University of Maryland
CMSC652 — Complexity Theory
Professor Jonathan Katz

Problem Set 2 — Solutions

1. Given an $\mathcal{NP}$-complete language L such that

$$L = \{x \mid \exists y : \ (x, y) \in R_L\}$$

(for some appropriate R_L), define the $\mathcal{NP}$ language language L_{pre} as follows:

$$L_{\text{pre}} = \{(x, y_1) \mid \exists y_2 : \ (x, y_1 \circ y_2) \in R_L\}$$

(where $\circ$ denotes concatenation). Note that $(x, y_1) \in L_{\text{pre}}$ iff y_1 is a prefix of a witness $y = y_1 \circ y_2$ that $x \in L$. Since L is $\mathcal{NP}$-complete, there is a Karp reduction f from L_{pre} to L.

We now describe how to reduce the *search* problem for L to the *decision* problem for L. Assume we are given access to an oracle solving the decision problem for L. Then, on input x, proceed as follows (we assume for simplicity that the witness y — if one exists — has length exactly $n \stackrel{\text{def}}{=} |x|$, but the algorithm can be suitably modified when this is not the case):

Set $y := \varepsilon$ (the empty string)
If $f(x, y) \notin L$ then reject
Else, for $i = 1$ to n:
    If $f(x, y0) \in L$ set $y := y0$
    Else set $y := y1$
Output y

2. We have already seen the solution to this problem in class.

3. (**optional**—**unsolved problem**) Sorry, I don't have a solution either. The best I can do is refer you to: `http://en.wikipedia.org/wiki/Sudoku`.

4. (a) Let co$\mathcal{NP}$ refer to the first definition (i.e., co$\mathcal{NP} = \left\{L \mid \bar{L} \in \mathcal{NP}\right\}$), and let co$\mathcal{NP}'$ refer to the second definition (i.e., $L \in$ co$\mathcal{NP}'$ if there exists a poly-time computable [and polynomially-bounded] relation R such that $x \in L \Leftrightarrow \forall y : R(x, y)$).[1] We prove equivalence.
   Let $L \in$ co$\mathcal{NP}$. Then $\bar{L} \in \mathcal{NP}$ by definition. So there exists a poly-time relation $R_{\bar{L}}$ such that $x \in \bar{L} \Leftrightarrow \exists y : \ (x, y) \in R_{\bar{L}}$. This means that

$$x \in L \Leftrightarrow \forall y : \ (x, y) \notin R_{\bar{L}}.$$

[1] Technically speaking, "$\forall y$" does not quantify over *all* strings y, but only all strings y of at most some bounded (polynomial) length depending on the relation. We omit this for clarity.

Let R_L be the complementary relation to $R_{\bar{L}}$ (i.e., $x \in R_L$ iff $x \notin R_{\bar{L}}$). Then R_L is poly-time computable (and polynomially-bounded; see the footnote); also:

$$x \in L \Leftrightarrow \forall y : \ (x, y) \in R_L.$$

So $L \in \text{co}\mathcal{NP}'$.

Now let $L \in \text{co}\mathcal{NP}'$. Then we know there exists a poly-time relation R_L such that $x \in L \Leftrightarrow \forall y : (x, y) \in R_L$. So, $x \in \bar{L}$ iff $\exists y : (x, y) \notin R_L$. Letting $R_{\bar{L}}$ be the complementary relation to R_L, we have

$$x \in \bar{L} \Leftrightarrow \exists y : \ (x, y) \in R_{\bar{L}}.$$

Since $R_{\bar{L}}$ is poly-time computable (and poly-bounded), we have $\bar{L} \in \mathcal{NP}$ and so $L \in \text{co}\mathcal{NP}$.

(b) Let $L' \in \text{co}\mathcal{NP}$. Then $\bar{L}' \in \mathcal{NP}$. So there exists a Karp reduction f from $\bar{L}'$ to our $\mathcal{NP}$-complete language L such that $x \in \bar{L}' \Leftrightarrow f(x) \in L$. To determine whether $x \in L'$ given oracle access to L, simply compute $f(x)$ and ask whether $f(x) \in L$. If it is, output "no"; if not, output "yes." Correctness follows easily.

This is *not* a Karp reduction because here

$$x \in L' \Leftrightarrow f(x) \notin L.$$

That is, we *negate* the answer received from the oracle. In a Karp reduction we must return the same answer we get from the oracle.

(c) Say there exists a language $L \in \mathcal{NP}$ such that there is a Karp reduction from every language $L' \in \text{co}\mathcal{NP}$ to L. So for any $L' \in \text{co}\mathcal{NP}$ we have a poly-time function $f_{L'}$ such that

$$x \in L' \Leftrightarrow f_{L'}(x) \in L.$$

Let R_L be the relation for L. Then

$$x \in L' \Leftrightarrow \exists y : \ (f_{L'}(x), y) \in R_L.$$

Define relation R'_L as follows:

$$(x, y) \in R'_L \Leftrightarrow (f_{L'}(x), y) \in R_L.$$

Note that R'_L is poly-time computable and polynomially-bounded; also,

$$x \in L' \Leftrightarrow \exists y : \ (x, y) \in R'_L.$$

So, $L' \in \mathcal{NP}$ and we conclude that $\text{co}\mathcal{NP} \subseteq \mathcal{NP}$. Applying the next problem shows that $\text{co}\mathcal{NP} = \mathcal{NP}$.

Note that in part (b) we showed a *Turing* reduction from every language in $\text{co}\mathcal{NP}$ to a language in $\mathcal{NP}$. But it is widely believed that $\mathcal{NP} \neq \text{co}\mathcal{NP}$, so the result we just proved does *not* hold for Turing reductions (at least as far as we know). It is good to understand where the difference arises.

5. Assume co$\mathcal{C} \subseteq \mathcal{C}$. We want to show that $\mathcal{C} \subseteq$ co$\mathcal{C}$. So, let $L \in \mathcal{C}$. Then $\bar{L} \in$ co$\mathcal{C}$ (by definition), which implies $\bar{L} \in \mathcal{C}$ (by our assumption that co$\mathcal{C} \subseteq \mathcal{C}$). So $L \in$ co$\mathcal{C}$ (by definition), and we are done.

6. (*Note*: in class I gave an alternate solution for the case of co$\mathcal{NP}$ in terms of *proofs of non-membership* which I find more intuitive. But many students seemed to find the following approach more intuitive, and it has the advantage of unifying all the solutions.)

   For $\mathcal{P}$, $\mathcal{NP}$, and co$\mathcal{NP}$ we will use the same algorithm; just the analysis will be different. (For $\mathcal{NP}$ there is also a much easier solution.) For a string x, let x_i denote the i^{th} character in x and let $x_{i \to j}$ denote the substring of x from position i to j (inclusive). (When $j < i$ then by convention $x_{i \to j}$ is the empty string.) Given an algorithm M_L (of the appropriate type) for L, we decide L^* as follows (we use M_L each time we want to know whether a given string is in L):

   On input x of length n do:
   If $n = 0$ then output "accept" (the empty string is always in L^*)
   Maintain an array $A[0], \ldots, A[n]$, initially all 0
   Set $A[0] := 1$
   For $i = 1$ to n:
   For $j = 0$ to $i - 1$:
   If ($A[j] = 1$ and $M_L(x_{j+1 \to i})$ outputs 1), then $A[i] := 1$
   If $A[n] = 1$ output "accept", else output "reject"

   The intuition is that $A[i] = 1$ iff $x_{1 \to i} \in L^*$ (although, strictly speaking, this intuition is only correct when $L \in \mathcal{P}$; see below). It is clear that the above always runs in polynomial time. Let us prove correctness in each case:

   $\underline{L \in \mathcal{P}}$: In this case, M_L always returns the correct answer. We claim that at the end of the i^{th} iteration of the outer loop, the following holds: for all $0 \leq i' \leq i$, $A[i'] = 1$ iff $x_{1 \to i'} \in L^*$ (correctness of the algorithm follows immediately). It can be verified by inspection that this is true at the end of the first iteration of the outer loop. So assume it is true at the end of the $(i-1)^{\text{th}}$ iteration of the outer loop, and we will prove it holds at the end of the i^{th} iteration.

   - If $x_{1 \to i} \in L^*$, then there exists some j (with $0 \leq j < i$) such that $x_{1 \to j} \in L^*$ and $x_{j+1 \to i} \in L$. By our inductive assumption this means that $A[j] = 1$ and $x_{j+1 \to i} \in L$. In this case $A[i]$ is indeed set to 1 when the inner loop reaches this value of j.
   - If $A[i]$ gets set to 1, then at some point in the inner loop we have found a value j such that $A[j] = 1$ and $x_{j+1 \to i} \in L$. By our inductive assumption this means that $x_{1 \to j} \in L^*$; this then implies that x_i is indeed in L^*, so $A[i]$ is set correctly.

   $\underline{L \in \mathcal{NP}}$: Our algorithm M_L now has the following weaker guarantees: when $x \in L$ then there is an execution of $M_L(x)$ that outputs 1; when $x \notin L$, then $M_L(x)$ always outputs 0. On the other hand we also need only prove similar guarantees about our

above algorithm. Define an array $B[0], \ldots B[n]$ such that $B[i] = 1$ iff $x_{1\to i} \in L^*$. We claim now that at the end of the i^{th} iteration of the outer loop, the following holds: (1) it is always the case that for all $i' \leq i$, if $B[i'] = 0$ then $A[i'] = 0$; and (2) there is an execution of the algorithm such that $A[i'] = B[i']$ for all $i' \leq i$. (Note that correctness follows once we prove this.) As before, the base case $i = 1$ can be verified by inspection. So assume the above are true at the end of the $(i-1)^{\text{th}}$ iteration of the outer loop, and we will prove they hold at the end of the i^{th} iteration.

- If $A[i]$ gets set to 1 in any execution, then at some point in the inner loop we have found a value j such that $A[j] = 1$ and $M_L(x_{j+1\to i})$ outputs 1. By our inductive assumption, $A[j] = 1$ always implies $B[j] = 1$ and so $x_{1\to j} \in L^*$. Furthermore, we must have $x_{j+1\to i} \in L$. So indeed $x_{1\to i} \in L^*$. (This proves that if $B[i] = 0$ then $A[i] = 0$ in every execution.)
- Assume $B[i] = 1$. By our inductive assumption there is some execution in which $A[i'] = B[i']$ for all $i' < i$. Let us focus only on such executions. Since $x_{1\to i} \in L^*$, there exists some j (with $0 \leq j < i$) such that $x_{1\to j} \in L^*$ and $x_{j+1\to i} \in L$. So $A[j] = 1$ in the executions where everything goes right, and furthermore there is some execution of $M_L(x_{j+1\to i})$ that outputs 1. Taken together, this means there is some execution of our algorithm that results in $A[i]$ being set to 1.

$\underline{L \in \text{co}\mathcal{NP}}$: Our algorithm M_L now has the following guarantees: when $x \in L$ then $M_L(x)$ always outputs 1; when $x \notin L$, then there is an execution of $M_L(x)$ that outputs 0. Define an array $B[0], \ldots B[n]$ exactly as before. We claim now that at the end of the i^{th} iteration of the outer loop, the following holds: (1) it is always the case that for all $i' \leq i$, if $B[i'] = 1$ then $A[i'] = 1$; and (2) there is an execution of the algorithm such that $A[i'] = B[i']$ for all $i' \leq i$. (Note that correctness follows once we prove this.) As before, the base case $i = 1$ can be verified by inspection. So assume the above are true at the end of the $(i-1)^{\text{th}}$ iteration of the outer loop, and we will prove they hold at the end of the i^{th} iteration.

- Assume $B[i] = 1$, so $x_{1\to i} \in L^*$. Then there exists some j (with $0 \leq j < i$) such that $x_{1\to j} \in L^*$ (and hence $B[j] = 1$) and $x_{j+1\to i} \in L$. By our inductive assumption, $A[j] = 1$ in all executions. Furthermore, $M_L(x_{j+1\to i})$ always outputs 1. So $A[i]$ always gets set to 1 and condition (1) holds.
- Assume $A[i]$ gets set to 1 in every execution. We want to show that this implies $B[i] = 1$. By our inductive assumption there is some execution in which $A[i'] = B[i']$ for all $i' < i$; let us focus only on such executions. If $A[i]$ gets set to 1 in every execution, that means it gets set to 1 even in executions where everything has gone right so far *and* M_L gets the right answer every time in the current iteration of the loop. But then we have found a value j with $A[j] = 1$ (hence $B[j] = 1$ and $x_{1\to j} \in L^*$) and where $M_L(x_{j+1\to i})$ outputs 1 (hence $x_{j+1\to i} \in L$). But this means $x_{1\to 1} \in L$ and so $B[i] = 1$.