CSE 531: Computability and Complexity — Autumn 2005
Problem Set #1 — Instructor: Paul Beame
Due by **October 14, 2005** before 4:30 p.m.

Reminder: If you haven't done so already, subscribe to CSE 531 email group ASAP by visiting `https://mailman.cs.washington.edu/mailman/listinfo/cse531`.

Instructions: You are allowed to collaborate with fellow students taking the class in solving problem sets, but you must write up your solutions on your own. If you do collaborate in solving problems, you must acknowledge for each problem the people you worked with on that problem.

You are expected to refrain from referring to any outside source other than Sipser's text and your class notes in coming up with your solutions. Using any pre-existing solutions from outside sources is not allowed.

Most of the problems require only one or two key ideas for their solution – spelling out these ideas should give you most of the credit for the problem even if you don't get all the details right. Make sure you clearly write down the main idea(s) behind your solution even if you cannot figure out a complete solution.

---

1. (10 points) Show that a language is decidable iff some enumerator enumerates the language in lexicographic order. (Recall that the lexicographic order, puts shorts strings before longer strings and uses dictionary order for strings of the same length. For example, the lexicographic ordering of all strings over $\{0,1\}$ is $\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$.)

2. (10 points) Show that a language C is Turing-recognizable iff there exists a decidable language D such that $C = \{x \mid \exists y(\langle x, y\rangle \in D)\}$)

3. (10 points) Define the language

$$S = \{\ \langle M\rangle \mid M \text{ is a DFA that accepts } w^{\mathcal{R}} \text{ whenever it accepts } w\}.$$

   Prove that S is **decidable**.

4. (10 points) Prove that deterministic pushdown automata (DPDA) that have 2 stacks can simulate Turing machines. (The transition function of the 2-DPDA depends on the input symbol as well as the top symbols of the two stacks; the 2-DPDA can push/pop from both stacks and like a regular PDA, it has one-way read-only access to the input but the input is followed by a blank symbol as an end marker. Moreover, at each step precisely one transition of the DPDA is defined but, unlike a DFA, a 2-DPDA may may may make a transition from a state without reading any input symbol.)

5. (10 points) A (deterministic) *queue automaton* is like a (deterministic) pushdown automaton except that the stack is replaced by a *queue*. A *queue* is a tape allowing symbols to be written on the left end and read only at the right end. Each write operation (a push) adds a symbol to the left end of the queue and each read operation reads and removes the symbol at the right end of the queue. The input is given on a separate one-way read-only input tape with a blank end marker. Prove that deterministic queue automata are equivalent to Turing Machines.

6. (10 points) An *unrestricted grammar* (or a *rewriting system*) is a 4-tuple $G = (V, \Sigma, R, S)$ where

   - V is an alphabet;
   - $\Sigma \subset V$ is the set of *terminal* symbols, and $V - \Sigma$ is called the set of *nonterminal* symbols;
   - $S \in V - \Sigma$ is the *start* symbol; and
   - R, the set of *rules*, is a finite subset of $(V^*(V - \Sigma)V^*) \times V^*$.

   (Thus the "only" difference from context-free grammars is that the left-hand sides of rules need not consist of single nonterminals.) Let us write $\alpha \to \beta$ if $(\alpha, \beta) \in R$; and let's define $u \Rightarrow_G v$ iff, for some $w_1, w_2 \in V^*$ and some rule $\alpha \to \beta \in R$, $u = w_1\alpha w_2$ and $v = w_1\beta w_2$. Let $\overset{*}{\Rightarrow}_G$ denote the reflexive, transitive closure of $\Rightarrow_G$. We say that a string $w \in \Sigma^*$ is generated by G if and only if $S \overset{*}{\Rightarrow}_G w$. Finally, $L(G) \subseteq \Sigma^*$, the *language generated by* G, is defined to be the set of all strings in Σ^* generated by G.

   Your exercise is now to prove that a language is generated by an unrestricted grammar if and only if it is Turing-recognizable.

   (Hint: Using nondeterminism and/or multiple tapes might aid in constructing Turing machines to simulate a grammar. For the other direction, to simulate a Turing Machine M by a grammar, try to construct a grammar whose rules simulate backward moves of M, and whose derivations will consequently simulate backward computations of M.)