The Polynomial Time Hierarchy Collapses if the Boolean Hierarchy Collapses

Jim Kadin*

87-843
June 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

*Supported in part by NSF research grant DCR-8520597.

The Polynomial Time Hierarchy Collapses if the Boolean Hierarchy Collapses

Jim Kadin*
Cornell University

June 12, 1987

Abstract

The structure of the Boolean hierarchy (BH) is related to the polynomial time hierarchy (PH) by showing that if the BH collapses, then $\mathrm{PH} \subseteq \Delta_3^{\mathrm{P}}$.

1 Introduction

It was recently shown in [Kad87] that if $\mathrm{D^P} = \text{co-}\mathrm{D^P}$, then there exists a sparse set S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, there exist small NP machines for initial segments of $\overline{\mathrm{SAT}}$, and the polynomial time hierarchy (PH) [Sto77] collapses to $\mathrm{P^{NP^{NP}}}$ (Δ_3^{P}).

This paper generalizes that result to any level of the Boolean hierarchy (BH) [CH86,Wec85]. Thus if the BH is finite, the PH collapses to $\mathrm{P^{NP^{NP}}}$.

The BH is intertwined with the constant query bounded hierarchy, $\mathrm{P^{SAT[1]}}$, $\mathrm{P^{SAT[2]}}$, $\cdots$, where $\mathrm{P^{SAT[\mathit{k}]}}$ is the class of languages recognized by deterministic polynomial time machines that make at most k queries to an oracle for SAT on all inputs. Thus this same result holds for this hierarchy too.

*Supported in part by NSF research grant DCR-8520597.

The key idea of the proof is that if the k^{th} level of the BH collapses, then there is a polynomial time reduction g mapping co-NP(k) to NP(k). The function g induces reductions relative to sparse oracles for all the levels of the BH below k. Thus there exists a sparse set S and a polynomial time function f such that f^S reduces $\overline{\mathrm{SAT}}$ to SAT. This implies co-NP $\subseteq$ NP^S, and thus the PH collapses.

2 Preliminaries

We use capital letter N, possibly subscripted, to represent nondeterministic polynomial time machines.

For any machine N, $|N|$ is the size of the state transition table of N.

For any string x, $|x|$ is the length of x.

For any set of strings C, $C^{=n}$ is the set of strings in C of length n. $C^{\leq n}$ is the set of strings in C of length less then or equal to n.

We write $\|C\|$ for the cardinality of any set C.

For any two sets of strings B and C, the *disjoint union* of B and C is

$$B \oplus C \stackrel{\text{def}}{=} \{1x \mid x \in B\} \cup \{0x \mid x \in C\}.$$

The BH is defined in [CH86,Wec85] as:

$$\begin{aligned}
\mathrm{NP}(1) &\stackrel{\text{def}}{=} \mathrm{NP}, \\
\mathrm{NP}(2) &\stackrel{\text{def}}{=} \{L_1 \cap \overline{L_2} | L_i \in \mathrm{NP}\} \ (= \mathrm{D^P}), \\
\mathrm{NP}(3) &\stackrel{\text{def}}{=} \{(L_1 \cap \overline{L_2}) \cup L_3 | L_i \in \mathrm{NP}\}, \\
\mathrm{NP}(4) &\stackrel{\text{def}}{=} \{((L_1 \cap \overline{L_2}) \cup L_3) \cap \overline{L_4} | L_i \in \mathrm{NP}\}, \\
&\vdots
\end{aligned}$$

The class of languages whose complements are in NP(i), co-NP(i), is defined similarly:

$$\begin{aligned}
\text{co-NP}(1) &\stackrel{\text{def}}{=} \text{co-NP}, \\
\text{co-NP}(2) &\stackrel{\text{def}}{=} \{\overline{L_1} \cup L_2 | L_i \in \mathrm{NP}\} \ (= \text{co-D}^{\mathrm{P}}), \\
\text{co-NP}(3) &\stackrel{\text{def}}{=} \{(\overline{L_1} \cup L_2) \cap \overline{L_3} | L_i \in \mathrm{NP}\},
\end{aligned}$$

$$\text{co-NP}(4) \stackrel{\text{def}}{=} \{((\overline{L_1} \cup L_2) \cap \overline{L_3}) \cup L_4 | L_i \in \text{NP}\},$$
$$\vdots$$

Finally, BH $\stackrel{\text{def}}{=} \bigcup_k \text{NP}(k)$.

Many results about the BH are proved in [CH86,Wec85]. For instance, $\text{NP}(k) = \text{co-NP}(k) \iff \text{NP}(k) = \text{BH}$. To see that the structure of the BH is related to the structure of the PH, we need to understand the structure of the $\leq_m^P$-complete sets defined in [CH86,Wec85]. For each k, a complete language, $L_{\text{NP}(k)}$, can be defined in terms of SAT and $\overline{\text{SAT}}$.

$$\begin{aligned}
L_{\text{NP}(1)} &\stackrel{\text{def}}{=} \text{SAT}.\\
L_{\text{NP}(2)} &\stackrel{\text{def}}{=} \text{SAT} \,\&\, \overline{\text{SAT}}\\
&\stackrel{\text{def}}{=} \{\,(F_1, F_2) \,|\, F_1 \in \text{SAT and } F_2 \in \overline{\text{SAT}}\}.\\
L_{\text{NP}(3)} &\stackrel{\text{def}}{=} (\text{SAT} \,\&\, \overline{\text{SAT}}) \,|\, \text{SAT}\\
&\stackrel{\text{def}}{=} \{\,(F_1, F_2, F_3) \,|\, (F_1 \in \text{SAT and } F_2 \in \overline{\text{SAT}}) \text{ or } F_3 \in \text{SAT}\}.\\
L_{\text{NP}(4)} &\stackrel{\text{def}}{=} ((\text{SAT} \,\&\, \overline{\text{SAT}}) \,|\, \text{SAT}) \,\&\, \overline{\text{SAT}}\\
&\stackrel{\text{def}}{=} \{\,(F_1, F_2, F_3, F_4) \,|\, ((F_1 \in \text{SAT and } F_2 \in \overline{\text{SAT}}) \text{ or } F_3 \in \text{SAT}) \text{ and } F_4 \in \overline{\text{SAT}}\}.\\
&\vdots
\end{aligned}$$

$L_{\text{co-NP}(k)}$, a complete language for co-NP(k), can be defined similarly.

$$\begin{aligned}
L_{\text{co-NP}(1)} &\stackrel{\text{def}}{=} \overline{\text{SAT}}.\\
L_{\text{co-NP}(2)} &\stackrel{\text{def}}{=} \overline{\text{SAT}} \,|\, \text{SAT}\\
&\stackrel{\text{def}}{=} \{\,(F_1, F_2) \,|\, F_1 \in \overline{\text{SAT}} \text{ or } F_2 \in \text{SAT}\}.\\
L_{\text{co-NP}(3)} &\stackrel{\text{def}}{=} (\overline{\text{SAT}} \,|\, \text{SAT}) \,\&\, \overline{\text{SAT}}\\
&\stackrel{\text{def}}{=} \{\,(F_1, F_2, F_3) \,|\, (F_1 \in \overline{\text{SAT}} \text{ or } F_2 \in \text{SAT}) \text{ and } F_3 \in \overline{\text{SAT}}\}.\\
L_{\text{co-NP}(4)} &\stackrel{\text{def}}{=} ((\overline{\text{SAT}} \,|\, \text{SAT}) \,\&\, \overline{\text{SAT}}) \,|\, \text{SAT}\\
&\stackrel{\text{def}}{=} \{\,(F_1, F_2, F_3, F_4) \,|\, ((F_1 \in \overline{\text{SAT}} \text{ or } F_2 \in \text{SAT}) \text{ and } F_3 \in \overline{\text{SAT}}) \text{ or } F_4 \in \text{SAT}\}.\\
&\vdots
\end{aligned}$$

Note the structure of k-tuples in $L_{\mathrm{NP}(k)}$ and $L_{\text{co-NP}(k)}$. For all k, $L_{\mathrm{NP}(k)}$ has the structure "$(\cdots)$ & $\overline{\mathrm{SAT}}$", and $L_{\text{co-NP}(k)}$ has the structure "$(\cdots) \mid \mathrm{SAT}$" or vice versa. It will become clear that this is the key that relates the collapse of the BH to the collapse of the PH

We will make the technical restriction that for each k-tuple in $L_{\mathrm{NP}(k)}$ and $L_{\text{co-NP}(k)}$, all k formulas have the same length. This restricted version is still $\leq_m^{\mathrm{P}}$-complete since the shorter formulas in any k-tuple can be padded up to the length of the longest formula in time that is polynomial in the length of the original k-tuple.

Definition *$\forall k, \forall n$, $L_{\mathrm{NP}(k),n}$ is the subset of $L_{\mathrm{NP}(k)}$ where each k-tuple only contains formulas of length n.*

For all k, $L_{\mathrm{NP}(k)}$ and $L_{\text{co-NP}(k)}$ are complements. Thus any function f that reduces $L_{\mathrm{NP}(k)}$ to $L_{\text{co-NP}(k)}$ also reduces $L_{\text{co-NP}(k)}$ to $L_{\mathrm{NP}(k)}$. Such an f also reduces $L_{\mathrm{NP}(k),n}$ to $L_{\text{co-NP}(k)}$ and $L_{\text{co-NP}(k),n}$ to $L_{\mathrm{NP}(k)}$.

Definition *A set of strings S is* sparse *if there exists a polynomial p such that $\forall n$ the number of strings in S of length n is less than $p(n)$, i.e. $\| S^{=n} \| \leq p(n)$.*

There is a standard technique of using the prefixes of strings in a sparse set to find the strings in the set. For any sparse set S, let

$$\mathrm{prefix}(S) \stackrel{\mathrm{def}}{=} \{y\#^i \mid \exists x \in S, |x| = |y| + i, \text{ and } y \text{ is a prefix of } x\}.$$

For every length n, prefix(S) contains n times as many strings of length n as S does. Therefore prefix(S) is sparse if S is. A deterministic polynomial time machine with an oracle for prefix(S) can generate all the strings in $S^{\leq n}$ on input 1^n. The machine builds up each string one character at a time by asking the oracle about longer and longer prefixes. A machine can actually generate all the strings in prefix(S) up to a given length this way. We use the phrase *closing S under prefixes* to mean adding all the strings in prefix(S) to S.

Definition *A set S is* P^B-printable *for an oracle B if there exists a P^B machine that on input 1^n prints all the strings in $S^{\leq n}$.*

Thus if a sparse set S is closed under prefixes, then S is P^S-printable.

3 Main Result

The basic idea is that if for some k, $L_{\mathrm{NP}(k)}$ is polynomial time reducible to $L_{\text{co-NP}(k)}$, then NP can be reduced to co-NP by a polynomial time function that accesses a sparse oracle.

Theorem 1 *$\forall k > 1$ if $\exists S_k, g_k$ where S_k is sparse and $g_k^{S_k}$ is a polynomial time reduction of $L_{\mathrm{NP}(k)}$ to $L_{\text{co-NP}(k)}$, then $\exists S_{k-1}, g_{k-1}$ where S_{k-1} is sparse and $g_{k-1}^{S_{k-1}}$ is a polynomial time reduction of $L_{\mathrm{NP}(k-1)}$ to $L_{\text{co-NP}(k-1)}$.*

Proof: Suppose $g_k^{S_k}$ reduces $L_{\mathrm{NP}(k)}$ to $L_{\text{co-NP}(k)}$. Assume without loss of generality that S_k is closed under prefixes so that S_k is P^{S_k}-printable.

For each length n, there are two cases to distinguish. In each case we will show how g_{k-1} works for strings of length n and what strings need to be put into $S_{k-1}^{=n}$.

To understand the first case, consider a formula F of length n. Suppose there exist formulas $F_1, \cdots, F_{k-1}$ of length n such that

$$\begin{aligned} g_k^{S_k}(F_1, \cdots, F_{k-1}, F) &= (G_1, \cdots, G_{k-1}, G), \quad \text{and} \\ G &\in \mathrm{SAT}. \end{aligned}$$

Then because of the structure of $L_{\mathrm{NP}(k)}$ and $L_{\text{co-NP}(k)}$, F must be in $\overline{\mathrm{SAT}}$. If k is even,

$$G \in \mathrm{SAT} \Longrightarrow (G_1, \cdots, G_{k-1}, G) \in L_{\text{co-NP}(k)}.$$

Thus since $g_k^{S_k}$ is a reduction, $(F_1, \cdots, F_{k-1}, F) \in L_{\mathrm{NP}(k)}$, but this implies $F \in \overline{\mathrm{SAT}}$. If k is odd, the same reasoning holds with $\mathrm{NP}(k)$ and co-$\mathrm{NP}(k)$ reversed. Therefore if there exists such $F_1, \cdots, F_{k-1}$, there is an NP^{S_k} algorithm for recognizing F as unsatisfiable. On input F, guess $F_1, \cdots, F_{k-1}$, compute $g_k^{S_k}(F_1, \cdots, F_{k-1}, F)$, and accept if the last component of the output of g_k is satisfiable.

Call the machine that executes this NP^{S_k} algorithm $N_{easy,k}$. Since $g_k^{S_k}$ reduces $L_{\mathrm{NP}(k)}$ to $L_{\text{co-NP}(k)}$,

$$\mathrm{L}(N_{easy,k}^{S_k}) \subseteq \overline{\mathrm{SAT}}.$$

The unsatisfiable formulas accepted by $N_{easy,k}^{S_k}$ are "easy", or more precisely, *$g_k^{S_k}$-easy*.

Definition *A formula F is $g_k^{S_k}$-easy if $\exists F_1, \cdots, F_{k-1}$ of length $|F|$ such that*

$$g_k^{S_k}(F_1, \cdots, F_{k-1}, F) = (G_1, \cdots, G_{k-1}, G)$$

where $G \in$ SAT.

Since S_k is P^{S_k}-printable, there exists a polynomial time reduction r such that r^{S_k} reduces $\mathrm{L}(N_{easy,k}^{S_k})$ to SAT. On input F, r^{S_k} first generates the subset S'_k of strings in S_k up to the length $N_{easy,k}$ can ask on input F. Then since it is an NP question whether or not $N_{easy,k}(F)$ accepts using oracle S'_k, r can map $\langle N_{easy,k}, F, S'_k \rangle$ to a formula that is satisfiable if and only if $N_{easy,k}$ accepts.

Case 1: Suppose all the unsatisfiable formulas of length n are $g_k^{S_k}$-easy. Then

$$\mathrm{L}(N_{easy,k}^{S_k})^{=n} = \overline{\mathrm{SAT}}^{=n},$$

and $\forall F$ of length n

$$r^{S_k}(F) \in \mathrm{SAT} \Longleftrightarrow F \in \overline{\mathrm{SAT}}.$$

Thus for such n's, r^{S_k} can be used to reduce $L_{\text{co-NP}(k-1),n}$ to SAT and thus to $L_{\mathrm{NP}(k-1)}$. Call the reduction that uses r in this way $g_{k,easy}^{S_k}$.
End of Case 1

If it is not the case that all formulas of length n are $g_k^{S_k}$-easy, then at least one formula must be "hard".

Definition *A formula F of length n is $g_k^{S_k}$-hard if $F \in \overline{\mathrm{SAT}}$ and $\forall F_1, \cdots, F_{k-1}$ where $|F_i| = n$,*

$$g_k^{S_k}(F_1, \cdots, F_{k-1}, F) = (G_1, \cdots, G_{k-1}, G)$$

where $G \in \overline{\mathrm{SAT}}$.

Case 2: Suppose there exists a formula F of length n such that F is $g_k^{S_k}$-hard. If k is even, then for all k-tuples of formulas of length n, $F_1, \cdots, F_{k-1}$,

$$(F_1, \cdots, F_{k-1}) \in L_{\mathrm{NP}(k-1),n} \Longleftrightarrow (F_1, \cdots, F_{k-1}, F) \in L_{\mathrm{NP}(k),n}$$

since $F \in \overline{\mathrm{SAT}}$. Since $g_k^{S_k}$ is a reduction,

$$(F_1, \cdots, F_{k-1}, F) \in L_{\mathrm{NP}(k),n} \Longleftrightarrow (G_1, \cdots, G_{k-1}, G) \in L_{\text{co-NP}(k)}.$$

Since F is $g_k^{S_k}$-hard, $G \notin \mathrm{SAT}$, thus

$$(G_1, \cdots, G_{k-1}, G) \in L_{\text{co-NP}(k)} \Longleftrightarrow (G_1, \cdots, G_{k-1}) \in L_{\text{co-NP}(k-1)}.$$

If k is odd, the same reasoning holds with NP(i) and co-NP(i) reversed. Thus there is a polynomial time function which using S_k can reduce $L_{\mathrm{NP}(k-1),n}$ to $L_{\text{co-NP}(k-1)}$ (and $L_{\text{co-NP}(k-1),n}$ to $L_{\mathrm{NP}(k-1)}$). Call this function $g_k[F]$. On input $F_1, \cdots, F_{k-1}$, $g_k[F]$ computes $g_k^{S_k}(F_1, \cdots, F_{k-1}, F)$ and outputs the first $k-1$ components.
End of Case 2

In summary, if all the strings of length n are $g_k^{S_k}$-easy, then $g_{k,easy}^{S_k}$ reduces $L_{\text{co-NP}(k-1),n}$ to $L_{\mathrm{NP}(k-1)}$. If some formula F of length n is $g_k^{S_k}$-hard, then $g_k[F]^{S_k}$ reduces $L_{\text{co-NP}(k-1),n}$ to $L_{\mathrm{NP}(k-1)}$.

Thus S_{k-1} can be defined as S_k with some extra strings added to indicate which reduction works for each length n. For each n, put the lexicographically least $g_k^{S_k}$-hard string of length n into the set S_{hard}, and close S_{hard} under prefixes. Let

$$S' \stackrel{\text{def}}{=} S_{hard} \cup \{0^n \mid \text{there are no } g_k^{S_k}\text{-hard strings of length } n\}$$

(we can assume that 0^n does not represent a valid Boolean formula). Let

$$S_{k-1} \stackrel{\text{def}}{=} S_k \oplus S'.$$

S_{k-1} is sparse since S_k and S' are both sparse.

Then $g_{k-1}^{S_{k-1}}$ on input $F_1, \cdots, F_{k-1}$, (all of length n) works as follows. First ask if $0^n \in S'$. If the answer is yes, output $g_{k,easy}^{S_k}(F_1, \cdots, F_{k-1})$. If the answer is no, by prefixes generate F, the $g_k^{S_k}$-hard string of length n, and output $g_k[F]^{S_k}(F_1, \cdots, F_{k-1})$.

For each n, $g_{k-1}^{S_{k-1}}$ reduces $L_{\text{co-NP}(k-1),n}$ to $L_{\mathrm{NP}(k-1)}$ and thus reduces $L_{\text{co-NP}(k-1)}$ to $L_{\mathrm{NP}(k-1)}$ and vice versa. □

Corollary 2 *If $\exists k$ with* NP(k) = co-NP(k), *then $\exists S, g$ such that S is sparse and g^S is a polynomial time reduction from $\overline{\mathrm{SAT}}$ to* SAT.

Proof: If NP(k) = co-NP(k), then there exists a polynomial time reduction (that needs no oracle) from $L_{\mathrm{NP}(k)}$ to $L_{\text{co-NP}(k)}$. By k repetitions of the theorem above, we can construct S and g. □

Lemma 3 *If S is sparse and g^S is a polynomial time reduction from a language L to* SAT*, then $L \in \mathrm{NP}^S$.*

Proof: $L = \mathrm{L}(N^S)$ where on input x, N computes $g^S(x) = y$ and accepts if $y \in \mathrm{SAT}$. □

Thus we have:

Theorem 4 *If $\exists k$ with* NP(k) = co-NP(k)*, then $\exists$ a sparse set S such that* $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$.

At this point we can conclude from results of Yap that PH $\subseteq \Sigma_3^{\mathrm{P}}$ ($\mathrm{NP}^{\mathrm{NP}^{\mathrm{NP}}}$) [Yap83]. With a little more work, we can push it down to Δ_3^{P} ($\mathrm{P}^{\mathrm{NP}^{\mathrm{NP}}}$).

From the results in [Kad87], we know that if there exists a sparse set S that is $\mathrm{P}^{\mathrm{NP}^{\mathrm{NP}}}$-printable and $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, then PH $\subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{NP}}}$. We will show that there exists such an S.

The S that we have in mind is basically the set built up by applications of Theorem 1, but we can simplify that set a bit.

Suppose NP(k) = co-NP(k). Then by applying the theorem, we can back down the BH from NP(k) to NP building up a sparse set S and a polynomial time function g such that g^S reduces $\overline{\mathrm{SAT}}$ to SAT. In the proof, we define the set S_{hard} consisting of the lexicographically least hard string of each length. We close S_{hard} under prefixes so that the deterministic reduction functions can generate the hard strings. Once we put the reduction functions into an NP machine as in Lemma 3, the prefixes are not necessary. The NP machine can guess a string and verify with the oracle that the string is hard. Thus if NP(k) = co-NP(k), define

$$
\begin{aligned}
S_k &\stackrel{\mathrm{def}}{=} \phi, \\
S_{i-1} &\stackrel{\mathrm{def}}{=} S_i \oplus (\{0^n \mid \text{all formulas in } \overline{\mathrm{SAT}}^{=n} \text{ are } g_i^{S_i}\text{-easy}\ \} \cup \\
&\qquad\qquad \{x \mid x \text{ is the lexicographically least } g_i^{S_i}\text{-hard} \\
&\qquad\qquad\qquad \text{string of length } |x|\}).
\end{aligned}
$$

Then $S \stackrel{\text{def}}{=} S_{k-1} \oplus \cdots \oplus S_1$ is a sparse set such that $\overline{\text{SAT}} \in \text{NP}^S$. Since each S_i for $1 \leq i \leq k-1$ contains exactly one string of each length, S contains $k-1$ strings of each length.

Lemma 5 *$\forall k$, if* $\text{NP}(k) = \text{co-NP}(k)$*, then the set S defined above is* $\text{P}^{\text{NP}^{\text{NP}}}$*-printable.*

Proof: We will show that S_{k-1} is $\text{P}^{\text{NP}^{\text{NP}}}$-printable, and thus inductively, S will be too.

Let g_k be the polynomial time reduction of $L_{\text{NP}(k)}$ to $L_{\text{co-NP}(k)}$. The set of g_k-hard strings is in co-NP since it is the complement of the union of SAT and the set of g_k-easy strings. This implies that the set of prefixes of hard strings,

$$\{y\#^i \mid \exists F \text{ of length } |y| + i \text{ with } y \text{ a prefix of } F, \text{ and } F \text{ is hard}\},$$

is in NP^{SAT}.

Therefore a $\text{P}^{\text{NP}^{\text{NP}}}$ machine on input 1^n can determine whether or not there are any hard strings of length n by asking whether there are any strings of length n in this prefix set. If there are strings, it can generate the lexicographically least one by the standard prefix technique. $\Box$

Theorem 6 *$\forall k$, if* $\text{NP}(k) = \text{co-NP}(k)$*, there exists a* $\text{P}^{\text{NP}^{\text{NP}}}$*-printable, sparse set S containing exactly $k-1$ strings of each length such that* $\overline{\text{SAT}} \in \text{NP}^S$.

Corollary 7 *If the* BH *collapses, then* $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}}$ $(= \Delta_3^{\text{P}})$.

Corollary 8 *If the constant query bounded hierarchy collapses, then* $\text{PH} \subseteq \text{P}^{\text{NP}^{\text{NP}}}$.

This work can be viewed as a study of oracle access mechanisms within the PH. The results given above generalize for every Δ_i^{P} in the PH. Recall

$$\Delta_i^{\text{P}} \stackrel{\text{def}}{=} \text{P}^{\Sigma_{i-1}^{\text{P}}}.$$

A Boolean hierarchy and a constant query bounded hierarchy, $\text{P}^{\Sigma_{i-1}^{\text{P}}[k]}$, can be defined within each Δ_i^{P}. If these hierarchies collapse, there exists a sparse set S such that $\Pi^{\mathrm{P}}_{i-1} \subseteq \Sigma^{\mathrm{P},S}_{i-1}$, and this collapses the PH to $\Delta^{\mathrm{P}}_{i+1}$. In other words, for any Δ^{P}_{i} and k, if k queries are as powerful as $k+1$ queries, the PH collapses.

In contrast, for each Σ^{P}_{i} in the PH, one query is enough. Recall

$$\Sigma^{\mathrm{P}}_{i} \stackrel{\mathrm{def}}{=} \mathrm{NP}^{\Sigma^{\mathrm{P}}_{i-1}}.$$

Lane Hemachandra has recently shown that for all i, $\mathrm{NP}^{\Sigma^{\mathrm{P}}_{i}} = \mathrm{NP}^{\Sigma^{\mathrm{P}}_{i}[1]}$ [Hem87]. This is a generalization of the result in [FSS84] that $\mathrm{NP}^{\mathrm{NP}} = \mathrm{NP}^{\mathrm{NP}[1]}$. Thus for all the NP levels of the PH, one query is enough, yet if one query is enough at any of the P levels, then the PH collapses.

This work can also be viewed as a downward structural result. If the BH collapses or if the PH collapses to the BH, then the languages in co-NP are forced to have a certain structure.

A language L has *small* NP *machines* if there exists a sequence of NP machines $\{N_i\}$ such that:

1. $\mathrm{L}(N_i) = L^{\leq i}$,
2. there is a polynomial p_{size} such that for all i, $|N_i| \leq p_{size}(i)$,
3. there is a single polynomial that bounds the running time of all the N_i.

If L has small NP machines, then nondeterminism is somehow useful in recognizing L. In [Kad87] it was shown that L has small NP machines if and only if there exists a sparse set S such that $L \in \mathrm{NP}^S$. Thus the following corollary is true.

Corollary 9 *$\forall k$, if* $\mathrm{NP}(k) = \text{co-NP}(k)$, *then every language in* co-NP *has small* NP *machines.*

Acknowledgements

I am grateful to Professor Hartmanis for his support and guidance. It was partly through discussions with him that the formulation of the collapse in terms of sparse sets evolved. I am also grateful to Stuart Allen, Lane Hemachandra, Steve Mitchell, and Geoff Smith for helpful discussions.

References

[CH86] J. Cai and L. Hemachandra. The Boolean hierarchy: hardware over NP. In *Structure in Complexity Theory*, pages 105–124, Springer-Verlag *Lecture Notes in Computer Science #223*, 1986.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the Polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[Hem87] L. Hemachandra. 1987. Private communication.

[Kad87] J. Kadin. *Is One* NP *Question as Powerful as Two?* Technical Report TR 87-842, Cornell Department of Computer Science, June 1987.

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

[Wec85] G. Wechsung. On the Boolean closure of NP. In *Proc. of the 1985 International Conference on Fundamentals of Computation Theory*, pages 485–493, Springer-Verlag *Lecture Notes in Computer Science* , 1985.

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