CS 182, Complexity Theory
Fall 2010
Homework 5
Due Wednesday, October 6

Woohoo! This is the last homework assignment in the course! The take-home exam will be available on Friday, October 8 and will be due back no later than Friday, October 15 at 5 PM.

1. **[35 Points] 2SAT is NL-complete!** In class, we briefly reviewed the proof that 2SAT is in P (which we did rigorously in a homework assignment in CS 140). You may assume that result here without reproving it. Now, you'll show that 2SAT is NL-complete. Amazing, but true!

   (a) In class we showed that PATH is NL-complete. Here, it will be useful to show that a related problem, DAG-PATH, is NL-complete: DAG-PATH is the language of triples G, s, t where G is a directed acyclic graph (DAG) and there is a path from s to t in G. Prove that DAG-PATH is NL-complete.

   (b) Now prove that 2SAT is NL-hard. You may wish to use the fact that DAG-PATH is NL-complete. It's also good to keep in mind that NL = co-NL.

   (c) Finally, show that 2SAT is in NL.

   (d) Conclude that 2SAT is NL-complete.

2. **[25 Points] The Time Hierarchy Theorem!** Recall the amazing Space Hierarchy Theorem that stated that if $f(n)$ is space-constructible then there exists a language L such that L is decided in space $O(f(n))$ but is not decided by any machine that uses asymptotically less space - that is space $o(f(n))$. In this problem, you will prove a similar theorem for time.

   There are several ways of defining the notion of a "time constructible" function. The definition that we'll use is this: *A function $f(n)$ is time constructible if there exists a standard single-tape Turing Machine that on any input of length n runs for exactly $f(n)$ steps and then halts.* Although we won't prove it here, the set of time constructible functions is closed under multiplication, addition, exponentiation, composition, etc. That is, it is a very rich set of functions!

   Your goal is to prove the following Time Hierarchy Theorem: "If $f(n)$ is a time constructible function then there exists a language that is decidable on a

1-tape Turing Machine in $O([f(n)]^2)$ time but is not decidable by any 1-tape Turing Machine in $o(f(n))$ time."

Prove this result using the same approach that we used to prove the Space Hierarchy Theorem. (Note that here we are being specific about the fact that the TM's in question have just one tape because we can do things faster with two tapes than with one. It's true that a 1-tape TM can simulate a multi-tape TM, but there is a time cost in performing this simulation. It's worth recalling the cost of simulating a multiple tape TM with a 1-tape TM.)

3. **[20 Points] Speedup!** Consider a 1-tape TM whose worst-case running time is given by a function $f(n)$. (By this we mean that the actual worst-case running time is upper-bounded by $f(n)$, not $O(f(n))$.) You may assume that $f(n) \geq kn^c$ where k and c are some constants that you get to specify.

   (a) Prove that we can build a 1-tape TM M' that decides the same language as does M and whose worst-case running time is twice as good: That is the worst-case running time of M' is at most $f(n)/2$. In other words for "sufficiently large" functions $f(n)$, a 1-tape TM running in time $f(n)$ can be sped up by a factor of 2.

   (b) Professor Lai is worried (which is uncharacteristic of him!). "By repeating this result enough times, you can make a TM as fast as you want, contradicting the Time Hierarchy Theorem in at least some cases." What's wrong with Prof. Lai's logic?

4. **[20 Points] Multi-Tape Time Constructibility.**

   In Problem 2, we mentioned that the set of time constructible functions is closed under a variety of operations. Here we will see how such closure properties can be proved. However, we will change the model slightly just to avoid having to deal with a lot of piddly issues.

   In particular, in this problem, we consider time constructibility in multi-tape Turing Machine means. Specifically, throughout this a problem a TM may have some constant number of tapes, each with its own tape head. The tapes are assumed to be infinite in both directions. The input string appears on the first tape with the tape head under the leftmost (non-blank) symbol of the string. In one time step, the machine consults all of its tape heads, writes new symbols under each tape head, and moves each tape head independently to the left, right, *or keeps it in the same location.*

We say that a function f is time constructible in this model if there exists a TM (as defined above) such that on any input of length n the machine runs for exactly $f(n)$ steps and then halts. Note that the running time depends only on the length of the input.

Prove that if f and g are time constructible then so is $f \circ g$ where $f \circ g$ is the composition of f and g. That is, $(f \circ g)(n) = f(g(n))$. Note that $f(g(n)) \neq f(g(n)) + g(n)$.

5. **[20 Point Optional Bonus Problem] The Strange Complexity Class P/log!**

   Here's an interesting "new" complexity class: It's called P/log. This class comprises all languages (unary, binary, or over any other alphabet) with the following property: There exists a polynomial time deterministic verifier V, a constant k, and an omnipotent prover P; On input of length n, the prover P sends V an advice string of length at most $k \log n$. This advice string is the same for all inputs of length n. In other words, the advice depends only on the length of the input and not on the particular symbols in the input. The verifier V then computes deterministically in time polynomial in n and chooses to accept or reject its input.

   Notice that we are *not* saying that the verifier accepts its input if there exists some advice string that would make it accept. Rather, we are saying that there exists a specific prover that works with this verifier to issue just the right advice string. In other words, just because some advice string of length $O(\log n)$ makes the verifier accept does not mean that it correctly decided whether its input string is in the language.

   (a) Show that the the class P/log contains undecidable languages. Aha, so short advice strings can be very useful! Funky!

   (b) Since P/log contains undecidable languages, we might be tempted to believe that it contains all of NP. However, in part (c) below you'll show the amazing fact that if SATISFIABILITY is in P/log then P = NP. **Before** doing this though, prove that if we had a polynomial time decider for the SATISFIABILITY problem then we could construct a polynomial time algorithm for finding an actual satisfying assignment (a "valuation") for any instance of SATISFIABILITY if one exists. This idea is called "self reducibility" and you may recall that we talked about this in the "Algorithms" course. (*Recall:* SATISFIABILITY is the general form of 3SAT:

The expression is in conjunctive normal form but there can be any number of literals per clause.)

(c) **OK, now show that** if SATISFIABILITY is in P/log then P = NP. *Note:* Without loss of generality, you may assume – if you wish – that an instance of SATISFIABILITY whose ending is malformed, that is it ends with an "AND" symbol or the last clause is only partially formed (i.e. the last clause is malformed because it ends with an "OR" symbol or is missing a closing parethesis, etc.), is interpreted as an instance with the malformed ending omitted. This can just make your "housekeeping" a bit simpler. This assumption is actually not necessary.