S \rightarrow aSb | b$ is $LR(0)$.

   True. The LR(0) DFA for $L$ has no conflicts:
5. The CFG $S \to 00S1S \mid 0S1S0 \mid \varepsilon$ describes a **regular** language.

**False.** Notice that all strings of the form $0^{2n}1^n$ are in the CFG, and every string it contains has twice as many 0s as 1s. Suppose that the language of the CFG was regular and let $n$ be its pumping length. Let $z = 0^{2n}1^n$, which is in the language. By the pumping lemma, there is a way to write $z = uvw$ where $|uv| < n$ and $v \neq \varepsilon$ such that $uv^2w$ is in the language. However, the string $uv^2w$ has more than twice as many 0s as 1s, so it is not in the language. Therefore the language cannot be regular.

6. The language $L = \{0^i1^j0^k1^l : i, j, k \geq 0\}$ is **context-free**.

**True.** The following CFG generates $L$:

$$
S \rightarrow ZA \\
A \rightarrow 1A1 \mid Z \\
Z \rightarrow 0Z \mid \varepsilon.
$$

7. The language $L = \{\langle M \rangle : TM M$ accepts some input of length 1 $\}$ is **decidable**.

**False.** Suppose $L$ is decidable and let $D$ be a decider for it. We use $D$ to decide $A_{TM}$. To do so, we need to convert a pair $\langle M, w \rangle$ into a TM $M'$ such that $M'$ accepts some input of length 1 if and only if $M$ accepts $w$. This can be done by asking $M'$ to ignore its input and simulate $M$ on input $w$. This gives the following TM for $A_{TM}$:

$E :=$

**On input** $\langle M, w \rangle$,

**Construct the following TM** $M'$:

- **On any input**, simulate $M$ on $w$ and return its answer.
- **Run** $D$ on input $\langle M' \rangle$ and return its answer.

By construction, $E$ accepts $\langle M, w \rangle$ if and only if $D$ accepts $M'$, that is $M'$ accepts some input of length 1, which happens if and only if $M$ accepts $w$. Therefore $E$ decides $A_{TM}$, which is impossible, so $L$ must be undecidable.

8. The language: $L = \{\langle G \rangle : CFG G$ generates all strings except $\varepsilon\}$ is **decidable**.

(Assume the alphabet of $G$ is $\Sigma = \{0, 1\}$.)

**False.** Suppose $L$ is decidable by a TM $D$. We show how to use $D$ to decide $ALL_{CFG}$. To do so, we want to convert a CFG $G$ into a CFG $G'$ so that $G$ accepts all strings if and only if $G'$ accepts all strings except $\varepsilon$. To do so, we make the CFG $G'$ include all rules of $G$, plus the rules $S' \rightarrow 0S$ and $S' \rightarrow 1S$, where $S$ is the start variable of $G$, and $S'$ is a new variable which will be the start variable of $G'$. If $G$ generates all strings, then $G'$ will generate all strings except for $\varepsilon$. Conversely, if $G$ fails to generate some string $w$, then $G'$ will fail to generate the strings $0w$ and $1w$.

More formally, the following TM decides $ALL_{CFG}$:

$E :=$

**On input** $\langle G \rangle$,

- **Construct the CFG** $G'$ as described above.
- **Run** $D$ on input $\langle G' \rangle$ and return its answer.
Then $E$ accepts $G$ if and only if $D$ accepts $G'$, that is if and only if $G'$ accepts all strings but $\varepsilon$, which happens if and only if $G$ accepts all strings. So $E$ decides $ALL_{CFG}$, which is impossible. It must be that $L$ is undecidable.

9. The language $L = \{ \langle M \rangle : \text{TM } M \text{ accepts some input of the form } xx^R \}$ is **recognizable**.

**True.** To recognize $L$, we want to simulate $M$ on all inputs of the form $xx^R$ and see if it ever accepts. The trouble is that there are infinitely many some inputs, and even when $x = \varepsilon$, $M$ might go on forever. So we do a “fair simulation” in the sense that for $k$ ranging from 0 to infinity, we simulate $M$ on all inputs of the form $xx^R$ of length at most $2k$ for at most $k$ steps. This way we can make sure that even if $M$ loops on some inputs, as long as it accepts $xx^R$, we will be eventually given enough time to simulate $M$ on this input until it accepts.

Formalizing this discussion, the following TM recognizes $L$:

$E :=$

On input $\langle M \rangle$,

For $k := 1$ to infinity:

For every string $x$ of length at most $k$:

Simulate $M$ on input $xx^R$ for at most $k$ steps.

If it accepts, accept.

If $M$ accepts some input $xx^R$ of length $2t$ in at most $s$ steps, then by the time $k = \max\{t, m\}$, $E$ will have accepted $M$. If $M$ does not accept any such input, $E$ will loop forever on input $M$. So $E$ recognizes $L$.

10. The following language is NP-**complete** (i.e., it is in NP and it is NP-hard):

$L = \{ \langle G, k \rangle : G \text{ is a graph that has two or more cliques of size } k \}$

**True.** To show $L$ is in NP, we describe a solution for $L$ and a polynomial-time verifier that checks the solution is correct. A solution for $L$ is a pair of sets of vertices $S, T$. The verifier checks that $S$ and $T$ both have size $k$, they are not the same, and $S$ and $T$ are both cliques (i.e. for every pair of vertices $u, v \in S$ and $u, v \in T$, $\{u, v\}$ is an edge in $G$). We can do so by iterating over all $O(k^2)$ pairs of vertices in $S$ and $T$, so the verifier certainly runs in polynomial time.

We now give a polynomial-time reduction from $CLIQUE$ to $L$. Given an instance $\langle G, k \rangle$ of clique, we produce the instance $\langle G', k \rangle$ of $L$, where $G'$ is a graph consisting of two disjoint copies of $G$. Clearly this reduction can be implemented in polynomial time (as it merely requires making a copy of the input). Moreover, $G'$ has two or more cliques of size $k$ if and only if $G$ has a clique of size $k$, so $\langle G, k \rangle \in CLIQUE$ if and only if $\langle G', k \rangle \in L$. Since $CLIQUE$ is NP-complete, $L$ must be NP-complete too.