CS 151 Complexity Theory **Spring 2019**

Solution Set 2

Posted: April 19 *Chris Umans*

1. Suppose $L \in \mathbf{NP} \cap \mathbf{coNP}$. Then there exist languages R_1 and R_2 in $\mathbf{P}$ for which

$$\begin{aligned} L &= \{x : \exists y, |y| \leq |x|^{k_1}, (x,y) \in R_1\} \\ \overline{L} &= \{x : \exists z, |z| \leq |x|^{k_2}, (x,z) \in R_2\} \end{aligned}$$

On input x, our strong nondeterministic Turing Machine M will guess y and guess z. If $(x,y) \in R_1$ then we accept; if $(x,z) \in R_2$ then we reject; otherwise we output "?". The above equations imply that if $x \in L$ then some path leads to accept and if $x \notin L$ then some path leads to reject, as required. Moreover, when $x \in L$, no computation path rejects (because that would imply that there exists a z for which $(x,z) \in R_2$) and when $x \notin L$ no computation path accepts (because that would imply that there exists a y for which $(x,y) \in R_1$).

In the other direction, suppose we have strong nondeterministic Turing Machine M that decides L in time n^k. We can modify it so that whenever it would have output "?" it instead rejects. This gives an ordinary non-deterministic Turing Machine that decides L, and so $L \in \mathbf{NP}$. We can also modify it so that whenever it would have rejected it instead accepts and vice versa, and whenever it would have output "?" it instead rejects. This gives an ordinary non-deterministic Turing Machine that decides $\overline{L}$, and so $L \in \mathbf{coNP}$. We conclude that $L \in \mathbf{NP} \cap \mathbf{coNP}$.

2. (a) Recall that R is the reduction from SAT to a unary languages $U \subseteq 1^*$. Consider $x = R(\phi)$, where ϕ is a formula in the self-reduction tree. If $x \notin 1^*$, then we can easily detect this, and we used this observation critically to assign a *single* color to such strings. Since every other color was identified with a string in 1^* of length at most $p(n)$ (where $p(n)$ is a bound on the length of the string output by R) the total number of colors was $p(n)+1$. If we know only that U is sparse, we have a similar polynomial bound on the number of "satisfiable" colors, but not on the number of "unsatisfiable" colors, as it is perfectly legal for R to map unsatisfiable formulas to strings outside U, and we have no efficient way of detecting these strings and grouping them all into a single color, as we did with unary languages. Since there may be exponentially many different colors, the tree-traversal procedure is no longer guaranteed to run in polynomial time.

   (b) Let R be the reduction from $\overline{\text{SAT}}$ to S, let ϕ be an instance of $\overline{\text{SAT}}$ with $|\phi| = n$, and let $q(n)$ be the polynomial bound on the length of the strings output by R. We perform the same tree-traversal of the self-reduction tree for ϕ as we saw in class, identifying colors with strings output by R, and pruning a given color after we have seen it $n+1$ times. Since R maps *unsatisfiable* formulas to strings in S, we know that there are at most $p(q(n)) = \text{poly}(n)$ "unsatisfiable" colors. In the case that ϕ is unsatisfiable, the only colors are unsatisfiable colors, and so we terminate after visiting at most $(n+1)p(q(n))$

nodes. If ϕ is satisfiable, we may visit up to $(n+1)p(q(n))$ unsatisfiable nodes, and n satisfiable nodes along a path from the root to the first leaf corresponding to a satisfying assignment, at which point we terminate, again in polynomial time. Thus given the reduction R from $\overline{\text{SAT}}$ to S we can solve SAT in polynomial time, and therefore $\mathbf{P} = \mathbf{NP}$.

(c) If we knew $c(|x|)$ we could nondeterministically guess *all* of the strings in S of length at most $|x|$ together with proofs that each is in S, and then accept only if x is not among those strings. Following essentially this strategy, we will show that $\hat{S} \in \mathbf{NP}$. Since $S \in \mathbf{NP}$ there is a language $L \in \mathbf{P}$ for which

$$S = \{y : \exists z,\ |z| \leq |y|^d,\ \ (y,z) \in L\}.$$

Using this, we describe the following language in **NP**:

$$\begin{aligned}\{(x, 1^k) \ \ : \ \ & \exists (y^{(1)}, y^{(2)}, \ldots, y^{(k)}, z^{(1)}, z^{(2)}, \ldots, z^{(k)}),\ |y^{(i)}| \leq |x|,\ |z^{(i)}| \leq |y^{(i)}|^d, \\ & (x, y^{(1)}, y^{(2)}, \ldots y^{(k)}, z^{(1)}, z^{(2)}, \ldots z^{(k)}) \in W\},\end{aligned}$$

where W accepts iff the $y^{(i)}$ are all distinct, $(y^{(i)}, z^{(i)}) \in L$ for all i, and x is not among the $y^{(i)}$. Clearly $W \in \mathbf{P}$, so the language above is in **NP**. We claim this language is $\hat{S}$. If $k < c(|x|)$ then we can guess k distinct strings $y^{(i)}$ from among the $c(|x|)$ strings of length at most $|x|$ in S, together with proofs $z^{(i)}$ that they are indeed in S, *while avoiding* x, and so we will accept $(x, 1^k)$ when $k < c(|x|)$. If $k = c(|x|)$ then the only way we can guess k distinct strings of length at most $|x|$ together with proofs that they are all in S is if we have guessed *exactly* the set of strings in $|S|$ of length at most $|x|$. This set avoids x iff $x \notin S$. Thus we accept $(x, 1^k)$ iff $x \notin S$ when $k = c(|x|)$. Finally, if $k > c(|x|)$ then there is no way to guess k distinct strings of length at most $|x|$ together with proofs that they are in S, so we will reject $(x, 1^k)$ if $k > c(|x|)$. This completes the proof that $\hat{S} \in \mathbf{NP}$.

(d) Define the "candidate reduction" $R_k(\phi) = U(T(\phi), 1^k)$. From the previous part we know that $(T(\phi), 1^k) \in \hat{S}$ if $k < c(|T(\phi)|)$, regardless of what $T(\phi)$ is, so $R_k(\phi) \in S$ if $k < c(|T(\phi)|)$ as required. Similarly, $(T(\phi), 1^k) \notin \hat{S}$ if $k > c(|T(\phi)|)$, and so $R_k(\phi) \notin S$ if $k > c(|T(\phi)|)$. In the case that $k = c(|T(\phi)|)$, we have

$$(T(\phi), 1^k) \in \hat{S} \Leftrightarrow T(\phi) \notin S \Leftrightarrow \phi \notin \text{SAT} \Leftrightarrow \phi \in \overline{\text{SAT}}$$

and $R_k(\phi) = U(T(\phi), 1^k) \in S \Leftrightarrow (T(\phi), 1^k) \in \hat{S}$ since U is a reduction. We conclude that when $k = c(|T(\phi)|)$, $R_k(\phi) \in S \Leftrightarrow \phi \in \overline{\text{SAT}}$, as required.

(e) We need to address a technical issue first:

Lemma 2.1 *If language $L \subseteq \Sigma^*$ is* **NP***-complete, then language $L' \in (\Sigma \cup \{\#\})^*$ defined by*

$$L' = \{x\#^i : x \in L, i \geq 0\}$$

is **NP***-complete. If L is sparse then L' is sparse.*

Proof: It is clear that if L is in **NP** then L' is as well. And, a reduction from an arbitrary language $A \in \mathbf{NP}$ to L *is* a reduction from A to L'. Finally, note that the

number of strings in L' of length at most n is at most the number of strings in L of length at most n, times n. Thus L' is sparse if L is. ■

Thus we may assume without loss of generality that S has the property that $x \in S$ iff $x\#^i \in S$ for all $i > 0$. This allows us to assume that reduction T has the property that on all strings ϕ of length n, the length of $T(\phi)$ is the same: since T is a polynomial time reduction, there is some polynomial bound $r(n)$ on the length of strings it produces on an input of length n. We can modify T if necessary so that it pads its output with "#"s up to length *exactly* $r(n)$. By our assumption on S, this does not change whether the output of T is in S or not, so T is still a reduction.

Now, we proceed with the problem. Assume S is **NP**-complete, and let ϕ be an instance of SAT with $|\phi| = n$. Let T be the reduction from SAT to S, and let $q(n)$ be the polynomial bound on the length of a string output by R_k on an input of length n, for any $k \leq p(|T(\phi)|)$. We will run the procedure from part (b) once for each k between 1 and $p(|T(\phi)|)$, using R_k as a "candidate reduction" from $\overline{\text{SAT}}$ to S in the k-th run. By the previous part, when $k = c(|T(\phi)|)$ the "candidate reduction" R_k will *actually be a legitimate reduction from $\overline{SAT}$ to S for strings of length n*. Since we only apply R_k on strings of length n (possibly after the minor modification of the solution to part (b) mentioned in the problem) *and* using our assumption that T outputs strings whose length depends only on the length of T's input, our analysis in part (b) applies, and we will find a satisfying assignment after at most $(n+1)p(q(n)) + n$ steps if there is one. For the other runs with $k \neq c(|T(\phi)|)$ we don't care what happens; we just need to be sure to stop every run after $(n+1)p(q(n)) + n$ steps, which is enough to allow the "correct" run to finish, and ensure that each run takes polynomial time. Since we perform $p(q(n)) = \text{poly}(n)$ runs, each of which takes polynomial time, the overall procedure runs in polynomial time and decides SAT. Thus $\mathbf{P} = \mathbf{NP}$.

(f) Let S be the sparse language consisting of exactly one string z (it doesn't matter what string we choose). S is clearly in **NP**. If $\mathbf{P} = \mathbf{NP}$ then we can reduce every language L in **NP** to S in the following way: on input x, our reduction decides in polynomial time whether $x \in L$ (remember, we are assuming $\mathbf{P} = \mathbf{NP}$); if it is, then we output z; if not, we output any other string. This shows that S is **NP**-complete

3. We'll show that STRONGLY CONNECTED is **NL**-complete. First, we observe that it is in **NL**: we can go through all (ordered) pairs of vertices (x, y) and nondeterministically guess a path from x to y one vertex at a time (as we did for S-T CONNECTIVITY). We reject whenever one of these paths fails to actually reach y from x. If the graph is strongly connected, then some sequence of guesses will succeed, and we will accept; otherwise no sequence of guesses will succeed (in particular, there will be some pair (x, y) with no directed path from x to y), and all computation paths will reject.

   Now, we reduce S-T CONNECTIVITY (which we know to be **NL**-complete) to STRONGLY CONNECTED as follows. Given a directed graph $G = (V, E)$ and vertices s and t we produce the directed graph $G' = (V, E')$ where $E' = E \cup \{(v, s) : v \in V\} \cup \{(t, v) : v \in V\}$. We can easily perform this reduction in logspace, as we only need to step through the vertices one by one, adding the required two edges for each. We claim that G' is strongly connected if and only if there is a directed path from s to t in G. ($\Leftarrow$) If there is a directed path from s to t in G, then to get from v to v' in G' we can traverse the edge from v to s, then the directed

path from s to t, and finally the edge from t to v'. ($\Rightarrow$) If G' is strongly connected, then in particular there must be a directed path from s to t in G', and then there also must be a directed path from s to t in G, since we only added ingoing edges to s and outgoing edges to t (which cannot have introduced a directed path from s to t if there was not one to begin with).

By the claim, we have reduced S-T CONNECTIVITY to STRONGLY CONNECTED in logspace, and thus STRONGLY CONNECTED is **NL**-complete. Therefore, if STRONG CONNECTED is in **L**, then **NL** = **L**. Note that for this conclusion it would have been sufficient to only prove that STRONGLY CONNECTED is **NL**-hard.