1. Given that C is Turing-recognizable, we know that there exists some Turing machine $T_C = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ such that $C = L(T_C)$. The machine T_C may, however, not halt on all inputs. For each $w \in C$, the definition of acceptance by a Turing machine requires that there be some sequence of configurations, $c_0, c_1, \ldots, c_n$ of the machine T_C such that $c_0 = (\epsilon, q_0, w)$, $c_n = (u, q_{\text{accept}}, v)$ for some $u, v \in \Gamma^*$, and each c_i yields c_{i+1} given the transition function δ. Given that we can assume T_C is deterministic, for each $w \in C$, there is a unique sequence of configurations $c_0, c_1, \ldots, c_n$ satisfying the requirements for acceptance of w by T_C. For $w \in L(T_C)$, let us define $ACC(w) = (c_0, c_1, \ldots, c_n)$. That is, we let $ACC(w)$ denote the sequence of configurations through which T_C would pass from its initial state with w as input to an accepting configuration.

   Let D be the language

   $$\{w\#\langle ACC(w)\rangle \mid w \in C\}$$

   where $\langle ACC(w)\rangle$ represents a string encoding the configurations in $ACC(w)$ using an alphabet Σ_D that includes sufficient symbols to encode the states of C and other features of the sequence of configurations. That is, D is the language of encoded pairs consisting of a string that belongs to C followed by an encoding of the complete sequence of configurations T_C would pass through in the process of accepting w. We claim that D is decidable.

   To decide this language, a Turing machine would need to:

   - Verify that the input satisfied the requirements of whatever scheme we choose for encoding inputs and configurations,
   - Verify that c_0 had the correct form (i.e., (q_0, ϵ, w)).
   - Verify that each c_i yielded c_{i+1}, the configuration that followed it on the input tape.
   - Verify that the state in c_n was q_{accept}.

   All of these tasks would be easy for a Turing machine to complete. Thus, D is decidable.

   The preceding construction defines D in a way that clearly corresponds to the "witness" idea suggested as a hint in the problem statement. An alternate, and much simpler solution is to let $D = \{x\#b(n) \mid b(n)$ is the representation of some number n in binary and C accepts x in n or few steps $\}$. A Turing machine T_D can decide D by simulating T_C for up to n steps, accepting if T_C accepts during these n steps and rejecting otherwise. Since T_D never runs more than n steps, it decides the language. Clearly if $x\#b(n) \in D$, $x \in C$. At the same time, if $x \in C$, then for some n, $x\#b(n) \in D$.

   To show the "if" direction, suppose that we have a decidable language D such that

   $$C = \{x \mid \text{there exists } y \text{ such that } x\#y \in D\ \}.$$

Given a Turing machine T_D that decides D, we can build a Turing machine T_C to recognize C as follows. On input x, T_C will begin generating all strings $y \in \Sigma^*$ starting with the shortest strings and continuing through strings of increasing length. For each y it generates, it will simulate T_D on $x\#y$. If T_D accepts some $x\#y$, then T_C will accept x. If T_D rejects $x\#y$, T_C will just move on to the next y. Since T_D halts on all inputs, given enough time T_C will consider any given value of y. Therefore, it the input x belongs to C, T_C will eventually accept the input. If not, the machine will never halt.

2.

Given $C = L(G)$ where $G = (V, \Sigma, s, R)$ is an unrestricted grammar, consider the language

$$D = \{w_n\#w_0\#w_1\#w_2\#\ldots\#w_n\# \mid w_0 = s, \text{ for all } i < n, w_i \Longrightarrow w_{i+1}\}$$

That is D is the language of consisting of strings of the form w followed by the steps of a derivation of w using the grammar G where each step in the derivation is separated from the next by a #. This language clearly satisfies the constraint that

$$C = \{x \,|\, \text{there exists } y \text{ such that } x\#y \in D\ \}.$$

So, all we need to conclude that $L(G)$ is Turing-recognizable is to verify that D is Turing-decidable.

We argue that it is clear that we can build a non-deterministic Turing-machine M that decides D. The machine first verifies that its input has the correct form (w contains only terminal symbols, the second substring is the start symbol of G, no-substring other than the first and last are empty). This is a regular language, so we know that it can be decided by a Turing machine. The machine next verifies that the first and last strings between #s in its input are identical. We have seen that a Turing machine can perform such a matching. Next, it verifies that each pair of strings after the first encodes a valid direct derivation relative to the grammar. For each pair, $\#w_i\#w_{i+1}\#$ this involves first making sure that the two strings start with some (possibly empty) identical prefix. At some point in this matching process, the Turing machine guesses that it has reached the point where the substitution that changes w_i into w_{i+1} occurs. It also guesses which rule in the grammar was used. To do this, the states of the TM will be designed to include chains of states that can match the left hand side of each rule of the grammar in w_i and then verify that the right hand side of the rule forms the next substring of w_{i+1}. If these states verify that w_i and w_{i+1} include substrings matching the strings in a rule of the grammar following identical prefixes, the TM next verifies that the suffixes of w_i and w_{i+1} up to the next #s also match.

Given this process, the Turing machine will clearly recognize D. In addition, since any computational path the machine guesses it should follow ends either when all derivation steps have been verified or as soon as the machine either encounters a steps that cannot be verified or guesses the wrong rule to verify, all possible computation paths will be finite. Therefore, this Turing machine will decide D. This, together with the result of problem 3 allows us to conclude that the language of any unrestricted grammar is Turing-recognizable.

3.

The language is: $STEPS_{TM}^{ALL} = \{\langle M \rangle \mid M$ is a TM description & for all w, M runs for at least $|\langle M \rangle|$ steps $\}$

The fact that this language is decidable hinges on the observation that a machine's behavior in its first N steps can only depend on the first N symbols in its input. That is, if M runs for at least N steps on input w with $w = pw'$ where $|p| = N$, then M must run for at least N steps on any input that begins with the prefix p. This must be true because the machine's head cannot even reach the first character after p in N steps. Similarly, if M runs for less than N steps on $w = pw'$, then M must run for less than N steps on any input that begins with p.

As a result, if we simulate M on all inputs of length less than or equal to $|\langle M \rangle|$ for at most $|\langle M \rangle|$ steps, we can accept if M has not accepted or rejected any prefix p up to length $|\langle M \rangle|$ before we finish simulating and reject immediately if M terminates its computation in less than $|\langle M \rangle|$ steps on any such p.

4. This language is not decidable.

5.

The language in question is

$$WRITE_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and on input } w \\ M \text{ changes its tape while processing } w\}$$

$WRITE_{TM}$ is decidable. If we simulate $M = (\Sigma, \Gamma, Q, \delta, q_o, q_{accept}, q_{reject})$ on w and accept as soon as M writes any symbol on its tape, we can keep track of all of the configurations the simulation reaches and halt and reject if M ever loops back to a configuration we have previously encountered or if M every moves more than $|Q|$ symbols to the right of the end of w. Basically, if M loops it must either do so while staying within w and the first $|Q|$ blanks to the right of w or it must loop in a way that moves farther and farther into the blank symbols to the right of w. This is true because if M makes more than $|Q|$ steps into the blanks without returning to a non-blank symbol, then after at most $|Q|$ transitions (and at most $|Q|$ right moves) it must have entered some state $q \in Q$ at least twice. As long as it stays in the blank section of the tape it will continue to loop back to q, moving infinitely to the right. Therefore, if M moves more than $|Q|$ steps into the blanks we can terminate the simulation and reject. On the other hand, if it stays within the first $|w| + |Q|$ symbols on the tape, the tape head can only assume one of $|w| + |Q|$ positions and $|Q|$ states. Therefore, if it has not written on the tape or halted in $(|Q| \times (|w| + |Q|))$ the simulator can conclude it is in a loop and reject its input.

6.

This language is not decidable.

7.

The language in question is

$$R_{DFA} = \{\langle D\rangle \mid D \text{ is a DFA and } w \in L(D) \Rightarrow w^R \in L(D)\}$$

Given $\langle D\rangle$, let $L = L(D)$ and $L^R = \{w \mid w^R \in L\}$. We know that L and L^R are both regular. We showed that the reverse of a regular language is regular in class using a proof in which we converted a DFA for L into an NFA for L^R by reversing its transitions, making its start state final and adding a new start state with transitions to all of the original machine's final states. Thus, given a DFA for L, we can produce an NFA for L^R and then use the subset construction to produce a DFA for L^R. Furthermore, if $w \in L(D) \Rightarrow w^R \in L(D)$, then $L^R \subseteq L$. In this case, $L^R \cap \overline{L} = \emptyset$. The proofs that regular languages are closed

under intersection and complement provide constructions we can follow to convert DFAs for the original languages into DFA for their complement or intersection. Therefore, we can construct a DFA for $L^R \cap \overline{L}$ and emptiness of the language of a DFA is decidable. Therefore, R_{DFA} is decidable.