CS 4510 : Automata and Complexity
Using Nondeterministic Machines as Subroutines

Subrahmanyam Kalyanasundaram

April 25, 2010

I noticed the same incorrect solution repeated in many of your submissions for Homework 5. I think this is because of some subtleties involved in the usage of nondeterministic machines as subroutines.

The problem asks to show that *STRONG-CONN* is NL-complete. There are two parts to the solution–first showing that it is in NL and then showing that it is NL-hard. The incorrect solutions that I referred to were in the solution for the first part.

The correct proof that *STRONG-CONN* is in NL is the following algorithm. Loop over all choices of $u \in V(G)$ and $v \in V(G)$. Guess a path from u to v, nondeterministically, and reject if the guess does not form a path. Else continue till all choices of u, v are exhausted. Accept if not rejected yet. If G is strongly connected, then there exists a set of nondeterministic choices such that all guesses accept. If G is not strongly connected, then certain pairs u, v have no path from u to v, so all guesses would lead to a reject.

The first wrong proof that I saw was the following. Instead of looping over all u, v, why not guess a pair $u, v \in V(G)$ nondeterministically? If G is strongly connected, for any pair of u, v, there exists a guess sequence that would accept. So this algorithm will accept all $G \in \textit{STRONG-CONN}$. But when G is not strongly connected, even then the algorithm might guess a pair u, v such that G has a path from u to v. In this case, the algorithm has an accepting computation even when G is not strongly connected. So this algorithm is incorrect.

The second, and the more common, incorrect proof that I came across was the following: since NL = co-NL, it is enough to show that $\overline{\textit{STRONG-CONN}}$ has an NL algorithm, let us call this algorithm A. If $G \notin \textit{STRONG-CONN}$, then there exists $u, v \in V(G)$ such that there is no path from u to v. So A nondeterministically guesses the pair u, v and uses *CHECKPATH*, the NL algorithm which checks whether there is a path from u to v. A flips the output of *CHECKPATH*. That is, if *CHECKPATH* accepts, A rejects and vice versa. The idea is that if there is no path from u to v, then *CHECKPATH* would reject and A shall accept. Again, this does not work, but the reason is more subtle. We cannot flip the output of *CHECKPATH*. To understand why, we need to see the whole algorithm as one piece. Remember, we are shooting for an algorithm for $\overline{\textit{STRONG-CONN}}$. Suppose G is strongly connected. Algorithm A would still accept. A guesses u, v. Since G is strongly connected, there is a path from u to v. But *CHECKPATH* would make numerous wrong guesses, and would have several rejecting computations. This would correspond to numerous accepting computations for A, even though A must accept only when G is not strongly connected.

However, there is way to fix this. Use the NL algorithm for $\overline{PATH}$ as a subroutine[1]. So if G is not strongly connected, then there exists u, v for which there is no path from u to v, and the NL algorithm for $\overline{PATH}$ would accept this u, v. So we accept. If G is strongly connected, any pair u, v are connected, and the algorithm for $\overline{PATH}$ would reject. So all nondeterministic guesses would lead to reject.

So while using nondeterministic algorithms as subroutines, try to view the "big picture" and imagine whether the algorithm has the desired behavior.

[1]We can do this since NL = co-NL, so $\overline{PATH}$ has an NL algorithm as well