THE SKY IS FALLING:

The Strong Exponential Hierarchy Collapses

Lane Hemachandra*

TR 86-777
August 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

*This work was supported by a Fannie and John Hertz Fellowship and NSF Research Grant DCR-8520597.

THE SKY IS FALLING[1]

The Strong Exponential Hierarchy Collapses

Lane A. Hemachandra[2]
Department of Computer Science
Cornell University

[1]Chicken Little thought that the sky was falling [Kel85]. It was not.

[2]This work was supported by a Fannie and John Hertz Fellowship and NSF Research Grant DCR-8520597.

Abstract

This paper investigates the complexity of the high levels of the exponential hierarchy [HY84,HIS85]: Are they hard, and if so *why* are they hard?

We show that

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

From this, we conclude that the strong exponential hierarchy collapses:

$$P^{NE} = NP^{NE} \cup NP^{NP^{NE}} \cup NP^{NP^{NP^{NE}}} \cup \cdots,$$

where NE is nondeterministic exponential time. This surprising result, a nontrivial hierarchy collapse, is based on P^{NE} overmastering the NP^{NE} computation tree by computing better and better partial census information. We carefully note why the combinatorics involved prevents us from similarly proving that the polynomial hierarchy collapses.

Next we look at the exponential hierarchy, which is NE given a rich database:

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

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

Thus the high levels of the strong exponential hierarchy are no harder than the low levels. The high levels of the exponential hierarchy separate completely only if NE floods its database with queries. Extending our techniques, we derive sufficient conditions for collapsing complexity classes, and use them to generate strong new quantitative relativization results.

1 Introduction

We wish to know if the high Δ and Σ levels of complexity hierarchies are hard, and if so, *why* they are hard. We see that the high levels of the strong exponential hierarchy are *not* harder than the low levels; the hierarchy collapses. We show that the high levels of the exponential hierarchy, say $E^{\Sigma_j^p}$ and $NE^{\Sigma_j^p}$, separate only if NE floods its oracle with queries.

Section 2 looks at the strong exponential hierarchy:

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

Shockingly, 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.

Section 2.2 shows that this result is neither trivial nor apocalyptic. The result is not trivial in the sense that $P^{PSPACE} = NP^{PSPACE} = PSPACE$ is trivial; there are worlds where $P^{NE} \neq NE$. On the other hand, the combinatorics involved keeps this technique from collapsing the polynomial hierarchy [Sto77] to P^{NP}.

Section 3 surveys and analyzes the candidates for the title "Exponential Hierarchy" [HY84,HIS85].

Section 4 uses the census techniques of Section 2 to prove strong new results on quantitative relativization. We first review the gems of recent work on quantitative relativization, in particular the work 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, yields exciting new quantified collapses of complexity classes and also unifies previous results. We outline powerful sufficient conditions for collapsing complexity classes.

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

2 The Sky Has Fallen

2.1 The Strong Exponential Hierarchy Collapses

This section proves that the strong exponential hierarchy collapses to its Δ_2 level. It suffices to collapse the strong exponential hierarchy's Δ_2 and Σ_2 levels. Then downward separation gives us a quick proof of the hierarchy collapse.

Definition 2.1

$$\mathrm{NE} = \bigcup_c \mathrm{NTIME}[2^{cn}] \qquad \mathrm{NEXP} = \bigcup_k \mathrm{NTIME}[2^{n^k}]$$

$$\mathrm{SEH} = \mathrm{E} \cup \mathrm{NE} \cup \mathrm{NP}^{\mathrm{NE}} \cup \mathrm{NP}^{\mathrm{NP}^{\mathrm{NE}}} \cup \cdots$$

$$\mathrm{SEXPH} = \mathrm{EXP} \cup \mathrm{NEXP} \cup \mathrm{NP}^{\mathrm{NEXP}} \cup \mathrm{NP}^{\mathrm{NP}^{\mathrm{NEXP}}} \cup \cdots$$

Lemma 2.2 $\mathrm{P}^{\mathrm{NE}} = \mathrm{NP}^{\mathrm{NE}}$.

Theorem 2.3 $\mathrm{P}^{\mathrm{NE}} = \mathrm{SEH}$.

Corollary 2.4 $\mathrm{P}^{\mathrm{NE}} = \mathrm{SEH} = \mathrm{SEXPH}$.

Proof of Theorem 2.3 Define $\Sigma_1^{\mathrm{SEH}} = \mathrm{NE}$, $\Sigma_{k+1}^{\mathrm{SEH}} = \mathrm{NP}^{\Sigma_k^{\mathrm{SEH}}}$ for $k \geq 1$, and $\Delta_2^{\mathrm{SEH}} = \mathrm{P}^{\mathrm{NE}}$. By Lemma 2.2, $\Delta_2^{\mathrm{SEH}} = \Sigma_2^{\mathrm{SEH}}$. Inductively assume (for some $k \geq 2$) that $\Delta_2^{\mathrm{SEH}} = \Sigma_k^{\mathrm{SEH}}$. Now

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

The last equality holds because there is an NP machine that takes over the job of the P machine (in $\mathrm{NP}^{\mathrm{P}^{\mathrm{NE}}}$) and does all the NE queries itself.

Inductively, $\Sigma_k^{\mathrm{SEH}} = \Delta_2^{\mathrm{SEH}}$ for all k. So $\mathrm{SEH} = \mathrm{P}^{\mathrm{NE}}$.

♠

Corollary 2.4 is proven in Section 3.2.

All the work of our result lies in the proof of Lemma 2.2. We make extraordinary use of the power of NE to guess query strings, witnesses, and paths in trees.

First, we must place graphically in you 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, Figure 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 paper we assume, without loss of generality, 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 write queries to an NE oracle. Each of the nondeterministic paths may, of course, write different queries than its brothers (Figure 2). We label the depth of the nodes computation trees in the standard way (Figure 2).

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

We can't just jump in and 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 are patient and 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 knowledge of the previous levels to help us binary search for the number of strings at the current level.

For concreteness, suppose our NP^{NE} language is that accepted by NP machine N_{17} with NE machine NE_{21} as its oracle. 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, and between 8 and 16 at level 5, respectively. 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?

Crucially, this is the kind of question that NE can answer. NE guesses the first six levels of NP'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 $\geq$ 12 alleged yes strings at level 5, and guesses the succinct proofs that these strings really are yes strings (i.e., are accepted by NE_{21}).

If 1,1,0,4,11 is correct, then the first 5 levels correspond to the first 5 levels of the actual computation tree of $N_{17}^{NE_{21}}(x)$, and if there are $\geq$ 12 yes strings at level 5 level 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 $N_{17}^{NE_{21}}(x)$.

Let's refer to the P and NE machines we use to simulate NP^{NE} as $P_\star$ and $NE_\star$. Figure 4 shows how the trees $NE_\star$ guesses increase in the (literal) depth of their accuracy at reflecting the tree of $N_{17}^{NE_{21}}(x)$. Crucially, $P_\star$ 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 $P_\star$ could remember all the yes *names* since deep in the $N_{17}^{NE_{21}}$ tree there may be as many as 2^{n^k} yes strings on a single level. Our use of increasingly accurate censuses of the number of queries per level is central to the success of our result.

Proof of Lemma 2.2 ($P^{NE} = NP^{NE}$) Let L be an arbitrary language in NP^{NE}. Without loss of generality, $L = L(N_{17}^{NE_{21}})$, where N_{17} is an NP machine and NE_{21} is an NE machine. For concreteness, suppose N_{17} runs in NTIME[n^{17}]. We describe machines $P_\star$ and $NE_\star$, respectively P and NE machines, so that $L = L(P_\star^{NE_\star})$. Thus $P^{NE} = NP^{NE}$.

Let us first describe $\mathrm{NE}_\star$.

$$L(\mathrm{NE}_\star) = \{final\#1^{|x|^{18}}\#c_1\#c_2\#\cdots\#c_l\#c_{l+1} \mid \text{There exist sets } C_1, C_2 \ldots, C_l, C_{l+1} \text{ of strings so:}$$

1. $|C_i| = c_i$, $\bigcup C_i \subseteq L(\mathrm{NE}_{21})$, and
2. If we simulate $\mathrm{N}_{17}{}^{()}(x)$, 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 actually queried at level i in this simulation, and
3. If *final* is 1, there is an accepting path in the simulation mentioned in 2 above.}

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

Note that $L(\mathrm{NE}_\star)$ *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, we can implement $\mathrm{NE}_\star$ easily in $\mathrm{NTIME}[2^{|x|^{18}}]$. Fortunately, we've padded so our input size is greater than $|x|^{18}$, so $\mathrm{NE}_\star$ runs in $\mathrm{NTIME}[2^{c\cdot \mathrm{Inputsize}}]$, and thus is in NE. Thus $L(\mathrm{NE}_\star) \in \mathrm{NE}$.

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

Stage i: Inductively, we have 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 $\mathrm{N}_{17}{}^{\mathrm{NE}_{21}}(x)$ (Figure 2). During this stage we find c_i, the actual number of yes strings at level i. This is easy– just binary search (varying z) using calls to $\mathrm{NE}_\star$ of the form

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

to find the value of c_i.

Query	$NE_\star$*'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*

Table 1: Binary search over calls to $NE_\star$ discovers that there are 3 yes strings at level 3 of the computation tree of ${N_{17}}^{NE_{21}}(11101)$

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

It is crucial to notice that when $c_0, \ldots, c_{i-1}$ are correct, the c_i we find is correct. This is because a branch, henceforward yclept B, of $NE_\star$ will really guess the true yes strings $C_0, \ldots, C_{i-1}$. The queries asked at level i depend only on the fact that we know the right oracle replies at levels 0 through $i-1$; thus the queries asked at level i on branch B will be the queries that are asked in the tree of ${N_{17}}^{NE_{21}}(x)$ at level i. Thus if there are at least z yes strings queried at level i of the tree of ${N_{17}}^{NE_{21}}(x)$, B will guess them.

On the other hand, any branch that does not guess the sets $C_0, \ldots, C_{i-1}$ correctly (it guesses, say, $C_0', \ldots, C_{i-1}'$) certainly will not accept. At the first level it errs from 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 ${N_{17}}^{NE_{21}}(x)$. Since the C_i' are correct at levels $0, \ldots, m-1$, 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 $L(NE_\star)$ (page 5) is violated and this branch won't accept. If w is not queried at level m, condition 2 of the definition of $L(NE_\star)$ is violated and this branch won't 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 $P_\star^{(\cdot)}$ is easily polynomial; it runs in TIME$[n^{4 \cdot 17+2}]$. Returning to the general case, if N_{17} is in NTIME$[n^k]$, then $P_\star$ is in deterministic TIME$[n^{4k+2}]$.

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

♠

2.2 Is This Collapse Trivial or is it Apocalyptic

We have just shown that $P^{NE} = NP^{NE}$, and thus SEH collapses to P^{NE}. Is this collapse of the strong exponential hierarchy trivial or earthshattering? Could the collapse be trivial in the sense that $P^{PSPACE} = NP^{PSPACE} = PSPACE$ is trivial? PSPACE is so powerful that both P^{PSPACE} and NP^{PSPACE} are equal to PSPACE. Is NE so powerful that $P^{NE} =^? NE$? Relativization techniques [CH86a,HH86b,HH86a] help us here. There is a relativized world where $P^{NE^A} \supsetneq NE^A$. That this is not a fluke side effect P^{NE}'s ability to reach 2^{n^k} long things (compared with NE's reach of 2^{cn}) is shown by Theorem 2.5. Section 3.2 discusses this "reach" anomaly in detail.

Theorem 2.5 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

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

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.

♠

On the other hand, the result is not apocalyptic in that these techniques do not collapse the polynomial hierarchy ($\mathrm{NP} \cup \mathrm{NP}^{\mathrm{NP}} \cup \cdots$) to P^{NP}. Suppose we tried to show that $\mathrm{P}^{\mathrm{NP}} = \mathrm{NP}^{\mathrm{NP}}$ using the above methods. $\mathrm{NP}^{\mathrm{NP}}$ 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 can not guess an exponentially large object: the names of the 2^{n^k} yes strings in the tree of $\mathrm{NP}^{\mathrm{NP}}$.

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

3 Will the Real Exponential Hierarchy Please Stand Up

3.1 Definitions

This section prepares for our quantified relativization results by defining and comparing the strong and vanilla exponential hierarchies.

Definition 3.1

1. $\mathrm{E} = \bigcup_c \mathrm{TIME}[2^{cn}] \qquad \mathrm{EXP} = \bigcup_k \mathrm{TIME}[2^{n^k}]$
2. $\mathrm{NE} = \bigcup_c \mathrm{NTIME}[2^{cn}] \qquad \mathrm{NEXP} = \bigcup_k \mathrm{NTIME}[2^{n^k}]$

3. EH = exponential hierarchy = $\mathrm{NE} \cup \mathrm{NE}^{\mathrm{NP}} \cup \mathrm{NE}^{\mathrm{NP}^{\mathrm{NP}}} \cup \cdots$ [HIS85]
   $\mathrm{EXPH} = \mathrm{NEXP} \cup \mathrm{NEXP}^{\mathrm{NP}} \cup \mathrm{NEXP}^{\mathrm{NP}^{\mathrm{NP}}} \cup \cdots$

4. SEH = strong exponential hierarchy = $\mathrm{NE} \cup \mathrm{NP}^{\mathrm{NE}} \cup \mathrm{NP}^{\mathrm{NP}^{\mathrm{NE}}} \cup \cdots$

Both E and EXP are commonly referred to as exponential time (compare [CT86] with [BH77]), though E is more common in the literature of structural complexity [Sel86,HY84]. In this paper we always make clear which exponential time we are speaking of. We'll see later in this section that sometimes using EXP instead of E will avoid anomalies.

3.2 Why SEH is Strong: Sensitivity to Padding

SEH is called strong because it is easy to see that 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}, while EH^A can only query strings of length 2^{cn}. To understand this, just reflect on the fact that

$$\mathrm{P}^{\mathrm{E}} \stackrel{?}{\neq} \mathrm{E}^{\mathrm{P}},$$

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

Theorem 3.2 There is a relativized world A so that $\mathrm{SEH}^A - \mathrm{EH}^A \neq \emptyset$. Indeed, in this world $\mathrm{P}^{\mathrm{NE}^A} - \mathrm{EH}^A \neq \emptyset$.

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 input of length n, we can easily diagonalize against each EH machine.

♠

Similarly we get the following trick separation that is due wholly to this padding anomaly. This is what we mean when we say that E causes anomalies that EXP avoids.

Trick Result 3.3 $\Sigma_0^{\mathrm{SEH}} \neq \Delta_2^{\mathrm{SEH}}$.

The extreme sensitivity of E and NE oracles to polynomial padding of their input strings has another consequence. The hierarchy SEXPH (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 sticks polynomial padding onto each query string, its NE oracle can in effect simulate the NEXP calls of $\mathrm{P}^{\mathrm{NEXP}}$. Thus we have extended the collapsing result of Section 2.

Corollary 2.4 $\mathrm{P}^{\mathrm{NE}} = \mathrm{SEH} = \mathrm{SEXPH} =_{\mathrm{def}} \mathrm{NEXP} \cup \mathrm{NP}^{\mathrm{NEXP}} \cup \mathrm{NP}^{\mathrm{NP}^{\mathrm{NEXP}}} \cup \cdots$.

3.3 Downward Separations

If we collapse the polynomial hierarchy at any level the entire hierarchy crumbles to that level; $\Sigma_i^p = \Pi_i^p \Rightarrow \Sigma_i^p = \mathrm{PH}$ [Sto77]. This is known as downward separation (and remembered as upward collapsing). One troubling feature of the exponential hierarchy is that it does not have downward separation. Hartmanis, Immerman, and Sewelson [HIS85] display a relativized world A where $\mathrm{E}^A = \mathrm{NE}^A \neq \mathrm{NE}^{\mathrm{NP}^A}$

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 3.4 (Downward Separation)

1. $\mathrm{E} = \mathrm{NE} \Rightarrow \mathrm{EXP} = \mathrm{SEH}$.
2. $\mathrm{NE} = \mathrm{CONE} \Rightarrow \mathrm{NEXP} = \mathrm{SEH}$.

Proof

1. Using Theorem 2.3 and our assumption that E = NE,

$$\mathrm{EXP} \subseteq \mathrm{SEH} = \mathrm{P}^{\mathrm{NE}} = \mathrm{P}^{\mathrm{E}} = \mathrm{EXP}.$$

2. When NE = CONE, P^{NE} can be simulated by NEXP, which just guesses the correct oracle answers along with their certificates of correctness.

♠

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

3.4 Which is the Real Exponential Hierarchy?

Unfortunately, the answer to this questions is a matter of taste. Each version has some desirable qualities and some weaknesses. Using E over EXP has paid off well in structural complexity [HY84], but admits the padding anomalies discussed in Section 3.2.

EH has a natural interpretation in terms of alternating Turing machines, but lacks downward separation. On the other hand, SEH nicely displays downward separation, but lacks an obvious alternating Turing machine model.

Each hierarchy will have to be used and studied intensively before it is understood. However, this paper seeks to make a strong start. Section 2.1 showed that SEH collapses; its high levels are no harder than its low levels. Section 4 shows that EH and EXPH can completely separate their levels only if they flood their oracles with queries.

4 Quantitative Relativization Results

4.1 Definitions

Quantitative relativization means relativization in which the power of some base machine to query its oracle is restricted. Before introducing this field, we must define our notation. Since many varied notations have been used, we take this opportunity to define a new notation that is simple and clear.

Definition 4.1 Let A and B be complexity classes. The class

$$A^{B[f(n)]_{path}[g(n)]_{tree}[h(n)]_{explicitly-named-restriction}}$$

is the class of languages accepted by an oracle Turing machine from A with a set from B as its oracle, under the restrictions that:

1. 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, and
2. the total number of strings B is queried about throughout A's entire computation tree is $g(n)$, and
3. there is an $h(n)$ bound on whatever is named by "explicitly-named-restriction."

Examples and Notes

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

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

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

4. Our quantitative classes count strings– not oracle calls. (This makes our theorems harder to prove but stronger.) That is, $[n^2]_{tree}$ means that for all x, $|\{y \mid y \text{ is queried in the tree on input } x\}| \leq |x|^2$. 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 $[\cdot]_{tree}$ measure for this whole set of queries– *not* $2^{2^{cn}}$ (Figure 5d).

Finally, we make it harder still to prove our theorems.

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

4.2 Introduction and Background

The techniques of Section 2.1 will be used to prove strong new theorems about quantitative relativizations. We'll see, for example, that:

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

Thus the only way $\mathrm{NE}^{\mathrm{NP}}$ can avoid collapsing to E^{NP} is by using its oracle fantastically often. Similarly, we'll see that:

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

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

Class $= A^B$ restrictions	
Restrictions	
$[timebound]_{path}$	*limit on the number of strings queried on each of A's computation paths*
$[timebound]_{tree}$	*limit on the number of strings queried throughout A's computation tree*
$[timebound]_{Y,tree}$	*limit on the number of strings queried throughout A's computation tree that are actually in B*
Timebounds	
log	$\bigcup_c$ TIME$[c \log n]$
poly	$\bigcup_k$ TIME$[n^k]$
e	$\bigcup_c$ TIME$[2^{cn}]$
exp	$\bigcup_k$ TIME$[2^{n^k}]$

Table 2: Nomenclature of Quantitative Relativization

We should think of P^{NE} as a frail sage (P) holding a tiger (NE) by the tail. If the P machine is wise in its use of NE, then P^{NE} wields tremendous power. Our P machines know their limitations, and are happy to bookkeep and let their tigers do the hard work.

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

$$\mathrm{P}^{\mathrm{NP}} = \mathrm{NP} \cup \mathrm{NP}^{\mathrm{NP}[poly]tree} \cup \mathrm{NP}^{(\mathrm{NP}^{\mathrm{NP}[poly]tree})[poly]tree} \cup \cdots,$$

which provides a polynomial analogue of our $\mathrm{P}^{\mathrm{NE}} = \mathrm{NP}^{\mathrm{NE}}$ result. Their important paper provides a complete and 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 General Conditions for Collapsing Complexity Classes

Given our success in Section 2, we wish to apply our census techniques to quantitative relativization. Given nice nondeterministic classes B, C, and D, and deterministic class A, the method of Section 2 will usually suffice to show that

$$A^B \supseteq C^{D\ldots restrictions\ldots}$$

when the following conditions hold (here we use a class and its time bound interchangeably).

1. $A \geq$ log of the number of yes strings in C's computation tree.
2. B running on inputs of size $A \geq$ the number of yes strings in C's computation tree.
3. $A \geq$ number of "interesting levels" of C.
4. B on inputs of size A is $\geq C$.

Notes

- Though we state this as a guide and not an absolute theorem, it is simply a generalization of the method of Lemma 2.2.

- [Lemma 2.2] $P^{NE} = NP^{NE}$
- [Corollary 4.9] $P^{NE} = NEXP^{NP[poly]_{Y,tree}}$
- [Lon85, Theorem 5.3] $P^{NP} = NP^{NP[poly]_{Y,tree}}$
- [Theorem 4.5] $P^{NE} \supseteq NE^{NP[poly]_{path}[exp]_{Y,tree}}$

Table 3: Theorems Generated by the Guidelines

- "Interesting levels" (Condition 3) refers to the number of times that A has to stop and compute a query-census of some "level" of C's computation tree. For example, Theorem 4.5 deals with deep trees that we can view as having few interesting levels (with level defined in a new loose way).

- Let's understand the conditions by example. We want to know if $E^{NP} \supseteq^{?} NEXP^{NP[e]_{Y,tree}}$ follows from the guidelines. Condition 1 is met: $E \geq$ poly. Condition 2 is met: $(2^{cn})^k = 2^{kcn}$ so $\text{poly}(E(\cdot)) = E(\cdot) \geq E(\cdot)$. Conditions 3 and 4 are violated, all because $E \not\geq EXP$. Thus we fail to show $E^{NP} \supseteq NEXP^{NP[e]_{Y,tree}}$. This is not surprising as it implies $P \neq NP$ (since if P=NP it says E=EXP, which is false).

- However, the conditions to give us the related equality $E^{NP} \supseteq NE^{NP[e]_{Y,tree}}$ (Corollary 4.4), as the violated conditions all become $E \geq E$.

- In the method of Lemma 2.2, we could afford to guess a whole nondeterministic computation tree. As we note in the proof of Theorem 4.3, that is a luxury that we can not afford in general. Instead, we guess just the tree paths on which strings we are interested in appear.

Table 3 shows theorems that follow from the guidelines.

4.4 Quantitative Relativization Theorems

This section proves theorems that give insight into what makes hierarchies non-trivial. The first, and most important, shows that the Σ_k and Δ_k levels of EH collapse unless the Σ_k level, $\mathrm{NE}^{\Sigma_{k-1}^p}$, uses its Σ_{k-1}^p oracle extensively. This is related to the polynomial time results of Long [Lon85, page 595].

Theorem 4.3 $\mathrm{E}^{\Sigma_k^p} = \mathrm{NE}^{\Sigma_k^p [e]_{Y,tree}}$

Corollary 4.4

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

Proof Sketch of Theorem 4.3 The method of Section 2.1 works with a subtle and 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. Luckily, since there is a $[e]_{Y,tree}$ bound, we are interested in checking at most 2^{cn} nodes of this tree. So now, our NP machine called by E will just guess the yes strings, the certificates that they are yes strings, and the paths in the computation tree of $\mathrm{NE}^{\mathrm{NP}}$ that led to them. All these things can be done with 2^{cn} nondeterministic steps, and allow us to guess and check all the portions of the huge $\mathrm{NE}^{\mathrm{NP}}$ tree that are of interest to us.

♠

The following theorem shows that $\mathrm{NE}^{\mathrm{NP}}$ is dominated by P^{NE} unless it makes extravagantly many oracle calls. The result is surprising as P^{NE} makes only polynomially many nondeterministic queries, but $\mathrm{NE}^{\mathrm{NP}[poly]_{path}[e]_{tree}}$ makes exponentially many queries, any one of which may be far too long for P to record!

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

Corollary 4.6

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

Proof 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 (Figure 6a).

Our trick is to identify with each query its "query depth": how many queries are made before it on its path (Figure 6b). P^{NE} will have polynomially many rounds. In each round i it will find, 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 2.1 now works. 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 NE.

♠

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.

♠

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

4.5 More Quantitative Relativization Theorems

If we really understand what is going on, we can when needed tailor and extend the techniques to a surprising breadth of problems. This section presents three interesting problems and briefly outlines the modifications needed.

4.5.1 Paying Only for the First Occurrence

Look carefully at q_0 and q_1 in Figure 6b, and consider the case when they both query the same string. There is no reason q_1 should eat up a query depth level on its path. After all, by the time we get to q_1, we know the census at query depth 0 and our NE machine (of P^{NE}) will on the key branch (the one that guesses correctly strings at query depth zero) already know that q_0 is a yes string if it is. If it is not, but we have an $[exp]_{tree}$ restriction on $\mathrm{NEXP}^{\mathrm{NP}}$, then the NE of P^{NE} can remember all the no answers at each census depth. Calling the census depth of a string with respect to first occurrence the first occurrence depth (Figure 7 gives examples), we have the following theorem.

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

Simply put, this says that once a query is paid for by a nondeterministic branch, its brother branches get to query the same string for free.

4.5.2 Many "No" Strings

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

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

The problem in proving these theorems is again one of levels. There may be "no" queries all over the NE tree– perhaps $2^{2^{cn}}$ of them. How can P^{NE} get away with only a polynomial number of calls to NE?

Our trick is that P^{NE} binary searches to find the *depth* of the first "yes" response in the run of $\mathrm{NE}^{\mathrm{NP}[poly]_{Y,tree}}(\cdot)$. Then as before it binary searches 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).

To find if the first yes in, for example, $\mathrm{NE}_{17}{}^{N_{21}[poly]_{Y,tree}}(x)$, is at level ≤ 1273, NE (of P^{NE}) guesses (for each level $j = 1, 2, \ldots, 1273$) that the first yes is on level

j and simulates $NE_{17}{}^{(\cdot)}(x)$, answering all queries at depth $< j$ "no," and accepting only if some query made at depth j is really accepted by N_{21} (it verifies this by guessing the certificate). Now, note that if the branch that guesses, say, $j = 1001$ accepts, then either the first yes is at level 1001, or a string at some level before 1001 is really a yes and has corrupted our simulation of $NE_{17}{}^{N_{21}}$. In either case, we correctly accept.

When looking for the second level that has a yes answers, we pass up to NE the name of the first level with yes strings and the number of yes strings at that level.

4.5.3 Truth-Table Classes

Finally, we mention some easy but surprising results, probably part of the folklore, about truth-table classes. They follow not from the method of Section 2.1, but simply from the weakness of truth table reductions.

Theorem 4.10

1. $\{S \mid S \leq^{p}_{\text{truth-table}} \text{NP}\} \subseteq \text{P}^{\text{NP}[log]}$.
2. $\{S \mid S \leq^{exp}_{\text{truth-table}} \text{NP}\} \subseteq \text{P}^{\text{NE}}$.

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

To prove 2 above, we have P binary search, using its NE oracle, to find the *number* of yes answers, and with one final query to the oracle have NE guess the yes answers and simulate the action of the truth-table reducer.

♠

Theorem 4.10 says that if EXP^{NP} is to dominate P^{NE}, it *must* use the answers to early queries to help it pose later ones. Of course, we don't hope to show

$\mathrm{EXP}^{\mathrm{NP}} \supsetneq \mathrm{P}^{\mathrm{NE}}$, as this implies $\mathrm{P} \neq \mathrm{NP}$. However, we suspect that $\mathrm{P} \neq \mathrm{NP}$ and even $\mathrm{EXP}^{\mathrm{NP}} \supsetneq \mathrm{P}^{\mathrm{NE}}$.

5 Open Problems and Conclusions

- EH has a nice characterization in terms of the alternating Turing machines of Chandra, Kozen, and Stockmeyer [CKS81] and in terms of quantified formulas. EH is exactly the languages of the form

$$\{x \mid (\exists_{\mathrm{E}} y_1)(\forall_{\mathrm{E}} y_2) \cdots (Q_{k,\mathrm{E}} y_k)[R(x, y_1, \ldots, y_k)]\},$$

  where R is a polynomial time predicate and an E subscript denotes a 2^{cn} bound on the quantifier size (e.g., "$(\exists_{\mathrm{E}} y)[$" is short for "$(\exists y)[|y| \leq 2^{cn} \wedge$") [HIS85]. In terms of alternating Turing machines, EH models alternating Turing machines with a bounded number of 2^{cn}-sized alternation blocks. Does SEH have similar representations via quantifiers and alternating Turing machines? The obvious approaches do not seem to work.

- Is there a relativized world where $\mathrm{EXP}^{\mathrm{NP}} \supsetneq \mathrm{P}^{\mathrm{NE}}$ (see the discussion following the proof of Theorem 4.10)? We conjecture there is.

Our goal was to find if exponential hierarchies collapse, or if not, why they might separate. We've seen that the strong exponential hierarchy collapses, via a tight analysis of the query census of $\mathrm{NP}^{\mathrm{NE}}$. Thus NE is so powerful that it overmasters the NP computation tree of $\mathrm{NP}^{\mathrm{NE}}$. We've viewed the exponential hierarchy as NE with a rich database, and seen that the hierarchy separates its Δ and Σ levels only if NE floods its database with queries.

Acknowledgements

Professor Juris Hartmanis provided sage advice, constant guidance, and important insights. I am grateful to Jin-yi Cai for many enlightening conversations and comments.

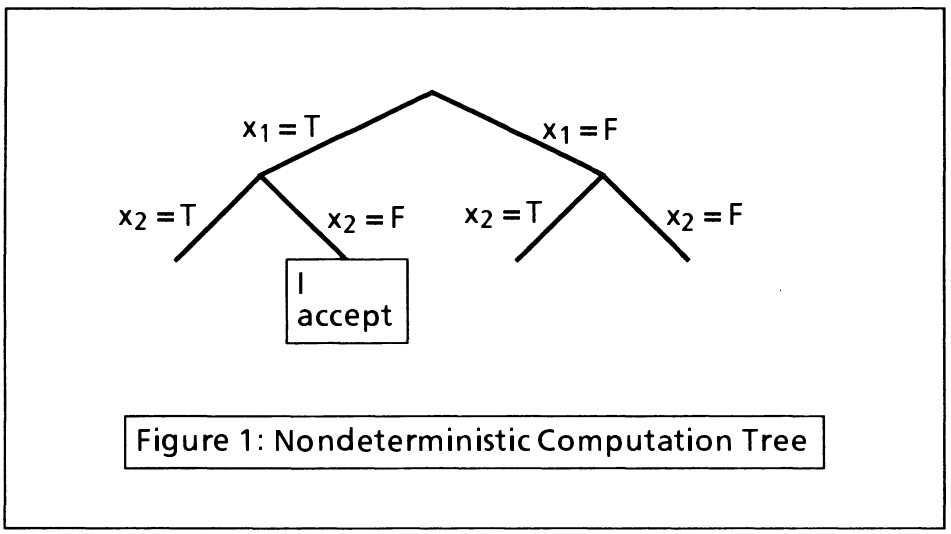


Figure 1: Nondeterministic Computation Tree

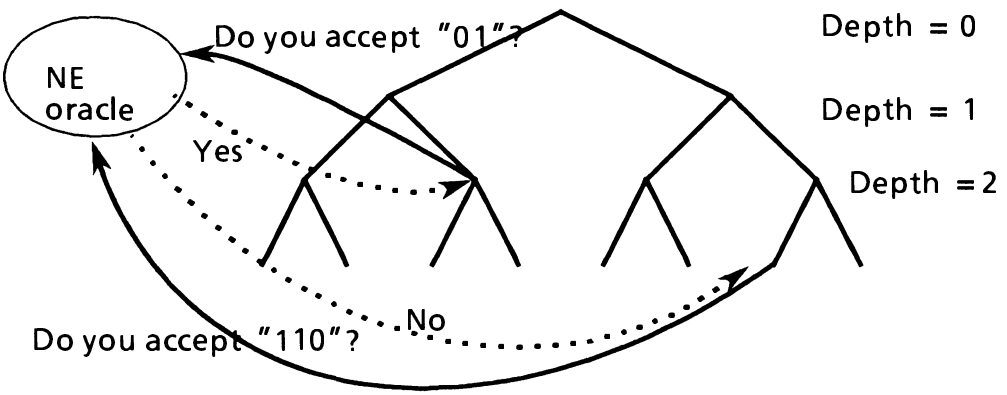


Figure 2: NP^{NE} Computation Tree: An NP computation tree with oracle calls allowed to an NE oracle.

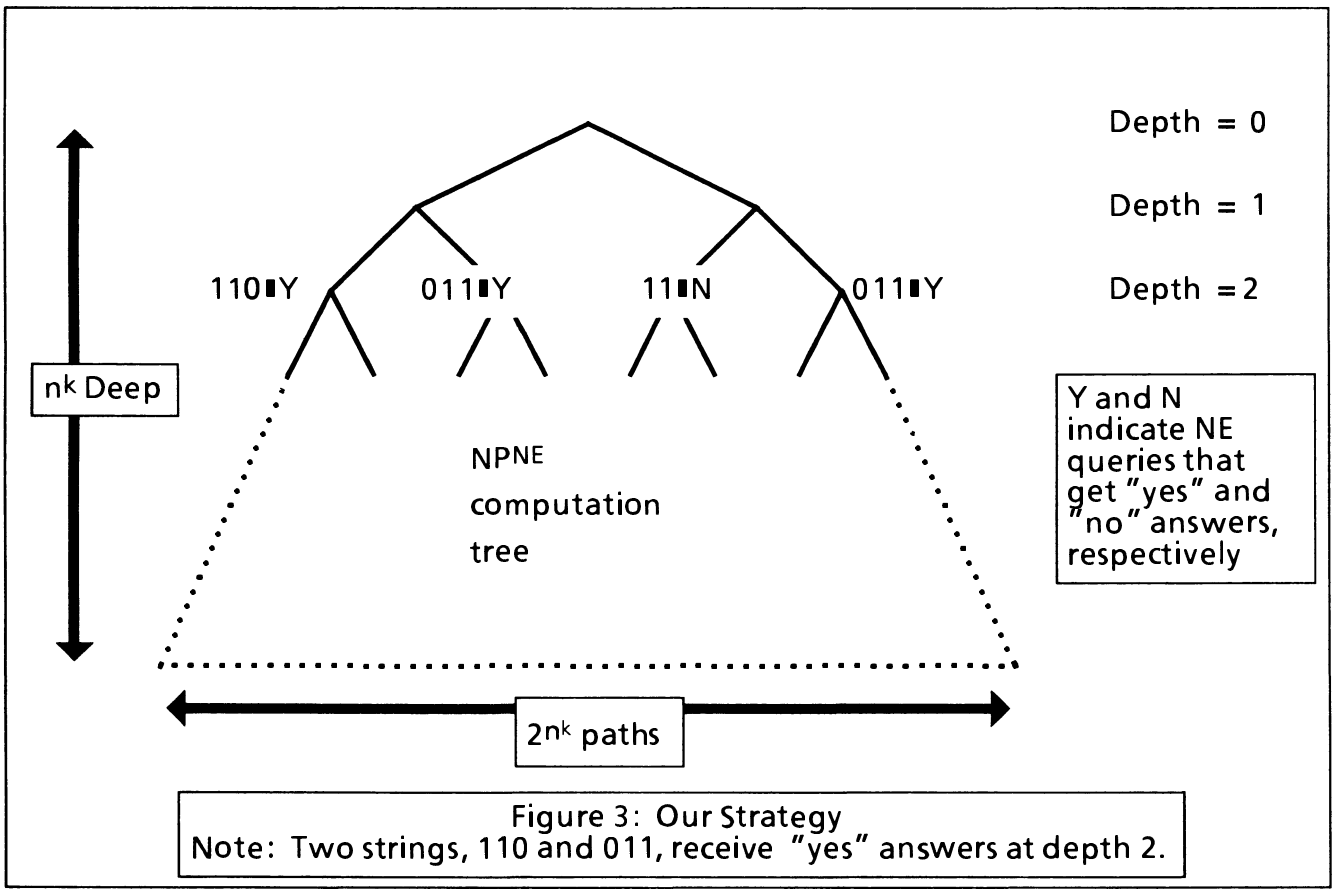
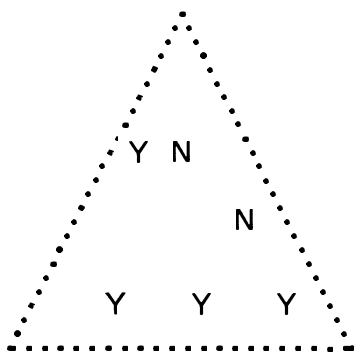
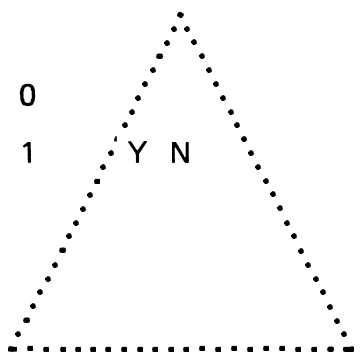


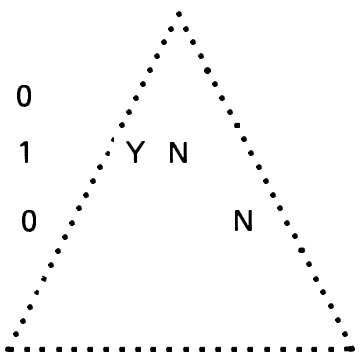
Figure 3: Our Strategy
Note: Two strings, 110 and 011, receive "yes" answers at depth 2.



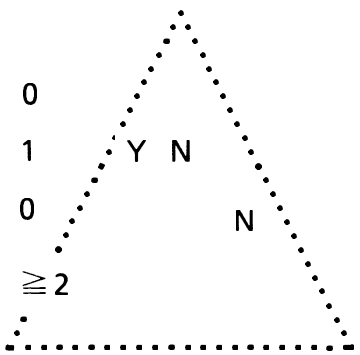
True NP^{NE} tree



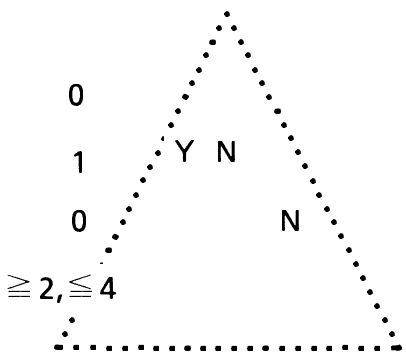
(1)



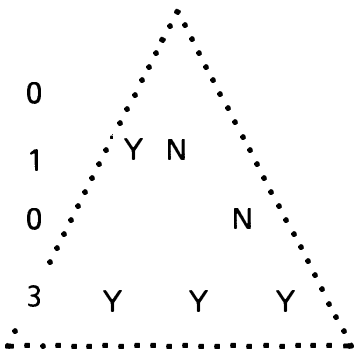
(2)



(3)



(4)



(5)

Figure 4: 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 yes queries on that level receiving a "yes" answer from the NE oracle.

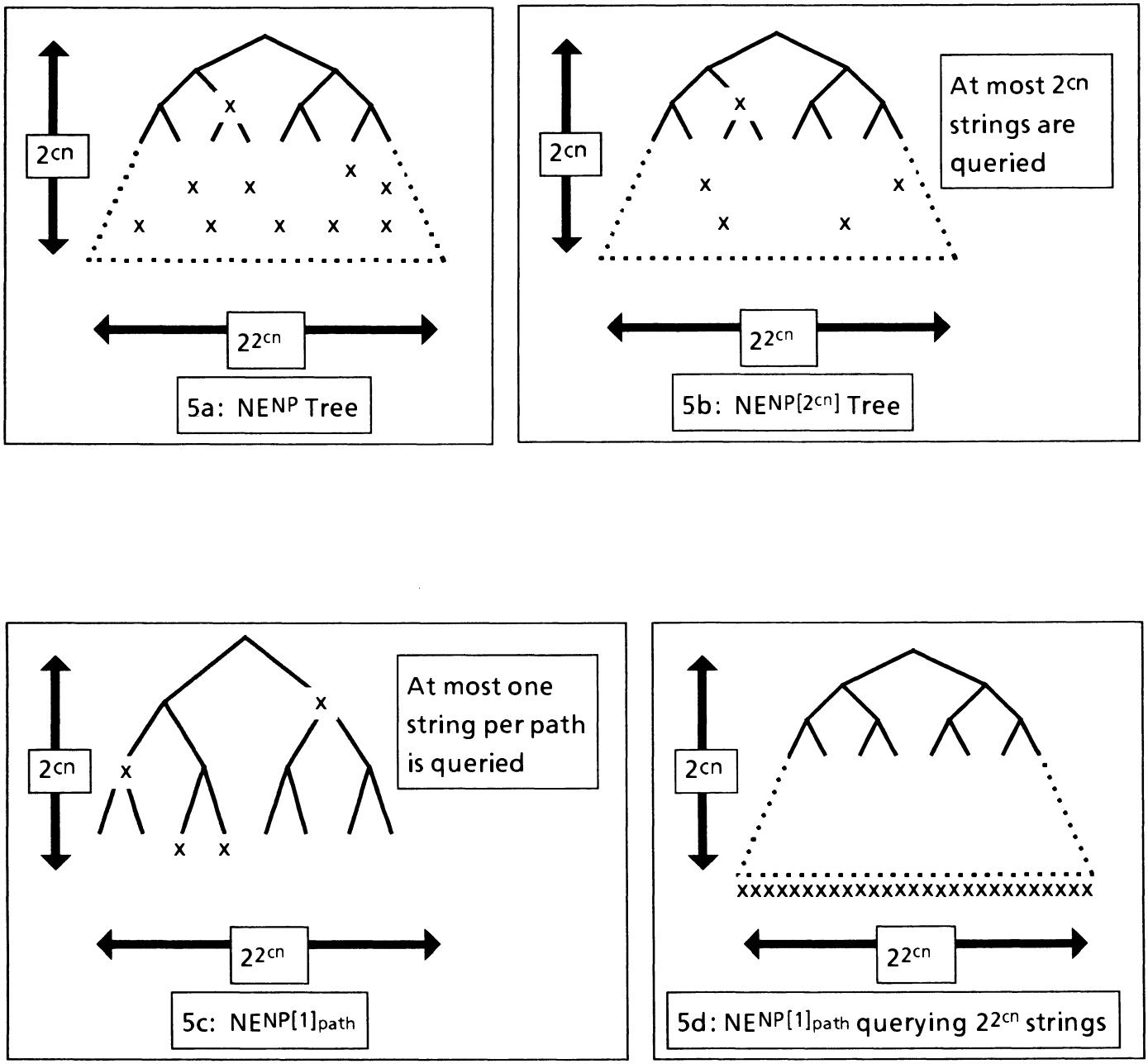


Figure 5: Quantitative Relativizations
Key: x = oracle query

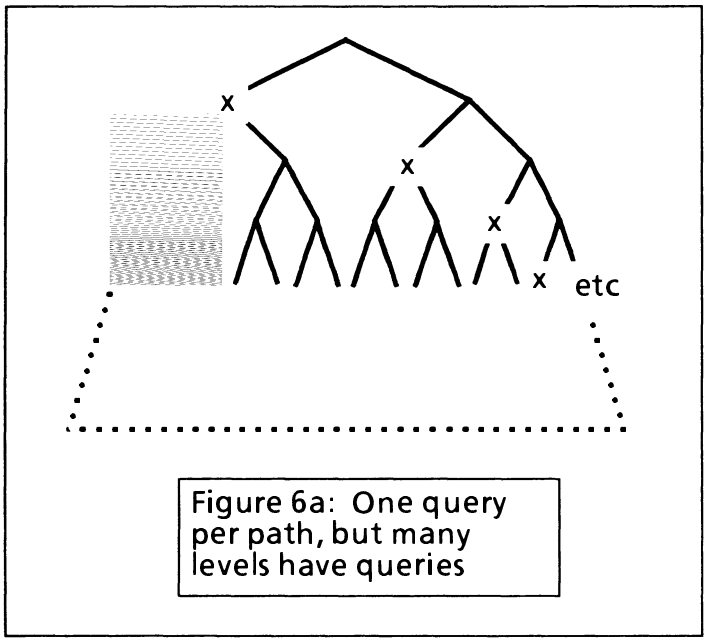


Figure 6a: One query per path, but many levels have queries

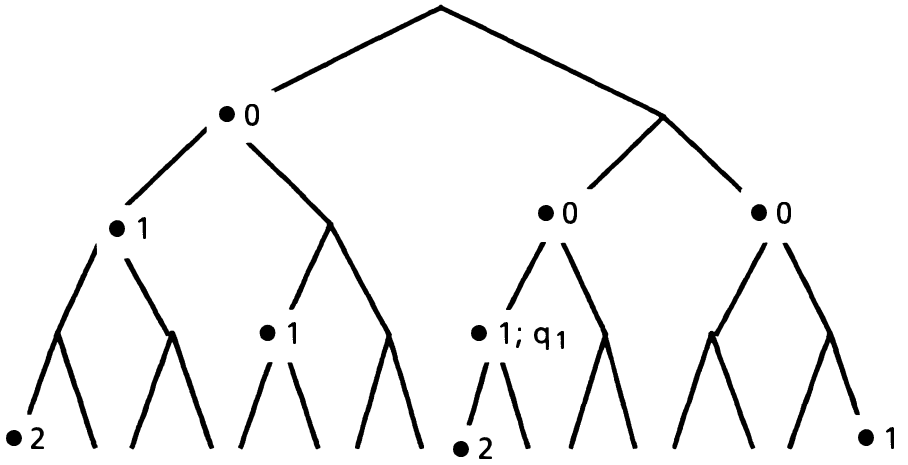


Figure 6b: Query Depths
Key: ● = Query; the numeric label indicates the query depth

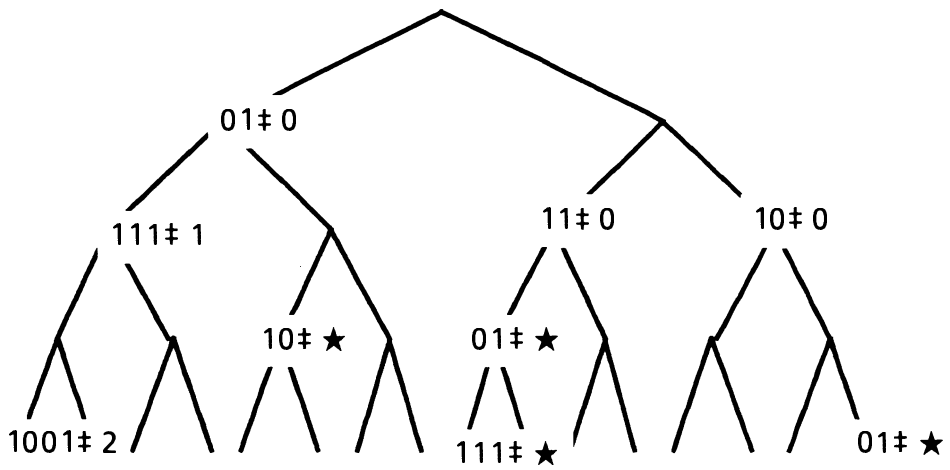


Figure 7: First-Occurance Depths
Key: s‡d means s = Query string; t = First-occurance depth of s; t = ★ indicates that this is not a first-occurance, and thus adds no depth.

References

[BGS75] T. Baker, J. Gill, and R. Solovay. Relativizations of the *P=?NP* question. *SIAM Journal on Computing*, 4(4):431–442, 1975.

[BH77] L. Berman and J. Hartmanis. On isomorphisms and density of NP and other complete sets. *SIAM Journal on Computing*, 6(2):305–322, 1977.

[BLS84] R. V. Book, T. J. Long, and A. L. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13(3):461–487, 1984.

[Boo81] R. V. Book. Bounded query machines: on NP and PSPACE. *Theoretical Computer Science*, 15:27–39, 1981.

[CH86a] J. Cai and Lane A. Hemachandra. The Boolean hierarchy: hardware over NP. In A. Selman, editor, *Structure in Complexity Theory*, pages 105–124, Springer-Verlag *Lecture Notes in Computer Science #223*, 1986.

[CH86b] J. Cai and Lane A. Hemachandra. *Exact Counting is as Easy as Approximate Counting.* Technical Report TR86-761, Cornell Department of Computer Science, June 1986.

[CKS81] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *JACM*, 26(1), 1981.

[CT86] P. Clote and G. Takeuti. Exponential time and bounded arithmetic. In A. Selman, editor, *Structure in Complexity Theory*, pages 125–143, Springer Verlag *Lecture Notes in Computer Science #223*, June 1986.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[HH86a] J. Hartmanis and Lane A. Hemachandra. Complexity classes without machines: on complete languages for UP. In *Automata, Languages, and*

Programming (ICALP 1986), Springer-Verlag *Lecture Notes in Computer Science*, 1986. To appear.

[HH86b] J. Hartmanis and Lane A. Hemachandra. On sparse oracles separating feasible complexity classes. In *STACS 1986: 3rd Annual Symnposium on Theoretical Aspects of Computer Science*, Springer-Verlag *Lecture Notes in Computer Science #210*, 1986.

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

[HS65] J. Hartmanis and R. Sterns. On the computational complexity of algorithms. *Trans. AMS*, 117:285–306, 1965.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

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

[Kel85] S. Kellogg. *Chicken Little.* W. Morrow, 1985.

[LLS75] R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.

[Lon85] T. J. Long. On restricting the size of oracles compared with restricting access to oracles. *SIAM Journal on Computing*, 14(3):585–597, 1985.

[Sel86] A. Selman, editor. *Structure in Complexity Theory.* Springer Verlag *Lecture Notes in Computer Science #223*, June 1986.

[SMB83] A. Selman, Xu Mei-Rui, and R. Book. Positive relativizations of complexity classes. *SIAM Journal on Computing*, 12:565–579, 1983.

[Sto77] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.