The Boolean Hierarchy: Hardware over NP

Jin-yi Cai*
Lane Hemachandra†

TR 85-724
December 1985

Department of Computer Science
Cornell University
Ithaca, NY 14853

* Research supported by NSF grant DCR-8301766.
† Research supported by a Fannie and John Hertz Foundation Fellowship and NSF grant DCR-8301766.

The Boolean Hierarchy: Hardware over NP

Jin-yi Cai*
Lane Hemachandra#

Cornell Computer Science Dept.
Ithaca, NY 14853

Draft: June 1985
Revised: December 1985

ABSTRACT

In this paper, we study the complexity of sets formed by boolean operations ($\bigcup$, $\bigcap$, and complementation) on NP sets. These are the sets accepted by trees of hardware with NP predicates as leaves, and together form the boolean hierarchy.

We present many results about the boolean hierarchy: separation and immunity results, complete languages, upward separations, connections to sparse oracles for NP, and structural asymmetries between complementary classes. Some results present new ideas and techniques. Others put previous results about NP and D^P in a richer perspective. Throughout, we emphasize the structure of the boolean hierarchy and its relations with more common classes.

*Research supported by NSF grant DCR-8301766.

#Research supported by a Fannie and John Hertz Foundation Fellowship and NSF grant DCR-8301766.

Contents

1. Introduction and Overview of Results

1.1. Introduction

NP has long been the collective obsession of computer scientists. Yet some have turned from her siren song to study less alluring classes. D^P, the closure of NP $\bigcup$ coNP under intersections, has recently become immensely fashionable [PY82][PW85][CM85]. In this paper, we study the natural completion of such structures— the closure of NP under boolean operations.

This closure, the boolean hierarchy (BH), has a clear "physical" interpretation. The sets in the boolean hierarchy are exactly those representable by *hardware over NP*. Each boolean hierarchy language is accepted by a hardware tree connecting NP machines [Figure 1].

We present many results about the boolean hierarchy: separations and immunity results, complete languages, connections with upward separations and sparse oracles for NP, and structural asymmetries between complementary classes. Some results present new ideas and techniques. Others put previous results about NP and D^P in a richer perspective. Throughout, we emphasize the structure of the boolean hierarchy and its relations with more common classes.

1.2. Definitions

Section 2 defines the levels of the boolean hierarchy [Figure 5]. Each level of the boolean hierarchy consists of sets represented by a certain fixed structure of boolean operators on NP sets. Sets in the boolean hierarchy have normal forms. For example, each can be written as a finite union of D^P sets.

$$NP(0) = P$$

$$NP(1) = NP$$

$$NP(2) = D^P = \{L_1 \cap \overline{L_2} \mid L_1, L_2 \in NP\}$$

$$NP(3) = \{(L_1 \cap \overline{L_2}) \cup L_3 \mid L_1, L_2, L_3 \in NP\}$$

$$NP(4) = \{((L_1 \cap \overline{L_2}) \cup L_3) \cap \overline{L_4} \mid L_1, L_2, L_3, L_4 \in NP\}$$

$$BH = \bigcup_{i>0} NP(i)$$

$$coNP(i) = \{ S \mid \overline{S} \in NP(i) \}.$$

Lemma: The following are equivalent:

1) $T \in BH$

2) T is a finite union of D^P sets.

3) T is a finite intersection of coD^P sets.

4) T is in the boolean closure of NP.

5) $T \leq^P_{btt}$ SAT.

1.3. Relativized Separations of the Boolean Hierarchy

Section 3 explores the relativized structure of the boolean hierarchy and presents strong separation results. We show relativized worlds in which the boolean hierarchy is infinite. This result is intriguing, as oracles making the polynomial hierarchy infinite took many years to discover [Y85].

To prove this result, we introduce the "sawing" technique. This method forms the basis of all our relativized results in the boolean hierarchy. The novel technique places successively more stringent requirements on oracle extensions, while seeking to *force* a desired behavior.

Indeed, this method can be tailored to address the interesting area of probabilistic separations. It is well known that $P^A \neq NP^A$ with probability one (over randomly chosen A) [BG81]. One can show that the boolean hierarchy is infinite with probability one; the proof will appear in a subsequent paper.

The proof of [H84, Theorem 2] shows that the polynomial hierarchy can be collapsed to D^P while separating NP from coNP, with appropriate relativization. For each k, we display worlds where the boolean hierarchy extends exactly k levels, with PSPACE collapsing to the $k+1^{st}$ level. In such worlds, understanding the boolean hierarchy is a prerequisite to understanding the structure of standard feasibly computable complexity classes.

Theorem: 1) $(\exists \text{ recursive } A)\ (\forall i \geq 1)\ [NP^A(i) \neq NP^A(i+1)]$.

2) $(\forall i \geq 0)\ (\exists \text{ recursive } A)\ [NP^A(1) \neq NP^A(2) \neq \cdots \neq NP^A(i) \neq NP^A(i+1) = PSPACE^A]$.

We now sketch the proof of part one of this theorem. A detailed and stronger version of this proof appears in Section 3. However, by carefully studying the simple sketch below, the reader will be introduced to the sawing technique that, in more complex variations, is the central tool of this paper.

Proof Sketch for part 1:

In the standard way, if we can separate a generic level by diagonalization, then by interleaving diagonalizations we can construct a single oracle simultaneously separating the entire hierarchy. For concreteness, we show an oracle A so $NP^A(17) \neq coNP^A(17)$. In particular, we show that L_A, defined below, is in $coNP^A(17) - NP^A(17)$.

$$L_A = \{0^{17j} \mid (\forall x, |x|=17j)[x \notin A] \text{ AND } \{(\exists x, |x|=17j+1)[x \in A] \text{ OR } \{(\forall x, |x|=17j+2)[x \notin A] \text{ AND } \{ \cdots \text{ OR } \{(\forall x, |x|=17j+16)[x \notin A]\} \cdots \}\}\} \}.$$

Stage 0: Let $A_0 = \varnothing$. We'll have $A = \cup A_i$.

Stage j: Let H be the jth NP(17) machine. It is composed of nine NP machines (N_i's) and eight coNP machines (A_i's) so that (for simplicity, we use low numbers for the machines instead of indexing them):

$$L(H^A) =_{\text{def}} L(N_1^A) \cup \{L(A_2^A) \cap \{L(N_3^A) \cup \{ \cdots \cup \{L(A_{16}^A) \cap L(N_{17}^A)\} \cdots \}\}\}$$

Choose w_j large. We now use our method of successive restriction; we see-saw between pressuring NP machines to accept and pressuring coNP machines to reject. Throughout, we obey the constraint that in stage j we never add to our oracle any string of length less than w_j.

Case 1: There exists an extension B of A_{j-1} so $N_1^B(0^{17w_j})$ accepts (otherwise go to case 2). Freeze into A_j an accepting path of the machine, thus we know that $H^A(0^{17w_j})$ accepts. Add an untouched length $17w_j$ string to A_j. Now we have assured that $0^{17w_j} \in L(H^A) - L_A$. Thus, machine H fails to accept L_A.

Case 2k $(1 \le k \le 8)$*:* There exists an extension B of A_{j-1} so B contains no size $17w_j, \ldots, 17w_j+2k-2$ strings and $A_1^B(0^{17w_j})$ rejects (otherwise go to case $2k+1$). Freeze into A_j a rejecting path of the machine, thus we know that $H^A(0^{17w_j})$ rejects. Add an untouched length $17w_j+2k-1$ string to A_j. Now we have assured that $0^{17w_j} \in L_A - L(H^A)$. Thus, machine H fails to accept L_A.

Case 2k+1 $(1 \le k \le 8)$*:* There exists an extension B of A_{j-1} so B contains no size $17w_j, 17w_j+1, \ldots, 17w_j+2k-1$ strings and $N_1^B(0^{17w_j})$ accepts (otherwise go to case $2k+2$).

Freeze into A_j an accepting path of the machine, thus we know that $H^A(0^{17w_j})$ accepts. Add an untouched length $17w_j+2k$ string to A_j. Now we have assured that $0^{17w_j} \in L(H^A)-L_A$. Thus, machine H fails to accept L_A.

Case 18: If we've made it to this case, we know that for any extension B of A_{j-1} that has no length $17w_j, 17w_j+1, \cdots, 17w_j+15$ strings, $L(H^B)$ rejects 0^{17w_k}. Let $A_j := A_{j-1}$, which safely avoids strings for the forbidden lengths. Thus $0^{17w_j} \in L_A - L(H^A)$.

In each case, we've seen that $L(H^A) \neq L_A$. Thus, by the preceding discussion, we are done.

▸.

1.4. Immunity Results

Section 4 studies the immunity structure of the boolean hierarchy. With appropriate relativization, we display a Δ_2 set that is BH-immune. Also, we show that no set in the boolean hierarchy can be either NP bi-immune or coNP bi-immune. Surprisingly, no boolean hierarchy set is NP(2)-immune, yet there are worlds where NP(2) has a coNP(2)-immune set. Thus NP(2) and coNP(2) are structurally asymmetric. Such asymmetry between complementary classes is rare in the realm of feasible computations. The only previous example is the asymmetry between NP and coNP of [HSI83b], which is extended in Section 5 of this paper.

Theorem: There is a recursive oracle A for which Δ_2^A has a BH^A-immune set.

Theorem: 1) No set in the boolean hierarchy is NP bi-immune or coNP bi-immune.

2) No set in the boolean hierarchy is NP(2)-immune, yet there is a relativized world with coNP(2)-immune sets.

1.5. Sparse Sets in the Boolean Hierarchy and E=?NE

[HSI83b] links the behavior of sparse sets in NP to the E=NE question. Section 5 shows that E=NE forces sparse sets in odd levels of the boolean hierarchy down one level. This result is the best possible among results holding in all relativizations; our "candy cane" relativization (below) alternates stripes with and without sparse sets throughout the boolean hierarchy [Figure 6b]. Indeed, since this oracle leaves sparse sets in $\mathrm{coNP}(2k+1)-\mathrm{NP}(2k+1)$ but not $\mathrm{NP}(2k+1)-\mathrm{coNP}(2k+1)$, we have an infinite set of structurally asymmetric complementary classes— the levels of the boolean hierarchy.

Theorem: If E=NE, then $(\forall k)[\mathrm{NP}(2k+1)-\mathrm{NP}(2k)$ has no sparse sets].

Theorem (Candy Cane Relativization): $(\exists A)$ $(\forall k \geq 0)$ $[\mathrm{NP}^A(2k+1)-\mathrm{NP}^A(2k)$ has no sparse sets, yet $\mathrm{NP}^A(2k+2)-\mathrm{NP}^A(2k+1)$ has sparse sets].

We introduce a notion of capturability analogous to the concept of immunity. A set is *capturable* if it is a subset of some sparse set in NP. When E=NE, all capturable sets in the boolean hierarchy fall into P.

Theorem: If E=NE, then BH−P has no capturable sets.

1.6. Complete Languages for the Boolean Hierarchy

Section 6 presents natural complete languages for the levels of the boolean hierarchy. First, we establish variations of SAT as canonical complete languages. E.g., $\mathrm{SAT}(3) = \{<f_1,f_2,f_3> \mid (f_1 \in \mathrm{SAT} \wedge f_2 \in \mathrm{UNSAT}) \vee f_3 \in \mathrm{SAT}\}$ is NP(3)-complete. By reductions from these languages, we show variations of vertex cover and chromatic number to be complete for the levels of the boolean hierarchy.

Theorem: The levels of the boolean hierarchy have natural complete languages.

1.7. Counting Classes

Counting classes, such as US [BG82], seem tempting as natural complete sets for the boolean hierarchy. Let S be a finite or cofinite set of natural numbers. We can define its associated counting class, CP_S, as the set of languages accepted by a nondeterministic polynomial time machine that has the following acceptance mechanism: an input x is accepted if and only if the number of accepting computation paths on input x is a member of S. Section 7 discusses natural complete languages, containment relationships, and relativized separations among counting classes. Our relativizations, which are obtained by combinatorially exploiting the limited control of counting machines, reveal the surprising weakness of counting classes; in appropriately relativized worlds, counting classes fail to even contain NP.

Theorem: There is a recursive set A so that $\bigcup_{S \text{ finite}} CP_S^A \not\supseteq NP^A$ and $\bigcup_{S \text{ cofinite}} CP_S^A \not\supseteq coNP^A$.

1.8. A Log-bounded Hierarchy

Section 8 defines a generalization of the boolean hierarchy: the log-bounded hierarchy (Σ^L). The section explores problems that may lie just beyond the power of hardware over NP.

1.9. The Boolean Hierarchy and Sparse Oracles for NP

Finally, Section 9 studies the relations between sparse oracles and the boolean hierarchy. [M82] shows that if NP has a sparse many-one complete set then P=NP. We say NP has a *sparse oracle* if there is a sparse set S so $NP \subseteq P^S$. [KL80][H82] show that if NP has a sparse Turing complete set or a sparse oracle, then $\Sigma_2^p = \Pi_2^p$. One wonders if the existence of a sparse oracle for NP implies something stronger than $\Sigma_2^p = \Pi_2^p$.

By displaying a relativized world where NP has a sparse oracle but in which the boolean hierarchy is infinite, we show that [KL80]'s conclusion can not be much strengthened among proofs that relativize. In terms of circuits, this result shows that (in relativized worlds) the existence of small circuits for NP does not imply that $P = NP$ or even that the boolean hierarchy is finite. This result improves the work of Kurtz, Immerman and Mahaney reported in [HSI83b]. Similarly, by displaying a relativized world where PSPACE has a sparse oracle but where the boolean hierarchy is infinite, we show that the Karp-Lipton result on sparse oracles for PSPACE (i.e., if PSPACE has a sparse oracle then PSPACE $= \Sigma_2^p$) is also as strong as possible. Indeed, we show the stronger result that $P^S = PSPACE^S$, S sparse, can be consistent with an infinite boolean hierarchy.

Theorem: There is a relativized world where the boolean hierarchy is infinite and PSPACE has sparse oracles (and, indeed, in which there is a sparse S so $P^S = PSPACE^S$).

2. Definitions and Representations

2.1. The Ubiquitous Boolean Hierarchy: Definitions and Algebraic Characterizations

In this section, we make precise the notation of fixing a certain amount of hardware logic over NP.

Definition 2.1.1:

Operator **C**:

$\mathbf{C}(L_1) = L_1$.

$$\mathbf{C}(L_1, \ldots, L_k) = \begin{cases} \mathbf{C}(L_1, \ldots, L_{k-1}) \bigcup L_k & \text{if } k \text{ is odd} \\ \mathbf{C}(L_1, \ldots, L_{k-1}) \bigcap \overline{L_k} & \text{if } k \text{ is even} \end{cases} \qquad \textit{for } k > 1$$

Operator **D**:

$$\mathbf{D}(L_1) = L_1.$$

$$\mathbf{D}(L_1, \ldots, L_k) = L_1 - \mathbf{D}(L_2, \ldots, L_k) \qquad \textit{for } k>1$$

Operator E:

$$\mathrm{E}(L_1) = L_1.$$

$$\mathrm{E}(L_1, L_2) = L_1 - L_2.$$

$$\mathrm{E}(L_1, \ldots, L_k) = \begin{cases} \mathrm{E}(L_1, \ldots, L_{k-1}) \bigcup L_k & \text{if } k \text{ is odd} \\ \mathrm{E}(L_1, \ldots, L_{k-2}) \bigcup (L_{k-1} - L_k) & \text{if } k \text{ is even} \end{cases} \quad \textit{for } k>2$$

Thus, $\mathbf{C}(L_1, \ldots, L_n)$ represents an alternating sum of the languages

$$L_1 - L_2 + L_3 \ldots + (-1)^{n-1} L_n \tag{C*}$$

(where the expression is associated to the left, and $+$, $-$ stand for set theoretic union and difference, respectively). Similarly, $\mathbf{D}(L_1, \ldots, L_n)$ represents the nested difference of the languages

$$L_1 - (L_2 - (\ldots (L_{n-1} - L_n) \ldots)) \tag{D*}$$

And E represents sums of differences of consecutive pairs.

$$\mathrm{E}(L_1, \ldots, L_n) = \begin{cases} (L_1 - L_2) \bigcup \ldots \bigcup (L_{n-2} - L_{n-1}) \bigcup L_n & \text{if } n \text{ is odd} \\ (L_1 - L_2) \bigcup \ldots \bigcup (L_{n-1} - L_n) & \text{if } n \text{ is even} \end{cases} \tag{E*}$$

We consider the language classes generated by applying **C** to a fixed number of NP languages.

Definition 2.1.2: For all $n \geq 1$, define NP(n) = $\{\mathbf{C}(L_1, \ldots, L_n) \mid L_i \in \mathrm{NP}, 1 \leq i \leq n\}$.

In particular, NP(1) = NP and NP(2) = $\mathrm{D^P}$. For any oracle set X, the relativized version $\mathrm{NP}^X(n)$ is naturally defined.

We now show how C, D, and E provide several normal forms for NP(n). Propositions 2.1.1−2.1.7 show equivalences that will be used in our proof of the following theorem.

Theorem 2.1.1: Any of the following classes is precisely NP(n), where the L_i's are in NP.

1) $\{\mathbf{C}(L_1, \dots, L_n)\}$.

1′) $\{\mathbf{C}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n\}$.

2) $\{\mathbf{D}(L_1, \dots, L_n)\}$.

2′) $\{\mathbf{D}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n\}$.

3) $\{\mathbf{E}(L_1, \dots, L_n)\}$.

3′) $\{\mathbf{E}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n\}$.

Proposition 2.1.1: If $L_1 \supseteq L_2 \supseteq \dots \supseteq L_n$, then $\mathbf{C}(L_1, \dots, L_n) = \mathbf{E}(L_1, \dots, L_n)$

If the L_i's are in NP, each of these is a disjoint union of $\lfloor n/2 \rfloor$ $\mathrm{D^P}$ sets, plus an NP set if n is odd.

Proof of Proposition 2.1.1: We use the fact that $A \bigcap C = \emptyset$ implies $A \bigcup (B-C) = (A \bigcup B)-C$. The proof is by induction. Clearly, equality holds for $n = 1$ and $n = 2$. Let $k > 2$ and inductively assume equality for all values less than k. If k is odd, the result follows directly by Definition 2.1.1 and the inductive hypothesis. If k is even, using Definition 2.1.1, the inductive hypothesis, and $L_1 \supseteq L_2 \supseteq \dots \supseteq L_n$, $\mathbf{C}(L_1, \dots, L_k) = (\mathbf{C}(L_1, \dots, L_{k-2}) \bigcup L_{k-1})-L_k = (\mathbf{E}(L_1, \dots, L_{k-2}) \bigcup L_{k-1})-L_k = \mathbf{E}(L_1, \dots, L_{k-2}) \bigcup (L_{k-1}-L_k) = \mathbf{E}(L_1, \dots, L_k)$. ▸

Corollary 2.1.1: If $L_1 \supseteq L_2 \supseteq \dots \supseteq L_n$ and $n \geq 3$, then $\mathbf{C}(L_1, \dots, L_n) = (L_1-L_2) \bigcup \mathbf{C}(L_3, \dots, L_n)$.

Proposition 2.1.2: $\mathbf{D}(L_1, \dots, L_n) = \mathbf{D}(L'_1, \dots, L'_n)$, where $L'_i = \bigcap_{1 \leq j \leq i} L_j$, $1 \leq i \leq n$.

Proof of Proposition 2.1.2: Again an easy induction using the facts that

$A-B = A-(A \cap B)$ and $A \cap (B-C) = (A \cap B)-C$.▸

Proposition 2.1.3 (The arithmetic lemma): If $L_1 \supseteq L_2 ... \supseteq L_n$, then $\mathbf{D}(L_1, ..., L_n) = \mathbf{C}(L_1, ..., L_n)$.

Proof of Proposition 2.1.3: Another induction, using Corollary 2.1.1 and the facts that $C \subseteq A \Rightarrow A-(B-C) = (A-B) \cup C$ and $\mathbf{D}(L_3, ..., L_k) \subseteq L_3 \subseteq L_1$. Clearly, the proposition is true for $n=1$ and $n=2$. Suppose $k>2$. $\mathbf{D}(L_1, ..., L_k) = L_1-(L_2-\mathbf{D}(L_3, ..., L_n))) = (L_1-L_2) \cup \mathbf{D}(L_3, ..., L_k) = (L_1-L_2) \cup \mathbf{C}(L_3, ..., L_k) = \mathbf{C}(L_1, ..., L_k)$. ▸

. (This is called the arithmetic lemma since it resembles the "simple arithmetic" of (C*) and (D*).)

It is also possible to express **C** in terms of **D**. Given any $L_1, ..., L_k$, by the definition of **C**:

$$x \in \mathbf{C}(L_1, ..., L_k) \Leftrightarrow \begin{cases} x \in L_k \text{ or } x \in \mathbf{C}(L_1, ..., L_{k-1}) & \text{if } k \text{ is odd} \\ x \notin L_k \text{ and } x \in \mathbf{C}(L_1, ..., L_{k-1}) & \text{if } k \text{ is even} \end{cases}$$

Proposition 2.1.4: $x \in \mathbf{C}(L_1, ..., L_k) \Leftrightarrow$ the largest i for which $x \in L_i$ is odd. (The i is zero if $\{i \mid x \in L_i\} = \varnothing$).

Proof of Proposition 2.1.4: We prove by induction. The $k=1$ case is trivial. Let $k>1$ and inductively assume the proposition true for all values less than k. If k is odd, then $x \in \mathbf{C}(L_1, ..., L_k) \Leftrightarrow [x \in L_k$ or $\max\{i \mid x \in L_i, 1 \le i < k\}$ is odd$] \Leftrightarrow [\max\{i \mid x \in L_i, 1 \le i \le k\}$ is odd$]$.

A similar argument applies for k even. ▸

Corollary 2.1.4: If $S_1 \supseteq S_2 ... \supseteq S_n$, then $[\, x \in \mathbf{D}(S_1, ..., S_n) \Leftrightarrow$ the last i for which $x \in S_i$ is odd$]$.

Proof of Corollary 2.1.4: Use the arithmetic lemma.▸

Given x, let $\{i \mid x \in L_i,\ 1 \le i \le k\}$ be ordered as $i_1 < i_2 < \ldots < i_l$. Consider all subsequences of $(i_1, i_2, \ldots, i_l)$ that have alternating parity and start with an odd number (e.g., (3,6,11)). Denote by $l(x)$ the length of the longest such subsequence. (If no such subsequence exists, then $l(x) = 0$.) We have the following proposition.

Proposition 2.1.5: $x \in \mathbf{C}(L_1, \ldots, L_k) \Leftrightarrow l(x)$ is odd.

Proof of Proposition 2.1.5: Consider a longest alternating parity subsequence of x that starts with an odd number. (Use the empty string if $l(x) = 0$.) Clearly $l(x)$ is odd if and only if this subsequence ends with an odd number i. Since it is the longest, there is no $j > i$ such that $x \in L_j$ and j is of different parity than i. Hence $l(x)$ is odd if and only if the last i for which $x \in L_i$ is odd. Proposition 2.1.5 now follows from Proposition 2.1.4. ▸

Now, we prove the following proposition.

Proposition 2.1.6: $\mathbf{C}(L_1, \ldots, L_n) = \mathbf{D}(S_1, \ldots, S_n)$ where $S_i = \{x \mid l(x) \ge i\} =$

$$\bigcup_{\substack{1 \le t_1 < \ldots < t_i \le n \\ t_1, t_3, \ldots \text{ are odd} \\ t_2, t_4, \ldots \text{ are even}}} (L_{t_1} \cap L_{t_2} \cap \ldots \cap L_{t_i}), \text{ for } 1 \le i \le n.$$

Hence, any set in $\mathrm{NP}(n)$ can be expressed as $\mathbf{D}(S_1, \ldots, S_n)$, where $S_i \in \mathrm{NP}$ and $S_1 \supseteq S_2 \supseteq \ldots \supseteq S_n$.

Proof of Proposition 2.1.6: For each x, let i_0 be the last S_i to which x belongs. Then $l(x) = i_0$. Now $x \in \mathbf{C}(L_1, \ldots, L_n) \Leftrightarrow l(x) = i_0$ is odd $\Leftrightarrow x \in \mathbf{D}(S_1, \ldots, S_n)$. ▸

Finally, we will express E in terms of C. Given sets $A_i, B_i,\ 1 \le i \le k$. Denote $\#A(x) = |\{i \mid x \in A_i,\ 1 \le i \le k\}|$, the number of A_i's to which x belongs. Define $\#B(x)$ similarly.

Proposition 2.1.7: $(\forall k \ge 1)[\ (\forall 1 \le i \le k)[A_i \supseteq B_i] \Rightarrow \mathbf{E}(A_1, B_1, \ldots, A_k, B_k) = \mathbf{C}(S_1, T_1, \ldots, S_k, T_k)]$ where $S_i = \{x \mid \#A(x) \ge i\} = \bigcup_{1 \le j_1 < \ldots < j_i \le k} (A_{j_1} \cap \ldots \cap A_{j_i})$ and $T_i = \{x \mid$

$\#B(x) \geq i\} = \bigcup_{1 \leq j_1 < \dots < j_i \leq k} (B_{j_1} \cap \dots \cap B_{j_i})$.

Proof of Proposition 2.1.7: Since each $A_i \supseteq B_i$, $\#A(x) \geq \#B(x)$, for every x. Furthermore, $x \in \mathrm{E}(A_1, B_1, \dots, A_k, B_k) \Leftrightarrow \#A(x) > \#B(x)$. Let $i_0 = \#A(x)$. If $i_0 > \#B(x)$ then $x \in S_{i_0}$ but $x \notin T_j$, for $i_0 \leq j \leq k$. $\mathrm{C}(S_1, T_1, \dots, S_k, T_k) = ((\dots((X \bigcup S_{i_0}) - T_{i_0}) \dots) \bigcup S_k) - T_k \ni x$, X is some set. Suppose $i_0 = \#B(x)$. If $i_0 = 0$ then clearly $x \notin \mathrm{C}(S_1, T_1, \dots, S_k, T_k)$. The case $i_0 = k$ is similar. If $0 < i_0 < k$, then $x \in T_{i_0}$, but $x \notin S_j$, for $i_0 < j \leq k$. Again $x \notin \mathrm{C}(S_1, T_1, \dots, S_k, T_k)$. ▸

Corollary 2.1.7: $\mathrm{E}(L_1, \dots, L_n)$ is a set in NP(n), where $L_i \in$ NP.

Proof of Corollary 2.1.7: The proof is by induction. For odd n, the statement follows trivially, using the inductive hypothesis and Definition 2.1.1. For even n, just note that $\mathrm{E}(A_1, B_1, \dots, A_k, B_k) = \mathrm{E}(A_1, A_1 \cap B_1, \dots, A_k, A_k \cap B_k)$.▸

We are now prepared for the proof of Theorem 2.1.1.

Proof Theorem 2.1.1: Clearly $1' \subseteq 1$, $2' \subseteq 2$, and $3' \subseteq 3$. Also, $1' \equiv 3'$, $2 \subseteq 2'$, $2' \equiv 1'$, $1 \subseteq 2'$, $3 \subseteq 1$, by, respectively Propositions 2.1.1,2,3,6 and Corollary 2.1.7.

Definition 2.1.3: coNP(n) $= \{\bar{L} \mid L \in \mathrm{NP}(n)\}$.

If one starts from coNP, then the following theorem can be proved similarly.

Theorem 2.1.1': The following classes are identical, where the L_i's are in coNP.

1) $\{\mathrm{C}(L_1, \dots, L_n)\}$.

1') $\{\mathrm{C}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n\}$.

2) $\{\mathrm{D}(L_1, \dots, L_n)\}$.

2') $\{\mathrm{D}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n\}$.

3) $\{\mathrm{E}(L_1, \dots, L_n)\}$.

3') $\{\mathrm{E}(L_1, \dots, L_n) \mid L_1 \supseteq \dots \supseteq L_n\}$.

Using Theorem 2.1.1' one can prove the following theorem, which will be useful later. (The proof is left to the reader.)

Theorem 2.1.2: $\{C(\overline{L_1}, \ldots, \overline{L_n}) \mid \overline{L_i} \in \mathrm{coNP}\} = \begin{cases} \mathrm{NP}(n) & \text{if } n \text{ is even} \\ \mathrm{coNP}(n) & \text{if } n \text{ is odd} \end{cases}$

Theorem 2.1.3 (Weak Hierarchy Theorem): For every $m<n$, $\mathrm{NP}(m) \bigcup \mathrm{coNP}(m) \subseteq \mathrm{NP}(n) \bigcap \mathrm{coNP}(n)$.

Proof of Theorem 2.1.3: Using normal form 3 from Theorem 2.1.1, we get $\mathrm{NP}(n) \subseteq \mathrm{NP}(n+1)$, since $E(L_1, \ldots, L_n) = E(L_1, \ldots, L_n, \varnothing)$. Using normal form 1, we have $C(L_1, \ldots, L_n) = \overline{C(\Sigma^*, L_1, L_2, \ldots, L_n)}$, hence $\mathrm{NP}(n) \subseteq \mathrm{coNP}(n+1)$. Therefore, $\mathrm{NP}(n) \subseteq \mathrm{NP}(n+1) \bigcap \mathrm{coNP}(n+1)$. Since the right-hand side is closed under complementation, the theorem follows. ▸

Definition 2.1.4: $\mathrm{BH} = \bigcup_{k \geq 1} \mathrm{NP}(k)$. This defines the *boolean hierarchy*.

Theorem 2.1.4: $\mathrm{BH} = \bigcup_{k \geq 1} \{ C(\overline{L_1}, \ldots, \overline{L_n}) \mid \overline{L_i} \in \mathrm{coNP} \}$.

Whether the boolean hierarchy collapses (at some level or entirely or not at all) is unknown. We note that $\mathrm{P} = \mathrm{NP} \Rightarrow \mathrm{P} = \mathrm{BH}$, hence a proof that the boolean hierarchy doesn't collapse entirely would entail $\mathrm{P} \neq \mathrm{NP}$. On the other hand, even assuming $\mathrm{P} \neq \mathrm{NP}$, a proof that the boolean hierarchy is infinite would still subsume such major open problems as $\mathrm{D^P} \neq^? \mathrm{coD^P}$. Therefore settling whether the boolean hierarchy is a true hierarchy will be extremely difficult.

Note that $\mathrm{NP}(k) = \mathrm{coNP}(k)$ implies that $\mathrm{NP}(k) = \mathrm{BH}$, using the normal forms. As a final remark, the boolean hierarchy sits between the first and second levels of the polynomial hierarchy: $\mathrm{BH} \subseteq \mathrm{P^{SAT}} = \Delta_2^{\mathrm{P}}$.

2.2. The Boolean Hierarchy is the Boolean Closure of NP

The boolean hierarchy *is* the boolean closure of NP. To prove this claim, we simply note that any amount of hardware over NP can be reshaped into a finite union of $\mathrm{D^P}$ sets. Thus two levels of hardware on top of $\mathrm{NP} \bigcup \mathrm{coNP}$ are as powerful as n levels. Though every hardware tree has an equivalent two level tree, we may need many more gates on the shallow tree.

Lemma 2.2.1: If S is expressible by hardware over NP predicates (i.e., if S is in the closure of NP with respect to $\{\bigcup, \bigcap$, complementation$\}$), then S is a finite union of $\mathrm{D^P}$ sets and a finite intersection of $\mathrm{coD^P}$ sets.

Proof of Lemma 2.2.1: To show S is the finite union of $\mathrm{D^P}$ sets, rewrite the tree describing S in disjunctive normal form over the predicates and their negations. We get the two-level logic tree of Figure 4, whose leaves are NP and coNP predicates. Since any intersection of sets from $\mathrm{NP} \bigcup \mathrm{coNP}$ is in $\mathrm{D^P}$, we can write S as the finite union of $\mathrm{D^P}$ sets. Similarly, using conjunctive normal form, S is the finite intersection of $\mathrm{coD^P}$ sets. ▶

Theorem 2.2.1: The boolean hierarchy is the closure of NP under boolean operations.

Proof of Theorem 2.2.1: Clearly every set in the boolean hierarchy is the in the boolean closure of NP. By our lemma, every set S in the boolean closure of NP is the finite union of $\mathrm{D^P}$ sets. Thus for some k and $\{L_i \in \mathrm{NP}\}$, S is $\mathrm{E}(L_1, \ldots, L_{2k})$. Thus S is in $\mathrm{NP}(2k)$ by Theorem 2.1.1, so S is in the boolean hierarchy.▶

We can give another characterization of our boolean hierarchy. The bounded truth-table reducibility of recursion theory has the following analogue among polynomial time computable classes ([Y83]):

$A \leq^{\mathrm{P}}_{btt} B$ iff there are polynomial time transducers M_1 and M_2 and an integer n so that for every x, $M_1(x)$ prints a finite set of at most n elements and $M_2(x)$ prints a finite collection of finite sets, such that $x \in A$ iff $B \cap M_1(x)$ is present in $M_2(x)$.

We have the following theorem.

Theorem 2.2.2: The boolean hierarchy is the closure of NP under polynomial time bounded truth-table reductions: BH $= \{T \mid T \leq^{\mathrm{P}}_{btt} \mathrm{SAT}\}$.

Proof of Theorem 2.2.2: $\Rightarrow$(BH $\subseteq \{T \mid T \leq^{\mathrm{P}}_{btt} \mathrm{SAT}\}$): Let $A \in$ BH, then there are sets $L_1 \supseteq L_2 \supseteq \cdots \supseteq L_k$ so that $A = \mathrm{C}(L_1, L_2, \cdots L_k)$, where $L_i \in$ NP. Let f_i be a many-one p-time reduction from L_i to SAT, a complete language for NP. Now for all x, $M_1(x)$ prints $\{f_1(x),...,f_k(x)\}$ and $M_2(x)$ prints the following sets $\{f_1(x),...,f_{2i-1}(x)\}$, where $2i-1 \leq k$.

$\Leftarrow$(BH $\supseteq \{T \mid T \leq^{\mathrm{P}}_{btt} \mathrm{SAT}\}$): Let $A \leq^{\mathrm{P}}_{btt}$SAT via M_1 and M_2. W.l.o.g., $M_1(x)$ always prints k element sets. Let $M_1(x) = \{f_1(x),...,f_k(x)\}$, and $L_i = \{ x \mid f_i(x) \in \mathrm{SAT} \}$. Then $L_i \in$ NP. Now it is easy to construct a union of 2^k many D^p sets that equals A, using the definition of $A \leq^{\mathrm{P}}_{btt}$SAT.▸

Another interesting point is that $A \leq^{\mathrm{P}}_{btt} B$ is equivalent to A being accepted by a p-time oracle Turing machine using oracle B that, for some fixed j, queries at most j strings on any input. The proof is left to the reader.

3. Separations

How far does the boolean hierarchy extend? This section shows that all reasonable answers to this question can be obtained in appropriately relativized worlds. Section 3.1 gives a detailed relativization that makes the boolean hierarchy infinite. Section 3.2 shows that, for each k, the boolean hierarchy can extend exactly k levels, with everything up to PSPACE collapsing into the $k+1$'st level.

3.1. An Infinite Relativized Boolean Hierarchy

This section proves the following theorem.

Theorem 3.1.1: There exists a recursive oracle X separating the entire boolean hierarchy. Specifically,

$$P^X \subset NP^X \bigcap coNP^X \begin{array}{c} \subset NP^X \subset \\ \neq \\ \subset coNP^X \subset \end{array} NP^X \bigcup coNP^X \subset NP^X(2) \bigcap coNP^X(2) \cdots$$

We first describe the strategies for X to achieve each of these separations.

- $P^X \neq NP^X \cap coNP^X$ is done in the usual way: Allocate two "segments" of strings with exponentially many for each n. Diagonalize each p-time deterministic machine over a tally set $L_1^{\cap}$, where "$1^n \overset{?}{\in} L_1^{\cap}$" is determined by the existence of some strings in the appropriate "segment." To insure that $L_1^{\cap} \in NP^X \cap coNP^X$, we impose certain global constraints on X: X will have strings from exactly one segment of the two allocated, corresponding to any particular n. The oracle set X is constructed in stages $\{X_s\}$; an extension of X_s is termed "admissible" only if it satisfies all these global constraints. (There are other constraints to be discussed later.)

To diagonalize, we pick an n sufficiently large, fix an extension $\hat{X}_s$ of X_s that is "admissible" except for length n, where $\hat{X}_s$ has no strings from either segments. Run the next p-time machine P_i on 1^n, using $\hat{X}_s$ to answer queries, and "freeze" to $\overline{X}$ any string that is queried negatively. Since at most polynomially many strings are "frozen" when P_i halts, we may, according to whether P_i accepts or rejects 1^n, choose the appropriate segment to enumerate a string (corresponding to length n) into X.

Then extend X_s according to $\hat{X}_s$ so that, the computation of P_i on 1^n is preserved.

Thus, the final X will be "admissible" and $P^X \neq NP^X \cap coNP^X$.

The other requirements are of the following two types:

$$\begin{cases} NP^X(k) \neq coNP^X(k) \\ NP^X(k) \cup coNP^X(k) \neq NP^X(k+1) \cap coNP^X(k+1) \end{cases}$$

for all $k \geq 1$.

Note that this is enough since

$$NP^X(k) \neq coNP^X(k) \Rightarrow NP^X(k) \cap coNP^X(k) \begin{matrix} \subset NP^X(k) \subset \\ \subset coNP^X(k) \subset \end{matrix} NP^X(k) \bigcup coNP^X(k)$$

Furthermore, for requirements $NP^X(k) \neq coNP^X(k)$ it is enough to satisfy for all odd k.

• Strategies to insure $NP^X(k) \neq coNP^X(k)$ for odd k.

We will illustrate with the case k = 3. Let's fix 3 separate "segments" S_i, $1 \leq i \leq 3$, each of which having an n^{th} "block" S_i^n with exponentially many strings for every n. Define

$$L_3 = \{1^n | (\exists s_1 \wedge \neg \exists s_2) \vee \exists s_3\},$$

where "$\exists\ s_i$" stands for: $\exists\ s_i \in X \cap S_i^n$.

Clearly $L_3 \in NP^X(3)$. We want to make sure that for each triple $\langle N_i, N_j, N_k \rangle$, the $coNP^X(3)$ language $W_3 = (\overline{L(N_i^X)} \cup L(N_j^X)) \cap \overline{L(N_k^X)}$ is *not* L_3 ; hence $L_3 \notin coNP^X(3)$.

The following generic step embodies a "sawing" argument that is the key to our separation results, and can be readily generalized to the $k > 3$ case.

Pick an n sufficiently large so that the S_i^n 's are all "clean," in the sense that no string from these blocks has been committed to either X or $\overline{X}$.

Case 1: If N_k accepts 1^n under some "admissible" extension of the current X_s, then we "freeze" an accepting computation, extend X_s "admissibly," ("freeze" those used negatively in the computation to $\overline{X}$.) Then enumerate an "unfrozen" $s_3 \in S_3^n$ into X, (if there is none in X so far). We will see that there will always be some s_3 "unfrozen," if n is large, and $s_3 \in X$ does not violate any global constraints. Thus $1^n \notin W_3$ but $1^n \in L_3$.

Case 2: N_k never accepts 1^n under any "admissible" extension. We'll say an extension is *without* T if $X \cap T^n = \varnothing$. Consider all "admissible" extensions without S_3 (i.e., extensions in which $X \cap S_3^n = \varnothing$).

Case 2-1: If N_j accepts 1^n under some such extension, then "freeze" an accepting computation and extend X_s "admissibly" without S_3. Enumerate an "unfrozen" $s_2 \in S_2^n$ into X (if none exists in X so far). Thus $1^n \in W_3$ but $1^n \notin L_3$.

Case 2-2: N_j never accepts 1^n under any admissible "without S_3" extension. We consider all "admissible" extensions that are "without S_3 and without S_2."

Case 2-2-1: If N_i accepts 1^n under one of such extension, "freeze" it. Extend X_s "admissibly" without S_3, S_2; put an "unfrozen" $s_1 \in S_1^n$ in X. Thus $1^n \notin W_3$ but $1^n \in L_3$.

Case 2-2-2: N_i never accepts 1^n under any "admissible without S_3, S_2" extension. Then extend X_s "admissibly" without S_3, S_2, S_1. Thus $1^n \in W_3$, but $1^n \notin L_3$.

Hence, the final X will be admissible and $\mathrm{NP}^X(3) \neq \mathrm{coNP}^X(3)$.

• Strategies for $\mathrm{NP}^X(k) \cup \mathrm{coNP}^X(k) \neq \mathrm{NP}^X(k+1) \cap \mathrm{coNP}^X(k+1)$, for all $k \geq 1$.

We illustrate with k = 2. Fix 6 separate "segments" S_i, S_i', $i = 1,2,3$ as before. Denote S_i^n, S'^n_i their n^{th} blocks. We impose the following global constraints on X:

For each n, either

$$[(\exists s_1 \wedge \neg \exists s_2) \vee \exists s_3] \wedge [(\neg \exists s_1' \vee \exists s_2') \wedge \neg \exists s_3'] \tag{I}$$

or

$$[(\neg \exists s_1 \vee \exists s_2) \wedge \neg \exists s_3] \wedge [(\exists s_1' \wedge \neg \exists s_2') \vee \exists s_3'] \tag{II}$$

where "$\exists s_i$" "$\exists s_i'$" are shorthands as before. E.g., "$\exists s_i$" means "$\exists\, s_i \in S_i^n \cap X$."

Define

$$L_3^{\cap} = \{1^n | (\exists s_1 \wedge \neg \exists s_2) \vee \exists s_3\},$$

Clearly, if X satisfies the above constraints, then $L_3^{\cap} \in \mathrm{NP}^X(3) \cap \mathrm{coNP}^X(3)$.

We need to insure for each tuple $\langle N_i, N_j \rangle$, that neither $L(N_i^X) \cap \overline{L(N_j^X)}$ nor $\overline{L(N_i^X)} \cup L(N_j^X)$ equals $L_3^{\cap}$, hence $L_3^{\cap} \notin \mathrm{NP}^X(2) \cup \mathrm{coNP}^X(2)$.

To insure this, we use the following variant of the "sawing" argument discussed before.

Step (1). For $\overline{L(N_i^X)} \cup L(N_j^X)$:

Pick an n large enough so that the corresponding "blocks" S_i^n, S'^n_i are all "clean."

Temporarily ignore global constraints (I) (II) for this particular n; consider all "admissible" extensions of X_s without S_3, S'_3 (that is, for which $X \cap S_3^n = \varnothing$ and $X \cap S'^n_3 = \varnothing$).

Case 1: If N_j accepts 1^n under such an extension, then we "freeze" an accepting computation. Extend X_s "admissibly," and for this n, add an "unfrozen" member of S_2^n and an "unfrozen" member of S'^n_3 to X, thus satisfying (II); therefore $1^n \in L(N_j^X)$ but $1^n \notin L_3^{\cap}$.

Case 2: N_j never accepts 1^n under any such extension. Consider all "admissible" (again ignore (I) (II) for this n) extensions of X_s without $S_3, S'_3, S_2,$ and S'_2.

Case 2-1: If N_i accepts 1^n for such an extension, then "freeze" one accepting computation, extend X_s "admissibly." For this n, add to X an s_1 and an s'_2 "unfrozen" from the appropriate blocks, respectively S_1^n and S'^n_2. Thus satisfy (I), and $1^n \notin \overline{L(N_i^X)} \cup L(N_j^X)$ but $1^n \in L_3^{\cap}$.

Case 2-2: N_i never accepts 1^n under such extensions. Then it rejects when we extend X_s "admissibly" without S_1 and with some $s'_1 \in S'^n_1$, and "without S_3, S'_3, S_2, S'_2." Then (II) is satisfied. Now $1^n \in \overline{L(N_i^X)}$ but $1^n \notin L_3^{\cap}$.

This completes the diagonalization of Step (1).

Step (2) is similar, except we do it for $L(N_i^X) \cap \overline{L(N_j^X)}$.

A formal proof follows.

Proof of Theorem 3.1.1:

We use a standard pairing function $\langle \cdot,\cdot \rangle$, for which $|\langle x,y \rangle| \sim |x| + |y|$. Whenever convenient, we identify binary string x with integer $1x$. Define $\langle a,b,c \rangle = \langle a,\langle b,c \rangle\rangle$, etc.

Let $\{P_i\}$ ($\{N_i\}$) be standard enumerations of deterministic (non-deterministic) TM's with polynomial time clocks p_i.

Define $S^n_{i,j,k} = \{ \langle i,j,k,x \rangle \mid |x| = n \}$. Note that $S^n_{i,j,k} \cap S^{n'}_{i',j',k'} = \emptyset$, unless they are identical. Define propositions $p^n_{i,j} \equiv S^n_{i,j,1} \cap X \neq \emptyset$, $q^n_{i,j} \equiv S^n_{i,j,2} \cap X \neq \emptyset$. We also use the operator **C** on propositions in a natural way.

Fact: $\{1^n \mid \mathbf{C}(q_1, \ldots, q_t)\} \in \mathrm{NP}(t)$, provided $\{1^n | q_s\} \in \mathrm{NP}$, $1 \le s \le t$.

Define $L_k = \{ 1^n | \mathbf{C}(q^n_{k,1},\ldots,q^n_{k,k})\}$ for odd k. Then $L_k \in \mathrm{NP}^X(k)$.

We will impose the following global constraints $GC_{k,n}$ on X :

$$\mathbf{C}(p^n_{k,1},\ldots,p^n_{k,k}) \Leftrightarrow \neg \mathbf{C}(p^n_{k,k+1},\ldots,p^n_{k,2k}). \qquad (*)$$

Define $L^{\cap}_k = \{ 1^n | \mathbf{C}(p^n_{k,1},\ldots,p^n_{k,k}) \}$

Fact: If X satisfies $GC_{k,n}$, then $L^{\cap}_k \in \mathrm{NP}^X(k) \cap \mathrm{coNP}^X(k)$.

We have the following requirements (relativized to X):

$R_{\langle 0,i \rangle}$: $L(P_i) \neq L^{\cap}_1$,

for $k \ge 1$,

$R_{\langle 2k-1,i \rangle}$: $\overline{\mathbf{C}(L(N_{i_1}),\ldots,L(N_{i_{2k-1}}))} \neq L_{2k-1}$, where $i = \langle i_1,\ldots,i_{2k-1} \rangle$,

$R_{\langle 2k,i \rangle}$: $\mathbf{C}(L(N_{i_1}),\ldots,L(N_{i_k})) \neq L^{\cap}_{k+1}$ and $\overline{\mathbf{C}(L(N_{i_1}),\ldots,L(N_{i_k}))} \neq L^{\cap}_{k+1}$, where $i = \langle i_1, \ldots ,i_k \rangle$.

Each $S^n_{i,j,k}$ is called a "block." Of importance to us are the blocks concerned in $p^n_{i,j}$ and $q^n_{i,j}$. Specifically, $S^n_{i,j,1}$ for $1 \le j \le 2i$ and $S^n_{i,j,2}$ for odd i, $1 \le j \le i$. We call $\bigcup_{j=1}^{2k} S^n_{k,j,1}$ and

$\bigcup_{j=1}^{k} S_{k,j,2}^{n}$ "large blocks." Moreover, the former is called a "global constraint block," $GCB_{k,n}$.

We further suppose a well-ordering $<<$ (of type ω) is defined, which "holds together" each block and large block. (I.e., it induces a well-ordering on the block quotient space. This can be easily accomplished, e.g., imagine all integers on a line, then bring everything in $S_{i,j,k}^{n}$ to just to the right of its leftmost member. That is, we'll say that $x_1 << x_2$ if (1) the smallest integer in the block x_1 belongs to is less than the smallest integer in the block x_2 belongs to *or* (2) x_1 and x_2 are in the same block and $x_1 < x_2$. Also, do this for the large blocks.

X is constructed in stages; $X = \bigcup X_s$. At the end of stage s, we have an initial (under $<<$) segment X_s of X. Membership in X or $\overline{X}$ of any string in a block or a large block is determined in a single stage. If we define block(X_s) = the set of blocks on which X has been defined by stage s, then block(X_s) is an initial segment in the block quotient space and

$$X_s \cap \bigcup \text{block}(X_s) = X \cap \bigcup \text{block}(X_s). \qquad (*)$$

An *admissible extension* of X_s is a set with X_s as initial segment (condition (*)) and satisfies all $GC_{k,n}$. We will maintain the condition that for each $GCB_{k,n}$, if $GCB_{k,n} \cap \bigcup \text{block}(X_s) \neq \emptyset$, then $GC_{k,n}$ is satisfied by X_s; thus such admissible extensions always exist.

We say a block B is *clean* if $B \cap \bigcup \text{block}(X_s) = \emptyset$.

Construction of X :

$X_0 = \emptyset$; $s = 0$; $n_0 = 0$.

Stage $s+1 = \langle 0, i \rangle + 1$: We satisfy $R_{\langle 0,i \rangle}$.

Let $n_{s+1} = \min\{n \mid n > n_s \wedge GCB_{1,n}$ is clean $\wedge\ 2^n > p_i(n)\}$. (Recall that $p_i(\cdot)$ is the run time of deterministic polynomial time machine P_i.)

Take an extension $\hat{X_s}$ of X_s satisfying $\hat{X_s} \cap GCB_{1,\, n_{s+1}} = \varnothing$ and otherwise admissible.

Run P_i on 1^{n_s+1} using $\hat{X_s}$. "Freeze" into $\overline{X}$ all strings not in $\hat{X_s}$ that are queried by P_i. According to whether $P_i(1^{n_s+1})$ accepts or rejects, enumerate an unfrozen x from $S_{1,2,1}^{n_s+1}$ or $S_{1,1,1}^{n_s+1}$ respectively into $\hat{X_s}$. (This satisfies $GC_{1,n_{s+1}}$.) Let B be the <<-last (large) block that was queried, or $GCB_{1,\, n_{s+1}}$, whichever is later.

Set X_{s+1} to $\hat{X_s}$ up to B. $s := s+1$.

Stage $s+1 = \langle 2k-1, i\rangle + 1$: We satisfy $R_{\langle 2k-1,i\rangle}$

(Recall that $i = \langle i_1, \ldots, i_{2k-1}\rangle$.)

Let $n_{s+1} = \min\{n \mid n > n_s \wedge \bigcup_{j=1}^{2k-1} S_{2k-1,j,2}^{n}$ is clean $\wedge\ 2^n > \sum_{j=1}^{2k-1} p_{i_j}(n)\}$.

saw $:= 2k-1$; *done* := false;

Repeat

Consider all admissible extensions of X_s, satisfying $\neg q_{2k-1,j}^{n_s+1}$, for $saw < j \leq 2k-1$. (If $saw = 2k-1$, the previous condition is void.)

If $N_{i_{saw}}(1^{n_s+1})$ has an accepting computation under such an extension $\hat{X_s}$, we will "freeze" an accepting computation as follows:

Extend X_s according to $\hat{X_s}$ up to the <<-last block for which there exists a queried string (round up to a large block if the block is a part of it) or $\bigcup_{j=1}^{2k-1} S_{2k-1,j,2}^{n_s+1}$, whichever is later.

Then enumerate an $s_{saw} \in S_{2k-1,saw,2}^{n_s+1}$ into X that is not queried negatively in the computation (if there does not exist one in X already).

done := true;

Else, $saw := saw - 1$.

Until (saw=0) or (*done*).

If $saw = 0$, take any admissible extension $\hat{X}_s$ of X_s satisfying $\neg q_{2k-1,j}^{n_s+1}$, for$1 \le j \le 2k-1$, then extend X_s according to $\hat{X}_s$ up to large block $\bigcup_{j=1}^{2k-1} S_{2k-1,j,2}^{n_s+1}$.

$s := s+1$.

Stage $s+1 = \langle 2k, i \rangle + 1$: We satisfy $R_{\langle 2k,i \rangle}$.

Step 1. Diagonalize C($L(N_{i_1}),\ldots,L(N_{i_k})$).

Let $n_{s+1} = \min \{n | n > n_s \wedge GCB_{k+1,n}$ is clean $\wedge\ 2^n > \sum_{j=1}^{k} p_{i_j}(n)\}$

$saw := k$; *done* := false;

Repeat

Consider all admissible extensions of X_s (except temporarily ignore $GC_{k+1,n_{s+1}}$) satisfying $(\neg p_{k+1,j}^{n_s+1}) \wedge (\neg p_{k+1,k+1+j}^{n_s+1})$, for $saw < j \le k+1$.

If $N_{i_{saw}}(1^{n_s+1})$ has an accepting computation under such an extension $\hat{X}_s$, we will "freeze" an accepting computation as follows:

Extend X_s according to $\hat{X}_s$ up to the <<-last block for which there exists a queried string (round up to a large block if the block is part of it) or $GCB_{k+1,n_{s+1}}$, whichever is later.

Then enumerate an $s_{saw+1} \in S_{k+1,saw+1,1}^{n_s+1}$ and an $s_{saw}{}' \in S_{k+1,k+1+saw,1}^{n_s+1}$ into X that are not queried negatively in the computation (if there do not exist such two strings in X already). *done* := true;

Else, $saw := saw - 1$

Until ($saw = 0$) or (*done*).

If $saw = 0$ take any admissible extension of X_s that has no string from $S^{n_s+1}_{k+1,j,1}$, for $1<j\le k+1$, and no string from $S^{n_s+1}_{k+1,k+1+j,1}$, for $1\le j\le k+1$, and has an $s_1 \in S^{n_s+1}_{k+1,1,1}$ (this satisfies $GC_{k+1,n_{s+1}}$). Then extend X_s up to large block $GCB_{k+1,n_{s+1}}$.

Step 2. Diagonalize $\overline{\mathbf{C}(L(N_{i_1}),\dots,L(N_{i_k}))}$.

We omit the details since this step is symmetric to Step 1.

Let's prove, say, that stage $\langle 2k-1,i\rangle+1$ does satisfy requirement $R_{\langle 2k-1,i\rangle}$. The other parts are left to the reader.

Suppose the Repeat loop ends with $saw > 0$. Then clearly for some admissible extension $\hat{X_s}$ of X_s satisfying $\neg q^{n_s+1}_{2k-1,j}$, $saw<j\le 2k-1$, $N_{i_{saw}}$ accepts 1^{n_s+1}. Meanwhile N_{i_j}, $saw<j\le 2k-1$, reject 1^{n_s+1} for all such admissible extensions of X_s.

There are at most $p_{i_{saw}}(n_{s+1})$ strings being queried on the accepting computation path. Thus an $s_{saw} \in S^{n_s+1}_{2k-1,saw,2}$ can be found and enumerated into X. Therefore the accepting computation is preserved, and $q^{n_s+1}_{2k-1,saw}$ is satisfied. Hence for the final X, $1^{n_s+1} \in \mathbf{C}(L(N_{i_{saw}}),\dots,L(N_{i_{2k-1}}))$.

If saw is odd, then $1^{n_s+1} \in L_{2k-1} - \overline{\mathbf{C}(L(N_{i_1}),\dots,L(N_{i_{2k-1}}))}$.

If saw is even, then $1^{n_s+1} \in \overline{\mathbf{C}(L(N_{i_1}),\dots,L(N_{i_{2k-1}}))} - L_{2k-1}$.

Similarly, suppose the Repeat loop ends with $saw=0$, then none of the N_{i_j}, $1\le j\le 2k-1$, will accept 1^{n_s+1} for any admissible extensions with $\neg q^{n_s+1}_{2k-1,j}$, $1<j\le 2k-1$. Since the final X has no string in $\bigcup_{j=1}^{2k-1} S^{n_s+1}_{2k-1,j,2}$, we have $1^{n_s+1} \in \overline{\mathbf{C}(L(N_{i_1}),\dots,L(N_{i_{2k-1}}))}$, and $1^{n_s+1} \notin L_{2k-1}$. ▸

3.2. Relativized Boolean Hierarchies Extending Exactly k Levels

An understanding of the boolean hierarchy may be essential to an understanding of the structure of feasible computations. In some relativized worlds, the boolean hierarchy separates at the k'th level, yet PSPACE falls into the $k+1$'st level. This fact opens the possibility that in the real world the structure of feasible computations degenerates exactly in the boolean hierarchy.

Theorem 3.2.1: For each $k \geq 1$, there is a recursive oracle A so that $\mathrm{NP}^A(k) \neq \mathrm{coNP}^A(k)$, yet $\mathrm{NP}^A(k+1) = \mathrm{coNP}^A(k+1) = \mathrm{PSPACE}^A$.

We state a proof for the case $k = 2$, and show the form of the general proof. Our strategy is simple. While coding to assure that $\mathrm{PSPACE}^A \subseteq \mathrm{NP}^A(3)$, we diagonalize so that $\mathrm{D^P}(A) \neq \mathrm{coD^P}(A)$.

Proof of Theorem 3.2.1 (k=2 case):

We display a recursive oracle A for which $\mathrm{D^P}(A) \neq \mathrm{coD^P}(A)$ and $\mathrm{NP}^A(3) = \mathrm{PSPACE}^A$.

Let M_i be an enumeration of PSPACE machines. W.l.o.g., M_i runs in space at most n^i+i and queries strings of length at most n^i+i. By adopting polynomial length queries, we follow the standard model of relativized PSPACE [FSS81].

Let $U = \{M_i \# x \# 1^{|x|^i+i} \mid M_i^A(x) \text{ accepts}\}$. Note that U is PSPACE^A complete. We will encode to ensure that $U \in \mathrm{NP}^A(3)$. In particular we'll assure that if y is of length i:

$$y \in U \Leftrightarrow [(\exists z, |z| = 4i+1)[yz \in A] \wedge (\forall z, |z| = 4i+2)[yz \notin A]] \vee (\exists z, |z| = 4i+3)[yz \in A] \quad (*)$$

Let $L = \{0^i \mid (\exists z, |z| = i)[z \in A]\} \bigcap \{0^i \mid (\forall z, |z| = i+1)[z \notin A]\}$. Clearly $L \in \mathrm{D^P}(A)$. We show that no $\mathrm{coD^P}(A)$ machine accepts L. The k'th $\mathrm{coD^P}$ machine, H_k, will be such that $L(H_k^A) = L(N_j^A) \bigcup L(A_l^A)$, $<j,l> = k$ and $\{N_i\}$, $\{A_i\}$ are enumerations of NP and coNP machines, respectively. That is, to "run" H_k, we "run" both N_j and A_l and accept if either accepts.

Our oracle is built in stages. The following constraint applies throughout the construction: In stage $5i+j$, $1 \le j \le 5$, no string of length less than $5i+1$ can be added to the oracle.

Oracle Construction

Stage "5i+1, 5i+2, 5i+3" (Encoding):

For each y of length i, assure that equation (*) is satisfied. A possible problem is that our ability to control strings of lengths $5i+1,2,3$ may be constrained by the effects of some previous diagonalization. Various cases of the diagonalizations put various types of constraints on us. Fortunately, for each type of constraint we retain the freedom of coding both possibilities: $y \notin U$ or $y \in U$. After we list the diagonalization cases, we give a recipe for codings done under each case.

Stage "5i+4, 5i+5" (Diagonalization):

If strings of length $\geq 5i+1$ have been queried by any previous diagonalization phase, do nothing during this stage. Otherwise, let H_k be the first $\mathrm{coD^P}$ machine that still might accept L. Let's say $k = <j,l>$ so, informally speaking, $H_k = N_j \bigcup A_l$. If either N_j or A_l has run time (on inputs of length $5i+4$) more than $2^{\sqrt{i}}$, do nothing.

Otherwise, run $N_j(0^{5i+4})$ with, as oracles, all extensions of A that add no new string having length congruent to three mod five.

Case 1: $N_j^A(0^{5i+4})$ accepts on some such extension

Freeze in A all elements along some accepting path. So $H_k^A(0^{5i+4})$ accepts. Make it a liar by adding some unconstrained length $5i+5$ string to A. We win, since $0^{5i+4} \in L(H_k^A)$ but $0^{5i+4} \notin L$.

Case 2: $N_j^A(0^{5i+4})$ **rejects on all such extensions**

Then, as long as we make sure all strings of lengths congruent to three mod five remain out of A, H_k is simply an overworked A_l, trying to accept a hard language, L. So, run $A_l(0^{5i+4})$ on all oracle extensions that 1) add no new string with length $\equiv 3 \bmod 5$, and 2) have no strings of lengths $5i+2$ or $5i+5$.

Case 2a: $A_l^A(0^{5i+4})$ **rejects on some such extension**

Freeze in A elements along some rejecting path. So $H_k^A(0^{5i+4})$ rejects. Make it a liar by putting an unconstrained length $5i+4$ string into A (recall that by the conditions imposed, we have no length $5i+5$ strings). We win, as $0^{5i+4} \notin L(H_k^A)$ but $0^{5i+4} \in L$.

Case 2b: $A_l^A(0^{5i+4})$ **accepts on all such extensions**

Then, in particular, it accepts on the extension that puts *no* strings of length $5i+4$ or $5i+5$ into A. $H_k^A(0^{5i+4})$ accepts but $0^{5i+4} \notin L$, so we have again made H_k a liar.

END OF CASES

Finally, we must provide the promised coding strategies. Since our diagonalizations only take place when out of the range of any previous diagonalization, each time we code we are constrained by at most one case of the diagonalization. Figure 7 shows these strategies.

For example, consider trying to code $y \in U$, when our last diagonalization succeeded on Case 2a. Case 2 obligates us to keep strings of lengths $\equiv 3 \bmod 5$ out of A. Also, it may have fixed some strings of length $5i+1$ in and out of A, and it may have fixed some strings of length $5i+2$ out of A. Nonetheless, we have no problem here. We code $y \in U$ by placing an unconstrained length $5i+1$ string into A. Now we have no length $5i+2$ strings in A; by (*) the new length $5i+1$ string codes the fact that $y \in U$. Note that it does not interfere with the diagonalization.

4. Immunity Results

We first prove a theorem that shows the difficulty of finding immunity relations within the boolean hierarchy: No BH set is NP bi-immune or coNP bi-immune. We attribute the following to Jockusch, who first proved an analogue for r.e. sets.

Theorem 4.1: No set in the boolean hierarchy can be NP bi-immune. That is, for each $S \in \mathrm{BH}$ there is an infinite set $L \in \mathrm{NP}$ such that either $L \subseteq S$ or $L \subseteq \overline{S}$. Similarly, no set in the boolean hierarchy is coNP bi-immune.

Proof of Theorem 4.1: By definition, $\mathrm{BH} = \bigcup_{k \geq 1} \mathrm{NP}(k)$. We prove the first part; the second part is similar, using $\mathrm{BH} = \bigcup_{k \geq 1} \{ \mathrm{C}(\overline{L_1}, \ldots, \overline{L_k}) | \overline{L_i} \in \mathrm{coNP} \}$

Our proof is by induction on the level of S in the boolean hierarchy. Our base case is that S is in NP. If S is infinite, set $L = S$. If S is finite set $L = \overline{S}$. In either case, we see that S is not bi-immune.

Suppose $S \in \mathrm{NP}(k)$, $k > 1$, and (inductively) no set in $\mathrm{NP}(k-1)$ is NP bi-immune. By definition, there are NP sets L_i, $1 \leq i \leq k$, for which $S = \mathrm{C}(L_1, \ldots, L_k)$. If k is odd, then $S = S_1 \bigcup L_k$, where $S_1 \in \mathrm{NP}(k-1)$. If L_k is infinite or S_1 has an infinite NP subset, then so does S. Otherwise, L_k is finite and $\overline{S_1}$ has an infinite NP subset X. Then $\overline{S} = \overline{S_1} \bigcap \overline{L_k} \supseteq X - L_k$. $X - L_k$ is an infinite NP set, so we're done.

A dual argument can be applied for k even. ▸

Is it possible for a set in a higher level of the boolean hierarchy, to be immune to lower levels? We show the following theorem.

Theorem 4.2: There is a recursive oracle X, such that $\mathrm{NP}^X(2)$ has a $\mathrm{coNP}^X(2)$−immune set.

Proof of Theorem 4.2: Since coNP(2) $= \{ \overline{L_i} \bigcup L_j | L_i, L_j \in \mathrm{NP} \}$, it is sufficient to construct an X for which some set in $\mathrm{NP}^X(2)$ is simultaneously NP^X-immune and coNP^X-immune.

As before, define $L = \{ 1^n | \ (\exists 0x \in X, \ |x| = n) \wedge \neg (\exists 1y \in X, |y| = n) \}$. Then $L \in \mathrm{NP}^X(2)$. We sketch the proof that L can be made $\mathrm{coNP}^X(2)-$immune.

X is constructed in stages. At each stage we keep a finite list of coNP(2) machines, represented as pairs of NP machines, that have not been "kicked out." At stage s, get a large n; we achieve the following objective:

> *either* some machine on the list accepts 1^n (this machine's accepting computation will be preserved) and we will arrange to have $1^n \notin L$ *or* no machine on the list accepts (this fact will also be preserved) and we will arrange to have $1^n \in L$.

In the former case, the "mistaken" machines, those we have fooled, are deleted from the list. In the second case, we append to the list one more $\langle N_i, N_j \rangle$. From stage s to $s+1$, we will arrange to have some $1y$ of the "skipped" length so that L is not enlarged.

Hence every $\overline{L(N_i)} \bigcup L(N_j)$ once on the list will not accept any more members of L, unless it is "kicked out" later. Since at any moment the list is finite, we can only "kick out" a finite number of $\langle N_i, N_j \rangle$'s consecutively. Thus the second possibility does occur infinitely often, so L is infinite and every $\overline{L(N_i)} \bigcup L(N_j)$ is eventually considered. Therefore L only has finite subsets from $\mathrm{coNP}^X(2)$.

The strategy to achieve that objective is the following:

Assume at some point there are k pairs $\langle N_{i_t}, N_{j_t} \rangle$, $1 \le t \le k$, on the list. Construct NP machines N_σ, N_τ such that,

$$\overline{L(N_\sigma)} \bigcup L(N_\tau) = \bigcup_{1 \le t \le k} (\overline{L(N_{i_t})} \bigcup L(N_{j_t})).$$

Consider all extensions that contain no string previously frozen to $\overline{X}$, see if N_τ accepts 1^n

under some such extension, and if so

(Case 1) Freeze one accepting computation path of that extension. Choose a $1y$ unfrozen with $|y|=n$; put $1y$ into X. Thus $1^n \notin L$ and we may kick out all mistaken pairs (there is at least one).

(Case 2) Otherwise, consider all extensions that have no string of the form $1y, |y|=n$; see if N_σ accepts 1^n on one of these extensions. If so

(Case 2a) freeze one accepting computation and add an unfrozen string $0x, |x|=n$ to X. Thus $1^n \in L$ and no pair on the list accepts 1^n.

(Case 2b) Otherwise, do nothing. At length n, we have neither $0x$ nor $1y$; hence $1^n \notin L$, but some $\overline{L(N_{i_l})}$ said yes to 1^n. ▸

On the other hand, it follows from our normal form theorem that the situation is drastically different for NP(2). By Theorem 2.1, every set in the boolean hierarchy can be expressed as a finite union of NP(2) sets. Clearly, this implies the following theorem.

Theorem 4.3: No set in the boolean hierarchy is NP(2)$-$immune.

The argument used to prove Theorem 4.3 yields more. In fact, for every L

$$L \text{ is NP(2)}-\text{immune} \Leftrightarrow L \text{ is BH}-\text{immune}.$$

The proof is immediate: If L has an infinite subset M from the boolean hierarchy then M in turn must have an infinite subset from NP(2). Thus L has an infinite NP(2) subset.

The above observation can be refined to prove the following theorem.

Theorem 4.4: There is a recursive oracle X so that $\Delta_2^{p,X}$ has a set that is BH^X-immune.

Proof of Theorem 4.4: Define predicates $p_k^n \equiv$ "$\exists\, x,\ |x|=n,\ 1^k 0x \in X$," $1 \le k \le 2n$.

(Note that $1^k 0x = 1^l 0y \Rightarrow k=l,\ x=y$.)

Consider $L_\Delta = \{ 1^n | C(p_1^n,...,p_{2n}^n) \}$. Clearly $L_\Delta \in \Delta_2^{p,X}$.

We will construct X, so that L_Δ is BH^X-immune.

X is constructed in stages; $X = \bigcup X_s$. At each stage we keep a finite list of BH machines, represented by finite sequences of NP machine indices. NP($2k$) consists of unions of k NP(2) sets. We may construct at stage s a finite sequence of NP indices $\langle i_1,...,i_t \rangle$, so that $C(L(N_{i_1}),...,L(N_{i_t}))$ is precisely the union of those on the list, relativized to any oracle set. Pick n sufficiently large, then apply our "sawing" argument to diagonalize..

Construction of X:

$X_0 = \varnothing$; $X_0' = \varnothing$; $list_0 = \varnothing$; $n_0 = 0$; $s = 0$.

Stage $s+1$:

Construct $\langle i_1,...,i_t \rangle$, so that for every X,

$$C(L(N_{i_1}^X),..., L(N_{i_t}^X)) = \bigcup L(M^X);$$

the union is over all $M \in list_s$, represented as finite sequences of NP indices via C.

Let p be the sum of time bounds of $N_{i_1},...,N_{i_t}$. W.l.o.g., t is even.

Let $n_{s+1} = \min \{ n \mid n > n_s$ and $2n > t$ and $(\forall m \geq n)[$ m is "clean"$]$ and $(\forall m \geq n)$ $[2^m > p(m)] \}$.

$\forall$ n: $n_s < n < n_{s+1}$, keep 1^n out of L_Δ as follows:

{Find $1^{2n}0x \notin X_s'$, $|x| = n$, set $X_s := X_s \bigcup \{1^{2n}0x\}$.}

Set $X_s' := X_s' \bigcup \{1^k 0x \mid |x| = n_{s+1}, t+1 < k \leq 2n_{s+1}\}$

(This effectively strips away the last $2n_{s+1} - (t+1)$ quantifiers in the definition of L_Δ for 1^{n_s+1}.)

For some $10x \notin X_s'$, set $X_s := X_s \bigcup \{10x\}$.

saw := 0; *done* := false;

repeat

{ If there is an extension Y of X_s, so that $X_s \subseteq Y, Y \cap X_s' = \emptyset$ and $N^Y_{i_{t-saw}}(1^{n_s+1})$ has an accepting computation, we will freeze one accepting computation as follows:

{ $X_s := X_s \bigcup$ { those in Y that are queried }

$X_s' := X_s' \bigcup$ { those not in Y that are queried }

Pick a $1^{t+1-saw}0x \notin X_s'$, $|x| = n_{s+1}$, and add it to X_s.

done := true.

}

else $\{X_s' := X_s' \bigcup \{1^{t+1-saw}0x \mid |x| = n_{s+1}\}$; $saw := saw + 1\}$

}

until ($saw = t$) or ($done =$ true)

If saw is even then append the next BH machine to $list_s$

else delete all $M \in list_s$ with $1^{n_s+1} \in L(M^{X_s})$ (there must be one)

$X_{s+1} := X_s$, $X_{s+1}' := X_s'$, $list_{s+1} := list_s$, $s := s+1$

End of stage $s+1$

End of construction

We leave the proof that the construction works to the reader.

5. Sparse Sets in the Boolean Hierarchy and E=?NE

5.1. Basic Results

In 1983, Hartmanis, Immerman, and Sewelson [HSI83b] noted a connection between the density of sets in NP and the structure of exponential time classes. They proved the following theorem, in which E = $\bigcup_{c>0}$ TIME$[2^{cn}]$ and NE = $\bigcup_{c>0}$ NTIME$[2^{cn}]$.

Theorem 5.1.1 [HSI83b]: The following are equivalent:

1) There is a sparse set in NP$-$P.

2) There is a tally set in NP$-$P.

3) E$\neq$NE.

Corollary 5.1.1 [HSI83b]: There is a recursive set A for which $E^A = NE^A$ and $coNP^A - P^A$ has sparse sets.

This section considers the implications of the existence of sparse sets in the boolean hierarchy. We start by noting that there are tally sets in BH$-$P iff E$\neq$NE. It follows that forcing tally sets from NP$-$P banishes them from the entire boolean hierarchy; that is, BH$-$P has tally sets iff NP$-$P does.

Theorem 5.1.2: There are tally sets in BH$-$P iff E$\neq$NE.

Proof of Theorem 5.1.2:

$\Rightarrow$: Assume E$=$NE. Let L be a tally set from the boolean hierarchy. In normal form, as the finite union of D^P sets, $L = (N_1 \cap A_1) \cup \dots \cup (N_z \cap A_z)$, where the $N_i \in$NP and $A_i \in$coNP. Since $L \subseteq 1^*$, $L=((N_1 \cap 1^*) \cap (A_1 \cap 1^*)) \cup \dots \cup ((N_z \cap 1^*) \cap (A_z \cap 1^*))$. When E$=$NE, all tally sets in NP $\cup$ coNP fall to P [HSI83b], so L is the boolean combination of P sets. Thus L is in P.

$\Leftarrow$: If E$\neq$NE, there are even tally sets in NP$-$P, by an easy padding procedure. ▸

Corollary 5.1.2: There are tally sets in NP$-$P iff there are tally sets in BH$-$P.

The behavior of sparse sets in the boolean hierarchy when E$=$NE at first seems exotic: sparse sets fall out of alternate levels. That is, NP$-$P, NP(3)$-$DP, NP(5)$-$NP(4), ... have no sparse sets when E$=$NE [Figure 6a]. Also, $coD^P$$-$coNP, ..., coNP($2k+2$)$-$coNP($2k+1$) have no sparse sets.

Surprisingly, there is a relativized world where sparse sets do fall from alternate levels, yet they find sanctuary in the gaps: P, $D^P - NP$, $NP(4) - NP(3)$, ... Since in this world the boolean hierarchy is striped with regions of sparseness and non-sparseness, we call this the candy cane relativization [Figure 6b]. Corollary 5.1.1 reflects the base case of this result.

Theorem 5.1.3:

a) $E = NE \Leftrightarrow (\forall k)[NP(2k+1) - NP(2k)$ has no sparse sets.[1]]

b) $E = NE \Rightarrow (\forall k)[coNP(2k+2) - coNP(2k+1)$ has no sparse sets.]

Proof of Theorem 5.1.3:

a) $\Leftarrow$: By Theorem 5.1.1.

$\Rightarrow$: Let $E = NE$ and T be a sparse set in $NP(2k+1)$. T can, by the definition of $NP(2k+1)$, be written $N \cup V$, where N is in NP and V is in $NP(2k)$. Since T is sparse, N must be sparse. So N is a sparse set in NP and thus is in P, since $E = NE$. $NP(2k)$ is closed under intersection with P sets, so T is in $NP(2k)$.

b) Similar. ▸

Theorem 5.1.4: There is a recursive set A for which

1) $E^A = NE^A$ (thus $NP^A - P^A$, $NP^A(3) - DP^A$, $\cdots$, $NP^A(2k+1) - NP^A(2k)$, $\cdots$ have no sparse sets) and

2) $(\forall k)[coNP^A(2k+1) - NP^A(2k+1)$ has sparse sets].

Proof: Deferred to section 5.3.

It seems strange that Theorem 5.1.3a is a complete characterization, yet Theorem 5.1.3b is only an implication. Can Theorem 5.1.3b be made into an "if and only if" statement? In certain relativized worlds, the answer is *no*, as it is easy to construct an oracle

[1] Recall that $NP(0) = P$.

for which NP−P contains sparse sets (thus by [HSI83b] E $\neq$NE) yet for which coNP = NP (so BH−coNP = $\varnothing$).

5.2. Capturable Sets in the Boolean Hierarchy

So far, we've noted that E=NE forces all tally sets out of BH−P, but may leave sparse sets. Is there some natural class of sparse sets that is richer than the class of tally sets and can be forced from the boolean hierarchy?[2] To study this question, we define *capturability* of sets, a notion we feel to be of general interest.

Definition 5.2.1: A set is **capturable** if it is a subset of some sparse set in NP.

Immunity of sets has been intensely studied (e.g., [SB84][R85]). A set is NP-immune if it contains no infinite NP subset. Our definition of capturability provides a complement to this notion. A set T is uncapturable if every NP superset of T is non-sparse. Intuitively, capturable sets are "easy" sparse sets.

Capturable sets are a rich class. Nonetheless, capturable sets are sufficiently simple that we can answer the question that motivated this section. When E=NE, all capturable sets in the boolean hierarchy fall into P.

We first prove the following weaker result.

Lemma 5.2.1: If E=NE, then BH−coNP has no capturable sets.

Proof of Lemma 5.2.1: Assume E=NE and let L be a capturable set from the boolean hierarchy. Write L in normal form as the finite union of D^P sets: $L = \bigcup_{i \le k_L} (N_i \cap A_i)$, where $N_i \in$NP and $A_i \in$coNP. Let S be the sparse NP set capturing L. Then $L = \bigcup_{i \le k_L} ((N_i \cap S) \cap A_i)$. But since E=NE each $N_i \cap S$ is in P, so L is the finite union of

[2]Certainly, for the class "subsets of P-printable sets [HY84][HH86]" this property holds. We seek a far richer class.

coNP sets and thus is itself in coNP.▸

Theorem 5.2.1: If E=NE, then all capturable sets in the boolean hierarchy are in P.

Proof of Theorem 5.2.1: Assume E=NE and let S be a capturable set in the boolean hierarchy, captured by S': $S \subseteq S' \in$ NP, S' and S are sparse. Let

$L = \{ n\#k \mid z_{n,k} \in S$, where $z_{n,k}$ is the k^{th} string (lexicographically) in $W_n = S' \bigcap (\Sigma+\varepsilon)^n \}$

Since E=NE, S' is P-printable [HY84], and by Lemma 5.2.1, $S \in$ coNP. Thus $L \in$ coNE, so $L \in$ E.

Since $L \in$ E, the following program accepting S is in P.

{Let $n := |x|$;
P-print W_n;
If $x \notin W_n$ then {reject x}
else {suppose x is the k^{th} string (lexicographically) in W_n;
accept x if and only if $n\#k \in L$.
}
}

▸

5.3. The Candy Cane Construction

Recall [Section 5.1] that the candy cane construction leaves sparse sets in patches throughout the boolean hierarchy [Figure 6b]. Our light-hearted name notwithstanding, this construction is somewhat involved. Nevertheless, though the tactics are elaborate, the strategy is straightforward. While coding an NE complete set into E, we simultaneously perform diagonalizations in the boolean hierarchy to insure an infinite class of sets sanctuary at various levels of the hierarchy.

We achieve $E^A = NE^A$ by coding into our oracle an NE-complete language. Let N be a machine so that for all oracles A, $L(N^A)$ is complete for NE^A. W.l.o.g. (i.e., by padding), N runs in time 2^{2n}. We'll code our oracle so $1\#x\#1^{2^{2|x|}} \in A \Leftrightarrow x \in L(N^A)$.

Simultaneously, we wish to offer sanctuary to sparse sets in coNP, coNP(3), coNP(5), ..., coNP($2k+1$), ... We define a family of languages $\{S_{2k+1}\}$, so that each S_{2k+1} is obviously in coNP($2k+1$). By careful diagonalization we show that, for each k, no NP($2k+1$) machine accepts S_{2k+1}. In addition, we poison the oracle to insure that each S_{2k+1} remains sparse.

To space out our diagonalizations so no two interfere with each other, we define widely spaced columns in which the diagonalizations are based. Let $C_k = \{m \mid (\exists i)[m = 2^{2^{<k,i>}}]\}$, where $<\cdot,\cdot>$ is a standard pairing function. Each diagonalization involving S_{2k+1} will use a diagonalizing string whose length is in C_{2k+1}.

Taking advantage of these columns as spacing mechanisms, define our sanctuary-finding sets as follows. Note that each S_{2k+1} is in coNP($2k+1$).

$$S_1 = \{x \mid |x| \in C_1 \wedge (\forall y \ni |y| = |x|+1 : 0xy \notin A)\}$$

$$S_3 = \{x \mid |x| \in C_3 \wedge$$

$$[(\forall y \ni |y| = |x|+1 : 0xy \notin A) \wedge [(\exists y \ni |y| = |x|+2 : 0xy \in A) \vee (\forall y \ni |y| = |x|+3 : 0xy \notin A)]]\}$$

$$S_{2k+1} = \{x \mid |x| \in C_{2k+1} \wedge [(\forall y \ni |y| = |x|+1 : 0xy \notin A) \wedge [(\exists y \ni |y| = |x|+2 : 0xy \in A) \vee [(\forall y \ni |y| = |x|+3 : 0xy \notin A) \wedge [\ldots[(\exists y \ni |y| = |x|+2k : 0xy \in A) \vee (\forall y \ni |y| = |x|+2k+1 : 0xy \notin A)] \ldots]]]]\}$$

Now, for each k, we must assure that S_{2k+1} is accepted by no NP($2k+1$) machine. We model an NP($2k+1$) language, H, as a collection of $k+1$ NP machines N_i and k coNP machines A_i, so $H = N_1 \bigcup [A_2 \bigcap [N_2 \bigcup [\ldots \bigcup [A_{k+1} \bigcap N_{k+1}] \ldots]]]$. We casually use the machines and their languages synonymously. W.l.o.g. let the i'th coNP($2k+1$) machine be composed of machines running each in time $n^i + i$.

Using the method of successive restrictions (the "sawing" technique) of Section 3, we easily perform the needed diagonalizations.

Candy Cane Construction:

Stage n, Part A:

If $n \notin \bigcup_{l \geq 0} C_{2l+1}$ do nothing. Otherwise, suppose $n \in C_{2k+1}$. Let H be the first NP($2k+1$) machine that might still accept S_{2k+1}; let's say that H is the l'th NP($2k+1$) machine. If $l > \frac{1}{2}\log n$ then skip Part A, otherwise go on. We show that $L(H^A) \neq S^A_{2k+1}$.

Find an x_0, $|x_0|=n$, such that for all y with $n \leq |y| \leq n+2k+1$, $0x_0y$ is *unspecified* (i.e., a string that has been neither placed in nor barred from A).

· Recall that $L(H)=L(N_1)\bigcup[L(A_2)\bigcap[L(N_2)\bigcup[\ \dots \bigcup[L(A_{k+1})\bigcap L(N_{k+1})]\dots]]]$.

Case 1: There is some extension of A so $N_1(x_0)$ accepts. Thus $x_0 \in L(H^A)$ on this extension. Fix the oracle strings queried along some accepting path of $N_1^A(x_0)$ on this extension. To assure that $x_0 \notin S_{2k+1}$, find a y, $|y|=n+1$, so $0x_0y$ is unspecified and add it to the oracle.

Case 2: For every extension of A, $N_1^A(x_0)$ rejects and for some extension of A so that $(\forall y \ni |y|=n+1 : 0x_0y \notin A)$, $A_2^A(x_0)$ rejects. Thus $x_0 \notin L(H^A)$ on this extension. Fix in A the elements along some rejecting path. Now assure $x_0 \in S_{2k+1}$ by adding an unspecified string $0x_0y$, $|y|=n+2$, to A.

Case 2j (1≤j≤k): Cases 1, 2, ..., $2j-1$ fail to hold and for some extension of A so that $(\forall y \ni n+1 \leq |y| \leq n+2j-1 : 0x_0y \notin A)$, $A^A_{j+1}(x_0)$ rejects. Fixing a rejecting path assures $x_0 \notin L(H^A)$. Insure that $x_0 \in S_{2k+1}$ by choosing a $|y|=n+2j$ so $0x_0y$ is unspecified and add it to our oracle.

Case 2j+1 (0≤j≤k): Cases 1, 2, ..., $2j$ fail to hold and for some extension of A so that $(\forall y \ni n+1 \leq |y| \leq n+2j : 0x_0y \notin A)$, $N^A_{j+1}(x_0)$ accepts. Fixing an accepting path puts x_0 in $L(H^A)$. Assure $x_0 \notin S_{2k+1}$ by choosing a y, $|y|=n+2j+1$, so $0x_0y$ is unspecified and add

$0x_0y$ to our oracle.

Case 2k+2: Cases 1, 2, ..., $2k+1$ fail to hold. So H rejects x_0 on any extension where $(\forall y \ni n+1 \leq |y| \leq n+2k+1 : 0x_0y \notin A)$. Freeze all these strings out of A. Now $x_0 \in S_{2k+1}$, but $x_0 \notin L(H^A)$.

All cases: If any of the cases 1,2,..., $2k+2$ has occurred, we must act to maintain the sparseness of S_{2k+1}. For each $x \neq x_0$, find a y_x, $|y_x| = |x|+1$ so $0xy_x$ has not yet been specified and add that string to A. Thus S_{2k+1} has at most one string of length n.

Stage n, Part B:

Recall that N is a machine so $L(N^A)$ is NE^A complete for all A. We will code so that $1\#x\#1^{2^{2|x|}} \in A \Leftrightarrow x \in L(N^A)$.

For each string x so $\log^2 n \leq |x| \leq \log^2(n+1)$, in order from the smallest such x to the largest, do the following. If there is an extension of A so $N^A(x)$ accepts, fix the elements along that path and add $1\#x\#1^{2^{2|x|}}$ to A. Otherwise, fix $1\#x\#1^{2^{2|x|}}$ as out of A.

End Construction

Notes on the Construction:

1) We can find an x_0 in Part A, as there are 2^n candidates and at most $2^{1+\log^2 n}2^{2\log^2 n} = 2n^{3\log n}$ strings of length at least n have been fixed. All these have been fixed in Part B. Our Part A actions are spaced double exponentially far apart in length. Each Part A action can't get near the strings involved in the next Part A diagonalization.

2) Similarly, we can find the strings y_x with which we poison strings to maintain sparseness in Part A.

3) In Part B, each coding string $1\#x\#1^{2^{2|x|}}$ will certainly be unspecified when we assign it. No Part B computation will have touched this string. Any Part A machine run at stage n

can't even reach things of length $n^{\log n}$, by our assumption that the i'th NP($2k+1$) machine is composed of machines running in time at most n^i+i and our choice of H. So no part A machine has touched our coding strings.

6. Complete Languages for the Boolean Hierarchy

Complete languages are important in the understanding of a complexity class; a complete language is the quintessence of its class.

Theorem 6.1: For each k, NP(k) has a complete language.

Proof of Theorem 6.1:

Fix $k \geq 1$, we claim that the universal language

$$U = \{ \langle i_1, i_2, ..., i_k, x, 0^{p_i(|x|)} \rangle \mid x \in \mathrm{C}(L(N_{i_1}), .., L(N_{i_k})), i = \langle i_1, ..., i_k \rangle \},$$

is complete for NP(k) with respect to polynomial time reductions, where $p_i(\cdot)$ is a sum of the polynomial time bounds for $N_{i_1}, ..., N_{i_k}$. Clearly U is in NP(k), owing to the padding $0^{p_i(|x|)}$. Given any $L \in$ NP(k), there exist $i_1, .., i_k$, so that $L = \mathrm{C}(L(N_{i_1}), .., L(N_{i_k}))$. Let $p_i(\cdot)$ be a sum of time bounds of N_{i_j}'s, then $x \to \langle i_1, ..., i_k, x, 0^{p_i(|x|)} \rangle$ is a one-to-one polynomial time reduction from L to U. ▸

Languages such as the one constructed in Theorem 6.1 are unnatural. Natural complete languages more forcefully demonstrate the usefulness of language classes. Natural complete languages exist for every level of the boolean hierarchy.

Define SAT(k) $= \{ \langle f_1, ..., f_k \rangle \mid \mathrm{C}(p_1, ..., p_k) \}$, where $p_i \equiv$ "f_i is satisfiable." By the proof of Cook's Theorem [C71] we have the following theorem.

Theorem 6.2: SAT(k) is NP(k)-complete.

Indeed, using the normal forms of Section 2, we may assume in the definition of SAT(k) that $(\forall i<j)\,[f_j$ is satisfiable $\Rightarrow f_i$ is satisfiable$]$ and SAT(k) remains NP(k)-

complete.

We continue on our quest for natural complete languages. Let f be a boolean formula. Then f is satisfiable $\Leftrightarrow$ $f' = (f \vee y) \wedge \bar{y}$ is satisfiable where y is a new variable. Furthermore, $f'' = (f \vee y) \wedge z$ is always satisfiable. Using these observations, we can apply the standard transformation from SAT to 3SAT to Vertex Cover (VC) [GJ79]; we get a polynomial time transformation $f \rightarrow (G_f, l_f)$, so that if f is satisfiable, then the minimum size vertex cover for G_f is l_f; otherwise, that minimum number is l_f+1.

For each $k \geq 0$, define:

$$\mathrm{VC}(2k+2) = \{\langle G, l\rangle \mid \text{ The minimum size VC of } G \text{ is } l+2i+1,\ 0 \leq i \leq k\},$$

$$\mathrm{VC}(2k+1) = \{\langle G, l\rangle \mid \text{ The minimum size VC of } G \text{ is } \leq l, \text{ or equal to } l+2i \text{ for some } 0 < i \leq k\}.$$

Theorem 6.3: VC(k) is NP(k)-complete.

Proof of Theorem 6.3: Start with formulae $f_1,...,f_k$, where $(\forall\ i<j)\,[f_j \in \mathrm{SAT} \Rightarrow f_i \in SAT]$. Then apply the transformation described before to each f_i separately: f_i maps to $\langle G_i, l_i\rangle$. We get a graph G having k distinct components G_i, $1 \leq i \leq k$. Let $l = \Sigma l_i$.

Clearly the minimum size of a vertex cover for G is $l+j$, where $j =$ the number of f_i's that are unsatisfiable. Hence, $\langle f_1,..., f_k\rangle \in$ SAT(k) $\Leftrightarrow \langle G,l\rangle \in$ VC(k). ▸

One can elaborate on several variants of the Vertex Cover problem, such as Independent Set, Clique, and others. As another example, we consider the graph coloring problem. Using the same trick, one gets a polynomial time transformation, $f \rightarrow G_f$, such that $\chi(G_f)=3$ if $f \in$SAT, and $\chi(G_f) = 4$ otherwise [CM85].

Define

$$\mathrm{COLORABILITY}(k) = \{G \mid \chi(G) \leq 3k \text{ or } \chi(G) = 3k+2i, \text{ for some } 0 < i < \frac{k}{2}\}, \text{ when } k \text{ is odd.}$$

$$\mathrm{COLORABILITY}(k) = \{G \mid \chi(G) = 3k+2i+1, 0 \leq i < \frac{k}{2}\}, \text{ when } k \text{ is even.}$$

Theorem 6.4: COLORABILITY(k) is NP(k)-complete.

Proof of Theorem 6.4: We will reduce SAT(k) to COLORABILITY(k).

If $G_i = (V_i, E_i)$, $i=1,2$, define $G_1 \times G_2 = (V, E)$, where $V = V_1 \bigcup V_2$, and $E = E_1 \bigcup E_2 \bigcup \{\langle v_1, v_2 \rangle \mid v_i \in V_i\}$. Clearly $\chi(G_1 \times G_2) = \chi(G_1) + \chi(G_2)$. Given $\langle f_1, \ldots, f_k \rangle$ as before, let $f_i \rightarrow G_i$ be the transformation described before. Then $G = (\ldots(G_1 \times G_2) \times G_3 \cdots) \times G_k$ is just what we want. ▸

There are other results about complete languages in the intersection of each level of the boolean hierarchy, their existence, and properties they have if they do exist. We defer them to a subsequent paper.

7. Counting Classes

Counting classes at first seem tempting as natural complete sets for the boolean hierarchy. In this section, we define the notion of a counting class, and display relativized worlds where counting classes are *not* complete for the levels of the boolean hierarchy.

NP is the class of sets accepted by nondeterministic polynomial-time Turing machines [NPTMs] that (by definition) accept when *at least one* computation path accepts. US is the class of sets accepted by NPTMs that accept when *exactly one* computation path accepts. [BG82] shows there are relativized worlds where UNIQUE-SAT, a natural complete set for US, is and is not complete for D^P.

We are interested in more generalized notions of machines that count accepting paths. Our machines accept an input if the number of accepting paths generated by that input is in a prescribed set. These counting classes have natural complete sets: formulas with specified numbers of satisfying assignments. By way of contrast, counting classes in which certain numbers of accepting paths are forbidden have no complete sets in appropriately relativized worlds [HHta].

Please note that throughout Section 7, **even when not explicitly mentioned, every set named S is a subset of {1,2,3,...} and is either finite or cofinite.**

Definition 7.1:

a) SAT_S is the set of all formulas whose number of satisfying assignments is in S. For example, SAT is $\mathrm{SAT}_{\{1,2,3,\ldots\}}$ and UNIQUE-SAT is $\mathrm{SAT}_{\{1\}}$.

b) CP_S is the class of sets accepted by NPTMs that accept, by definition, when they have a number of accepting computations in S. Mnemonically, CP stands for "counting polynomial time." Examples: NP is $\mathrm{CP}_{\{1,2,3,\ldots\}}$ and US is $\mathrm{CP}_{\{1\}}$.

Lemma 7.1:

a) SAT_S is many-one complete for CP_S.

b) For any S and any complexity class C closed downwards under many-one polynomial time reductions,

$$\mathrm{SAT}_S \text{ is } A\text{-hard}^3 \Leftrightarrow \mathrm{CP}_S \supseteq A.$$

For example, SAT_S is NP-hard $\Leftrightarrow \mathrm{CP}_S \supseteq \mathrm{NP}$.

c) For S finite, $\mathrm{SAT}_S \supseteq \mathrm{coNP}$. For S cofinite, $\mathrm{SAT}_S \supseteq \mathrm{NP}$.

Proof of Lemma 7.1:

a) Cook's reduction [C71,GJ79] is parsimonious.

b) Immediate.

c) Let S be finite. Let a_z be the largest element of S. Given a formula f, we can quickly either find a solution (so f is not in UNSAT) or find a_z non-satisfying assignments. Thus f with these assignments ORed on has a_z more solutions than f, and is thus in SAT_S if and only if f is in UNSAT, thus SAT_S is coNP-hard. Thus we are done by part b.

Similarly, for S cofinite, SAT_S is NP-hard because we can add j true assignments to a formula, $j = \max\{i \mid i \geq 0 \wedge i \notin S\}$. ▸

Clearly, counting of this sort can be done in hardware. For example, a set in CP_S, $|S| = z$, is the union of z $\mathrm{D^P}$ sets, and so is in NP($2z$). Similarly, for cofinite S, $\mathrm{CP}_S \subseteq \mathrm{coNP}(2y)$, when $|\bar{S}| = y$. The containment can easily be made an equality in relativized worlds. Is the containment always an equality?

We show that in appropriately relativized worlds counting is weak. For example, we display a world where for *no* choice of S is SAT_S complete for any NP(k) or coNP(k), $k > 1$! Furthermore, in this world for no choice of finite S can SAT_S even be NP-hard and for no choice of cofinite S can SAT_S be coNP-hard (where we relativize satisfiability in the standard way— adding membership testing gates). Indeed, each counting class with S, S finite (cofinite), fails to contain NP (coNP) even when its run time is boosted from polynomial to a certain outright *exponential* time.

Similarly, we separate many counting classes so strongly that even vast boosts in run time fail to give containment. All these relativized containments are obtained by combinatorially exploiting the limited control of counting machines. On the other hand, we note that some classes *are* nicely ordered.

7.1. A Counting Hierarchy

Suppose you have a party at which each guest may know the names of some of the other guests. The following combinatorial lemma states conditions under which your party will contain a large set of "strangers": each person in the set of strangers knows the name of no person in the strangers set (except perhaps his own name). This lemma will be useful in proving our separations.

Combinatorial Lemma:

$$(\forall S_1,\ldots,S_l \subseteq \mathbf{N})\ [\sum |S_i| < \left\lceil \frac{l(l-1)}{k(k-1)} \right\rceil \Rightarrow (\exists T \subseteq \{1,2,\ldots,l\},\ |T| \geq k)(\forall e \in T)[S_e \cap (T-e) = \emptyset]]$$

Proof of Combinatorial Lemma: Indeed, we show there is a T of size exactly k. There are $\binom{l}{k}$ candidate k-tuples for T. Each element added to some set destroys the candidacy of at most $\binom{l-2}{k-2}$ new tuples. E.g., adding $x \leq l$ to set S_y eliminates tuples containing both x and y. Thus we need at least $\left\lceil \binom{l}{k} \Big/ \binom{l-2}{k-2} \right\rceil = \left\lceil \frac{l(l-1)}{k(k-1)} \right\rceil$ elements to eliminate all tuples. ▸

Our first theorem shows that "the number of values accepted on" yields a hierarchy: *more counting is more powerful*, and strictly so in some worlds.

Theorem 7.1.1:

a) $\mathrm{CP}_{\{1\}} = \bigcup_{|S|=1} \mathrm{CP}_S \subseteq \bigcup_{|S|=2} \mathrm{CP}_S \subseteq \bigcup_{|S|=3} \mathrm{CP}_S \subseteq \cdots \subseteq \Delta_2^p$.

b) There is a recursive oracle A so $\bigcup_{|S|=1} \mathrm{CP}_S^A \subset \bigcup_{|S|=2} \mathrm{CP}_S^A \subset \bigcup_{|S|=3} \mathrm{CP}_S^A \subset \cdots$.

Proof of 7.1.1, Part a:

Clearly counting is in Δ_2^p. Indeed, CP_S questions can be answered with at most $2z$ queries to a SAT oracle, when $|S| = z$. If $L \in \mathrm{CP}_{\{a_1,\ldots,a_z\}}$, there is a $\mathrm{CP}_{\{1,2+a_1,2+a_2,\ldots,2+a_z\}}$ machine that simulates the $\mathrm{CP}_{\{a_1,\ldots,a_z\}}$ machine for L, and mindlessly adds two accepting paths. This new machine accepts L. Since this holds in general, $\bigcup_{|S|=1} \mathrm{CP}_S \subseteq \bigcup_{|S|=2} \mathrm{CP}_S \subseteq \bigcup_{|S|=3} \mathrm{CP}_S \subseteq \cdots$. Finally, $\mathrm{CP}_{\{j\}} = \mathrm{CP}_{\{1\}}$ from Theorem 7.2.1.1. ▸

Proof of Theorem 7.1.1, Part b:

---

[3]We always use hard in the many-one sense.

Because of space limitations, we just show a world in which $\bigcup_{|S|=1} \mathrm{CP}_S^A \subset \bigcup_{|S|=2} \mathrm{CP}_S^A$. By Theorem 7.1.1, part (a), it suffices to show a language L in $\mathrm{CP}_{\{1,2\}}^A$ that is not in $\mathrm{CP}_{\{1\}}^A$. We diagonalize over all $\mathrm{CP}_{\{1\}}$ machines to show that $L = \{0^i \mid 1 \leq |\Sigma^i \cap A| \leq 2\}$ is not in $\mathrm{CP}_{\{1\}}$. Since L is clearly in $\mathrm{CP}_{\{1,2\}}$ this suffices. Initially, set $A := \emptyset$.

Stage s: We'll establish that the s^{th} $\mathrm{CP}_{\{1\}}$ machine, call it C_i, does not accept L. Let $p(\cdot)$ be the polynomial bound on the length of each of C_i's computation paths (and thus on the length of the strings it queries). Choose j so large that no string of length j or larger has yet been touched, and so large that $p(j) < (2^j - 1)/2$.

Case 1: $C_i^A(0^j)$ has accepting paths.

If it has one accepting path, freeze A for all strings of length up to $p(j)$; now $0^j \notin L$ but $0^j \in L(C_i^A)$.

If it has more than one accepting path, fix the elements along two paths, put some unfrozen length j string into A, and then freeze A for all strings of length up to $p(j)$; now $0^j \in L$ but $0^j \notin L(C_i^A)$.

Case 2: For some length j string y, $C_i^{A \cup y}(0^j)$ rejects.

Set $A := A \cup \{y\}$ and freeze A for all strings of length up to $p(j)$. Now $0^j \in L$, $0^j \notin L(C_i^A)$.

Case 3: For all length j strings, y: $C_i^{A \cup y}(0^j)$ accepts (i.e., it has a unique accepting path).

For each length j string w define a set

$$S_w = \{z \mid |z| = j \text{ and } C_i^{A \cup w}(0^j) \text{ queries } z \text{ along its accepting path}\}.$$

We use the Combinatorial Lemma from the start of this section with $k = 2$, $l = 2^j$, and using the 2^j sets S_w for $|w|=j$. The precondition of the lemma is met since by our conditions on the choice of j we have $2^j p(j) < 2^j \frac{2^j-1}{2}$. Thus there exist strings y', y'' so y' is not in $S_{y''}$ and y'' is not in $S_{y'}$ Thus if we add y' and y'' simultaneously to A we will get at least two accepting paths: the ones corresponding to $S_{y'}$ and $S_{y''}$. (Why are these two paths distinct? If $y'' \notin S_{y''}$ then we have an accepting path in $C_i^A(0^j)$, and this was treated in Case 1. Hence $S_{y''}$ and $S_{y'}$ differ on y''.) So $C_i^{A \cup y' \cup y''}(0^j)$ has at least two accepting paths and rejects. Setting $A := A \bigcup \{y', y''\}$ and fixing A up to length $p(j)$ we have insured that $0^j \in L$ but $0^j \notin L(C_i^A)$. ▸

More counting gives more power, as these theorems show. Indeed, we have an infinite hierarchy of counting power. Attempts to make the polynomial hierarchy infinite in some relativization took many years to succeed [BS79,Y85]. Thus it is somewhat surprising that Theorem 7.1.1 shows a natural infinite hierarchy.

Nonetheless, the most interesting aspect of counting is that it can be made *weak*. Below, we show that counting is too weak to even capture NP or coNP. As a consequence, we show that no amount of counting is hard for even low levels of the boolean hierarchy.

Theorem 7.1.2: There is a recursive oracle A for which $\bigcup_{S \text{ finite}} \mathrm{CP}_S^A$ does not contain NP^A and $\bigcup_{S \text{ cofinite}} \mathrm{CP}_S^A$ does not contain coNP^A.

Corollary 7.1.1: There is a relativized world where *no* amount of counting is hard for $\mathrm{NP}^A(k)$ or $\mathrm{coNP}^A(k)$, $k>1$ (i.e., where no counting class contains either $\mathrm{NP}^A(k)$ or $\mathrm{coNP}^A(k)$, $k>1$).

Proof of Corollary 7.1.1: $\mathrm{NP}^A(k)$ and $\mathrm{coNP}^A(k)$ are each NP^A-hard and coNP^A-hard for every $k>1$, yet Theorem 7.1.2 shows that the artificial and natural complete languages for counting classes fail to be hard for at least one of these. ▸

Proof of Theorem 7.1.2:

Let $L_A = \{0^n | (\exists y)[|y|=n \wedge y \in A]\}$. Let $LL_A = \{0^n | (\forall y)[|y|=n \Rightarrow y \notin A]\}$. Clearly $L_A \in \mathrm{NP}^A$ and $LL_A \in \mathrm{coNP}^A$.

Stage $<<i,z,a_1,...,a_z>,0>$, $0<a_1<...<a_z$:

We show that the i^{th} NP machine, viewed as a $\mathrm{CP}^A_{\{a_1,...,a_z\}}$ machine, does not accept L_A. W.l.o.g., NP_i runs in Ntime$[n^i+i]$.

Choose m so large that nothing of length greater than m has yet been fixed, and so large that $2^m(m^i+i) < \left\lfloor \frac{2^m(2^m-1)}{a_z(a_z+1)} \right\rfloor$. Let B be the current version of A. Run $N_i^B(0^m)$.

Case 1: $N_i^B(0^m)$ accepts. $L(N_i^B) \neq L_A$ and we're done.

Case 2a: $N_i^B(0^m)$ rejects and for one of the 2^m ways we can add a length m string, y, to B, $N_i^{B \cup y}(0^m)$ rejects. Set our current A to $B \cup y$ and again $L(N_i^A) \neq L_A$.

Case 2b: $N_i^B(0^m)$ rejects and for all of the 2^m ways we can add a length m string, y, to B, $N_i^{B \cup y}(0^m)$ accepts. We apply the combinatorial lemma (from the start of this section) to get a contradiction!

That is, for each of the 2^m extensions, $B \cup y$, mark one accepting path of $N_i^{B \cup y}(0^m)$ and say the path is *named* by y. Let S_y denote the strings queried on the path named y. By our assumption that N_i runs in Ntime$[n^i+i]$, $\sum |S_y| \leq 2^m(m^i+i)$. By our choice of m, the combinatorial lemma applies (with $l=2^m$, $k=a_z+1$) and chooses for us a set T of a_z+1 special marked paths, none of which queries the name of any of its a_z brethren. Furthermore, the a_z+1 paths are all distinct, since each one contains its own name and none of its brethren does. Each special path does indeed contain its name, otherwise it would have caused $N_i^B(0^m)$ to accept, but we supposed it didn't. Thus $N_i^{B \cup T}(0^m)$ has at least a_z+1 accepting paths and, viewed as (i.e., with the acceptance mechanism of) a

$\mathrm{CP}^A_{\{a_1,...,a_z\}}$ machine, rejects. Yet $0^m \in L_A$. So setting our current A to $B \bigcup T$, $L(H_i^A) \neq L_A$ and we have diagonalized away another machine.

Stage <<i,z,a$_1$,...,a$_z$>,b>, 0<a$_1$<...<a$_z$<b :

We show that the i^{th} NP machine, viewed as a $\mathrm{CP}^A_{\{a_1,...,a_z,b,b+1,b+2,\ \cdots,\}}$ machine, does not accept LL_A.

Choose m large. Denote by B the current version of A. Run $N_i^B(0^m)$. If it rejects, we're done. If it accepts, for each of the 2^m ways we can add a length m string, y, to B, run $N_i^{B \bigcup y}(0^m)$. If any of these 2^m ways accepts, we're done. Now if extension $B \bigcup y$ has no accepting paths or has some accepting paths but queries y on none of them, y must appear on some accepting path of $N_i^B(0^m)$. Thus all but at most $(b-1)m^i+i$ of our 2^m extensions cause "nice" rejections: rejections (with oracle $B \bigcup y$) where there is some accepting path that queries y.

From here, we are done by the argument of the previous part. In brief: for each "nice" extension mark a path that queries the extension's name; appeal to the combinatorial lemma to get a set T of b paths that do not contain each others names; thus, $N_i^{B \bigcup T}(0^m)$ has b accepting paths and accepts, but $0^m \notin LL_A$, thus $L(N_i^A) \neq LL_A$. ▸

Indeed, counting is even weaker than these results suggest. By making full use of our combinatorial control arguments, we can show that even giving counting machines *exponential* run times does not allow them to subsume NP.

Theorem 7.1.3: There is a recursive oracle A for which $\mathrm{NP}^A \not\subseteq \bigcup_{S \text{ finite}} \mathrm{CX}_S^A$, where CX means that our time bound has been boosted from P to $\bigcup_{c \geq 0} \mathrm{Time}[c + \frac{2^n}{(a_z+1)^2}]$, where a_z is the maximum element of S.

Corollary 7.1.2: There is a recursive set A for which $NP^A \not\subseteq \bigcup_{c,\ \text{finite}\ S} \text{CTime}[2^{\frac{cn}{\log n}}]_S^A$.

Proof of Corollary 7.1.2: The corollary follows directly from Theorem 7.1.3. ▸

Proof of Theorem 7.1.3: In our proof of Theorem 7.1.2, we have a lot of slack in our use of the combinatorial lemma. Pushing it to the limit allows the exponential time claimed. That is, we add c to our stage pairing function and choose m so large that $2^m(c+\frac{2^m}{(a_z+1)^2}) < \frac{2^m(2^m-1)}{a_z(a_z+1)}$; more simply, $0 < \frac{1}{a_z}2^{2m} - \left[\frac{a_z+1}{a_z} + c(a_z+1)^2\right]2^m$. ▸

Indeed, the same tricks apply within the counting hierarchy. For example, there is a world where

$$\bigcup_{c,|S|=z} \text{CTime}[2^{\frac{cn}{\log n}}]_S^A \not\supseteq \bigcup_{|S|=z+1} \text{CP}_S^A.$$

7.2. Relations and Natural Containments among the CP Classes

7.2.1. Relations Based on Values

The previous section shows that "the number of values accepted on" gives a hierarchy. In this section, we fix the *number* of values, k, our classes accept on and see what effects the actual *values* have. For $k=1$, the number of paths makes no difference. We can multiply paths (to show $\text{CP}_{\{1\}} \supseteq \text{CP}_{\{k\}}$) and simulate k-tuples of paths (to show $\text{CP}_{\{k\}} \supseteq \text{CP}_{\{1\}}$) [see, for example, Theorem 7.2.2.1]. Hence we have the following theorem.

Theorem 7.2.1.1: For every j, k and A, $\text{CP}_{\{j\}}^A = \text{CP}_{\{k\}}^A$.

The situation differs markedly when $k=2$. Our classes here are $\text{CP}_{\{a,b\}}$, and we'll call a the *minor* subscript and b the *major* subscript. Here (with proper relativizations) a class never contains any class with a larger major subscript (Theorem 7.2.1.2). When the major subscripts are the same, we can usually relativize so the class with the larger minor subscript is not contained by the other. On the other hand, there is more than lexicographical ordering at work here. We can also relativize away from lexicographical ordering. Indeed, one can combine the relativizations to obtain a world where, for example, $\mathrm{CP}^A_{\{1,\,73\}} \underset{\not\supseteq}{\not\subseteq} \mathrm{CP}^A_{\{40,\,73\}}$.

Theorem 7.2.1.2:

a) There is a recursive oracle A so for $i<j<j'$, $\mathrm{CP}^A_{\{i,\,j\}} \subset \mathrm{CP}^A_{\{i,\,j'\}}$.

b) There is a recursive oracle A so that for all $1<j<k$ $\mathrm{CP}^A_{\{1,\,k\}} \not\subseteq \mathrm{CP}^A_{\{j,\,k\}}$.

c) There is a recursive set A so that for all i, j, k, l with $i<k, j<k, jl\le k$, and $i+l>k$, $\mathrm{CP}^A_{\{i,\,k\}} \not\supseteq \mathrm{CP}^A_{\{j,\,k\}}$.

Corollary 7.2.1.1: There is a world where no $\mathrm{CP}^A_{\{i',\,j'\}}$ contains $\bigcup_{i,\,j}\mathrm{CP}^A_{\{i,\,j\}}$.

Our results extend naturally to $k=3,4,\ldots$, and by interlacing the relativizations, we can simultaneously obtain these results for all k.

7.2.2. Containments

As we've just seen, combinatorial manipulations allow us to construct exotic and perverse relativized separations. Yet these tricks give a deceptive impression of chaos. There is a clear and ordered set of containments among counting classes that holds universally. Figure 2 shows the containments implied by Corollary 7.2.2.1. Figure 3 shows further containments from Theorem 7.2.2.1.

Theorem 7.2.2.1: For $z\ge 1$, $l\ge 1$, $c_l\ge 1$, $k\ge 0$, and $i\ge \min(k_l)\ge 1$:

$$\mathrm{CP}_{\{i,\,j\}} \subseteq \mathrm{CP}_{\left\{c_1\binom{i}{k_1}+c_2\binom{i}{k_2}+\ldots+c_z\binom{i}{k_z}+k\ ,\ c_1\binom{j}{k_1}+c_2\binom{j}{k_2}+\ldots+c_z\binom{j}{k_z}+k\right\}}$$

Corollary 7.2.2.1:

a) $\mathrm{CP}_{\{i,j\}} \subseteq \mathrm{CP}_{\{i,j'\}}$ when $i<j<j'$.

b) $\mathrm{CP}_{\{i,j\}} \subseteq \mathrm{CP}_{\{ki+l,\ kj+l\}}$ for $k \geq 1$, $l \geq 0$.

Proof of Theorem 7.2.2.1: Simulate the $\mathrm{CP}_{\{i,j\}}$ machine by simulating with c_l-fold replication each k_l-tuple of its paths (accepting and replicating the tuple if all k_l paths accept). Also, add k dummy accepting paths. If the $\mathrm{CP}_{\{i,j\}}$ machine has i (j) accepting paths, we have $f(i)$ ($f(j)$) accepting paths, where $f(m) = \sum c_l \begin{bmatrix} m \\ k_l \end{bmatrix} + k$. If the number of paths is neither i nor j, we will *not* get either $f(i)$ or $f(j)$ paths, as f is strictly increasing in the range $[\min(k_l)-1, \infty)$. ▸

8. A Log-bounded Hierarchy

This paper discusses the boolean hierarchy, a hierarchy of classes of sets obtained by boolean operations on NP sets. The levels of the boolean hierarchy have a rigidly defined structure of logical operations. If we relax this structure and allow, for example, unbounded size gates, we quickly end up with trivialities. Let Σ_j^b be the class of sets obtained by a depth j logic tree whose top level is OR, whose levels alternate between ANDs and ORs, and whose leaves are predicates from $\mathrm{NP} \bigcup \mathrm{coNP}$ [Figure 4]. Lemma 2.2.1 implies that $\Sigma_2^b = \Pi_2^b = \bigcup_j \Sigma_j^b$. That is, this new hierarchy collapses.

In this section, we briefly mention a more general hierarchy, which does not obviously collapse. The polynomial hierarchy is composed of levels, Σ_i^p, described by alternating TM's that spend at most polynomial time in each of their i alternation blocks. The Σ_j^b classes above correspond to alternating polynomial time Turing machines that have j constant time initial blocks, followed by a polynomial final alternation block on each path. We define Σ_j^L to be the set of languages accepted by a nondeterministic TM that has j

short initial blocks, each of at most $k\ log(|\text{input}|)$ steps, the first of these j blocks is existential, the second is universal, and so on; after the jth short block follows a single polynomial-length alternation block (which may be existential on some paths and universal on others).

Clearly this hierarchy is at least as powerful as the fixed hardware of previous sections. We now state some basic facts about this hierarchy, analogous to well-known facts about the polynomial hierarchy, and classify L_1, a natural problem from it as $\Pi_2^L(=\text{co}\Sigma_2^L)$ hard. This provides a natural example of the place where hardware over NP no longer suffices. In the relativized world where the boolean hierarchy is infinite, (the appropriately relativized version of) this problem is not in the boolean hierarchy (as L_1 is BH-hard by Theorems 8.1b and 8.2, so if $L_1 \in \text{BH}$ then the boolean hierarchy is finite).

Theorem 8.1:

a) $\text{D}^\text{P} = \Pi_1^b = \Pi_1^L$. $\text{coD}^\text{P} = \Sigma_1^b = \Sigma_1^L$.

b) $\Sigma_2^b = \Pi_2^b = \bigcup_j \Sigma_j^b = \text{BH} \subseteq \Sigma_2^L \cap \Pi_2^L$.

c) $\bigcup_j \Sigma_j^L \subseteq \Delta_2^p$.

d) (Downward Separation) If $\Sigma_j^L = \Pi_j^L$ then $\Sigma_j^L = \bigcup_i \Sigma_i^L$.

e) If $\bigcup_j \Sigma_j^L = \Delta_2^p$ then for some k, $\Sigma_k^L = \Pi_k^L$.

Theorem 8.2: Let SAT-UNSAT $= \{f\#g \mid f \in \text{SAT} \wedge g \in \text{UNSAT}\}$, the canonical D^P-complete problem. Let $L_1 = \{T \mid T$ is a list of pairs of formulas, *exactly* one of which belongs to SAT-UNSAT$\}$. Then $L_1 \in \Sigma_3^L \cap \Pi_3^L$ and L_1 is Π_2^L-hard.[4]

[4]With respect to many-one polynomial time reductions.

9. The Boolean Hierarchy and Sparse Oracles for NP

[M82] shows that if NP has a sparse many-one set then P=NP. We say that NP has a sparse oracle if there is a sparse set S so $\mathrm{NP} \subseteq \mathrm{P}^S$. [KL80][H82] show that if NP has a sparse Turing complete set or a sparse oracle, then $\Sigma_2^p = \Pi_2^p$. Does the existence of a sparse oracle for NP imply something stronger than $\Sigma_2^p = \Pi_2^p$? By displaying a relativized world where NP has a sparse oracle but where the boolean hierarchy is infinite (thus $\mathrm{P} \neq \mathrm{NP} \neq \mathrm{D^P} \neq \Sigma_2^p$), we show that the conclusion of [KL80] cannot be substantially strengthened. In terms of circuits, this result shows that (in relativized worlds) the existence of small circuits for NP does not imply P=NP or even that the boolean hierarchy is finite. This result improves work of Kurtz, Immerman and Mahaney reported in [HSI83b].

Similarly, by displaying a relativized world where PSPACE has a sparse oracle but where the boolean hierarchy is infinite, we show that the Karp-Lipton result on sparse oracles for PSPACE (i.e., that if PSPACE has a sparse oracle then $\Sigma_2^p = \mathrm{PSPACE}$) is also as strong as possible. Intuitively, these results say that a strengthening of the Karp-Lipton results can be achieved only by proof techniques that do not relativize.

Indeed, one can easily extend these results to prove the following stronger and truly surprising theorem: There is a relativized world A in which $\mathrm{P}^{A,S} = \mathrm{NP}^{A,S}$ yet BH^A is infinite.

Our proofs combine the methods of diagonalization and encoding of computations. By diagonalization, we win a block of space to use. We store the location of this space in the sparse oracle. We devote the space to encoding the computations of machines. Thus $\mathrm{NP}^A \subseteq \mathrm{P}^{A,S}$, because the sparse oracle S lets us find NP^A computations encoded in A. We carefully diagonalize to separate the boolean hierarchy. For simplicity, we prove the case of $D^P(A) \neq coD^P(A)$.

Theorem 9.1:

a) There is a recursive set A and a sparse set S so $NP^A \subseteq P^{A,S}$ and $D^P(A) \neq coD^P(A)$.

b) There is a recursive set A and a sparse set S so $PSPACE^A \subseteq P^{A,S}$ and $D^P(A) \neq coD^P(A)$.

Corollary 9.1: There is a relativized world where the boolean hierarchy is infinite and PSPACE has sparse oracles.

Proof of Theorem 9.1:

a) Let $\{N_i\}$ and $\{A_i\}$ be enumerations of standard clocked NP and coNP machines, respectively. W.l.o.g., N_i and A_i run at most $n^{\log^* i} + \log^* i$ steps on input of size n for $i > 20$, and one step for $i \leq 20$. Define an enumeration $\{H_i\}$ of standard clocked coD^P machines so $L(H_i) = L(N_j) \bigcup L(A_k)$, $i = <j,k>$.

Let $U_A = \{N_i \# x \# 1^l | \ N_i^A(x)$ accepts in at most l steps$\}$. U_A encodes the computations of NP^A machines. We'll code U_A into A and S so that $NP^A \subseteq P^{A,S}$.

Let $L_A = \{0^n | (\exists y)[|y| = 3n+1 \wedge y \in A]\} \bigcap \{0^n | \ (\forall y)[|y| = 3n+2 \Rightarrow y \notin A]\}$. Surely $L_A \in D^P(A)$. We diagonalize over the H_i's to show that $L_A \notin coD^P(A)$.

Stage q, encoding section: Choose an *address*, a_q, $|a_q| = q$, so no length $3q$ string starting with a_q has yet been fixed. The H_i's fix few enough strings that such an a_q can always be found. Now use the block of length $3q$ strings starting with a_q as a "treasure box" to encode the information of all length q strings of U_A: $[a_q y 0^q \in A \Leftrightarrow y \in U_A]$, for all $|y| = q$. Crucially, the computations being coded query strings of length at most q, and our length q strings have long since been settled.

Now we must code the precious address string, a_q, into S. For $j \leq q$ put $1^{<q,j>}$ in S if and only if bit j of a_q is 1. Now S codes a_q; with q calls to S we can recover a_q. Note that S will be sparse.

Stage q, diagonalizing section: Suppose H_i is the first coD^P machine that still might accept L_A. We will extend A so that $L(H_i^A) \neq L_A$.

Recall $L(H_i^A) = L(N_j^A) \bigcup L(A_k^A)$, $i = \langle j,k \rangle$. Run $N_j(0^q)$ with all possible extensions of A as its oracle.

Case 1: It accepts for some extension. Freeze all queried elements on some accepting path. Thus $L(H_i^A)$ accepts 0^q. Now, put an unfixed length $3q+2$ string into A, so $0^q \notin L_A$. H_i^A does not accept L_A.

Case 2: It rejects on all extensions. So $L(H_i^A) = L(A_k^A)$, and a humble coNP machine is left with the task of accepting a D^P set! Consider all extensions of A in which $(\forall z)[|z| = 3q+2 \Rightarrow z \notin A]$.

Case 2a: For some such extension, B, $A_i^B(0^q)$ rejects. Freeze in A the elements on a rejecting path and add to A an unfixed length $3q+1$ string. Thus $H_i^A(0^q)$ rejects but $0^q \in L_A$.

Case 2b: For all such extensions, A_i accepts. Then it accepts with A untouched, but $0^q \notin L_A$ so it should reject.

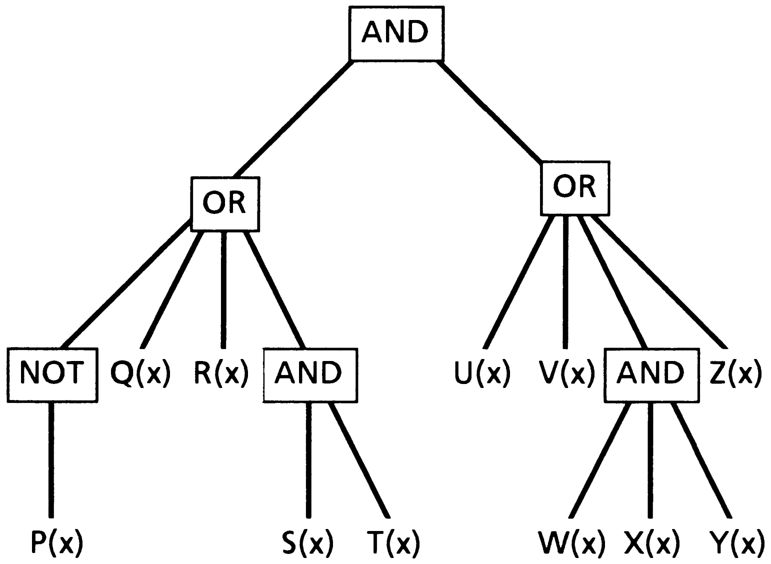
b) See the discussion below. ▸

The same tricks can be used with the "sawing" diagonalization. This combination creates a relativized world in which NP (or, as we note below, PSPACE) has a sparse oracle and the boolean hierarchy is infinite.

Note that once we win ourselves some coding space we can use the space to code the computations of *any* machine class that queries strings of length at most polynomial in its input's size. We just code the universal set of that class into our addressed treasure boxes. Thus the same proof works for PSPACE (part b), with the standard conventions about relativized PSPACE. Indeed, with a relativization we could even make the boolean hierarchy infinite while finding a sparse oracle for the badly non-recursive class RE_{poly}, the class of sets accepted by machines that query strings of length at most polynomial in their input size.

Acknowledgement

We warmly acknowledge Professor Juris Hartmanis's guidance and significant contribution to the content and scope of this work. We also thank Professors R. Book, P. Odifreddi, V. Sewelson, and S. Zachos for stimulating discussions.



Each predicate is an NP predicate.

Figure 1: Hardware over NP

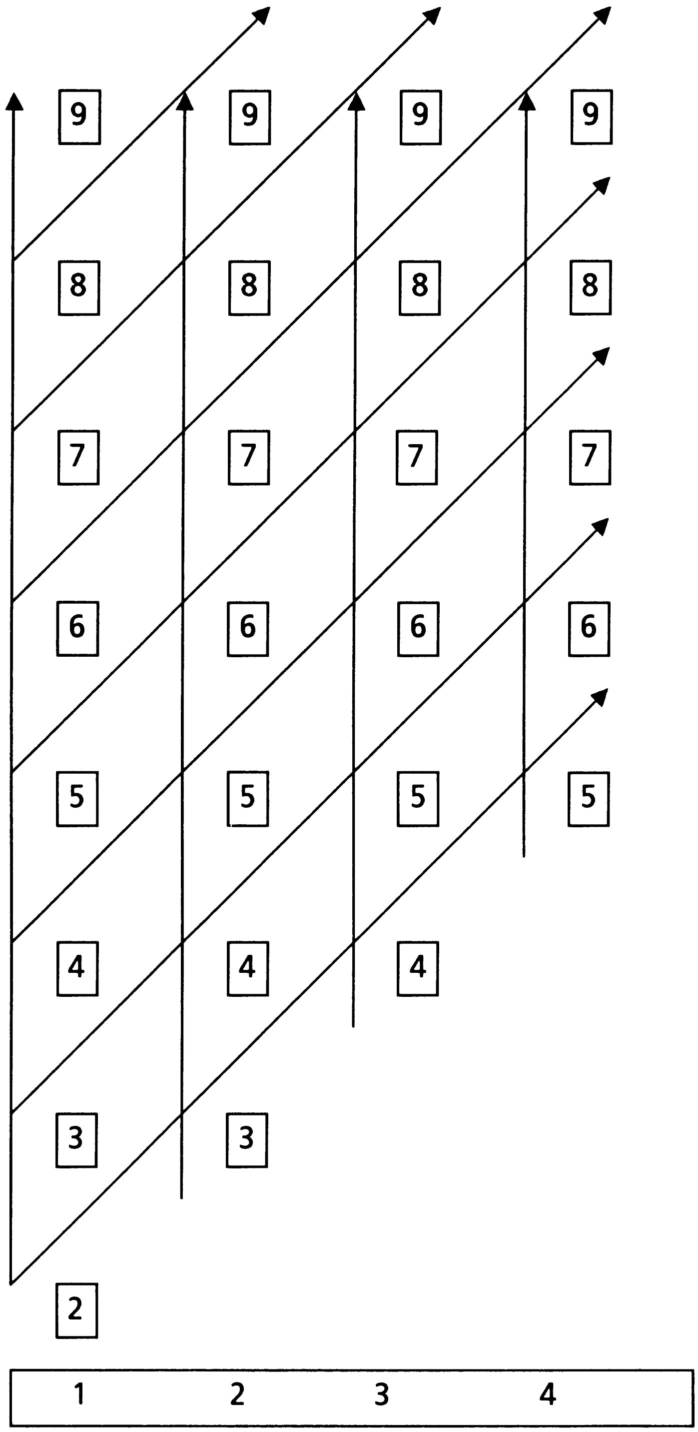


Figure 2: Simple Containments

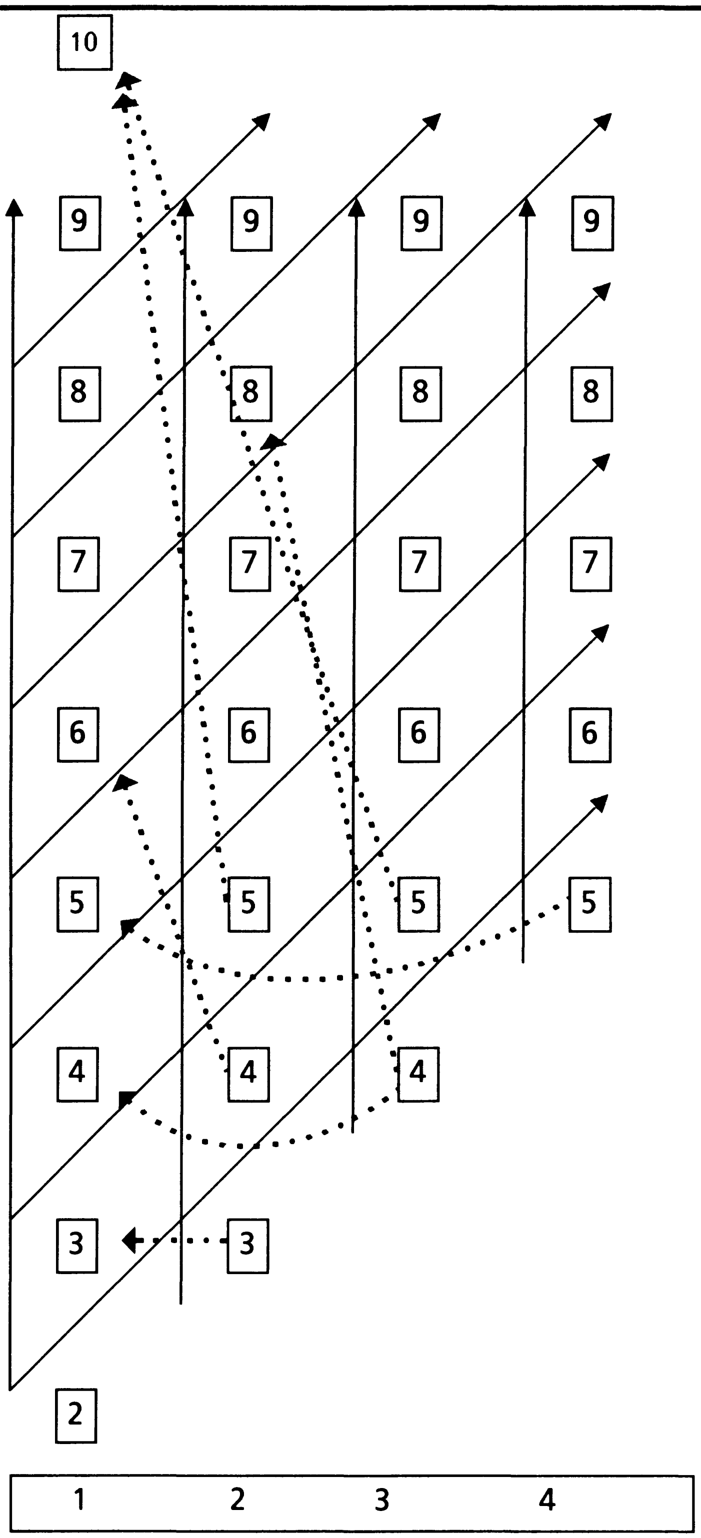


Figure 3: Additional Containments

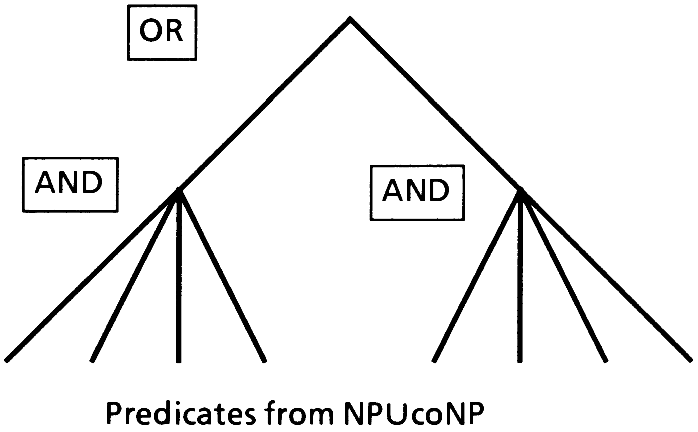


Figure 4: Tree of a Π_2^b set

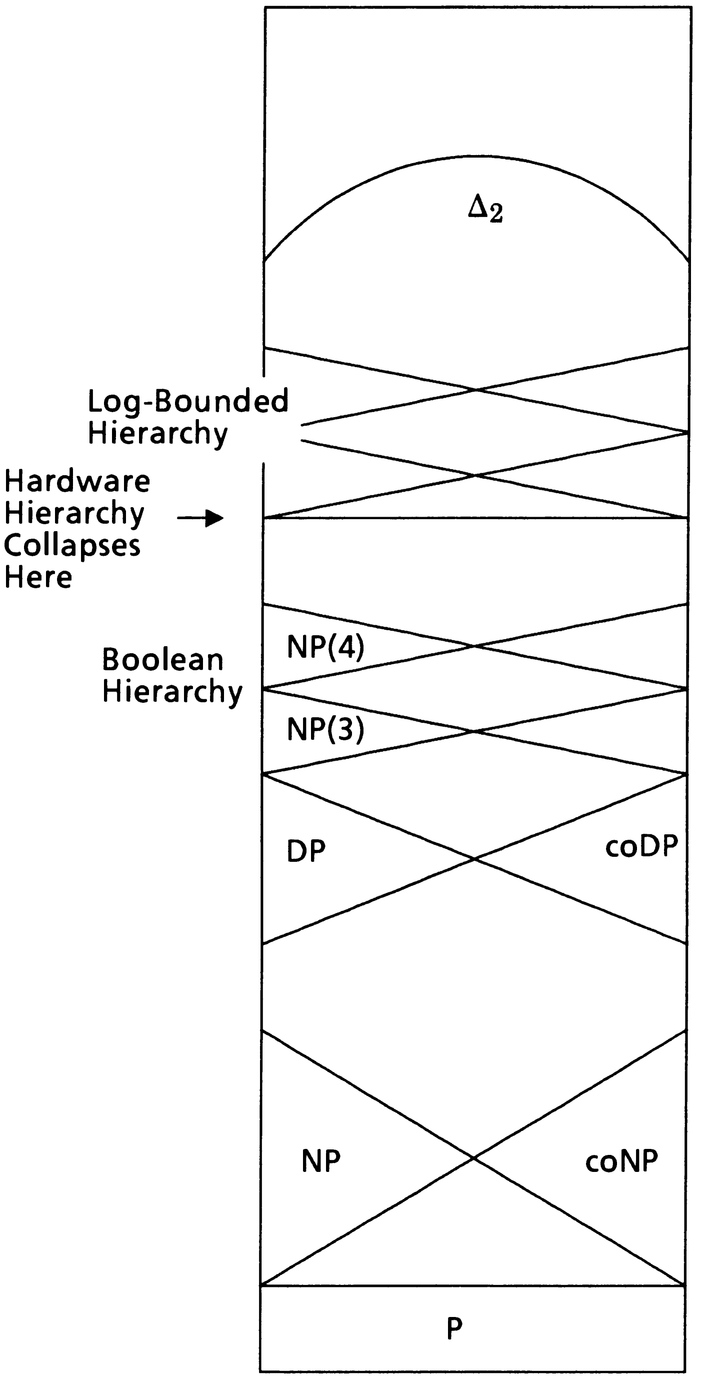


Figure 5: Complexity Hierarchies

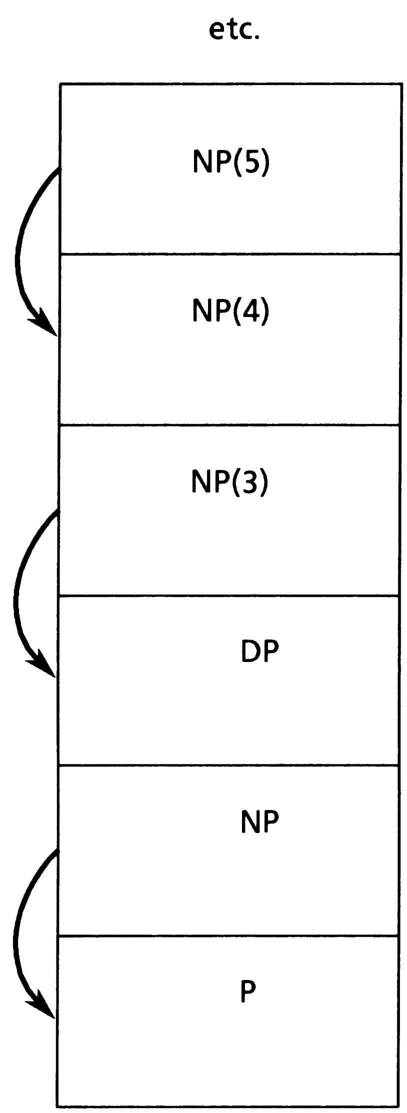


Figure 6a: The arrows indicate the movement of sparse sets when E = NE.

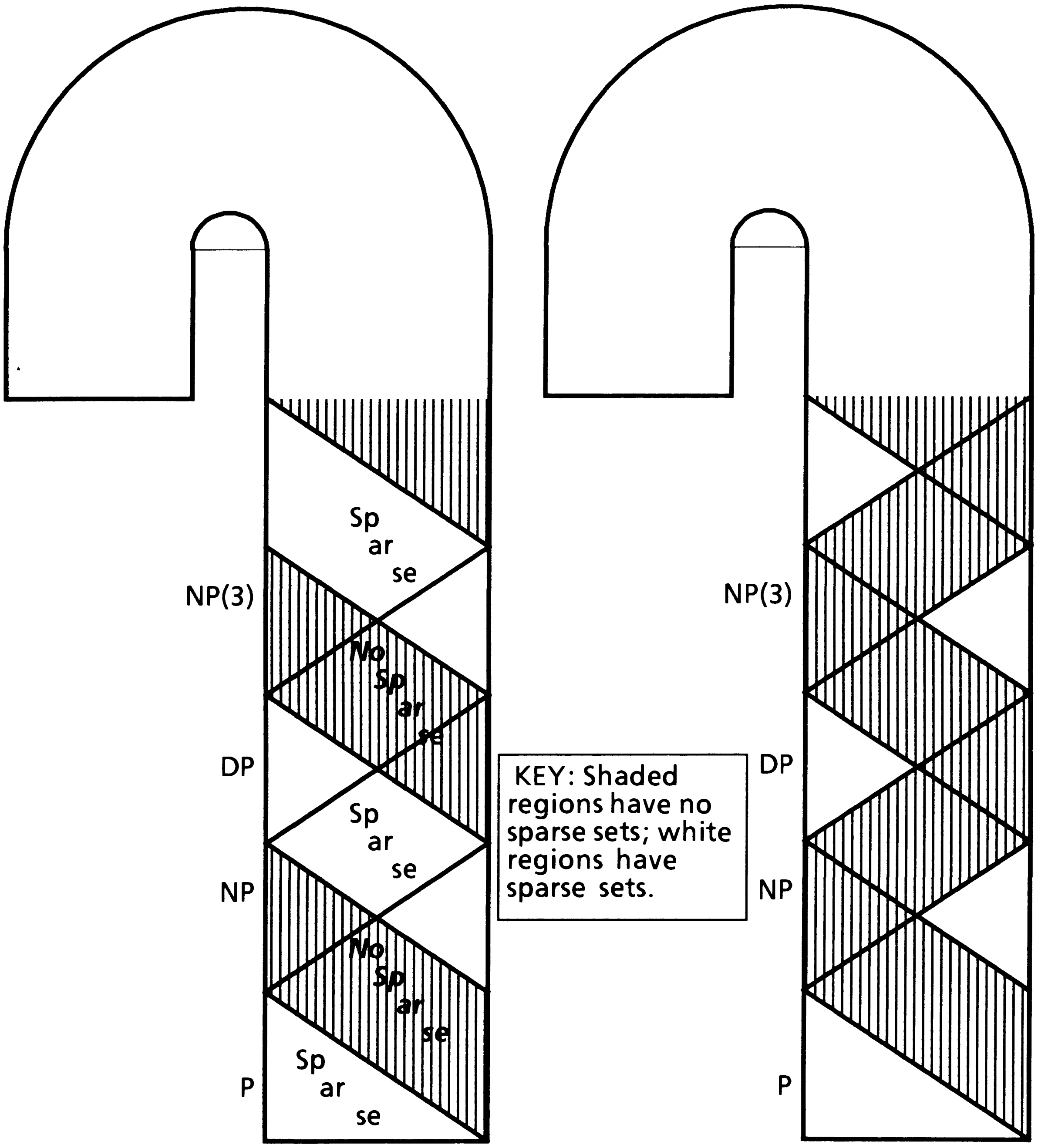


Figure 6b: In our candy cane relativization, regions with and without sparse sets alternate. The left cane displays the world of Theorem 5.1.4. An easy modification of the oracle (noting Theorem 5.1.3b) gives the right cane.

Case	To code y $\in$ U		To code y $\notin$ U
1	X O	5i + 1	X O
	X O	5i + 2	X O ☒
	☒ O	5i + 3	O
2a	☒ O X	5i + 1	X O
	O	5i + 2	☒ O
	EMPTY	5i + 3	EMPTY
2b	☒	5i + 1	EMPTY
	EMPTY	5i + 2	EMPTY
	EMPTY	5i + 3	EMPTY

KEY: X strings may be fixed in A
O strings may be fixed out of A
☒ string we add
EMPTY all strings fixed out of A

Figure 7: Encoding Strategies Under Various Restrictions

Bibliography

[BG81]
C. Bennett and J. Gill, "Relative to a Random Oracle, $P^A \neq NP^A \neq coNP^A$ with Probability 1," *SIAM Journal on Computing*, Vol. 10, #1, Feb. 1981, pp. 96-113.

[BG82]
A. Blass and Y. Gurevich, "On the Unique Satisfiability Problem," *Information and Control*, **55**, 1982, pp. 80-88.

[BS79]
T. Baker and A. Selman, "A Second Step Towards the Polynomial Hierarchy," *Theoretical Computer Science*, **8**, 1979, pp. 177-187.

[CM85]
J. Cai and G.E. Meyer, "Graph Minimal Uncolorability is D^P-Complete," Cornell Computer Science Department Technical Report TR85-688, June 1985.

[C71]
S.A. Cook, "The Complexity of Theorem-Proving Procedures," *Proceedings of the 3rd Annual Symposium on the Theory of Computation,* 1971, pp. 151-158.

[FSS81]
M. Furst, J.B. Saxe and M. Sipser, "Parity, Circuits, and the Polynomial-Time Hierarchy," *Proceedings of the 22rd Annual Symposium on Foundations of Computer Science*, 1981, pp. 260-270.

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

[HH86]
Juris Hartmanis and Lane Hemachandra, "On Sparse Oracles Separating Feasible Complexity Classes," to appear in *Proceedings of the 3rd Annual Symposium on Theoretical Aspects of Computer Science*, Springer-Verlag Lecture Notes in Computer Science, January 1986.

[HHta]
Juris Hartmanis and Lane Hemachandra, "Complexity Classes Without Machines: On Complete Sets for UP," to appear.

[HSI83a]
J. Hartmanis, V. Sewelson, and N. Immerman, "Sparse Sets in NP−P: EXPTIME versus NEXPTIME," Cornell Computer Science Department Technical Report TR83-544, February 1983.

[HSI83b]
J. Hartmanis, V. Sewelson, and N. Immerman, "Sparse Sets in NP−P: EXPTIME versus NEXPTIME," *Proceedings of the 15th Annual Symposium on the Theory of Computation,* 1983, pp. 382-391.

[HY84]
J. Hartmanis and Y. Yesha, "Computation Times of NP Sets of Different Densities," *Theoretical Computer Science*, V. 34, 1984, pp. 17-32.

[H82]
J. Hartmanis, "On the Structure of Feasible Computations," Cornell Department of

Computer Science Technical Report TR82-484, March 1982.

[H84]
H. Heller, "Relativized Polynomial Hierarchies Extending Two Levels," *Mathematical Systems Theory*, V. 17, #2, May 1984, pp. 71-84.

[KL80]
R.M. Karp and R.J. Lipton, "Some Connections Between Nonuniform and Uniform Complexity Classes," *Proceedings of the 12th Annual Symposium on the Theory of Computation,* 1980, pp. 302-309.

[M82]
S. Mahaney, "Sparse Complete Sets for NP: Solution to a Conjecture of Berman and Hartmanis," *Journal of Computer and Systems Sci.*, **25**, 1982, pp. 130-143.

[PW85]
C.H. Papadimitriou and D. Wolfe, "The Complexity of Facets Resolved," *Proceedings of the* 26th *Annual Symposium on Foundations of Computer Science*, 1985.

[R85]
D. A. Russo, "Structural Properties of Complexity Classes," Thesis: University of California at Santa Barbara Department of Mathematics, March 1985.

[SB84]
U. Schoning and R. V. Book, "Immunity, Relativizations, and Nondeterminism," *SIAM Journal on Computing*, Vol. 13, #2, May 1984, pp. 329-337.

[Y83]
Y. Yesha, "On Certain Polynomial-Time Truth-Table Reducibilities of Complete Sets to Sparse Sets," *SIAM Journal on Computing*, Vol. 12, #3, August 1983, pp. 411-425.

[Y85]
A. Yao, "Separating the Polynomial-Time Hierarchy by Oracles," *Proceedings of the* 26th *Annual Symposium on Foundations of Computer Science*, 1985.