CS500, Theory of Computation
Homework #3

Note: You may discuss this homework with others in the class. However, you must do your own writeup, and you must clearly state on your homework who you worked with. Due by email in .pdf format by midnight on Friday, March 12th.

1. Suppose I have a Turing machine M with a doubly-infinite tape, i.e. a tape that stretches both left and right to $x = \pm\infty$. Prove that such a machine can be simulated with a standard Turing machine M' whose tape is only infinite on one side, with x ranging from 0 to ∞. You don't have to describe the simulation in complete detail, but you should describe it well enough to say how many states and tape symbols M' has as a function of how many M has, and how long the computation time of M' will be in terms of that of M.

2. A *queue automaton* or QA is like a PDA, but with a queue rather than a stack. That is, it is a finite-state automaton which can look at an input symbol, its own state, and the front symbol of a queue; its transition function allows it to change its state, pop the front symbol, and/or push a symbol onto the back of the queue.

   Show that, unlike PDAs which can only recognize context-free languages, QAs can simulate Turing machines. Hint: simulating T steps of the Turing machine might take the QA more than $O(T)$ steps. How long does it take?

3. For the simulation in the previous problem to work, it is vital that the QA be able to make ϵ-transitions. A *real-time* QA is one which is not allowed to make ϵ-transitions, i.e., it is required to take exactly one step of its transition function for each symbol of the input, and is given no additional time to think; it must accept or reject as soon as it has read the input.

   How powerful are real-time QAs? Give an example of a non-context-free language which can be recognized by a real-time QA. On the other hand, give an example of a context-free language which seems intuitively hard for a real-time QA to recognize (you don't need to give a proof).

4. Suppose that instead of being able to move left or right, a Turing machine is only allowed to stay put or move right. Show that such a machine can only recognize regular languages.

5. Let's say that a language L is *enumerable in order* if there is a Turing machine that prints out the words in L in lexicographic order (although it might take arbitrary time to print each one). Show that if L is enumerable in order, then L is decidable.

6. Write the pseudocode of a C program which eventually prints out every C program (which run on their own with no input) which halts. Feel free to use words like "compile" and "run for t steps", but make sure that every program which will ever halt is printed out by your program at some finite time.

   Now, is there such a program which prints out halting programs in order of increasing length? Give the pseudocode for such a program, or prove that none exists.

7. True or false: any subset of a decidable language is decidable. If true, give a proof; if false, give a counterexample.

8. The Goldbach Conjecture states that all even numbers can be written as the sum of two odd primes. Let L be the language

$$L = \begin{cases} Y & \text{if Goldbach's conjecture is true} \\ N & \text{if it isn't} \end{cases}$$

   Is L decidable?

9. A Turing machine is a *decider* if it halts on all inputs. IsDecider is the following problem: given the description of a Turing machine M, tell whether is a decider. Show that IsDecider is undecidable by reducing the Halting Problem to it.

10. Now, show that the set of deciders is not even recursively enumerable! In other words, show that there is no Turing machine enumerator which prints out descriptions of all the deciders. This proves that telling whether a Turing machine halts on *all* inputs is actually harder than telling whether it halts on a particular input. Hint: suppose such an enumerator exists, and use diagonalization.