Computability and Complexity

Lecture 7, Tuesday August 22, 2023

Ari Feiglin

Proposition 7.1:

If $\mathbf{NP} \neq \mathbf{coNP}$ then $\mathbf{NP}$ is not closed under Cook reductions.

Proof:

We can always define a Cook reduction from S to S^c (given an oracle for S^c, return its negation). So if $\mathbf{NP}$ were closed under Cook reductions, given a problem S in $\mathbf{NP}$ we can define a reduction from $S^c \in \mathbf{coNP}$ to S, and then $S^c \in \mathbf{NP}$. This would mean $\mathbf{NP} \subseteq \mathbf{coNP}$ and so $\mathbf{NP} = \mathbf{coNP}$, in contradiction. ■

Proposition 7.2:

A decision problem S is in $\mathbf{coNP}$ if and only if there exists a polynomial time algorithm V such that $x \in S$ if and only if for all y with $|y| \leq p(|x|)$, $V(x, y) = 1$.

Proof:

If $S \in \mathbf{coNP}$ then $S^c \in \mathbf{NP}$ so it has a polynomial time verifier V' and a polynomial p which satisfy the conditions for $\mathbf{NP}$. Let us define $V(x, y) = 1 - V'(x, y)$, and so if $x \in S$ then $x \notin S$ so $V'(x, y) = 0$ for all y, so $V(x, y) = 1$. And if $x \notin S$ then $x \in S^c$ so there exists a y with $|y| \leq p(|x|)$ such that $V'(x, y) = 1$ and so $V(x, y) = 0$.

And if this condition holds, then let us define $V'(x, y) = 1 - V(x, y)$. Then we claim this is a verifier for S^c. If $x \in S^c$ then by the condition there exists a y with $|y| \leq p(|x|)$ and $V(x, y) = 0$ so $V'(x, y) = 1$ as required. And if $x \notin S^c$ then $x \in S$ so for every y with $|y| \leq p(|x|)$, $V(x, y) = 1$ and so $V'(x, y) = 0$. If $|y| > p(|x|)$, we can change V' so that it returns zero, and this becomes a polynomial proof system for S^c, so $S^c \in \mathbf{NP}$ meaning $S \in \mathbf{coNP}$. ■

Definition 7.3:

For $k \geq 0$, we say that a decision problem S is in Σ_k if there exists a polynomial p and a polynomial-time algorithm V such that (let $\mathsf{Q}_{2i} = \forall$ and $\mathsf{Q}_{2i+1} = \exists$)

$$x \in S \iff \mathsf{Q}_1 |y_1| \leq p(|x|)\ \mathsf{Q}_2 |y_2| \leq p(|x|)\ \cdots\ \mathsf{Q}_k |y_k| \leq p(|x|) \big(V(x, y_1, \ldots, y_k) = 1\big)$$

The **polynomial hierarchy** is defined to be

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

Notice then that $\Sigma_0 = \mathbf{P}$ and $\Sigma_1 = \mathbf{NP}$.

Example 7.4:

Let us define the decision problem

$$\mathsf{MaxClique} = \{(G, k) \mid G \text{ is an undirected graph whose maximum clique size is } k\}$$

Let us define an algorithm V which takes input $\big((G, k), S_1, S_2\big)$ and it returns one if and only if

(1) S_1 is a clique of size k, and

(2) $|S_2| \leq k$ or S_2 is not a clique.

V is a polynomial-time algorithm, and $|S_1|, |S_2| \leq |G|$. Notice that (G, k) is in $\mathsf{MaxClique}$ if and only if there exists such an S_1 (which is the maximum clique) such that for every S_2, $V\big((G, k), S_1, S_2\big) = 1$. Therefore $\mathsf{MaxClique} \in \Sigma_2$.

Proposition 7.5:

For every k, $\Sigma_k \subseteq \Sigma_{k+1}$.

Proof:

Let $S \in \Sigma_k$, and let V and p be the polynomial-time algorithm and polynomial which satisfy the conditions for S to be in Σ_k. Then let us define $V'(x, y_1, \dots, y_{k+1}) = V(x, y_2, \dots, y_{k+1})$, then V' and p satisfy the conditions for S to be in Σ_{k+1}.

Definition 7.6:

We define

$$\Pi_k = \mathsf{co}\Sigma_k = \{S \mid S^c \in \Sigma_k\}$$

Now, let us define $\mathtt{Q}_{2i} = \exists$ and $\mathtt{Q}_{2i+1} = \forall$, then $S \in \Pi_k$ if and only if

$$x \in S \iff \mathtt{Q}_1|y_1| \leq p(|x|)\mathtt{Q}_2|y_2| \leq p(|x|) \ \cdots \ \mathtt{Q}_k|y_k| \leq p(|x|)\big(V(x, y_1, \dots, y_k)\big)$$

Example 7.7:

Let us define the following decision problem

$$\mathsf{MinExpression} = \left\{\varphi \;\middle|\; \begin{array}{l}\varphi \text{ is a minimal boolean formula. In other words, for every equivalent boolean formula} \\ \psi\text{, the length of } \varphi \text{ is less than (or equal) to the length of } \psi.\end{array}\right\}$$

(The length of a boolean formula is defined recursively, but exactly how this is done doesn't really matter here.)

We claim that $\mathsf{MinExpression} \in \Pi_2$. We define a verifier $V(\varphi, \psi, \tau)$ where ψ is another boolean formula and τ is a boolean vector. V verifies that either $|\psi| \geq |\varphi|$ (meaning ψ is longer than φ) or $\varphi(\tau) \neq \psi(\tau)$. Now, φ is minimal if and only if for every boolean formula ψ, either ψ is longer than φ or ψ and φ are not equivalent. Thus

$$\varphi \in \mathsf{MinExpression} \iff \forall\psi\exists\tau\big(V(\varphi, \psi, \tau) = 1\big)$$

Now, we can also require that $|\psi| \leq |\varphi|$ as φ is minimal if and only if every smaller boolean formula is not equivalent to it. And we can require that $|\tau| \leq |\varphi|$ as we can assume that we are only valuating the variables in φ. So

$$\varphi \in \mathsf{MinExpression} \iff \forall|\psi| \leq |\varphi|\exists|\tau| \leq |\varphi|\big(V(\varphi, \psi, \tau) = 1\big)$$

Notice that

(1) $\Pi_0 = \mathbf{P}$

(2) $\Pi_1 = \mathbf{coNP}$

(3) $\Pi_k \subseteq \Pi_{k+1}$ as if V is a verifier for a problem in Π_k, we can define $V'(x, y_1, \dots, y_{k+1}) = V(x, y_2, \dots, y_k)$, and so V' verifies the problem in Π_{k+1}.

(4) $\Sigma_k \subseteq \Pi_{k+1}$ as if V is a verifier for a problem in Σ_k, $V'(x, y_1, \dots, y_{k+1}) = V(x, y_2, \dots, y_k)$ verifies the problem in Π_{k+1}.

(5) $\Pi_k \subseteq \Sigma_{k+1}$ as the construction for the two previous points also proves this.

Since $\Sigma_k \subseteq \Pi_{k+1} \subseteq \Sigma_{k+1}$,

$$\mathbf{PH} = \bigcup_{k=0}^{\infty} \Pi_k$$

Lemma 7.8:

For every $S \in \Sigma_{k+1}$, there exists a polynomial p and a problem $S' \in \Pi_k$ such that

$$S = \{x \mid \text{There exists a } y \text{ such that } |y| \leq p(|x|) \text{ and } (x, y) \in S'\}$$

Proof:

Let $S \in \Sigma_{k+1}$, then there exists a polynomial p and a verifier V such that $x \in S$ if and only if

$$\mathtt{Q}_1 y_1 \cdots \mathtt{Q} y_{k+1} \big(V(x, y_1, \ldots, y_{k+1}) = 1\big)$$

where $\mathtt{Q}_{2i} = \forall$ and $\mathtt{Q}_{2i+1} = \exists$ and $|y_i| \leq p(|x|)$ (we can think of restricting our domain). Let us define

$$S' = \big\{(x, y) \;\big|\; |y| \leq p(|x|),\, \mathtt{Q}_2 y_2 \cdots \mathtt{Q}_{k+1} y_{k+1} \big(V(x, y, y_2, \ldots, y_{k+1}) = 1\big)\big\}$$

It is obvious that $S' \in \Pi_k$, as $V'((x, y_1), y_2, \ldots, y_{k+1}) = V(x, y_1, \ldots, y_{k+1})$ is its verifier in Π_k. And

$$x \in S \iff \exists y_1 \mathtt{Q}_2 y_2 \cdots \mathtt{Q}_{k+1} y_{k+1} \big(V(x, y_1, \ldots, y_{k+1}) = 1\big) \iff \exists y_1 \big((x, y_1) \in S'\big)$$

as required. ∎

For the sake of not repeating myself, let us define

$$\mathtt{Q}_i = \begin{cases} \forall & i \equiv 0 \pmod 2 \\ \exists & i \equiv 1 \pmod 2 \end{cases}, \qquad \mathtt{Q}'_i = \mathtt{Q}_{i+1} = \begin{cases} \forall & i \equiv 1 \pmod 2 \\ \exists & i \equiv 0 \pmod 2 \end{cases}$$

Lemma 7.9:

For every $k \geq 1$, if $\Pi_k \subseteq \Sigma_k$ then $\Sigma_k = \Sigma_{k+1}$.

Proof:

Let $S \in \Sigma_{k+1}$, then there exists a problem $S' \in \Pi_k$ such that $x \in S$ if and only if there exists a y such that $(x, y) \in S'$. By the assumption, $S' \in \Sigma_k$ and so there exists a verifier V' and polynomial p' such that

$$(x, y) \in S' \iff \mathtt{Q}_1 y_1 \cdots \mathtt{Q}_k y_k \big(V'((x, y), y_1, \ldots, y_k) = 1\big)$$

where $|y_i| \leq p(|x|)$. Let us define $y^* = y y_1$ (the concatenation of y and y_1). Then $|y^*| = |y| + |y_1| \leq p(|x|) + p'(|x|)$.

$$V^*(x, y^*, y_2, \ldots, y_k) = V'((x, y), y_1, \ldots, y_k)$$

then

$$\begin{aligned} x \in S \iff \exists y\big((x, y) \in S'\big) &\iff \exists y \mathtt{Q}_1 y_1 \cdots \mathtt{Q}_k y_k \big(V'((x, y), y_1, \ldots, y_k) = 1\big) \\ &\iff \exists y^* \mathtt{Q}_2 y_2 \cdots \mathtt{Q}_k \big(V^*(x, y^*, y_2, \ldots, y_k) = 1\big) \end{aligned}$$

Where $|y_i| \leq p(|x|) + p'(|x|)$. Since $\exists = \mathtt{Q}_1$, this means that V^* is a verifier for S in Σ_k. Thus $\Sigma_{k+1} \subseteq \Sigma_k$, meaning $\Sigma_k = \Sigma_{k+1}$. ∎

Theorem 7.10:

If there exists a k such that $\Sigma_k = \Sigma_{k+1}$, then $\mathbf{PH} = \Sigma_k$.

Proof:

Since $\Sigma_k = \Sigma_{k+1}$, we have $\Pi_k = \Pi_{k+1}$ (by their definition since $S \in \Pi_k$ if and only if $S^c \in \Sigma_k$ if and only if $S^c \in \Sigma_{k+1}$ if and only if $S \in \Pi_{k+1}$). Then we have $\Pi_{k+1} = \Pi_k \subseteq \Sigma_{k+1}$. So by the previous lemma, we have $\Sigma_k = \Sigma_{k+1} = \Sigma_{k+2}$. Inductively this means that $\Sigma_i = \Sigma_k$ for every $i \geq k$. And so $\Sigma_i \subseteq \Sigma_k$ for every i and thus

$$\mathbf{PH} = \bigcup_{i=0}^{\infty} \Sigma_i = \Sigma_k$$

as required. ∎

Corollary 7.11:

If $\mathbf{P} = \mathbf{NP}$ then $\mathbf{PH} = \mathbf{P}$.

This is because $\mathbf{P} = \Sigma_0$ and $\mathbf{NP} = \Sigma_1$, so this is a specific case of the previous theorem.

7.2 Non-deterministic Algorithms

Definition 7.1:

A non-deterministic algorithm is an algorithm such that at every step it can make multiple choices. We say that a non-deterministic algorithm A decides a decision problem S if

(1) When $x \in S$, then there exists a run of A where $A(x) = 1$.

(2) When $x \in S$, then for every run of A, $A(x) = 0$.

A's runtime on an input x is the maximum number of steps it can take. We denote this as $t_A(x)$. We say that A **runs in polynomial time** or is a **polynomial-time algorithm** if there exists a polynomial p such that $t_A(x) \leq p(|x|)$.

Theorem 7.2:

A decision problem S is in $\mathbf{NP}$ if and only if there exists a non-deterministic polynomial-time algorithm A which decides S.

Proof:

If $S \in \mathbf{NP}$, then there exists a polynomial-time verifier V and its polynomial p. Let us define the non-determinstic algorithm

1. **function** $A(x)$
2. $\quad y \leftarrow \varepsilon$
3. $\quad$ **while** $(|y| \leq p(|x|))$
4. $\quad\quad$ **choose**
5. $\quad\quad\quad$ **(1)** $y \leftarrow y0$
6. $\quad\quad\quad$ **(2)** $y \leftarrow y1$
7. $\quad\quad\quad$ **(3)** **break**
8. $\quad\quad$ **endchoose**
9. $\quad$ **end while**
10. $\quad$ **return** $V(x, y)$
11. **end function**

Lines two to nine are essentially "choose $y \in \left\{w \in \{0,1\}^* \;\middle|\; |w| \leq p(|x|)\right\}$".

Then A decides S, as if $x \in S$ then it has a witness y whose length is at most $p(|x|)$, and so there exists some run of A which will return true. If $x \notin S$ then no matter what A will return false, so A does indeed decide S. And A runs in polynomial time, as the while loop takes at most $p(|x|)$ time, and V takes polynomial time.

Now if S is decided by a non-deterministic polynomial-time algorithm A, suppose its runtime is bound by the polynomial p. Let us define the verifier $V(x, y)$ where y is a sequence of choices that A can do, and V runs A where the ith choice is decided by y_i. If $x \in S$ then there exists a run of A which returns one, let y be the sequence of choices done by A, then $V(x, y) = 1$. And since $A(x)$ runs in $p(|x|)$ time, $|y| \leq p(|x|)$ as required. And if $x \notin S$ then no sequence of choices for A will make it return one, so $V(x, y) = 0$ for all y.

Thus V is a polynomial time verifier and p is its polynomial. Thus $S \in \mathbf{NP}$ as required. ■

Historically the equivalent definition given via non-deterministic algorithms was the standard definition for $\mathbf{NP}$. This actually gives the reasoning for the name of $\mathbf{NP}$, as it actually stands for *non-deterministic polynomial*.

Definition 7.3:

Let S be a decision problem, then we define $\mathbf{P}^S$ to be the set of all decision problems which can be decided by a deterministic oracle machine for S (a determinitic algorithm with an oracle for S). Thus $\mathbf{P}^S$ is the set of all decision problems where there exists a Cook reduction from them to S. And we similarly define $\mathbf{NP}^S$ to be the set of all decision problems which can be decided by a non-deterministic oracle machine for S.

Similarly if C is a set of decision problems then we define

$$\mathbf{P}^C = \bigcup_{S \in C} \mathbf{P}^S, \qquad \mathbf{NP}^C = \bigcup_{S \in C} \mathbf{NP}^S$$

Notice that if a decision problem requires oracles for two or more problems in C, it is not necessarily in $\mathbf{P}^C$ or $\mathbf{NP}^C$.

Example 7.4:

We will show that $\mathsf{MinExpression}^c \in \mathbf{NP}^{\mathbf{NP}}$. So we want to find a problem $S \in \mathbf{NP}$ and a non-deterministic polynomial-time oracle machine A^S which decides $\mathsf{MinExpression}^c$. The idea is for A^S, given a boolean formula φ, to choose some boolean formula ψ which is shorter than φ and show that they are equivalent using its oracle for S.

So let us define the decision problem

$$S = \{(\varphi, \psi) \mid \varphi \text{ and } \psi \text{ are equivalent boolean formulas}\}$$

this is not necessarily in $\mathbf{NP}$, but it is in $\mathbf{coNP}$ as

$$S^c = \{(\varphi, \psi) \mid \varphi \text{ and } \psi \text{ are non-equivalent boolean formulas}\}$$

as we can define the verifier $V((\varphi, \psi), \tau)$ which returns if $\varphi(\tau) \neq \psi(\tau)$, so $S^c \in \mathbf{NP}$.

So if we have an oracle for S^c, then we can ask if $(\varphi, \psi) \in S^c$ and then return the inverse. Explicitly, we define

1. **function** $A^{S^c}(\varphi)$
2.   **choose** a boolean formula ψ such that $|\psi| < |\varphi|$
3.   **return** $\neg\big((\varphi, \psi) \in S^c\big)$
4. **end function**

Then if $\varphi \in \mathsf{MinExpression}^c$, there exists a boolean formula ψ where $|\psi| < |\varphi|$ and then $A^{S^c}(\varphi)$ will return 1 when ψ is chosen. If $\varphi \in \mathsf{MinExpression}$ then for every choice of ψ, if ψ is a shorter boolean formula, φ and ψ are not equivalent and so $(\varphi, \psi) \in S^c$ and so $A^{S^c}(\varphi)$ will always return zero.

So A^{S^c} is a non-deterministic polynomial-time (since the oracle takes polynomial time, and choosing ψ takes polynomial time) algorithm which decides $\mathsf{MinExpression}^c$. Since $S^c \in \mathbf{NP}$, we have that $\mathsf{MinExpression} \in \mathbf{NP}^{\mathbf{NP}}$ as required.

Notice that if we have an oracle for S, this essentially gives us an oracle of S^c, as we can just negate its answer. Thus for any decision problem S,

$$\mathbf{P}^S = \mathbf{P}^{S^c}, \quad \mathbf{NP}^S = \mathbf{NP}^{S^c}$$

and so for any set of decision problems C,

$$\mathbf{P}^C = \mathbf{P}^{\mathsf{co}C}, \quad \mathbf{NP}^C = \mathbf{NP}^{\mathsf{co}C}$$

Proposition 7.5:

We can equivalently define the polynomial hierarchy by

$$\Sigma_0 = \mathbf{P}$$
$$\Sigma_1 = \mathbf{NP}$$
$$\Sigma_{k+1} = \mathbf{NP}^{\Sigma_k}$$

Notice that

$$\Sigma_1 = \mathbf{NP} = \mathbf{NP}^{\mathbf{P}} = \mathbf{NP}^{\Sigma_0}$$

so the recurrence $\Sigma_{k+1} = \mathbf{NP}^{\Sigma_k}$ is true for all $k \geq 0$.

Proof:

We will show inductively that $\Sigma_{k+1} = \mathbf{NP}^{\Sigma_k}$. Suppose $S \in \Sigma_{k+1}$, then by a previous lemma there exists a $S' \in \Pi_k$ and a polynomial p such that

$$x \in S \iff \exists y, |y| \leq p(|x|)\big((x,y) \in S'\big)$$

and so $S \in \mathbf{NP}^{S'}$, as we can define a non-deterministic algorithm which guesses y and checks if $(x,y) \in S'$. And therefore

$$S \in \mathbf{NP}^{S'} \subseteq \mathbf{NP}^{\Pi_k} = \mathbf{NP}^{\mathsf{co}\Sigma_k} = \mathbf{NP}^{\Sigma_k}$$

Now suppose $S \in \mathbf{NP}^{\Sigma_k}$, and so there exists a $S' \in \Sigma_k$ and a non-deterministic polynomial-time oracle machine $A^{S'}$ which decides S. We can assume that $A^{S'}$ functions in two parts (by redefining it):

(1) First $A^{S'}$ functions non-deterministically but without querying the oracle. At every point it wants to query the oracle with the query q_i, instead it guesses an answer a_i and continues with that answer.

(2) At the end it queries the oracle with every q_i and verifies that the correct answer is a_i. Ie. it verifies that $q_i \in S'$ if and only if $a_i = 1$.

$A^{S'}$ returns one if and only if it wants to return one in both the first step and the second step (meaning all the answers are correct).

So $x \in S$ if and only if there exists a list of queries $\vec{q}$ and a list of answers $\vec{a}$ such that $A^{S'}$ returns one when running on x and the answer to the query $\vec{q_i}$ is $\vec{a}_i$, and these are correct answers (as in $\vec{q_i} \in S'$ if and only if $\vec{a}_i = 1$). This is if and only if

$$\exists \vec{q}, \vec{a}\Big(* \wedge \bigwedge_{i=1}^{n}\big((q_i \in S' \wedge a_i = 1) \vee (q_i \notin S' \wedge a_i = 0)\big)\Big)$$

which is equivalent to

$$\exists q, a\Big(* \wedge \bigwedge_{i=1}^{n}\Big(\Big(\mathsf{Q}_1 y_i^1 \cdots \mathsf{Q}_{k-1} y_i^{k-1}\big(V'(q_i, y_i^1, \ldots, y_i^{k-1}) = 1 \wedge a_i = 1\big)\Big) \vee$$
$$\Big(\mathsf{Q}'_1 z_i^1 \cdots \mathsf{Q}'_{k-1} z_i^{k-1}\big(V'(q_i, z_i^1, \ldots, z_i^{k-1}) = 0 \wedge a_i = 0\big)\Big)\Big)\Big)$$

Let us denote $**$ to be the conjunction of $V'(q_i, y_i^1, \ldots, y_i^{k-1}) = 1 \wedge a_i = 1$ and $***$ to be the conjunction of $V'(q_i, y_i^1, \ldots, y_i^{k-1}) = 0 \wedge a_i = 0$.

This is equivalent to

$$\mathsf{Q}_1 y, a \vec{y}^1 \mathsf{Q}_2 \vec{y}^2, \vec{z}^1 \cdots \mathsf{Q}_{k-1} \vec{y}^{k-1}, \vec{z}^{k-2} \mathsf{Q}_k \vec{z}^k (* \wedge ** \wedge ***)$$

So this is an alternating list of quantifiers as well as a polynomial which checks the condition within the parentheses, and so $S \in \Sigma_{k+1}$ as required. ■

Note then that since $\mathsf{MinExpression} \in \Pi_2$, $\mathsf{MinExpression}^c \in \Sigma_2 = \mathbf{NP}^{\mathbf{NP}}$. This is a shorter proof than what we did above.

Notice then that if there is a Cook reduction S' to $S \in \Sigma_k$ then $S' \in P^{\Sigma_k} \subseteq \mathbf{NP}^{\Sigma_k} = \Sigma_{k+1}$ and so in particular $\mathbf{PH}$ is closed under Cook reductions.