1 # Commutative Queries

2 *Richard Beigel*[*] — University of Illinois at Chicago

Richard Chang[†] — UMBC

3 Version: June 21, 2000 13:28

6 **Abstract**

7 We consider polynomial-time Turing machines that have access to two oracles and investigate when the order of oracle queries is significant. The oracles used here are complete languages for the Polynomial Hierarchy (PH). We prove that, for solving decision problems, the order of oracle queries does not matter. This improves upon the previous result of Hemaspaandra, Hemaspaandra and Hempel, who showed that the order of the queries does not matter if the base machine asks only one query to each oracle. On the other hand, we prove that, for computing functions, the order of oracle queries does matter, unless PH collapses.

---

4 [*]Address: Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, 851 South Morgan Street (M/C 154), Chicago, IL 60607-7053. Supported in part by the National Science Foundation under grants CCR-9415410 & CCR-9700417 and by NASA under grant NAG 52895. Research performed while this author was at the University of Maryland Human–Computer Interaction Lab while on sabbatical from Yale University. Email: `beigel@eecs.uic.edu`

5 [†]Address: Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, USA. Supported in part by the National Science Foundation under grants CCR-9309137 & CCR-9610457 and by the University of Maryland Institute for Advanced Computer Studies. Email: `chang@umbc.edu`

1 Introduction

8 In this paper we examine the complexity of languages and functions computed by polynomial-time Turing machines which have access to two oracles. We ask whether the order of the queries is significant. That is, given oracles E and H, does it matter if we ask the queries to E first or to H first? For this question to be nontrivial, the complexity of E and H must be significantly different. Otherwise, the queries to E and H would be trivially interchangeable. We choose our oracles E and H to be complete languages for different levels of the Polynomial Hierarchy (PH) — for example, E might be complete for NP and H complete for Σ_2^{P}. (We use E for the "easier" oracle and H for the "harder" one.) Our results show that when a polynomial-time machine is allowed parallel queries to E and H, then the order of the queries does not matter when the machine is recognizing a language — i.e., the queries to E and H are commutative. In particular, we show that for all polynomial bounded $r(n)$ and $s(n)$,

9 $$\mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} = \mathrm{P}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}},$$

10 where $\mathrm{P}^{A_{a(n)\text{-tt}};B_{b(n)\text{-tt}}}$ denotes the class of languages recognized by polynomial-time Turing machines that ask $a(n)$ parallel queries to A followed by $b(n)$ parallel queries to B. This result improves upon the previous results of Hemaspaandra, Hemaspaandra and Hempel [HHH98] who showed that the order of the queries does not matter if the base machine asks just one query to each oracle. The techniques they use do not generalize to computations that involve more than two queries in total. Furthermore, our new results extend to machines that ask several rounds of queries to E and H. For example, we can show that

11 $$\mathrm{P}^{H_{a\text{-tt}};E_{b\text{-tt}};H_{c\text{-tt}};E_{d\text{-tt}}} = \mathrm{P}^{H_{a\text{-tt}};H_{c\text{-tt}};E_{b\text{-tt}};E_{d\text{-tt}}}.$$

12 In the proofs of these results, it is simple to show that the queries to the easy oracle E can be delayed — i.e., we can always ask the hard questions first. The difficulty is in showing that the queries to the hard oracle H can also be delayed.

13 In this paper, we also consider *functions* computable by polynomial-time Turing machines with access to two oracles. In contrast to the language classes discussed above, we show that for function classes the queries to E and H do not commute. First, we show that every function computed by a machine that queries E first has an equivalent machine that queries H first. That is, for polynomial bounded $r(n)$ and $s(n)$,

14 $$\mathrm{PF}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} \subseteq \mathrm{PF}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}.$$

15 However, asking queries to H first is strictly more powerful unless PH collapses, because for polynomial-time computable $r(n) \leq \epsilon \log n$ (for some $\epsilon < 1$) and for $s(n) \in O(\log n)$

16 $$\mathrm{PF}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} \subseteq \mathrm{PF}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} \Longrightarrow \mathrm{PH} \subseteq \mathrm{NP}^{E}.$$

17 The proof of this result extends in a straightforward manner to the case with more than two rounds of parallel queries. For example, we can show that

18 $$\mathrm{PF}^{H_{r(n)\text{-tt}};H_{s(n)\text{-tt}};E_{p(n)\text{-tt}}} \subseteq \mathrm{PF}^{H_{r(n)\text{-tt}};E_{p(n)\text{-tt}};H_{s(n)\text{-tt}}} \Longrightarrow \mathrm{PH} \subseteq \mathrm{NP}^{E}$$

19 where $r(n)$, $s(n)$ and $p(n)$ are in $O(\log n)$ such that $r(n) + s(n) \leq \epsilon \log n$ for some $\epsilon < 1$.

20 Several other studies have examined the effect of the order of access to multiple oracles. Hemaspaandra, Hempel and Wechsung [HHW95] were the first to consider this problem. They determined when the order of queries to complete languages for the Boolean Hierarchy can be reversed. Related results were obtained by Agrawal, Beigel and Thierauf [ABT96]. Also, McNicholl [McN] has investigated the order of oracle queries in a recursion theoretic setting.

2 Preliminaries

21 **Definition 1** Let $\mathrm{P}^{A_{a(n)\text{-tt}}}$ be the class of languages recognized by deterministic polynomial-time Turing machines which ask at most $a(n)$ parallel (a.k.a. truth-table) queries to the oracle A on inputs of length n. The polynomial-time machine computes a sequence of $a(n)$ query strings and submits them to the oracle simultaneously. The oracle answers with an $a(n)$-bit string which specifies the membership of each query string in A. The polynomial-time machine makes no additional use of the oracle. We use $\mathrm{PF}^{A_{a(n)\text{-tt}}}$ to denote the analogous class of functions.

22 Another oracle access mechanism considered in the bounded query literature allows a Turing machine to make serial queries to the oracle. That is, subsequent queries to the oracle can depend on the answers to the previous queries. In this paper, we do not consider the case of serial queries explicitly. However, serial queries to an oracle can be considered handled as several rounds of parallel queries where the machine makes only one query per round.

23 For bounded queries to a single oracle, we use the standard notation defined above. For multiple oracle queries, new notation is needed.

24 **Definition 2**

25
- Let $\mathrm{P}^{A_{a(n)\text{-tt}};B_{b(n)\text{-tt}}}$ denote the class of languages recognized by polynomial-time Turing machines that ask $a(n)$ parallel queries to the oracle A followed by $b(n)$ parallel queries to the oracle B on inputs of length n.

26
- Let $\mathrm{P}^{A_{a(n)\text{-tt}}\|B_{b(n)\text{-tt}}}$ denote the class of languages recognized by polynomial-time Turing machines that ask $a(n)$ parallel queries to A simultaneous with $b(n)$ parallel queries to B.

27
- Let $\mathrm{PF}^{A_{a(n)\text{-tt}};B_{b(n)\text{-tt}}}$ denote the class of functions recognized by polynomial-time Turing machines that ask $a(n)$ parallel queries to A followed by $b(n)$ parallel queries to B.

28
- Let $\mathrm{PF}^{A_{a(n)\text{-tt}}\|B_{b(n)\text{-tt}}}$ denote the class of functions recognized by polynomial-time Turing machines that ask $a(n)$ parallel queries to A simultaneous with $b(n)$ parallel queries to B.

29 Note that $\mathrm{P}^{A_{a(n)\text{-tt}}\|B_{b(n)\text{-tt}}}$ is trivially contained in both $\mathrm{P}^{A_{a(n)\text{-tt}};B_{b(n)\text{-tt}}}$ and $\mathrm{P}^{B_{b(n)\text{-tt}};A_{a(n)\text{-tt}}}$. In the case that $\mathrm{P}^{A_{a(n)\text{-tt}};B_{b(n)\text{-tt}}} = \mathrm{P}^{B_{b(n)\text{-tt}};A_{a(n)\text{-tt}}}$, we say that $a(n)$ queries to A and $b(n)$ queries to B are *commutative* for language classes. Commutative queries for function classes is defined analogously.

30 Classes of languages and functions defined by machines that ask more than two rounds of parallel queries are defined similarly. For example, $\mathrm{P}^{A_{a\text{-tt}};B_{b\text{-tt}};C_{c\text{-tt}};D_{d\text{-tt}}}$ is the class of languages accepted by polynomial-time Turing machines that ask a queries to A, b queries to B, c queries to C and d queries to D in that order.

31 **Definition 3** For $k \geq 1$, we define a Σ_k^{P} machine to be an NP machine with an oracle that is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\Sigma_{k-1}^{\mathrm{P}}$. By convention, $\Sigma_0^{\mathrm{P}} = \mathrm{P}$. The Σ_k^{P} level of the Polynomial Hierarchy (PH) contains exactly the languages recognized by Σ_k^{P} machines.

32 **Definition 4** We use $\leq_{\mathrm{m}}^{\mathrm{P}}$, $\leq_{\mathrm{m}}^{\mathrm{NP}}$, $\leq_{\mathrm{conj}}^{\mathrm{P}}$ and $\leq_{r\text{-tt}}^{\mathrm{P}}$ to denote, respectively, polynomial-time many-one, nondeterministic polynomial-time many-one, polynomial-time conjunctive and polynomial-time truth-table reductions. Let A and B be any two languages over some alphabet Σ. Then, $A \leq_{\mathrm{m}}^{\mathrm{P}} B$ if there exists a deterministic polynomial-time computable function f such that all $x \in \Sigma^*$,

33 $x \in A \iff f(x) \in B$.

34 Also, $A \leq_{\mathrm{m}}^{\mathrm{NP}} B$ if there exists an NP machine N such that for all $x \in \Sigma^*$, $x \in A$ if and only if some computation path of $N(x)$ outputs a string $y \in B$. We say that $A \leq_{\mathrm{conj}}^{\mathrm{P}} B$ if there exists a polynomial-time computable function f such that for all $x \in \Sigma^*$, $f(x) = \langle y_1, \ldots, y_{r(x)} \rangle$ and

35 $$x \in A \iff (\forall i, 1 \leq i \leq r(x))[y_i \in B].$$

36 Finally, $A \leq_{r\text{-tt}}^{\mathrm{P}} B$ if $A \in \mathrm{P}^{B_{r\text{-tt}}}$. Furthermore, for a language B and a reduction R, we use $R(B)$ to denote the set of languages that are R-reducible to B. For example, $\leq_{\mathrm{m}}^{\mathrm{P}}(B) = \{A : A \leq_{\mathrm{m}}^{\mathrm{P}} B\}$.

37 **Notation 5** Let A and B be any two languages:

38 - $A(x)$ is the characteristic function of the set A at x
39 - $\chi_t^A(x_1, \ldots, x_t) = A(x_1) \cdots A(x_t)$, where juxtaposition means concatenation
40 - $\#_t^A(x_1, \ldots, x_t) = \|\{i : (1 \leq i \leq t) \wedge (x_i \in A)\}\|$
41 - $A^{\leq m} = \{x \in A : |x| \leq m\}$.
42 - $A^{=m} = \{x \in A : |x| = m\}$.
43 - $A^{[m]} = \{S : S \subseteq A \text{ and } |S| \leq m\}$
44 - $A \times A = \{(x, y) : (x \in A) \wedge (y \in A)\}$
45 - $A \triangle B = (A \times \overline{B}) \cup (\overline{A} \times B) = \{(x, y) : ((x \in A) \wedge (y \notin B)) \vee ((x \notin A) \wedge (y \in B))\}$

46 We also use $\#_\omega^A$ and χ_ω^A to denote the analogs of $\#_t^A$ and χ_t^A that take vectors of any dimension as input. Furthermore, let $\{0,1\}^{m \times t}$ denote the set of vectors $\vec{x} = \langle x_1, \ldots, x_t \rangle$ with t components where each x_i has length m.

47 **Notation 6** Let A be any language. For a fixed dimension t, the language ODD_t^A consists of those vectors $\vec{x} = \langle x_1, \ldots, x_t \rangle$ such that $\#_t^A(\vec{x})$ is odd. The language $\mathrm{ODD}_\omega^A = \bigcup_{t \geq 1} \mathrm{ODD}_t^A$ is defined for vectors of any dimension. The languages EVEN_t^A and EVEN_ω^A are defined analogously. As usual, $\mathrm{ODD}_t^A(\vec{x})$, $\mathrm{ODD}_\omega^A(\vec{x})$, $\mathrm{EVEN}_t^A(\vec{x})$ and $\mathrm{EVEN}_\omega^A(\vec{x})$ in functional form denote the characteristic functions of the respective languages. Finally, we use $\oplus$ to denote addition modulo 2.

48 **Definition 7** A function g from X to Y is m-enumerable if there is a polynomial-time computable function f from X to $Y^{[m]}$ such that $(\forall x)[g(x) \in f(x)]$.

49 Note that if g can be computed by a polynomial-time machine that makes t queries to A then g is 2^t-enumerable. The function χ_t^A can be computed using t parallel queries to A. In many cases, it has been shown that t queries to A are necessary. In particular, if A is disjunctively self-reducible and χ_t^A is $(2^t - 1)$-enumerable, then $A \in \mathrm{P}$ using a tree pruning procedure [BKS95].

50 It will be helpful if the reader is familiar with mind-change proofs, which have been used to show the relationships between serial and parallel queries [Bei91], and with hard/easy arguments, which have been used to show that a collapse of the Boolean Hierarchy implies a collapse of the Polynomial Hierarchy [Kad88, BCO93, CK96, HHH99, BF99]. We use the mind-change technique to show that $\mathrm{ODD}_r^H \triangle \mathrm{ODD}_s^E$ is $\leq_{1\text{-tt}}^{\mathrm{P}}$-complete for $\mathrm{P}^{H_{r\text{-tt}};E_{s\text{-tt}}}$. The Boolean Hierarchy comes into play because ODD_r^H is complete for the rth level of the Boolean Hierarchy over Σ_k^{P}. (Recall that H is a Σ_k^{P}-complete language.)

51 The Boolean Hierarchy over NP is a generalization of the class D^{P} defined by Papadimitriou and Yannakakis [PY82]. For constant k, the kth level of the Boolean Hierarchy can be defined simply as nested differences of NP languages [CGH⁺88, CGH⁺89]. In general, we can consider the Boolean Hierarchy over Σ_k^{P} for $k > 1$, defined as follows:

52 **Definition 8** For constant t, a language L is in BH_t^k if there exists a Σ_k^{P} language L'

53 $$x \in L \iff \max(\{i : 1 \le i \le t \text{ and } (x,i) \in L'\} \cup \{0\}) \text{ is odd.}$$

54 Also, $\overline{L} \in \mathrm{BH}_t^k$ implies that $L \in \mathrm{coBH}_t^k$.

55 The connection between the Boolean Hierarchy and bounded query computations has been used to prove many results in bounded query complexity. For example, to prove that $\mathrm{P}^{\mathrm{SAT}(t-1)\text{-tt}} = \mathrm{P}^{\mathrm{SAT}_{t\text{-tt}}} \implies$ PH collapses, the standard proof is to show that $\mathrm{P}^{\mathrm{SAT}(t-1)\text{-tt}} = \mathrm{P}^{\mathrm{SAT}_{t\text{-tt}}} \implies \mathrm{BH}_t^1 = \mathrm{coBH}_t^1$ and then cite the fact that a collapse of the Boolean Hierarchy implies a collapse of PH. This proof also extends to $\mathrm{P}^{\mathrm{SAT}_{t(n)\text{-tt}}}$ where $t(n) \in o(n)$ is an increasing function [Wag88]. However, dealing with non-constant levels of the Boolean Hierarchy introduces many subtleties and notational complications (q.v. [Wag88, Wag90, Cha97]). In this paper, we do work with machines which use a non-constant number of queries, but we can avoid some of the notational difficulties by working directly with the complete languages for the Boolean Hierarchy rather than the hierarchy itself. Recall that if H is $\le_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P}, then the language ODD_t^H is $\le_{\mathrm{m}}^{\mathrm{P}}$-complete for BH_t^k. However, when we use the hard/easy arguments, it is more convenient to use the "Boolean Languages" defined below rather than ODD_t^H.

56 **Definition 9** For a language A, we define BL_t^A recursively:

$$\begin{aligned}
\mathrm{BL}_1^A &= A \\
\mathrm{BL}_{2t}^A &= \{\langle x_1,\ldots,x_{2t}\rangle : \langle x_1,\ldots,x_{2t-1}\rangle \in \mathrm{BL}_{2t-1}^A \text{ and } x_{2t} \notin A\} \\
\mathrm{BL}_{2t+1}^A &= \{\langle x_1,\ldots,x_{2t+1}\rangle : \langle x_1,\ldots,x_{2t}\rangle \in \mathrm{BL}_{2t}^A \text{ or } x_{2t+1} \in A\} \\
\mathrm{coBL}_t^A &= \{\langle x_1,\ldots,x_t\rangle : \langle x_1,\ldots,x_t\rangle \notin \mathrm{BL}_t^A\} \\
\mathrm{BL}_\omega^A &= \bigcup_{t\ge 1}^{\infty} \mathrm{BL}_t^A \\
\mathrm{coBL}_\omega^A &= \bigcup_{t\ge 1}^{\infty} \mathrm{coBL}_t^A
\end{aligned}$$

57 We will work with either ODD_t^A or BL_t^A, whichever one is more convenient for the situation at hand. We ask the reader to confirm the following relationships between ODD_t^A and BL_t^A. Consider a sequence $\langle x_1,\ldots,x_t\rangle$. Let z be the largest index such that $x_z \in A$. Then, the sequence $\langle x_1,\ldots,x_t\rangle \in \mathrm{BL}_t^A$ if and only if z is odd. Also, for a *nested sequence* $\langle x_1,\ldots,x_t\rangle$, where $x_{i+1} \in A \implies x_i \in A$, we have that $\langle x_1,\ldots,x_t\rangle \in \mathrm{ODD}_t^A$ if and only if $\langle x_1,\ldots,x_t\rangle \in \mathrm{BL}_t^A$. Thus, if $\le_{\mathrm{m}}^{\mathrm{P}}(A)$ is closed under disjunctive reductions, ODD_t^A and BL_t^A are $\le_{\mathrm{m}}^{\mathrm{P}}$-equivalent. Since the languages H and E are $\le_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P} and Σ_j^{P}, which are closed under disjunctive reductions, $\mathrm{ODD}_t^H \equiv_{\mathrm{m}}^{\mathrm{P}} \mathrm{BL}_t^H$ and $\mathrm{ODD}_t^E \equiv_{\mathrm{m}}^{\mathrm{P}} \mathrm{BL}_t^E$.

58 If A is a complete language for some level of PH, then for constant t, $\mathrm{P}^{A(t-1)\text{-tt}} = \mathrm{P}^{A_{t\text{-tt}}}$ implies that $\mathrm{BL}_t^A \le_{\mathrm{m}}^{\mathrm{P}} \mathrm{coBL}_t^A$ which in turn implies that PH collapses using the hard/easy argument [Kad88]. To generalize this beyond constants, say to the $\log n$ level, we might define $\mathrm{BL}_{\log n}^A$ and $\mathrm{coBL}_{\log n}^A$ as follows:

59 $$\mathrm{BL}_{\log n}^A = \{\vec{x} : \vec{x} = \langle x_1,\ldots,x_t\rangle \text{ and } t = \log|\vec{x}|\} \cap \mathrm{BL}_\omega^A$$

60 $$\mathrm{coBL}_{\log n}^A = \{\vec{x} : \vec{x} = \langle x_1,\ldots,x_t\rangle \text{ and } t = \log|\vec{x}|\} \cap \mathrm{coBL}_\omega^A.$$

61 Now, given $\vec{x} = \langle x_1, \dots, x_t \rangle$ where $t = \log |\vec{x}|$, let α be any string in $\overline{A}$ with length $|\vec{x}|$. Then, the following mapping

62 $$\langle x_1, \dots, x_t \rangle \longmapsto \langle \alpha, x_1, \dots, x_t \rangle$$

63 is a $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduction from $\mathrm{BL}_{\log n}^{A}$ to $\mathrm{coBL}_{\log n}^{A}$, because $|\langle \alpha, x_1, \dots, x_t \rangle| = 2|\vec{x}|$ and $\langle \alpha, x_1, \dots, x_t \rangle$ has $t + 1 = \log(2|\vec{x}|)$ components. If we wanted to show that $\mathrm{P}^{A(\log n\,-\,1)\text{-tt}} = \mathrm{P}^{A_{\log n\text{-tt}}} \Longrightarrow \mathrm{PH}$ collapses, we *cannot* use the following chain of reasoning:

64 $$\mathrm{P}^{A(\log n\,-\,1)\text{-tt}} = \mathrm{P}^{\mathrm{SAT}_{\log n\text{-tt}}} \Longrightarrow \mathrm{BL}_{\log n}^{A} \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{coBL}_{\log n}^{A} \Longrightarrow \mathrm{PH} \text{ collapses}$$

65 which is what you might expect to be the generalization of the constant case. The problem here is that the reduction can output a vector with more components than its input. Thus, for hard/easy arguments, we should restrict ourselves to dimension-preserving reductions — functions whose input and output are vectors with the same number of components.

66 Then we can prove that $\mathrm{P}^{A(\log n\,-\,1)\text{-tt}} = \mathrm{P}^{A_{\log n\text{-tt}}}$ implies PH collapses as follows. First we show that $\mathrm{P}^{A(\log n\,-\,1)\text{-tt}} = \mathrm{P}^{A_{\log n\text{-tt}}}$ implies the existence of a dimension-preserving polynomial-time function f such that for all $\vec{x} = \langle x_1, \dots, x_t \rangle$ where $t = \log |\vec{x}|$, $f(\vec{x}) = \vec{y} = \langle y_1, \dots, y_t \rangle$ and

67 $$\vec{x} \in \mathrm{BL}_{\omega}^{A} \iff \vec{y} \in \mathrm{coBL}_{\omega}^{A}.$$

68 The existence of such a "reduction" collapses PH using a straightforward generalization of the hard/easy argument used for the constant case. Note that the dimension of $\vec{y}$ and its length are not directly related since the dimension must be t and the length of each y_i might vary over a wide range. Thus, $\vec{y}$ might not be an element of $\mathrm{coBL}_{\log n}^{A}$. It is for this reason that we will, for the rest of the paper, use the notation like BL_{ω}^{A} and $\mathrm{coBL}_{\omega}^{A}$ rather than $\mathrm{BL}_{\log n}^{A}$ or $\mathrm{coBL}_{\log n}^{A}$.

3 Language classes

69 In this section we consider classes of languages recognized by polynomial-time Turing machines which have access to a Σ_k^{P} oracle and a Σ_j^{P} oracle. The results in this section show that for language classes, the order of the queries does not matter — in fact, the queries can be made in parallel. In Theorem 10, we show that when the easier questions are asked first, the queries can be made in parallel. This relationship even holds for function classes. This is the simple direction of our results; the difficult direction handles the case where the harder questions are asked first.

70 **Theorem 10** For $k > j$, let H and E be $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P} and Σ_j^{P} respectively. Then, for all polynomial bounded $r(n)$ and $s(n)$,

71 $$\mathrm{PF}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} \subseteq \mathrm{PF}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}} \subseteq \mathrm{PF}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}.$$

72 **Proof:** The second containment is obvious. To prove the first containment, we modify the techniques used by Hemaspaandra *et al.* [HHH98]. Let M be a polynomial-time bounded Turing machine that asks $s(n)$ parallel queries to the oracle E followed by $r(n)$ parallel queries to the oracle H. Let $e_1, \dots, e_{s(n)}$ be the queries that M asks the oracle E on a particular input x. Note that the queries $e_1, \dots, e_{s(n)}$ can be generated in polynomial time. Since $k > j$, a Σ_k^{P} machine can generate the set of queries $e_1, \dots, e_{s(n)}$, determine the answers to these queries and then generate the second set of queries $h_1, \dots, h_{r(n)}$ that $M(x)$ would ask to the oracle H. Thus, M' does not have to query E before asking H about the answers to $h_1, \dots, h_{r(n)}$. The machine M' can simply ask the oracle H the following question h_i':

73 "Let h_i be the ith query that $M(x)$ asks H. Is $h_i \in H$?"

74 The oracle H can answer such queries because H is complete for Σ_k^{P}. Clearly, $h'_i \in H$ if and only if $h_i \in H$. In parallel with the queries to H, $M'(x)$ also asks the oracle E for answers to $e_1, \ldots, e_{s(n)}$, the same questions that $M(x)$ asked originally. Thus, $M'(x)$ has answers to all of the oracle queries that $M(x)$ asked and $M'(x)$ can complete the simulation of $M(x)$ step by step. $\square$

75 Note that we do not really need H to be complete for Σ_k^{P}. The conditions that $E \leq_{\mathrm{m}}^{\mathrm{P}} H$, $\overline{E} \leq_{\mathrm{m}}^{\mathrm{P}} H$ and $\leq_{\mathrm{m}}^{\mathrm{NP}}(H) = \leq_{\mathrm{conj}}^{\mathrm{P}}(H) = \leq_{\mathrm{m}}^{\mathrm{P}}(H)$ are sufficient to prove Theorem 10. For constant $r(n)$ and $s(n)$, we only need the conditions

76 $$E \leq_{\mathrm{m}}^{\mathrm{P}} H,\ \overline{E} \leq_{\mathrm{m}}^{\mathrm{P}} H,\ H \times H \leq_{\mathrm{m}}^{\mathrm{P}} H \text{ and } \overline{H} \times \overline{H} \leq_{\mathrm{m}}^{\mathrm{P}} \overline{H}.$$

77 For example, the theorem holds when E is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_j^{P} and H is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for PSPACE. Also, by restricting Theorem 10 to characteristic functions, we obtain the following corollary for language classes.

78 **Corollary 11** For $k > j$, let H and E be $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P} and Σ_j^{P} respectively. Then, for all polynomial bounded $r(n)$ and $s(n)$,

79 $$\mathrm{P}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{r(n)\text{-tt}} \| E_{s(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}.$$

80 Theorem 10 and Corollary 11 show that we can always postpone the easy questions (the queries to E). In the next theorem, we show somewhat surprisingly that, when recognizing languages, we can also postpone the hard questions. In fact, in either case, all the questions can be asked in parallel, as we show in Theorem 13. First, we prove a technical lemma using the mind-change technique. For those familiar with this technique, the basic structure of this proof is the same as the proof which shows that every language in $\mathrm{P}^{H_{r\text{-tt}}}$ can be $\leq_{1\text{-tt}}^{\mathrm{P}}$-reduced to ODD_r^H [Bei91], except in this case the polynomial-time machine is also allowed to make parallel queries to E.

81 We illustrate the mind-change technique with a simple example that every $\mathrm{P}^{A_{3\text{-tt}}}$ language $\leq_{1\text{-tt}}^{\mathrm{P}}$-reduces to ODD_3^A where A is an NP-complete language. Figure 1 is a truth table for the accepting and rejecting behavior of a polynomial-time machine M that asks 3 parallel queries to A. If a 1 appears in column x_i in a row of the truth table, then we say that x_i is a *positive query* in that row. In Figure 1, the positive queries in Row 6 are x_1 and x_2. For a mind-change proof, we will only consider the rows of the truth-table that are *consistent* with A in the sense that the positive queries in that row are strings in A. In Figure 1, the consistent rows are Rows 0, 1, 4 and 5. Two consistent rows form a *mind change* if one row accepts, the other rejects and the positive queries of one row is a subset of the positive queries in the other row. In our example, Rows 1 and 5 form a mind change, but Rows 1 and 4 do not. Next we consider sequences of rows where each pair of successive rows forms a mind change. In particular, we are interested in such sequences that make the most number of mind changes. The first row of such a sequence must have the same accept/reject behavior as Row 0. The last row must have the same accept/reject behavior as the row which has the correct answers. If this were not the case, then adding Row 0 or the row with the correct answers to the sequence would increase the number of mind changes. In Figure 1, there are two sequences that make 2 mind changes: $\langle 0, 1, 4\rangle$ and $\langle 0, 1, 5\rangle$. Let b be a bit that is 0 if and only if the machine M accepts in Row 0. Since the accept/reject behavior of the machine in Row 0 can be computed in polynomial time without using any queries to A, the bit b is polynomial time computable. Furthermore, whether the maximum number of mind changes is even or odd tells us whether the machine accepted or rejected in the row with the correct answer. In our example, the

		x_1	x_2	x_3	$M^A(w)$
	0*	0	0	0	accept
	1*	0	0	1	reject
	2	0	1	0	accept
84	3	0	1	1	accept
	4*	1	0	0	accept
	5*	1	0	1	accept
	6	1	1	0	reject
	7	1	1	1	accept

85 Figure 1: Truth table for a machine M on input w asking 3 parallel queries, x_1, x_2, and x_3, to an oracle A. In this example, $x_1 \in A$, $x_2 \notin A$, and $x_3 \in A$. The asterisked rows in the truth table are consistent with A and are possible participants in a sequence of mind changes.

86

---

maximum number of mind changes is 2 and is even. Thus, the machine must accept because it accepted in Row 0. Finally, since A is NP-complete, we can compute in polynomial time three strings y_1, y_2, y_3 such that $M(w)$ makes at least i mind changes if and only if $y_i \in A$. Then,

82 $$M(w) \text{ accepts } \iff b \oplus \mathrm{ODD}_3^A(y_1, y_2, y_3) = 1.$$

83 Therefore, $L(M)$ is $\leq^{\mathrm{P}}_{1\text{-tt}}$-reducible to ODD_3^A.

87 **Lemma 12** For $k > j$, let H and E be $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for Σ^{P}_k and Σ^{P}_j respectively. Then, for each $L \in \mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}$ there exists a polynomial-time computable function h such that for all w, $|w| = n$, $h(w) = \langle b, \vec{y}, \vec{x} \rangle$ where $b \in \{0,1\}$, $\vec{y} = \langle y_1, \ldots, y_{s(n)} \rangle$, $\vec{x} = \langle x_1, \ldots, x_{r(n)} \rangle$ and

88 $$w \in L \iff b \oplus \mathrm{ODD}^E_\omega(\vec{y}) \oplus \mathrm{ODD}^H_\omega(\vec{x}) = 1.$$

89 Furthermore, $\vec{x}$ and $\vec{y}$ are nested sequences.[1]

91 **Proof:** We will use two mind-change proofs — one for the queries to H and one for the queries to E. We start with the mind-change proof for H.

92 Fix a language $L \in \mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}$ and let M be a polynomial-time Turing machine that computes the characteristic function of L using $r(n)$ parallel queries to H followed by $s(n)$ parallel queries to E on inputs of length n. Let $Q_H(w)$ be the set of queries to H made by M on input w, and let $Z_H(w) = Q_H(w) \cap H$. Given a set $Z \subseteq Q_H(w)$, let the value of $f_H(Z, w)$ be 1 if $M(w)$ outputs 1 when its queries to H were answered according to Z. Otherwise, $f_H(Z, w) = 0$. In terms of the simple example above, each set Z is the set of positive queries for a row in the truth table and the set $Z_H(w)$ is the set of positive queries in the row with the correct answers. Note that $f_H(Z, w)$ can be computed in polynomial time using $s(n)$ parallel queries to E. Let $g_H(m, w)$ be true if and only if M can make m mind changes on input w with respect to the queries to H — i.e., $g_H(m, w)$ is true when

93 $$(\exists Z_0, \ldots, Z_m)[Z_0 = \emptyset \wedge (\forall 1 \leq i \leq m)[(Z_{i-1} \subsetneq Z_i \subseteq Z_H(w)) \wedge (f_H(Z_{i-1}, w) \neq f_H(Z_i, w))]].$$

94 Since a Σ^{P}_k machine has an oracle that can answer E queries, a Σ^{P}_k machine can determine whether $g_H(m, w)$ is true by guessing an increasing sequence $Z_0, \ldots, Z_m$, checking that $Z_i \subseteq Z_H(w)$, and

90 [1]Note that in the notation $\langle x_1, \ldots, x_{r(n)} \rangle$, n is not a formal parameter but is the length of w. More importantly, n is not the length of $\langle x_1, \ldots, x_{r(n)} \rangle$.

confirming that $f_H(Z_{i-1}, w) \neq f_H(Z_i, w)$ for $1 \leq i \leq m$. Note that $g_H(0, w)$ is trivially true and that the maximum number of mind changes is bounded by $r(n)$ since $|Z_H(w)| \leq r(n)$. Since H is Σ_k^{P}-complete, we can construct $\vec{x} = \langle x_1, \ldots, x_{r(n)} \rangle$ such that

95 $$x_i \in H \iff g_H(i, w) \text{ is true.}$$

96 If $g_H(i+1, w)$ is true, then $g_H(i, w)$ must also be true. Thus, $x_{i+1} \in H \implies x_i \in H$. Therefore, the sequence $\vec{x}$ is a nested sequence. Now consider the maximum number of mind changes made by $M(w)$,

97 $$\mu = \max\{\, m : g_H(m, w) \wedge (0 \leq m \leq r(n)) \,\}$$

98 and a sequence of sets $Z_0, \ldots, Z_\mu$, where $\emptyset = Z_0 \subsetneq Z_1 \subsetneq \cdots \subsetneq Z_\mu \subseteq Z_H(w)$, which achieves the maximum number of mind changes. It must be the case that $f(Z_\mu, w) = f(Z_H(w), w)$. Otherwise, adding $Z_H(w)$ to the end of the sequence would result in $\mu + 1$ mind changes. Furthermore, $f(Z_\mu, w) = f(Z_0, w)$ if and only if μ is even. Since $M(w) = f_H(Z_H(w), w)$ and $Z_0 = \emptyset$, it follows that $M(w) = f_H(\emptyset, w)$ if and only if μ is even. Therefore,

99 $$w \in L \iff f_H(\emptyset, w) \oplus \mathrm{ODD}_\omega^H(\vec{x}) = 1. \tag{1}$$

100 Next, we use another mind-change proof to reduce $f_H(\emptyset, w)$ to $b \oplus \mathrm{ODD}_\omega^E(\vec{y})$ where b is a polynomial-time computable bit. Let $M'(w)$ be a Turing machine which computes $f_H(\emptyset, w)$ using $s(n)$ parallel queries to E. We will also use the mind-change technique to compute $M'(w)$. Let $Q_E(w)$ be the set of queries asked by $M'(w)$. Let $Z_E(w) = Q_E(w) \cap E$. We define $f_E(Z, w)$ and $g_E(m, w)$ analogously. That is, for $Z \subseteq Q_E(w)$, $f_E(Z, w) = 1$ if $M'(w)$ outputs 1 when its queries to E are answered according to Z and $g_E(m, w)$ is true when

101 $$(\exists Z_0, \ldots, Z_m)[Z_0 = \emptyset \wedge (\forall 1 \leq i \leq m)[(Z_{i-1} \subsetneq Z_i \subseteq Z_E(w)) \wedge (f_E(Z_{i-1}, w) \neq f_E(Z_i, w))]].$$

102 In this case, $f_E(Z, w)$ can be computed in polynomial time without using any oracle queries because we only have to simulate the original machine M on input w assuming all of the queries to H are answered NO and all of the queries to E are answered according to Z.

103 Let the maximum number of mind changes made by $M'(w)$ be denoted by

104 $$\mu' = \max\{\, m : g_E(m, w) \wedge (0 \leq m \leq s(n)) \,\}.$$

105 As before, given any sequence of sets $Z_0, \ldots, Z_{\mu'}$, where $\emptyset = Z_0 \subsetneq Z_1 \subsetneq \cdots \subsetneq Z_{\mu'} \subseteq Z_E(w)$, which achieves the maximum number of mind changes, $f_E(Z_{\mu'}, w)$ must be equal to $f_E(Z_E(w), w)$ (otherwise, the maximality of μ' is violated). Then, as before, $M'(w) = f_E(\emptyset, w)$ if and only if μ' is even. Since E is Σ_j^{P}-complete, we can construct a nested sequence $\vec{y} = \langle y_1, \ldots, y_{s(n)} \rangle$ such that $y_i \in E \iff g_E(i, w)$ is true. Furthermore, let the bit $b = f_E(\emptyset, w)$ which is polynomial-time computable. Then,

106 $$f_H(\emptyset, w) = b \oplus \mathrm{ODD}_\omega^E(\vec{y}). \tag{2}$$

107 Combining (1) and (2) produces the desired result. □

108 Given the values $\langle b, \vec{y}, \vec{x} \rangle$ output by the function h of Lemma 12, a $\mathrm{P}^{H_{r(n)\text{-tt}} \| E_{s(n)\text{-tt}}}$ machine can compute $\mathrm{ODD}_\omega^H(\vec{x})$ and $\mathrm{ODD}_\omega^E(\vec{y})$ by asking the $r(n)$ queries to H and $s(n)$ queries to E in parallel directly. Thus, $\mathrm{P}^{H_{r(n)\text{-tt}}; E_{s(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{r(n)\text{-tt}} \| E_{s(n)\text{-tt}}}$. Since $\mathrm{P}^{H_{r(n)\text{-tt}} \| E_{s(n)\text{-tt}}}$ is trivially contained in $\mathrm{P}^{E_{s(n)\text{-tt}}; H_{r(n)\text{-tt}}}$ and in $\mathrm{P}^{H_{r(n)\text{-tt}}; E_{s(n)\text{-tt}}}$, the next theorem follows.

109 **Theorem 13** For $k > j$, let H and E be $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for Σ^{P}_k and Σ^{P}_j respectively. Then, for all polynomial-time computable and polynomial bounded $r(n)$ and $s(n)$,

110 $$\mathrm{P}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}.$$

111 In the proof of Theorem 13, we could have computed the value of $f_H(\emptyset, w)$ in Lemma 12 directly using $s(n)$ parallel queries to E instead of resorting to mind changes. (We will need the stronger conditions of Lemma 12 later.) Hence, we have the following extensions:

112 **Theorem 14**

113 1. Let H and E be languages such that $E \leq^{\mathrm{P}}_{\mathrm{m}} H$, $\overline{E} \leq^{\mathrm{P}}_{\mathrm{m}} H$ and $\leq^{\mathrm{NP}}_{\mathrm{m}}(H) = \leq^{\mathrm{P}}_{\mathrm{conj}}(H) = \leq^{\mathrm{P}}_{\mathrm{m}}(H)$. Then, for all polynomial-time computable and polynomial bounded $r(n)$ and $s(n)$,

114 $$\mathrm{P}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}.$$

115 2. Let H and E be languages such that $E \leq^{\mathrm{P}}_{\mathrm{m}} H$, $\overline{E} \leq^{\mathrm{P}}_{\mathrm{m}} H$, $H \times H \leq^{\mathrm{P}}_{\mathrm{m}} H$ and $\overline{H} \times \overline{H} \leq^{\mathrm{P}}_{\mathrm{m}} \overline{H}$. Then, for all constants $r \geq 0$ and $s \geq 0$,

116 $$\mathrm{P}^{E_{s\text{-tt}};H_{r\text{-tt}}} = \mathrm{P}^{H_{r\text{-tt}}\|E_{s\text{-tt}}} = \mathrm{P}^{H_{r\text{-tt}};E_{s\text{-tt}}}.$$

117 Corollary 11 showed that for language classes, asking the easy questions first is equivalent to asking the hard questions first. In fact, this observation generalizes to several rounds of parallel queries to the oracles H and E. Again, there is an equivalent machine that asks all the queries to H before the queries to E. Thus, for example, for polynomially bounded $a(n)$, $b(n)$, $c(n)$ and $d(n)$

118 $$\mathrm{P}^{H_{a(n)\text{-tt}};E_{b(n)\text{-tt}};H_{c(n)\text{-tt}};E_{d(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{a(n)\text{-tt}};H_{c(n)\text{-tt}};E_{b(n)\text{-tt}};E_{d(n)\text{-tt}}}.$$

119 We can use a result of Beigel [Bei91, Theorem 4.9] to combine consecutive rounds of parallel queries to the same oracle into a single round. Thus,

120 $$\mathrm{P}^{H_{a(n)\text{-tt}};H_{c(n)\text{-tt}};E_{b(n)\text{-tt}};E_{d(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}},$$

121 where $r(n) = (a(n) + 1)(c(n) + 1) - 1$ and $s(n) = (b(n) + 1)(d(n) + 1) - 1$. Furthermore, by Theorem 13,

122 $$\mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} = \mathrm{P}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}}.$$

123 In the next theorem, we show that $\mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{a(n)\text{-tt}};E_{b(n)\text{-tt}};H_{c(n)\text{-tt}};E_{d(n)\text{-tt}}}$. Therefore, the following five classes are all equal:

$$\begin{aligned}\mathrm{P}^{H_{a(n)\text{-tt}};E_{b(n)\text{-tt}};H_{c(n)\text{-tt}};E_{d(n)\text{-tt}}} &= \mathrm{P}^{H_{a(n)\text{-tt}};H_{c(n)\text{-tt}};E_{b(n)\text{-tt}};E_{d(n)\text{-tt}}} \\ = \mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} &= \mathrm{P}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}}.\end{aligned}$$

124 **Theorem 15** Let H and E be $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for Σ^{P}_k and Σ^{P}_j respectively, where $k > j$. Furthermore, let $r(n) = (a(n)+1)(c(n)+1)-1$ and $s(n) = (b(n)+1)(d(n)+1)-1$ where $a(n)$, $b(n)$, $c(n)$ and $d(n)$ are polynomial bounded. Then,

125 $$\mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{a(n)\text{-tt}};E_{b(n)\text{-tt}};H_{c(n)\text{-tt}};E_{d(n)\text{-tt}}}.$$

126 **Proof:** Let L be any language in $\mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}$ and let h be the function given by Lemma 12. For a fixed string w, $|w| = n$, let $h(w) = \langle b, \vec{y}, \vec{x}\rangle$. Using Lemma 12, it suffices to show that for $\vec{x} = \langle x_1, \ldots, x_{r(n)}\rangle$ and $\vec{y} = \langle y_1, \ldots, y_{s(n)}\rangle$, $\mathrm{ODD}^H_\omega(\vec{x})$ and $\mathrm{ODD}^E_\omega(\vec{y})$ can be computed by a $\mathrm{P}^{H_{a(n)\text{-tt}};E_{b(n)\text{-tt}};H_{c(n)\text{-tt}};E_{d(n)\text{-tt}}}$ machine. We show how $\mathrm{ODD}^H_\omega(\vec{x})$ can be computed in two rounds of parallel queries to H.[2] Since H is Σ^{P}_k-complete, we can construct q_i such that $q_i \in H$ if and only if $\#^H_\omega(\vec{x}) \geq i(c(n)+1)$. The first round of queries is $\langle q_1, \ldots, q_{a(n)}\rangle$. Let z be the index of the largest $q_i \in H$. If none of the q_i are in H, let $z = 0$. Then, after the first round, we know that

128 $$z(c(n)+1) \leq \#^H_\omega(\vec{x}) < (z+1)(c(n)+1).$$

129 Thus, we have restricted $\#^H_\omega(\vec{x})$ to $c(n)+1$ values. In the second round of queries to H, we construct $c(n)$ queries $p_1, \ldots, p_{c(n)}$ such that $p_i \in H$ if and only if $\#^H_\omega(\vec{x}) \geq z(c(n)+1)+i$. The answers to this second round of queries will determine the exact value of $\#^H_\omega(\vec{x})$ which is sufficient to determine $\mathrm{ODD}^H_\omega(\vec{x})$. The value of $\mathrm{ODD}^E_\omega(\vec{y})$ can be determined using an analogous procedure. Thus, $\mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{a(n)\text{-tt}};E_{b(n)\text{-tt}};H_{c(n)\text{-tt}};E_{d(n)\text{-tt}}}$. □

4 Hierarchy Theorems for Language Classes

130 In the previous section, we showed that the complexity of the language classes defined by machines with access to the oracles H and E is characterized by the number of queries and not by the order of the queries. In this section, we will show that the $\mathrm{P}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}}$ classes form a nice linear hierarchy where additional queries to E are nested inside the additional queries to H:

131 $$\mathrm{P}^{H_{r(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{r(n)\text{-tt}}\|E_{1\text{-tt}}} \subseteq \mathrm{P}^{H_{r(n)\text{-tt}}\|E_{2\text{-tt}}} \subseteq \cdots \subseteq \mathrm{P}^{H_{(r(n)\,+\,1)\text{-tt}}}.$$

132 **Theorem 16** For $k > j$, let H and E be $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for Σ^{P}_k and Σ^{P}_j respectively. Then, for all polynomial bounded $r(n)$ and $s(n)$, $\mathrm{P}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{(r(n)\,+\,1)\text{-tt}}}$.

133 **Proof:** Corollary 11 and Theorem 13 show that under this lemma's hypotheses:

134 $$\mathrm{P}^{E_{s(n)\text{-tt}};H_{r(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}}.$$

135 Let L be a language in $\mathrm{P}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}$. For a fixed input string w, let $h(w) = \langle b, \vec{y}, \vec{x}\rangle$ be as described in Lemma 12 such that

136 $$L(w) = b \oplus \mathrm{ODD}^E_\omega(\vec{y}) \oplus \mathrm{ODD}^H_\omega(\vec{x}).$$

137 Since $\vec{x}$ has $r(n)$ components, $\mathrm{ODD}^H_\omega(\vec{x})$ can be determined using $r(n)$ parallel queries to H. Furthermore, since H is Σ^{P}_k complete and $E \in \Sigma^{\mathrm{P}}_j$, $\mathrm{ODD}^E_\omega(\vec{y})$ can be determined using a single query to H. Thus, $\mathrm{P}^{H_{r(n)\text{-tt}}\|E_{s(n)\text{-tt}}} \subseteq \mathrm{P}^{H_{(r(n)\,+\,1)\text{-tt}}}$. □

138 Finally, we prove in the following theorem that even when two oracles are used, each additional query adds additional computational power unless PH collapses. The proof of the theorem uses a hard/easy argument over the exclusive-or operator [BCO93]. We give this proof separately in Lemma 18.

127 [2]Note that we are not computing the function ODD^H_ω in two rounds of parallel queries to H. We are computing $\mathrm{ODD}^H_\omega(\vec{x})$ where $\vec{x}$ is a bound variable (not a formal parameter) and has a predefined dimension $r(n)$.

139 **Theorem 17** For $k > j$, let H and E be $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P} and Σ_j^{P} respectively. Then, for all $0 < \epsilon < 1$ and for all $r(n)$ and $s(n)$ in $O(n^\epsilon)$,

140 $$\mathrm{P}^{H_{r(n)\text{-tt}} \| E_{s(n)\text{-tt}}} = \mathrm{P}^{H_{r(n)\text{-tt}} \| E_{(s(n)+1)\text{-tt}}} \implies \text{PH collapses.}$$

141 **Proof:** First, we define a language L as follows. For a fixed length n, let $r = r(n)$, $s = s(n)$, $s' = s(n) + 1$, $m = n/(r + s')$. Let the set $V = \{0,1\}^{m \times s'}$, the vectors with s' components where each component has length m, and let $U = \{0,1\}^{m \times r}$. Without loss of generality we assume that for each pair $(\vec{v}, \vec{u}) \in V \times U$, the length of $(\vec{v}, \vec{u})$ is exactly n. The strings of length n in L are pairs $(\vec{v}, \vec{u}) \in V \times U$ such that $(\vec{v}, \vec{u}) \in \mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H$.

142 Since $\vec{v}$ has $s' = s(n)+1$ components and $\vec{u}$ has $r = r(n)$ components, the language L can be recognized by a $\mathrm{P}^{H_{r\text{-tt}} \| E_{s'\text{-tt}}}$ machine. By hypothesis, $\mathrm{P}^{H_{r\text{-tt}} \| E_{s'\text{-tt}}} \subseteq \mathrm{P}^{H_{r\text{-tt}} \| E_{s\text{-tt}}}$, so $L \in \mathrm{P}^{H_{r\text{-tt}} \| E_{s\text{-tt}}}$. Now, let h be the function specified in Lemma 12 such that for each $(\vec{v}, \vec{u}) \in V \times U$, $h(\vec{v}, \vec{u}) = \langle b, \vec{y}, \vec{x} \rangle$ and

143 $$(\vec{v}, \vec{u}) \in \mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H \iff b \oplus \mathrm{ODD}_\omega^E(\vec{y}) \oplus \mathrm{ODD}_\omega^H(\vec{x}) = 1.$$

144 We construct a new polynomial-time computable function f from h as follows. On input $(\vec{v}, \vec{u})$, f first computes $h(\vec{v}, \vec{u}) = \langle b, \vec{y}, \vec{x} \rangle$. Let y_{in} and y_{out} be two fixed strings such that $y_{in} \in E$ and $y_{out} \notin E$. If $b = 0$, then f outputs $(\vec{y}\,', \vec{x})$ where $\vec{y}\,' = \langle y_{in}, \vec{y} \rangle$. Otherwise, $b = 1$ and f outputs $(\vec{y}\,', \vec{x})$ where $\vec{y}\,' = \langle \vec{y}, y_{out} \rangle$. Then,

145 $$(\vec{v}, \vec{u}) \in \mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H \iff \mathrm{EVEN}_\omega^E(\vec{y}\,') \oplus \mathrm{ODD}_\omega^H(\vec{x}) = 1.$$

146 Since $\vec{x}$ and $\vec{y}\,'$ are nested sequences, we have also established that

147 $$(\vec{v}, \vec{u}) \in \mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H \iff f(\vec{v}, \vec{u}) = (\vec{y}\,', \vec{x}) \in \mathrm{coBL}_\omega^E \triangle \mathrm{BL}_\omega^H.$$

148 Thus, f is a dimension-preserving reduction from $\mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H$ to $\mathrm{coBL}_\omega^E \triangle \mathrm{BL}_\omega^H$ in the sense that the vectors output by f have the same number of components as the input vectors. This is enough for us to collapse PH using the hard/easy argument, as we show in the following lemma. □

149 **Lemma 18** For $k > j$, let H and E be $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P} and Σ_j^{P} respectively. Let f be a polynomial-time computable dimension-preserving function. Suppose that there exists polynomial-time computable polynomial-bounded functions $\tilde{r}(m)$ and $\tilde{s}(m)$ such that for all lengths m, for all $\vec{v} = \langle v_1, \ldots, v_{\tilde{s}(m)} \rangle \in \{0,1\}^{m \times \tilde{s}(m)}$ and $\vec{u} = \langle u_1, \ldots, u_{\tilde{r}(m)} \rangle \in \{0,1\}^{m \times \tilde{r}(m)}$,

150 $$(\vec{v}, \vec{u}) \in \mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H \iff f(\vec{v}, \vec{u}) \in \mathrm{coBL}_\omega^E \triangle \mathrm{BL}_\omega^H.$$

151 Then, $\overline{H} \in \Sigma_k^{\mathrm{P}}/\mathrm{poly}$ and $\mathrm{PH} \subseteq \Sigma_{k+2}^{\mathrm{P}}$.

152 **Proof:** At the end of this proof, we will show that for each length m, we either have a Σ_k^{P} machine that recognizes $\overline{H}^{=m}$ or a Σ_j^{P} machine that recognizes $\overline{E}^{=m}$. The sizes and running times of these machines will be bounded by a single polynomial in m. Without loss of generality we assume that all strings in E and H with length less than m can be padded to length exactly m.

153 In the second case, where we have a Σ_j^{P} machine for $\overline{E}$, we use a standard oracle replacement argument to show that $\Sigma_{k+1}^{\mathrm{P}} = (\Sigma_{k+1-j}^{\mathrm{P}})^E \subseteq \Sigma_k^{\mathrm{P}}$. Since $\overline{H} \in \Sigma_{k+1}^{\mathrm{P}}$, we also get a Σ_k^{P} machine for $\overline{H}$. However, this Σ_k^{P} machine would recognize $\overline{H}$ for strings of shorter length, because the length of the oracle queries to E can be stretched by a polynomial factor. Nevertheless, we can choose $m \geq n$ to be long enough, but still bounded by a polynomial in n, so that in either case (whether

we have a Σ_k^{P} machine for $\overline{H}^{=n}$ or a Σ_j^{P} machine for $\overline{E}^{=n}$) we have a Σ_k^{P} machine that recognizes $\overline{H}^{=n}$.

154 For now, let us fix a length m. To simplify our notation, let $r = \tilde{r}(m)$ and $s = \tilde{s}(m)$. Let $V = \{0,1\}^{m \times s}$ and $U = \{0,1\}^{m \times r}$. That is, the sets V and U consist of all the vectors with s and r components respectively where each component has length m. For a pair $(\vec{v}, \vec{u}) \in V \times U$, if $n = |(\vec{v}, \vec{u})|$, then $s = \tilde{s}(m) = s(n)$ and $r = \tilde{r}(m) = r(n)$ for the functions $s(n)$ and $r(n)$ defined in Theorem 17.

155 Recall that f is a dimension-preserving function means that the outputs of f have the same dimensions as the inputs. Thus, for all $(\vec{v}, \vec{u}) \in V \times U$, $f(\vec{v}, \vec{u}) = (\vec{y}, \vec{x})$ where $\vec{y}$ and $\vec{x}$ have s and r components respectively such that

156 $$(\vec{v}, \vec{u}) \in \mathrm{BL}_\omega^E \bigtriangleup \mathrm{BL}_\omega^H \iff (\vec{y}, \vec{x}) \in \mathrm{coBL}_\omega^E \bigtriangleup \mathrm{BL}_\omega^H.$$

157 We now give the formal definition of a hard sequence, which is central to the hard/easy argument. In this definition, for a sequence $\vec{z} = \langle z_1, \ldots, z_t \rangle$, we use $\vec{z}^{\mathrm{R}}$ to denote the reversal of the sequence $\langle z_t, \ldots, z_1 \rangle$. We say that $\vec{z}$ is a *hard sequence* for length m if $\vec{z}$ is the empty sequence or if all of the following conditions hold.

158 **Hard Sequence**

159 1. $1 \leq t \leq r + s - 1$.

160 2. $\langle z_1, \ldots, z_{t-1} \rangle$ is a hard sequence for length m.

161 3. For $1 \leq i \leq t$, $|z_i| = m$.

162 4. For $1 \leq i \leq \min(r, t)$, $z_i \in \overline{H}$.

163 5. If $t > r$, then for $r + 1 \leq i \leq t$, $z_i \in \overline{E}$.

164 6. If $t \leq r$, let $\ell = r - t$. For all $\vec{v} \in V$ and for all $\langle u_1, \ldots, u_\ell \rangle \in \{0,1\}^{m \times \ell}$, let

165 $$f(\vec{v}, \langle u_1, \ldots, u_\ell, \vec{z}^{\mathrm{R}} \rangle) = (\vec{y}, \langle x_1, \ldots, x_r \rangle).$$

166 Then,$(\vec{v}, \vec{y}) \in \mathrm{BL}_\omega^E \bigtriangleup \mathrm{coBL}_\omega^E \Longrightarrow x_{\ell+1} \in \overline{H}$.

167 7. If $t > r$, let $\ell = r + s - t$. For all $\langle v_1, \ldots, v_\ell \rangle \in \{0,1\}^{m \times \ell}$, let

168 $$f(\langle v_1, \ldots, v_\ell, z_t, \ldots, z_{r+1} \rangle, \langle z_r, \ldots, z_1 \rangle) = (\langle y_1, \ldots, y_s \rangle, \vec{x}).$$

169 Then, $y_{\ell+1} \in \overline{E}$.

170 Given a hard sequence $\vec{z} = \langle z_1, \ldots, z_t \rangle$, we refer to t as the *order* of the hard sequence. Furthermore, we say that a hard sequence $\vec{z}$ is a *maximal* hard sequence if for all $w \in \{0,1\}^m$, $\langle z_1, \ldots, z_t, w \rangle$ is not a hard sequence. Since the empty sequence is a hard sequence by definition, a maximal hard sequence exists for every length m. Also, any tuple with more than $r + s - 1$ components cannot be a hard sequence. Thus, every hard sequence with order $r + s - 1$ is a maximal hard sequence.

171 We now argue that a maximal hard sequence will allow us to either recognize $\overline{H}^{=m}$ with a Σ_k^{P} machine or $\overline{E}^{=m}$ with a Σ_j^{P} machine (depending on the order of the maximal hard sequence). Suppose that $\vec{z} = \langle z_1, \ldots, z_t \rangle$ is a maximal hard sequence where $t < r$. We claim that the following is a Σ_k^{P} procedure for $\overline{H}^{=m}$.

172 **Procedure EasyH**

173 1. Input: $w \in \{0,1\}^m$.

174 2. Nondeterministically guess $\vec{v} \in V$ and $\langle u_1, \ldots, u_{\ell-1} \rangle \in \{0,1\}^{m\times(\ell-1)}$, where $\ell = r - t$.

175 3. Compute $f(\vec{v}, \langle u_1, \ldots, u_{\ell-1}, w, z_t, \ldots, z_1 \rangle) = (\vec{y}, \langle x_1, \ldots, x_r \rangle)$.

176 4. If $(\vec{v}, \vec{y}) \notin \mathrm{BL}_\omega^E \triangle \mathrm{coBL}_\omega^E$, reject

177 5. Accept if $x_\ell \in H$.

178 This procedure is computable by a Σ_k^{P} machine because a Σ_k^{P} machine can recognize BL_ω^E deterministically using parallel queries to a $\Sigma_{k-1}^{\mathrm{P}}$ oracle. Suppose that $w \in \overline{H}$ and Procedure EasyH does not accept. Then, $\langle z_1, \ldots, z_t, w \rangle$ would satisfy the definition of a hard sequence, which violates the maximality of $\vec{z}$. It remains to show that if Procedure EasyH accepts, then w is really in $\overline{H}$. First, since $\vec{z}$ is a hard sequence, each $z_i \in \overline{H}$ and each $x_i \in \overline{H}$ for $\ell + 1 \le i \le r$. Since the procedure accepted, $x_\ell \in H$ and $(\vec{v}, \vec{y}) \in \mathrm{BL}_\omega^E \triangle \mathrm{coBL}_\omega^E$. Suppose that ℓ is odd. Then $\vec{x} \in \mathrm{BL}_\omega^H$ since ℓ is the largest index of the $x_i \in H$. Furthermore, since f is a reduction from $\mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H$ to $\mathrm{coBL}_\omega^E \triangle \mathrm{BL}_\omega^H$, we have

179 $$(\vec{v}, \langle u_1, \ldots, u_{\ell-1}, w, z_t, \ldots, z_1 \rangle) \in \mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H \iff (\vec{y}, \vec{x}) \in \mathrm{coBL}_\omega^E \triangle \mathrm{BL}_\omega^H.$$

180 Therefore, $\langle u_1, \ldots, u_{\ell-1}, w, z_t, \ldots, z_1 \rangle \notin \mathrm{BL}_\omega^H$. Thus, $w \in \overline{H}$ (otherwise the largest index of the components in H would be odd). The reasoning for even ℓ is analogous.

181 Next, suppose that $\vec{z} = \langle z_1, \ldots, z_t \rangle$ is a maximal hard sequence and $t \ge r$. Then we claim that the following is a Σ_j^{P} procedure for $\overline{E}^{=m}$.

182 **Procedure EasyE**

183 1. Input: $w \in \{0,1\}^m$.

184 2. Nondeterministically guess $\langle v_1, \ldots, v_{\ell-1} \rangle \in \{0,1\}^{m\times(\ell-1)}$, where $\ell = r + s - t$.

185 3. Compute $f(\langle v_1, \ldots, v_{\ell-1}, w, z_t, \ldots, z_{r+1} \rangle, \langle z_r, \ldots, z_1 \rangle) = (\langle y_1, \ldots, y_s \rangle, \vec{x})$.

186 4. Accept if $y_\ell \in E$.

187 This procedure is clearly computable by a Σ_j^{P} machine. As before, if $w \in \overline{E}$ and the procedure rejects, then $\langle z_1, \ldots, z_t, w \rangle$ would constitute a hard sequence and violate the maximality of $\vec{z}$. Since $\vec{z}$ is a hard sequence, $\vec{z}\,' = \langle z_1, \ldots, z_r \rangle$ is also a hard sequence. By the definition of a hard sequence, for all $\vec{v} \in V$, if $f(\vec{v}, \vec{z}\,'^{\mathrm{R}}) = (\langle y_1, \ldots, y_s \rangle, \langle x_1, \ldots, x_r \rangle)$, then for all i, $1 \le i \le r$,

188 $$(\vec{v}, \vec{y}) \in \mathrm{BL}_\omega^E \triangle \mathrm{coBL}_\omega^E \Longrightarrow x_i \in \overline{H}.$$

189 Now, suppose $(\vec{v}, \vec{y})$ is indeed in $\mathrm{BL}_\omega^E \triangle \mathrm{coBL}_\omega^E$. Then, $\vec{x} = \langle x_1, \ldots, x_r \rangle \notin \mathrm{BL}_\omega^H$. Since $z_i \in \overline{H}$ for $1 \le i \le r$, it is also the case that $\vec{z}\,'^{\mathrm{R}} \notin \mathrm{BL}_\omega^H$. However, $(\vec{v}, \vec{y}) \in \mathrm{BL}_\omega^E \triangle \mathrm{coBL}_\omega^E$, $\vec{x} \notin \mathrm{BL}_\omega^H$ and $\vec{z}\,'^{\mathrm{R}} \notin \mathrm{BL}_\omega^H$ implies that either

190 $$(\vec{v}, \vec{z}\,') \in \mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H \text{ and } (\vec{y}, \vec{x}) \notin \mathrm{coBL}_\omega^E \triangle \mathrm{BL}_\omega^H$$

191 or

192 $$(\vec{v}, \vec{z}\,') \notin \mathrm{BL}_\omega^E \triangle \mathrm{BL}_\omega^H \text{ and } (\vec{y}, \vec{x}) \in \mathrm{coBL}_\omega^E \triangle \mathrm{BL}_\omega^H.$$

193 This contradicts the properties of f in the statement of this lemma. Thus, it must be the case that $(\vec{v}, \vec{y}) \notin \mathrm{BL}_\omega^E \triangle \mathrm{coBL}_\omega^E$, or in other words,

194 $$\vec{v} \in \mathrm{BL}_\omega^E \iff \vec{y} \in \mathrm{coBL}_\omega^E.$$

195 In particular, this is true when $\vec{v} = \langle v_1, \ldots, v_{\ell-1}, w, z_t, \ldots, z_{r+1}\rangle$.

196 Suppose that Procedure EasyE accepts the input string w. Then $y_\ell \in E$. Since $\vec{z}$ is a hard sequence, we also know that $y_i \in \overline{E}$ for $\ell + 1 \le i \le s$. If ℓ is odd, then y_ℓ implies that $\vec{y} \notin \mathrm{coBL}_\omega^E$. Thus, $\langle v_1, \ldots, v_{\ell-1}, w, z_t, \ldots, z_{r+1}\rangle \notin \mathrm{BL}_\omega^E$. Again, since $\vec{z}$ is a hard sequence, $z_i \notin E$ for $t \ge i \ge r+1$. Therefore, $w \notin E$. (Otherwise, $\langle v_1, \ldots, v_{\ell-1}, w, z_t, \ldots, z_{r+1}\rangle$ would be in BL_ω^E, because the index of the largest component in E would be odd.) The argument for ℓ even is analogous.

197 Finally, we point out that we have for each length m either a Σ_k^{P} machine that recognizes strings in $\overline{H}^{=m}$ or a Σ_j^{P} machine that recognizes strings in $\overline{E}^{=m}$. A polynomial length advice function can store a maximal hard sequence for each length m which is needed by the two machines. As argued at the beginning of this proof, if m is long enough, then we have $\overline{H} \in \Sigma_k^{\mathrm{P}}/\mathrm{poly}$. Therefore, the Polynomial Hierarchy collapses to $\Sigma_{k+2}^{\mathrm{P}}$ by Yap's Theorem [Yap83]. □

198 **Corollary 19** For $k > j$, let H and E be $\le_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P} and Σ_j^{P} respectively. Assuming that PH does not collapse, for all $0 < \epsilon < 1$ and for all $r(n)$ and $s(n)$ in $O(n^\epsilon)$, we have the strict containments:

199 $$\mathrm{P}^{H_{r(n)\text{-tt}} \| E_{s(n)\text{-tt}}} \subsetneq \mathrm{P}^{H_{r(n)\text{-tt}} \| E_{(s(n)\,+\,1)\text{-tt}}} \subsetneq \mathrm{P}^{H_{(r(n)\,+\,1)\text{-tt}}}.$$

5 Function classes

200 In this section we show that for *functions* computed by polynomial-time Turing machines with access to two oracles, the order of the queries is critical. Theorem 10 showed that if the easy questions are asked first, then there is an equivalent machine that asks the hard questions and the easy questions in parallel. The main theorem in this section states that the converse does not hold unless the Polynomial Hierarchy collapses.

201 **Theorem 20** For $k > j \ge 1$, let H and E be $\le_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P} and Σ_j^{P} respectively. Then, for all polynomial-time computable functions $r(n)$ and $s(n)$, where $r(n) \le \epsilon \log n$ for some $\epsilon < 1$ and $s(n) \in O(\log n)$, $\mathrm{PF}^{H_{r(n)\text{-tt}}; E_{s(n)\text{-tt}}} \not\subseteq \mathrm{PF}^{E_{s(n)\text{-tt}}; H_{r(n)\text{-tt}}}$ unless $\mathrm{PH} \subseteq \Sigma_{j+1}^{\mathrm{P}}$.

202 Theorem 20 follows immediately from Lemma 22 which we prove below. To motivate the proof of this lemma, we first prove a restricted version of Theorem 20 where H is Σ_2^{P} complete, E is NP complete and just one query is asked to each oracle.

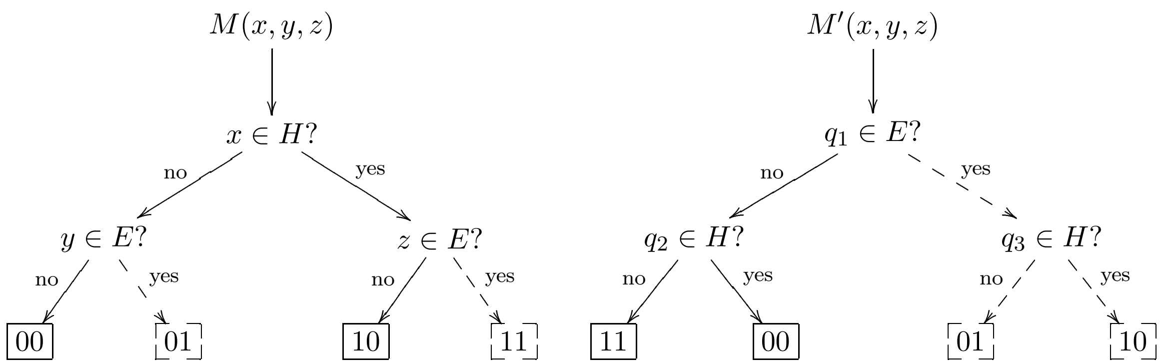
203 **Theorem 21** Let H and E be $\le_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_2^{P} and NP respectively. Then

204 $$\mathrm{PF}^{H_{1\text{-tt}}; E_{1\text{-tt}}} \subseteq \mathrm{PF}^{E_{1\text{-tt}}; H_{1\text{-tt}}} \Longrightarrow \mathrm{PH} \subseteq \Sigma_2^{\mathrm{P}}.$$

205 **Proof:** Consider the function:

206 $$\textsc{FirstH}(x, y, z) = \begin{cases} H(x)E(y) & \text{if } x \notin H \\ H(x)E(z) & \text{if } x \in H \end{cases}$$

207 Recall that $H(\cdot)$ and $E(\cdot)$ denote the characteristic functions of H and E and that $H(x)E(y)$ represents the concatenation of $H(x)$ and $E(y)$. The function $\textsc{FirstH}(x, y, z)$ is easily computable



214 Figure 2: An example of the easy case. Using an E oracle we determine that $y \notin E$, $z \notin E$ and $q_1 \notin E$. Thus, $\textsc{Out}^E(M(x,y,z)) = \{00, 10\}$ and $\textsc{Out}^E(M'(x,y,z)) = \{11, 00\}$. We then conclude that $\textsc{FirstH}(x,y,z)$ must be 00 and $x \notin H$.

by a $\mathrm{PF}^{H_{1\text{-tt}};E_{1\text{-tt}}}$ machine M that asks whether $x \in H$ followed by the appropriate query to E. However, there is no obvious way to compute $\textsc{FirstH}$ in $\mathrm{PF}^{E_{1\text{-tt}};H_{1\text{-tt}}}$ since it is not clear which of $y \in E$ and $z \in E$ to ask.

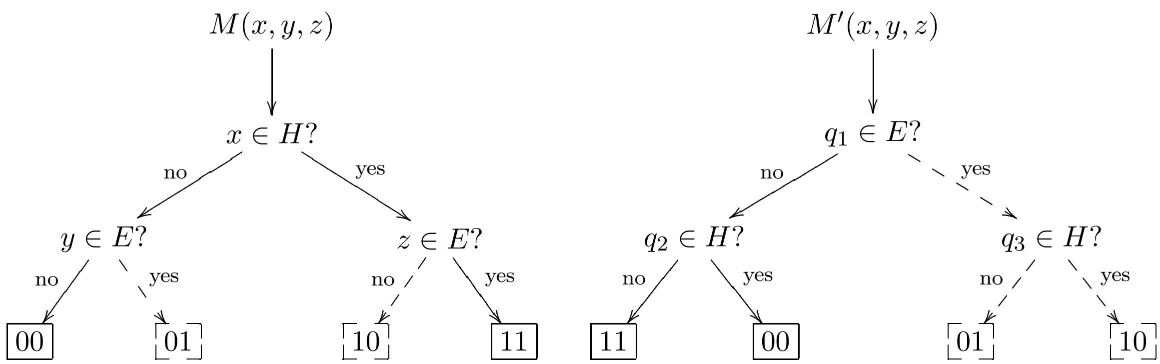
208 If $\mathrm{PF}^{H_{1\text{-tt}};E_{1\text{-tt}}} \subseteq \mathrm{PF}^{E_{1\text{-tt}};H_{1\text{-tt}}}$, then $\textsc{FirstH} \in \mathrm{PF}^{E_{1\text{-tt}};H_{1\text{-tt}}}$ via some polynomial-time Turing machine M'. Consider the branches of the oracle query trees of M and M' on input (x,y,z) where the queries to E are answered correctly. Let $\textsc{Out}^E(M(x,y,z))$ and $\textsc{Out}^E(M'(x,y,z))$ denote the set of outputs of the machines on these paths. Since the queries to H may be correctly or incorrectly answered, the sets $\textsc{Out}^E(M(x,y,z))$ and $\textsc{Out}^E(M'(x,y,z))$ have at most two values each. From our construction of M, we know that

209
$$\textsc{Out}^E(M(x,y,z)) = \{\, 0E(y), 1E(z) \,\}.$$

210 Now, suppose that $\textsc{Out}^E(M'(x,y,z))$ is not equal to $\textsc{Out}^E(M(x,y,z))$ (see Figure 2). Since both M and M' compute $\textsc{FirstH}$, the correct value of $\textsc{FirstH}(x,y,z)$ must appear in both sets. Thus,

211
$$\begin{aligned}&\textsc{Out}^E(M'(x,y,z)) \neq \textsc{Out}^E(M(x,y,z))\\ &\quad\Longrightarrow \textsc{Out}^E(M'(x,y,z)) \cap \textsc{Out}^E(M(x,y,z)) = \{\textsc{FirstH}(x,y,z)\}.\end{aligned}$$

212 By definition, the first bit of $\textsc{FirstH}(x,y,z)$ is $H(x)$. So, if there exists (y,z) such that the sets $\textsc{Out}^E(M'(x,y,z))$ and $\textsc{Out}^E(M(x,y,z))$ differ, we can determine whether $x \in H$ or $x \in \overline{H}$ by guessing (y,z) and answering queries to E. When such a (y,z) pair exists, we call x an *easy string*. Otherwise, no such (y,z) pair exists and we call x a *hard string* (see Figure 3). For each length n, we will consider two cases. Either all of the strings of length n are easy or there exists a hard string for length n. Combining the two cases will allow us to collapse PH. The simplest way to combine the two cases is to use an advice function to provide a hard string, if one exists. This will collapse PH to the Σ_4^{P} level. A more sophisticated approach will bring PH down to the Σ_2^{P} level.

216 

217 Figure 3: An example of the hard case. When x is a hard string, for all choices of y and z, $\textsc{Out}^E(M(x, y, z)) = \textsc{Out}^E(M'(x, y, z))$. In this example, for this choice of y and z, we used an E oracle to determine that $y \notin E$, $z \in E$ and $q_1 \notin E$. Thus, $\textsc{Out}^E(M(x, y, z)) = \textsc{Out}^E(M'(x, y, z)) = \{00, 11\}$.

218

219 Formally, the two cases are (for polynomial bounded $\ell(n)$ specified later):

220 **Case 1:** (All strings of length n are easy.)

221 $$(\forall x, |x| = n)(\exists y, z, |y| = |z| = \ell(n))[\textsc{Out}^E(M(x, y, z)) \neq \textsc{Out}^E(M'(x, y, z))].$$

222 **Case 2:** (There exists a hard string for length n.)

223 $$(\exists x, |x| = n)(\forall y, z, |y| = |z| = \ell(n))[\textsc{Out}^E(M(x, y, z)) = \textsc{Out}^E(M'(x, y, z))].$$

224 For each n, we can provide an NP^E machine with 1 bit of advice stating whether Case 1 or Case 2 holds for strings of length n. If Case 1 holds, then the NP^E machine can determine whether $x \in \overline{H}$ by guessing (y, z) and checking whether $\textsc{Out}^E(M(x, y, z)) \neq \textsc{Out}^E(M'(x, y, z))$. When the "correct" (y, z) is guessed, the set,

225 $$\textsc{Out}^E(M(x, y, z)) \cap \textsc{Out}^E(M'(x, y, z))$$

226 contains one string ab where $a = H(x)$. Thus, when all strings are easy, there exist NP^E machines which can recognize $H^{=n}$ and $\overline{H}^{=n}$.

227 If there exists a hard string for length n, then the advice function also provides a hard string x. Note that $\textsc{Out}^E(M'(x, y, z))$ can be computed using only 1 query to E. However, in Case 2 we have

228 $$\textsc{Out}^E(M'(x, y, z)) = \textsc{Out}^E(M(x, y, z)) = \{\, 0E(y), 1E(z) \,\}.$$

229 Thus, a P machine with the advice and 1 query to E can compute χ_2^E — that is, χ_2^E is 2-enumerable. Using standard tree pruning techniques [ABG90, BKS95, Ogi95, AA96], we can show that E can be solved in P with advice. (A detailed discussion of the tree pruning procedure follows this proof.)

230 So, for each length n, given polynomial advice, we either have an NP^E machine that recognizes $H^{=n}$ (Case 1) or a P machine that recognizes $E^{=n}$ (Case 2). Furthermore, the sizes and the running times of these machines are bounded by a single polynomial in n. Thus, as in Lemma 18, we can combine the two cases to get $\Pi_2^{\mathrm{P}} \subseteq \Sigma_2^{\mathrm{P}}/\mathrm{poly}$ which collapses PH to Σ_4^{P} using Yap's theorem

[Yap83]. We can improve upon the collapse of PH in two ways. First, we can modify the definitions of Case 1 and 2 to get a $\mathrm{P}^{E}_{\mathrm{tt}}$ machine for $H^{=n}$ in Case 1 and a P machine for $E^{=n}$ in Case 2. This method uses the techniques of Amir, Beigel and Gasarch [ABG90] to construct a better polynomial advice function. This improvement would collapse PH to Σ^{P}_3. The second method uses the latest refinements of the hard/easy argument [HHH99, BF99] to show that PH actually collapses to Σ^{P}_2. One key difference in the new approach is that we do not have to look for a hard string x; we simply guess whether the input string is a hard string. We sketch the proof of the second method next.

231 We construct an $\mathrm{NP}^{\mathrm{NP}}$ machine N to recognize $\overline{H}$ as follows. First, we rewrite H as

232 $$H = \{\, x : (\exists^{\mathrm{P}} u)(\forall^{\mathrm{P}} v)[R(x,u,v)] \,\}$$

233 for some polynomial-time computable predicate R. On input x, the computation of N is divided into two parallel strategies. The first strategy presupposes that x is an easy string. In this first strategy, N guesses $(y,z) \in \{0,1\}^{\ell(n)} \times \{0,1\}^{\ell(n)}$ and checks whether $\textsc{Out}^{E}(M(x,y,z))$ differs from $\textsc{Out}^{E}(M'(x,y,z))$ using its NP oracle. Note that when the correct pair (y,z) is found, the machine N can prove that x is an easy string. Let ab be the single string in $\textsc{Out}^{E}(M(x,y,z)) \cap \textsc{Out}^{E}(M'(x,y,z))$. Then, $x \in \overline{H}$ if and only if $a = 0$. If no such (y,z) pair exists, then all computation paths following the first strategy will reject.

234 The second strategy presupposes that the input string x is a hard string. In this case, N asks its NP oracle whether $(\forall^{\mathrm{P}} u)(\exists^{\mathrm{P}} v)[\neg R(x,u,v)]$. Normally, an NP oracle cannot answer this Π^{P}_2 question. However, using x as a hard string, we can use Procedure Prune (Figure 4) to find the witness v such that $\neg R(x,u,v)$ is true (for any fixed u).[3] Let, N' be an NP machine which guesses u and looks for the witness v deterministically using the tree pruning procedure. This procedure requires an algorithm to 2-enumerate χ^{E}_2 for strings up to a certain polynomial length $\ell(n)$. This $\ell(n)$ is the bound used in the formal definitions of Case 1 and Case 2. If such a witness v is found, then N' rejects. If Procedure Prune terminates without producing a witness, then N' accepts. So, if $N'(x)$ rejects on all paths, $x \in \overline{H}$. Hence, in the second strategy, the base machine N will simply ask the NP oracle whether $N'(x)$ rejects. Note that in the second strategy, the base machine N will accept only if the search for witnesses succeeds for all u. Thus, acceptance by the second strategy is correct even if it turns out that x is not a hard string. Combining the two strategies, we have an $\mathrm{NP}^{\mathrm{NP}}$ algorithm for $\overline{H}$. Therefore, PH collapses to Σ^{P}_2. □

236 The proofs of Theorem 21 and Lemma 22 use a tree pruning technique to search for witnesses. This tree pruning technique was discovered independently by Beigel, Kummer and Stephan [BKS95], by Ogihara [Ogi95] and by Agrawal and Arvind [AA96]. This technique was used to show that if a language A is d-self-reducible and is a bd-cylinder, then either $A \in \mathrm{P}$ or A is p-superterse. Here a language A is called a bd-cylinder if there exists a polynomial-time computable binary OR function f such that for all x and y

237 $$x \in A \vee y \in A \iff f(x,y) \in A.$$

238 In our applications we need to consider the unbounded analog of a bd-cylinder. We call a set A a d-cylinder[4] if there exists a polynomial-time computable any-ary OR function f such that for all sequences $\langle x_1, \ldots, x_t \rangle$

240 $$\#^{A}_{\omega}(x_1, \ldots, x_t) \geq 1 \iff f(x_1, \ldots, x_t) \in A.$$

235 [3] Other tree pruning procedures, such as those by Amir, Beigel and Gasarch [ABG90] or Hoene and Nickelsen [HN93] would also work in this restricted case.

239 [4] Alternative terminology in the literature refer to such sets A as sets that have OR_{ω} [CK95, AA96].

241 We will use this tree pruning algorithm in several situation, so we describe it now in general terms. The requirements and parameters for using Procedure Prune (Figure 4) are:

242 1. An input string w of length n.

243 2. A language B formulated as $B = \{w : (\exists u \in \{0,1\}^{b(|w|)})[P(w,u)]\}$ where $b(n)$ is a polynomial in n and $P(w,u)$ is some predicate, not necessarily computable in polynomial time.

244 If $w \in B$ and $P(w,u)$ holds, then we say that u is a witness for $w \in B$. The objective of the tree pruning is to find such a witness if it exists. From B we define an auxiliary language B' to be the set of witness prefixes. (Clearly, $w \in B$ if and only if $(w,\epsilon) \in B'$.)

245 $$B' = \{(w,u) : (\exists v \in \{0,1\}^*)[|uv| = b(|w|) \wedge P(w,uv)]\}.$$

246 3. A language A that is a d-cylinder via an any-ary OR function f.

247 The language B' must reduce to A via a $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduction h.

248 4. A polynomial-time computable function $t(n) \in O(\log n)$.

249 From $t(n)$ we define a related polynomial-time computable function $\ell(n) \in n^{O(1)}$ as follows. For a single instance w of B, with $|w| = n$, we need to consider $2^{t(n)-1}$ instances of B' each of length up to $n + b(n)$. Each instance of B' is then reduced to A using the reduction h. This generates $2^{t(n)-1}$ instances of A which is combined into one instance of A using the any-ary OR function f. Then $\ell(n)$ is defined to be a bound on the length of this output from f.

250 5. A pruning function $g : \{0,1\}^{\ell(n)\times t(n)} \to \{0,1\}^{t(n)}$ such that

251 $$\forall x_1, \ldots, x_{t(n)} \in \{0,1\}^{\ell(n)},\ g(x_1, \ldots, x_{t(n)}) \neq \chi_\omega^A(x_1, \ldots, x_{t(n)}).$$

252 The pruning function g is not necessarily computable in polynomial time.

272 The general strategy in Procedure Prune is a fairly standard tree pruning strategy. The procedure maintains a list Q of potential witness prefixes. In each iteration of the main loop, each prefix is extended by appending a 0 and a 1 to the prefix. This doubles the number of prefixes in Q. The list Q is then pruned down to $2^{t(n)} - 1$ elements. The entire procedure terminates when the prefixes in Q have reached full length and cannot be further extended.

273 We claim that if $w \in B$, then Procedure Prune finds a witness for $w \in B$. The main observation is that if $w \in B$, then at every step of the procedure, Q contains some (w,u) where u is the prefix of a witness — i.e., $(w,u) \in B'$. This is certainly true at the beginning of the procedure, since in Step 2 we add every prefix of length $t(n)$ to Q. Suppose that during some iteration of Step 4, the pair (w,u_z) removed from Q is the only pair in $Q \cap B'$. Then, $y_z \in A$ and for $1 \leq i \leq 2^{t(n)}$, $i \neq z \Longrightarrow y_i \notin A$. In that case, $\chi_\omega^A(x_1, \ldots, x_{t(n)})$ is in fact equal to σ_z. (Here $\{\sigma_1, \ldots, \sigma_{2^{t(n)}}\} = \{0,1\}^{t(n)}$ and $\sigma_i[d]$ denotes the d-th bit of σ_i.) This violates our assumptions about g. Thus, $w \in B$ implies that $Q \cap B'$ is never empty throughout the execution of the procedure. Obviously, if $w \notin B$, then the procedure does not produce any witnesses. Also, note that since $|D_d| = 2^{t(n)-1}$, we can also guarantee that each x_d has length bounded by $\ell(n)$.

274 Procedure Prune takes a polynomial number of iterations, since $p(n)$ is bounded by a polynomial and $|Q|$ never exceeds $2^{t(n)+1}$. However, the complexity of the entire procedure also depends on the complexity of deciding $P(w,u)$ and on the complexity of g. In the proof below, we will use Procedure Prune in two different settings, each with a different complexity.

Procedure Prune

1. Input: w with $|w| = n$

   W.l.o.g. assume that $b(n) \geq t(n)$.

2. $Q = \{(w, u) : u \in \{0,1\}^{t(n)}\}$.

   Initialize Q *to be all witness prefixes of length* $t(n)$.

3. If for each $(w, u) \in Q$, $|u| = b(n)$, then output the first $(w, u) \in Q$ such that $P(w, u)$ holds. Terminate the procedure.

   When $|u| = b(n)$, *then the prefix* u *cannot be further extended.*

4. Repeat until $|Q| = 2^{t(n)} - 1$:

   (a) let $(w, u_1), \ldots, (w, u_{2^{t(n)}})$ be the first $2^{t(n)}$ elements in Q.

   (b) for $1 \leq i \leq 2^{t(n)}$, let $y_i = h(w, u_i)$. *Recall that* h *reduces* B' *to* A.

   (c) for $1 \leq d \leq t(n)$, let $D_d = \{y_i : \sigma_i[d] = 1\}$.

   Here $\{\sigma_1, \ldots, \sigma_{2^{t(n)}}\} = \{0,1\}^{t(n)}$ *and* $\sigma_i[d]$ *denotes the* d*-th bit of* σ_i.

   (d) for $1 \leq d \leq t(n)$, use the any-ary OR function f to construct x_d such that

   $$x_d \in A \iff A \cap D_d \neq \emptyset.$$

   (e) let z be the index such that $\sigma_z = g(x_1, \ldots, x_{t(n)})$. Remove (w, u_z) from Q.

   The proof in the text shows that u_z *cannot be the unique witness prefix in* Q.

5. Replace each $(w, u) \in Q$ with $(w, u0)$ and $(w, u1)$. Goto Step 3.

   Extend each prefix by one bit. This doubles the size of Q.

Figure 4: Tree-pruning procedure used in Theorem 21 and Lemma 22.

Lemma 22 For $k > j \geq 1$, let H and E be $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for Σ_k^{P} and Σ_j^{P} respectively. Suppose that $r(n)$ and $s(n)$ are polynomial-time computable functions such that $r(n) \leq \epsilon \log n$ for some $\epsilon < 1$ and $s(n) \in O(\log n)$. Then, for all oracles X,

$$\mathrm{PF}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}} \subseteq \mathrm{PF}^{E_{s(n)\text{-tt}};X_{r(n)\text{-tt}}} \Longrightarrow \mathrm{PH} \subseteq \Sigma_{j+1}^{\mathrm{P}}.$$

Proof: This proof is similar to the proof of Theorem 21. As before, in the easy case we have $\mathrm{Out}^E(M(\cdots)) \neq \mathrm{Out}^E(M'(\cdots))$ and in the hard case we have $\mathrm{Out}^E(M(\cdots)) = \mathrm{Out}^E(M'(\cdots))$. There are two differences between this proof and the proof of Theorem 21. First, in the proof of Theorem 21, k is coincidentally equal to $j+1$. In this proof, instead of showing that $\overline{H} \in \mathrm{NP}^E$, we prove that a Π_{j+1}^{P} complete language L is contained in NP^E. The second difference is that in the easy case of Theorem 21, $\mathrm{Out}^E(M(x,y,z)) \cap \mathrm{Out}^E(M'(x,y,z))$ contains exactly one string. This allowed us to determine whether $x \in \overline{H}$ immediately. In the easy case of this proof, $\mathrm{Out}^E(M(\cdots)) \cap \mathrm{Out}^E(M'(\cdots))$ may contain more than one string. Nevertheless we can still determine whether $w \in L$ using Procedure Prune to find a witness for $w \in L$.

278 Consider the function $\textsc{FirstH}(\vec{x}, \vec{y}_0, \ldots, \vec{y}_{q(n)}) = \sigma\chi^E_\omega(y_\sigma)$ where $\sigma = \chi^H_\omega(\vec{x})$ and $q(n) = 2^{r(n)}-1$. Here $\vec{x}$ is a sequence with $r(n)$ components and each $\vec{y}_i$ is a sequence with $s(n)$ components. Each component of $\vec{x}$ and $\vec{y}_\sigma$ has length $m = n/(r(n)+s(n)2^{r(n)})$ so that $|(\vec{x}, \vec{y}_0, \ldots, \vec{y}_{q(n)})| = n$. W.l.o.g. we assume that there exist polynomial-time computable functions $\tilde{r}(m)$ and $\tilde{s}(m)$ such that for $n = m\tilde{r}(m) + m\tilde{s}(m)2^{\tilde{r}(m)}$, $\tilde{r}(m) = r(n)$ and $\tilde{s}(m) = s(n)$. (This is possible because $r(n) \leq \epsilon \log n$.) This allows us to express the number of components in $\vec{x}$ and $\vec{y}_\sigma$ in terms of the length of each component rather than the length of $(\vec{x}, \vec{y}_0, \ldots, \vec{y}_{q(n)})$. Since $r(n)$ and $s(n)$ are in $O(\log n)$, $\tilde{r}(m)$ and $\tilde{s}(m)$ are also in $O(\log n)$.

279 Clearly, $\textsc{FirstH}(\vec{x}, \vec{y}_0, \ldots, \vec{y}_{q(n)})$ can be computed by a $\mathrm{PF}^{H_{r(n)\text{-tt}};E_{s(n)\text{-tt}}}$ machine M which uses $r(n)$ parallel queries to H to compute $\sigma = \chi^H_\omega(\vec{x})$ and then uses $s(n)$ parallel queries to E to compute $\chi^E_\omega(\vec{y}_\sigma)$. Now, suppose that there exists a $\mathrm{PF}^{E_{s(n)\text{-tt}};X_{r(n)\text{-tt}}}$ machine M' which computes $\textsc{FirstH}$. Then, we claim that there exists an NP^E machine which recognizes L, a Π^{P}_{j+1} complete language.

280 We construct an NP^E machine which uses Procedure Prune to look for witnesses for $w \in \overline{L}$, where $|w| = n$. Since $L \in \Pi^{\mathrm{P}}_{j+1}$, $\overline{L}$ can be written as:

281 $$\overline{L} = \{ w : (\exists^{\mathrm{P}} u)(\forall^{\mathrm{P}} v) R(w, u, v) \},$$

282 where $R(w, u, v)$ is a $\Delta^{\mathrm{P}}_{j-1}$ computable predicate. Here, $\overline{L}$ will take the place of the language B in Procedure Prune described above and H will take the place of the language A. During the execution of Procedure Prune, we will encounter many instances of $\vec{x} = \langle x_1, \ldots, x_{t(n)} \rangle$ produced in Step 4(c). Here, each x_i has length $\leq \ell(n)$. We may assume by padding that each x_i has length exactly $m = \max\{\ell(n), \ell'(n)\}$ for a polynomial-bounded $\ell'(n)$ specified later. Then we can set $t(n) = \tilde{r}(m)$. To satisfy the requirements of Procedure Prune, we must also provide a function g such that $g(\vec{x}) \neq \chi^H_\omega(\vec{x})$. This is accomplished by an NP^E procedure described next.

283 Let $\tilde{q}(m) = 2^{\tilde{r}(m)} - 1$. For each pair of vectors $(\vec{x}, \vec{y})$, where $\vec{x} = \langle x_1, \ldots, x_{\tilde{r}(m)} \rangle \in \{0,1\}^{m \times \tilde{r}(m)}$ and $\vec{y} = \langle \vec{y}_0, \ldots, \vec{y}_{\tilde{q}(m)} \rangle \in \{0,1\}^{m \times \tilde{s}(m) \times \tilde{q}(m)}$, let $\textsc{Out}^E(M(\vec{x}, \vec{y}))$ be the set of outputs of $M(\vec{x}, \vec{y})$ on branches of the oracle query tree where the queries to E are answered correctly. The set $\textsc{Out}^E(M'(\vec{x}, \vec{y}))$ is defined analogously. From the description of the machine M, we know that

284 $$\textsc{Out}^E(M(\vec{x}, \vec{y})) = \{ \sigma\chi^E_\omega(\vec{y}_\sigma) : \sigma \in \{0,1\}^{\tilde{r}(m)} \}.$$

285 We call $\vec{x}$ *easy* if for some $\vec{y}$, $\textsc{Out}^E(M(\vec{x}, \vec{y})) \neq \textsc{Out}^E(M'(\vec{x}, \vec{y}))$. In this case, for at least one string $\sigma \in \{0,1\}^{\tilde{r}(m)}$, $\sigma\chi^E_\omega(\vec{y}_\sigma) \notin \textsc{Out}^E(M'(\vec{x}, \vec{y}))$. Then, we can eliminate the string σ as a possible value for $\chi^H_\omega(\vec{x})$. Thus, we have an NP^E algorithm which given $\vec{x}$ as input, produces $\sigma \in \{0,1\}^{\tilde{r}(m)}$ such that $\sigma \neq \chi^H_\omega(\vec{x})$: guess $\vec{y}$, use the E oracle to compute $\textsc{Out}^E(M(\vec{x}, \vec{y}))$ and $\textsc{Out}^E(M'(\vec{x}, \vec{y}))$, then find σ such that $\sigma\chi^E_\omega(\vec{y}_\sigma) \notin \textsc{Out}^E(M'(\vec{x}, \vec{y}))$. This algorithm computes the function g required in Procedure Prune. The entire tree pruning procedure can be accomplished by an NP^E computation because deciding the predicate $(\forall^{\mathrm{P}} v) R(w, u, v)$ can be done with the E oracle.

286 Now, suppose that all the instances of $\vec{x}$ encountered during this execution of Procedure Prune are indeed easy. When the procedure terminates in Step 2, if no witnesses for $w \in \overline{L}$ were found, the NP^E algorithm for L accepts. On the other hand, suppose that one of the $\vec{x}$ is a hard string, then every computation branch of the NP^E computation for L will reject. This is because no computation branch managed to guess $\vec{y}$ such that

287 $$\textsc{Out}^E(M(\vec{x}, \vec{y})) \neq \textsc{Out}^E(M'(\vec{x}, \vec{y})).$$

288 To guard against the possibility that $\vec{x}$ is a hard string, every time a new instance of $\vec{x}$ is generated in Step 4(c), we start a new tree pruning procedure which assumes that $\vec{x}$ is a hard string. Recall that if $\vec{x} \in \{0,1\}^{m \times \tilde{r}(m)}$ is a hard string, then

289 $$\forall \vec{y} = \langle \vec{y}_0, \ldots, \vec{y}_{\tilde{q}(m)} \rangle \in \{0,1\}^{m \times \tilde{s}(m) \times \tilde{q}(m)}, \textsc{Out}^E(M(\vec{x}, \vec{y})) = \textsc{Out}^E(M'(\vec{x}, \vec{y})).$$

290 Observe that given $\textsc{Out}^E(M(\vec{x},\vec{y}))$, we can recover $\chi^E_\omega(\vec{y}_\sigma)$ for $0 \le \sigma \le \tilde{q}(m)$, since $\sigma\chi^E_\omega(\vec{y}_\sigma)$ is the unique string in $\textsc{Out}^E(M(\vec{x},\vec{y}))$ with prefix σ. In order to use Procedure Prune, we need to produce a function g that, for any $\vec{z} \in \{0,1\}^{m\times t(n)}$, outputs a value in $\{0,1\}^{t(n)}$ that is not $\chi^E_\omega(\vec{z})$. Note that $t(n)$ must be in $O(\log n)$ whereas $\vec{y}$ has $\tilde{s}(m)2^{\tilde{r}(m)}$ components. In our procedure for g, we fill most of $\vec{y}$ with dummy strings. Let $t(n) = \tilde{s}(m)+1$, then $t(n) \in O(\log n)$. On input $\vec{z} = \langle z_1,\ldots,z_{t(n)}\rangle \in \{0,1\}^{m\times t(n)}$, our procedure for g constructs $\vec{y} = \langle 0^m,\ldots,0^m,z_1,\ldots,z_{t(n)}\rangle$ where the 0^m components are repeated $\tilde{s}(m)2^{\tilde{r}(m)} - t(n)$ times.

291 Then, the output of $g(\vec{z})$ can be computed as follows. We simulate $M'(\vec{x},\vec{y})$ where $\vec{x}$ is the possible hard string and $\vec{y}$ is defined as above with $\vec{z}$ embedded. Recall that M' is a $\mathrm{PF}^{E_{s(n)\text{-tt}};X_{r(n)\text{-tt}}}$ computation and queries the E oracle first. For now, fix a sequence $\xi \in \{0,1\}^{\tilde{s}(m)}$ of possible responses from the E oracle. We simulate $M'(\vec{x},\vec{y})$ using ξ as the response from E and consider the $2^{\tilde{r}(m)}$ possible computation paths that follow. Each of these computation paths assumes a different response from the X oracle. At the end of each path, $M'(\vec{x},\vec{y})$ should output a value of the form $\sigma\alpha$ where $\sigma \in \{0,1\}^{\tilde{r}(m)}$ and $\alpha \in \{0,1\}^{\tilde{s}(m)}$. Let $\textsc{Out}^\xi(M'(\vec{x},\vec{y}))$ be the set of these $2^{\tilde{r}(m)}$ outputs. Each σ should appear exactly once as a prefix of a string in $\textsc{Out}^\xi(M'(\vec{x},\vec{y}))$. If not, then we know that either ξ is not the correct response from E or that $\vec{x}$ is not a hard string, since $\textsc{Out}^E(M'(\vec{x},\vec{y}))$ must equal $\textsc{Out}^E(M(\vec{x},\vec{y}))$ if $\vec{x}$ is a hard string. In any case we can move on to the next value for ξ. Now suppose that σ does appear exactly once in $\textsc{Out}^\xi(M'(\vec{x},\vec{y}))$. Then, for each string $\sigma\alpha$ in $\textsc{Out}^\xi(M'(\vec{x},\vec{y}))$, α is a possible value for $\chi^E_\omega(\vec{y}_\sigma)$. By concatenating the α's in the correct order, we obtain a possible value for $\chi^E_\omega(\vec{y})$. Thus, for each $\xi \in \{0,1\}^{\tilde{s}(m)}$ we have a possible value for $\chi^E_\omega(\vec{y})$. Since one of the ξ is actually the correct response from E, one of these possible values is in fact $\chi^E_\omega(\vec{y})$. Next, for each candidate for $\chi^E_\omega(\vec{y})$, we consider the last $t(n)$ bits as a candidate for $\chi^E_\omega(\vec{z})$. (The leading bits corresponds to the dummy components 0^m.) Again, one of these candidates is in fact $\chi^E_\omega(\vec{z})$. Thus, we have $2^{t(n)-1}$-enumerated $\chi^E_\omega(\vec{z})$ for any $\vec{z} \in \{0,1\}^{m\times t(n)}$. Finally, since $2^{t(n)} > 2^{t(n)-1}$, we can use any string in $\{0,1\}^{t(n)}$ that is not one of the candidates for $\chi^E_\omega(\vec{z})$ as the output for $g(\vec{z})$. Note that the entire procedure for $g(\vec{z})$ did not use any oracle queries to E or to X. Therefore, assuming that $\vec{x}$ is really a hard string, we have a deterministic polynomial time algorithm for the function g. This satisfies the requirements of Procedure Prune.

292 So, we can again use Procedure Prune. This time we use the tree pruning procedure to find a witness for $(w,u) \in L_2$ where

293 $$L_2 = \{\, (w,u) : (\exists^{\mathrm{P}} v)\neg R(w,u,v) \,\}.$$

294 Here, $R(w,u,v)$ is the same predicate used in the definition of the $\Sigma^{\mathrm{P}}_{j+1}$-complete language $\overline{L}$. Clearly, $w \in L$ if and only if for all u, $(w,u) \in L_2$. In this execution of Procedure Prune, L_2 takes the place of the language B and E takes the place of the language A. We know that $B' \le^{\mathrm{P}}_{\mathrm{m}} A$ because $R(w,u,v)$ is a $\Delta^{\mathrm{P}}_{j-1}$ predicate and E is complete for Σ^{P}_j. Also, in this case, $t(n) = \tilde{s}(m)+1$ and the function g is computed as described above. Finally, since $R(w,u,v)$ is a $\Delta^{\mathrm{P}}_{j-1}$ predicate, the entire tree pruning procedure for L_2 can be executed by a $\Delta^{\mathrm{P}}_{j-1}$ machine.

295 Let N be a Σ^{P}_j machine which on input w, guesses u and uses the $\Delta^{\mathrm{P}}_{j-1}$ tree pruning procedure described above to find a witness v for $(w,u) \in L_2$. If such a witness is found, N *rejects*. If the tree pruning procedure terminates without producing a witness, then N *accepts*. Thus, if $N(w)$ rejects on all paths, a witness for $(w,u) \in L_2$ was found for every u. Our NP^E algorithm for L is simply to ask E whether $N(w)$ accepts. If the answer is no, then the NP^E algorithm accepts. Furthermore, let $\ell'(n)$ denote the length of the longest component of the sequences given to the function g in this execution of Procedure Prune. Since $m = \max\{\ell(n), \ell'(n)\}$, this guarantees that a single hard string $\vec{x}$ is enough for all the tree pruning procedures invoked by N.

296 Finally, suppose that $\vec{x}$ is not a hard string and our NP^E procedure accepted. Then, we claim that w is nevertheless in L. To see this, simply note that the NP^E algorithm will accept only when a witness v is found for every u. The validity of this witness was checked in the $\Delta^{\mathrm{P}}_{j-1}$ tree pruning procedure by evaluating $R(w,u,v)$ directly. Thus, even when $\vec{x}$ is not a hard string, it is possible that the NP^E algorithm is lucky and accepts correctly. However, this algorithm will never accept incorrectly.

297 Therefore, in both the easy case and the hard case, the tree pruning procedures have one-sided error — the strategies might reject incorrectly, but never accept incorrectly. That is, either all the $\vec{x}$ are easy strings and the top level tree pruning succeeds, or some $\vec{x}$ is a hard string and the second tree pruning procedure succeeds. In either case, the overall procedure accepts x when $x \in L$ and rejects on all branches when $x \notin L$. Thus, $\Pi^{\mathrm{P}}_{j+1} \subseteq \mathrm{NP}^E$ and PH collapses to $\Sigma^{\mathrm{P}}_{j+1}$. □

298 The proof of Theorem 20 can be extended to the case where more than two rounds of queries are made. For example, we can modify the proof to show that for polynomial-time computable $r(n)$, $s(n)$ and $t(n)$, such that $r(n) + s(n) \leq \epsilon \log n$ (for some $\epsilon < 1$) and $p(n) \in O(\log n)$

299 $$\mathrm{PF}^{H_{r(n)\text{-tt}};H_{s(n)\text{-tt}};E_{p(n)\text{-tt}}} \subseteq \mathrm{PF}^{H_{r(n)\text{-tt}};E_{p(n)\text{-tt}};H_{s(n)\text{-tt}}} \Longrightarrow \mathrm{PH} \subseteq \mathrm{NP}^E.$$

300 In the modified proof, the two sets $\textsc{Out}^E(M(\vec{x},\vec{y}))$ and $\textsc{Out}^E(M'(\vec{x},\vec{y}))$ would be defined as before. If the two sets are not equal, then χ^H_ω on inputs from $\{0,1\}^{m\times(\tilde{r}(m)+\tilde{s}(m))}$ is $(2^{\tilde{r}(m)+\tilde{s}(m)}-1)$-enumerable. If the two sets are equal, then χ^E_ω is $2^{\tilde{p}(m)}$-enumerable on inputs from $\{0,1\}^{m\times(\tilde{p}(m)+1)}$. Combining the two tree pruning procedures produces an NP^E algorithm for a coNP^E language. We leave the details of this proof to the reader.

301 It is interesting to note that in the proof of Theorem 20, the complexity of the language H is not used very much. The only requirement that we have for H is that it is hard for coNP^E. In fact, even when E is NP complete and H is Σ^{P}_{17} complete, we only use the fact that all Π^{P}_2 languages reduce to H. The same holds for H being PSPACE complete.

302 However, when the internal complexity of H is very high, that is, when H is bi-immune for NP^E, then we can exploit the complexity of H itself to obtain a stronger collapse:

303 **Theorem 23** Let E be $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for Σ^{P}_j where $j \geq 1$ and let H be a set bi-immune to NP^E. Then, for all oracles X,

304 $$\mathrm{PF}^{H_{1\text{-tt}};E_{1\text{-tt}}} \subseteq \mathrm{PF}^{E_{1\text{-tt}};X_{1\text{-tt}}} \Longrightarrow \mathrm{P} = \mathrm{NP}.$$

305 **Proof Sketch:** Suppose that $\mathrm{PF}^{H_{1\text{-tt}};E_{1\text{-tt}}} \subseteq \mathrm{PF}^{E_{1\text{-tt}};X_{1\text{-tt}}}$. Let $\textsc{FirstH}(x,y,z)$ be the function we defined in Theorem 21. As before, $\textsc{FirstH}(x,y,z)$ is easily computable by a $\mathrm{PF}^{H_{1\text{-tt}};E_{1\text{-tt}}}$ machine M. Now, suppose that there exists a $\mathrm{PF}^{E_{1\text{-tt}};X_{1\text{-tt}}}$ machine M' which also computes $\textsc{FirstH}(x,y,z)$. Then, we define $\textsc{Out}^E(M(x,y,z))$, $\textsc{Out}^E(M'(x,y,z))$ as we did in Theorem 21. As before, we say a string x is *easy* if $\textsc{Out}^E(M(x,y,z)) \neq \textsc{Out}^E(M'(x,y,z))$ for some $y,z \in \{0,1\}^{\ell(n)}$.

306 Now, suppose there are infinitely many easy strings. Then, one of the following two NP^E algorithms must be infinite:

307 **Algorithm 1:**

308 On input x, guess $y,z \in \{0,1\}^{\ell(n)}$. If $\textsc{Out}^E(M(x,y,z)) = \textsc{Out}^E(M'(x,y,z))$, reject. Otherwise, $\textsc{Out}^E(M(x,y,z)) \cap \textsc{Out}^E(M'(x,y,z))$ contains a single two-bit string. Accept if the first bit of that string is 1.

309 **Algorithm 2:**

310 On input x, guess $y, z \in \{0,1\}^{\ell(n)}$. If $\text{OUT}^E(M(x,y,z)) = \text{OUT}^E(M'(x,y,z))$, reject. Otherwise, $\text{OUT}^E(M(x,y,z)) \cap \text{OUT}^E(M'(x,y,z))$ contains a single two-bit string. Accept if the first bit of that string is 0.

311 Then, we would have an infinite subset of H or $\overline{H}$ which contradicts the bi-immunity of H. Thus, all strings with length greater than some n_0 must be hard. Therefore, for y, z with length greater than $\ell(n_0)$, $\chi_2^E(y,z)$ is 2-enumerable. Since we can encode $E^{\leq \ell(n_0)}$ in a finite table, $\chi_2^E(y,z)$ is 2-enumerable for all lengths. Finally, since $j \geq 1$ and E is Σ_j^{P}-complete, we have $\mathrm{SAT} \leq_{\mathrm{m}}^{\mathrm{P}} E$. Thus, χ_2^{SAT} is also 2-enumerable and we have P = NP [Bei91] (or we can use Procedure Prune directly). $\square$

6 Open Problems

312 In this paper, we have combined several proof techniques from bounded query complexity — namely mind changes, tree pruning and the hard/easy argument. These techniques do have their limitations, however. For example, the hard/easy argument was used to show that for all $f(n) \in O(n^\epsilon)$ for some $\epsilon < 1$, $\mathrm{P}^{\mathrm{SAT}_{f(n)\text{-tt}}} = \mathrm{P}^{\mathrm{SAT}_{(f(n)+1)\text{-tt}}}$ implies that PH collapses [Kad88, Wag88]. This hard/easy argument does not generalize to $f(n) = O(n)$ or higher. For essentially the same reasons, we are not able to generalize Theorem 17 for $r(n)$ and $s(n)$ beyond $O(n^\epsilon)$ and Theorem 20 for $r(n)$ beyond $\epsilon \log n$. Improvements to these theorems, we believe, would require significant advances in the state of the art of these proof techniques. As of this writing, we are not aware of any oracle relativizations where the relativized versions of $\mathrm{P}^{\mathrm{SAT}_{(n+1)\text{-tt}}}$ and $\mathrm{P}^{\mathrm{SAT}_{n\text{-tt}}}$ are equal but the relativized PH has infinitely many distinct levels. Such an oracle might exist (e.g., by combining the results of Yao [Yao85] and of Cai *et al.* [CGH⁺88]). However, the implication of its existence on the limits of the hard/easy argument is unclear.

Acknowledgments

313 The authors would like to thank Bill Gasarch for proofreading this paper and for many fruitful discussions.

References

314 [AA96] M. Agrawal and V. Arvind. Quasi-linear truth-table reductions to p-selective sets. *Theoretical Computer Science*, 158(1&2):361–370, 1996.

315 [ABG90] A. Amir, R. Beigel, and W. I. Gasarch. Some connections between bounded query classes and non-uniform complexity. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 232–243, 1990.

316 [ABT96] M. Agrawal, R. Beigel, and T. Thierauf. Modulo information from nonadaptive queries to NP. In *Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *Lecture Notes in Computer Science*, pages 322–334. Springer-Verlag, 1996.

317 [BCO93] R. Beigel, R. Chang, and M. Ogiwara. A relationship between difference hierarchies and relativized polynomial hierarchies. *Mathematical Systems Theory*, 26(3):293–310, July 1993.

318 [Bei91] R. Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, July 1991.

319 [BF99] H. Buhrman and L. Fortnow. Two queries. *Journal of Computer and System Sciences*, 59(2):182–194, 1999.

320 [BKS95] R. Beigel, M. Kummer, and F. Stephan. Approximable sets. *Information and Computation*, 120(2):304–314, 1995.

321 [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, December 1988.

322 [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, February 1989.

323 [Cha97] R. Chang. Bounded queries, approximations and the boolean hierarchy. Technical Report TR 97-035, Electronic Colloquium on Computational Complexity, 1997. To appear in *Information and Computation*.

324 [CK95] R. Chang and J. Kadin. On computing Boolean connectives of characteristic functions. *Mathematical Systems Theory*, 28(3):173–198, May/June 1995.

325 [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.

326 [HHH98] E. Hemaspaandra, L. A. Hemaspaandra, and H. Hempel. Query order in the polynomial hierarchy. *Journal of Universal Computer Science*, 4(6):574–588, June 1998.

327 [HHH99] E. Hemaspaandra, L. A. Hemaspaandra, and H. Hempel. Downward collapse within the polynomial hierarchy. *SIAM Journal on Computing*, 28(2):383–393, April 1999.

328 [HHW95] L. A. Hemaspaandra, H. Hempel, and G. Wechsung. Query order. *SIAM Journal on Computing*, 28(2):637–651, April 1995.

329 [HN93] A. Hoene and A. Nickelsen. Counting, selecting, sorting by query-bounded machines. In *Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science*, volume 665 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

330 [Kad88] J. Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, December 1988.

331 [McN] T. McNicholl. Some results on commutative oracles. To appear in *Journal of Symbolic Logic*.

332 [Ogi95] M. Ogihara. Polynomial-time membership comparable sets. *SIAM Journal on Computing*, 24(5):1168–1181, 1995.

333 [PY82] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). In *ACM Symposium on Theory of Computing*, pages 255–260, 1982.

334 [Wag88] K. Wagner. Bounded query computations. In *Proceedings of the 3rd Structure in Complexity Theory Conference*, pages 260–277, June 1988.

335 [Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19:833–846, 1990.

336 [Yao85] A. C. Yao. Separating the polynomial-time hierarchy by oracles. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.

337 [Yap83] C. Yap. Some consequences of non-uniform conditions on uniform classes. *Theoretical Computer Science*, 26(3):287–300, 1983.