1. In the following, whenever VALCOMPS is mentioned, we mean the version where every other configuration is reversed.

   (a) This is undecidable. The complement of VALCOMPS(M) for a Turing machine M is a CFL. If it were decidable whether \( L = R \), we can then set \( L = \overline{\text{VALCOMPS}(M)} \), and \( R = \Sigma^* \), and decide whether \( L(M) = \emptyset \).

   (b) This is decidable. Note that \( L \subseteq R \iff L \cap \overline{R} = \emptyset \). Since \( L \cap \overline{R} \) is still a CFL, we can decide whether it’s empty. We just check whether it contains any string of length \( \leq 2n \), where \( n \) is the pumping constant.

   (c) Decidable. \( D = R \iff (D \cap \overline{R} = \emptyset) \land (\overline{D} \cap R = \emptyset) \). Again, \( D \cap \overline{R} \) is still a CFL, and so is \( \overline{D} \cap R \). Note that the latter assertion depends on the fact that \( \overline{D} \) is also a DCFL (cf Homework 9). In part (a), this argument breaks down since \( \overline{L} \) might not be a CFL.

   (d) Undecidable. This is because VALCOMPS(M) is the intersection of two DCFLs, so if we could decide whether \( D \cap D' = \emptyset \), then we can decide whether \( L(M) = \emptyset \).
Problem 2. Show the following theorem: Let $P$ be a property of languages. Define $L_P = \{M \mid L(M) \text{ satisfies } P\}$. $P$ is said to have containment property if for all languages $L$ in $P$ and for all r.e. $L' \supseteq L$, $L'$ is also in $P$. Show that if $P$ violates the containment property, then $L_P$ is not r.e. Hint. Note that $L_r$ and $L_{nr}$ both violate the containment property.

$L_r = \{M \mid L(M) \text{ is recursive}\}$.

$L_{nr} = \{M \mid L(M) \text{ is non-recursive}\}$.

Solution. Since the property $P$ does not satisfy containment, there must be a machine $M_1$ and a machine $M_2$ such that $M_1 \in L_P$, $M_2 \notin L_P$, and $L(M_2) \supseteq L(M_1)$. We do a reduction from $\overline{L_u} = \{<M, w> \mid M \text{ does not accept } w\}$. Given a pair $<M, w>$, we construct a machine $M'$ as follows. $M'$ has two tapes. On input $x$, $M'$ does the following:

- On one tape it simulates the machine $M_1$ on input $x$ and accepts if $M_1$ accepts $x$.
- On the other tape, $M'$ first starts simulating the machine $M$ on $w$. If the machine $M$ accepts $w$, then $M'$ start simulating the machine $M_2$ on the input $x$. It accepts if this computation accepts $x$.

The computations on the two strings are dovetailed, and the result is a OR of the two computations, i.e. $x$ is accepted if either of the tapes say it should be accepted.

So if the machine $M$ does not accept $w$, then the language of $M'$ is the same as the language accepted by the computation done on the second tape, i.e.

$M \text{ does not accept } w \Rightarrow L(M') = L(M_1)$.

If the machine $M$ does accept $w$, then

$M \text{ accepts } w \Rightarrow L(M') = L(M_1) \cup L(M_2) = L(M_2)$.

Thus, $M' \in L_P$ iff $M$ does not accept $w$. Hence this language $L_P$ is not r.e.
Problem 3. We can show that \( \{ M \mid L(M) \cap L_u \neq \emptyset \} \) is r.e. by constructing a Turing machine \( M' \) that accepts this language. Given the description of a Turing machine \( M \), \( M' \) does the following:

Simulate an enumeration machine for \( L_u \) (which we know to be r.e.). Whenever the simulation enumerates a pair \( (M'', w'') \), nondeterministically decide whether this pair is in the language \( L(M) \). If we decide it is, we can verify this by simulating \( M \) on \( (M'', w'') \), and if it accepts, we know that \( L(M) \cap L_u \) is nonempty.

We can show that \( \{ M \mid L(M) - L_u \neq \emptyset \} \) is not r.e. by a reduction from the set \( \overline{L_u} \). Note that \( \overline{L_u} \) is known to be not r.e. since \( L_u \) is r.e. but not recursive. Given a pair \( (M, w) \), we construct a Turing machine \( M' \) that accepts exactly the language \( \{(M, w)\} \). Then \( M' \in \{ M \mid L(M) - L_u \neq \emptyset \} \) iff \((M, w) \notin L_u\) iff \((M, w) \in \overline{L_u}\).
Problem 4. We have an oracle which can test if $L(M)$ is regular for any Turing machine $M$. Use the oracle to decide if a given machine has a finite language.

Given $M$ as input, we construct a machine $M'$ which accepts $a^{\left|w\right|}b^{\left|w\right|}$ for all $w \in L(M)$. If $M'$ accepts an infinite number of strings then its language is not regular, and a finite language is regular. Hence, the oracle can use $L(M')$ regular $\iff L(M)$ finite.