CSE 531: Computability and Complexity — Autumn 2004
Problem Set #1 — Instructor: Venkatesan Guruswami
Due on **October 19, 2004** in class.

Reminder: If you haven't done so already, subscribe to CSE 531 email group ASAP by visiting `http://majordomo.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 source other than Sipser's text and your class notes in coming up with your solutions. The problems have been carefully chosen for their pedagogical value and hence might be similar or identical to those given out in past offerings of this course at UW, or similar courses at other schools. Using any pre-existing solutions from these sources, or using solution material from the Web is, needless to say, strictly prohibited.

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 err in some finer details. So, make sure you clearly write down the main idea(s) behind your solution even if you could not figure out a complete solution.

Each problem is worth 10 points unless noted otherwise.

1. Problem 3.13, Sipser's book (Turing machines with stay put instead of left)

2. Problem 4.21, Sipser's book.

3. Problem 3.16, Sipser's book (A language is decidable if and only if some enumerator enumerates it in standard order)
   - Note that Sipser's book uses the terminology lexicographic order to mean the familiar dictionary order except that shorter strings precede longer strings. Thus, the lexicographic ordering of all strings over $\{0, 1\}$ is $\{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. We will refer to this ordering as the standard order throughout the course.

4. Define the language

$$A = \{ \langle M \rangle \mid M \text{ is a nondeterministic finite automaton (NFA) that only accepts palindromes in } \{0,1\}^* \} .$$

   (Note that for $\langle M \rangle$ to be in A, it need *not* accept all palindromes, but any string it accepts must be a palindrome.) Prove that A is **decidable**.

5. (20 points)
   (a) Prove that a pushdown automaton (PDA) that has 2 stacks can simulate Turing machines. (The transition function of the 2-PDA depends on the input symbol as well as the top symbols of the two stacks; the 2-PDA can push/pop from both stacks and like a regular PDA, it has one-way read-only access to the input.)

(b) In this exercise, your goal is to show that the two stacks above can be replaced by just two counters (or equivalently, stacks over a *unary* alphabet). Formal details follow.

A k-counter machine (k-CM, for short) is a Turing machine with a read-only input tape that contains the input to the machine (assume that the ends of the input tape are marked with special symbols). The k-CM can move its head on the input tape in either direction (i.e. left or right), but it cannot write anything onto the tape. For its writable memory, a k-CM is given access to k counters $C_1, C_2 \ldots, C_k$ each of which can hold a non-negative integer. The k-CM cannot access the contents of the counters, but can check whether $C_i = 0$ for each i, $1 \leq i \leq k$. It can thus find out which subset of the counters, if any, are zero. A k-CM computes as follows: initially, the tape head is at the left end of the input, the control is in the start state, and all counters equal zero. In a single step, depending upon its current state, the symbol under the head on the input tape, and the subset of counters which are zero, a k-CM:

(i) changes its finite control to the appropriate next state;

(ii) moves its input tape head to the left or right by one cell; and

(iii) decrements or increments one of the k counters by 1 (assume that attempts to decrement a zero counter do not affect the counter)

As with a Turing machine, a k-CM accepts by entering a special accept state, rejects by entering a special reject state, and could also run forever without halting. Now, to your exercise.

- Prove that a 2-counter machine can simulate an arbitrary Turing machine. Conclude that 2-counter machines recognize precisely the class of Turing-recognizable languages.

Suggestion: First, show how two counters can be used to simulate an arbitrary stack. Then use what you showed in Part (a) to conclude that 4-counter machines can simulate Turing machines. Finally, does one really need four counters? – try to simulate their functionality using just two counters (hint: use a one-to-one map from $\mathbb{N}^4$ to $\mathbb{N}$, where $\mathbb{N}$ is the set of positive integers).