Is One NP Question as Powerful as Two?

Jim Kadin*

87-842
June 1987

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

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

Is One NP Question as Powerful as Two?

Jim Kadin*
Cornell University

June 9, 1987

Abstract

For any integer k, $\mathrm{P}^{\mathrm{SAT}[k]}$ is the class of languages accepted by deterministic polynomial time oracle machines that make at most k queries to an oracle for SAT. It is easy to see that $\mathrm{P}^{\mathrm{SAT}[1]} \subseteq \mathrm{D}^{\mathrm{P}} \subseteq \mathrm{P}^{\mathrm{SAT}[2]}$. We use a technique called oracle replacement to show that if $\mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}}$, then there is a sparse set S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, there exist "small" NP machines that recognize initial segments of $\overline{\mathrm{SAT}}$, and the polynomial time hierarchy (PH) is contained in Δ_3^{P}. Thus for deterministic polynomial time oracle machines, if one NP question is as powerful as two, then the PH collapses to Δ_3^{P}. As another application of oracle replacement, we show that if there exists a sparse set $S \in \mathrm{NP}^{\mathrm{SAT}}$ such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, then $\mathrm{PH} \subseteq \Delta_3^{\mathrm{P}}$. We also discuss how oracle replacement is a unifying principle for many of the results concerning sparse oracles for NP.

1 Introduction

It is widely conjectured that SAT, the set of satisfiable Boolean formulas, cannot be recognized in polynomial time. If this conjecture is true, then the class of languages recognized by deterministic polynomial time machines

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

that can access an oracle for SAT contains languages that are not in P. If SAT is really all that difficult for polynomial time machines, one might expect that machines that can make two or more queries to the oracle would be more powerful than those that can make only one query. That is, each new query answer would really provide a deterministic machine with new information. In this paper, we show that if this is not the case, then there is a sparse set S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, and the polynomial time hierarchy (PH) [Sto77] collapses to its third level (Δ_3^{P}). Since it is also widely conjectured that the PH is infinite, one query being as powerful as two implies that the world is very different from what is conjectured.

Krentel has shown similar results for the classes of *functions* computable by polynomial time machines that access an oracle for SAT [Kre86]. He has proven that if certain classes of these functions can be computed with significantly fewer queries, then P = NP, thus totally flattening the PH. Until now, no results were known that related the collapse of query bounded *language* classes in $\mathrm{P}^{\mathrm{SAT}}$ to the collapse of the PH.

This result can also be viewed as a downward structural result. If certain classes above NP and co-NP collapse, then there exists a sparse set S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$. A set S is sparse if there exists a polynomial bound on the number of strings in S of each length. We show that the existence of such an S is equivalent to the existence of "small" NP machines for $\overline{\mathrm{SAT}}$.

$\overline{\mathrm{SAT}}$ has *small* NP *machines* if there exists a sequence of NP machines $\{N_i\}$ such that:

1. $\mathrm{L}(N_i)$ is the set of unsatisfiable formulas of length less than or equal to i,

2. there is a polynomial p_{size} such that for all i, the size of N_i is bounded by $p_{size}(i)$,

3. there is a single polynomial that bounds the running time of all the N_i.

A single NP machine that recognizes all the strings in $\overline{\mathrm{SAT}}$ would be a strong NP representation of $\overline{\mathrm{SAT}}$. The existence of such a machine would imply that co-NP = NP. If $\overline{\mathrm{SAT}}$ has small NP machines, then there is a sequence of NP machines that recognize initial segments of $\overline{\mathrm{SAT}}$. The sequence is uniform in the sense that a single polynomial bounds the running time of all the machines and the difference in size from one machine to the next grows polynomially. The bound on the size of the machines implies that the machines do not simply contain tables of all the satisfiable or unsatisfiable strings of a certain length since there is no polynomial that bounds the number of such strings. Thus each machine in the sequence really executes some NP algorithm to recognize unsatisfiable strings. So small NP machines for $\overline{\mathrm{SAT}}$ would be an NP representation of $\overline{\mathrm{SAT}}$. The existence of such machines would imply that nondeterminism is useful in recognizing $\overline{\mathrm{SAT}}$ and that co-NP is "almost" NP.

Thus we prove that $\overline{\mathrm{SAT}}$, and therefore every language in co-NP, has small NP machines if certain classes above NP collapse. We are not aware of any previous downward structural results of this type.

Our notation for specifying a bound on the number of queries is to write the bound enclosed in brackets after the name of the oracle. For example, $\mathrm{P}^{\mathrm{SAT}[1]}$ is the class of languages recognized by deterministic polynomial time machines that make at most one query to an oracle for SAT.

These implications of $\mathrm{P}^{\mathrm{SAT}[1]} = \mathrm{P}^{\mathrm{SAT}[2]}$ should also be considered in light of the fact that $\mathrm{NP}^{\mathrm{SAT}[1]} = \mathrm{NP}^{\mathrm{SAT}}$. For an NP machine accessing an oracle for SAT, one query is enough since SAT is "accessible" to NP machines. That is, an $\mathrm{NP}^{\mathrm{SAT}}$ machine can figure out a lot about its queries without accessing its oracle. If deterministic machines enjoy this same query collapsing, then SAT is somehow pretty accessible to P machines. The results in this paper show that such accessibility would cause co-NP to have small NP machines and collapse the PH.

Specifically, we show that if $\mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}}$, then there exists a sparse set S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^{S}$, there exist small NP machines for $\overline{\mathrm{SAT}}$, and $\mathrm{PH} \subseteq \Delta_3^{\mathrm{P}}$.

D^{P} is the class of languages that are the intersection of an NP language and a co-NP language. The canonical $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete set for D^{P} [PY82] is

$$\text{SAT-UNSAT} \stackrel{\mathrm{def}}{=} \{x\#y \mid x \in \mathrm{SAT} \text{ and } y \in \overline{\mathrm{SAT}}\}.$$

Clearly each of the strings in SAT-UNSAT is composed of an NP "question" and a co-NP "question".

The class of languages whose complements are in D^{P} is called $\text{co-}\mathrm{D}^{\mathrm{P}}$. Thus each language in $\text{co-}\mathrm{D}^{\mathrm{P}}$ is the union of an NP language and a co-NP language. The canonical $\leq_m^P$-complete set for co-$D^P$is

$$\text{UNSAT-OR-SAT} \stackrel{\text{def}}{=} \{x\#y | x \in \overline{\text{SAT}} \text{ or } y \in \text{SAT}\}$$

which is just the complement of SAT-UNSAT.

Thus if D^P = co-D^P, then SAT-UNSAT $\leq_m^P$ UNSAT-OR-SAT via some polynomial time function g. This g maps any pair of an NP question and a co-NP question to another pair with the property that both questions of the first pair can be answered by answering only one question in the second pair. Thus g acts like a reduction from SAT to $\overline{\text{SAT}}$. The existence of a polynomial time reduction from SAT to $\overline{\text{SAT}}$ would cause PH $\subseteq$ NP. Assuming the existence of g is a similar yet weaker hypothesis. It implies a similar yet weaker result.

Getting back to queries, it is not difficult to prove that

$$\begin{aligned} P^{SAT[1]} &\subseteq D^P \cap \text{co-}D^P, \quad \text{and} \\ D^P \cup \text{co-}D^P &\subseteq P^{SAT[2]}. \end{aligned}$$

Therefore, $P^{SAT[1]} = P^{SAT[2]}$ implies D^P = co-D^P, which forces the existence of the sparse set and the small NP machines for $\overline{\text{SAT}}$ and collapses the PH.

The proof is broken into three parts. In section 3 we use a technique called *oracle replacement* to show that if $\overline{\text{SAT}}$ has small NP machines that can be generated by a $P^{NP^{SAT}}$ machine, then PH $\subseteq P^{NP^{SAT}} = \Delta_3^P$.

In section 4 we show that the existence of small NP machines for $\overline{\text{SAT}}$ is equivalent to the existence of a sparse set S such that $\overline{\text{SAT}} \in NP^S$.

Section 5 completes the proof by showing that if D^P = co-D^P, then the reduction from SAT-UNSAT to UNSAT-OR-SAT can be used to define just such a sparse set – or equivalently – a sequence of small NP machines for $\overline{\text{SAT}}$. A $P^{NP^{SAT}}$ machine will be able to generate and write down the elements of the sparse set or the small machines themselves as needed.

These results are another step in a sequence of results concerning sparse oracles and the PH. We discuss some of the older results in section 6 showing how oracle replacement is a unifying concept. The proof that the existence of a sparse $S \in NP^{SAT}$ such that $\overline{\text{SAT}} \in NP^S$ implies PH $\subseteq P^{NP^{SAT}}$ is also presented in this section.

In section 7 we show that the collapse of D^P and co-D^P or $P^{SAT[1]}$ and $P^{SAT[2]}$ also cause a more local collapse within P^{SAT}. The classes affected are the Boolean hierarchy [CH86,Wag85] and $P^{SAT[\log n]}$.

2 Notation

We use capital letters M and N, possibly subscripted, to represent polynomial time machines. M's represent deterministic machines; N's represent nondeterministic machines.

For any machine M, we write $|M|$ for the size of the state transition table of M (similarly for N's).

For any string x, we write $|x|$ for the length of x.

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

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

3 Oracle Replacement

Oracle replacement is a technique for translating a question of the form "does a particular NP^B machine accept x?" to a question of the form "does a particular NP^A machine accept x?". The basic idea is simple. Suppose you have an NP machine N that accesses an oracle B. If you have two other NP machines N_{yes} and N_{no} such that $\mathrm{L}(N_{yes}^A) = B$ and $\mathrm{L}(N_{no}^A) = \overline{B}$, then you can easily produce a new NP machine N_{new} such that $\mathrm{L}(N_{new}^A) = \mathrm{L}(N^B)$.

On input x, N_{new} simulates $N(x)$. When N would make a query q, N_{new} nondeterministically determines the answer to q by using N_{yes}^A and N_{no}^A. It first simulates $N_{yes}^A(q)$. If the simulation of N_{yes}^A leads to acceptance, $q \in B$, and the query answer is "yes". N_{new} continues its simulation of N from its "yes" state. If $N_{yes}^A(q)$ does not accept, q may be in B, but N_{yes}^A simply guessed incorrectly. So N_{new} simulates $N_{no}^A(q)$. If it accepts, $q \in \overline{B}$, and N_{new} continues simulating N from its "no" state. If neither simulation of N_{yes}^A and N_{no}^A accepts q, N_{new} halts without accepting. Since either $q \in B$ or $q \in \overline{B}$, there is some sequence of guesses that causes either N_{yes}^A or N_{no}^A to accept q. Thus $x \in \mathrm{L}(N^B)$ if and only if there is some sequence of guesses that causes $N_{new}^A(x)$ to accept. Therefore $\mathrm{L}(N_{new}^A) = \mathrm{L}(N^B)$. The construction of N_{new} from N is essentially a replacement of N's query state with a fixed NP subroutine; this can be done in polynomial time in the sizes of N, N_{yes}, and N_{no}.

Thus if there exist N_{yes} and N_{no} such that $B = \mathrm{L}(N_{yes}^A)$ and $\overline{B} = \mathrm{L}(N_{no}^A)$, then we can effectively (in polynomial time) "replace" N's oracle, that is, alter N so that it accepts the same set with a different oracle. Clearly if we can produce N_{new}, then $\mathrm{NP}^B \subseteq \mathrm{NP}^A$, and we can translate the question "does $N^B(x)$ accept?" to "does $N_{new}^A(x)$ accept?".

We can use this same oracle simulation idea to translate NP^B questions to NP^A questions even if $\mathrm{NP}^B \not\subseteq \mathrm{NP}^A$. For a particular x_0, we do not need to find N_{new} such that $\mathrm{L}(N_{new}^A) = \mathrm{L}(N^B)$, we only need to find N_{new} such that $x_0 \in \mathrm{L}(N_{new}^A) \iff x_0 \in \mathrm{L}(N^B)$. N_{new}^A does not need to be able to decide if an arbitrary string is in B to simulate $N^B(x_0)$. It only needs to be able to handle the queries N makes on input x_0. Since N runs in polynomial time, the length of the queries N can make is bounded by a polynomial in the length of x_0. Thus we only need to find NP^A machines N_{yes} and N_{no} that accept the strings in B and $\overline{B}$ up to that length.

We will use this technique to show that under certain circumstances, a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine will be able to translate $\mathrm{NP}^{\mathrm{SAT}}$ questions to NP questions, and this will cause $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$. In this particular case, the B from above is SAT, and the A is the empty set. So we are interested in NP machines that accept initial segments of SAT and $\overline{\mathrm{SAT}}$. Since there exists a fixed NP machine to accept all of SAT, we are really only concerned with NP machines that accept initial segments of $\overline{\mathrm{SAT}}$. Since we are interested in machines that can be generated by a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine, we need a polynomial bound on their size. This is beginning to sound like the definition of small NP machines for $\overline{\mathrm{SAT}}$. In fact, the existence of small NP machines for $\overline{\mathrm{SAT}}$ is exactly what is needed to carry out oracle replacement.

Definition *For any language L, there exist* small NP machines for *L if and only if there exists a sequence of* NP *machines $\{N_{L,1}, N_{L,2}, \cdots\}$ and polynomials p_{size} and p_{run} such that for all i:*

1. $\mathrm{L}(N_{L,i}) = L^{\leq i}$,
2. $|N_{L,i}| \leq p_{size}(i)$,
3. *$N_{L,i}$ runs in time bounded by p_{run}.*

In section 4 we show that $\overline{\mathrm{SAT}}$ has small NP machines if and only if there exists a sparse set S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$. In section 5 we show that if $\mathrm{D^P} = \text{co-}\mathrm{D^P}$, then there exist small NP machines for $\overline{\mathrm{SAT}}$ that can be generated by a $\mathrm{P^{NP^{SAT}}}$ machine. Here, we show how to use the small machines to capture the PH.

First, we use oracle replacement to show that if there exist small NP machines for $\overline{\mathrm{SAT}}$, then there exist small $\mathrm{NP^{SAT}}$ machines for every language in $\mathrm{NP^{NP^{SAT}}}$. If the small machines for $\overline{\mathrm{SAT}}$ can be generated by a $\mathrm{P^{NP^{SAT}}}$ machine, then a $\mathrm{P^{NP^{SAT}}}$ machine can generate the small $\mathrm{NP^{SAT}}$ machines for $\mathrm{NP^{NP^{SAT}}}$ languages.

Lemma 1 *If there exist small* NP *machines, $\{N_{\overline{\mathrm{SAT}},1}, N_{\overline{\mathrm{SAT}},2}, \cdots\}$, for* $\overline{\mathrm{SAT}}$, *and there exists a $\mathrm{P^{NP^{SAT}}}$ machine that on input 1^i can generate $N_{\overline{\mathrm{SAT}},i}$, then there exist small $\mathrm{NP^{SAT}}$ machines and a $\mathrm{P^{NP^{SAT}}}$ machine generator for every language in $\mathrm{NP^{NP^{SAT}}}$. That is, $\forall L \in \mathrm{NP^{NP^{SAT}}}$, $\exists$ machines $\{N_{L,i}\}$, polynomials q_{size}, q_{run}, and a $\mathrm{P^{NP^{SAT}}}$ machine M_{enum}, such that:*

1. $\mathrm{L}(N_{L,i}^{\mathrm{SAT}}) = L^{\leq i}$,
2. $|N_{L,i}| \leq q_{size}(i)$,
3. *$N_{L,i}^{\mathrm{SAT}}$ runs in time bounded by q_{run},*
4. *on input 1^i, $M_{enum}^{\mathrm{NP^{SAT}}}$ outputs $N_{L,i}$.*

Proof: Let p_{run} and p_{size} be the polynomial bounds on the running time of $N_{\overline{\mathrm{SAT}},i}$ and $|N_{\overline{\mathrm{SAT}},i}|$ respectively.

Let $L = \mathrm{L}(N^{\mathrm{NP^{SAT}}})$ be any language in $\mathrm{NP^{NP^{SAT}}}$.

Let t be a polynomial that bounds the running time of N.

We will show how to define $\{N_{L,i}\}$, q_{run}, q_{size}, and M_{enum}. On input 1^n, $M_{enum}^{\mathrm{NP^{SAT}}}$ will output $N_{L,n}$, and $\mathrm{L}(N_{L,n}^{\mathrm{SAT}})$ will be $\mathrm{L}(N^{\mathrm{NP^{SAT}}})^{\leq n}$.

We will actually describe an enumerator for a sequence of machines $\{N_{L,=n}\}$. When given an input of length n, $N_{L,=n}^{\mathrm{SAT}}$ accepts if and only if the string is in L. We will show that there exist polynomials q'_{size} and q'_{run} such that for all n, $|N_{L,=n}| \leq q'_{size}(n)$, and the running time of $N_{L,=n}^{\mathrm{SAT}}$ on an input of length n is bounded by $q'_{run}(n)$. To construct $N_{L,n}$ from these machines, enumerate $N_{L,=k}$ for each k from 1 to n. Glue the $N_{L,=k}$ machines together with a preprocessor containing n states that count the length of the input and call the $N_{L,=k}$ machine for that length. If the input is longer than n, the $N_{L,n}$ machine just halts without accepting. Then

$$|N_{L,n}| \leq n\, q'_{size}(n) + n.$$

The running time of $N_{L,n}$ on any input x is no more than $|x| + q'_{run}(|x|)$ for all n. Thus we will have q_{run} and q_{size} as needed.

The rest of the proof is concerned with showing how a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine, $M_{enum'}$, can generate $N_{L,=n}$ on input 1^n.

$N_{L,=n}^{\mathrm{SAT}}$ is to accept exactly the strings of length n that $N^{\mathrm{NP}^{\mathrm{SAT}}}$ accept, and for strings not of length n, we do not care what $N_{L,=n}$ does since we will never run it on such strings.

On an input x of length n, N runs along nondeterministically and occasionally makes queries to an oracle for $\mathrm{NP}^{\mathrm{SAT}}$. Without loss of generality, we can assume the queries are to the set

$$L_{\mathrm{NP}^{\mathrm{SAT}}} \stackrel{\mathrm{def}}{=} \{\langle N_i, x_i, pad_i\rangle \mid N_i \text{ is an NP machine, and } N_i^{\mathrm{SAT}}(x_i) \text{ accepts in } |x_i| + |pad_i| \text{ steps }\}$$

which is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{NP}^{\mathrm{SAT}}$.

The length of each query $\langle N_i, x_i, pad_i\rangle$ is bounded by $t(|x|)$. The machine N_i in the query can only run for $|x_i| + |pad_i|$ steps, so the length of the queries it makes are also bounded by $t(|x|)$.

Let $m \stackrel{\mathrm{def}}{=} t(n)$.

By hypothesis, a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can generate $N_{\overline{\mathrm{SAT}},m}$ on input 1^m. So our $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine, $M_{enum'}$, can generate $N_{\overline{\mathrm{SAT}},m}$ in polynomial time on input 1^n. With $N_{\overline{\mathrm{SAT}},m}$ and N_{SAT}, the fixed machine that accepts SAT, $M_{enum'}$ can generate an NP subroutine for determining the answers to queries of length $\leq m$ to an oracle for SAT.

The idea is that $N_{L,=n}$ is N altered so that it does oracle replacement on each of its queries. $M_{enum'}$ changes N to $N_{L,=n}$ by changing the way N asks its queries. $N_{L,=n}$ runs just like N, but when N would query about whether $x_i \in \mathrm{L}(N_i^{\mathrm{SAT}})$, $N_{L,=n}$ asks a different query. $N_{L,=n}$ changes N_i to a new machine, $N_{new,i}$, by replacing N_i's query state with the NP subroutine for deciding SAT oracle answers, and asks whether $x_i \in \mathrm{L}(N_{new,i})$. Since N_{SAT} and $N_{\overline{\mathrm{SAT}},m}$ correctly answer the queries N_i makes on input x_i,

$$x_i \in \mathrm{L}(N_{new,i}) \iff x_i \in \mathrm{L}(N_i^{\mathrm{SAT}}).$$

Thus

$$x \in \mathrm{L}(N_{L,=n}^{\mathrm{NP}}) \iff x \in \mathrm{L}(N^{\mathrm{NP}^{\mathrm{SAT}}})$$

where $N_{L,=n}$ makes queries to the NP-complete set

$$L_{\mathrm{NP}} \stackrel{\mathrm{def}}{=} \{\langle N_j, x_j, pad_j\rangle \mid N_j \text{ is an NP machine, and } N_j(x_j) \text{ accepts in } |x_j| + |pad_j| \text{ steps }\}.$$

In other words, $N_{L,=n}$ is N with its query state replaced by a routine that maps queries for $L_{\mathrm{NP}^{\mathrm{SAT}}}$ to queries for L_{NP} before entering the query state. This mapping involves changing the N_i in the original query to a new machine $N_{new,i}$. $N_{new,i}$ is just N_i with its query state replaced by the NP subroutine for deciding answers to SAT queries.

With the basic form of the proof in place, we need to fill in the details.

One detail is that when $N_{L,=n}$ changes a query about N_i to a query about $N_{new,i}$, it must add padding to ensure that $N_{new,i}(x_i)$ can be run in linear time in the total length of the query. $N_{new,i}$ runs slower than N_i since $N_{new,i}$ runs the NP subroutine to get each query answer while N_i gets each answer in one step. Thus the mapping of queries from $L_{\mathrm{NP}^{\mathrm{SAT}}}$ to L_{NP} must also include adding new padding, $pad_{new,i}$, such that $N_{new,i}(x_i)$ runs in $|x_i| + |pad_{new,i}|$ steps.

We have to argue that $M_{enum'}^{\mathrm{NP}^{\mathrm{SAT}}}$ can produce $N_{L,=n}$ in polynomial time on input 1^n and that q'_{run} and q'_{size} exist. Once $M_{enum'}$ has generated $N_{\overline{\mathrm{SAT}},m}$, $N_{L,=n}$ is produced by adding the query altering routine to the fixed code for N. Thus the polynomial running time of $M_{enum'}$ will follow from the existence of q'_{size}.

How big is $N_{L,=n}$? It consists of the code for N and the code for mapping queries from $L_{\mathrm{NP}^{\mathrm{SAT}}}$ to L_{NP}. The size of the code for N is a constant. The query altering routine adds the "N_{SAT}, $N_{\overline{\mathrm{SAT}},m}$" routine to a machine N_i, replacing its query state, and repads. The code for adding the routine to N_i consists of a constant amount of code for moving things around (maybe for copying all of N_i except its query state to a work tape, etc.) and the code for copying in "N_{SAT}, $N_{\overline{\mathrm{SAT}},m}$". Thus the size of the code is

$$c + |N_{\mathrm{SAT}}| + |N_{\overline{\mathrm{SAT}},m}|$$

where c is a constant. $|N_{\mathrm{SAT}}|$ is also a constant, and

$$|N_{\overline{\mathrm{SAT}},m}| \leq p_{size}(m) = p_{size}(t(n)).$$

Thus the size of this part of the code for $N_{L,=n}$ is bounded by a polynomial in n.

The code to do the repadding can simply consist of one state per pad character, and this will be polynomial in n if the amount of padding is. How much padding do we need? We need enough so that the length of the padding bounds the number of steps $N_{new,i}$ can execute on input x_i. $N_{new,i}$ runs just like N_i except for each query q that N_i makes, $N_{new,i}$ runs the "N_{SAT}, $N_{\overline{\mathrm{SAT}},m}$" subroutine on q. This subroutine may run both $N_{\mathrm{SAT}}(q)$ and $N_{\overline{\mathrm{SAT}},m}(q)$. The polynomial p_{run} bounds the running time of $N_{\overline{\mathrm{SAT}},m}$, and we can assume without loss of generality that it also bounds the running time of N_{SAT}. Thus the subroutine takes at most $2p_{run}(|q|)$ steps. Since $N_i^{\mathrm{SAT}}(x_i)$ runs in at most $|x_i| + |pad_i|$ steps and $|x_i| + |pad_i| \leq m$,

$$|q| \leq m, \quad \text{and}$$

$$\text{runtime } N_{new,i}(x_i) \leq 2m\, p_{run}(m).$$

This is polynomial in n since $m = t(n)$.

Thus $q'_{size}(n)$ is roughly

$$|N| + |N_{\mathrm{SAT}}| + |N_{\overline{\mathrm{SAT}},m}| + (\text{the size of the padding}).$$

The running time of $M_{enum'}$ on input 1^n is roughly the amount of time it takes to generate $N_{\overline{\mathrm{SAT}},m}$ plus $q'_{size}(n)$.

The only detail left is to show that there exists a polynomial q'_{run} such that

$$\forall n, \forall x, |x| = n \implies \text{runtime } N^{\mathrm{NP}}_{L,=n}(x) \leq q'_{run}(n).$$

$N^{\mathrm{NP}}_{L,=n}(x)$ runs just like $N^{\mathrm{NP}^{\mathrm{SAT}}}(x)$ except that $N_{L,=n}$ alters queries before entering its query state. Altering the queries involves writing down the "N_{SAT}, $N_{\overline{\mathrm{SAT}},m}$" subroutine and repadding. We have already argued that all of this can be done in time bounded by a polynomial in n. Thus q'_{run} exists.

Finally, note that since SAT is NP-complete and $L_{\mathrm{NP}} \in \mathrm{NP}$, we could include a reduction from L_{NP} to SAT as the last step in $N_{L,=n}$'s query altering routine. Thus we have shown that if $\overline{\mathrm{SAT}}$ has small NP machines, then every language in $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ has small $\mathrm{NP}^{\mathrm{SAT}}$ machines. $\square$

The key to this lemma is that given the small NP machines for $\overline{\mathrm{SAT}}$, oracle replacement is such an effective process that a deterministic machine can reprogram an $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine to reprogram its queries. Thus $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$

can recognize any $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ language by generating a small $\mathrm{NP}^{\mathrm{SAT}}$ machine that accepts a large enough initial segment of the language and asking its $\mathrm{NP}^{\mathrm{SAT}}$ oracle if the generated machine accepts a particular input.

Theorem 2 *If* $\overline{\mathrm{SAT}}$ *has small* NP *machines,* $\{N_{\overline{\mathrm{SAT}},1}, N_{\overline{\mathrm{SAT}},2}, \cdots\}$*, and there exists a* $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ *machine that on input* 1^i *can generate* $N_{\overline{\mathrm{SAT}},i}$*, then* $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$.

Proof: For any $L \in \mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$, there exists a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine that on input x can generate an $\mathrm{NP}^{\mathrm{SAT}}$ machine $N_{L,n}$ (where $n = |x|$) such that

$$x \in \mathrm{L}(N_{L,n}^{\mathrm{SAT}}) \iff x \in L.$$

Since there also exists a polynomial q_{run} that bounds the running time of $N_{L,n}$ for all n, the $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can phrase a question to an oracle for $L_{\mathrm{NP}^{\mathrm{SAT}}}$ to see if $x \in \mathrm{L}(N_{L,n}^{\mathrm{SAT}})$.

Therefore $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$.

Therefore $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$. □

4 Sparse Oracles and Small Machines

A set of strings S is sparse if there exists a polynomial bound on the number strings in S of each length. There is a well known result of Meyer published in [BH77] that relates sparse oracles for NP and polynomial size circuits for SAT.

NP has a sparse oracle if there exists a sparse set S such that $\mathrm{SAT} \in \mathrm{P}^S$.

A sequence of Boolean circuits, $\{C_i\}$, constructed of AND, OR, and NOT gates, are circuits for SAT if for any string x of length n, running C_n on x outputs a 1 if and only if $x \in \mathrm{SAT}$. SAT has polynomial size circuits if there exists a polynomial p such that $\forall i$, $|C_i| \leq p(i)$.

The main result of this section is a similar result relating sparse oracles that allow an NP machine to accept $\overline{\mathrm{SAT}}$ and small (polynomial size) NP machines for $\overline{\mathrm{SAT}}$.

Theorem 3 $\exists$ *sparse set* S *such that* $\overline{\mathrm{SAT}} \in \mathrm{NP}^S \iff \exists$ *small* NP *machines for* $\overline{\mathrm{SAT}}$.

Proof:
($\Longrightarrow$) Let N be the NP machine such that $\overline{\text{SAT}} = \text{L}(N^S)$.
Let p_1 bound $\| S^{\leq n} \|$.
Let p_2 bound the running time of N.
Define $N_{\overline{\text{SAT}},i}$ to be N with $S^{\leq p_2(i)}$ encoded as a table in $N_{\overline{\text{SAT}},i}$'s states and a routine to look up strings in the table in place of N's query state. The table must be sorted by increasing length so that for an input x of length $n \leq i$, only the portion of the table containing strings of length less than or equal to $p_2(n)$ has to be searched to find a query answer. This portion of the table contains at most $p_1(p_2(n))$ strings, each of at most $p_2(n)$ characters long. So each query takes $N_{\overline{\text{SAT}},i}$ roughly $p_2(n)p_1(p_2(n))$ steps to answer. Therefore

$$\begin{aligned} p_{run}(n) &= p_2(n)\,(p_2(n)p_1(p_2(n))), \quad \text{and} \\ p_{size}(i) &= |\, N \,| + p_2(i)p_1(p_2(i)) + c \end{aligned}$$

which is the size of N plus the size of the table and lookup routine.
($\Longleftarrow$) Let $\{N_{\overline{\text{SAT}},i}\}$ be the small machines for $\overline{\text{SAT}}$ with polynomials p_{run} and p_{size} bounding the running time and size respectively. We can assume p_{size} is monotonically increasing. Add dummy states to each $N_{\overline{\text{SAT}},i}$ so that it is exactly $p_{size}(i)$ characters long. Let the sparse oracle S consist of the machines $N_{\overline{\text{SAT}},i}$ themselves. S is sparse since it has at most one string of each length.

The NP^S machine N for recognizing $\overline{\text{SAT}}$ works as follows. On input x with $|\, x \,| = n$, N guesses an NP machine of length $p_{size}(n)$ and verifies with one query to S that it has guessed $N_{\overline{\text{SAT}},n}$. It then simulates $N_{\overline{\text{SAT}},n}$ on x. $\Box$

This result also holds for deterministic machines. That is, there exists a sparse oracle for $\overline{\text{SAT}}$ (and SAT) if and only if there exist small P machines for $\overline{\text{SAT}}$ (and SAT).

Theorem 4 $\exists$ *sparse* S *such that* $\overline{\text{SAT}} \in \text{P}^S \iff \exists$ *small* P *machines for* $\overline{\text{SAT}}$.

Proof: The proof is parallel to the proof of the previous theorem except in the ($\Longleftarrow$) direction a P machine cannot guess the small P machine for

$\overline{\mathrm{SAT}}$. The prefixes of the small P machines must be encoded into the sparse oracle so a deterministic machine can get hold of the small machines. □

Combining this theorem with the result of Meyer mentioned earlier, we know that the following are equivalent:

1. $\exists$ sparse S such that SAT $\in \mathrm{P}^S$.

2. $\exists$ small P machines for SAT.

3. $\exists$ polynomial size circuits for SAT.

A special case of results by Yap in [Yap83] is that if there exists a sparse S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, then $\mathrm{PH} \subseteq \Sigma_3^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$. Thus we now know that if there exist small NP machines for $\overline{\mathrm{SAT}}$, $\mathrm{PH} \subseteq \Sigma_3^{\mathrm{P}}$.

Corollary 5 *If $\exists$ small* NP *machines for* $\overline{\mathrm{SAT}}$, *then* $\mathrm{PH} \subseteq \Sigma_3^{\mathrm{P}}$.

The techniques used in section 3 can be used to prove this directly. An $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can guess an NP machine $N_{\overline{\mathrm{SAT}},n}$ and verify with its oracle that it has a machine that accepts $\overline{\mathrm{SAT}}^{\leq n}$. Once it has the machine, it can use oracle replacement to produce an $\mathrm{NP}^{\mathrm{SAT}}$ machine for an initial segment of any $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ language as we did in Lemma 1, and thus determine whether or not an $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine accepts its input. Thus $\text{co-NP}^{\mathrm{NP}^{\mathrm{SAT}}} \subseteq \mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ implying $\mathrm{PH} \subseteq \mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$.

In the next section, we show that if $\mathrm{D^P} = \text{co-D}^{\mathrm{P}}$, then a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can get hold of the small machines and thus do the same trick.

5 Collapsing the Polynomial Time Hierarchy

Now we are ready to complete the proof that if $\mathrm{D^P} = \text{co-D}^{\mathrm{P}}$, then $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$.

It should be clear from the proof of Theorem 3 that if there exists a sparse set S with $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, then generating small machines for $\overline{\mathrm{SAT}}$ is no harder than generating the elements of S (up to a certain length). Thus Theorem 2 can be restated as:

> if there exists a sparse set S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$ and there exists a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine that outputs $S^{\leq n}$ on input 1^n, then $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$.

This is the formulation we will use to complete our proof.

We will show that if $\mathrm{D^P} = \text{co-}\mathrm{D^P}$, then there exists such a sparse set that can be enumerated by a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ algorithm.

We could of course complete the proof by directly producing the small NP machines for $\overline{\mathrm{SAT}}$. This involves essentially the same argument as producing the sparse set, but it also involves a bit more technical difficulty to show the existence of p_{size} and p_{run}.

Theorem 6 *If* $\mathrm{D^P} = \text{co-}\mathrm{D^P}$*, then there exists a sparse set* S *such that* $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$*, and there exists a deterministic polynomial time machine* M *such that on input* 1^n*,* $M^{\mathrm{NP}^{\mathrm{SAT}}}$ *outputs the strings in* $S^{\leq n}$*.*

Proof: The main idea is that if $\mathrm{D^P} = \text{co-}\mathrm{D^P}$, then

$$\text{SAT-UNSAT} \leq^{\mathrm{P}}_{\mathrm{m}} \text{UNSAT-OR-SAT}$$

via some polynomial time function g. This g can be used to define the sparse set.

The function g maps any pair of formulas $F_1\#F_2$ to a pair $G_1\#G_2$ such that

$$F_1 \in \mathrm{SAT} \text{ and } F_2 \in \overline{\mathrm{SAT}} \iff G_1 \in \overline{\mathrm{SAT}} \text{ or } G_2 \in \mathrm{SAT}.$$

Suppose $G_2 \in \mathrm{SAT}$. Then $F_2 \in \overline{\mathrm{SAT}}$ and $F_1 \in \mathrm{SAT}$. That is, G_2 is an "NP witness" that $F_2 \in \overline{\mathrm{SAT}}$. There is an NP algorithm for recognizing unsatisfiable strings with such NP witnesses. Given a formula F of length n, guess a string F' of length n and compute $g(F'\#F) = G'\#G$. Guess an assignment and accept F if the assignment satisfies G ($|\,G\,|$ is polynomial in n since g is a polynomial time function).

Call the NP machine that executes this algorithm N_{easy} since it accepts the strings that are easily seen to be in $\overline{\mathrm{SAT}}$. Call the strings that N_{easy} accepts easy strings

Definition *A formula* F *is* easy *if* $\exists F'$ *with* $|\,F'\,| = |\,F\,|$ *such that* $g(F'\#F) = G'\#G$ *where* $G \in \mathrm{SAT}$*. (if* F *is easy, then* $F \in \overline{\mathrm{SAT}}$*.)*

If every unsatisfiable string is easy, then NP = co-NP, and we can let our sparse set by the empty set. So most likely, there are unsatisfiable strings that are not easy. That is, there are hard strings.

Definition *A formula F is* hard *if $F \in \overline{\mathrm{SAT}}$ and $\forall F'$ with $|F'| = |F|$, $g(F'\#F) = G'\#G$ where $G \in \overline{\mathrm{SAT}}$.*

While it does not seem easy to find an NP algorithm for accepting hard strings, each hard string has a very peculiar property. The property is that given a hard string $\hat{F}$ with $|\hat{F}| = n$, then $\hat{F}$ and g can be used to define an NP algorithm for accepting $\overline{\mathrm{SAT}}^{=n}$.

The algorithm takes an input F with $|F| = n$. First, compute $g(F\#\hat{F}) = G\#\hat{G}$. Since $\hat{F}$ is hard, $\hat{G} \notin$ SAT. Since g is a reduction,

$$F\#\hat{F} \in \text{SAT-UNSAT} \Longleftrightarrow G\#\hat{G} \in \text{UNSAT-OR-SAT}.$$

Then

$$F\#\hat{F} \in \text{SAT-UNSAT} \Longleftrightarrow G \in \overline{\mathrm{SAT}}.$$

Since $\hat{F} \in \overline{\mathrm{SAT}}$,

$$F \in \mathrm{SAT} \Longleftrightarrow G \in \overline{\mathrm{SAT}}$$

which means

$$F \in \overline{\mathrm{SAT}} \Longleftrightarrow G \in \mathrm{SAT}.$$

So accept F if $G \in$ SAT. Call the NP machine that executes this algorithm $N_{hard(\hat{F})}$.

Thus each hard string is a key string that can be used to define an NP algorithm for accepting all the unsatisfiable formulas of that length.

We are now ready to construct our sparse set S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$. There will be one string of each length in S. The string of length n will be a hard string or 0^n if all the strings of length n are easy (we assume that for each n, 0^n does not represent a valid formula).

If $0^n \in S$, then N_{easy} works to recognize $\overline{\mathrm{SAT}}^{=n}$. If $0^n \notin S$, then some hard string, $\hat{F}$, of length n is in S. An NP^S machine can guess $\hat{F}$, verify that $\hat{F} \in S$, and then use $N_{hard(\hat{F})}$ to recognize $\overline{\mathrm{SAT}}^{=n}$. So an NP^S machine on an input of length n can use its oracle to figure out which NP algorithm accepts $\overline{\mathrm{SAT}}^{=n}$, and then it can run that algorithm and thus recognize $\overline{\mathrm{SAT}}$.

Thus by putting an arbitrary hard string into S for each length (whenever one exists), we can define a sparse S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$. If we are a bit more careful about which hard string we choose, a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can generate the elements of S. Define S as above, but put the lexicographically first hard string of each length into S. This will be the hard string that a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can get hold of.

The set of hard strings is in co-NP since it is the complement of the union of SAT and the set of easy strings. This implies that the set of prefixes of hard strings,

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

is in $\mathrm{NP}^{\mathrm{SAT}}$.

Let N_{pre} be the $\mathrm{NP}^{\mathrm{SAT}}$ machine that accepts this prefix set. A $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine on input 1^n can use N_{pre}^{SAT} to

1. determine whether or not there are any hard strings of length n (if not, then $0^n \in S$).
2. if there are hard strings, determine the lexicographically least one of length n (which would be the string in $S^{=n}$).

It can determine (1) by asking whether $N_{pre}^{SAT}(0\#^{n-1})$ or $N_{pre}^{SAT}(1\#^{n-1})$ accepts. If neither accepts, then there are no hard strings of length n. If either accepts, there are hard strings. Given that one exists, the $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can generate longer and longer prefixes of the lexicographically first hard string one bit at a time by asking how N_{pre}^{SAT} reacts to the prefixes. After n queries, it will have the hard string of length n in S.

Therefore there exists a sparse set S that can be generated by a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$. $\square$

So the existence of a reduction from SAT-UNSAT to UNSAT-OR-SAT implies the existence of hard unsatisfiable strings. These hard strings can be used as keys to produce small NP algorithms for initial segments of $\overline{\mathrm{SAT}}$. Since the set of hard strings is in co-NP, a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can generate the lexicographically least hard string of each length and thus generate small NP machines for $\overline{\mathrm{SAT}}$. These small machines give a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine the power to take any $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine and reprogram it to reprogram its queries so that it becomes an $\mathrm{NP}^{\mathrm{SAT}}$ machine that accepts an initial segment of the language of the original machine. A $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can use its oracle to determine if the reprogramed machine accepts an input, and thus recognize any $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ language. Since $\mathrm{P}^{\mathrm{SAT}[1]} = \mathrm{P}^{\mathrm{SAT}[2]}$ implies SAT-UNSAT $\leq_{\mathrm{m}}^{\mathrm{P}}$ UNSAT-OR-SAT, we have

Theorem 7 $\mathrm{P}^{\mathrm{SAT}[1]} = \mathrm{P}^{\mathrm{SAT}[2]} \Longrightarrow \mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}} \Longrightarrow$

1. *$\exists$ sparse S that can be generated by a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine such that* $\overline{\mathrm{SAT}} \in \mathrm{NP}^{S}$.
2. $\overline{\mathrm{SAT}}$ *has small* NP *machines that can be generated by a* $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ *machine.*
3. $\mathrm{PH} \subseteq \Delta_3^{\mathrm{P}}$.

Perhaps the most amusing fact in this proof is that a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can capture the PH by generating an NP machine for recognizing a finite set, and NP machines are considerably less powerful than $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machines. This fact is a good indication that small NP machines for a language is a powerful representation of the language.

There are a number of interesting corollaries to this result. We state a few of them.

Corollary 8 *If* $\mathrm{P}^{\mathrm{SAT}} = \mathrm{P}^{\mathrm{SAT}[1]}$, *then* $\mathrm{PH} \subseteq \Delta_3^{\mathrm{P}}$.

Corollary 9 *If the Boolean hierarchy collapses to* D^{P}, *then* $\mathrm{PH} \subseteq \Delta_3^{\mathrm{P}}$.

Corollary 10 *If the* PH *collapses to* D^{P}, *then $\exists$ small* NP *machines for* $\overline{\mathrm{SAT}}$.

6 Oracle Replacement Revisited

The results of the paper are another step in a sequence of results relating sparse oracles to the collapse of the PH. The idea of oracle replacement is not new. It is implicitly embedded in the proofs of many of these older results. It is a unifying concept, and thus it is insightful to review these results and see how the technique and the small machines were used in each case.

In 1980 Karp and Lipton, with help from Sipser, showed that if SAT has polynomial size circuits (or equivalently, if NP has a sparse oracle), then $\mathrm{PH} \subseteq \mathrm{NP}^{\mathrm{SAT}}$ [KL80]. The circuits are essentially small polynomial time machines for SAT, and since they are deterministic, also for $\overline{\mathrm{SAT}}$. An $\mathrm{NP}^{\mathrm{SAT}}$ machine can guess a circuit and verify that it works. With the circuit in hand, the $\mathrm{NP}^{\mathrm{SAT}}$ machine can replace the oracle of another $\mathrm{NP}^{\mathrm{SAT}}$ machine and convert it to an NP machine. It can ask its oracle whether or *not* the NP machine accepts an input, and thus accept a co-$\mathrm{NP}^{\mathrm{SAT}}$ language. Thus $\mathrm{NP}^{\mathrm{SAT}} = \text{co-}\mathrm{NP}^{\mathrm{SAT}}$, and the PH collapses. Their proof was not exactly formulated this way, but it was not far off.

The method of verifying that the guessed circuit works correctly is particularly interesting. It depends upon the self reducibility property or SAT. Informally, self reducibility means that any deterministic polynomial time algorithm (in this case, circuit), for SAT can effectively be turned into a deterministic polynomial time machine that reads a formula and outputs an assignment that satisfies the formula if it is in SAT.

After guessing a circuit to work for $\mathrm{SAT}^{\leq n}$, an $\mathrm{NP}^{\mathrm{SAT}}$ machine can turn the circuit into a machine M that produces assignments based on the circuit's answers. The answer to the NP question,

> "$\exists F, A$, such that $|F| = n$, A satisfies F, and yet the output of $M(F)$ does not satisfy F?"

determines whether or not the circuit works. An affirmative answer means it does not work; a negative answer means it does.

In terms of a sparse set S such that $\mathrm{SAT} \in \mathrm{P}^S$, the proof is identical. Guess the strings in S up to a certain length, and using the self reducibility of SAT, verify that the guessed strings "work". The strings provide a small P machine for SAT, and thus $\overline{\mathrm{SAT}}$, which allows an $\mathrm{NP}^{\mathrm{SAT}}$ machine to capture any language in PH.

Yap's result for the nondeterministic case can be proven similarly. Recall that he showed that if $\exists$ a sparse S such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, then $\mathrm{PH} \subseteq \mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ [Yap83]. Self reducibility is no help in this case since it only works for deterministic algorithms. Thus the power of SAT queries does not seem to be enough to verify that the guessed strings work. $\mathrm{NP}^{\mathrm{SAT}}$ queries are powerful enough though. Let N be the machine such that $\overline{\mathrm{SAT}} = \mathrm{L}(N^S)$. An $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can guess the polynomially many strings in S that N could ask on an input of length n. Call this set of strings S'. Then the answer to the $\mathrm{NP}^{\mathrm{SAT}}$ query,

> "$\exists F, \forall A, w$, A does not satisfy F, and $N(F)$ does not accept with guess sequence w and oracle S'?"

verifies if the guessed strings work. So an $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can verify that it has the small machines for $\overline{\mathrm{SAT}}$, and as we have seen, this means an $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can do oracle replacement.

Another result in this area was proved by Mahaney. He showed that if $\exists$ sparse S such that $\mathrm{NP} \subseteq \mathrm{P}^S$ where $S \in \mathrm{NP}$, then $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{SAT}}$ [Mah82]. The idea here is that since $S \in \mathrm{NP}$, a $\mathrm{P}^{\mathrm{SAT}}$ machine can generate the actual members of S one bit at a time by asking about an NP machine that accepts prefixes of lists of strings in S. Again, once a $\mathrm{P}^{\mathrm{SAT}}$ machine can generate the initial segments of S, it has small machines for SAT and $\overline{\mathrm{SAT}}$, and so on.

Long generalized Mahaney's result by showing that if the sparse set S is anywhere in $\mathrm{P}^{\mathrm{SAT}}$, then the self reducibility of SAT and the $\mathrm{P}^{\mathrm{SAT}}$ recognizer for S can be used to enumerate all the strings in S that are needed by the P^S recognizer for SAT [Lon82]. Thus this too implies $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{SAT}}$.

A nondeterministic companion to Long's result follows easily from oracle replacement.

Theorem 11 *If $\exists$ sparse $S \in \mathrm{NP}^{\mathrm{SAT}}$ such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, then* $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$.

Proof: If $S \in \mathrm{NP}^{\mathrm{SAT}}$, then $\exists$ an $\mathrm{NP}^{\mathrm{SAT}}$ machine that accepts prefixes of lexicographically ordered lists of strings in S up to a given length. A $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can then ask queries to build the list of strings in S up to a given length one character at a time. Thus a $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$ machine can generate small NP machines for $\overline{\mathrm{SAT}}$. $\square$

In each of the oracle results mentioned so far, the idea is to enumerate enough of the sparse set to produce small P or NP machines for an initial segment of $\overline{\mathrm{SAT}}$. More recently, Mahaney's result was improved by showing that if $\exists$ sparse $S \in \mathrm{NP}$ such that $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$ (or $\mathrm{SAT} \in \mathrm{P}^S$), then

$\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{SAT}[\log n]}$, the class of languages recognizable in polynomial time with only $\mathrm{O}(\log n)$ queries [Kad87]. A $\mathrm{P}^{\mathrm{SAT}[\log n]}$ machine can enumerate small machines for $\overline{\mathrm{SAT}}$ not by enumerating the elements of S, but instead by computing the census function of S (the number of strings in S up to a given length). With this result came the realization that the small machines are more important than the elements of the sparse oracle. The reason the elements of the oracle are useful is that they can be used to construct the small machines.

The collapse to $\mathrm{P}^{\mathrm{SAT}[\log n]}$ has been shown to be optimal in the sense that there exist relativized worlds where NP has sparse $\leq_{\mathrm{T}}^{\mathrm{P}}$-complete sets, and yet the PH collapses no further than $\mathrm{P}^{\mathrm{NP}[\log n]}$ [Kad87]. The proof is interesting in that it shows the collapse goes no further because no machine that makes fewer than $\mathrm{O}(\log n)$ queries can compute the census function. Thus intuitively, the census function is the least amount of information needed to construct the small machines.

In summary, any class, $\mathrm{P}^{\mathrm{SAT}[\log n]}$, $\mathrm{P}^{\mathrm{SAT}}$, $\mathrm{NP}^{\mathrm{SAT}}$, $\mathrm{P}^{\mathrm{NP}^{\mathrm{SAT}}}$, $\mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$, that can generate the small machines captures the PH by oracle replacement. The different hypotheses about sparse oracles make the small machines easier or harder to generate. Under no assumptions except that S is a sparse oracle in the nondeterministic case, i.e. $\overline{\mathrm{SAT}} \in \mathrm{NP}^S$, $\mathrm{PH} \subseteq \mathrm{NP}^{\mathrm{NP}^{\mathrm{SAT}}}$. In the deterministic case, $\overline{\mathrm{SAT}} \in \mathrm{P}^S$, $\mathrm{PH} \subseteq \mathrm{NP}^{\mathrm{SAT}}$. The difference between these cases seems to be that the self reducibility of SAT can be used in the deterministic case but not in the nondeterministic case.

If we further assume that $S \in \mathrm{NP}$, then the fact that a $\mathrm{P}^{\mathrm{SAT}[\log n]}$ machine can compute the census function of S is more useful than the self reducibility of SAT. Thus in both the deterministic and nondeterministic cases, the PH collapses to $\mathrm{P}^{\mathrm{SAT}[\log n]}$.

Long's theorem and Theorem 11 represent the implications for two intermediate assumptions. Again self reducibility is helpful in the deterministic case.

7 Collapsing Below

The collapse of $\mathrm{D^P}$ and co-$\mathrm{D^P}$ or $\mathrm{P}^{\mathrm{SAT}[1]}$ and $\mathrm{P}^{\mathrm{SAT}[2]}$ implies a more local collapse in addition to the collapse of the PH. The local collapse involves the Boolean hierarchy [CH86,Wag85], query bounded classes within $\mathrm{P}^{\mathrm{SAT}}$, and $\mathrm{P}^{\mathrm{SAT}[\log n]}$.

Definition *For any positive integer k, $\mathrm{P}^{\mathrm{SAT}[k]}$ is the class of languages accepted by deterministic polynomial time machines that make at most k queries to an oracle for* SAT *on all inputs.*

It is clear that

$$\forall k, \mathrm{P}^{\mathrm{SAT}[k]} \subseteq \mathrm{P}^{\mathrm{SAT}[k+1]} \subseteq \bigcup_i \mathrm{P}^{\mathrm{SAT}[i]} \subseteq \mathrm{P}^{\mathrm{SAT}[\log n]}.$$

We call the hierarchy of $\mathrm{P}^{\mathrm{SAT}[k]}, \mathrm{P}^{\mathrm{SAT}[k+1]}, \cdots$ the *constant query bounded hierarchy.*

A deterministic polynomial time machine M that makes $\mathrm{O}(\log n)$ queries (or a constant number) on all inputs has the property that there exists a polynomial time function f which on input x produces a *query tree* representing the possible query behavior of $M(x)$. The internal nodes of the query tree are query strings that M may ask on input x. The leaves are labeled "accept" or "reject". The root of the tree is the first query M asks. The left child of a node represents the behavior of M on x if the query answer is no. The right child represents the behavior if the answer is yes. The function, f, builds the tree on input x by running M on x trying both possible answers to every query M makes. Thus the height of the tree is bounded by the number of queries M can make.

A path from the root to a leaf is *accepting* if it ends at a leaf labeled "accept". The path from the root to a leaf with all the queries answered correctly is the *valid path.* The valid path of the tree for $M(x)$ is accepting if and only if $x \in \mathrm{L}(M^{\mathrm{SAT}})$.

If $\mathrm{P}^{\mathrm{SAT}[1]} = \mathrm{P}^{\mathrm{SAT}[2]}$, then $\mathrm{P}^{\mathrm{SAT}[\log n]}$ and the constant query bounded hierarchy both collapse.

Theorem 12 $\mathrm{P}^{\mathrm{SAT}[1]} = \mathrm{P}^{\mathrm{SAT}[2]} \Longrightarrow \mathrm{P}^{\mathrm{SAT}[\log n]} \subseteq \mathrm{P}^{\mathrm{SAT}[1]}$.

Proof: If $\mathrm{P}^{\mathrm{SAT}[1]} = \mathrm{P}^{\mathrm{SAT}[2]}$, then there is a polynomial time mapping from the set of query trees of height two to the set of trees of height one. This mapping preserves the acceptance behavior of the valid path. Starting with the $O(\log n)$ high query tree for any $P^{SAT[\log n]}$ machine on input x, the mapping can be repeatedly applied to the subtrees of height two until the tree is of height one. One query is then enough to determine if the valid path is accepting. Therefore $P^{SAT[\log n]} \subseteq P^{SAT[1]}$. □

This also collapses the Boolean hierarchy since it is intertwined with the constant query bounded hierarchy.

Similarly, $D^P = \text{co-}D^P \Longrightarrow P^{SAT[\log n]} \subseteq D^P$. The proof requires knowledge of the Boolean hierarchy which we will not go into here. The basic idea is that $D^P = \text{co-}D^P$ collapses the Boolean hierarchy to D^P which implies $P^{SAT[3]} \subseteq P^{SAT[2]}$. Thus there exists a polynomial time function mapping query trees of height three to trees of height two. As above, query trees of height $O(\log n)$ can then be shrunk to height two.

This local collapse is based entirely on query tree shrinking which is different from the oracle replacement used to collapse the PH. The obvious question is could $P^{SAT[\log n]}$ collapse to D^P and the PH collapse to $P^{NP^{SAT}}$ while everything between $P^{SAT[\log n]}$ and $P^{NP^{SAT}}$ remains intact? Is it that the PH could simply fall this way, or is it that we do not yet have the techniques to show that

$$D^P = \text{co-}D^P \Longrightarrow PH \subseteq D^P?$$

8 Conclusion

For deterministic polynomial time machines, is one NP query as powerful as two? We now know that if it is, many complexity classes including co-NP are drastically affected.

Like most new results, the results presented here raise more questions than they answer. Does the collapse of other classes in P^{SAT} cause the PH to collapse? Are there small NP machines for $\overline{SAT}$, and if so, how hard are they to enumerate? Does $D^P = \text{co-}D^P$ cause the PH to collapse further than $P^{NP^{SAT}}$? What about the "window" between $P^{SAT[\log n]}$ and $P^{NP^{SAT}}$?

The downward structural implication of small machines for $\overline{SAT}$ if $D^P = \text{co-}D^P$ is significant since it relates the structure of the PH to the structure of co-NP in the opposite direction from what we are familiar with. Can we improve this result and conclude the existence of the small machines if classes above D^P and co-D^Pcollapse?[1]

Acknowledgements

I am grateful to Professor Hartmanis for his support and guidance and for reading a draft of this paper. I am also grateful to Steve Mitchell and Terry Ticknor for their comments on this paper.

References

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

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

[Kad87] J. Kadin. $P^{NP[\log n]}$ and sparse Turing complete sets for NP. In *Proc. 2nd Structure in Complexity Theory Conf.*, Ithaca, New York, June 1987. To appear.

[KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *ACM Symposium on Theory of Computing*, pages 302–309, 1980.

[Kre86] M. Krentel. The complexity of optimization problems. In *ACM Symposium on Theory of Computing*, pages 69–76, May 1986.

[Lon82] T. J. Long. A note on sparse oracles for NP. *Journal of Computer and System Sciences*, 24:224–232, 1982.

[Mah82] S. Mahaney. Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis. *Journal of Computer and System Sciences*, 25(2):130–143, 1982.

[1] Recently, the techniques developed in this paper have been generalized to show that the sparse set exists and the PH collapses if any level of the Boolean hierarchy collapses. These results will be presented in a forthcoming technical report.

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

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

[Wag85] K. Wagner. On the Boolean closure of NP. In *Proc. of the 1985 International Conference on Fundamentals of Computation Theory*, pages 485–493, Springer-Verlag *Lecture Notes in Computer Science* , 1985.

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