Homework #7 Solutions

Problem 1.

We prove that if NP-hard $\subseteq$ PSPACE-hard, then PSPACE = NP. We know that NP $\subseteq$ PSPACE by the arguments given in Sipser (see p. 282). We must show the opposite inclusion, namely that PSPACE $\subseteq$ NP. To see this, note that SAT $\in$ PSPACE and SAT is NP-complete. By the definition of NP-complete, SAT is also NP-hard, and by our assumption, SAT is PSPACE-hard. Thus, for any $L \in$ PSPACE, we have $L \leq_P$ SAT. But since SAT $\in$ NP, we have that L $\in$ NP. Therefore PSPACE $\subseteq$ NP.

Problem 2.

We prove that PARENS $\in$ L by giving a deterministic log-space algorithm to decide it. Our algorithm maintains a single binary counter k on the work tape; the counter is initally set to zero. The algorithm scans the input string from left to right. If a "(" character is encountered, $k \leftarrow k+1$. If a ")" character is encountered and $k = 0$, reject; otherwise, if $k > 0$, $k \leftarrow k-1$. Once all input characters have been examined, accept if $k = 0$ and reject otherwise.

The algorithm presented is valid since the number of open-parents is always greater than or equal to the number of close-parens in a proper expression. In addition, our counter requires only $\log_2 n$ bits, even in the worst case where the input string consists of n open-parens. Hence PARENS $\in$ L.

Problem 3.

We prove that STRONGLY-CONNECTED is NL-Complete. We begin by demonstrating that STRONGLY-CONNECTED $\in$ NL. Our algorithm A for testing whether a graph G is strongly connected nondeterministically selects two verticies v_0 and v_1 from G, and runs PATH(G, v_0, v_1). If there is a path from v_0 to v_1, A accepts. Otherwise, A rejects. Since PATH $\in$ NL, and each vertex chosen requires only a single counter, we conclude that STRONGLY-CONNECTED $\in$ NL.

To see that STRONGLY-CONNECTED is NL-Complete, we reduce PATH to it in deterministic log-space. Our log-space transducer T operates as follows. T begins with an instance of the PATH problem consisting of directed graph G, and two distinguished verticies s and t on it's input tape. T first copies the contents of the input tape onto the output tape, and clears it's work area. For each vertex $v \in G$, T appends the two edges $v \to s$ and $t \to v$ onto the output tape. Since we only need to count through the verticies in the graph, we only require logarithmic work space for this reduction.

We now prove that the new graph G' generated by T is strongly connected if and only if G has a path from s to t. Suppose G does not have a path from s to t. Then G' also has no path from s to t, and hence G' is by definition not strongly connected. On the other hand, suppose G does have a path from s to t. Then for any two verticies $v_0, v_1 \in G'$, there is a path $v_0 \to s \to \ldots \to t \to v_1$ between them, and by definition G' is strongly connected. This completes our proof.

Problem 4.

The error in the proof appears in the statement "Because every language in NP is polynomial time reducible to SAT, NP $\subseteq$ TIME(n^k)." In particular, it was asserted that for some constant k, SAT $\in$ TIME(n^k). However, polynomial time reductions take just that — polynomial time — and hence the reduction of a language $L \in$ NP to SAT is guaranteed to take TIME(n^{k+j}) where the reduction $\in$ TIME(n^j).

Problem 5.

Given that $A \in$ TIME(n^2), we want to show that $pad(A, n^2) \in$ TIME(m), where m is the length of the padded input. We give a deterministic $O(m)$ algorithm T to decide $pad(A, n^2)$. On input string w, algorithm T begins by counting the number of characters n before the first # character. T then determines whether $n^2 = |w|$. To do so, T simply marks the input characters from left to right, marking n copies of n characters. If any characters are unmarked, or if there were too few characters to mark, T rejects. This phase can clearly run in $O(m)$ time. Finally, T sends the first n characters to the decider for A, which runs in $O(n^2) = O(m)$ time. If A accepts, T accepts; otherwise, T rejects.