9.3.4

a) This is RE, you can see this when you read why \( L_{\text{ne}} \) (non-emptiness) is RE. Given \( M \), construct non-deterministic \( M' \) to run two copies of \( M \) on guessed inputs \( w \) and \( w' \), accept when both are accepted.

b) We can reduce emptiness problem to infinity problem in the following way:

Given \( M \), construct \( M' \) such that \( M' \) on input \( i \), constructs \( x_1 x_2 \ldots x_i \) and runs \( M \) on these inputs for \( i \) steps each. If \( M \) does not accept any of these, then \( M' \) accepts \( i \). If \( L(M) \) is empty, then obviously \( L(M') \) is infinite, since \( M \) will never accept anything so \( M' \) will accept all \( i \). If \( L(M) \) is non-empty, then for some \( j \), \( M \) will accept \( x_j \) in \( S_j \) steps. Since we construct the same \( x_i \)'s at the \( i+1 \)th step and only \( x_{i+1} \) is new, for every \( i > \max(j, S_j) \), \( M' \) will reject \( i \), making \( L(M') \) finite.

Therefore, if deciding \( L(M') \)'s infinity property was RE, then we could decide \( L(M) \)'s emptiness RE. So, infinity is not RE.

Here is the proof on the web page of the book for 9.3.4d:

Exercise 9.3.4(d)
We shall reduce the problem \( L_e \) (does a TM accept the empty language?) to the question at hand: does a TM accept a language that is its own reverse? Given a TM \( M \), we shall construct a nondeterministic TM \( M' \), which accepts either the empty language (which is its own reverse), or the language \{10\} (which is not its own reverse). We shall make sure that if \( L(M) \) is empty, then \( L(M') \) is its own reverse (the empty language, in particular), and if \( L(M) \) is not empty, then \( L(M') \) is not its own reverse. \( M' \) works as follows:

1. First, check that its input is 10, and reject if not.
2. Guess an input \( w \) for \( M \).
3. Simulate \( M \) on \( w \). If \( M \) accepts, then \( M' \) accepts its own input, 10.

Thus, if \( L(M) \) is nonempty, \( M' \) will guess some string \( M \) accepts and therefore accept 10. If \( L(M) \) is empty, then all guesses by \( M' \) fail to lead to acceptance by \( M \), so \( M' \) never accepts 10 or any other string.

The proof for c is similar:

c) Instead of \{10\}, use a non-CFL (e.g. \{a^n b^n c^n \mid n >= 1\}) as part of the \( L \) of \( M' \). Note that empty language is in CFL. The construction follows in the exact same way, we accept a non-CFL if \( M \) accepts \( w \). Therefore, \( M' \) accepts non-CFL if \( L(M) \) != empty and it accepts CFL if \( L(M) = \text{empty} \). Deciding CFL is not RE.

Common Problems: Some of you did the reductions in the wrong direction. This is a very common mistake in this type of problems. Keep this in mind; you have two problems \( P_1 \) and \( P_2 \), and you know \( P_1 \) is hard. You want to reduce \( P_1 \) to \( P_2 \). This means if we have an
algorithm for solving $P_2$ easily (in NP context this means polynomial) then you could solve $P_1$ easily, which is a contradiction since you know $P_1$ is hard. So what you want to do is express $P_1$ in terms of $P_2$ somehow. In this way, the solution to $P_2$ would map to a solution to $P_1$. This is what confuses people the most! This mapping goes from $P_2$ to $P_1$ whereas the reduction is from $P_1$ to $P_2$. So be careful on the final, and good luck!

Another issue that people confused was the difference between RE and recursive. Recursive means decidable, all recursive problems are RE. However not all RE are recursive. In RE, it is okay for a TM to run forever if it will never accept an input $x$. If you look at the proof that $L_{ne}$ is RE, then you would see that if $L(M)$ is empty, the machine $M'$ will run forever. In fact, it can never stop and reject. That is why Halting Problem is undecidable but it is RE. It is very similar to $L_{ne}$. Read the book for more details.