Answer one of the following. For full credit answer part a. For part credit answer part b.

a. (5 points) An enumerator, as described in class and in the text, is a Turing recognizer but not a Turing decider. Describe a modification of an enumerator that is a Turing decider.

b. (3 points) What is the difference between a Turing recognizer and a Turing decider?

Answer to a. Have the enumerator print out the strings in order of length – i.e., all strings of length 1 are printed, then all strings of length 2, etc. If a string $w$ of length $k$ is not printed out and the enumerator is printing strings longer than $k$, then the enumerator rejects $w$.

Here’s another possibility that works only for finite languages. Make the enumerator send a “done” signal. When this signal is read, all strings in the language have been printed. Any string not printed is not in the language and can be rejected.

Answer to b. A Turing recognizer may not halt. A Turing decider halts (accepts or rejects) on any input string.
(2) (5 points) Let \( \text{NOINT}_{\text{DFA}} = \{<A,B> | \text{A and B are DFA’s such that no string accepted by A is accepted by B}\} \). Prove that \( \text{NOINT}_{\text{DFA}} \) is a decidable language by describing a Turing machine that decides \( \text{NOINT}_{\text{DFA}} \). You may use any of the languages we have proven are decidable in class. In particular, you may call the decider that decides any of these languages within the decider you describe. I suggest you consider \( \text{E}_{\text{DFA}} = \{<A> | \text{A is a DFA such that } L(A) = \emptyset\} \).

The following Turing machine decides \( \text{NOINT}_{\text{DFA}} \):

\[ M = \text{“On input } <A,B> \]
\[ \quad 1. \text{ Generate Turing machine } C \text{ such that } L(C) = L(A) \cap L(B) \]
\[ \quad 2. \text{ Submit } C \text{ to the TM that decides } \text{E}_{\text{DFA}} \]
\[ \quad 3. \text{ If it accepts, then } \text{accept}; \text{ if it rejects, then } \text{reject”} \]