JOURNAL OF COMPUTER AND SYSTEM SCIENCES 39, 299–322 (1989)

The Strong Exponential Hierarchy Collapses*

LANE A. HEMACHANDRA†

Department of Computer Science,
Cornell University, Ithaca, New York 14853

Received April 20, 1988; revised October 15, 1988

Composed of the levels E (i.e., $\bigcup_c \mathrm{DTIME}[2^{cn}]$), NE, P^{NE}, $\mathrm{NP}^{\mathrm{NE}}$, etc., the strong exponential hierarchy is an exponential-time analogue of the polynomial-time hierarchy. This paper shows that the strong exponential hierarchy collapses to P^{NE}, its Δ_2 level.

$$\mathrm{E} \neq \mathrm{P}^{\mathrm{NE}} = \mathrm{NP}^{\mathrm{NE}} \cup \mathrm{NP}^{\mathrm{NP}^{\mathrm{NE}}} \cup \cdots.$$

The proof stresses the use of partial census information and the exploitation of nondeterminism. Extending these techniques, we derive new quantitative relativization results: if the weak exponential hierarchy's Δ_{j+1} and Σ_{j+1} levels, respectively $\mathrm{E}^{\Sigma_j^p}$ and $\mathrm{NE}^{\Sigma_j^p}$, do separate, this is due to the large number of queries NE makes to its Σ_j^p database. Our techniques provide a successful method of proving the collapse of certain complexity classes. © 1989 Academic Press, Inc.

1. INTRODUCTION

1.1. *Background*

During 1986 and 1987, many structurally defined complexity hierarchies collapsed. The proofs of collapse shared a technique, the use of census functions, and shared a background, the previous work on quantitative relativizations of Mahaney [Mah82], Book, Long, and Selman [BLS84], and Long [Lon85]. These three papers use the idea that if you know the strings in a set, you implicitly know the strings in the complement. Thus, Mahaney shows that:

THEOREM 1.1 [Mah82]. *If* NP *has a sparse Turing complete set* (*i.e., if there is a sparse set* $S \in \mathrm{NP}$ *such that* $\mathrm{NP} \subseteq \mathrm{P}^S$), *then the polynomial hierarchy collapses to* P^{NP},

by showing that for any polynomial $p(\ )$, a P^{NP} machine can, on input x, explicitly find all strings of length at most $p(|x|)$ that are in S. Once the P^{NP} machine has

* Presented at the "Second Annual IEEE Computer Society Symposium on Structure in Complexity Theory," held June 16–19, 1987, at Cornell University, Ithaca, New York.

† Supported in part by a Fannie and John Hertz Foundation Fellowship, NSF Research Grants DCR-8520597 and CCR-8809174, and a Hewlett Packard Corporation equipment grant. Preliminary versions of these results were presented at the Second Annual Structure in Complexity Theory Conference [Hem87c] and the Nineteenth Annual ACM Symposium on Theory of Computing [Hem87b]. Present address: Department of Computer Science, University of Rochester, Rochester, NY 14627.

0022-0000/89 $3.00

computed these strings, it can easily solve NP^{NP} problems since $NP \subseteq P^S$. The papers of Book, Long, and Selman [BLS84] and Long [Lon85] also exploit this notion of finding the strings in an oracle set and using those strings to prove collapses of quantitatively restricted relativized complexity classes.

The work described in this paper develops, as an extension of the ideas of Mahaney [Mah82], the following paradigm. We find not the *names* of strings in the oracle set, but the *number* of strings, the census function, of the oracle set. We prove that $P^{NE} = NP^{NE}$ by showing that it is easy to compute the number of strings in the NE oracle of NP^{NE} that are queried at each level of the NP machine's computation tree.

This technique of census functions [Hem86b, Hem87b] is used in many of the hierarchy collapsing proofs. Census functions are used by Kadin [Kad87] to improve Mahaney's result (Theorem 1.1), by Toda to collapse the linear-space alternation hierarchy [Tod87], and by Lange, Jenner, and Kirsig [LJK87] to collapse the logspace alternation hierarchy. More recently, Wagner and Schöning extended the direct census techniques used in our Theorem 4.10 and in [Kad87] to prove the collapse of the logspace oracle hierarchy and provide a lovely extension to our Lemma 3.1. Immerman has since showed that NSPACE$[s(n)]$ = co-NSPACE$[s(n)]$ for space constructible $s(n) = \Omega(\log n)$, which strengthens the logspace and linear-space results of [Tod87, LJK87, SW88].

Many of these results have been reproved using alternate techniques. Beigel proves collapses via the theory of mind changes [Bei87]. Kilian and Maley [Kil87] and Book, Toda, and Watanabe [Wat87] have also studied simplification and generalization of hierarchy collapses.

1.2. *Overview*

We wish to know if the high Δ and Σ levels of complexity hierarchies are computationally complex, and if so, *why* they are complex.

Section 3 looks at the strong exponential hierarchy:

$$E \cup NE \cup NP^{NE} \cup NP^{NP^{NE}} \cup \cdots.$$

We show that the strong exponential hierarchy collapses to its Δ_2 level, P^{NE}. Our proof is based on a careful inspection of the computation tree involved in an NP^{NE} computation. We show how P^{NE} can construct increasingly accurate partial census information about the number of "yes" responses NE makes to queries from NP in the action of NP^{NE}. Finally, we have the correct census and collapse the classes.

The main result of Section 3 is:

LEMMA 3.1. $P^{NE} = NP^{NE}$.

It follows that

$$E \neq P^{NE} = NP^{NE} \cup NP^{NP^{NE}} \cup \cdots.$$

Section 3.1.2 shows that this result does not follow simply from the fact that NE is a hard set. The section also notes that the combinatorics involved prevents this technique from collapsing the polynomial hierarchy [Sto77] to P^{NP}.

Section 4 uses the census techniques of Section 3 to prove new results on quantitative relativization—relativization with restrictions placed on oracle access. We first review the work on quantitative relativization of Book, Long, and Selman [BLS84, Lon85]. Then we show how our method of computing *partial census functions*, instead of the *names of strings* used in previous work, collapses complexity classes and unifies previous results.

For the main result of Section 4, we study the weak exponential hierarchy, which is NE given the polynomial hierarchy as a database:

$$\text{weak exponential hierarchy} = \mathrm{NE} \cup \mathrm{NE}^{\mathrm{NP}} \cup \mathrm{NE}^{\mathrm{NP}^{\mathrm{NP}}} \cup \cdots.$$

We show that if the weak exponential hierarchy's Δ_i and Σ_i levels do separate, this is due not to the power of the database but to the large number of queries NE makes to the database.

COROLLARY 4.4, Part 3. $\mathrm{E}^{\Sigma_k^p} = \{L \mid L \in \mathrm{NE}^{\Sigma_k^p}$ *and some* $\mathrm{NE}^{\Sigma_k^p}$ *machine accepting* L, *for some* c *and every* x, *queries its* Σ_k^p *oracle at most* $2^{c|x|}$ *times in its entire computation tree on input* $x\}$, *where* Σ_j^p *is the* jth *level of the polynomial hierarchy* [Sto77].

We also show that if $\mathrm{EXP}^{\mathrm{NP}}$ is to dominate P^{NE}, it *must* use the answers to early queries to help it pose later ones, as the non-adaptive version of $\mathrm{EXP}^{\mathrm{NP}}$ *is* contained in P^{NE}.

THEOREM 4.10, Part 2. $\{S \mid S \leqslant^{exp}_{\text{truth-table}} \mathrm{NP}\} = \mathrm{P}^{\mathrm{NE}}$.

Finally, Section 5 lists open problems and summarizes the implications of our results.

Thus, we prove that the high levels of the strong exponential hierarchy are no harder than the low levels and that high levels of the weak exponential hierarchy separate completely only if NE floods its database with queries.

2. EXPONENTIAL HIERARCHIES

2.1. *Definition of the Strong Exponential Hierarchy*

Both $\bigcup_c \mathrm{DTIME}[2^{cn}]$ and $\bigcup_k \mathrm{DTIME}[2^{n^k}]$ are commonly referred to as exponential time (compare [CT86] with [BH77]), though the former is more common in the literature of structural complexity [Sel86, HY84]. We always make clear which exponential time we are speaking of. Our main result—the strong exponential hierarchy collapses—holds under either definition.

DEFINITION 2.1.[1]

$$\Sigma_0^{SE} = E = \bigcup_c DTIME[2^{cn}]$$

$$\Sigma_1^{SE} = NE = \bigcup_c NTIME[2^{cn}]$$

$$\Sigma_{k+1}^{SE} = NP^{\Sigma_k^{SE}}, \quad \text{for } k \geq 1$$

$$\Delta_{k+1}^{SE} = P^{\Sigma_k^{SE}}, \quad \text{for } k \geq 1$$

$$SEH = \text{strong exponential hierarchy} = \bigcup_{i \geq 0} \Sigma_i^{SE}.$$

Similarly, $EXP = \bigcup_k DTIME[2^{n^k}]$, $NEXP = \bigcup_k NTIME[2^{n^k}]$, and $SEXPH = EXP \cup NEXP \cup NP^{NEXP} \cup NP^{NP^{NEXP}} \cup \cdots$.

2.2. *Definition of the Weak Exponential Hierarchy*

The weak exponential hierarchy, which was studied by Hartmanis, Immerman, and Sewelson, is defined as follows.

DEFINITION 2.2. 1. The weak exponential hierarchy, EH [HIS85], is $NE \cup NE^{NP} \cup NE^{NP^{NP}} \cup \cdots$.

2. The weak EXP hierarchy, EXPH, is $NEXP \cup NEXP^{NP} \cup NEXP^{NP^{NP}} \cup \cdots$.

The weak exponential hierarchy, EH, has characterizations in terms of the alternating Turing machines of Chandra, Kozen, and Stockmeyer [CKS81] and in terms of quantified formulas. EH is exactly the languages L of the form that for some k and c:

$$L = \{x \mid (\exists_E y_1)(\forall_E y_2) \cdots (Q_{k,E} y_k)[R(x, y_1, ..., y_k)]\},$$

where R is a polynomial-time predicate and an "E" subscript denotes a 2^{cn} bound on the quantifier size (e.g., "$(\exists_E y)[$" is short for "$(\exists y)[|y| \leq 2^{cn} \wedge$") [HIS85]. In terms of alternating Turing machines, EH models alternating Turing machines with

[1] It might seem natural to define an exponential hierarchy as $E \cup NE \cup NE^{NE} \cup NE^{NE^{NE}} \cup \cdots$. However, by asking long queries to their oracles, these machines can unnaturally boost their power. For example, NE^{NE} contains double exponential time and $NE^{NE^{NE}}$ contains triple exponential time. Thus we can obtain trivial separations in this hierarchy by using the Hartmanis–Stearns time hierarchy theorem [HS65]. Even worse, an exponential hierarchy defined this way would not even be contained in EXPSPACE. What causes this strange behavior is that a polynomial composed with a polynomial yields a polynomial, but an exponential function composed with an exponential function does not yield an exponential function. To avoid these anomalous behaviors, exponential hierarchies are defined by composing a single exponential function with many polynomial functions.

a bounded number of 2^{cn}-sized alternation blocks. Similarly, EXPH models alternating Turing machines with a bounded number of 2^{n^k}-sized alternation blocks.

2.3. *Properties of the Strong and Weak Exponential Hierarchies*

2.3.1. *The Strong Exponential Hierarchy and Sensitivity to Padding*

SEH is "strong" because in a relativized world A, SEH^A is not contained in EH^A. This is simply because from its Δ_2^A level ($\mathrm{P}^{\mathrm{NE}^A}$) on up, SEH^A can query strings in A of length 2^{n^k}, but EH^A can only query strings of length 2^{cn}. To understand this, just reflect on the fact that

$$\mathrm{P}^{\mathrm{E}} \supsetneq \mathrm{E}^{\mathrm{P}},$$

since $\mathrm{E}^{\mathrm{P}} = \mathrm{E}$ but $\mathrm{P}^{\mathrm{E}} = \mathrm{EXP}$, and $\mathrm{EXP} \supsetneq \mathrm{E}$ by the time hierarchy theorem of Hartmanis and Sterns [HS65].

THEOREM 2.3. *There is a relativized world A so that* $\mathrm{SEH}^A - \mathrm{EH}^A \neq \varnothing$. *Indeed, in this world* $\mathrm{P}^{\mathrm{NE}^A} - \mathrm{EH}^A \neq \varnothing$.

Proof. We make $L_A \in \mathrm{P}^{\mathrm{NE}^A} - \mathrm{EH}^A$, where

$$L_A = \{0^n \mid (\exists y)[y \in A \wedge |y| = 2^{n^2}]\}.$$

L_A is clearly in $\mathrm{P}^{\mathrm{NE}^A}$, but since no EH^A machine can reach strings of length 2^{n^2} on inputs of length n, we can easily diagonalize against each EH machine. ∎

We get the following trivial separation simply from the padding anomaly that $\mathrm{E} \neq \mathrm{EXP} \subseteq \mathrm{P}^{\mathrm{NE}}$.

FACT 2.4. $\Sigma_0^{\mathrm{SE}} \neq \Delta_2^{\mathrm{SE}}$.

This sensitivity of E and NE oracles to polynomials padding of their input strings has another consequence. The hierarchy SEXPH (see text following Definition 2.1) equals SEH. Why? Clearly $\mathrm{P}^{\mathrm{NE}} = \mathrm{P}^{\mathrm{NEXP}}$, $\mathrm{NP}^{\mathrm{NEXP}} = \mathrm{NP}^{\mathrm{NE}}$, and so forth, since if P (in P^{NE}) just adds polynomial padding to each query string, its NE oracle can stimulate the NEXP calls of $\mathrm{P}^{\mathrm{NEXP}}$. This extends the collapsing result of Section 3.

COROLLARY 3.3. $\mathrm{E} \neq \mathrm{P}^{\mathrm{NE}} = \mathrm{SEH} = \mathrm{P}^{\mathrm{NEXP}} = \mathrm{SEXPH}$.

2.3.2. *Downward Separations*

If we collapse the polynomial hierarchy at any level, the entire hierarchy collapses to that level; $\Sigma_i^P = \Pi_i^P \Rightarrow \Sigma_i^P = \mathrm{PH}$ [Sto77]. This is known as *downward separation.* Many other complexity hierarchies, such as the boolean hierarchy over NP [CGH*88, CGH*] and the random polynomial time hierarchy [Zac86], exhibit this behavior. One troubling feature of the weak exponential hierarchy is that it

does not have downward separation. Hartmanis, Immerman, and Sewelson [HIS85] display a relativized world A where $E^A = NE^A \neq NE^{NP^A}$.

On the other hand, the strong exponential hierarchy *does* have downward separation of a sort. A subtlety is that we must account for the sensitivity to padding discussed in the previous section.

THEOREM 2.5 (Downward separation). 1. $E = NE \Rightarrow EXP = SEH$.

2. $NE = coNE \Rightarrow NEXP = SEH$.
3. $EXP = NEXP \Rightarrow EXP = SEXPH$.
4. $NEXP = coNEXP \Rightarrow NEXP = SEXPH$.

Proof of Theorem 2.5. Part 1. Using Theorem 3.2 and our assumption that $E = NE$, $EXP \subseteq SEH = P^{NE} = P^E = EXP$. Part 2. When $NE = coNE$ (and thus even "no" answers to NE queries have certificates), P^{NE} can be simulated by NEXP, which just guesses the correct oracle answers along with their certificates of correctness. Parts 3 and 4. Similar to the proofs of parts 1 and 2, respectively.

Note that the theorem can be proved directly, without an appeal to Theorem 3.2 [Hem87a, Theorem 3.9]. ∎

There is no point in stating more general downward separation results. Since we already know that $P^{NE} = SEH$ with no assumptions needed, the results above are the only nontrivial downward separations possible in the strong exponential hierarchy.

3. On the Strong Exponential Hierarchy

3.1. *The Strong Exponential Hierarchy Collapses*

3.1.1. *Proof of Collapse*

This section proves that the strong exponential hierarchy collapses to its P^{NE} level. It suffices to collapse the strong exponential hierarchy's P^{NE} and NP^{NE} levels. Then downward separation gives us a quick proof of the hierarchy's collapse.

LEMMA 3.1. $P^{NE} = NP^{NE}$.

THEOREM 3.2. $P^{NE} = SEH$.

COROLLARY 3.3. $E \neq P^{NE} = SEH = P^{NEXP} = SEXPH$.

Proof of Theorem 3.2. By Lemma 3.1, $\Delta_2^{SE} = \Sigma_2^{SE}$. Inductively assume (for some $k \geq 2$) that $\Delta_2^{SE} = \Sigma_k^{SE}$. Now

$$\Sigma_{k+1}^{SE} = NP^{\Sigma_k^{SE}} = NP^{\Delta_2^{SE}} = NP^{P^{NE}} = NP^{NE}.$$

The last equality holds because there is an NP machine that takes over the job of the P machine (in $NP^{P^{NE}}$) and does all the NE queries itself. Thus, by induction, $\Sigma_k^{SE} = \Delta_2^{SE}$ for all k, so $SEH = P^{NE}$. ∎

Corollary 3.3 is proven in Section 2.3.1.

The work of our result lies in the proof of Lemma 3.1. We make extensive use of the power of NE to guess query strings, witnesses, and paths in trees. We now sketch the proof of Lemma 3.1; a formal proof follows.

First, we wish to place graphically in mind our image of an NP^{NE} computation. An NP computation tree has branches for each nondeterministic guess made by the NP machine. The machine is said to accept if any branch accepts [HU79]. For example, Fig. 1 shows an NP machine checking the satisfiability of the formula $x_1 \wedge \overline{x_2}$. The NP machine has nondeterministically guessed all possible assignments and has found one that satisfies the formula. Throughout this section we assume, for simplicity of presentation, that our nondeterministic machines have at most two successor states for any given state.

We view an NP^{NE} computation similarly, except the NP machine can pose queries to an NE oracle. Each of the nondeterministic paths may, of course, pose different queries than its brothers do (Fig. 2). We label the depth of the nodes in computation trees in the standard way (Fig. 2).

Now we describe our strategy. Figure 3 shows the computation tree of an NP^{NE} machine. Our goal is to accept, with a P^{NE} machine, the same language the NP^{NE} machine accepts. The P^{NE} machine computes *the number of query strings receiving yes answers from NE at each depth of the tree* (we will refer to these as "yes strings"). For example, there are two yes strings at depth two in Fig. 3.

We do not simply jump in and try to compute the number of yes answers deep in the tree. To know which strings are even queried deep in the tree, we must first know the answers to queries more shallow in the tree. Thus we first find the number of yes responses in the first level of the tree. Then using this knowledge, we find the number of yes responses at the second level, and so on. At each level we use

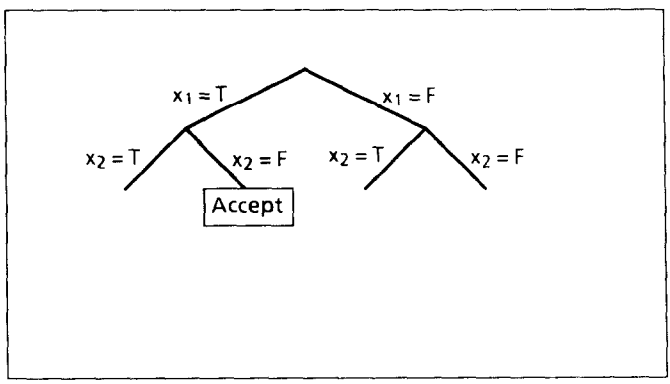


FIG. 1. Nondeterministic computation tree.

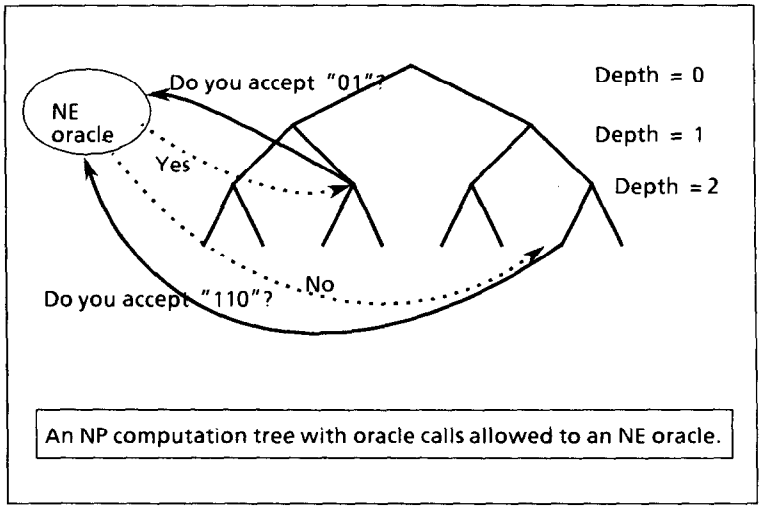


FIG. 2. NP^{NE} computation tree.

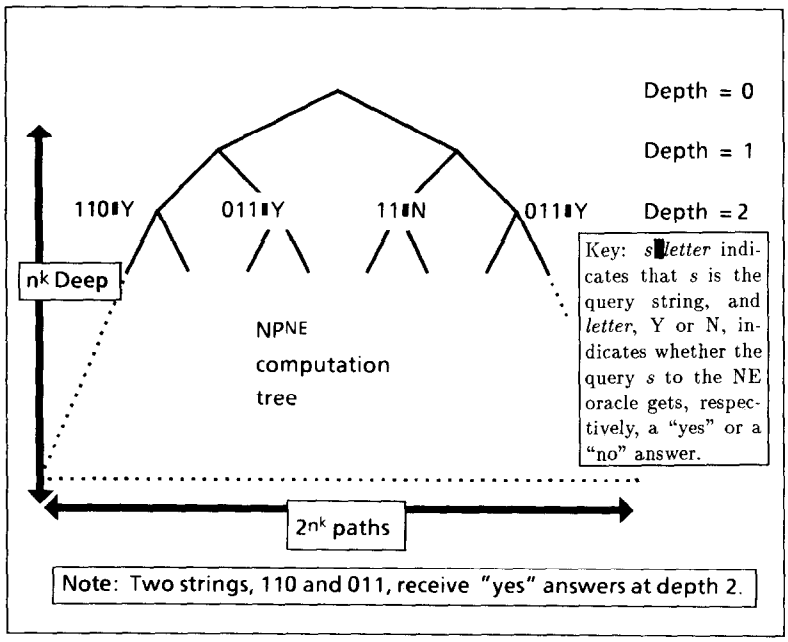


FIG. 3. Our strategy.

knowledge of the previous levels to help us do a binary search for the number of strings at the current level.

For concreteness, suppose our NP^{NE} language is accepted by NP machine N_{17} with NE machine NE_{21} as its oracle.[2] At a typical stage we know, for example, that the computation tree for $N_{17}^{NE_{21}}(x)$ has exactly 1, 1, 0, 4, and 11 yes answers (queries of strings accepted by NE_{21}) at levels 0, 1, 2, 3, 4, respectively, and between 8 and 16 at level 5. Our question (asked by P to NE) is:

> given x and assuming 1, 1, 0, 4, 11 are the correct number of yes answers at levels 0, 1, 2, 3, 4, are there at least 12 yes strings queried at level 5? (**)

This is the kind of question that an NE oracle can answer. The NE computation guesses the first six levels of N_{17}'s computation tree, checks that the tree and queries written are really the actions that N_{17} would take given the query answers guessed, checks that there are 1, 1, 0, 4, 11 alleged yes strings at levels 0, 1, 2, 3, 4, and $\geqslant 12$ alleged yes strings at level 5, and guesses the proofs that these strings really are yes stings (i.e., are accepted by NE_{21}).

If 1, 1, 0, 4, 11 is correct, then the first five levels correspond to the first five levels of the actual computation tree of $N_{17}^{NE_{21}}(x)$, and if there are $\geqslant 12$ yes strings at level five we will have guessed them.

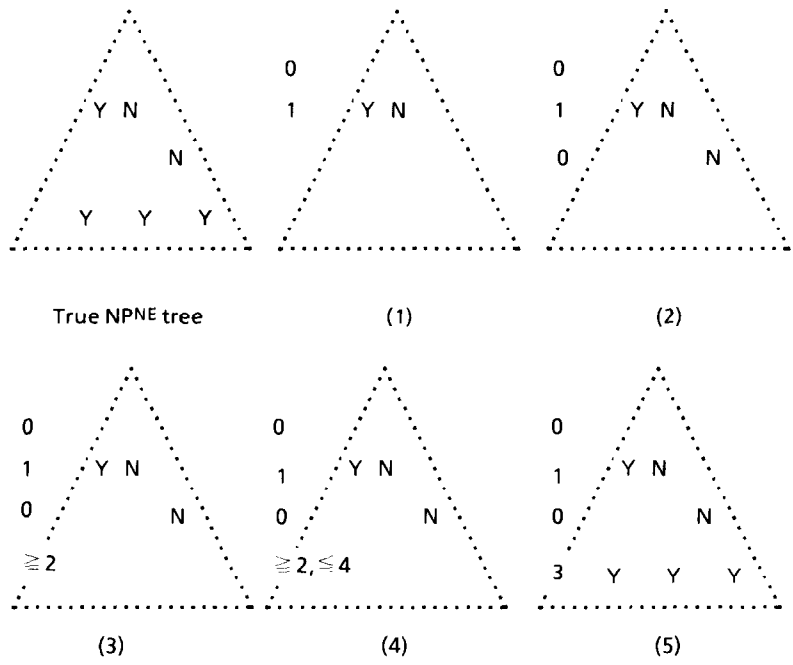
Eventually we know the number of yes answers at each depth, and a final call of P to its NE oracle lets NE guess and check the correct computation tree of $N_{17}^{NE_{21}}(x)$, and search for an accepting path in this tree.

Crucially, we do not need to verify that the no strings deserve "no" responses. We know the total number of yes answers at each level and have proofs that strings do indeed get yes answers. Thus, in a guessed tree that gets all the desired yes proofs, our no strings must indeed deserve "no" answers, as we have ensured that our tree agrees with the action of N_{17}. Put another way, our NE oracle, for queries of the form of (**) above, gives correct answers.

Let us refer to the P and NE machines we use to simulate NP^{NE} as P_* and NE_*, i.e., we ensure that $L(P_*^{NE_*}) = L(N_{17}^{NE_{21}})$. Figure 4 shows how the trees NE_* guesses increase in the (literal) depth of their accuracy at reflecting the tree of $N_{17}^{NE_{21}}(x)$. Crucially, P_* learns only the *number* of yes answers at each depth of $N_{17}^{NE_{21}}(x)$'s tree. This is fortunate; there is no way that P_* could remember all the yes *names*, since deep in the $N_{17}^{NE_{21}}$ tree there may be as many as $2^{n^{17}}$ yes strings on a single level. Our use of increasingly accurate censuses of the number of queries per level is central to the succes of our result.

We now make precise the proof just sketched.

[2] Throughout this paper, we will use the notation M^N, where M and N are machines, as a shorthand for $M^{L(N)}$. $\{N_i\}$ will be a standard enumeration of NP machines, such that N_i runs in $NTIME[n^i]$. Similarly, $\{NE_i\}$ will be a standard enumeration of NE machines, such that NE_i runs in $NTIME[2^{in}]$.



The true NP^{NE} tree and five snapshots showing a few of the increasingly correct images of the tree and its partial census information created in NE's mind. The numbers denote the number of queries on that level receiving a "yes" answer from the NE oracle.

FIG. 4. The true NP^{NE} tree and five snapshots.

Proof of Lemma 3.1 ($P^{NE} = NP^{NE}$). Let L be an arbitrary language in NP^{NE}. For concreteness, suppose $L = L(N_{17}^{NE_{21}})$, where N_{17} is an NP machine running in $NTIME[n^{17}]$ and NE_{21} is an NE machine. We describe machines P_* and NE_*, respectively, P and NE machines, so that $L = L(P_*^{NE_*})$. Thus $P^{NE} = NP^{NE}$.

Let us first describe NE_*,

$L(NE_*) = \{final \# x \# 1^{|x|^{18}} \# c_0 \# c_1 \# c_2 \# \cdots \# c_{l-1} \# c_l |$. There exist sets $C_0, C_1, ..., C_{l-1}, C_l$ of strings so:

1. $|C_i| = c_i$ and $\bigcup_i C_i \subseteq L(NE_{21})$,
2. if one simulates $N_{17}^{\{\}}(x)$, answering each oracle query q at depth i with a yes if and only if $q \in C_i$, then each y in C_i is really queried at level i in this simulation, and
3. if *final* (which can be any string of bits) is 1, there is an accepting path in the simulation mentioned in 2 above.$\}$

Note that $L(\mathrm{NE}_*)$ *is* in NE. The *value* of each c_i is at most $2^{|x|^{17}}$, since this is the maximum width of the computation tree $\mathrm{N}_{17}^{\mathrm{NE}_{21}}(x)$. So by guessing (at most) $|x|^{17} \cdot 2^{|x|^{17}}$ strings and then guessing proofs that each is in $L(\mathrm{NE}_{21})$ and guessing the paths by which each occurs in the simulation, NE_* can easily be implemented in $\mathrm{NTIME}[2^{|x|^{18}}]$. Note that we have padded, so our input size to NE_* is greater than $|x|^{18}$, so NE_* runs in $\mathrm{NTIME}[2^{c \cdot \mathrm{Inputsize}}]$, and thus is in NE. Thus $L(\mathrm{NE}_*) \in \mathrm{NE}$.

Given that the P^{NE} simulator knows the number of yes answers that appear in the first k levels of the computation tree of $\mathrm{N}_{17}^{\mathrm{NE}_{21}}(x)$, it can use binary search on NE_* to find the number of yes answers that appear at level $k+1$.

Now we describe the action of our machine P_* (on input x). P_* uses NE_* to find the correct number of yes strings at each level of $\mathrm{N}_{17}^{\mathrm{NE}_{21}}(x)$.

Stage i. Inductively, P_* has numbers $c_0, c_1, \ldots, c_{i-1}$, so $c_0, c_1, \ldots, c_{i-1}$ are the correct number of yes strings at levels $0, 1, \ldots, i-1$ of the actual computation tree of $\mathrm{N}_{17}^{\mathrm{NE}_{21}}(x)$ (Fig. 2). During this stage P_* finds c_i, the actual number of yes strings at level i. This is easy—P_* just performs binary search (varying z) using calls to NE_* of the form

$$\{0 \# x \# 1^{|x|^{18}} \# c_0 \# \cdots \# c_{i-1} \# z\}$$

to find the value of c_i.

Recall that, when $c_0, c_1, \ldots, c_{i-1}$ are correct, NE_* says yes if z is a lower bound for the number of yes answers at level i. Table I gives a sample binary search run. At the end of it, P_* has learned that there are exactly three yes strings at level 3.

It is crucial to notice that when $c_0, \ldots, c_{i-1}$ are correct, the c_i found is correct. This is because some branch B of NE_* will guess the true yes strings $C_0, \ldots, C_{i-1}$. The queries asked at level i depend only on the fact that the right oracle replies at levels 0 through $i-1$ are known; thus the queries asked at level i on branch B will be the queries that are asked in the tree of $\mathrm{N}_{17}^{\mathrm{NE}_{21}}(x)$ at level i. Thus if there are at least z yes strings queried at level i of the tree of $\mathrm{N}_{17}^{\mathrm{NE}_{21}}(x)$, an extension of B will guess them and accept.

On the other hand, any branch that does not guess the sets $C_0, \ldots, C_{i-1}$ correctly (it guesses, say, $C_0', \ldots, C_{i-1}'$) will certainly not accept. At the first level it errs from

TABLE I

Binary Search over Calls to NE_* Discovers that There Are Three Yes Strings at Level 3 of the Computation Tree of $\mathrm{N}_{17}^{\mathrm{NE}_{21}}(11101)$

Query	NE_*'s answer
$\{0 \# 11101 \# 1^{5^{18}} \# 0 \# 0 \# 1 \# 1\}$	Accept
$\{0 \# 11101 \# 1^{5^{18}} \# 0 \# 0 \# 1 \# 2\}$	Accept
$\{0 \# 11101 \# 1^{5^{18}} \# 0 \# 0 \# 1 \# 4\}$	Reject
$\{0 \# 11101 \# 1^{5^{18}} \# 0 \# 0 \# 1 \# 3\}$	Accept

the true set of C_i's (say level m), it will not be able to find all the strings of its incorrect C'_m in the simulation tree. Why? Since $C'_m \neq C_m$, yet $|C'_m| = |C_m| = c_m$, some string w in C'_m is not a yes string of $N_{17}^{NE_{21}}(x)$ at level m. Since the C'_i are correct at levels $0, ..., m-1$, the strings queried at level m in the simulation are exactly those queried at level m in the tree of $N_{17}^{NE_{21}}(x)$. So if w is queried at level m, it is not in $L(NE_{21})$ (if it were it would have to be in C_m); thus condition 1 of the definition of $L(NE_*)$ is violated and the branch will not accept. If w is not queried at level m, condition 2 of the definition of $L(NE_*)$ is violated and the branch will not accept. **End of Stage i.**

Since c_i can be at most $2^{|x|^{17}}$ in value (this is as wide as the tree of $N_{17}^{NE_{21}}(x)$ gets), the binary search process at stage i takes at most around $|x|^{17}$ steps, each requiring writing a string of length at most about $|x|^{17} \cdot |x|^{17} + |x|^{18} + |x|$. There are at most $|x|^{17}$ stages, so the total run time of $P_*^{(\cdot)}$ is easily polynomial; it runs in $TIME[n^{4 \cdot 17+2}]$. Returning to the general case, if N_{17} runs in $TIME[n^k]$, then P_* runs in deterministic $TIME[n^{4k+2}]$.

After stage $|x|^{17}$, the correct values $c_0, ..., c_{|x|^{17}}$ are known. At this point, a single call to NE_* suffices. P_* accepts if and only if NE_* accepts $1 \# x \# 1^{|x|^{18}} \# c_0 \# \cdots \# c_{|x|^{17}}$ (thus stating that $N_{17}^{NE_{21}}(x)$ accepts). ∎

3.1.2. *Relationship of Our Collapse to Other Collapses and to the Polynomial Hierarchy*

We have just shown that $P^{NE} = NP^{NE}$, and thus the strong exponential hierarchy collapses to P^{NE}. Is this collapse of the strong exponential hierarchy trivial in the same sense that $P^{PSPACE} = NP^{PSPACE} = PSPACE$ is trivial? PSPACE is so powerful that both P^{PSPACE} and NP^{PSPACE} are equal to PSPACE. Do we similarly have $P^{NE} = NE$? Relativization techniques help us here. There is a relativized world where $P^{NE^A} \supsetneq NE^A$. That this is not a side effect of the ability of P^{NE} to reach length 2^{n^k} strings (compared with NE's reach of 2^{cn}) is shown by Theorem 3.4. Section 2.3.1 discusses this "reach" anomaly in detail.

THEOREM 3.4. *There is a recursive oracle A for which*

$$P^{NE^A} \supsetneq NEXP^A \supsetneq NE^A.$$

Proof Sketch. This is a straightforward diagonalization using the techniques of Baker, Gill, and Solovay [BGS75]. We separate P^{NE^A} from $NEXP^A$ by forcing a $coNE^A$ language out of $NEXP^A$. In particular, we diagonalize so that

$$L_A = \{0^n | (\forall y)[|y| = 2^n \Rightarrow y \notin NEXP^A\}.$$

Interlaced with this, we separate $NEXP^A$ from NE^A simply using the fact that the former class is sensitive to length 2^{n^k} oracle strings. ∎

More generally, the collapse does not occur simply because NE is hard. Even though $P^{NE} = NP^{NE}$, there are many NE-hard sets, A, for which $P^A \neq NP^A$. This

follows from direct diagonalization and is related to Hartmanis's results on the re-relativization of PSPACE [Har85, HH88a]. Specifically, given any NE-complete set B and any set C, $B \oplus C$ will be NE-hard, where $\oplus$ indicates disjoint union; however, by diagonalization, we can construct a set C so that $P^{B \oplus C} \neq NP^{B \oplus C}$.

On the other hand, our techniques do not collapse the polynomial hierarchy $(NP \cup NP^{NP} \cup \cdots)$ to P^{NP}. Suppose we tried to show that $P^{NP} = NP^{NP}$ using the above methods. NP^{NP} may have exponentially many yes replies given to its lower NP machine by the upper one. The P machine *can* record an exponential *count*, but the NP machine sitting over it (in P^{NP}) certainly cannot guess an exponentially large object: the names of the 2^{n^k} yes strings in the tree of NP^{NP}.

Of course, if we change the game so that, in NP^{NP}, few queries are made to the oracle, then the same argument works. This "quantitative relativization" approach is what is done by Book, Long, and Selman in [BLS84], where they show that a hierarchy of quantified relativizations collapses. Section 4 of this paper studies quantified relativizations of the exponential hierarchy, where we will see the partial census technique of this section put to further use.

3.2. *A Comparison of Two Census Techniques*

The proof just given originally appeared in [Hem86b, Hem87c]. The technique is to study NP^{NE} by looking in turn at the levels of the NP computation tree and computing census functions. We call this the *base-machine-census* method. Schöning and Wagner [SW88] have reproved Lemma 3.1 using the *direct census* method introduced in [Hem86b, Theorem 4.10] (which is Theorem 4.10 of this paper) and [Kad86, Theorem 14]. Schöning and Wagner obtain a proof that is quite elegant and that generalizes Lemma 3.1 to base classes with super-polynomial run times but with restrictions on the lengths of the oracle queries posed ([SW88, Corollary 6], [Hem 86b, page 15]).

The direct census technique works as follows. Given an NP^{NE} language $L = L(N_i^{NE_j})$, a P^{NE} simulator trying to determine if $x \in L$ approaches this by first computing directly the number of strings in $L(NE_j)$ of length at most $|x|^i$, by binary search. Since N_i's computation tree can touch at most 2^{n^i} strings, this census information can be computed by a P^{NE} machine. Finally, with the census known, one final call to an NE oracle suffices. This call guesses and checks exactly which strings are the ones of length at most $|x|^i$ that are in L and then simulates $N_i^{NE_j}$.

Both the base-machine-census technique and the direct census technique have uses in which they prove theorems that are beyond the reach of the other. When the base machine has a long time bound but touches few strings of the oracle, e.g., our Theorems 4.3, 4.5, and 4.8, the base-machine-census method yields stronger results than the direct census technique. When the base machine has a long time bound but queries only short strings, the direct census method yields stronger results [SW88].

4. Quantitative Relativization Results

4.1. *Definitions*

Quantitative relativization means relativization in which the power of the base machine to query its oracle is restricted. The literature's terminology uses "quantitative" relativization to refer to restricting the *number* of oracle calls, and "qualitative" relativization to refer to restricting the *form and pattern* of oracle calls [BLS84, BLS85]. Since our restrictions limit the number and pattern of the queries, both terms apply in part. Thus we will be somewhat informal in our use of the terms and will sometimes use "quantitative" or "qualitative" to apply to a bound that is both.

Before studying quantitative relativization, we define our notation.

Definition 4.1. Let A and B be complexity classes. The class $A^{B[f(n)]_X}$ is the class of languages accepted by an oracle Turing machine from a standard enumeration of machines for A with a set from B as its oracle, under the restriction specified by X with bound $f(n)$. The restrictions we will consider include the following types and their combinations:

$[f(n)]_{\text{path}}$: on input of size n, each path of A's computation tree (A may be nondeterministic) queries B about at most $f(n)$ different strings,

$[g(n)]_{\text{tree}}$: the total number of strings B is queried about throughout A's entire computation tree is $g(n)$, and

$[h(n)]_{\text{restriction}}$: there is an $h(n)$ bound on whatever is named by "restriction."

Examples and Notes. 1. $\text{NE}^{\text{NP}} = \bigcup_c \text{NE}^{\text{NP}[2^{cn}]_{\text{tree}}}$. Since the NE computation tree is 2^{cn} deep it has less than $2 \cdot 2^{2^{cn}}$ nodes, so it cannot make more than $2 \cdot 2^{2^{cn}}$ queries (Fig. 5a).

2. $\text{NE}^{\text{NP}[2^{cn}]_{\text{tree}}}$ *does* put some restriction on the querying action of NE (Fig. 5b). Note that $\text{NE}^{\text{NP}[1]_{\text{path}}}$ (Fig. 5c) may query $2^{2^{cn}}$ many strings—one on each of NE's $2^{2^{cn}}$ computation paths (Fig. 5d).

3. In classes such as NEXP^{NP}, NEXP may query NP about strings of length exponential in the input size. Thus it is not obvious and probably not true that NEXP equals NEXP^{NP}, even though $\text{NP} \subseteq \text{NEXP}$.

4. Our quantitative classes count strings—not oracle calls. That is, $[n^2]_{\text{tree}}$ means that for all x we have

$$|\{y \mid y \text{ is queried in the tree on input } x\}| \leqslant |x|^2.$$

For example, an NE computation tree that queries string y on each of its $2^{2^{cn}}$ branches would be charged just "1" in the $[\cdot]_{\text{tree}}$ measure for this whole set of queries—*not* $2^{2^{cn}}$ (Fig. 5d).

Finally, we make it harder to prove our theorems.

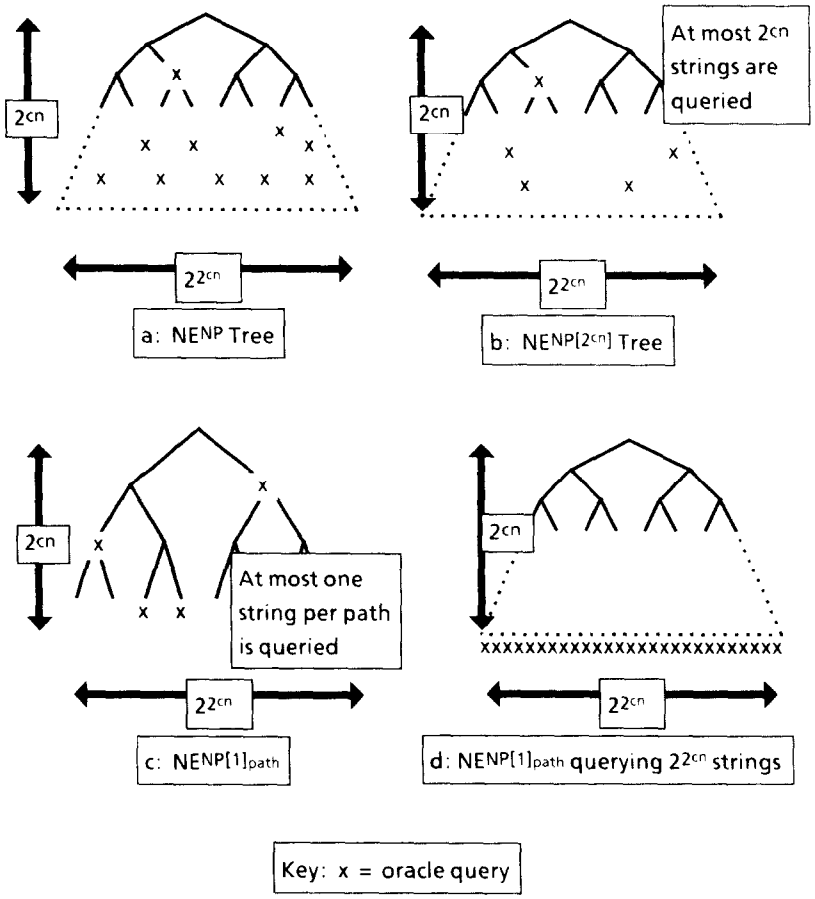


FIG. 5. Quantitative relativizations.

Notation 4.2. Adding a "Y" (e.g., $\mathrm{NE}^{\mathrm{NP}[2^{cn}]_{Y,\mathrm{path}}}$) means we are counting just the number of strings queried that get a "yes" answer from the oracle (and "no" answers do not count). This makes our theorems harder to prove and stronger; any theorem we prove with a "Y" also holds without the "Y." Table II summarizes our notation.

4.2. *Introduction and Background*

The census techniques of Section 3.1 will be used to prove new theorems about quantitative relativizations. We will see, for example, that

$$\mathrm{E}^{\mathrm{NP}} = \mathrm{NE}^{\mathrm{NP}[e]_{\mathrm{tree}}},$$

TABLE II

Nomenclature of Restricted Relativization

Class $= A^{B_{\text{restrictions}}}$	
Restrictions	
$[\text{Qualitative bound}]_{\text{path}}$	Limit on the number of strings queried on each of A's computation paths
$[\text{Qualitative bound}]_{\text{tree}}$	Limit on the number of strings queried throughout A's computation tree
$[\text{Qualitative bound}]_{Y,\text{tree}}$	Limit on the number of strings queried throughout A's computation tree that are in B
Qualitative bounds	
Log	$\bigcup_c \text{TIME}[c \log n]$
Poly	$\bigcup_k \text{TIME}[n^k]$
e	$\bigcup_c \text{TIME}[2^{cn}]$
Exp	$\bigcup_k \text{TIME}[2^{n^k}]$

where "e," from Table II, denotes an exponential bound. Thus the only way NE^{NP} can avoid collapsing to E^{NP} is by using its oracle often. Similarly, we will see that

$$\text{P}^{\text{NE}} \supseteq \text{NE}^{\text{NP}[\text{poly}]_{\text{tree}}}.$$

Note that the P machine here cannot write down even one of the long queries (length 2^{cn}) the NE machine makes to NP, yet P^{NE} can nonetheless simulate the action of $\text{NE}^{\text{NP}[\text{poly}]_{\text{tree}}}$.

This highlights the difference between our techniques and those found in previous work. Previous quantitative relativization results, centered in the polynomial hierarchy, require that the base machine obtain the *names* of all strings queried. However, our base class P can only use a polynomial amount (n^j) of tape. So previous methods did not prove that $\text{P}^{\text{NE}} \supseteq \text{NE}^{\text{NP}[\text{poly}]_{\text{tree}}}$, as each queried name is too long (length 2^{cn}). Previous methods did not prove that $\text{P}^{\text{NE}} = \text{NP}^{\text{NE}}$, as there are too many names (the NP tree has 2^{n^k} nodes).

The body of previous work in quantitative relativization contains many gems [Boo81, SMB83, BLS84, Lon85]. Book, Long, and Selman [BLS84] prove that $\text{P}^{\text{NP}} = \text{NP} \cup \text{NP}^{\text{NP}[\text{poly}]_{\text{tree}}} \cup \text{NP}^{(\text{NP}^{\text{NP}[\text{poly}]_{\text{tree}}})[\text{poly}]_{\text{tree}}} \cup \cdots$, which provides a polynomial analogue of our $\text{P}^{\text{NE}} = \text{NP}^{\text{NE}}$ result. Their paper provides a detailed study of quantitative relativization in a polynomial setting. Recently, Long [Lon85] extended this work by noting that many quantitative relativization results still hold when we restrict our counting to yes queries.

4.3. *Quantitative Relativization Theorems*

This section proves theorems that give insight into what makes hierarchies nontrivial. The first and most important theorem shows that the Σ_{k+1} and Δ_{k+1} levels of EH (Definition 2.2) collapse unless the Σ_{k+1} level, $\mathrm{NE}^{\Sigma_k^p}$, uses its Σ_k^p oracle extensively. This is related to Long's [Lon85, p. 595] results that develop collapses among restricted relativizations of the polynomial hierarchy. One might expect that in the general case the Δ_{k+1} level of the weak exponential hierarchy is strictly contained in the Σ_{k+1} level (see [Sew83, AW88] for some connections between collapses of the delta and sigma levels of exponential hierarchies).

THEOREM 4.3. $\mathrm{E}^{\Sigma_k^p} = \mathrm{NE}^{\Sigma_k^p[\mathrm{e}]_{Y,\mathrm{tree}}}$, *for all* $k > 0$.

COROLLARY 4.4. 1. $\mathrm{E}^{\mathrm{NP}} = \mathrm{NE}^{\mathrm{NP}[\mathrm{e}]_{Y,\mathrm{tree}}}$.

2. $\mathrm{E}^{\mathrm{NP}} = \mathrm{NE}^{\mathrm{NP}[\mathrm{e}]_{\mathrm{tree}}}$.

3. $\mathrm{E}^{\Sigma_k^p} = \mathrm{NE}^{\Sigma_k^p[\mathrm{e}]_{\mathrm{tree}}}$, *for all* $k > 0$.

Proof Sketch of Theorem 4.3. The method of Section 3.1 works with a subtle but important modification. In that section, we could guess a whole computation tree and check it. Here, the computation tree of NE is too huge to do this; it has around $2^{2^{cn}}$ nodes (where 2^{cn} is the nondeterministic of the NE machine). Luckily, since there is a $[\mathrm{e}]_{Y,\mathrm{tree}}$ bound we are interested in checking at most 2^{cn} nodes of this tree. Our new trick is that we just guess and check the parts of the tree that are of interest to us.

So now, our Σ_k^p machine called by E will just (1) guess the yes strings, (2) guess the paths in the computation tree of $\mathrm{NE}^{\Sigma_k^p}$ that lead to them, and (3) verify that the strings really are accepted by NE's oracle. Note that (3) is subtle; Σ_k^p is closed under intersection, so checking if all our guessed strings are accepted by NE's oracle is a Σ_k^p question. We use the same existential block (i.e., the first existential block of E's oracle) that tackles (1) and (2) to start on the first existential block of the membership checking question.

This can all be done in $2^{c'n}$ steps, and thus allows E's Σ_k^p oracle to guess and check all the portions of the huge $\mathrm{NE}^{\Sigma_k^p}$ tree that are of interest to it.

Having noted this, the proof follows the method of Lemma 3.1. Our $\mathrm{E}^{\Sigma_k^p}$ simulator moves, level by level, down the computation tree of the $\mathrm{NE}^{\Sigma_k^p[\mathrm{e}]_{Y,\mathrm{tree}}}$ machine it simulates. At each level, it performs a binary search to determine the exact number of yes answers at that level of the tree. ∎

The following theorem shows that $\mathrm{NE}^{\mathrm{NP}}$ machines must make many oracle calls in order to accept languages outside of P^{NE}. The result is perhaps surprising, as P^{NE} makes only polynomially many nondeterministic queries, but $\mathrm{NE}^{\mathrm{NP}[\mathrm{poly}]_{\mathrm{path}}[\exp]_{\mathrm{tree}}}$ makes exponentially many queries, any one of which may be far too long for P to record.

THEOREM 4.5. $\mathrm{P}^{\mathrm{NE}} \supseteq \mathrm{NE}^{\mathrm{NP}[\mathrm{poly}]_{\mathrm{path}}[\exp]_{Y,\mathrm{tree}}}$.

COROLLARY 4.6. 1. $P^{NE} = NEXP^{NP[poly]_{path}[exp]_{Y,tree}}$.

2. $P^{NE} = NEXP^{NP[poly]_{tree}}$.

Proof Sketch of Theorem 4.5. We need a new trick here. Our problem is that P^{NE} can only store the query census for polynomially many levels. However, though $NE^{NP[poly]_{path}[exp]_{Y,tree}}$ has only polynomially many queries per path, it might have queries at exponentially many levels (Fig. 6).

Our trick is to associate with each query its *query depth*: now many queries are made before it on its path (Fig. 7). P^{NE} will have polynomially many rounds. In each round i it will found, by binary search, the number of yes answers received by queries at query depth i.

Since the query depth of the tree of $NE^{NP[poly]_{path}[exp]_{Y,tree}}$ is polynomial, and since getting the right string to ask for a query depth i query depends only on having queries of query depth less than i correctly answered, the method of Section 3.1 now works.

Our P^{NE} simulator proceeds as follows. First it performs a binary search to find the census of yes answers at query depth one. Then it uses this census and another binary search to find the census of yes answers at query depth two, and so on. Due to the $[poly]_{path}$ restriction in the theorem's statement, every query is of at most polynomial query depth. Thus, after polynomially many rounds of binary search the P machine knows the census of yes answers at each query depth and can finish off its simulation with one final query to an NE oracle. ∎

Proof of Corollary 4.6. These results use the same argument as Theorem 4.5, but since $P^{NE} \subseteq NEXP^{NP[poly]_{tree}} \subseteq NEXP^{NP[poly]_{path}[exp]_{Y,tree}}$, we get equality in this corollary. ∎

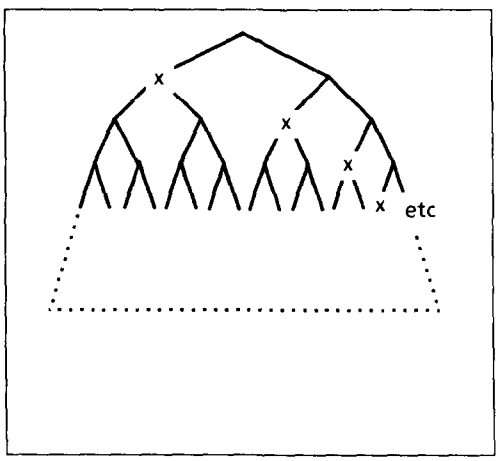


FIG. 6. On query per path, but many levels have queries.

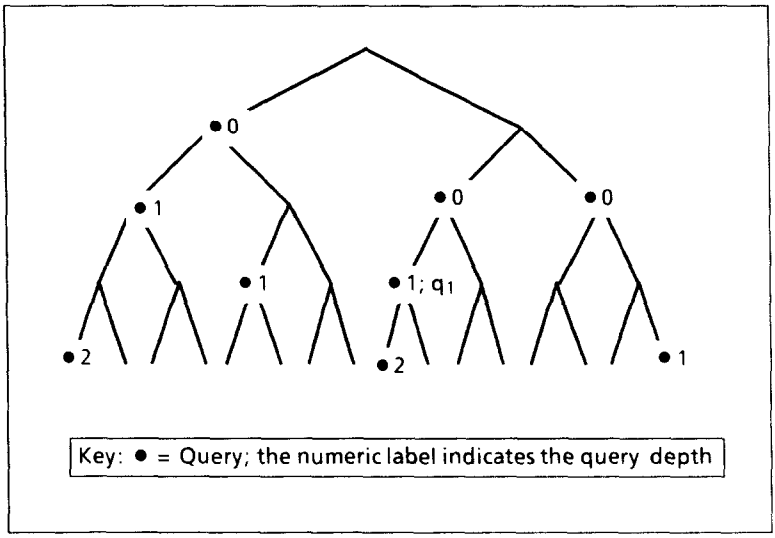


FIG. 7. Query depths.

It is easy to see from Corollary 4.6 that NEXP^{NP} is a delicate class. If you restrict its query access too much, it is weakened to the point of collapsing to P^{NE}.

4.4. *More Quantitative Relativization Theorems*

We can tailor and extend the techniques we have developed to a range of problems. This section presents three such problems and briefly outlines the modifications needed.

4.4.1. *Paying Only for the First Occurrence*

Note that, within a computation tree, a query consists of both a string queried and the location in the tree at which the query is made. The *first occurrence depth* (Fig. 8 gives examples) of an oracle query in a computation tree is 0 if there are no queries made along the path between it and the root. Continuing in a breadth-first left-to-right fashion through the computation tree, an oracle query is of *first occurrence depth* $i+1$ if (1) the string associated with the query has not already been assigned a query depth less that $i+1$ and (2) the largest query depth of a query along the path from the query to the root is i (Fig. 8). Intuitively, the first occurrence depth of a query string is how many times we must expand from the root our "frontier" of queries to reach the query string.

The following theorem says that once a query is paid for by a nondeterministic branch, its brother branches get to query the same string for free. The proof of Theorem 4.7 is essentially the same as that of Theorem 4.5. The P^{NEXP} simulator uses binary search to find the census of yes answers to queries at first occurrence

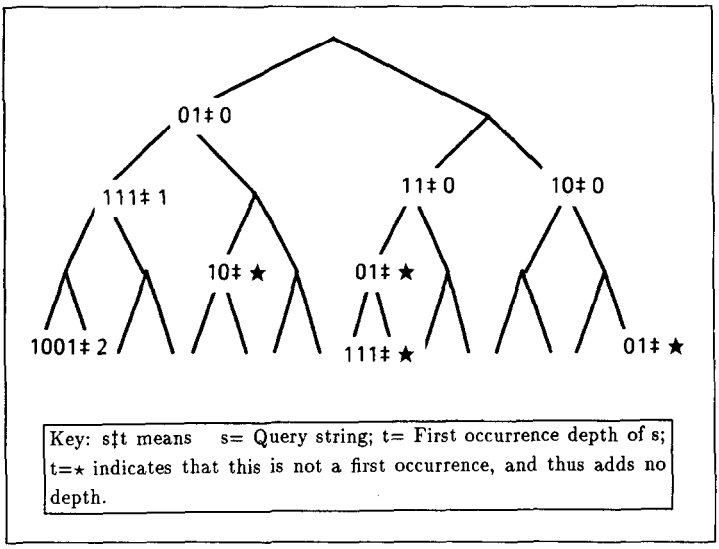


FIG. 8. First-occurrence depths.

depth zero. Then it does the same for queries at first occurrence depth one (the difference from Theorem 4.5 is that here the simulator may pass by many occurrences of strings assigned first occurrence depth one—the membership status of these strings is already known to the simulator), and so on. Finally, when the census functions for each first occurrence depth are known, a single final query to an NE oracle determines if the simulator should accept.

One might, in the case without restrictions, expect $\mathrm{NEXP}^{\mathrm{NP}}$ to be larger than $\mathrm{P}^{\mathrm{NEXP}}$.

THEOREM 4.7. $\mathrm{P}^{\mathrm{NEXP}} = \mathrm{NEXP}^{\mathrm{NP}[\exp]_{\mathrm{tree}}\,[\mathrm{poly}]_{\text{first occurrence depth}}}$.

4.4.2. *Many "No" Strings*

P^{NE} can query its oracle only a polynomial number of times. It is possible that P^{NE} can contain $\mathrm{NE}^{\mathrm{NP}[\mathrm{poly}]_{Y,\mathrm{tree}}}$, a class that may make up to $2^{2^{cn}}$ oracle calls (most receiving "no" answers)? The following theorem shows that it does. This theorem is related to Theorem 4.5, but note that below NE is permitted to ask exponentially many queries on many of its computation paths.

THEOREM 4.8. $\mathrm{P}^{\mathrm{NE}} \supseteq \mathrm{NE}^{\mathrm{NP}[\mathrm{poly}]_{Y,\mathrm{tree}}}$.

COROLLARY 4.9. $\mathrm{P}^{\mathrm{NE}} = \mathrm{NEXP}^{\mathrm{NP}[\mathrm{poly}]_{Y,\mathrm{tree}}}$.

Our trick is for P^{NE} to do binary search to find the *depth* of the first "yes" response in the run of $\mathrm{NE}^{\mathrm{NP}[\mathrm{poly}]_{Y,\mathrm{tree}}}(\cdot)$. Then as before it does binary search to find the number of yes strings at that level. P^{NE} repeats this level-finding/census-finding

alternation until it has gone through all the levels that have yes responses (it can detect this).

4.4.3. *Qualitative = Quantitative: Truth-Table Classes and Sparse Sets*

In the nomenclature discussed at the start of Section 4, truth-table classes [Yes83] are qualitatively restricted classes, and $P^{NP[\log]}$ is a quantitatively restricted class. This section shows that these notions sometimes describe the same class in different ways.

The following results about truth-table classes are proved not by the method of Section 3.1, but follow from the weakness of truth-table reductions and the power of census information. Theorem 4.10, Part 1, though simple to prove, seems rather fundamental and has been independently noted [Hem86b, Hem86a, KSW86, Wag88, BH88]. The class the theorem is about, $P^{NP[\log]}$, was first discussed by Papadimitriou and Zachos [PZ83]; more recent studies have shown that $P^{NP[\log]}$ has many equivalent characterizations [Wag88].

THEOREM 4.10. 1. $\{S \mid S \leqslant^{p}_{\text{truth-table}} \text{NP}\} = P^{NP[\log]}$.

2. $\{S \mid S \leqslant^{exp}_{\text{truth-table}} \text{NP}\} = P^{NE}$.

Proof. We prove the second part; the first part is analogous. $S \leqslant^{exp}_{\text{truth-table}} \text{NP}$ means there is an exponential-time machine that answers "$x \in S$?" by making queries to SAT [GJ79], such that the queries asked of SAT are independent of the answers received (see Ladner, Lynch, and Selman [LLS75] for a discussion of $\leqslant^{p}_{\text{truth-table}}$).

To prove the $\subseteq$ part, we have P perform binary search, using its NE oracle, to find the *number* of yes answers, and with one final query to the oracle, we have NE guess the yes answers and simulate the action of the truth-table reducer.

The $\supseteq$ part is trivial—the truth-table reducer asks all queries that might be formed by any of the $2^{n^{O(1)}}$ possible sets of oracle answers of the run of P^{NE}. ∎

Theorem 4.10 says that if EXP^{NP} is to strictly contain P^{NE}, it *must* use the answers to early queries to help it pose later ones. Of course, we do not hope to show $EXP^{NP} \supsetneq P^{NE}$, as this implies $P \neq NP$. However, we suspect that $P \neq NP$ and even $EXP^{NP} \supsetneq P^{NE}$.

Similarly, we can prove collapsing results for sparse sets, which have historically played a central role in computational complexity theory [Mah82, HIS85, HH88b]. A sparse set S is a set so that, for some polynomial $p(\ )$, for all n there are at most $p(n)$ strings in S of length at most n. Let SPARSE represent the class of sparse sets.

THEOREM 4.11. 1. $NE^{NP \cap SPARSE} \subseteq P^{NE}$.

2. $NP^{NE \cap SPARSE} \subseteq P^{NE[\log]}$.

Proof. The two parts have similar proofs. For the second part, as in Theorem 4.10 above, P^{NE} with $O(\log n)$ queries computes the exact number of strings reachable by the NP machine that are in the sparse NE set. ∎

5. Conclusions

Our goal was to find out if exponential hierarchies collapse, or if not, why they might separate. We have seen that the strong exponential hierarchy collapses, via a tight analysis of the query census of NP^{NE}. We have viewed the weak exponential hierarchy as NE with a rich database and seen that the Δ and Σ levels of the weak exponential hierarchy separate only if NE floods its database with queries.

More generally, the census techniques of this paper can be used as a general procedure for collapsing complexity classes that meet certain counting conditions [Hem86b, Section 4.3].

The weak exponential hierarchy has crisp characterizations in terms of quantifiers and in terms of alternating Turing machines (Section 2.2). An interesting open question is: does the strong exponential hierarchy have similar representations via quantifiers and alternating Turing machines? If so, can we prove the collapse of the strong exponential hierarchy via quantifier manipulations? Immerman's improved collapse of the logspace hierarchy followed from his study of logical formulas [Imm88].

Acknowledgments

Juris Hartmanis provided sage advice and important insights. I am grateful to Jin-yi Cai, Dexter Kozen, Jeffrey Lagarias, Daniel Leivant, Uwe Schöning, Richard Shore, Klaus Wagner, and Osamu Watanabe for many enlightening conversations and suggestions. Two anonymous referees and the editor, Steve Mahaney, made extremely helpful comments on improving the paper's presentation.

References

[AW88] E. Allender and O. Watanabe, Kolmogorov complexity and the degrees of tally sets, *in* "Proceedings, 3rd Structure in Complexity Theory Conference," pp. 102–111, IEEE Computer Society Press, New York, June 1988.

[Bei87] R. Beigel, "Bounded Queries to SAT and the Boolean Hierarchy," Technical Report TR-7, Johns Hopkins Department of Computer Science, Baltimore, MD, June 1987.

[BGS75] T. Baker, J. Gill, and R. Solovay, Relativizations of the P = ? NP question, *SIAM J. Comput.* **4**, No. 4 (1975), 431–442.

[BH77] L. Berman and J. Hartmanis, On isomorphisms and density of NP and other complete sets, *SIAM J. Comput.* **6**, No. 2 (1977), 305–322.

[BH88] S. Buss and L. Hay, On truth-table reducibility to SAT and the difference hierarchy over NP, *in* "Proceedings, 3rd Structure in Complexity Theory Conference," pp. 224–233, IEEE Computer Society Press, New York, June 1988.

[BLS84] R. Book, T. Long, and A. Selman, Quantitative relativizations of complexity classes, *SIAM J. Comput.* **13**, No. 3 (1984), 461–487.

[BLS85] R. Book, T. Long, and A. Selman, Qualitative relativizations of complexity classes, *J. Comput. System Sci.* **30** (1985), 395–413.

[Boo81] R. Book, Bounded query machines: On NP and PSPACE, *Theoret. Comput. Sci.* **15** (1981), 27–39.

[CGH*] J. CAI, T. GUNDERMANN, J. HARTMANIS, L. HEMACHANDRA, V. SEWELSON, K. WAGNER, AND G. WECHSUNG, The boolean hierarchy. II. Applications, *SIAM J. Comput.* **18**, No. 1 (1989), 95–111.

[CGH*88] J. CAI, T. GUNDERMANN, J. HARTMANIS, L. HEMACHANDRA, V. SEWELSON, K. WAGNER, AND G. WECHSUNG, The boolean hierarchy. I. Structural properties, *SIAM J. Comput.* **17**, No. 6 (1988), 1232–1252.

[CKS81] A. CHANDRA, D. KOZEN, AND L. STOCKMEYER, Alternation, *J. Assoc. Comput. Mach.* **26**, No. 1 (1981).

[CT86] P. CLOTE AND G. TAKEUTI, Exponential time and bounded arithmetic, *in* "Proceedings, 1st Structure in Complexity Theory Conference," pp. 125–143, Lecture Notes in Computer Science, Vol. 223, Springer-Verlag, New York/Berlin, 1986.

[GJ79] M. GAREY AND D. JOHNSON, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco, 1979.

[Har85] J. HARTMANIS, Solvable problems with conflicting relativizations, *Bull. European Assoc. Theoret. Comput. Sci.* **27** (1985), 40–49.

[Hem86a] L. HEMACHANDRA, "Can P and NP Manufacture Randomness?" Technical Report TR86-795, Cornell Computer Science Department, Ithaca, NY, December 1986.

[Hem86b] L. HEMACHANDRA, "THE SKY IS FALLING: The Strong Exponential Hierarchy Collapses," Technical Report TR86-777, Department of Computer Science, Cornell University, Ithaca, NY, August 1986.

[Hem87a] L. HEMACHANDRA, "Counting in Structural Complexity Theory," Ph.D. thesis, Cornell University, Ithaca, NY, May 1987; Cornell Department of Computer Science Technical Report TR87-840.

[Hem87b] L. HEMACHANDRA, The strong exponential hierarchy collapses, *in* "19th ACM Symposium on Theory of Computing," pp. 110–122, May 1987.

[Hem87c] L. HEMACHANDRA, The strong exponential hierarchy collapses (Abstract), *in* "Proceedings, 2nd Structure in Complexity Theory Conference," IEEE Computer Society Press, New York, June 1987.

[HH88a] J. HARTMANIS AND L. HEMACHANDRA, Complexity classes without machines: On complete languages for UP, *Theoret. Comput. Sci.* **58** (1988), 129–142.

[HH88b] J. HARTMANIS AND L. HEMACHANDRA, On sparse oracles separating feasible complexity classes, *Inform. Process. Lett.* **28** (1988), 291–295.

[HIS85] J. HARTMANIS, N. IMMERMAN, AND V. SEWELSON, Sparse sets in NP–P: EXPTIME versus NEXPTIME, *Inform. and Control*, **65**, Nos. 2/3 (1985), 159–181.

[HS65] J. HARTMANIS AND R. STEARNS, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965), 285–306.

[HU79] J. HOPCROFT AND J. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA, 1979.

[HY84] J. HARTMANIS AND Y. YESHA, Computation times of NP sets of different densities, *Theoret. Comput. Sci.* **34** (1984), 17–32.

[Imm88] N. IMMERMAN, Nondeterministic space is closed under complementation, *in* "Proceedings, 3rd Structure in Complexity Theory Conference," p. 112–115, IEEE Computer Society Press, New York, June 1988.

[Kad86] J. KADIN, "Deterministic Polynomial Time with $O(\log(n))$ Queries," Technical Report TR-86-771, Cornell University, Ithaca, NY, August 1986.

[Kad87] J. KADIN, $P^{NP[\log n]}$ and sparse Turing-complete sets for NP, *in* "Proceedings, 2nd Structure in Complexity Theory Conference," p. 33–40, IEEE Computer Society Press, New York, June 1987.

[Kil87] J. KILIAN, June 1987, personal communication.

[KSW86] J. KÖBLER, U. SCHÖNING, AND K. WAGNER, "The Difference and Truth-Table Hierarchies for NP," Technical Report, Fachberichte Informatik, EWH Rheinland-Pfalz, Koblenz, West Germany, July 1986.

[LJK87] K. LANGE, B. JENNER, AND B. KIRSIG, The logarithmic alternation hierarchy collapses: $A\Sigma_2^L = A\Pi_2^L$, *in* "Automata, Languages, and Programming (ICALP 1987)," Lecture Notes in Computer Science, Springer-Verlag, New York/Berlin, 1987.

[LLS75] R. LADNER, N. LYNCH, AND A. SELMAN, A comparison of polynomial time reducibilities, *Theoret. Comput. Sci.* **1**, No. 2 (1975), 103–124.

[Lon85] T. LONG, On restricting the size of oracles compared with restricting access to oracles, *SIAM J. Comput.* **14**, No. 3 (1985), 585–597; Erratum, **17**, No. 3 (1988), 628.

[Mah82] S. MAHANEY, Sparse complete sets for NP: Solution of a conjecture of Berman and Hartmanis, *J. Comput. System Sci.* **25**, No. 2 (1982), 130–143.

[PZ83] C. PAPADIMITRIOU AND S. ZACHOS, Two remarks on the power of counting, *in* "Proceedings, 6th GI Conference on Theoretical Computer Science," pp. 269–276, Lecture Notes in Computer Science, Vol. 145, Springer-Verlag, New York/Berlin, 1983.

[Sel86] A. SELMAN (ED.), "Proceedings, 1st Structure in Complexity Theory Conference," Lecture Notes in Computer Science, Vol. 223, Springer-Verlag, New York/Berlin, June 1986.

[Sew83] V. SEWELSON, "A Study of the Structure of NP," Ph.D. thesis, Cornell University, Ithaca, NY, August 1983; Cornell Department of Computer Science Technical Report No. 83-575.

[SMB83] A. SELMAN, X. MEI-RUI, AND R. BOOK, Positive relativizations of complexity classes, *SIAM J. Comput.* **12** (1983), 565–579.

[Sto77] L. STOCKMEYER, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977), 1–22.

[SW88] U. SCHÖNING AND K. WAGNER, Collapsing oracle hierarchies, census functions, and logarithmically many queries, *in* "STACS 1988: 5th Annual Symposium on Theoretical Aspects of Computer Science," Lecture Notes in Computer Science, Springer-Verlag, New York/Berlin, February 1988.

[Tod87] S. TODA, Σ_2 SPACE[n] is closed under complement, *J. Comput. System Sci.* **35** (1987), 145–152.

[Wag88] K. WAGNER, Bounded query computation, *in* "Proceedings, 3rd Structure in Complexity Theory Conference," pp. 260–277, IEEE Computer Society Press, New York, June 1988.

[Wat87] O. WATANABE, May 1987, personal communication.

[Yes83] Y. YESHA, On certain polynomial-time truth-table reducibilities of complete sets to sparse sets, *SIAM J. Comput.* **12**, No. 3 (1983), 411–425.

[Zac86] S. ZACHOS, Probabilistic quantifiers, adversaries, and complexity classes: An overview, *in* "Proceedings, 1st Structure in Complexity Theory Conference," pp. 383–400, IEEE Computer Society Press, New York, June 1986.