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

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.

*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.

One would like to prove that $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$ implies $\mathbf{NP} = \mathbf{coNP}$. Hemaspaandra, Hemaspaandra and Hempel [HHH97a] 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 **NP** = **coNP** or **NP** has polynomial-size circuits.
- $\mathbf{P}^{\mathbf{NP}} = \mathbf{P}^{\mathbf{NP}[1]}$.
- $\Sigma_2^p = \mathbf{UP}^{\mathbf{NP}[1]} \cap \mathbf{RP}^{\mathbf{NP}[1]}$.
- $\mathbf{PH} = \mathbf{BPP}^{\mathbf{NP}[1]}$.

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 $x \in A$.

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 $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}_{tt}^{\mathbf{NP}[2]}$.

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.

1093-0159/98 $10.00 © 1998 IEEE

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.

A function h is polynomial bounded if for all $n > 0$, $h(n) \leq p(n)$ for some polynomial p. Let $\mathcal{C}$ be a complexity class, a set A is in $\mathcal{C}/poly$ if there exists a polynomial bounded function h, and a set $B \in \mathcal{C}$ such that $x \in A$ iff $<x, h(|x|)> \in B$.

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 $\phi'(x_1, \ldots, x_m)$ is a node in the tree then $\phi'(x_1 := \text{true}, x_2, \ldots, x_m)$ and $\phi'(x_1 := \text{false}, x_2, \ldots, x_m)$ are the two children of ϕ'. We say $\phi'(x_1, \ldots, x_m)$ *self-reduces* to the formulae $\phi'(x_1 := \text{true}, x_2, \ldots, x_m)$ and $\phi'(x_1 := \text{false}, x_2, \ldots, x_m)$. 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 [HHH97a] exhibited a strong collapse if $\mathbf{P}^{\Sigma_k^p[1]} = \mathbf{P}_{tt}^{\Sigma_k^p[2]}$ for $k > 2$.

Theorem 3.1 (HHH) *For every level $k > 2$, 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 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

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

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

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

where the quantification is done over strings of length n.

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

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

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

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

By assumption, there exists a polynomial-time computable function $h : \Sigma^* \times \Sigma^* \to \Sigma^* \times \{+,-\}$ such that $(x,\phi) \in D$ if $h(x,\phi) = (z,+)$ 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 bottom row. 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 rows 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 bottom row is true then the self-reduction will always find the appropriate v.

If the assumption in the bottom row 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 row accepting respectively. □

Hemaspaandra, Hemaspaandra and Hempel [HHH97a] 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 [HHH97b] 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 **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 **UP**-generics as developed by Fortnow and Rogers [FR94]. To create a **UP**-generic start with an oracle like **TQBF** that makes $\mathbf{P} = \mathbf{PSPACE}$ and add a generic set restricted to have at most one string at lengths that are towers of 2 and no strings at any other lengths. **UP**-generics also plays 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 the oracle. 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 **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 **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 **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 **NP** question and accept if and only if $M^G(x)$ accepts. □

As a bonus we get the following corollary about complete sets for **PSPACE**.

Corollary 4.3 *There exists a relativized world where the 1-tt-complete degree for* **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 **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 **NP** is not the same as the many-one complete degree.

5 Collapses for $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$

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 (BCO) *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 [HHH97a], 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* **coNP**, $A = B \cup C$ *where* B *is in* **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. $\Sigma_2^p = \mathbf{UP}^{\mathbf{NP}[1]}$.
4. $\Sigma_2^p = \mathbf{RP}^{\mathbf{NP}[1]}$.
5. $\mathbf{PH} = \mathbf{BPP}^{\mathbf{NP}[1]}$.

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

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

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

Informally, we use the term *easy* to denote formulae that have proofs of nonsatifiability. Formally, we will need several types of easy formulae.

Definition 5.4

1. *The formula ϕ is* Easy-I *if there is a ψ such that $h(\phi,\psi) = (\tau,+)$ and $\tau \in$ **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 $\tau \in$ **SAT**.*
4. *The formula ϕ is* Easy-IV *if*
   (a) *ϕ is Easy-III, or*
   (b) *There is a $\psi \in$ **SAT** such that $g(\phi,\psi) = (\tau,+)$ and $\tau \in$ **SAT**.*
   (c) *There is an Easy-III formula ψ such that $g(\phi,\psi) = (\tau,-)$ and $\tau \in$ **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 $\phi \notin$ **SAT** and ϕ is not Easy-I, Easy-II, Easy-III or Easy-IV respectively.*

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. *Easy-I $\subseteq$ Easy-II $\subseteq$ Easy-III $\subseteq$ Easy-IV $\subseteq \overline{\mathbf{SAT}}$.*
4. *Hard-IV $\subseteq$ Hard-III $\subseteq$ Hard-II $\subseteq$ Hard-I $\subseteq \overline{\mathbf{SAT}}$.*
5. *If ϕ is Hard-I then for all ψ, ψ is in* **SAT** *if and only if $h(\phi,\psi) = (\tau,-)$ with $\tau \notin$ **SAT**.*

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

Lemma 5.7 *For any Hard-IV formula ϕ we have*

1. *If $\psi \in$ **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 **SAT** and thus ϕ would be Easy-IV. □

We also show how to get a separator between **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. □

We can use the formula α to separate **SAT** from the Hard-III formulae.

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

1. *If $\psi \in$ **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. □

Now we can put Lemmas 5.7 and 5.9 together.

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

Fix n and suppose there is some ϕ of length n in $\overline{\mathbf{SAT}} - B$. 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 $\overline{\textbf{SAT}}$ 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. □

Proof of Theorem 5.3(2): The proof follows from Lemmas 5.10 and 5.11.

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

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

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 $\mathbf{P}^{\mathbf{NP}[k]} = \mathbf{P}^{\mathbf{NP}[k+1]}$ implies $\mathbf{P}^{\mathbf{NP}[k]} = \mathbf{P}^{\mathbf{NP}}$. He then applies these results to approximation questions of various **NP**-complete problems.

Proof of Lemma 5.10:

Fix an input x to our $\mathbf{P}^{\mathbf{NP}}_{tt}$ machine M. Let Q be the polynomial-size set of queries to **SAT** made by $M(x)$.

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 S and E guess 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 $Q \cap$ **SAT** then the above algorithm rejects. Let ϕ be a Hard-I element of Q and ψ be in $Q\cap\textbf{SAT}-S$. We have ϕ and ψ in $Q-(S\cup E)$. By Lemma 5.6, $h(\phi,\psi) = (\tau,-)$ with $\tau \notin$ **SAT** so the third test will fail. □

Proof of Lemma 5.11: Let $M^{\mathbf{SAT}}$ be a $\mathbf{P}^{\mathbf{NP}}$ machine that runs in time n^k. Consider the formulae ϕ_i that for each i, $1 \le i \le n^k$ encodes: There exists a computation path of $M(x)$ where for the first i queries $q_1,\ldots,q_i$, 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, $0 \le i \le n^k$ encodes: There exists an accepting computation path of $M(x)$ such that

1. for the first i queries $q_1,\ldots,q_i$, 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 a **SAT** oracle in parallel.

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

Proof of Theorem 5.3(3): Toda and Ogihara [TO92] show that $\mathbf{UP}^{\mathbf{NP}} = \mathbf{UP}^{\mathbf{NP}[1]}$. We need only prove $\Sigma^p_2 = \mathbf{UP}^{\mathbf{NP}}$.

Let L be in Σ^p_2. 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 $\phi_y \notin$ **SAT**.

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 $\tau \in$ **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. □

Proof of Theorem 5.3(4): By Theorem 5.3(2) we need only 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 advice 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. □

Proof of Theorem 5.3(5): Zachos [Zac88] gives a relativizable proof that $\mathbf{NP} \subseteq \mathbf{BPP}$ implies $\mathbf{PH} = \mathbf{BPP}$. Relativizing to **SAT** we have $\Sigma_2^p \subseteq \mathbf{BPP}^{\mathbf{NP}}$ implies $\mathbf{PH} = \mathbf{BPP}^{\mathbf{NP}}$. The result follows by applying Theorem 5.3((4) and (2)). □

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

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

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

even in relativized worlds.

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

Acknowledgments

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

References

[BBF98] R. Beigel, H. Buhrman, 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*. ACM, New York, 1998. To appear.

[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.

[HHH97a] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. A downward translation in the polynomial hierarchy. In *Proceedings of the 14th Symposium on Theoretical Aspects of Computer Science*, volume 1200 of *Lecture Notes in Computer Science*, pages 319–328. Springer, Berlin, 1997. To appear in the SIAM Journal on Computing.

[HHH97b] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Translating equality downward. Technical Report TR 97-657, University of Rochester Department of Computer Science, April 1997.

[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.

[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.