Exercise: (Section 3.3.2 from textbook)

Consider the following languages over \( \Sigma = \{0,1\} \):

\[
L_e = \{ E(M) \mid L(M) = \emptyset \}
\]
\[
L_{\neg e} = \{ E(M) \mid L(M) \neq \emptyset \}
\]

\( L_e \) is the set of all strings that encode TMs which accept the empty language.

\( L_{\neg e} \) is the complement of \( L_e \).

Claim 1: \( L_{\neg e} \) is R.E.

Proof: construct NTM \( N \) for \( L_{\neg e} \) (and then convert \( N \) to an ordinary TM.)

\( N \) works as follows: on input \( E(M) \):

1) Guess a string \( w \in \Sigma^* \)
2) Simulate \( M \) on \( w \) (like a UTM)
3) Accept \( E(M) \) if \( M \) accepts \( w \)

We have:

\[
E(M) \in L(N) \iff \exists w \text{ s.t. } \langle M, w \rangle \in E(U) \implies \exists w \text{ s.t. } w \in L(M) \implies E(M) \in L_{\neg e}
\]
Claim 2: \( L_{ne} \) is non-recursively enumerable

Proof: by reduction from \( L_m \) to \( L_{ne} \)

Reduction \( R \) is a function computable by a halting T.M.
with input: existence \( \langle M, w \rangle \) of \( L_m \)
output: existence \( \exists (M') \) of \( L_{ne} \)
end set: \( \langle M, w \rangle \in L_m \iff \exists (M') \in L_{ne} \)

Description of \( M' \):
- \( M' \) ignores completely its own input string \( \bar{x} \)
- instead, it replaces its input by the string \( \langle M, w \rangle \) and runs \( M \) on \( w \) (see (*) below)

- if \( M \) accepts \( w \), then \( M' \) accepts \( \bar{x} \)
- if \( M \) never halts on \( w \) or rejects \( w \)
  then \( M' \) also never halts on \( \bar{x} \)

Note: if \( w \in L(M) \Rightarrow L(M') = \Sigma^* \)
if \( w \notin L(M) \Rightarrow L(M') = \emptyset \)

hence \( \langle M, w \rangle \in L_m \iff \exists (M') \in L_{ne} \)

We can construct a halting T.M. \( M_R \) that, given \( \langle M, w \rangle \) as input, constructs \( \exists (M') \) for an \( M' \) that behaves as above.

(\textit{*)} \( M' \) has the following form: (let \( w = a_0, \ldots, a_n \))

\[ \xymatrix{ 0/1/3 \ar[r] & 1/6/2 \ar[r] & 1/6/2 \ar[r] & \cdots \ar[r] & 0/2 \ar[r] & 1/6 \ar[r] & 1/6 \ar[r] & 0/9 \ar[r] & \cdots } \]

\[ 0/9 \] \( M \)

To sum up, we have that \( L_{ne} \) is RE but non-recursively enumerable.
Hence \( L_{ne} \) must be non-RE.
The halting problem $L_{\text{Halt}}$, the set $\langle M, w \rangle$ s.t. $M$ halts on $w$ (with or without accepting) is R.E. but not recursive.

To show R.E., we construct a T.M. $H$ s.t.

$L_H = \{ \langle M, w \rangle | M \text{ halts on } w \}$

To show that $L_H$ is not recursive, we assume by contradiction it is R.E., and derive that $L_H$ is recursive.

By contradiction, let $H$ be an algorithm for $L_H$ and $U$ a procedure for $L_H$.

$A_n$ would be an algorithm for $L_H$, contradiction.
Let \( L \) be R.E. and \( \overline{L} \) be non-R.E.

Consider \( L' = \{0w \mid w \in L\} \cup \{1w \mid w \notin L\} \).

What do we know about \( L' \) and \( \overline{L}' \)?

We show that \( L' \) is non-R.E.

Suppose by contradiction that we have a procedure \( M_L \) for \( L' \).

Then we can construct a procedure \( M_{\overline{L}} \) for \( \overline{L} \) as follows:

- on input \( w \), \( M_{\overline{L}} \) changes the input to \( 1w \) and simulates \( M_L \).

- if \( M_L \) accepts \( 1w \), then \( w \in L \), and \( M_{\overline{L}} \) accepts.

- if \( M_L \) does not terminate or terminates and answers no, then \( w \notin L \), and \( M_{\overline{L}} \) does not terminate or terminates and answers no.

\[ \Rightarrow M_{\overline{L}} \text{ would accept exactly } \overline{L}. \text{ Contradiction} \]

\[ \overline{L}' = \{0w \mid w \in L\} \cup \{1w \mid w \notin L\} \cup \\{\varepsilon\} \]

Reversing as for \( L' \), we get that \( \overline{L}' \) is non-R.E.
Fl, the complement of the halting problem, i.e.,
the set of pairs \( \langle M, w \rangle \) such that \( M \) on input \( w \)
does not halt, is non-R.E.

Proof: By reduction from \( E_n \), which is non-R.E.

Idea: we show how to convert any TM \( M \) into another
TM \( M'_h \) such \( M'_h \) halts on \( w \) iff \( M \) accepts \( w \).

Construction:
1) Ensure that \( M'_h \) does not halt unless \( M \) accepts.
   - Add to the states of \( M \) a new loop state \( q, \) with
     \[ \delta(q, x) = (q, x, r) \] for all \( x \in \Gamma \)
   - For each \( \delta(q, y) \) that is undefined and \( q \neq F \)
     - add \( \delta(q, y) = (q, y, r) \)

2) Ensure that, if \( M \) accepts, then \( M'_h \) halts
   - Make \( \delta(q, x) \) undefined for all \( q \neq F \) and \( x \in \Gamma \)

3) The other moves of \( M'_h \) are as those of \( M \).

\( \square \)