Question 9.3.7

Since we are trying to prove the following to be non-RE, we reduce a known problem which is non-RE to the given problems.

a) http://www-db.stanford.edu/~ullman/ialcsols/sol9.html#sol93

We reduce the complement of $L_u$ to this problem, which is the complement of the halting problem for Turing Machines. The crux of the argument is that we can convert any TM $M$ into another TM $M'$, such that $M'$ halts on input $w$ if and only if $M$ accepts $w$. The construction of $M'$ from $M$ is as follows:

1. Make sure that $M'$ does not halt unless $M$ accepts. Thus, add to the states of $M$ a new state $p$, in which $M'$ runs right, forever; i.e., $\delta(p,X) = (p,X,R)$ for all tape symbols $X$. If $M$ would halt without accepting, say $\delta(q,Y)$ is undefined for some nonaccepting state $q$, then in $M'$, make $\delta(q,Y) = (p,Y,R)$; i.e., enter the right-moving state and make sure $M'$ does not halt.
2. If $M$ accepts, then $M'$ must halt. Thus, if $q$ is an accepting state of $M$, then in $M'$, $\delta(q,X)$ is made undefined for all tape symbols $X$.
3. Otherwise, the moves of $M'$ are the same as those of $M$.

The above construction reduces the complement of $L_u$ to the complement of the halting problem. That is, if $M$ accepts $w$, then $M'$ halts on $w$, and if not, then not. Since the complement of $L_u$ is non-RE, so is the complement of the halting problem.

b) Solution is similar in spirit to 9.3.4(d), proving $L(M) = L(M)^R$ is non-RE

We reduce the problem $L_e$ (does a TM accept the empty language?) to the problem $L(M_1) \cap L(M_2) = \emptyset$. Given any TM $M$, we convert it into 2 other TMs $M_1$ & $M_2$ (with $M_1$ being nondeterministic), such that $L(M_1) \cap L(M_2) = \emptyset$ if and only if $M$ accepts the empty language. If $M$ accepts the empty language, we make $M_1$ & $M_2$ accept the empty language as well (giving $L(M_1) \cap L(M_2) = \emptyset$). Else if $L(M) \neq \emptyset$, then we make $L(M_1) = L(M_2) = \{01\}$ (and therefore $L(M_1) \cap L(M_2) = L(M) \neq \emptyset$). The construction of $M_1$ & $M_2$ from $M$ is as follows:

1. Check that its input is 01, and reject if not.
2. Have $M_1$ make a guess, $w$ as an input for $M$.
3. Simulate $M$ on $w$.
4. If $M$ accepts $w$, then both $M_1$ & $M_2$ accepts 01.

Therefore, if $L(M)$ is not empty, both $M_1$ & $M_2$ will accept the string 01 and their intersection will not be empty. If $L(M)$ is empty, then all guesses by $M_1$ & $M_2$ fail to lead to acceptance by $M$, so $M_1$ & $M_2$ will not accept any strings, and therefore, their intersection will be empty. Since the problem $L_e$ is non-RE, then so is the problem $L(M_1) \cap L(M_2) = \emptyset$.

c)
We shall again reduce the problem \( L_e \) to the given problem, \( L(M_1) = L(M_2)L(M_3) \). Given any TM \( M \), we convert it into 3 other TMs \( M_1, M_2 \) & \( M_3 \) (with \( M_1 \) being nondeterministic), such that \( L(M_1) = L(M_2)L(M_3) \) if and only if \( M \) accepts the empty language. Using a similar argument as the above, If \( M \) accepts the empty language, we make \( M_1, M_2 \) & \( M_3 \) accept the empty language as well (giving \( L(M_1) = L(M_2)L(M_3) \)). However, if \( L(M) \neq \emptyset \), then we make \( L(M_1) = \{01\} \), \( L(M_2) = \{01\} \) and \( L(M_3) = \{01\} \) (hence \( L(M_1) \neq L(M_2)L(M_3) \)). The construction of \( M_1, M_2 \) & \( M_3 \) from \( M \) is as follows:

1. Check that its input is \( 01 \), and reject if not.
2. Have \( M_1 \) make a guess, \( w \) as an input for \( M \).
3. Simulate \( M \) on \( w \).
4. If \( M \) accepts \( w \), then \( M_1, M_2 \) & \( M_3 \) accepts \( 01 \).

Suppose \( M \) accepts the string \( w \). Then \( L(M_1) = L(M_2) = L(M_3) = L(M) \) and \( 01 \neq 0101 \). However, if \( M \) accepts the empty language, then \( L(M_1) = L(M_2) = L(M_3) = \emptyset \), and the concatenation of empty languages gives the empty language. Since the problem \( L_e \) is non-RE, then so is the problem \( L(M_1) = L(M_2)L(M_3) \).
For the classes of recursive sets and for the class of r.e. sets and for each of the operations union, concatenation, star, homomorphism, and inverse homomorphism either prove closure or non-closure.

Recursive sets are closed under union: simply run their associated machines and take the union of their results. Since they have to terminate, we do not have to worry about infinite loops.

R.E. sets are closed under union by a similar argument, with a simple twist. Since r.e. sets can loop forever, we have to run the first machine for 1 step and then run the second for 2 steps then switch back to the first machine and run it for 3 steps. Keep doing this and take the union of the results.

Star on recursive and R.E. sets can have the same argument applied to it: simply take the output and instead of taking the union, append it to itself an arbitrary number of times. Note: Concatenation is just a special case of star, so it is also closed.

Homomorphisms are closed under R.E. sets but not closed under recursive sets. Using homomorphisms we can change a recursive set into a set that never terminates, therefore making it an R.E. set.

Inverse homomorphisms are closed under both recursive and R.E. sets. We can show this by machine construction by taking the homomorphism of the language and going backwards from there.
3. Define an invalid computation of a Turing machine.

A valid computation can be expressed as a set of instantaneous descriptions:

\[ ID_1 \rightarrow ID_2 \rightarrow ID_3 \rightarrow \ldots \rightarrow ID_f \]

Any error introduced to this will create an invalid computation. Below are a few:
1) One of the ID’s could contain a move that is not valid for the Turing machine.
2) The set of instantaneous description is missing one of the ID’s such as the final one.
Construct a context-free grammar for the set of invalid computations of a Turing machine starting with a given input x.

Given a Turing Machine \( M = (Q, \Sigma, \Gamma, \delta, q_0, B, F) \), a valid computation will be of the form:

\[
    w_1 \# w_2^R \# w_3 \# w_4^R \# w_5 \# \ldots \# w_n
\]

or

\[
    w_1 \# w_2^R \# w_3 \# w_4^R \# w_5 \# \ldots \# w_{n1} \# w_n^R
\]

For a string to be a valid computation the following must hold:\footnote{http://www.dicom.uninsubria.it/mbenini/corsi/infteo03/Decidability.pdf, 11}

1. each \( w_i \) is a configuration of \( M \), that is, a string in \( \Gamma^*Q\Gamma^* \), which does not end by a B,
2. \( w_1 \) is the initial configuration, that is, \( w_1 = q_0x \) where \( x \) is our string \( x \in \Sigma^* \),
3. \( w_n \) is a final configuration, that is, a string in \( \Gamma^*F\Gamma^* \),
4. for each \( i, 1 \leq i \leq n \), \( w_i \to w_{i+1} \).

First, assume:

\[
Q = \{ \text{all states from } M \} \\
G \to \Gamma G | \epsilon \\
N_f = \{ y | y \not\in F \} \\
N_i = \{ y | y \in Q \text{ and not } q_0 \} \\
N_x = \{ y | y \neq x \} \text{ (where } x \text{ is the input string to } M) \\
A \to \Gamma Q A | \# 
\]

We can create a CFG for each of these conditions:

1. Basically what this constraint tells us is that the string must be of the form \( (\Gamma^*Q\Gamma^*)^*(\Gamma^*Q\Gamma^*) \), so

\[
S_1 \to \# A A \# A \# A A \# A A \# A A Q Q A \epsilon
\]
2. \( S_2 \rightarrow N_i A \mid q_0 N_x \mid \# A \)^

Note that given some \( x \) that we are taking as input, we could write \( N_x \) as a CFL (in fact it is actually regular).

3. \( S_3 \rightarrow A \# G N_f G \)

4. This one is a little bit trickier. It could be that:

- We made an invalid move right. A valid move right for a production \( \delta(q, a) = (p, b, R) \) will look like \( 0qa11\#11pb0 \).

  \[
  S_4 \rightarrow \Sigma S_4 \Sigma | E_1 \\
  E_1 \rightarrow qaE_2pb \\
  E_2 \rightarrow \Sigma E_2 \Sigma | \#
  \]

  Where an \( E_1 \) production is created for all choices of \( q, p \in Q, a, b \in \Sigma \) such there does not exist a transition of the form \( \delta(q, a) = (p, b, R) \).

- We made an invalid move left. A valid move left for a production \( \delta(q, a) = (p, b, L) \) will look like \( 0qa11\#11bp0 \).

  \[
  S_5 \rightarrow \Sigma S_5 \Sigma | E_3 \\
  E_3 \rightarrow cqaE_4 bcp \\
  E_4 \rightarrow \Sigma E_4 \Sigma | \#
  \]

  Where an \( E_3 \) production is created for all choices of \( q, p \in Q, a, b, c \in \Sigma \) such there does not exist a transition of the form \( \delta(q, a) = (p, b, L) \).

Finally, we tie this all together by creating:

\[
S \rightarrow S_1 | S_2 | S_3 | S_4 | S_5
\]