

Two Queries

Harry Buhrman*
CWI
PO Box 94079
1090 GB Amsterdam
The Netherlands

Lance Fortnow†
University of Chicago
Department of Computer Science
1100 E. 58th St.
Chicago, IL 60637

Abstract

We consider the question whether two queries to **SAT** are as powerful as one query. We show that if $\mathbf{P}^{\mathbf{NP}^{[1]}} = \mathbf{P}^{\mathbf{NP}^{[2]}}$ then

- Locally either $\mathbf{NP} = \mathbf{coNP}$ or \mathbf{NP} has polynomial-size circuits.
- $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{NP}^{[1]}}$.
- $\Sigma_2^p \subseteq \Pi_2^p/1$.
- $\Sigma_2^p = \mathbf{UP}^{\mathbf{NP}^{[1]}} \cap \mathbf{RP}^{\mathbf{NP}^{[1]}}$.
- $\mathbf{PH} = \mathbf{BPP}^{\mathbf{NP}^{[1]}}$.

Moreover we extend work of Hemaspaandra, Hemaspaandra and Hempel to show that if $\mathbf{P}^{\Sigma_2^p[1]} = \mathbf{P}^{\Sigma_2^p[2]}$ then $\Sigma_2^p = \Pi_2^p$. We also give a relativized world where $\mathbf{P}^{\mathbf{NP}^{[1]}} = \mathbf{P}^{\mathbf{NP}^{[2]}}$ but $\mathbf{NP} \neq \mathbf{coNP}$.

1 Introduction

Are two queries to **SAT** as powerful as one query? This question has a long history in computational complexity theory. When computing functions, Krentel [Kre88] showed that if two queries can be simulated by one query to **SAT**, that is $\mathbf{FP}^{\mathbf{NP}^{[1]}} = \mathbf{FP}^{\mathbf{NP}^{[2]}}$, then $\mathbf{P} = \mathbf{NP}$.

When we focus on languages instead of functions life gets more complicated. Kadin [Kad88] showed that if $\mathbf{P}^{\mathbf{NP}^{[1]}} = \mathbf{P}^{\mathbf{NP}^{[2]}}$ then $\mathbf{NP} \subseteq \mathbf{coNP}/poly$ and thus $\mathbf{PH} \subseteq \Sigma_3^p$ [Yap83]. Beigel, Chang and Ogihara [BCO93] building on Chang and Kadin [CK96] improve this to show that every language in the polynomial-time hierarchy can be solved by an **NP** query and a Σ_2^p query.

One goal might be to show that $\mathbf{P}^{\mathbf{NP}^{[1]}} = \mathbf{P}^{\mathbf{NP}^{[2]}}$ implies $\mathbf{NP} = \mathbf{coNP}$. Hemaspaandra, Hemaspaandra and Hempel [HHH99a] made a step into that direction. They showed that for $k > 2$, if $\mathbf{P}^{\Sigma_k^p[1]} = \mathbf{P}^{\Sigma_k^p[2]}$ then $\Sigma_k^p = \Pi_k^p$.

We extend their techniques to show that if $\mathbf{P}^{\Sigma_2^p[1]} = \mathbf{P}^{\Sigma_2^p[2]}$ then $\Sigma_2^p = \Pi_2^p$. However, the techniques cannot be pushed down to $k = 1$. We show a relativized world where $\mathbf{P}^{\mathbf{NP}^{[1]}} = \mathbf{P}^{\mathbf{NP}^{[2]}}$ but $\mathbf{NP} \neq \mathbf{coNP}$.

What does happen when $\mathbf{P}^{\mathbf{NP}^{[1]}} = \mathbf{P}^{\mathbf{NP}^{[2]}}$?

Building on the techniques of the above papers we show several new collapses if $\mathbf{P}^{\mathbf{NP}^{[1]}} = \mathbf{P}^{\mathbf{NP}^{[2]}}$ including:

- Locally either $\mathbf{NP} = \mathbf{coNP}$ or \mathbf{NP} has polynomial-size circuits.
- $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{NP}^{[1]}}$.
- $\Sigma_2^p \subseteq \Pi_2^p/1$.
- $\Sigma_2^p = \mathbf{UP}^{\mathbf{NP}^{[1]}} \cap \mathbf{RP}^{\mathbf{NP}^{[1]}}$.
- $\mathbf{PH} = \mathbf{BPP}^{\mathbf{NP}^{[1]}}$.

*Email: buhrman@cwi.nl. URL: <http://www.cwi.nl/~buhrman>. Partially supported by the Dutch foundation for scientific research (NWO) by SION project 612-34-002, and by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556, and HC&M grant nr. ERB4050PL93-0516.

†Email: fortnow@cs.uchicago.edu. URL: <http://www.cs.uchicago.edu/~fortnow>. Work done while on leave at CWI. Supported in part by NSF grant CCR 92-53582, the Dutch Foundation for Scientific Research (NWO) and a Fulbright Scholar award.

2 Preliminaries

We assume the reader familiar with basic notions of complexity theory as can be found in many textbooks in the area (such as [GJ79, HU79, BDG88, BDG90]).

For a set A we will identify A with its characteristic function. Hence for a string x , $A(x) \in \{0, 1\}$ and $A(x) = 1$ iff $x \in A$.

For languages A and B define $A\Delta B$ to be the symmetric difference of A and B , i.e., $(A \cap \overline{B}) \cup (\overline{A} \cap B)$. For complexity classes \mathcal{C} and \mathcal{D} define the class $\mathcal{C}\Delta\mathcal{D}$ as

$$\{A\Delta B \mid A \in \mathcal{C} \text{ and } B \in \mathcal{D}\}.$$

An oracle Turing machine is nonadaptive if it produces a list of all of the queries it is going to make before it makes the first query. **SAT** is the set of satisfiable boolean formulae. For any set A , $\mathbf{P}^{A[k]}$ is the class of languages that are recognized by polynomial time Turing machines that access the oracle A at most k times on each input. The class $\mathbf{P}_{tt}^{A[k]}$ will allow only nonadaptive access to A . We note that $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$ if $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}_{tt}^{\mathbf{NP}[2]}$ [CK95], so all our results could be stated assuming $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}_{tt}^{\mathbf{NP}[2]}$.

If A is in $\mathbf{P}_{tt}^{B[1]}$ then there is a polynomial-time function $h(x) : \Sigma^* \rightarrow \Sigma^* \times \{+, -\}$ such that x is in A if and only if $h(x) = (z, +)$ and z is in B or $h(x) = (z, -)$ and z is not in B . The string z is the query made by the Turing machine and the $+$ and $-$ refer whether the machine accepts iff the query is in or rejects iff the query is in. The machine could ignore the query or not make it at all, in which case we just use a z known to be in (or out) of B .

UP is the set of languages that are recognized by polynomial-time nondeterministic Turing machines that have at most one accepting path on each input.

We can generalize **NP** by defining the *polynomial-time hierarchy*. We define $\Sigma_0^p = \mathbf{P}$ and inductively define $\Sigma_{i+1}^p = \mathbf{NP}^{\Sigma_i^p}$ for $i > 0$. We let $\Pi_i^p = \mathbf{co}\Sigma_i^p$. In particular, we have $\mathbf{NP} = \Sigma_1^p$ and $\mathbf{coNP} = \Pi_1^p$. Many complexity theorists conjecture that the polynomial-time hierarchy is infinite, i.e., $\Sigma_{i+1}^p \neq \Sigma_i^p$ for all i .

Let \mathcal{C} be a complexity class. We say a set A is in $\mathcal{C}/f(n)$ if there exists an arbitrary function h such that $|h(n)| = f(n)$ and a set $B \in \mathcal{C}$ such that $x \in A$ iff $\langle x, h(|x|) \rangle \in B$. We say a language A is in $\mathcal{C}/poly$ if A is in $\mathcal{C}/p(n)$ for some polynomial p .

Given a formula ϕ on n variables we define the *self-reduction tree* of ϕ as follows: ϕ is the root and if the formula $\phi'(x_1, \dots, x_m)$ is a node in the tree then $\phi'(x_1 := \text{true}, x_2, \dots, x_m)$ and $\phi'(x_1 := \text{false}, x_2, \dots, x_m)$ are the two children of ϕ' . We say $\phi'(x_1, \dots, x_m)$ *self-reduces* to the formulae $\phi'(x_1 := \text{true}, x_2, \dots, x_m)$ and $\phi'(x_1 := \text{false}, x_2, \dots, x_m)$. A formula with no free variables is a leaf in the tree. A node in a tree is satisfiable if and only if either of its children are satisfiable. One can determine easily in polynomial time whether a leaf is true or false.

3 Collapse if $\mathbf{P}^{\Sigma_2^p[1]} = \mathbf{P}_{tt}^{\Sigma_2^p[2]}$

Hemaspaandra, Hemaspaandra and Hempel [HHH99a] exhibited a strong collapse if $\mathbf{P}^{\Sigma_k^p[1]} = \mathbf{P}_{tt}^{\Sigma_k^p[2]}$ for $k > 2$.

Theorem 3.1 (Hemaspaandra-Hemaspaandra-Hempel) *For every $k > 2$, if $\mathbf{P}^{\Sigma_k^p[1]} = \mathbf{P}_{tt}^{\Sigma_k^p[2]}$ then $\Sigma_k^p = \Pi_k^p$.*

Extending their techniques we improve their result to $k = 2$.

Theorem 3.2 *If $\mathbf{P}^{\Sigma_2^p[1]} = \mathbf{P}_{tt}^{\Sigma_2^p[2]}$ then $\Sigma_2^p = \Pi_2^p$.*

Proof: For a predicate $R(y)$ we use the notation $\exists^m y R(y)$ to mean $\exists y (|y| = m \wedge R(y))$ and $\forall^m y R(y)$ to mean $\forall y (|y| = m \Rightarrow R(y))$.

Since $\Sigma_2^p \Delta \mathbf{NP} \subseteq \mathbf{P}_{tt}^{\Sigma_2^p[2]}$, Theorem 3.2 follows immediately from the following lemma.

Lemma 3.3 *If $\Sigma_2^p \Delta \mathbf{NP} \subseteq \mathbf{P}^{\Sigma_2^p[1]}$ then $\Sigma_2^p = \Pi_2^p$.*

Proof: Fix K a complete set for Σ_2^p . Given an input x we will give a Σ_2^p algorithm for determining that x is not in K . Let $n = |x|$. We can assume there exists a polynomial-time predicate P such that

$$x \in K \Leftrightarrow \exists^n y \forall^n z P(x, y, z)$$

	$\exists \phi, u$		
\exists	$h(x, \phi) = (z, +)$ $\phi \in \mathbf{SAT}$	$h(x, \phi) = (z, -)$ $\phi \notin \mathbf{SAT}$	
	$\forall v$	$\forall v$	$\forall u$
\forall	$P(z, u, v)$	$P(z, u, v)$	Assuming that for all ϕ $\phi \in \mathbf{SAT} \Leftrightarrow h(x, \phi) = (z, -)$ use self-reduction to find v such that $\neg P(x, u, v)$

Figure 1: Σ_2^p algorithm to determine $x \notin K$. The \exists and \forall quantifiers have appropriate polynomial-length bounds. The Σ_2^p algorithm accepts if any column accepts.

where the quantification is done over strings of length n .

Define the set $D \in \Sigma_2^p \Delta \mathbf{NP}$ as follows:

$$D = K \times \overline{\mathbf{SAT}} \cup \overline{K} \times \mathbf{SAT} = \{(x, \phi) \mid (x \in K \wedge \phi \notin \mathbf{SAT}) \vee (x \notin K \wedge \phi \in \mathbf{SAT})\}$$

By assumption, there exists a polynomial-time computable function $h : \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \times \{+, -\}$ such that $(x, \phi) \in D$ iff $h(x, \phi) = (z, +)$ and $z \in K$ or $h(x, \phi) = (z, -)$ and $z \notin K$.

We give the Σ_2^p algorithm for determining that x is not in K in Figure 1. Lemma 3.3 follows from the following claim.

Claim 3.4 *The algorithm in Figure 1 accepts exactly when x is not in K .*

Proof: Suppose the algorithm accepts in the rightmost column. For every u we will have found a counterexample v to $P(x, u, v)$ so x is not in K .

If the first or second columns accepts then we have $x \notin K \iff z \in K \iff \exists^{|z|} u \forall^{|z|} v P(z, u, v)$.

Suppose that x is not in K . If the assumption in the rightmost column is true then the self-reduction will always find the appropriate v .

If the assumption in the rightmost column is wrong then either there is some ϕ such that $\phi \in \mathbf{SAT}$ and $h(x, \phi) = (z, +)$ or $\phi \notin \mathbf{SAT}$ and $h(x, \phi) = (z, -)$. Then we will have either the first or second column accepting respectively. \square

Hemaspaandra, Hemaspaandra and Hempel [HHH99a] give a more general version of Theorem 3.1 for the boolean hierarchy over Σ_k^p for $k > 2$. In a later paper, Hemaspaandra, Hemaspaandra and Hempel [HHH99b] show that our Theorem 3.2 similarly extends to the boolean hierarchy over Σ_2^p .

Beigel and Chang [BC97] use the techniques in the proof of Theorem 3.2 for some new results on commutative queries.

4 Limitations of $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$

We show that Theorem 3.2 cannot carry over for \mathbf{NP} with a relativizable proof.

Theorem 4.1 *There exists a relativized world relative to which $\mathbf{NP} \neq \mathbf{coNP}$ but $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]} = \mathbf{PSPACE}$.*

We will use \mathbf{UP} -generics as developed by Fortnow and Rogers [FR94]. To create a \mathbf{UP} -generic start with an oracle like \mathbf{TQBF} that makes $\mathbf{P} = \mathbf{PSPACE}$ and add a generic set U restricted to have at most one string at lengths that are towers of 2 and no strings at any other lengths. So $G = \mathbf{TQBF} \oplus U$. \mathbf{UP} -generics also play an important role in creating a relativized world where the Berman-Hartmanis isomorphism conjecture holds and one-way functions exist [Rog97].

Given an input x a polynomial-time process can only access one interesting string in U . The others are either too large to be queried or so small that they can be found quickly. We refer to this interesting string as the ‘‘cookie’’.

Fortnow and Rogers [FR94] show that relative to \mathbf{UP} -generics G :

1. $\mathbf{P}^G \neq \mathbf{NP}^G$.
2. $\mathbf{P}^G = \mathbf{NP}^G \cap \mathbf{coNP}^G$.

Immediately we have the following corollary.

Corollary 4.2 *Relative to UP-generics G ,*

$$\mathbf{NP}^G \neq \mathbf{coNP}^G.$$

Fix a \mathbf{PSPACE}^G language L accepted by some alternating polynomial-time Turing machine M^G . We now describe the $\mathbf{P}^{\mathbf{NP}^G[1]}$ algorithm for L .

Use the $\mathbf{P} = \mathbf{PSPACE}$ base oracle to determine if x is accepted by M^G if there is no cookie. There are two cases:

No: Accept if the following \mathbf{NP} question is true (using $\mathbf{P} = \mathbf{PSPACE}$ base oracle): Does there exist a cookie such that $M^G(x)$ accepts?

Yes: Accept if the following \mathbf{NP} question is false (using $\mathbf{P} = \mathbf{PSPACE}$ base oracle): Does there exist a cookie such that $M^G(x)$ rejects?

In either case we ask a single \mathbf{NP} question and accept if and only if $M^G(x)$ accepts. \square

As a bonus we get the following corollary about complete sets for \mathbf{PSPACE} .

Corollary 4.3 *There exists a relativized world where the 1-tt-complete degree for \mathbf{PSPACE} is not the same as the many-one complete degree.*

We cannot extend theorem 4.1 to get $\mathbf{NP} \neq \mathbf{coNP}$ and $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{EXP}$ since Homer, Kurtz and Royer [HKR93] give a relativizable proof that the 1-tt-complete degree for \mathbf{EXP} is the same as the many-one complete degree. In a later paper, Beigel, Buhrman and Fortnow [BBF98] give a relativized world where the 1-tt-complete degree for \mathbf{NP} is not the same as the many-one complete degree.

5 Consequences of $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$

In this section we examine collapses that occur if $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$.

Kadin [Kad88] showed that the polynomial-time hierarchy collapse under this assumption.

Theorem 5.1 (Kadin) $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$ implies that $\mathbf{NP} \subseteq \mathbf{coNP}/poly$.

Yap [Yap83] shows that if $\mathbf{NP} \subseteq \mathbf{coNP}/poly$ then $\mathbf{PH} = \Sigma_3^p$.

Beigel, Chang and Ogihara [BCO93] building on work of Chang and Kadin [CK96] improved the collapse to just above Σ_2^p .

Theorem 5.2 (Beigel-Chang-Ogihara) *If $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$ then the polynomial-time hierarchy collapses to $\Sigma_2^p \Delta \mathbf{NP}$.*

Building on the work of Kadin [Kad88], Chang and Kadin [CK96], Beigel, Chang and Ogihara [BCO93] and Hemaspaandra, Hemaspaandra and Hempel [HHH99a], we will show several other collapses under this same assumption.

Theorem 5.3 *If $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$ then*

1. For all A in \mathbf{coNP} , $A = B \cup C$ where B is in \mathbf{NP} and C is in $\mathbf{P}/poly$. Moreover for every n either
 - (a) $A \cap \Sigma^n = B \cap \Sigma^n$, or
 - (b) $A \cap \Sigma^n = C \cap \Sigma^n$.
2. $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{NP}[1]}$.
3. $\Sigma_2^p \subseteq \Pi_2^p/1$.

4. $\Sigma_2^p = \mathbf{UP}^{\mathbf{NP}^{[1]}}$.
5. $\Sigma_2^p = \mathbf{RP}^{\mathbf{NP}^{[1]}}$.
6. $\mathbf{PH} = \mathbf{BPP}^{\mathbf{NP}^{[1]}}$.

For clarity, we leave off the polynomial-length bounds on the \exists and \forall quantifiers in the proofs below. Define the languages D , E and F by

$$\begin{aligned} D &= \mathbf{SAT} \times \overline{\mathbf{SAT}} \cup \overline{\mathbf{SAT}} \times \mathbf{SAT} = \{(\phi, \psi) \mid (\phi \in \mathbf{SAT} \wedge \psi \notin \mathbf{SAT})\} \cup \{(\phi, \psi) \mid (\phi \notin \mathbf{SAT} \wedge \psi \in \mathbf{SAT})\}. \\ E &= \overline{\mathbf{SAT}} \times \mathbf{SAT} = \{(\phi, \psi) \mid \phi \notin \mathbf{SAT} \wedge \psi \in \mathbf{SAT}\}. \\ F &= \mathbf{SAT} \times \{+\} \cup \overline{\mathbf{SAT}} \times \{-\} = \{(\tau, +) \mid \tau \in \mathbf{SAT}\} \cup \{(\tau, -) \mid \tau \notin \mathbf{SAT}\}. \end{aligned}$$

We have D and E in $\mathbf{P}^{\mathbf{NP}^{[2]}} = \mathbf{P}^{\mathbf{NP}^{[1]}}$ by assumption. So there exist polynomial-time computable functions g and h that reduce D to F and E to F respectively.

To help understand our proofs let us review the basic Kadin [Kad88] technique. First let us restrict our attention to formulas of some fixed length. Call a string ϕ *easy* if there is some ψ such that $h(\phi, \psi) = (\tau, +)$ for some satisfiable τ .

The key point to note is that if a formula is *easy* then it has a short proof of nonsatisfiability: The formula ψ and a satisfying assignment for τ . By the definitions of E and h , if τ is satisfiable then ϕ must be unsatisfiable and ψ satisfiable.

If every nonsatisfiable string is easy then every string has a short proof of satisfiability or nonsatisfiability. Otherwise there must exist some nonsatisfiable noneasy string. We call such strings *hard*.

Suppose we had a hard string ϕ and consider $h(\phi, \psi)$. If ψ is satisfiable then $h(\phi, \psi)$ must map to some $(\tau, -)$ or ϕ would have been easy. Also in this case τ would not be satisfiable. If ψ is not satisfiable then either $h(\phi, \psi)$ maps to a $(\tau, +)$ for some nonsatisfiable τ or would map to a $(\tau, -)$ with a satisfiable τ . Given ϕ we have a short proof of nonsatisfiability for ψ : ψ is nonsatisfiable if (1) $h(\phi, \psi) = (\tau, +)$ for some τ , or (2) $h(\phi, \psi) = (\tau, -)$ for some satisfiable τ .

Thus we have that \mathbf{NP} is in \mathbf{coNP} with advice: The advice being a hard string or a bit telling us there are no hard strings.

One simple idea not used by Kadin or the researchers that followed him is that every nonsatisfiable formula is either easy or hard. We use this idea to give stronger collapses under the assumption that $\mathbf{P}^{\mathbf{NP}^{[2]}} = \mathbf{P}^{\mathbf{NP}^{[1]}}$. To make full use of this technique we need to define several kinds of easy and hard strings. Our Easy-I strings are the same as Kadin's. Easy-II strings are formulas with short proofs of nonsatisfiability based on the self-reducibility of \mathbf{SAT} . Easy-III are formulas with a short proof of nonsatisfiability given that they failed to be Easy-II. Easy-IV strings are formulas with a short proof of nonsatisfiability using techniques from Hemaspaandra, Hemaspaandra and Hempel [HHH99a] discussed in Section 3.

Definition 5.4

1. The formula ϕ is Easy-I if there is a ψ such that $h(\phi, \psi) = (\tau, +)$ and $\tau \in \mathbf{SAT}$.
2. The formula ϕ is Easy-II if
 - (a) ϕ is Easy-I, or
 - (b) ϕ is a leaf of a self-reduction tree and false or
 - (c) ϕ self-reduces to two Easy-I formulae.
3. The formula ϕ is Easy-III if
 - (a) ϕ is Easy-II, or
 - (b) There exists an Easy-II formula ψ such that $h(\psi, \phi) = (\tau, -)$ and $\tau \in \mathbf{SAT}$.
4. The formula ϕ is Easy-IV if
 - (a) ϕ is Easy-III, or

- (b) There is a $\psi \in \mathbf{SAT}$ such that $g(\phi, \psi) = (\tau, +)$ and $\tau \in \mathbf{SAT}$.
- (c) There is an Easy-III formula ψ such that $g(\phi, \psi) = (\tau, -)$ and $\tau \in \mathbf{SAT}$.

Nonsatisfiable formulae that are not easy are called hard formulae.

Definition 5.5 The formula ϕ is Hard-I, Hard-II, Hard-III or Hard-IV if $\phi \notin \mathbf{SAT}$ and ϕ is not Easy-I, Easy-II, Easy-III or Easy-IV respectively.

First we argue that for any i in $\{I, II, III, IV\}$, every string is exactly one of Hard- i , Easy- i or satisfiable.

The Easy-I and Hard-I strings are basically the Hard and Easy strings used by Kadin [Kad88]. By looking at the self-reduction tree we will show that if there is a Hard-I string there is a Hard-I string that is Easy-II. This is a Hard-I string whose nonsatisfiability is nondeterministically verifiable.

The Easy-III strings take advantage of this property so that we can use any Hard-I Easy-II string to deterministically separate the Hard-III strings from the satisfiable strings. The Easy-IV definition uses a technique from Hemaspaandra, Hemaspaandra and Hempel [HHH99a] to use Hard-IV strings to deterministically separate the Easy-III strings from the satisfiable strings.

Putting it all together we show that if there are any Hard-IV strings, we will have polynomial advice separating the nonsatisfiable strings from the satisfiable ones.

The following facts are easily derivable from the above definitions.

Lemma 5.6

1. The sets Easy-I, Easy-II, Easy-III and Easy-IV all sit in NP.
2. The sets Hard-I, Hard-II, Hard-III and Hard-IV all sit in coNP.
3. $\text{Easy-I} \subseteq \text{Easy-II} \subseteq \text{Easy-III} \subseteq \text{Easy-IV} \subseteq \overline{\mathbf{SAT}}$.
4. $\text{Hard-IV} \subseteq \text{Hard-III} \subseteq \text{Hard-II} \subseteq \text{Hard-I} \subseteq \overline{\mathbf{SAT}}$.
5. If ϕ is Hard-I then for all ψ , ψ is in \mathbf{SAT} if and only if $h(\phi, \psi) = (\tau, -)$ with $\tau \notin \mathbf{SAT}$.

If we have a Hard-IV formula ϕ , then ϕ gives us a polynomial-time separator between \mathbf{SAT} and the Easy-III formulae:

Lemma 5.7 For any Hard-IV formula ϕ we have

1. If $\psi \in \mathbf{SAT}$ then $g(\phi, \psi) = (\tau, -)$ for some τ .
2. If ψ is Easy-III then $g(\phi, \psi) = (\tau, +)$ for some τ .

Proof: If either of these items were not true we would have τ in \mathbf{SAT} and thus ϕ would be Easy-IV. \square

We also show how to get a separator between \mathbf{SAT} and the Hard-III formulae.

Lemma 5.8 If there is a Hard-I formula ϕ then there is a Hard-I formula α that is Easy-II.

Proof: Consider the self-reduction tree for ϕ . All the formulae in the tree are unsatisfiable. Consider the lowest Hard-I formula α in the tree. Either α is a leaf of the tree or α self-reduces to two Easy-I formulae. \square

We can use the formula α to separate \mathbf{SAT} from the Hard-III formulae.

Lemma 5.9 For any Hard-I Easy-II formula α we have

1. If $\psi \in \mathbf{SAT}$ then $h(\alpha, \psi) = (\tau, -)$ for some τ .
2. If ψ is Hard-III then $h(\alpha, \psi) = (\tau, +)$ for some τ .

Proof: If the first case fails then α is Easy-I. If the second case fails then ψ is Easy-III. \square

Now we can put Lemmas 5.7 and 5.9 together.

Proof of Theorem 5.3(1): We need only prove this item for $A = \overline{\text{SAT}}$ since $\overline{\text{SAT}}$ is complete for coNP and has nice padding properties. Let B be the set of Easy-IV formulae. By Lemma 5.6, B is an NP subset of $\overline{\text{SAT}}$.

Fix n and suppose there is some ϕ of length n in $\overline{\text{SAT}} - B$. By definition ϕ is Hard-IV. By Lemma 5.6, the formula ϕ is also Hard-I so there is some Hard-I Easy-II formula α by Lemma 5.8.

Using as advice ϕ and α and a bit indicating whether or not a Hard-IV formula of length n exists, we define the following \mathbf{P}/poly language C on inputs ψ of length n :

1. If there are no Hard-IV formulae of length n then reject.
2. If $g(\phi, \psi) = (\tau, +)$ for some τ then accept.
3. If $h(\alpha, \psi) = (\tau, +)$ for some τ then accept.
4. Otherwise reject.

If ψ is in SAT then by Lemma 5.7 and 5.9, both lines 2 and 3 will reject. If ψ is in $\overline{\text{SAT}}$ then either ψ is Easy-III or Hard-III. If ψ is Easy-III then line 2 will accept. If ψ is Hard-III then line 3 will accept. \square

Proof of Theorem 5.3(2)($\mathbf{P}^{\text{NP}} = \mathbf{P}^{\text{NP}[1]}$): The proof follows from Lemmas 5.10 and 5.11.

Lemma 5.10 (Chang-Kadin) *If $\mathbf{P}^{\text{NP}[1]} = \mathbf{P}^{\text{NP}[2]}$ then $\mathbf{P}^{\text{NP}[1]} = \mathbf{P}_{tt}^{\text{NP}}$.*

Lemma 5.11 *If $\mathbf{P}^{\text{NP}[1]} = \mathbf{P}^{\text{NP}[2]}$ then $\mathbf{P}_{tt}^{\text{NP}} = \mathbf{P}^{\text{NP}}$.*

Chang and Kadin [CK95] prove Lemma 5.10 by looking at computation trees. Their proof can not be used to generalize the result to k versus $k+1$ queries. We present a different proof using hard and easy strings. Chang [Cha97] uses the ideas of our proofs of Lemma 5.10 and 5.11 to extend Theorem 5.3(2) to show $\mathbf{P}^{\text{NP}[k]} = \mathbf{P}^{\text{NP}[k+1]}$ implies $\mathbf{P}^{\text{NP}[k]} = \mathbf{P}^{\text{NP}}$. He then applies these results to approximation questions of various NP -complete problems.

Proof of Lemma 5.10:

Fix an input x to our $\mathbf{P}_{tt}^{\text{NP}}$ machine M . Let Q be the polynomial-size set of queries to SAT made by $M(x)$. We will show how to compute $\mathbf{P}_{tt}^{\text{NP}}$ with 2 queries to NP , which then by assumption implies that $\mathbf{P}_{tt}^{\text{NP}} = \mathbf{P}^{\text{NP}[1]}$.

For the first query, ask if every member of Q is either satisfiable or Easy-I.

If the answer to the first query is yes then ask if $M(x)$ accepts using “yes” for each satisfiable element of Q and “no” for each Easy-I element of Q .

If the answer to the first query is no then some element of Q is Hard-I. We then ask for our other query whether the following nondeterministic algorithm accepts.

1. Guess S a set of satisfiable formula in Q . Guess satisfying assignments for each element of S .
2. Guess E a set of Easy-I elements in Q . Verify that each of the elements of E is Easy-I.
3. For each ϕ and ψ in $Q - (S \cup E)$ check if $h(\phi, \psi) = (\tau, +)$ for any τ or $h(\phi, \psi) = (\tau, -)$ for some τ in SAT .
4. If all of the above tests pass then simulate M using “yes” for queries in S and “no” for queries in $Q - S$.

If the guess of S and E was such that they contain all of the SAT and Easy-I elements of Q respectively then the remaining formulae are all Hard-I so the third test will pass by Lemma 5.6.

We need to show that if S is not $Q \cap \text{SAT}$ then the above algorithm rejects. Let ϕ be a Hard-I element of Q and ψ be in $Q \cap \text{SAT} - S$. We have ϕ and ψ in $Q - (S \cup E)$. By Lemma 5.6, $h(\phi, \psi) = (\tau, -)$ with $\tau \notin \text{SAT}$ so the third test will fail. \square

Proof of Lemma 5.11: Let M^{SAT} be a \mathbf{P}^{NP} machine that runs in time n^k . Consider the formulae ϕ_i that for each i , $1 \leq i \leq n^k$ encodes: There exists a computation path of $M(x)$ where for the first i queries q_1, \dots, q_i , either q_i is satisfiable and q_i is answered “yes” or q_i is Easy-I and q_i is answered “no”.

Also consider the formulae ψ_i that for each i , $0 \leq i \leq n^k$ encodes: There exists an accepting computation path of $M(x)$ such that

1. for the first i queries q_1, \dots, q_i , either q_i is satisfiable and q_i is answered “yes” or q_i is Easy-I and q_i is answered “no”, and
2. For each $q_j, j > i$, either
 - (a) q_j is answered “yes” and q_j is satisfiable,
 - (b) q_j is answered “no” and $h(q_{i+1}, q_j) = (\tau, +)$ for some τ or
 - (c) q_j is answered “no” and $h(q_{i+1}, q_j) = (\tau, -)$ for some τ in **SAT**.

We ask all of the ϕ_i and ψ_i questions to **SAT** in parallel.

Consider the largest i such that ϕ_i is satisfiable. If $i = n^k$ then $M^{\text{SAT}}(x)$ accepts if and only if ψ_i is satisfiable.

If $i < n^k$ then consider an accepting path encoded by a satisfying assignment of ψ_i . The query q_{i+1} must be Hard-I or ϕ_{i+1} would be satisfiable. By Lemma 5.6, all the answers to the queries are correct. Again we have that $M^{\text{SAT}}(x)$ accepts if and only if ψ_i is satisfiable. \square

Proof of Theorem 5.3(3)($\Sigma_2^p \subseteq \Pi_2^p/1$): Let L be in Σ_2^p . Express L as

$$\{x \mid \exists y \forall z P(x, y, z)\}$$

for some polynomial-time predicate P . Fix x and let ϕ_y encode “ $\exists z \neg P(x, y, z)$ ”. We have $x \in L$ if and only if there exists a y such that $\phi_y \notin \text{SAT}$.

Let $A = \overline{\text{SAT}}$ and B and C be as derived in Theorem 5.3(1). Let $D \in \mathbf{P}$ be such that x is in C iff $(x, a_{|x|})$ is in D where $a_{|x|}$ is the polynomial advice.

Fix an input x of length n . The one bit of advice is whether case (1a) or (1b) of Theorem 5.3 holds for n .

If case (1a) holds then we have x in L if and only if there is a y such that ϕ_y is in B .

If case (1b) holds then we have x in L if and only if for all a_n there exists a formula ψ such that either

- ψ is a leaf of a self-reduction tree and $\psi \in \text{SAT}$ iff $(\psi, a_n) \notin D$, or
- ψ is not a leaf of a self-reduction tree and ψ reduces to ψ_0 and ψ_1 and $(\psi, a_n) \in D$ iff $(\psi_0, a_n) \notin D$ and $(\psi_1, a_n) \notin D$, or
- there is a y such that $(\phi_y, a_n) \notin D$.

The argument for case (1b) is based on the proof by Karp and Lipton [KL80] that if $\mathbf{NP} \subseteq \mathbf{P}/poly$ then $\Sigma_2^p = \Pi_2^p$.

One can verify that in each case we get a Π_2^p expression for L . \square

Whether we can eliminate the advice bit remains an interesting open question.

Proof of Theorem 5.3(4)($\Sigma_2^p = \mathbf{UP}^{\mathbf{NP}[1]}$): Toda and Ogihara [TO92] show that $\mathbf{UP}^{\mathbf{NP}} = \mathbf{UP}^{\mathbf{NP}[1]}$. Hence we only need to prove that $\Sigma_2^p = \mathbf{UP}^{\mathbf{NP}}$.

Consider L, P and the ϕ_y as in the proof of Theorem 5.3(3).

Consider a formula ψ_y that encodes “ ϕ_y is satisfiable or there is some $w < y$ such that $h(\phi_y, \phi_w) = (\tau, +)$ for some τ or $h(\phi_y, \phi_w) = (\tau, -)$ for some $\tau \in \text{SAT}$.”

Our $\mathbf{UP}^{\mathbf{NP}}$ machine works as follows:

1. Query the **NP** oracle to determine if there are any y such that ϕ_y is Easy-I. If so immediately accept.
2. Otherwise accept if there exists a y such that ψ_y is not satisfiable.

If the first step does not accept then all the ψ_y are either satisfiable or Hard-I. If all of the ϕ_y are satisfiable then so are all of the ψ_y . If there is some $w < y$ such that ϕ_w and ϕ_y are both Hard-I then by Lemma 5.6, ψ_y will be satisfiable. If y is the lexicographically least string such that ϕ_y is Hard-I then by Lemma 5.6, ψ_y is not satisfiable. \square

Proof of Theorem 5.3(5)($\Sigma_2^p = \mathbf{RP}^{\mathbf{NP}[1]}$): By Theorem 5.3(2) we only need to prove $\Sigma_2^p = \mathbf{RP}^{\mathbf{NP}}$.

Consider L, P and the ϕ_y as in the proof of Theorem 5.3(3).

Our **RP** algorithm first queries the **NP** oracle to determine if there are any Easy-IV ϕ_y . If so then immediately accept.

If this fails then either all of the ϕ_y are satisfiable or one of them is Hard-IV. If the second condition holds then by the proof of Theorem 5.3(1) there exists polynomial-advice for **SAT**.

Use the algorithm of Bshouty, Cleve, Gavaldà, Kannan and Tamon [BCG⁺96] that randomly using an **NP** oracle finds the advice for **SAT**. If it fails to find the advice for **SAT** then reject. Otherwise query the **NP** oracle again to determine if there is some y such that the advice says ϕ_y is not satisfiable. \square

Proof of Theorem 5.3(6)(PH = BPP^{NP[1]}): Zachos [Zac88] gives a relativizable proof that **NP** \subseteq **BPP** implies **PH** = **BPP**. Relativizing to **SAT** we have $\Sigma_2^p \subseteq \mathbf{BPP}^{\mathbf{NP}}$ implies **PH** = **BPP^{NP}**. The result follows by applying Theorem 5.3((5) and (2)). \square

Corollary 5.12 *If $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$ and **NP** does not have measure zero in **EXP** then **PH** = **P^{NP[1]}**.*

Proof: Lutz [Lut97] shows that if **NP** does not have measure zero in **EXP** then **BPP^{NP}** = **P^{NP}**. \square

6 Open Questions

Theorem 5.3 still leaves many questions open. In particular we do not know whether $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$ implies

1. **PH** = **P^{NP[1]}**
2. $\Sigma_2^p = \Pi_2^p$
3. $\overline{\mathbf{SAT}}$ is the union of an **NP** set and a **BPP/1** set
4. **PH** \subseteq **PP**

even in relativized worlds.

One might also look at implications of related statements on two queries, such as **BPP^{NP[2]}** = **BPP^{NP[1]}**.

Acknowledgments

We thank Leen Torenvliet, Richard Chang, Dieter van Melkebeek and Steve Fenner for helpful discussions and Richard Beigel for comments on an earlier draft.

References

- [BBF98] R. Beigel, H. Buhman, and L. Fortnow. NP might not be as easy as detecting unique solutions. In *Proceedings of the 30th ACM Symposium on the Theory of Computing*, pages 203–208. ACM, New York, 1998.
- [BC97] R. Beigel and R. Chang. Commutative queries. In *Proceedings of the 5th Israeli Symposium on Theory of Computing and Systems*, pages 159–165, 1997. To appear in *Information and Computation*.
- [BCG⁺96] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52(3):421–433, June 1996.
- [BCO93] R. Beigel, R. Chang, and M. Ogihara. A relationship between difference hierarchies and relativized polynomial hierarchies. *Mathematical Systems Theory*, 26:293–310, 1993.
- [BDG88] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Springer, 1988.
- [BDG90] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II*. Springer-Verlag, 1990.
- [Cha97] R. Chang. Bounded queries, approximations and the boolean hierarchy. Technical Report TR CS-97-04, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1997. To appear in *Information and Computation*.
- [CK95] R. Chang and J. Kadin. On computing boolean connectives of characteristic functions. *Mathematical Systems Theory*, 28:173–198, 1995.

- [CK96] R. Chang and J. Kadin. The Boolean hierarchy and the polynomial hierarchy: A closer connection. *SIAM Journal on Computing*, 25(2):340–354, April 1996.
- [FR94] L. Fortnow and J. Rogers. Separability and one-way functions. In *Proceedings of the 5th Annual International Symposium on Algorithms and Computation*, volume 834 of *Lecture Notes in Computer Science*, pages 396–404. Springer, Berlin, 1994.
- [GJ79] M. Garey and D. Johnson. *Computers and intractability. A Guide to the theory of NP-completeness*. W. H. Freeman and Company, New York, 1979.
- [HHH99a] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. A downward collapse within the polynomial hierarchy. *SIAM Journal on Computing*, 28(2):383–393, 1999.
- [HHH99b] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Extending downward collapse from 1-versus-2 queries to j -versus- $j + 1$ queries. In *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 270–280. Springer, Berlin, 1999.
- [HKR93] S. Homer, S. Kurtz, and J. Royer. A note on many-one and 1-truth table complete sets. *Theoretical Computer Science*, 115(2):383–389, July 1993.
- [HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass., 1979.
- [Kad88] J. Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, December 1988.
- [KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on the Theory of Computing*, pages 302–309. ACM, New York, 1980.
- [Kre88] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.
- [Lut97] J. Lutz. Observations on measure and lowness for Δ_2^p . *Theory of Computing Systems*, 30(4):429–442, July/August 1997.
- [Rog97] J. Rogers. The isomorphism conjecture holds and one-way functions exist relative to an oracle. *Journal of Computer and System Sciences*, 54(3):412–423, June 1997.
- [TO92] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2):316–328, 1992.
- [Yap83] C. Yap. Some consequences of nonuniform conditions on uniform classes. *Theoretical Computer Science*, 26:287–300, 1983.
- [Zac88] S. Zachos. Probabilistic quantifiers and games. *Journal of Computer and System Sciences*, 36:433–451, 1988.