Comments on Homework 3

1. (5 pts) Show that if L is recursively enumerable but not recursive then any TM accepting L must run forever (without crashing) on infinitely many inputs (also 10.3 on page 307).

Key idea: Any finite language is recursive (it can be decided by a FSM), so if some TM T accepts L while looping forever on only a finite set of strings then this finite set of strings can easily be recognized. This allows us to build a TM T′ that recognizes (i.e. computes the characteristic function for) L as follows.

First, check to see if the input x is one of the finite strings where T loops forever. If so, erase tape, output 0, and halt as x \not\in L. This can be by the finite state control without changing the tape contents.

Second, (assuming first part doesn’t halt) use universal TM to simulate T on x. If T crashes, then erase tape and output 0 (as x \not\in L). If T halts, then erase tape and output 1 (as x \in L).

Note that the recognizing TM must compute the characteristic function.

2. (5 pts) Problem 10.4 on page 307 (Countable union of recursively enumerable languages is recursively enumerable).

The key idea here is that any language is countable, and thus is the countable union of single-element languages. Since every single-element language is recursively enumerable (and even regular), every language is the countable union of recursively enumerable languages. However some languages (such as NSA) are not recursively enumerable, giving a contradiction.

3. (5 pts) Problem 10.6 on page 307 (recursively enumerable languages closed under concatenation and *).

Several groups wrote answers similar to how the regular languages are closed under these operations. However, a slightly different idea is needed here. For example, let \( L = \{ x \in 1^* : |x| \text{ is odd} \} \). One TM accepting this language loops through the string keeping track of its parity until it hits a blank, and then either keeps looping (if the parity
is even) or halts (if the parity is odd). This TM loops forever on the input “1111” (which is “1” concatenated with “111”), so we can’t simply splice two copies of the TM together to accept $L$ concatenated with $L$. The essential trick is to use non-determinism to “guess” where the input $x$ gets split into two strings and then simulate the TM accepting $L$ on each of the two strings. The Kleene-∗ construction is similar with the machine accepting $L^*$ repeatedly: guessing a prefix of the remaining input, verifying that the prefix is in $L$, and deleting the prefix; until the input is exhausted.

4. (5 pts) Problem 10.8 on page 307 (harder, for which canonical orders does enumerability in that order equate with recursiveness?). You may want to solve 10.7 as a first step.

Here the big idea is that if you can compute the ordering then (if $L$ is recursive) you can examine strings in the ordering and list those in $L$. Similarly, if $L$ is finite you can decide $L$, and if $L$ is infinite you can use an enumerator in the ordering to decide if a string is in $L$ by waiting until you see it or a later string.

However, there are many non-computable orderings such that even simple languages cannot be listed in that ordering. For example, consider the ordering that lists strings on order of increasing length, and within each length first lists the strings in NSA (in lexicographic order) followed by the strings not in NSA (again, in lexicographic order). Any TM that enumerates $\Sigma^*$ in this ordering would let us build an acceptor for NSA (which we know doesn’t exist).