Computability and Complexity

Assignment 3

Ari Feiglin

Exercise 3.1:

We define the class of **almost coNP** problems as follows: a decision problem S is in **Almost-coNP** if and only if there exists a polynomial-time algorithm V and a polynomial p such that for every x, $x \in S$ if and only if for every y whose length is at most $p(|x|)$ other than one satisfies $V(x, y) = 1$.

Show that **NP** = **Almost-coNP** if and only if **NP** = **coNP**.

Firstly we will show that **coNP** $\subseteq$ **Almost-coNP**. Suppose $S \in$ **coNP**, then there exists a polynomial-time algorithm V and a polynomial p such that for every x,

$$x \in S \iff \forall y\big(V(x, y) = 1\big)$$

where $|y| \leq p(|x|)$. For the sake of conciseness, I will leave out explicitly stating that the binary strings being discussed have a length bound by $p(|x|)$. We can assume that for every x, $V(x, \varepsilon) = 1$. Otherwise, we could define $V_0(x, y)$, where $V_0(x, \varepsilon) = 1$ and $V_0(x, 0)$ is equal to zero if $V(x, 0) = 0$ and otherwise is equal to $V(x, \varepsilon)$. For every other y, $V_0(x, y) = V(x, y)$. Then if $x \in S$, for every y, $V_0(x, y) = 1$ (for $y = 0$, $V(x, 0) = 1$ and $V(x, \varepsilon) = 1$). And if $x \notin S$, then there exists a y such that $V(x, y) = 0$, if this $y = \varepsilon$ then $V_0(x, 0) = 0$ and otherwise $V_0(x, y) = 0$. So if $x \notin S$ then there exists a y where $V_0(x, y) = 0$, so

$$x \in S \iff \forall y\big(V_0(x, y) = 1\big)$$

and $V_0(x, \varepsilon) = 1$ as required.

So, now assuming that $V(x, \varepsilon) = 1$, let us define the algorithm $V'(x, y)$, where if $y = \varepsilon$, then return zero. Otherwise return $V(x, y)$. Then if $x \in S$, for every $y \neq \varepsilon$, $V'(x, y) = V(x, y) = 1$. And if $x \notin S$, there exists a y such that $V(x, y) = 0$ and since y couldn't be ε, this means there are two ys where $V'(x, y) = 0$ (for this y and for $y = \varepsilon$). Thus $x \in S$ if and only if for every y other than one (which is $y = \varepsilon$), $V'(x, y) = 1$. So by definition, $S \in$ **Almost-coNP**.

Thus **coNP** $\subseteq$ **Almost-coNP**. So if **NP** = **Almost-coNP**, then **coNP** $\subseteq$ **NP**, and so **NP** = **coNP**.

For the other direction, notice that if $S \in$ **Almost-coNP** then there exists a polynomial-time algorithm V and a polynomial p which satisfy the conditions of the definitions of **Almost-coNP**. Let us define another algorithm $V'(x, y_1, y_2)$ which does the following:

(1) If $y_1 = y_2$ then return one.

(2) Otherwise if $V(x, y_1) = 0$ and $V(x, y_2) = 1$, return one.

(3) Else return zero.

We claim that

$$x \in S \iff \exists y_1 \forall y_2\big(V'(x, y_1, y_2) = 1\big)$$

(with the conditions that the strings's lengths be bound by $p(|x|)$.) Thus this would mean $S \in \Sigma_2$, and so **Almost-coNP** $\subseteq \Sigma_2$.

So, if $x \in S$ then there exists a binary string y_1 such that for every other binary string y_2, $V(x, y_2) = 1$. Thus for every y_2, either $y_1 = y_2$ and so $V'(x, y_1, y_2) = 1$, or $y_1 \neq y_2$ and so $V(x, y_1) = 0$ and $V(x, y_2) = 1$ and so $V(x, y_1, y_2) = 1$. Thus we have shown

$$x \in S \implies \exists y_1 \forall y_2\big(V'(x, y_1, y_2) = 1\big)$$

Now, suppose the converse: that there exists a y_1 such that for every y_2, $V'(x, y_1, y_2)$. This means that for every y_2, either $y_1 = y_2$ or $V(x, y_1) = 0$ and $V(x, y_2) = 1$. Thus $V(x, y_1) = 0$ and for every other binary string y_2, $V(x, y_2) = 1$. And by the definition of V, this means that $x \in S$. So we have shown the equivalence, and thus $S \in \Sigma_2$.

So we have shown that **Almost-coNP** $\subseteq \Sigma_2$. Now, if **NP** = **coNP**, then the polynomial hierarchy collapses to **NP** $= \Sigma_1$, and in particular **NP** $= \Sigma_1 = \Sigma_2$, so **Almost-coNP** $\subseteq$ **NP**. Since **coNP** $\subseteq$ **Almost-coNP**, we have

$$\mathbf{NP} = \mathbf{coNP} \subseteq \mathbf{Almost\text{-}coNP} \subseteq \mathbf{NP} \implies \mathbf{Almost\text{-}coNP} = \mathbf{NP}$$

Thus we have shown that **NP** = **coNP** if and only if **Almost-coNP** = **NP**.

Exercise 3.2:

For the following statements, either prove, disprove, or show that they are equivalent to an open question:

(1) $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-hard.

(2) $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-complete.

(1) In order to show that $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-hard, we must show that it is both $\mathbf{NP}$ and $\mathbf{coNP}$-hard. Let us first show that it is $\mathbf{coNP}$-hard.

We know that

$$\overline{\mathsf{Clique}} = \{(G, k) \mid G \text{ is a graph where every clique's size is at most } k\}$$

is $\mathbf{coNP}$-complete. This is because Clique is $\mathbf{NP}$-complete, and its complement is equal to

$$\overline{\mathsf{Clique}} \cup \{\omega \mid \text{The binary string } \omega \text{ does not represent a graph and a natural number}\}$$

So this is $\mathbf{coNP}$-complete. But the set of binary strings which do not represent a graph and a natural number is in $\mathbf{P}$, and thus $\overline{\mathsf{Clique}}$ is $\mathbf{coNP}$-complete as well. We can generalize this in the following lemma:

Lemma:

Let $\mathcal{C}$ be a class of decision problems, and let $S_{\mathbf{P}} \in \mathbf{P}$. Then if a decision problem of the form $S \cup S_{\mathbf{P}}$ is $\mathcal{C}$-hard then S is also $\mathcal{C}$-hard.

Proof:

If $S \cup S_{\mathbf{P}}$ is $\mathcal{C}$-hard, let $S' \in \mathcal{C}$, then there exists a Karp reduction from S' to $S \cup S_{\mathbf{P}}$, let this be f. Let us choose $a \in S$, and so let us define

$$f'(x) = \begin{cases} a & f(x) \in S_{\mathbf{P}} \\ f(x) & f(x) \notin S_{\mathbf{P}} \end{cases}$$

This can be computed in polynomial time, since computing $f(x)$ and checking if it is in $S_{\mathbf{P}}$ both take polynomial time. And if $x \in S'$, then $f(x) \in S \cup S_{\mathbf{P}}$, so if $f(x) \in S_{\mathbf{P}}$, $f'(x) = a \in S$. Otherwise $f(x) \in S$, so $f'(x) = f(x) \in S$. And if $x \notin S'$, then $f(x) \notin S \cup S_{\mathbf{P}}$, so $f'(x) = f(x) \notin S$. Thus $x \in S'$ if and only if $f'(x) \in S$, so f' is a Karp reduction from S' to S.

Thus S is $\mathcal{C}$-hard, as required. ■

Now, we can define a Karp reduction from $\overline{\mathsf{Clique}}$ to $\mathsf{MaxClique}$: given an input (G, k) for $\overline{\mathsf{Clique}}$, we define a graph G' which takes G and adds a clique of k new vertices. These new vertices are not connected to the graph G. Let this new graph be G', and we claim that $(G, k) \mapsto (G', k)$ is a Karp reduction from $\overline{\mathsf{Clique}}$ to $\mathsf{MaxClique}$.

If $(G, k) \in \overline{\mathsf{Clique}}$, then every clique in G is of size $\leq k$, and so the clique of size k which we added in G' becomes the maximum clique, so $(G', k) \in \mathsf{MaxClique}$. And if $(G', k) \in \mathsf{MaxClique}$, then the maximum clique size in G' is k, and since G is a subgraph of G' this means that every clique in G' is at most k vertices, meaning $(G, k) \in \overline{\mathsf{Clique}}$. Thus $(G, k) \in \overline{\mathsf{Clique}}$ if and only if $(G', k) \in \mathsf{MaxClique}$, as required.

Since $\overline{\mathsf{Clique}}$ is $\mathbf{coNP}$-hard, this means that $\mathsf{MaxClique}$ is also $\mathbf{coNP}$-hard.

Now, all that remains is to show that $\mathsf{MaxClique}$ is $\mathbf{NP}$-hard. We will do this by first defining the decision problem

$$\mathsf{MaxIS} = \{(G, k) \mid G \text{ is an undirected graph whose maximal independent set is of size } k\}$$

Notice how S is an independent set in the graph $G = (V, E)$, if and only if S is a clique in the graph $G' = (V, E^c)$. This is because if S is an independent set in G, then for every $u, v \in S$, $\{u, v\} \notin E$ so $\{u, v\} \in E^c$, meaning S is a clique in G'. And similar for the converse. Thus $(G, k) \mapsto (G', k)$ is a Karp reduction from MaxIS to $\mathsf{MaxClique}$ (it is also a Karp reduction from $\mathsf{MaxClique}$ to MaxIS), so it is sufficient to show that MaxIS is $\mathbf{NP}$-hard.

Now, it turns out that in recitation 2, we constructed a Karp reduction from SAT to MaxIS, but we posed it as a Karp reduction from SAT to IS. I will define the Karp reduction again here, and show that it is indeed a Karp reduction from SAT to MaxIS.

Let φ be a boolean formula in CNF, then we define the graph G as follows:

(i) For every disjunction D_i, for every variable x_j which occurs in D_i, we add the vertex u_{ij}.

(ii) If u_{ij_1} and u_{ij_2} are vertices in the graph G, then they refer to different variables which both occur in the same disjunction, so we add an edge $\{u_{ij_1}, u_{ij_2}\}$.

(iii) If u_{i_1j} and u_{i_2j} are vertices in the graph G, then they refer to the same variable which occurs in different disjunctions. We add an edge $\{u_{i_1j}, u_{i_2j}\}$ if and only if the sign of the occurrences differ (eg. if x_j occurs in D_{i_1} and $\neg x_j$ occurs in D_{i_2}).

Suppose φ has m disjunctions, then we claim that $\varphi \mapsto (G, m)$ is a Karp reduction from SAT to MaxIS.

If $\varphi \in \mathsf{SAT}$, then there exists a boolean vector τ which satisfies φ. Every disjunction D_i must be satisfied by some variable x_j (meaning there exists a variable which occurs in D_i whose sign is the same as how it occurs in τ; if τ_j is true, then x_j occurs in D_i, and if τ_j is false, then $\neg x_j$ occurs in D_i). So choose a variable x_j which satisfies D_i and place u_{ij} into the set S.

We now claim that S is the maximum independent set, and it is of size m. Firstly, it is an independent set since if $u_{i_1j_1}$ and $u_{i_2j_2}$ are in S, suppose there exists an edge between them. By the definition of G', either $i_1 = i_2$ or $j_1 = j_2$, but since for every D_i we are choosing a single variable x_i to place into S, $i_1 \neq i_2$. So $j = j_1 = j_2$, but u_{i_1j} refers to the variable x_j in the disjunction D_{i_1} and u_{i_2j} refers to the same variable x_j in a different disjunction D_{i_2}. By definition, the edge only exists if x_j has differing signs in D_{i_1} and D_{i_2}, and so they cannot both be satisfied by x_j, in contradiction. So S is indeed an independent set.

Since we place a vertex from each disjunction into S, $|S| = m$. Now, we claim that every independent set in G' must have a size which is at most m. Otherwise, suppose S' is an independent set where $|S'| > m$, then S' would have two vertices from the same disjunction (by the pigeonhole principle), but vertices from the same disjunction have an edge between them, contradicting S' being an independent set. So S is a maximum independent set, meaning the size of the maximum independent set in G is m, so $(G, m) \in \mathsf{MaxIS}$.

Now, if $(G, m) \in \mathsf{MaxIS}$, then G has an independent set S of size m. As stated before, S cannot have two vertices from the disjunction, so S contains exactly one vertex from each disjunction. We define the boolean vector τ as follows: for each $u_{ij} \in S$, set τ_j to be the sign of x_j in D_i: if x_j occurs in D_i, then set τ_j to true, and if $\neg x_j$ occurs in D_i then set τ_j to false. For every other index of τ, set it arbitrarily.

We must prove that this construction of τ well-defined, as if u_{i_1j} and u_{i_2j} are both in S, they both set τ_j. But, since S is independent, there is no edge between u_{i_1j} and u_{i_2j}, so x_j occurs with the same sign in both D_{i_1} and D_{i_2}, so τ_j is set to the same value. So τ is well-defined.

τ satisfies φ, since for every D_i there exists some $u_{ij} \in S$, as stated before. Then τ_j is set such that x_j satisfies D_i, and so D_i is satisfied by τ. So every disjunction is satisfied by τ, and therefore φ is satisfied by τ, meaning $\varphi \in \mathsf{SAT}$.

Thus $\varphi \in \mathsf{SAT}$ if and only if $(G, m) \in \mathsf{MaxIS}$. Therefore $\varphi \mapsto (G, m)$ is a Karp reduction from SAT to MaxIS, meaning MaxIS is $\mathbf{NP}$-hard. Since we showed a Karp reduction from MaxIS to $\mathsf{MaxClique}$, this means $\mathsf{MaxClique}$ is $\mathbf{NP}$-hard. And since we already showed $\mathsf{MaxClique}$ is $\mathbf{coNP}$-hard, this means $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-hard, as required.

(2) $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-complete if and only if it is in $\mathbf{NP}$ or $\mathbf{coNP}$. Now, suppose $\mathsf{MaxClique} \in \mathbf{NP}$ and let $S \in \mathbf{coNP}$, since $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-hard, there exists a Karp reduction from S to $\mathsf{MaxClique}$. Since $\mathbf{NP}$ is closed under Karp reductions, this means that $S \in \mathbf{NP}$, meaning $\mathbf{coNP} \subseteq \mathbf{NP}$, which means that $\mathbf{NP} = \mathbf{coNP}$. Similarly, if $\mathsf{MaxClique} \in \mathbf{coNP}$, let $S \in \mathbf{NP}$ then there exists a Karp reduction from S to $\mathsf{MaxClique}$ and since $\mathbf{coNP}$ is closed under Karp reductions, $S \in \mathbf{coNP}$. Therefore $\mathbf{NP} \subseteq \mathbf{coNP}$ and so $\mathbf{NP} = \mathbf{coNP}$.

So we have shown that if $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-complete, then $\mathbf{NP} = \mathbf{coNP}$. We will now show the converse. Since $\mathbf{NP} = \mathbf{coNP}$, this means that the polynomial hierarchy collapses to $\mathbf{NP}$, ie. $\mathbf{PH} = \mathbf{NP} = \mathbf{coNP}$ and in particular $\Sigma_2 = \mathbf{NP}$. Since $\mathsf{MaxClique} \in \Sigma_2$ this means $\mathsf{MaxClique} \in \mathbf{NP} = \mathbf{NP} \cup \mathbf{coNP}$, so $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-complete.

Therefore $\mathsf{MaxClique}$ is $(\mathbf{NP} \cup \mathbf{coNP})$-complete if and only if $\mathbf{NP} = \mathbf{coNP}$.

Exercise 3.3:

Show that for every $k \geq 1$, there exists a Σ_k-complete problem.

We will show this inductively. For $k = 1$, this means we must show there exists an **NP**-complete problem, which we know exist (eg. SAT). Now suppose there exists a Σ_k-complete problem, S_k. We define S_{k+1} as follows:

$$S_{k+1} = \left\{ (M^{S_k},\, \omega, 1^t) \;\middle|\; \begin{array}{l} M^{S_k} \text{ is a non-deterministic oracle machine which uses an oracle for } S_k \text{, and } \omega \text{ is a binary} \\ \text{input which is accepted } M^{S_k} \text{ within } t \text{ steps.} \end{array} \right\}$$

By "ω is accepted by M^{S_k} within t steps" we mean that there exists a run of M^{S_k} on ω in which ω is accepted within t steps. We will now show that S_{k+1} is Σ_{k+1}-complete.

Firstly, S_{k+1} is in $\mathbf{NP}^{S_k}$, as we can define a non-deterministic oracle machine V^{S_k} which accepts S_{k+1}. Given an input $(M^{S_k}, \omega, 1^t)$, V^{S_k} runs M^{S_k} non-deterministically on ω for at most t steps. V^{S_k} will utilize its own oracle of S_k to answer M^{S_k}'s queries. If M^{S_k} accepts ω within t steps, then V^{S_k} will accept its input, and otherwise it will reject.

V^{S_k} is polynomial, as it runs in $t = |1^t|$ time. Furthermore it decides S_{k+1}, as it will only accept $(M^{S_k}, \omega, 1^t)$ if there exists a run of M^{S_k} on ω whose length is at most t in which M^{S_k} accepts ω. This is precisely the definition of S_{k+1}. Since V^{S_k} is a non-deterministic oracle machine of S_k, this means that $S_{k+1} \in \mathbf{NP}^{S_k} \subseteq \mathbf{NP}^{\Sigma_k} = \Sigma_{k+1}$ as required.

Now, we claim that $\Sigma_{k+1} = \mathbf{NP}^{S_k}$. Obviously $\mathbf{NP}^{S_k}$ is contained within Σ_{k+1}, so we must show the other direction.

Let $S \in \Sigma_{k+1} = \mathbf{NP}^{\Sigma_k}$, then there exists an $S' \in \Sigma_k$ and a non-deterministic polynomial-time oracle machine $A^{S'}$ which decides S. Since S_k is Σ_k-complete, there exists a Karp reduction from S' to S_k, let it be f, meaning $x \in S'$ if and only if $f(x) \in S_k$. Then we can define $A^{S_k}(x)$ which runs $A^{S'}(x)$ but when $A^{S'}$ performs a query of the form $q \overset{?}{\in} S'$, A^{S_k} will instead perform the equivalent query $f(q) \overset{?}{\in} S_k$ using its oracle. Since f can be computed in polynomial time, A^{S_k} still takes polynomial time. A^{S_k} decides S as well, as its run is equivalent to that of $A^{S'}$'s, and so $S \in \mathbf{NP}^{S_k}$.

Therefore $\Sigma_{k+1} \subseteq \mathbf{NP}^{S_k}$ and so $\Sigma_{k+1} = \mathbf{NP}^{S_k}$ as required.

Now, we will finally show that S_{k+1} is Σ_{k+1}-complete. Let $S \in \Sigma_{k+1}$, as we showed above this means $S \in \mathbf{NP}^{S_k}$ so there exists a polynomial-time non-deterministic oracle machine A^{S_k} which decides S. Let A^{S_k}'s runtime be bound by the polynomial p, and so we define the Karp reduction f from S to S_{k+1} by

$$f(x) = \left(A^{S_k},\, x,\, 1^{p(|x|)}\right)$$

Now $x \in S$ if and only if there exists a run of $A^{S_k}(x)$ which accepts x. Since the runtime of $A^{S_k}(x)$ is bound by $p(|x|)$, $x \in S$ if and only if $A^{S_k}(x)$ accepts x within $p(|x|)$ time, which is if and only if $f(x) \in S_{k+1}$. Thus $x \in S$ if and only if $f(x) \in S_{k+1}$, so f is indeed a Karp reduction from S to S_{k+1}.

Therefore S_{k+1} is both in Σ_{k+1} and Σ_{k+1}-hard, meaning it is Σ_{k+1}-complete. Therefore by induction we have shown that for every $k \geq 1$, there exists a Σ_k-complete problem, as required.

Exercise 3.4:

We will define an alternative polynomial hierarchy as follows:

$$\begin{aligned}\Sigma'_0 &= \mathbf{P}\\ \Sigma'_1 &= \mathbf{NP} \cap \mathbf{coNP}\\ \forall k \geq 1\colon \Sigma'_{k+1} &= \mathbf{NP}^{\Sigma'_k}\\ \mathbf{PH}' &= \bigcup_{k=0}^{\infty} \Sigma'_k\end{aligned}$$

Prove, disprove, or show that this is equivalent to an open question:

$$\mathbf{PH} = \mathbf{PH}'$$

We will prove that $\mathbf{PH} = \mathbf{PH}'$. Firstly, since $\mathcal{C} \subseteq \mathbf{NP}^{\mathcal{C}}$, Σ'_k is increasing.

We will show inductively that for every $k \geq 0$,

$$\Sigma_k \subseteq \Sigma'_{k+1} \subseteq \Sigma_{k+1}$$

For $k = 0$, this becomes

$$\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP} \subseteq \mathbf{NP}$$

which is true. Now suppose this is true for k, we will show it for $k+1$. This is true since

$$\Sigma_{k+1} = \mathbf{NP}^{\Sigma_k} \subseteq \mathbf{NP}^{\Sigma'_{k+1}} = \Sigma'_{k+2} \subseteq \mathbf{NP}^{\Sigma_{k+1}} = \Sigma_{k+2}$$

as required.

Since Σ'_k is increasing, we can start the union for $\mathbf{PH}'$ at any index. Therefore

$$\mathbf{PH}' = \bigcup_{k=0}^{\infty} \Sigma'_k = \bigcup_{k=1}^{\infty} \Sigma'_k \supseteq \bigcup_{k=1}^{\infty} \Sigma_{k-1} = \bigcup_{k=0}^{\infty} \Sigma_k = \mathbf{PH}$$

meaning $\mathbf{PH}' \supseteq \mathbf{PH}$. And on the other hand

$$\mathbf{PH}' \subseteq \bigcup_{k=0}^{\infty} \Sigma_k = \mathbf{PH}$$

so $\mathbf{PH}' \subseteq \mathbf{PH}$, meaning $\mathbf{PH}' = \mathbf{PH}$, as required.