CS 142 & Math 167
Fall 2008
Homework 7
Due Wednesday, October 29

Before embarking on this assignment, remind yourself of everything that we learned about the classes L, NL, co-NL and their relationships. The proof techniques that we used in class are also likely to be useful to you, so make sure that you understand those as well.

1. **[30 Points] Finishing Up the NL = co-NL Proof!** Your task is to write out *the entire proof* that NL = co-NL. Our approach was to show that co-PATH is in NL. First, briefly explain why showing this implies that NL = co-NL.

   In class we started the proof that co-PATH is in NL. The funny thing that we assumed in that proof is that the non-deterministic log space TM for co-PATH was given not just $< G, s, t >$, but also r, the number of vertices reachable from s in G. In order to complete the proof, you'll need to show that r can be computed by the non-deterministic log space TM by itself before launching into the step that we described in class. In other words, the non-deterministic log space TM for co-PATH will construct a tree that has two "parts". The "upper part" will compute r and the lower part will use r to determine that there is no path from s to t.

   (a) Professor I. Lai of P.I.T. has proposed the following approach for computing r: "It's easy!" explains Professor Lai nonchalantly. "You simply have your non-deterministic machine first use non-determinism to guess every possible value of r and place this guess in a counter. Writing r down takes log space. Now, each guess of the counter's value appears in a different subtree of our non-deterministic tree. In each subtree, we now verify that guess as follows: Use non-determinism again to sequentially guess the vertices that we believe to be reachable from s. For each such guessed vertex we first verify that we can reach that vertex (in the same way that we checked if s can reach t in our proof that PATH is in NL) and, if the answer is "yes", we decrement the counter. If we get to a point that the counter is zero, we conclude that we guessed the correct value of r and we now have r to use in the rest of the construction!" Now "r-gue" that Professor Lai is wrong!

(b) To get you started on a correct method for computing r, let r_i denote the number of vertices reachable from s using i or fewer edges. So, $r_0 = 1$ since there is exactly one vertex reachable from s using 0 or fewer edges (it's s itself!). If ℓ is the number of vertices in G, then $r = r_\ell$ (in fact, $r = r_{\ell-1}$). You'll want to compute r_ℓ. To that end, describe a method for computing r_i assuming that r_{i-1} has already been computed. Remember, you may use "lots" of time and non-determinism. Once you've described this process, put everything together to explain why co-PATH is in NL. Be sure to show that your algorithm only requires log space!

2. **[15 Points] Bipartite Graphs.** Recall that a bipartite graph is an undirected graph with no odd-length cycles. Let BIPARTITE be the language of all undirected graphs (represented using some reasonable convention) that are bipartite. Show that BIPARTITE is in NL. (*Note:* It was recently shown that BIPARTITE is in L, but this proof is quite complicated.)

3. **[35 Points] 2SAT is NL-complete!** We know that 3SAT is NP-complete. In this problem, you'll show that 2SAT is NL-complete. Amazing, but true!

   (a) First, let's prove that 2SAT is in P. (Some of you may have done this on an "Algorithms" assignment, but it's worth doing again here!). To that end, let's represent an instance of 2SAT with a graph. Notice that a clause of the form $(a \vee b)$ (where a and b are literals, so they could be plain or negated variables) can be written as "if a is false then b must be true" and "if b is false then a must be true" or, symbolically, $\overline{a} \rightarrow b$ and $\overline{b} \rightarrow a$. So now, let's introduce vertices for each literal. The implication $\overline{a} \rightarrow b$ will introduce an edge from vertex $\overline{a}$ to vertex b and the implication and $\overline{b} \rightarrow a$ will introduce an edge from vertex $\overline{b}$ to vertex a. Now argue that the instance of 2SAT is satisfiable if and only if the graph has a particular property (that you will need to state) and that this property can be tested in polynomial time.

   (b) 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.

   (c) 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.

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

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

(f) For a small amount of bonus credit, eat chocolate to celebrate this very cool result!

4. **[20 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. Specifically, let $f(n)$ be a function such that we can compute the binary representation of n in time $O(f(n))$ (this is one of several ways of defining a function to be "time constructible"). Your goal is to show that there exists some language L that is decided (on a standard single-tape deterministic TM) in time $O(f(n))$ but is not decided (on a standard single-tape deterministic TM) in time $(o(f(n)/\log f(n))$.

   To get started, let's define a TM D that does the following on input w of length n.

   (a) Compute the binary representation of $f(n)$ and then use that to compute the binary representation of $f(n)/\log f(n)$, which we will call C. This C is our binary time-out counter. We will decrement this counter at each computation step and we will halt and reject if the counter ever gets to 0.

   (b) If w is not of the form $< M > 0^*$ then reject.

   (c) Simulate M on w. If M accepts w then we reject w. If M rejects w then we accept w.

   Now it's your turn!

   - Your first task is to show that TM D runs in time $O(f(n))$. To that end, you will probably want your machine to use several tracks (that is, an expanded alphabet). In addition, during the simulation of M on w, it will be important that the tape head of D doesn't need to zip back and forth too much to decrement the counter or to consult the transition function of M, since too much zipping can slow things down dramatically!
   - Next, prove that there does not exist any TM M such that M accepts the language of D in time $o(f(n)/\log f(n))$. Very cool!
   - Finally, briefly explain why we get this strange $o(f(n)/\log n)$ here whereas in the Space Hierarchy Theorem we got the stronger $o(f(n))$ result.