A Downward Collapse within the Polynomial Hierarchy

Edith Hemaspaandra* Lane A. Hemaspaandra† Harald Hempel‡

June 16, 1997

Abstract

Downward collapse (a.k.a. upward separation) refers to cases where the equality of two larger classes implies the equality of two smaller classes. We provide an unqualified downward collapse result completely within the polynomial hierarchy. In particular, we prove that, for $k > 2$, if $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]}$ then $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$. We extend this to obtain a more general downward collapse result.

1 Introduction

The theory of NP-completeness does not resolve the issue of whether P and NP are equal. However, it does unify the issues of whether thousands of natural problems—the NP-complete problems—have deterministic polynomial-time algorithms. The study of downward collapse is similar in spirit. By proving downward collapses, we seek to tie together central open issues regarding the computing power of complexity classes. For example, the main result of this paper shows that (for $k > 2$) the issue of whether the kth level of the polynomial hierarchy is closed under complementation is identical to the issue of whether two queries to this level give more power than one query to this level.

Informally, downward collapse (equivalent terms are "downward translation of equality" and "upward separation") refers to cases in which the collapse of larger classes implies the collapse of smaller classes (for background, see, e.g., [All91,AW90]). For example, $\mathrm{NP}^{\mathrm{NP}} = \mathrm{coNP}^{\mathrm{NP}} \Rightarrow \mathrm{NP} = \mathrm{coNP}$ would be a (shocking, and inherently nonrelativizing [Ko89]) downward collapse, the "downward" part referring to the well-known fact that $\mathrm{NP} \cup \mathrm{coNP} \subseteq \mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$.

Downward collapse results are extremely rare, but there are some results in the literature that do have the general flavor of downward collapse. Cases where the collapse of larger

*Department of Mathematics, Le Moyne College, Syracuse, NY 13214, USA. Supported in part by grant NSF-INT-9513368/DAAD-315-PRO-fo-ab. Work done in part while visiting Friedrich-Schiller-Universität Jena.

†Department of Computer Science, University of Rochester, Rochester, NY 14627, USA. Supported in part by grants NSF-CCR-9322513 and NSF-INT-9513368/DAAD-315-PRO-fo-ab. Work done in part while visiting Friedrich-Schiller-Universität Jena.

‡Inst. für Informatik, Friedrich-Schiller-Universität Jena, 07743 Jena, Germany. Supported in part by grant NSF-INT-9513368/DAAD-315-PRO-fo-ab. Work done in part while visiting Le Moyne College.

classes forces sparse sets (but perhaps not non-sparse sets) to fall out of smaller classes were found by Hartmanis, Immerman, and Sewelson ([HIS85], see also [Boo74]) and by others (e.g., Rao, Rothe, and Watanabe [RRW94], but in contrast see also [HJ95]). Existential cases have long been implicitly known (i.e., theorems such as "If PH = PSPACE then $(\exists k)\,[\mathrm{PH} = \Sigma^p_k]$"—note that here one can prove nothing about what value k might have). Regarding probabilistic classes, Ko [Ko82] proved that "If NP $\subseteq$ BPP then NP = R," and Babai, Fortnow, Nisan, and Wigderson [BFNW93] proved the striking result that "If EH = E then P = BPP." Hemaspaandra, Rothe, and Wechsung have given an example involving degenerate certificate schemes [HRW], and examples due to Allender [All86, Section 5] and Hartmanis and Yesha [HY84, Section 4] are known regarding circuit-related classes.[1]

We provide an unqualified downward collapse result that is not restricted to sparse or tally sets, whose conclusion does not contain a variable that is not specified in its hypothesis, and that deals with classes whose *ex ante* containments[2] are clear (and plausibly strict). Namely, as is standard, let $\mathrm{P}^{\mathcal{C}[j]}$ denote the class of languages computable by P machines making at most j queries to some set from $\mathcal{C}$. We prove that, for each $k > 2$, it holds that

$$\mathrm{P}^{\Sigma^p_k[1]} = \mathrm{P}^{\Sigma^p_k[2]} \Rightarrow \Sigma^p_k = \Pi^p_k = \mathrm{PH}.$$

(As just mentioned in footnote 2, the classes in the hypothesis clearly have the property that they contain both Σ^p_k and Π^p_k.) The best previously known results from the assumption $\mathrm{P}^{\Sigma^p_k[1]} = \mathrm{P}^{\Sigma^p_k[2]}$ collapse the polynomial hierarchy only to a level that contains Σ^p_{k+1} and Π^p_{k+1} [CK96,BCO93].

Our proof actually establishes a $\Sigma^p_k = \Pi^p_k$ collapse from a hypothesis that is even weaker than $\mathrm{P}^{\Sigma^p_k[1]} = \mathrm{P}^{\Sigma^p_k[2]}$. Namely, we prove that, for $i < j < k$ and $i < k - 2$, if one query each (in parallel) to the ith and kth levels of the polynomial hierarchy equals one query each (in parallel) to the jth and kth levels of the polynomial hierarchy, then $\Sigma^p_k = \Pi^p_k = \mathrm{PH}$.

In the final section of the paper, we generalize from 1-versus-2 queries to m-versus-$(m+1)$ queries. In particular, we show that our main result is in fact a reflection of an even more general downward collapse: If the truth-table hierarchy over Σ^p_k collapses to its mth level, then the boolean hierarchy over Σ^p_k collapses one level further than one would expect.

2 Simple Case

Our proof works by extracting advice internally and algorithmically, while holding down the number of quantifiers needed, within the framework of a so-called "easy-hard" argument.

[1] Note that we are not claiming that all the above examples from the literature are totally unqualified downward collapse results, but rather we are merely stating that they have the strong general flavor of downward collapse. In some cases, the results mentioned above do not fully witness what one might hope for from the notion of "downward." Ideally, downward collapse results would be truly "downward" in the sense that they would be of the form "If $\mathcal{A} = \mathcal{B}$ then $\mathcal{C} = \mathcal{D}$," where the classes are such that (a) $\mathcal{A} \cap \mathcal{B} \supseteq \mathcal{C} \cup \mathcal{D}$ is a well-known result, and (b) it is not currently known that $\mathcal{A} \cap \mathcal{B} = \mathcal{C} \cup \mathcal{D}$. The downward collapses proven in this paper do have this strong "downward" form.

[2] I.e., in the case of Theorem 2.1, $\Sigma^p_k \cup \Pi^p_k \subseteq \mathrm{P}^{\Sigma^p_k[1]} \cap \mathrm{P}^{\Sigma^p_k[2]}$ is well-known to be true (and most researchers suspect that the inclusion is strict).

Easy-hard arguments were introduced by Kadin [Kad88], and were further used by Chang and Kadin ([CK96], see also [Cha91]) and Beigel, Chang, and Ogihara [BCO93] (we follow the approach of Beigel, Chang, and Ogihara).

Theorem 2.1 *For each $k > 2$ it holds that:*

$$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]} \Rightarrow \Sigma_k^p = \Pi_k^p = \mathrm{PH}.$$

Theorem 2.1 follows immediately[3] from Theorem 2.4 below, which states that, for $i < j < k$ and $i < k-2$, if one query each to the ith and kth levels of the polynomial hierarchy equals one query each to the jth and kth levels of the polynomial hierarchy, then $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$.

DPTM will refer to deterministic polynomial-time oracle Turing machines, whose polynomial time upper-bounds are clearly clocked, and are independent of their oracles. We will also use the following definitions.

Definition 2.2 *1. Let $M^{(A,B)}$ denote DPTM M making, simultaneously (i.e., in a truth-table fashion), at most one query to oracle A and at most one query to oracle B, and let*

$$\mathrm{P}^{(\mathcal{C},\mathcal{D})} = \{L \subseteq \Sigma^* \mid (\exists C \in \mathcal{C})(\exists D \in \mathcal{D})(\exists \text{DPTM } M)[L = L(M^{(C,D)})]\}.$$

2. (see [BCO93]) $A\tilde{\Delta}B = \{\langle x, y\rangle \mid x \in A \Leftrightarrow y \notin B\}$.

Lemma 2.3 *Let $0 \leq i < k$, let $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ be any set $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}^{\Sigma_i^p[1]}$, and let $L_{\Sigma_k^p}$ be any language $\leq_{\mathrm{m}}^p$-complete for Σ_k^p. Then $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$ is $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$.*

Proof

Clearly $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$ is in $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$. Regarding $\leq_{\mathrm{m}}^p$-hardness for $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$, let $L \in \mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$ via transducer M, Σ_i^p set A, and Σ_k^p set B. Without loss of generality, on each input x, M asks exactly one question a_x to A, and one question b_x to B. Define sets D and E as follows:

$D = \{x \mid M^{(A,B)}$ accepts x if a_x is answered correctly, and b_x is answered "no"$\}$.

$E = \{x \mid b_x \in B$ and the (one-variable) truth-table with respect to b_x of $M^{(A,B)}$ on input x induced by the correct answer to a_x is neither "always accept" nor "always reject"$\}$.

Note that $D \in \mathrm{P}^{\Sigma_i^p[1]}$, and that $E \in \Sigma_k^p$, since $i < k$. But $L \leq_{\mathrm{m}}^p D\tilde{\Delta}E$ via the reduction $f(x) = \langle x, x\rangle$. So clearly $L \leq_{\mathrm{m}}^p L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$, via the reduction $\widehat{f}(x) = \langle f'(x), f''(x)\rangle$, where f' and f'' are, respectively, reductions from D to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ and from E to $L_{\Sigma_k^p}$. ∎

[3] In particular, taking $i = 0$ and $j = k-1$ in Theorem 2.4 yields a statement that itself clearly implies Theorem 2.1.

Theorem 2.4 contains the following two technical advances. First, it internally extracts information in a way that saves a quantifier. (In contrast, the earliest easy-hard arguments in the literature merely ensure that $\Sigma_k^p \subseteq \Pi_k^p/\text{poly}$ and from that infer a weak polynomial hierarchy collapse. Even the interesting recent strengthenings of the argument [BCO93] still, under the hypothesis of Theorem 2.4, conclude only a collapse of the polynomial hierarchy to a level a bit above Σ_{k+1}^p.) The second advance is that previous easy-hard arguments seek to determine whether there exists a hard string for a length or not. Then they use the fact that if there is not a hard string, all strings (at the length) are easy. In contrast, we *never* search for a hard string; rather, we use the fact that the input itself (which we do not have to search for as, after all, it is our input) is either easy or hard. So we check whether the input is easy, and if so we can use it as an easy string, and if not, it must be a hard string so we can use it that way. This innovation is important in that it allows Theorem 2.1 to apply for all $k > 2$—as opposed to merely applying for all $k > 3$, which is what we would get without this innovation. (Following a referee's suggestion, we mention that during a first traversal the reader may wish to consider just the $i = 0$ and $j = 1$ special case of Theorem 2.4 and its proof, as this provides a restricted version that is easier to read.)

Theorem 2.4 *Let $0 \leq i < j < k$ and $i < k-2$. If $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)} = \mathrm{P}^{(\Sigma_j^p,\Sigma_k^p)}$ then $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$.*

Proof

Suppose $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)} = \mathrm{P}^{(\Sigma_j^p,\Sigma_k^p)}$. Let $L_{\mathrm{P}^{\Sigma_i^p[1]}}$, $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$, and $L_{\Sigma_k^p}$ be $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}^{\Sigma_i^p[1]}$, $\mathrm{P}^{\Sigma_{i+1}^p[1]}$, and Σ_k^p, respectively; such sets exist. From Lemma 2.3 it follows that $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$ is $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$. Since (as $i < j$) $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\tilde{\Delta}L_{\Sigma_k^p} \in \mathrm{P}^{(\Sigma_j^p,\Sigma_k^p)}$, and by assumption $\mathrm{P}^{(\Sigma_j^p,\Sigma_k^p)} = \mathrm{P}^{(\Sigma_i^p,\Sigma_k^p)}$, there exists a polynomial-time many-one reduction h from $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\Sigma_k^p}$. So, for all $x_1, x_2 \in \Sigma^*$: if $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$, then $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow x_2 \notin L_{\Sigma_k^p})$ if and only if $(y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}} \Leftrightarrow y_2 \notin L_{\Sigma_k^p})$. Equivalently, for all $x_1, x_2 \in \Sigma^*$:

> **Fact 1:**
> if $h(\langle x_1, x_2\rangle) = \langle y_1, y_2\rangle$,
> then
>
> $$(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow x_2 \in L_{\Sigma_k^p}) \text{ if and only if } (y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}} \Leftrightarrow y_2 \in L_{\Sigma_k^p}).$$

We can use h to recognize some of $\overline{L_{\Sigma_k^p}}$ by a Σ_k^p algorithm. The definitions of easy and hard used in this paper follow the easy and hard concepts used by Kadin [Kad88], Chang and Kadin ([CK96], see also [Cha91]), and Beigel, Chang, and Ogihara [BCO93], modified as needed for our goals. In particular, we say that a string x is *easy for length* n if there exists a string x_1 such that $|x_1| \leq n$ and $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$ where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

Let p be a fixed polynomial, which will be exactly specified later in the proof. We have the following Σ_k^p algorithm to test whether $x \in \overline{L_{\Sigma_k^p}}$ in the case that (our input) x is an

easy string for $p(|x|)$. On input x, guess x_1 with $|x_1| \leq p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$ and $y_2 \in L_{\Sigma_k^p}$. In light of Fact 1 above, it is clear that this is correct.

We say that x is *hard for length* n if $|x| \leq n$ and x is not easy for length n, i.e., if $|x| \leq n$ and for all x_1 with $|x_1| \leq n$, $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}})$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

If x is a hard string for length n, then x induces a many-one reduction from $\left(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\right)^{\leq n}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$, namely, $f(x_1) = y_1$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$. Note that f is computable in time polynomial in $\max(n, |x_1|)$.

We can use hard strings to obtain a Σ_k^p algorithm for $\overline{L_{\Sigma_k^p}}$. Let M be a Π_{k-i-1}^p machine such that M with oracle $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$ recognizes $\overline{L_{\Sigma_k^p}}$. Let the run-time of M be bounded by polynomial p, which without loss of generality satisfies $(\forall \widehat{m} \geq 0)[p(\widehat{m}+1) > p(\widehat{m}) > 0]$ (as promised above, we have now specified p). Then

$$\left(\overline{L_{\Sigma_k^p}}\right)^{=n} = L(M^{\left(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\right)^{\leq p(n)}})^{=n}.$$

If there exists a hard string for length $p(n)$, then this hard string induces a reduction from $\left(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\right)^{\leq p(n)}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$. Thus, with any hard string for length $p(n)$ in hand, call it w_n, $\widehat{M}$ with oracle $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ recognizes $\overline{L_{\Sigma_k^p}}$ for strings of length n, where $\widehat{M}$ is the machine that simulates M but replaces each query to q by the first component of $h(\langle q, w_n\rangle)$. It follows that if there exists a hard string for length $p(n)$, then this string induces a Π_{k-1}^p algorithm for $\left(\overline{L_{\Sigma_k^p}}\right)^{=n}$, and therefore certainly a Σ_k^p algorithm for $\left(\overline{L_{\Sigma_k^p}}\right)^{=n}$.

However, now we have an $\mathrm{NP}^{\Sigma_{k-1}^p} = \Sigma_k^p$ algorithm for $\overline{L_{\Sigma_k^p}}$: On input x, the NP base machine of $\mathrm{NP}^{\Sigma_{k-1}^p}$ executes the following algorithm:

1. Using its Σ_{k-1}^p oracle, it deterministically determines whether the input x is an easy string for length $p(|x|)$. This can be done, as checking whether the input is an easy string for length $p(|x|)$ can be done by one query to Σ_{i+2}^p, and $i+2 \leq k-1$ by our $i < k-2$ hypothesis.

2. If the previous step determined that the input is not an easy string, then the input must be a hard string for length $p(|x|)$. So simulate the Σ_k^p algorithm induced by this hard string (i.e., the input x itself) on input x (via our NP machine itself simulating the base level of the Σ_k^p algorithm and using the NP machine's oracle to simulate the oracle queries made by the base level NP machine of the Σ_k^p algorithm being simulated).

3. If the first step determined that the input x is easy for length $p(|x|)$, then our NP machine simulates (using itself and its oracle) the Σ_k^p algorithm for easy strings on input x.

We need one brief technical comment. The Σ_{k-1}^p oracle in the above algorithm is being used for a number of different sets. However, as Σ_{k-1}^p is closed under disjoint union, this presents no problem as we can use the disjoint union of the sets, while modifying the queries so they address the appropriate part of the disjoint union.

Since $\overline{L_{\Sigma_k^p}}$ is complete for Π_k^p, it follows that $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$. ∎

We conclude this section with three remarks. First, if one is fond of the truth-table version of bounded query hierarchies, one can certainly replace the hypothesis of Theorem 2.1 with $\mathrm{P}_{\text{1-tt}}^{\Sigma_k^p} = \mathrm{P}_{\text{2-tt}}^{\Sigma_k^p}$ (both as this is an equivalent hypothesis, and as it in any case clearly follows from Theorem 2.4). Indeed, one can equally well replace the hypothesis of Theorem 2.1 with the even weaker-looking hypothesis[4] $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p)$ (as this hypothesis is also in fact equivalent to the hypothesis of Theorem 2.1—just note that if $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p)$ then $\mathrm{DIFF}_2(\Sigma_k^p)$ is closed under complementation and thus equals the boolean hierarchy over Σ_k^p, see [CGH+88], and so in particular we then have $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p) = \mathrm{P}^{\Sigma_k^p[2]}$).

Of course, the two equivalences just mentioned—$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{\Sigma_k^p[2]} \Leftrightarrow \mathrm{P}_{\text{1-tt}}^{\Sigma_k^p} = \mathrm{P}_{\text{2-tt}}^{\Sigma_k^p} \Leftrightarrow \mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p)$—are well-known. However, Theorem 2.4 is sufficiently strong that it creates an equivalence that is quite new, and somewhat surprising. We state it below as Corollary 2.6.

Theorem 2.5 *For each $k > 2$ it holds that:*

$$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p) \cap \mathrm{coDIFF}_2(\Sigma_k^p) \Rightarrow \Sigma_k^p = \Pi_k^p = \mathrm{PH}.$$

Proof

Let $A \triangle B =_{\mathrm{def}} (A - B) \cup (B - A)$. Recalling that $k > 2$, it is not hard to see that $\mathrm{P}^{(\mathrm{NP},\Sigma_k^p)} \subseteq \mathrm{DIFF}_2(\Sigma_k^p)$. In particular, this holds due to Lemma 2.3, in light of the facts that (i) $\mathrm{DIFF}_2(\Sigma_k^p) = \{L \mid (\exists L_1 \in \Sigma_k^p)(\exists L_2 \in \Sigma_k^p)[L = L_1 \triangle L_2]\}$ (due to Köbler, Schöning, and Wagner [KSW87]—see the discussion just before Theorem 3.7), and (ii) $A\tilde{\Delta}B = \{\langle x, y\rangle \mid x \in A\} \triangle \{\langle x, y\rangle \mid y \in B\}$. So, since $\mathrm{P}^{(\mathrm{NP},\Sigma_k^p)}$ is closed under complementation, we have $\mathrm{P}^{\Sigma_k^p[1]} \subseteq \mathrm{P}^{(\mathrm{NP},\Sigma_k^p)} \subseteq \mathrm{DIFF}_2(\Sigma_k^p) \cap \mathrm{coDIFF}_2(\Sigma_k^p)$. However, this says, under the hypothesis of the theorem, that $\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{P}^{(\mathrm{NP},\Sigma_k^p)}$, which itself, by Theorem 2.4, implies that $\Sigma_k^p = \Pi_k^p = \mathrm{PH}$. ∎

Corollary 2.6 *For each $k > 2$ it holds that:*

$$\mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p) \cap \mathrm{coDIFF}_2(\Sigma_k^p) \Leftrightarrow \mathrm{P}^{\Sigma_k^p[1]} = \mathrm{DIFF}_2(\Sigma_k^p).$$

Our second remark is that Theorem 2.1 implies that, for $k > 2$, if the bounded query hierarchy over Σ_k^p collapses to its $\mathrm{P}^{\Sigma_k^p[1]}$ level, then the bounded query hierarchy over Σ_k^p equals the polynomial hierarchy (this provides a partial answer to the issue of whether, when a bounded query hierarchy collapses, the polynomial hierarchy necessarily collapses to it, see [HRZ95, Problem 4]).

[4] Where $\mathrm{DIFF}_2(\mathcal{C}) =_{\mathrm{def}} \{L \mid (\exists L_1 \in \mathcal{C})(\exists L_2 \in \mathcal{C})[L = L_1 - L_2]\}$, and $\mathrm{co}\mathcal{C} =_{\mathrm{def}} \{L \mid \overline{L} \in \mathcal{C}\}$ (see Definition 3.1 for background).

Third, in Lemma 2.3 and Theorem 2.4 we speak of classes of the form $\mathrm{P}^{(\Sigma_i^p,\Sigma_j^p)}$, $i \neq j$. It would be very natural to reason as follows: "$\mathrm{P}^{(\Sigma_i^p,\Sigma_j^p)}$, $i \neq j$, must equal $\mathrm{P}^{\Sigma^p_{\max(i,j)}[1]}$, as $\Sigma^p_{\max(i,j)}$ can easily solve any $\Sigma^p_{\min(i,j)}$ query "strongly" using the $\Sigma^p_{\max(i,j)-1}$ oracle of its base NP machine and thus the hypothesis of Theorem 2.4 is trivially satisfied and so you in fact are claiming to prove, unconditionally, that $\mathrm{PH} = \Sigma^p_3$. This reasoning, though tempting, is wrong for the following somewhat subtle reason. Though it is true that, for example, $\mathrm{NP}^{\Sigma^p_q}$ can solve any Σ^p_q query and then can tackle any Σ^p_{q+1} query, it does not follow that $\mathrm{P}^{(\Sigma^p_{q+1},\Sigma^p_q)} = \mathrm{P}^{\Sigma^p_{q+1}[1]}$. The problem is that the answer to the Σ^p_q query may *change the truth-table* the P transducer uses to evaluate the answer of the Σ^p_{q+1} query.

We mention that Buhrman and Fortnow [BF96], building on and extending our proof technique, have very recently obtained the $k = 2$ analog of Theorem 2.1. They also prove that there are relativized worlds in which the $k = 1$ analog of Theorem 2.1 fails. On the other hand, if one changes Theorem 2.1's left-hand-side classes to function classes, then the $k = 1$ analog of the resulting claim does hold due to Krentel (see [Kre88, Theorem 4.2]): $\mathrm{FP}^{\mathrm{NP}[1]} = \mathrm{FP}^{\mathrm{NP}[2]} \Rightarrow \mathrm{P} = \mathrm{PH}$. Also, very recent work of Hemaspaandra, Hemaspaandra, and Hempel [HHH97], building on the techniques of the present paper and those of Buhrman and Fortnow [BF96], has established the $k = 2$ analog of Theorem 3.2.

3 General Case

We now generalize the results of Section 2 to the case of m-truth-table reductions. Though the results of this section are stronger than those of Section 2, the proofs are somewhat more involved, and thus we suggest the reader first read Section 2.

For clarity, we now describe the two key differences between the proofs in this section and those of Section 2. (1) The completeness claims of Section 2 were simpler. Here, we now need Lemma 3.5, which extends [BCO93, Lemma 8] with the trick of splitting a truth-table along a simple query's dimension in such a way that the induced one-dimension-lower truth-tables cause no problems. (2) The proof of Theorem 3.6 is quite analogous to the proof of Theorem 2.4, except (i) it is a bit harder to understand as one continuously has to parse the deeply nested set differences caused by the fact that we are now working in the difference hierarchy, and (ii) the "input is an easy string" simulation is changed to account for a new problem, namely, that in the boolean hierarchy one models each language by a *collection* of machines (mimicking the nested difference structure of boolean hierarchy languages) and thus it is hard to ensure that these machines, when guessing an object, necessarily guess the same object (we solve this coordination problem by forcing them to each guess a lexicographically extreme object, and we argue that this can be accomplished within the computational power available).

The difference hierarchy was introduced by Cai et al. [CGH$^+$88,CGH$^+$89] and is defined below. Cai et al. studied the case $\mathcal{C} = \mathrm{NP}$, but a number of other cases have since been studied [BJY90,BCO93,HR].

Definition 3.1 *Let $\mathcal{C}$ be any complexity class.*

1. $\mathrm{DIFF}_1(\mathcal{C}) = \mathcal{C}$.
2. *For any* $k \geq 1$, $\mathrm{DIFF}_{k+1}(\mathcal{C}) = \{L \mid (\exists L_1 \in \mathcal{C})(\exists L_2 \in \mathrm{DIFF}_k(\mathcal{C}))[L = L_1 - L_2]\}$.
3. *For any* $k \geq 1$, $\mathrm{coDIFF}_k(\mathcal{C}) = \{L \mid \overline{L} \in \mathrm{DIFF}_k(\mathcal{C})\}$.

Note in particular that

$$\mathrm{DIFF}_m(\Sigma_k^p) \cup \mathrm{coDIFF}_m(\Sigma_k^p) \subseteq \mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} \subseteq \mathrm{DIFF}_{m+1}(\Sigma_k^p) \cap \mathrm{coDIFF}_{m+1}(\Sigma_k^p).$$

Theorem 3.2 *For each* $m > 0$ *and each* $k > 2$ *it holds that:*

$$\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p} \Rightarrow \mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p).$$

Theorem 2.1 is the $m = 1$ case of Theorem 3.2 (except the former is stated in terms of Turing access). Theorem 3.2 follows immediately from Theorem 3.6 below, which states that, for $i < j < k$ and $i < k - 2$, if one query to the ith and m queries to the kth levels of the polynomial hierarchy equals one query to the jth and m queries to the kth levels of the polynomial hierarchy, then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$. Note, of course, that by Beigel, Chang, and Ogihara [BCO93] the conclusion of Theorem 3.2 implies a collapse of the polynomial hierarchy. In particular, via [BCO93, Theorem 10], Theorem 3.2 implies that, for each $m \geq 0$ and each $k > 2$, it holds that: If $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p}$ then the polynomial hierarchy can be solved by a P machine that makes $m - 1$ truth-table queries to Σ_{k+1}^p, and that in addition is allowed unbounded queries to Σ_k^p. This polynomial hierarchy collapse is about one level lower in the difference hierarchy over Σ_{k+1}^p than one could conclude from previous papers, in particular, from Beigel, Chang, and Ogihara. In fact, one can claim a bit more. The *proof* of [BCO93, Theorem 10] in fact proves the following: $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p) \Rightarrow \mathrm{PH} = \mathrm{P}_{1,m-1\text{-tt}}^{(\Sigma_k^p, \Sigma_{k+1}^p)}$. Thus, in light of Theorem 3.2, we have the following corollary.

Corollary 3.3 *For each* $m \geq 0$ *and each* $k > 2$ *it holds that* $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{P}_{m+1\text{-tt}}^{\Sigma_k^p} \Rightarrow \mathrm{PH} = \mathrm{P}_{1,m-1\text{-tt}}^{(\Sigma_k^p, \Sigma_{k+1}^p)}$.

The following definition will be useful.

Definition 3.4 *Let* $M_{a,b\text{-tt}}^{(A,B)}$ *denote DPTM* M *making, simultaneously (i.e., all* $a+b$ *queries are made at the same time, in the standard truth-table fashion), at most* a *queries to oracle* A *and at most* b *queries to oracle* B*, and let*

$$\mathrm{P}_{a,b\text{-tt}}^{(\mathcal{C},\mathcal{D})} = \{L \subseteq \Sigma^* \mid (\exists C \in \mathcal{C})(\exists D \in \mathcal{D})(\exists \mathrm{DPTM}\ M)[L = L(M_{a,b\text{-tt}}^{(C,D)})]\}.$$

Lemma 3.5 *Let* $m > 0$*, let* $0 \leq i < k$*, let* $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ *be any set* $\leq_{\mathrm{m}}^p$*-complete for* $\mathrm{P}^{\Sigma_i^p[1]}$*, and let* $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ *be any language* $\leq_{\mathrm{m}}^p$*-complete for* $\mathrm{DIFF}_m(\Sigma_k^p)$*. Then* $L_{\mathrm{P}^{\Sigma_i^p[1]}} \tilde{\Delta} L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ *is* $\leq_{\mathrm{m}}^p$*-complete for* $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p, \Sigma_k^p)}$.

Lemma 3.5 does not require proof, as it is a use of the standard mind-change technique, and is analogous to [BCO93, Lemma 8], with one key twist that we now discuss. Assume, without loss of generality, that we focus on $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)}$ machines that always make exactly $m+1$ queries. Regarding any such machine accepting a set complete for the class $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)}$ of Lemma 3.5, we have on each input a truth-table with $m+1$ variables. Note that if one knows the answer to the one Σ_i^p query, then this induces a truth-table on m variables; however, note also that the two m-variable truth-tables (one corresponding to a "yes" answer to the Σ_i^p query and the other to a "no" answer) may differ sharply. Regarding $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$, we use $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ to determine whether the m-variable truth-table induced by the true answer to the one Σ_i^p query accepts or not when all the Σ_k^p queries get the answer no. This use is analogous to [BCO93, Lemma 8]. The new twist is the action of the $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ part of $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$. We use this, just as in [BCO93, Lemma 8], to find whether or not we are in an odd mind-change region *but now with respect to the m-variable truth-table induced by the true answer to the one Σ_i^p query.* Crucially, this still is a $\mathrm{DIFF}_m(\Sigma_k^p)$ issue as, since $i<k$, a Σ_k^p machine can first on its own (by its base NP machine making one deterministic query to its Σ_{k-1}^p oracle) determine the true answer to the one Σ_i^p query, and thus the machine can easily know which of the two m-variable truth-table cases it is in, and thus it plays its standard part in determining if the mind-change region of the m true answers to the Σ_k^p queries fall in an odd mind-change region *with respect to the correct m-variable truth-table.*

Theorem 3.6 *Let $m>0$, $0\leq i<j<k$ and $i<k-2$. If $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)}=\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_j^p,\Sigma_k^p)}$ then $\mathrm{DIFF}_m(\Sigma_k^p)=\mathrm{coDIFF}_m(\Sigma_k^p)$.*

Proof

Suppose $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)}=\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_j^p,\Sigma_k^p)}$. Let $L_{\mathrm{P}^{\Sigma_i^p[1]}}$, $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$, and $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ be $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}^{\Sigma_i^p[1]}$, $\mathrm{P}^{\Sigma_{i+1}^p[1]}$, and $\mathrm{DIFF}_m(\Sigma_k^p)$, respectively; such languages exist, e.g., via the standard canonical complete set constructions using enumerations of clocked machines. From Lemma 3.5 it follows that $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ is $\leq_{\mathrm{m}}^p$-complete for $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)}$. Since (as $i<j$) $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}\in\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_j^p,\Sigma_k^p)}$, and by assumption $\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_j^p,\Sigma_k^p)}=\mathrm{P}_{1,m\text{-tt}}^{(\Sigma_i^p,\Sigma_k^p)}$, there exists a polynomial-time many-one reduction h from $L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}\tilde{\Delta}L_{\mathrm{DIFF}_m(\Sigma_k^p)}$. So, for all $x_1,x_2\in\Sigma^*$:

if $h(\langle x_1,x_2\rangle)=\langle y_1,y_2\rangle$,
then

$(x_1\in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\Leftrightarrow x_2\in L_{\mathrm{DIFF}_m(\Sigma_k^p)})$ if and only if $(y_1\in L_{\mathrm{P}^{\Sigma_i^p[1]}}\Leftrightarrow y_2\in L_{\mathrm{DIFF}_m(\Sigma_k^p)})$.

We can use h to recognize some of $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ by a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm. In particular, we say that a string x is *easy for length n* if there exists a string x_1 such that $|x_1|\leq n$ and $(x_1\in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\Leftrightarrow y_1\notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$ where $h(\langle x_1,x\rangle)=\langle y_1,y_2\rangle$.

Let p be a fixed polynomial, which will be exactly specified later in the proof. We have the following algorithm to test whether $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ in the case that (our input) x is an easy string for $p(|x|)$. On input x, guess x_1 with $|x_1| \leq p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$ and $y_2 \in L_{\mathrm{DIFF}_m(\Sigma_k^p)}$. This algorithm is not necessarily a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm, but it does inspire the following $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm to test whether $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ in the case that x is an easy string for $p(|x|)$. Let $L_1, L_2, \cdots, L_m$ be languages in Σ_k^p such that $L_{\mathrm{DIFF}_m(\Sigma_k^p)} = L_1 - (L_2 - (L_3 - \cdots (L_{m-1} - L_m)\cdots))$. Then $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ if and only if $x \in L_1' - (L_2' - (L_3' - \cdots (L_{m-1}' - L_m')\cdots))$, where L_r' is computed as follows: On input x, guess x_1 with $|x_1| \leq p(|x|)$, let $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$, and accept if and only if (a) $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \notin L_{\mathrm{P}^{\Sigma_i^p[1]}})$, and (b) $(\forall z <_{lex} x_1)[z \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow w_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}}]$, where $h(\langle z, x\rangle) = \langle w_1, w_2\rangle$, and (c) $y_2 \in L_r$.

Since $i + 2 < k$, $L_r' \in \Sigma_k^p$, and thus our algorithm is in $\mathrm{DIFF}_m(\Sigma_k^p)$. Note that condition (b) has no analog in the proof of Theorem 2.4. We need this extra condition here as otherwise the different L_r' might latch onto *different* strings x_1 and this would cause unpredictable behavior (as different x_1s would create different y_2s).

We say that x is *hard for length* n if $|x| \leq n$ and x is not easy for length n, i.e., if $|x| \leq n$ and for all x_1 with $|x_1| \leq n$, $(x_1 \in L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}} \Leftrightarrow y_1 \in L_{\mathrm{P}^{\Sigma_i^p[1]}})$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$.

If x is a hard string for length n, then x induces a many-one reduction from $\left(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\right)^{\leq n}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$, namely, $f(x_1) = y_1$, where $h(\langle x_1, x\rangle) = \langle y_1, y_2\rangle$. Note that f is computable in time polynomial in $\max(n, |x_1|)$.

We can use hard strings to obtain a $\mathrm{DIFF}_m(\Sigma_{k-1}^p)$ algorithm for $L_{\mathrm{DIFF}_m(\Sigma_k^p)}$, and thus (since $\mathrm{DIFF}_m(\Sigma_{k-1}^p) \subseteq \mathrm{P}^{\Sigma_{k-1}^p} \subseteq \Sigma_k^p \cap \Pi_k^p$) certainly a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm for $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$. Again, let $L_1, L_2, \cdots, L_m$ be languages in Σ_k^p such that $L_{\mathrm{DIFF}_m(\Sigma_k^p)} = L_1 - (L_2 - (L_3 - \cdots (L_{m-1} - L_m)\cdots))$. For all $1 \leq r \leq m$, let M_r be a Σ_{k-i-1}^p machine such that $L_r = L(M_r^Y)$, where $Y = L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$. Let the run-time of all M_rs be bounded by polynomial p, which without loss of generality satisfies $(\forall \widehat{m} \geq 0)[p(\widehat{m}+1) > p(\widehat{m}) > 0]$ (as promised above, we have now specified p). Then for all $1 \leq r \leq m$,

$$(L_r)^{=n} = L(M_r^{\left(Y^{\leq p(n)}\right)})^{=n},$$

where $Y = L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}$. If there exists a hard string for length $p(n)$, then this hard string induces a reduction from $\left(L_{\mathrm{P}^{\Sigma_{i+1}^p[1]}}\right)^{\leq p(n)}$ to $L_{\mathrm{P}^{\Sigma_i^p[1]}}$. Thus, with any hard string for length $p(n)$ in hand, call it w_n, $\widehat{M_r}$ with oracle $L_{\mathrm{P}^{\Sigma_i^p[1]}}$ recognizes L_r for strings of length n, where $\widehat{M_r}$ is the machine that simulates M_r but replaces each query to q by the first component of $h(\langle q, w_n\rangle)$. It follows that if there exists a hard string for length $p(n)$, then this string induces a $\mathrm{DIFF}_m(\Sigma_{k-1}^p)$ algorithm for $\left(L_{\mathrm{DIFF}_m(\Sigma_k^p)}\right)^{=n}$, and therefore certainly a $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm for $\left(\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}\right)^{=n}$. It follows that there exist m Σ_k^p sets, say, $\widehat{L_r}$

for $1 \leq r \leq m$, such that the following holds: For all x, if x (functioning as $w_{|x|}$ above) is a hard string for length $p(|x|)$, then $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ if and only if $x \in \widehat{L_1} - (\widehat{L_2} - (\widehat{L_3} - \cdots (\widehat{L_{m-1}} - \widehat{L_m}) \cdots))$.

However, now we have an outright $\mathrm{DIFF}_m(\Sigma_k^p)$ algorithm for $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$: For $1 \leq r \leq m$ define a $\mathrm{NP}^{\Sigma_{k-1}^p}$ machine N_r as follows: On input x, the NP base machine of N_r executes the following algorithm:

1. Using its Σ_{k-1}^p oracle, it deterministically determines whether the input x is an easy string for length $p(|x|)$. This can be done, as checking whether the input is an easy string for length $p(|x|)$ can be done by one query to Σ_{i+2}^p, and $i+2 \leq k-1$ by our $i < k-2$ hypothesis.

2. If the previous step determined that the input is not an easy string, then the input must be a hard string for length $p(|x|)$. So simulate the Σ_k^p algorithm for $\widehat{L_r}$ induced by this hard string (i.e., the input x itself) on input x (via our NP machine itself simulating the base level of the Σ_k^p algorithm and using the NP machine's oracle to simulate the oracle queries made by the base level NP machine of the Σ_k^p algorithm being simulated).

3. If the first step determined that the input x is easy for length $p(|x|)$, then our NP machine simulates (using itself and its oracle) the Σ_k^p algorithm for L'_r on input x.

It follows that for all x, $x \in \overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ if and only if $x \in L(N_1) - (L(N_2) - (L(N_3) - \cdots (L(N_{m-1}) - L(N_m)) \cdots))$. Since $\overline{L_{\mathrm{DIFF}_m(\Sigma_k^p)}}$ is complete for $\mathrm{coDIFF}_m(\Sigma_k^p)$, it follows that $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$. ∎

Finally, remark that we have analogs of Theorem 2.5 and Corollary 2.6. The proof is analogous to that of Theorem 2.5; one just uses $\mathrm{P}_{1,m\text{-tt}}^{(\mathrm{NP},\Sigma_k^p)}$ in the way $\mathrm{P}^{(\mathrm{NP},\Sigma_k^p)}$ was used in that proof, and again invokes the relation between the difference and symmetric difference hierarchies (namely that $\mathrm{DIFF}_j(\Sigma_k^p)$ is exactly the class of sets L that for some $L_1, \cdots, L_j \in \Sigma_k^p$ satisfy $L = L_1 \triangle \cdots \triangle L_j$; this well-known equality is due to [KSW87, Section 3] in light of the standard equalities regarding boolean hierarchies (see [CGH+88, Section 2.1]); though both [KSW87,CGH+88] focus mostly on the $k = 1$ case, it is standard [Wec85, BBJ+89] that the equalities in fact hold for any class closed under union and intersection and containing $\emptyset$ and Σ^*).

Theorem 3.7 *Let $m \geq 0$ and $k > 2$. If $\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{DIFF}_{m+1}(\Sigma_k^p) \cap \mathrm{coDIFF}_{m+1}(\Sigma_k^p)$ then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p)$.*

Corollary 3.8 *For each $k > 2$ and $m \geq 0$, it holds that:*

$$\mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{DIFF}_{m+1}(\Sigma_k^p) \cap \mathrm{coDIFF}_{m+1}(\Sigma_k^p) \Leftrightarrow \mathrm{P}_{m\text{-tt}}^{\Sigma_k^p} = \mathrm{DIFF}_{m+1}(\Sigma_k^p).$$

Acknowledgments

The first two authors thank Gerd Wechsung's research group for its very kind hospitality during the visit when this research was performed. The authors are grateful to two anonymous referees, Lance Fortnow, and Jörg Rothe for helpful comments and suggestions.

References

[All86] E. Allender. The complexity of sparse sets in P. In *Proceedings of the 1st Structure in Complexity Theory Conference*, pages 1–11. Springer-Verlag *Lecture Notes in Computer Science #223*, June 1986.

[All91] E. Allender. Limitations of the upward separation technique. *Mathematical Systems Theory*, 24(1):53–67, 1991.

[AW90] E. Allender and C. Wilson. Downward translations of equality. *Theoretical Computer Science*, 75(3):335–346, 1990.

[BBJ+89] A. Bertoni, D. Bruschi, D. Joseph, M. Sitharam, and P. Young. Generalized boolean hierarchies and boolean hierarchies over RP. In *Proceedings of the 7th Conference on Fundamentals of Computation Theory*, pages 35–46. Springer-Verlag *Lecture Notes in Computer Science #380*, August 1989.

[BCO93] R. Beigel, R. Chang, and M. Ogiwara. A relationship between difference hierarchies and relativized polynomial hierarchies. *Mathematical Systems Theory*, 26:293–310, 1993.

[BF96] H. Buhrman and L. Fortnow. Two queries. Technical Report 96-20, University of Chicago, Department of Computer Science, Chicago, IL, September 1996.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BJY90] D. Bruschi, D. Joseph, and P. Young. Strong separations for the boolean hierarchy over RP. *International Journal of Foundations of Computer Science*, 1(3):201–218, 1990.

[Boo74] R. Book. Tally languages and complexity classes. *Information and Control*, 26:186–193, 1974.

[CGH+88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.

[CGH+89] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, 1989.

[Cha91] R. Chang. *On the Structure of NP Computations under Boolean Operators*. PhD thesis, Cornell University, Ithaca, NY, 1991.

[CK96] R. Chang and J. Kadin. The boolean hierarchy and the polynomial hierarchy: A closer connection. *SIAM Journal on Computing*, 25(2):340–354, 1996.

[HHH97] E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. Translating equality downwards. Technical Report TR-657, University of Rochester, Department of Computer Science, Rochester, NY, April 1997.

[HIS85] J. Hartmanis, N. Immerman, and V. Sewelson. Sparse sets in NP−P: EXPTIME versus NEXPTIME. *Information and Control*, 65(2/3):159–181, 1985.

[HJ95] L. Hemaspaandra and S. Jha. Defying upward and downward separation. *Information and Computation*, 121(1):1–13, 1995.

[HR] L. Hemaspaandra and J. Rothe. Unambiguous computation: Boolean hierarchies and sparse Turing-complete sets. *SIAM Journal on Computing*. To appear.

[HRW] L. Hemaspaandra, J. Rothe, and G. Wechsung. Easy sets and hard certificate schemes. *Acta Informatica*. To appear.

[HRZ95] L. Hemaspaandra, A. Ramachandran, and M. Zimand. Worlds to die for. *SIGACT News*, 26(4):5–15, 1995.

[HY84] J. Hartmanis and Y. Yesha. Computation times of NP sets of different densities. *Theoretical Computer Science*, 34:17–32, 1984.

[Kad88] J. Kadin. The polynomial time hierarchy collapses if the boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, 1988. Erratum appears in the same journal, 20(2):404.

[Ko82] K. Ko. Some observations on the probabilistic algorithms and NP-hard problems. *Information Processing Letters*, 14(1):39–43, 1982.

[Ko89] K. Ko. Relativized polynomial time hierarchies having exactly k levels. *SIAM Journal on Computing*, 18(2):392–408, 1989.

[Kre88] M. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36:490–509, 1988.

[KSW87] J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *RAIRO Theoretical Informatics and Applications*, 21:419–435, 1987.

[RRW94] R. Rao, J. Rothe, and O. Watanabe. Upward separation for FewP and related classes. *Information Processing Letters*, 52(4):175–180, 1994.

[Wec85] G. Wechsung. On the boolean closure of NP. In *Proceedings of the 5th Conference on Fundamentals of Computation Theory*, pages 485–493. Springer-Verlag *Lecture Notes in Computer Science #199* , 1985. (An unpublished precursor of this paper was coauthored by K. Wagner).