Notes on Complexity Theory: Fall 2005 Last updated: December, 2005

Lecture 6

Jonathan Katz

1 Sparse Languages and $\mathcal{P}$ vs. $\mathcal{NP}$

In this lecture, we will explore various consequences that follow if $\mathcal{NP}$-complete languages are reducible to *sparse* languages (with different results for *Karp* reductions and *(Cook-)Turing* reductions). First, let us define what it means for a language to be sparse.

Definition 1 A language L is *sparse* if there exists a polynomial p such that $|L \cap \{0,1\}^n| \leq p(n)$. In other words, L contains only polynomially-many strings of any given length. $\diamondsuit$

It is not hard to see that any sparse language is in $\mathcal{P}$/poly (use a lookup table). In fact:

Theorem 1 *An $\mathcal{NP}$-complete language L is Turing-reducible to a sparse language iff $\mathcal{NP} \subset \mathcal{P}/\text{poly}$.*

The proof is quite straightforward (one direction is trivial), and would make a good homework/exam question. (*Hint:* Use the formulation of $\mathcal{P}$/poly in terms of polynomial-length advice strings...) We will see later the *Karp-Lipton theorem* which gives evidence that $\mathcal{NP} \not\subset \mathcal{P}/\text{poly}$ (in which case, by the above, $\mathcal{NP}$-complete languages are *not* Turing-reducible to sparse languages).

We can show a stronger result for the case of Karp reductions:

Theorem 2 (Mahaney's theorem) *An $\mathcal{NP}$-complete language L is Karp-reducible to a sparse language iff $\mathcal{P} = \mathcal{NP}$.*

Proof One direction is easy: if $\mathcal{P} = \mathcal{NP}$ then any $L \in \mathcal{NP}$ is trivially Karp-reducible to the sparse language $\{1\}$. (Why?)

The other direction is more difficult. Note first that if some $\mathcal{NP}$-complete language L is Karp-reducible to a sparse language, then *every* language in $\mathcal{NP}$ is Karp-reducible to a sparse language. Consider the $\mathcal{NP}$-complete language $LSAT$ defined as follows: $(\phi, x) \in LSAT$ if ϕ is a boolean formula in n variables, $x \in \{0,1\}^n$, and there is a satisfying assignment for ϕ which is lexicographically at most x. (Note that $(\phi, 1^n) \in LSAT$ iff $\phi \in SAT$, so $LSAT$ is easily seen to be $\mathcal{NP}$-complete.) Let $LSAT$ be Karp-reducible to the sparse set S via the poly-time computable function f.

We now show a polynomial-time algorithm for SAT which implies that $\mathcal{P} = \mathcal{NP}$. Given an input ϕ with n variables, the algorithm will apply f to various instances (ϕ, x) of $LSAT$; since f is poly-time computable, this implies a polynomial upper bound $p(n)$ on the length of the output of f. Let $q(n) \overset{\text{def}}{=} |S \cap \{0,1\}^{\leq p(n)}|$ and note that q is polynomial by sparseness of f.

We will view our algorithm as running a breadth-first search on a tree in which a node labeled v has children labeled $v0$ and $v1$; the root is labeled with the empty string. A node labeled v is viewed as corresponding to ϕ with the first $|v|$ variables set to v. When we say that ϕ has a satisfying assignment lexicographically at most v for $|v| < n$, what we really mean is that ϕ has a satisfying assignment lexicographically at most $v1^{n-|v|}$ (alternately, there is a satisfying assignment of ϕ in which the values of the first $|v|$ variables are lexicographically at most v). We also use $f(v)$ as shorthand for $f((\phi, v1^{n-|v|}))$ (so that if $f(v) \in S$ then ϕ has a satisfying assignment lexicographically at most v). The algorithm proceeds as follows:

```
Let $\mathsf{LIVE}_0 := \{\varepsilon\}$
for $i = 1$ to $n - 1$:
    Let $\mathsf{LIVE}_i$ consist of all children of $\mathsf{LIVE}_{i-1}$
    if $|\mathsf{LIVE}_i| > q(n)$, prune $\mathsf{LIVE}_i$ (see below) until it contains at most $q(n)$ nodes
Output 1 iff one of the children of $\mathsf{LIVE}_{n-1}$ gives a satisfying assignment
```

Pruning of a set of nodes V is done in the following way: First compute $Z_V = \{f(v) \mid v \in V\}$. Then use repeated application of the following rules:

1. If $f(v_1) = f(v_2)$ and v_2 is lexicographically greater than v_1, then remove v_2 from V.
2. If Z_V contains more than $q(n)$ distinct values, remove the lexicographically smallest member of V.

Note that once we are done pruning V we are left with a set V of at most $q(n)$ nodes.

That the algorithm runs in polynomial time is immediate, since there are n levels and at most $2q(n)$ nodes are ever considered at each level. Say a node labeled v (with $|v| < n$) is an ancestor of a satisfying assignment w if v is a prefix of w. Correctness of the algorithm follows from the observation that, assuming ϕ is satisfiable, at the end of iteration i the set LIVE_i contains an ancestor of the lexicographically smallest satisfying assignment of ϕ. We can prove this formally by induction. Clearly it is true for LIVE_0. For the inductive step, note that if it is true for LIVE_{i-1} then it is true for LIVE_i before we do the pruning. Now, when we do the pruning we eliminate some nodes, but we argue that the claim remains true for LIVE_i even after pruning:

1. Say $v_1, v_2 \in \mathsf{LIVE}_i$ with v_2 lexicographically larger than v_1 and $f(v_1) = f(v_2)$. Then v_2 will be removed. This might be a problem if v_2 is an ancestor of the smallest satisfying assignment. But this cannot be the case since then $f(v_2) \in S \Rightarrow f(v_1) \in S$, and so ϕ has a satisfying assignment lexicographically smaller than v_1.
2. Let $V = \mathsf{LIVE}_i$. If Z_V contains more than $q(n)$ values then we know that for at least one $z \in Z$ we have $z \notin S$ and so for at least one $v \in V$, ϕ does *not* have a satisfying assignment lexicographically at most v. But then ϕ cannot have a satisfying assignment lexicographically at most v_0, where v_0 is the lexicographically smallest element in V. So, we do not lose anything by removing v_0.

This completes the proof. ■

2 The Polynomial Hierarchy

The polynomial hierarchy (PH) provides a "natural" extension of the classes $\mathcal{NP}$ and co$\mathcal{NP}$. There are multiple definitions; we will start with the one that is most natural in terms of what we have covered in class up to now. In what follows, we say that R is *polynomial-time relation* if membership of $(x, y_1, \ldots, y_n) \in R$ can be decided in time polynomial in $|x|$. (We also implicitly assume that R is *polynomially bounded*; i.e., there exists a polynomial p such that $(x, y_1, \ldots, y_n) \in R$ implies $|y_1| + \cdots + |y_n| \leq p(|x|)$.) The fact that R is polynomially bounded just means that the language

$$\big\{(x, y_1, \ldots, y_n) \mid (x, y_1, \ldots, y_n) \in R\big\}$$

is in $\mathcal{P}$.

Definition 2 We say $L \in \Sigma_i$ if there exists a polynomial-time relation R such that:

$$x \in L \;\Leftrightarrow\; \exists y_1 \forall y_2 \cdots Q_i y_i \text{ s.t. } (x, y_1, \ldots, y_i) \in R,$$

where $Q_i = \forall$ if i is even, and $Q_i = \exists$ if i is odd. By convention, we let $\Sigma_0 = \mathcal{P}$. ♢

Definition 3 We say $L \in \Pi_i$ if there exists a polynomial-time relation R such that:

$$x \in L \;\Leftrightarrow\; \forall y_1 \exists y_2 \cdots Q_i y_i \text{ s.t. } (x, y_1, \ldots, y_i) \in R,$$

where $Q_i = \exists$ if i is even, and $Q_i = \forall$ if i is odd. By convention, we let $\Pi_0 = \mathcal{P}$. ♢

It is not hard to prove that:

Proposition 3 $\Sigma_i \cup \Pi_i \subseteq \Sigma_{i+1} \cap \Pi_{i+1}$.

By definition, we have $\Sigma_1 = \mathcal{NP}$ and $\Pi_1 = \text{co}\mathcal{NP}$. For an example of a language in Π_2, consider the following language motivated by the problem of *circuit minimization*. For a boolean circuit C, define $L(C) = \{x \mid C(x) = 1\}$; i.e., $L(C)$ is exactly the language decided by C. Now define:

$$L_{\text{notmin}} = \{C \mid C \text{ is } \textit{not} \text{ the smallest circuit deciding } L(C)\}.$$

We show that $L_{\text{notmin}} \in \Pi_2$. Let R_{notmin} be the relation defined as follows:

$$R_{\text{notmin}} = \left\{(C_1, C_2, x) \mid C_1(x) = C_2(x) \text{ and } |C_1| > |C_2|\right\}.$$

Note that R_{notmin} is a poly-time relation. Now,

$$L_{\text{notmin}} = \left\{C \mid \exists C' \forall x : \; (C, C', x) \in R_{\text{notmin}}\right\}.$$

(I would more naturally write this as:

$$L_{\text{notmin}} = \left\{C \mid \exists C' \text{ with } |C'| < |C| \text{ s.t. } \forall x : \; C(x) = C'(x)\right\},$$

but you can check that this is just a reformulation of the above.) Similarly, define

$$L_{\text{min}} = \{C \mid C \text{ is the smallest circuit deciding } L(C)\}.$$

To see that $L_{\text{min}} \in \Sigma_2$, note that:

$$L_{\text{min}} = \left\{C \mid \forall C' \text{ with } |C'| < |C| \text{ s.t. } \exists x : \; C(x) \neq C'(x)\right\}.$$

The above is an example of the more general fact that $\Sigma_i = \text{co}\Pi_i$ for all i. (It is a good exercise to prove this...)

With the above in place we can define the polynomial hierarchy.

Definition 4 $\mathsf{PH} = \cup_{i \geq 0} \Sigma_i = \cup_{i \geq 0} \Pi_i$. ♢

(That the two definitions are equivalent follows using Proposition 3.) We note the following result:

Proposition 4 $\mathsf{PH} \subseteq \mathsf{PSPACE}$.

Proof (Sketch) The proposition is easy to prove directly (given x, just enumerate through all possible $y_1, \ldots, y_i \ldots$). Somewhat more illuminating is the proof using the PSPACE-complete language QBF (we will prove in a later lecture that QBF is in fact PSPACE-complete). This language is defined as follows:

$$QBF = \left\{\phi \mid \begin{array}{c} \phi \text{ is a boolean formula on } n \text{ inputs, and} \\ \exists x_1 \forall x_2 \cdots Q x_n : \; \phi(x_1, \ldots, x_n) = 1 \end{array}\right\}.$$

Note the difference between QBF and a language in Σ_i (or Π_i): the former allows *unbounded* (polynomial) alternation of quantifiers, while the latter allow only *bounded* alternation of quantifiers. ■

2.1 An Alternate Definition of PH

A second way to define PH is in terms of oracle machines.

Definition 5 Define Σ_i inductively as follows:

- $\Sigma_0 \stackrel{\text{def}}{=} \mathcal{P}$.
- $\Sigma_1 \stackrel{\text{def}}{=} \mathcal{NP}$.
- $\Sigma_{i+1} \stackrel{\text{def}}{=} \mathcal{NP}^{\Sigma_i}$.

(Actually, the second condition is redundant; it is included for simplicity.) Also, $\Pi_i \stackrel{\text{def}}{=} \text{co}\Sigma_i$. $\diamondsuit$

We show that the definitions are equivalent. *For this section only*, let Σ_i^T refer to the definition in terms of oracle Turing machines, and let Σ_i^Q refer to the definition in terms of quantifiers. We first prove by induction that $\Sigma_i^Q \subseteq \Sigma_i^T$. Clearly, the assertion is true for $i = 1$ (since they are both equal to $\mathcal{NP}$). Now, assume the assertion is true for i and we will prove it for $i+1$. Let $L \in \Sigma_{i+1}^Q$. Then there exists a poly-time relation R such that

$$x \in L \Leftrightarrow \exists y_1 \forall y_2 \cdots Q_{i+1} y_{i+1} \text{ s.t. } (x, y_1, \ldots, y_{i+1}) \in R.$$

In other words, there exists a language $L' \in \Pi_i^Q$ such that

$$x \in L \Leftrightarrow \exists y_1 \text{ s.t. } (x, y_1) \in L'.$$

By our inductive assumption, $\Sigma_i^Q \subseteq \Sigma_i^T$ and hence $\Pi_i^Q \subseteq \Pi_i^T$. Thus $L' \in \Pi_i^T$, and it then follows easily that $L \in \mathcal{NP}^{\Pi_i^T} = \mathcal{NP}^{\Sigma_i^T} \stackrel{\text{def}}{=} \Sigma_{i+1}^T$. (Why does the first equality hold?)

For the other direction, assume $\Sigma_i^T \subseteq \Sigma_i^Q$ and we now prove that this holds for $i+1$ as well. Suppose $L \in \Sigma_{i+1}^T$. This means there exists a $\mathcal{NP}$ machine M and a language $L' \in \Sigma_i^T$ such that $L \in M^{L'}$. In particular, then, $x \in L$ iff $\exists y, q_1, a_1, \ldots, q_n, a_n$ (here, y represents the non-deterministic choices of M, while q_j, a_j represent the j^{th} query of M to its oracle and the answers, respectively) such that:

- M, on input x, non-deterministic choices y, and oracle answers $a_1, \ldots, a_n$, makes queries $q_1, \ldots, q_n$ and accepts.
- For all j, we have $a_j = 1$ iff $q_j \in L'$.

Now, since $L' \in \Sigma_i^Q$ (by our inductive assumption), we can re-formulate the last condition as:

- $a_j = 1 \Leftrightarrow \exists y_1^j \forall y_2^j \cdots Q_i y_i^j$ s.t. $(q_j, y_1^j, \ldots, y_i^j) \in R_{L'}$
- $a_j = 0 \Leftrightarrow \forall y_1^j \exists y_2^j \cdots Q_i' y_i^j$ s.t. $(q_j, y_1^j, \ldots, y_i^j) \in \bar{R}_{L'}$

for some poly-time relation $R_{L'}$. Then the above leads to the following specification of L as a Σ_{i+1}^Q language: $x \in L$ iff $\exists \left(y, q_1, a_1, \ldots, q_n, a_n, \{y_1^j\}_{j \in Y}\right) \forall \left(\{y_1^j\}_{j \in N}, \{y_2^j\}_{j \in Y}\right) \cdots Q_{i+1} \left(\{y_i^j\}_{j \in N}\right)$:

- M, on input x, non-deterministic choices y, and oracle answers $a_1, \ldots, a_n$, makes queries $q_1, \ldots, q_n$ and accepts,
- Let Y be the set of j's such that $a_j = 1$, and let N be the set of j's such that $a_j = 0$. Then:
  - $(q_j, y_1^j, \ldots, y_i^j) \in R_{L'}$ for all $j \in Y$,
  - $(q_j, y_1^j, \ldots, y_i^j) \in \bar{R}_{L'}$ for all $j \in N$.

2.2 "Collapsing" PH

A generalization of the question $\mathcal{NP} \overset{?}{=} \text{co}\mathcal{NP}$ is the question $\Sigma_i \overset{?}{=} \Pi_i$ (for any i). Since it is believed that PH has infinitely-many levels, the following theorem is taken as evidence that $\Sigma_i \neq \Pi_i$ for any i (the proof is for the case of $\Sigma_1 = \mathcal{NP}$ and $\Pi_1 = \text{co}\mathcal{NP}$ but the argument extends easily for any i):

Theorem 5 *If $\mathcal{NP} = co\mathcal{NP}$ then* PH $= \mathcal{NP}$. *(More generally, if $\Sigma_i = \Pi_i$ then* PH $= \Sigma_i$.*)*

Proof Since $\mathcal{NP} \subseteq$ PH by definition, we need only show that PH $\subseteq \mathcal{NP}$ under the assumption of the theorem. We proceed by induction. Clearly, $\Sigma_1 = \mathcal{NP}$ by definition. Now, assume $\Sigma_i = \mathcal{NP}$ and we prove that $\Sigma_{i+1} \overset{\text{def}}{=} \mathcal{NP}^{\Sigma_i} = \mathcal{NP}^{\mathcal{NP}} \subseteq \mathcal{NP}$. Let $L \in \mathcal{NP}^{\mathcal{NP}}$. Then there exists an $\mathcal{NP}$ machine M with oracle access to a language $L' \in \mathcal{NP}$ such that $M^{L'}$ accepts L. Let $R_{L'}$ be the poly-time relation for L'. By the assumption that $\mathcal{NP} = \text{co}\mathcal{NP}$, we have that $\bar{L}'$ (with poly-time relation $R_{\bar{L}'}$) is also in $\mathcal{NP}$. Then $x \in L$ iff there exist $y, q_1, a_1, \ldots, q_n, a_n$ where:

- M, on input x, non-deterministic choices y, and oracle answers $a_1, \ldots, a_n$, makes queries $q_1, \ldots, q_n$ and accepts;
- If $a_i = 1$ then $\exists y_i$ such that $(q_i, y_i) \in R_{L'}$;
- If $a_i = 0$ then $\exists y_i$ such that $(q_i, y_i) \in R_{\bar{L}'}$.

This leads easily to the following $\mathcal{NP}$ formulation of L: $x \in L$ iff $\exists y, q_1, a_1, y_1, \ldots, q_n, a_n, y_n$ such that the above hold. ■

3 The Karp-Lipton Theorem

The following result is one of the main sources of our belief that $\mathcal{NP} \not\subseteq \mathcal{P}/\text{poly}$.

Theorem 6 (Karp-Lipton) *If $\mathcal{NP} \subseteq \mathcal{P}/\text{poly}$ then $\Sigma_2 = \Pi_2$ (and hence* PH $= \Sigma_2$*).*

Proof We begin with a claim that can be proved easily given our earlier work on self-reducibility of SAT: if $SAT \in \mathcal{P}/\text{poly}$ then there exists a polynomial-size circuit family $\{C_n\}$ such that $C_{|\phi|}(\phi)$ outputs a satisfying assignment for ϕ whenever ϕ is satisfiable. That is, if SAT can be *decided* by poly-size circuits, then SAT can be *solved* by poly-size circuits. Let $|C_n| \leq p(n)$ for some polynomial p.

We use this claim to prove that $\Pi_2 \subseteq \Sigma_2$ (from which the theorem follows). Let $L \in \Pi_2$. By definition, this means there exists a polynomial-time relation R such that[1]

$$x \in L \Leftrightarrow \forall y \exists z : (x, y, z) \in R.$$

Define $L' = \{(x, y) \mid \exists z : (x, y, z) \in R\}$. Note that $L' \in \mathcal{NP}$, and so there is a Karp reduction f from L' to SAT. We assume without loss of generality that $|f(x, y)| = q(|x|)$ for some polynomial q (note that if y is too long, then $(x, y) \notin L'$ and so $f(x, y)$ can output some unsatisfiable formula of the appropriate length); note also (using the same reasoning) that $f(x, y)$ can be computed in

[1]By convention throughout this proof (indeed, as we have been doing throughout this entire lecture), we implicitly assume that quantification is done over strings of length at most some (appropriate) fixed polynomial in $|x|$.

time polynomial in $|x|$. Letting R_{SAT} denote the obvious relation for SAT, we may thus express membership in L as follows:

$$x \in L \Leftrightarrow \forall y \exists w : (f(x,y), w) \in R_{SAT}. \tag{1}$$

Define the relation R^* via:

$$R^* \overset{\text{def}}{=} \left\{(x,y,C) \mid (f(x,y),\, C(f(x,y))) \in R_{SAT} \text{ and } |C| \leq p(q(|x|))\right\},$$

and note that R^* is a poly-time relation. Define $\hat{L}$ via:

$$x \in \hat{L} \Leftrightarrow \exists C \forall y : (x,y,C) \in R^*,$$

and note that $\hat{L} \in \Sigma_2$. We claim that $L = \hat{L}$. To see this, note that if $x \in L$ then $f(x,y) \in SAT$ for all y. But then there does indeed exist a C (namely, $C = C_{q(|x|)}$) such that for all y it is the case that $C(f(x,y))$ outputs a satisfying assignment for $f(x,y)$ (and furthermore $|C| \leq p(q(|x|))$. It follows that $x \in \hat{L}$. On the other hand, if $x \notin L$ then there exists a y^* such that $f(x,y^*) \notin SAT$. But then for any choice of C we have $(f(x,y^*), C(f(x,y^*))) \notin R_{SAT}$ and so $x \notin \hat{L}$. ■

Bibliographic Notes

Theorem 2 was proven by Mahaney [4], building on earlier work of Fortune [2]. The present proof of Theorem 2 is adapted from [1, Chap. 4]. A proof of a weaker version of Theorem 2 is given in [3, Lect. 8].

Section 2 is largely based on [3, Lect. 9]. The proof in Section 3 is adapted from the proof given in lecture notes by Cai [1, Lect. 10] and attributed to Hopcroft there; thanks to Yehuda Lindell for pointing out errors/omissions in an earlier version of the proof. (A more clever, but more complicated, proof of the Karp-Lipton theorem is also given in [3, Lect. 9].)

References

[1] J.-Y. Cai. Scribe notes for CS 810: Introduction to Complexity Theory. Nov. 30, 2003.

[2] S. Fortune. A Note on Sparse Complete Sets. *SIAM J. Computing* 8(3): 431–433, 1979.

[3] O. Goldreich. Introduction to Complexity Theory (July 31, 1999).

[4] S. Mahaney. Sparse Complete Sets of NP: Solution of a Conjecture of Berman and Hartmanis. *JCSS* 25(2): 130–143, 1982.