Abstract. In [3], Kadin showed that if the Polynomial Hierarchy has infinitely many levels, then for all \( k \), \( P^{\text{SAT}(k)} \subseteq \text{P}^{\text{SAT}(k+1)} \). In this paper, we extend Kadin's technique to show that a proper query hierarchy is not an exclusive property of NP complete sets. In fact, for any \( A \in \text{NP} - \text{low}_3 \), if \( PH \) is infinite, then \( P^{[k]} \subseteq P^{[k+1]} \). Moreover, for the case of parallel queries, we show that \( P^{[k+1]} \) is not contained in \( \text{P}^{\text{SAT}(k)} \). We believe that having a proper query hierarchy is a consequence of the oracle access mechanism rather than a result of the set being very "hard". To support this claim, we show that assuming \( PH \) is infinite, one can construct an intermediate set \( B \in \text{NP} \) so that \( \text{P}^{[k+1]} \nsubseteq \text{P}^{\text{SAT}(k)} \). That is, the query hierarchy for \( B \) grows as "tall" as the query hierarchy for \( SAT \). In addition, \( B \) is intermediate, so it is not \( NP \) "hard" in any sense (e.g., not \( NP \) hard under many-one, Turing, or strong nondeterministic reductions).

Using these same techniques, we explore some other questions about query hierarchies. For example, we show that if there exists any \( A \) such that \( P^{[k]} = \text{P}^{\text{SAT}(k)} \), then \( PH \) collapses to \( \Delta_2 \).

1 Introduction

In [3], Kadin made the Boolean and Query Hierarchies respectable by showing that if the Polynomial Hierarchy (PH) has infinitely many levels, then the Boolean Hierarchy (BH) and the Query Hierarchy (QH) are proper hierarchies. BH and QH are well studied complexity classes in \( \Delta_2 \) [2,3,10]. These hierarchies are essentially built from \( SAT \) and \( \text{SAT} \) with logical "and" and "or". QH is made by allowing any polynomial time computable combination of \( SAT \) and \( \text{SAT} \). Kadin uncovered some intricate structural properties of these hierarchies and showed that the collapse of BH implies the collapse of the PH.

In this paper, we look at hierarchies built with arbitrary sets in NP. We want to know when these hierarchies are proper, and how they relate to BH and QH. We will study these questions in the setting of Schöning's high-low sets and produce some results about these hierarchies similar to ones already known about BH and QH. In particular, we conclude that if a set is high and the query hierarchy built from this set collapses, then \( PH \) collapses. We also show that if \( PH \) is infinite, then bounded query hierarchies built from many languages in NP are also proper. (By "many", we mean all sets in \( \text{NP} - \text{low}_3 \).) We interpret this as evidence that a proper query hierarchy is not an exclusive property of hard sets in NP. To demonstrate this claim, we show that if \( PH \) is infinite, then we can construct a language \( B \) in \( NP \) so that for all \( k \), \( P^{[k+1]} \nsubseteq \text{P}^{\text{SAT}(k)} \). Moreover, this language \( B \) is not a high set, so it is not "hard" in any sense (e.g., not \( NP \) hard under many-one, Turing, or strong nondeterministic reductions).

We assume that the reader is familiar with \( P, NP, PH, PSAT \), sparse sets and oracle Turing Machines. We quickly define some familiar notation and concepts. For any language \( A \), we write \( A^{[m]} \) for the set of strings in \( A \) of length \( m \). Recall a set \( S \) is sparse if \( |S|^m \) is bounded by a polynomial. A padded prefix of a string \( x \) is a string \( w \) such that \( |w| = |x|, w = y \#^{k} \) and \( y \) is a prefix of \( x \). A set \( S \) is self-p-printable if there is a polynomial time oracle Turing Machine, \( D_{\#}^{S} \), which prints \( S^{[m]} \) on input \( 1^m \). All self-p-printable sets are sparse. A set \( A \) is closed under padded prefixes if \( x \in A \) and \( w \) is a padded prefix of \( x \) \( \Rightarrow \) \( w \in A \).

If a set is sparse and closed under padded prefixes, it is self-p-printable.

2 The Boolean Hierarchy and the Query Hierarchy

In this section, we summarize previous work in the Boolean and Query Hierarchies. This is neither a complete nor a chronological summary. For completeness, see [1,2,3,10].

We begin with the Parallel Query Hierarchy (\( QH_{\#} \)). This hierarchy is made by restricting a \( \text{P}^{\text{SAT}} \) machine's access to the \( SAT \) oracle in two ways. First, the number of queries is limited to a constant (i.e., does not depend on input length). Second, all the queries must be made at the same time. We allow the \( P \) base machine to do some computation to determine what the queries are, but we do not allow the query strings to depend on the results (oracle...
answers) of previous queries. In other words, the queries are made in parallel and the oracle answers with a bit vector. Parallel queries are sometimes called non-adaptive. We write \( \text{PSAT}^{[k]} \) for the class of languages accepted by a polynomial time Turing Machine that makes at most \( k \) parallel queries to \( \text{SAT} \) on input of any length. Of course,

\[
\text{PSAT}^{[1]} \subseteq \text{PSAT}^{[2]} \subseteq \ldots \subseteq \text{PSAT}^{[k]} \subseteq \text{PSAT}^{[k+1]} \subseteq \ldots
\]

This nested sequence has the upward collapsing property, in the sense that if \( \text{PSAT}^{[k]} = \text{PSAT}^{[k+1]} \), then the entire Parallel Query Hierarchy \( (\text{QH}^k = \bigcup_{j=1}^{\infty} \text{PSAT}^{[j]}) \) falls down to \( \text{PSAT}^{[k]} \). If \( \text{PH} \) is infinite, then \( \text{QH}^k \) is a proper hierarchy [3].

\[
\text{PSAT}^{[1]} \subseteq \text{PSAT}^{[2]} \subseteq \ldots \subseteq \text{PSAT}^{[k]} \subseteq \text{PSAT}^{[k+1]} \subseteq \ldots
\]

If we remove the parallel query restriction from \( \text{PSAT}^{[k]} \) and allow subsequent queries to depend on answers to previous queries, we obtain the serial Query Hierarchy. We write \( \text{PSAT}^{[2k]} \) for the class of languages accepted by a polynomial time Turing Machine which asks at most \( k \) serial queries to \( \text{SAT} \) for input strings of any length, and \( \text{QH} \) for \( \bigcup_{j=1}^{\infty} \text{PSAT}^{[j]} \). Beigel [1] showed by a clever binary search routine (dubbed the “mind change” trick) that

\[
\text{PSAT}^{[2k-1]} = \text{PSAT}^{[2k]}.
\]

So, we can think of the levels of \( \text{QH} \) simply as levels of \( \text{QH}^k \) exponentially far apart. In complexity theory, serial queries are generally considered more natural and relevant than parallel queries. However, \( \text{QH}^k \) has a finer structure than \( \text{QH} \) which makes it more amenable to analysis.

Now, we examine the Boolean Hierarchy [2]. Like \( \text{PH} \), each level of \( \text{BH} \) is composed of two complementary language classes \( \text{BH}(k) \) and \( \text{co-BH}(k) \), such that

\[
\text{co-BH}(k) = \{ L \mid \overline{L} \in \text{BH}(k) \}.
\]

\( \text{BH}(k) \) is defined inductively, starting with \( \text{NP} \) and building up with unions and intersections, as follows:

\[
\begin{align*}
\text{BH}(1) &= \text{NP} \\
\text{BH}(2) &= \{ L_1 \cap L_2 \mid L_1 \in \text{NP} \text{ and } L_2 \in \text{NP} \} \\
\text{BH}(2k) &= \{ L_1 \cap L_2 \mid L_1 \in \text{BH}(2k-1) \text{ and } L_2 \in \text{NP} \} \\
\text{BH}(2k+1) &= \{ L_1 \cup L_2 \mid L_1 \in \text{BH}(2k) \text{ and } L_2 \in \text{NP} \}.
\end{align*}
\]

The levels of \( \text{BH} \) are interlaced, much like \( \text{PH} \):

\[
\text{BH}(k) \subseteq \text{BH}(k+1) \cap \text{co-BH}(k+1) \subseteq \text{BH}(k+1)
\]

\[
\text{co-BH}(k) \subseteq \text{BH}(k+1) \cap \text{co-BH}(k+1) \subseteq \text{co-BH}(k+1)
\]

\( \text{BH} \) also has the upward collapsing property. If \( \text{BH}(k) = \text{co-BH}(k) \), then \( \text{BH} = \text{BH}(k) \cap \text{co-BH}(k) \). Many results in this area depend on the fact that \( \text{BH} \) and \( \text{QH}^k \) are intertwined [1,2]

\[
\begin{align*}
\text{BH}(k) \cup \text{co-BH}(k) &\subseteq \text{PSAT}^{[k]} \\
\subseteq \text{BH}(k+1) \cap \text{co-BH}(k+1) &\subseteq \text{PSAT}^{[k+1]}.
\end{align*}
\]

Clearly \( \text{BH} \) is a proper hierarchy iff \( \text{QH}^k \) is a proper hierarchy. Kadin showed that if \( \text{BH} \) collapses, then \( \text{PH} \) collapses. His proof depends on the structure of the canonical complete languages for the levels of \( \text{BH} \). For \( \text{BH}(2) \) and \( \text{BH}(3) \) these complete languages are:

\[
\text{BH} \cap \text{SAT} = \{ (F_1, F_2) \mid F_1 \in \text{SAT} \text{ and } F_2 \in \text{SAT} \}
\]

\[
\text{BH} \cap \text{SAT} \cap \text{SAT} = \{ (F_1, F_2, F_3) \mid (F_1 \in \text{SAT} \text{ and } F_2 \in \text{SAT} \text{ or } F_3 \in \text{SAT} \}. \]

### 3 High and Low Sets

In [7], Schöning defined the high and low hierarchies for \( \text{NP} \) to classify sets between \( \text{P} \) and \( \text{NP} \). Very roughly, the high-low classification measures the amount of information an \( \text{NP} \) language (acting as an oracle) can give to a base machine in \( \text{PH} \). If \( A \in \text{NP} \), then for all \( k \)

\[
\Sigma^P_k \subseteq \Sigma^P_{k+1} \subseteq \text{PSAT}^{[k+1]}.
\]

If \( \Sigma^P_k = \text{PSAT}^{[k]} \), then one might say that \( A \) tells the \( \Sigma^P_k \) base machine a lot—as much as \( \text{SAT} \) does. If \( \Sigma^P_k = \text{PSAT}^{[k+1]} \), then one could say that \( A \) doesn’t tell the \( \Sigma^P_k \) base machine anything that it couldn’t compute itself. In the previous case we call \( A \) a high set, in the latter a low set. More formally, we define:

\[
\text{high}_k = \{ A \mid A \in \text{NP} \text{ and } \Sigma^P_k = \text{PSAT}^{[k]} \}
\]

\[
\text{low}_k = \{ A \mid A \in \text{NP} \text{ and } \Sigma^P_k = \Sigma^P_{k+1} \}.
\]

Clearly,

\[
\text{low}_0 \subseteq \text{low}_1 \subseteq \text{low}_2 \subseteq \ldots \\
\text{high}_0 \subseteq \text{high}_1 \subseteq \text{high}_2 \subseteq \ldots
\]

Although, high and low sets are called hierarchies, they are not known to have the familiar upward collapsing behavior. For example, it is not known whether \( \text{low}_2 = \text{low}_3 \) implies \( \text{low}_2 = \text{low}_4 \). Nevertheless, the low hierarchy is quite interesting, because it uniformly classifies many well studied classes.

Schöning also defined a refinement of the low hierarchy [8]

\[
\text{low}_k = \{ A \mid A \in \text{NP} \text{ and } \Delta^P_k = \Delta^P_{k+1} \}.
\]

These classes lie between the levels of the low hierarchy,

\[
\text{low}_0 \subseteq \text{low}_1 \subseteq \text{low}_2 \subseteq \text{low}_2 \subseteq \text{low}_3 \subseteq \text{low}_3
\]

251
and allow a finer classification of languages classes in the low hierarchy. High and low sets have many interesting properties. We will briefly mention some here. See \cite{4,6,7,8,9} for proofs.

1. \( \text{low}_k = \text{high}_k \iff \text{PH} \subseteq \Sigma_k^p \).
2. If \( \text{PH} \) is infinite, then \( \exists I \in \text{NP} \forall k, I \notin \text{high}_k \) and \( I \notin \text{low}_k \).
3. If \( S \) is sparse and \( S \in \text{NP} \), then \( S \in \text{low}_2 \).
4. If \( A \in \text{NP} \) and for some sparse set \( S, A \in \text{P}^S \), then \( A \in \text{low}_3 \).
5. \( \text{R} \subseteq \text{BPP} \subseteq \text{low}_2 \).
6. \( \text{high}_0 = \{ A \mid A \) is \( \leq_{1^k} \)-complete for \text{NP} \} \).
7. \( \text{low}_0 = P \).
8. \( \text{low}_1 = \text{NP} \cap \text{co-NP} \).
9. \( \text{Graph Isomorphism} \in \text{low}_2 \).

4 Main Theorem

The three hierarchies \( \text{QH} \), \( \text{QH}^{[1]} \), and \( \text{BH} \) are essentially built from \( \text{SAT} \) and \( \text{SAT}^{[1]} \). We now consider analogous query hierarchies built from an arbitrary set in \( \text{NP} \). For any set \( A \in \text{NP} \) we can consider the classes \( \text{P}[A]^{[1]} \), \( \text{P}[A]^{[2]} \), and \( \text{P}[A]^{[3]} \). We immediately have

\[
\text{P}[A]^{[3]} \subseteq \text{P}[A]^{[2]} \subseteq \text{P}[A]^{[1]}, \quad \text{and} \quad \text{P}[A]^{[3]} \subseteq \text{P}[A]^{[2]} \subseteq \text{P}[A],
\]

but in general we do not know how to repeat Beigel’s “mind change” trick. So we can relate parallel and serial queries only loosely

\[
\text{P}[A]^{[3]} \subseteq \text{P}[A]^{[2]} \subseteq \text{P}[A]^{[1]}.
\]

Query hierarchies built from an arbitrary set have the upward collapsing property, too. That is

\[
\text{P}[A]^{[1]} = \text{P}[A] \quad \Rightarrow \quad \forall j > k, \quad \text{P}[A]^{[j]} = \text{P}[A]^{[k]} \quad \text{and} \quad \text{P}[A]^{[k]} = \text{P}[A]^{[k]}. 
\]

We will not attempt to define a boolean hierarchy based on \( A \in \text{NP} \), instead we define “boolean languages” analogous to the canonical complete languages for \( \text{BB}(k) \) and \( \text{co-BH}(k) \).

**Definition** For each set \( A \in \text{NP} \) we define a sequence of languages

\[
\begin{align*}
\text{BL}_A(1) &= A \\
\text{BL}_A(2k) &= \{ (x_1, \ldots, x_{2k-1}, x_{2k}) \mid (x_1, \ldots, x_{2k-1}) \in \text{BL}_A(2k-1) \text{ and } x_{2k} \in \overline{A} \} \\
\text{BL}_A(2k + 1) &= \{ (x_1, \ldots, x_{2k+1}) \mid (x_1, \ldots, x_{2k}) \in \text{BL}_A(2k) \text{ or } x_{2k+1} \in A \} \\
\text{co-BL}_A(1) &= \overline{A} \\
\text{co-BL}_A(2k) &= \{ (x_1, \ldots, x_{2k-1}, x_{2k}) \mid (x_1, \ldots, x_{2k-1}) \in \text{co-BL}_A(2k-1) \text{ or } x_{2k} \in A \} \\
\text{co-BL}_A(2k + 1) &= \{ (x_1, \ldots, x_{2k+1}) \mid (x_1, \ldots, x_{2k}) \in \text{co-BL}_A(2k) \text{ and } x_{2k+1} \in \overline{A} \}
\end{align*}
\]

Note that

\[
\begin{align*}
\text{BL}_{\text{SAT}}(2) &= \text{SAT} \land \text{SAT} \\
\text{BL}_{\text{SAT}}(3) &= (\text{SAT} \land \overline{\text{SAT}}) \lor \text{SAT}.
\end{align*}
\]

In the general case, \( \text{BL}_{\text{SAT}}(k) \) is \( \leq^p_k \)-complete for \( \text{BH}(k) \) and \( \text{co-BL}_{\text{SAT}}(k) \) is \( \leq^p_k \)-complete for \( \text{co-BH}(k) \). Clearly, \( \text{BL}_A(k) \in \text{P}[A]^{[k]} \) and \( \text{co-BL}_A(k) \in \text{P}[A]^{[k]} \). So, there is some interaction between these “boolean languages” built from \( A \) and the query hierarchies built from \( A \).

All these definitions in place we can pose some questions originally asked of \( \text{SAT} \). For example, we would like to know what happens if \( \text{P}[A]^{[1]} = \text{P}[A]^{[2]} \) or if \( \text{B}[A](k) \subseteq \text{co-B}[A](k) \). Also, we would like to know the relationship between queries to \( \text{SAT} \) and queries to \( A \). That is, we know \( \text{P}[\text{SAT}^{[1]}] \subseteq \text{P}[\text{SAT}^{[1]}] \), but where is \( \text{P}[\text{SAT}^{[2]}] \) in relation to \( \text{P}[\text{SAT}^{[1]}] \)? Instead of asking “Is one question to \( \text{SAT} \) as powerful as two?” we ask “Is one question to \( \text{SAT} \) as powerful as two questions to some other oracle?”

The following theorem answers some of these questions:

**Theorem** If \( A, B \in \text{NP} \) and \( \text{BL}_A(k) \subseteq \text{P}[B]^{[k]} \), then \( A \in \text{low}_3 \).

We prove this theorem in two parts. Lemma 1 is a rather technical lemma, and follows the same lines as Kadin’s original proof that \( \text{BL}_{\text{SAT}}(k) \subseteq \text{co-BL}_{\text{SAT}}(k) \) implies \( \text{PH} \subseteq \Delta^p_2 \). We relegate the proof of Lemma 1 to the next section.

**Lemma 1** If \( A, B \in \text{NP} \) and \( \text{BL}_A(k) \subseteq \text{P}[B]^{[k]} \), then there exists a self-p-printable set \( S \in \Delta^p_2 \) such that \( A \in \text{NP}^{F} \).
Lemma 2 If \( A \in \text{NP} \) and there exists a self-p-printable set \( S \subseteq \Delta_p^P \) such that \( A \in \text{NP}^S \), then \( A \in \text{low}_3 \).

Proof
If \( A \in \text{NP}^S \), then \( \text{NP}^A \subseteq \text{NP}^S \). (To answer a query to \( A \), the \( \text{NP}^S \) machine runs the \( \text{NP} \) algorithm for \( A \) and the \( \text{NP}^S \) algorithm for \( A \) in parallel. One of the algorithms will terminate.) So, we have

\[
P^{\text{NP}^A} \subseteq P^{	ext{NP}^S} \quad \text{(i.e., } \Delta_3^{P,A} \subseteq \Delta_3^{P,S})
\]

by replacing the \( \text{NP} \) oracle with an \( \text{NP}^S \) oracle. However, \( S \subseteq \Delta_3^P \) and is self-p-printable, so a \( \Delta_3^P \) machine can write down an initial segment of \( S \) that includes all queries to \( S \) in a \( \Delta_3^P \) computation. (The length of this initial segment is bounded by a polynomial.) Since \( \Delta_3^P \) consists of a \( P \) base machine and an \( \text{NP}^{\text{NP}} \) oracle, the \( P \) base machine (with the help of the \( \text{NP}^{\text{NP}} \) oracle) can write down this initial segment and send it along with subsequent oracle queries to \( \text{NP}^{\text{NP}} \). Now the \( \text{NP}^{\text{NP}} \) oracle does not need to consult an \( S \) oracle, so

\[
\Delta_3^{P,A} \subseteq \Delta_3^{P,S} \subseteq \Delta_3^P.
\]

Therefore, \( A \in \text{low}_3 \). \( \square \)

The theorem gives us a sufficient technical condition for a set to be in \( \text{low}_3 \). The following corollary clarifies the picture somewhat.

Corollary 1 If \( A \in \text{NP} \) and one of the following conditions holds, then \( A \in \text{low}_3 \).

1. \( P^A \subseteq P^{\text{SAT}^A} \)
2. \( P^{A[k+1]} \subseteq P^{\text{SAT}^A[k]} \), for some \( k \geq 1 \)
3. \( P^{A[k+1]} = P^{A[k]} \), for some \( k \geq 1 \)
4. \( P^{A[k+1]} = P[A] \), for some \( k \geq 1 \)

Proof
1. If \( P^A \subseteq P^{\text{SAT}^A} \), then

\[
\text{BL}_A(2) \in P^{A[2]} \subseteq P^{A[4]} \subseteq P^{\text{SAT}^A} \subseteq \text{co-BH}(2).
\]

However, \( \text{co-BL}_{\text{SAT}}(2) \) is \( \leq^m \) complete for \( \text{co-BH}(2) \), so \( \text{BL}_A(2) \leq^m_{\text{co-BL}_{\text{SAT}}(2)} \). Then, by the theorem, \( A \in \text{low}_3 \).

2. If \( P^{A[k+1]} \subseteq P^{\text{SAT}^A[k]} \), then

\[
\text{BL}_A(k+1) \in P^{A[k+1]} \subseteq P^{\text{SAT}^A[k]} \subseteq \text{co-BH}(k+1).
\]

\( \text{co-BL}_{\text{SAT}}(k+1) \) is \( \leq^m \) complete for \( \text{co-BH}(k+1) \), so \( \text{BL}_A(k+1) \leq^m_{\text{co-BL}_{\text{SAT}}(k+1)} \). Then, by the theorem, \( A \in \text{low}_3 \).

3. \( P^{A[k+1]} = P^{A[k]} \) implies \( P^{A[k+1]} \subseteq P^{\text{SAT}^A[k]} \), so \( A \in \text{low}_3 \).

4. \( P^{A[k+1]} = P[A] \) implies \( P^{A[k+1]} \subseteq P[A] \), because the whole query hierarchy based on \( A \) collapses. Thus, \( P^{A[k+1]} \subseteq P[A] \subseteq P^{A[k+1]} \). Then, by part 3, \( A \in \text{low}_3 \). \( \square \)

Parts 3 and 4 of Corollary 1 state that if the parallel or serial query hierarchies built from \( A \) collapses, then \( A \) is not very hard. Part 2 says that if a set \( A \) is not in \( \text{low}_3 \), then not only is the parallel query hierarchy built from \( A \) proper, it also rises in lock step with \( \text{QH}_{||} \) (see figure 1). That is, with \( k+1 \) parallel queries to \( A \), we rise above the \( k \)th level of \( \text{QH}_{||} \). We view this as evidence that having a proper query hierarchy is a result of the oracle access mechanism rather than the "hardness" of the oracle. We will come back to this theme later.

For hard sets in \( \text{NP} \), we can relate the theorem with the collapse of \( \text{PH} \).

Corollary 2 If \( A \in \text{high} \), for some \( j \) and one of the following holds, then \( \text{PH} \) is finite.

1. \( P^A \subseteq P^{\text{SAT}^A} \)
2. \( P^{A[k+1]} \subseteq P^{\text{SAT}^A[k]} \), for some \( k \geq 1 \)
3. \( P^{A[k+1]} = P^{A[k]} \), for some \( k \geq 1 \)
4. \( P^{A[k+1]} = P[A] \), for some \( k \geq 1 \)

Proof
Let \( i = \max(3, j) \). By Corollary 1, \( A \in \text{low}_3 \subseteq \text{low}_i \), so \( \Sigma_i^p = \Sigma_i^{P,A} \). However, \( A \in \text{high}_j \subseteq \text{high}_i \), means \( \Sigma_i^{P,A} = \Sigma_{i+1}^p \). Thus, \( \Sigma_i^p = \Sigma_i^{P,A} = \Sigma_{i+1}^p \) and \( \text{PH} \subseteq \Sigma_{i+1}^p \). \( \square \)

In particular, if the query hierarchy built from a set Turing complete for \( \text{NP} \) collapses, then \( \text{PH} \) collapses to \( \Delta_i^P \).

This generalizes Kadin's result for \( \text{SAT} \) and answers the question "Is one query to \( \text{SAT} \) as powerful as two queries to some other oracle?"

Corollary 3 If there exists \( A \) such that \( P^A = P^{\text{SAT}^A} \), then \( \text{PH} \subseteq \Delta_i^P \).

Proof
If \( P[A] = P^{\text{SAT}^A} \), then \( A \in P[A] = P^{\text{SAT}^A} \). So, we know that \( A \leq^m \text{SAT} \oplus \text{SAT} \) via some polynomial time function \( g : \text{SAT} \oplus \text{SAT} \) is defined by

\[
\text{SAT} \oplus \text{SAT} = \{ 0F \mid F \in \text{SAT} \} \cup \{ 1F \mid F \in \text{SAT} \}.
\]

We split \( A \) into two sets,
\[ A_0 = \{ x \mid x \in A \text{ and } g(x) = 0 \text{ for some } F \} \]
\[ A_1 = \{ x \mid x \in A \text{ and } g(x) = 1 \text{ for some } F \} \]

Clearly, \( A_0 \in \text{NP} \) and \( A_1 \in \text{co-NP} \). Now let \( C = A_0 \oplus A_1 \).
One can easily see that \( C \in \text{NP} \) and that
\[ P^{A_0} = P^{C}, \text{ for all } k. \]

So, \( P^{C} = P^{A_0} = P^{\text{SAT}} \). However, \( \text{SAT} \in P^{A_0} \) implies \( C \)
is Turing complete for \( \text{NP} \). So, \( C \) is a high set. Then, by
the proof of Corollary 2, part 1, \( P^{A_0} \subseteq P^{\text{SAT}} \) implies \( \text{PH} \)
collapses to \( \Delta^p_3 \).

Corollaries 4 and 5 explore some technical conditions
that allow us to strengthen some results about the serial
query hierarchies.

**Corollary 4**
If \( A \in \text{NP} \) is low, then there exist \( B \in \text{NP} \)
such that \( A \leq_m B, P^A = P^B \) and for all \( k, P^{B[k]} \subseteq P^{\text{SAT}} \).

**Proof (Sketch)**
Note that if \( P^B = P^A \) then \( A \) and \( B \) are in the same high
or low level. So, this corollary states that for every high
and low level above low there is a set \( B \) whose serial
query hierarchy rises in lock step with the serial
query hierarchy for \( \text{SAT} \).

Part 2 of Corollary 1 tells us that each additional parallel
query to \( A \) allows the oracle machine to recognize a set
not recognized with one less parallel query to \( \text{SAT} \). To ex-
tend this to serial queries, we need to mimic Beigel's "mind
change" trick. \( \text{SAT} \) has some special properties that make
the trick work.

**Definition**
\[ \text{OR}_2^C = \{ (x, y) \mid x \in C \text{ or } y \in C \} \]
\[ \text{AND}_2^C = \{ (x, y) \mid x \in C \text{ and } y \in C \} \]
(The "2" in the superscript means binary.)

\( \text{SAT} \) is a special set because \( \text{OR}_2^{\text{SAT}} \leq_m \text{SAT} \)
and \( \text{AND}_2^{\text{SAT}} \leq_m \text{SAT} \). By going over Beigel's "mind change" proof [1],
one can show that if a set \( C \in \text{NP} \) has the property that
\( \text{OR}_2^C \leq_m C \text{ and } \text{AND}_2^C \leq_m C \), then
\[ P^{\text{OR}_2^C} = P^{\text{AND}_2^C}. \]

Now we modify \( A \) slightly, so it has the desired proper-
ties. Consider, the set
\[ B = \{ (F, x_1, \ldots, x_n) \mid F \text{ is a boolean formula over } n \]
variables \( y_1, \ldots, y_n \) without negation, and when
\( y_i \) is evaluated as \( x_i \in A \), \( F \) evaluates to 1. \}

Clearly, \( B \) is in \( \text{NP} \) and \( A \leq_m B \). Also, \( P^B = P^A \) implies \( B \)
\( \in \text{NP} \). Moreover, \( \text{OR}_2^B \leq_m B \) and \( \text{AND}_2^B \leq_m B \), so
Now if \( P^{\#P[1]} \subseteq P^{SAT[1]} \), we know
\[
BL_A(2^k) \subseteq P^{\#P[1]} \subseteq P^{\#P[2^k+1]} = P^{\#P[2^k]}
\]
\[\subseteq P^{SAT[2^k]} = P^{SAT[3^k]} \subseteq co-BH(2^k)\]
which tells us \( BL_A(2^k) \leq^p co-SAT(2^k) \) and contradicts the assumption that \( B \) is not in \( \text{low}_2 \).

We can say more about the existence of easy sets whose serial query hierarchies rise in lock step with \( QH \). (It may be the case that the only sets in \( \text{NP} \) complete for \( \text{NP} \) or complete for \( \text{NP} \) in some other sense. However, in this case \( \text{PH} \) is finite.)

**Proof**

If \( \text{PH} \) is infinite, then by a Ladner-like delayed diagonalization \([5,6]\) we can construct a set \( I \in \text{NP} \) that is neither high nor low \((I \) stands for intermediate\). In particular, \( I \notin \text{low}_3 \), so using Corollary 4 we can obtain a set \( B \) such that for all \( k \),
\[
P^{\#P[1]} \not\subseteq P^{\text{SAT}[k]}.
\]
Since \( P^I = P^I \), \( B \) is intermediate iff \( I \) is intermediate. So, \( B \) is not high.

Note that if \( B \) is not high, then \( B \) is not \( \text{NP} \) hard under many-one, Turing, strong nondeterministic or other asorted reductions. In particular \( B \) is not \( \text{NP} \) complete for \( \text{NP} \), so \( \text{SAT} \notin P^B \). So, Corollary 5 says that the serial query hierarchy built from \( B \) rises in lock step with \( QH \), but never captures \( \text{SAT} \). In this sense, we know that if \( \text{PH} \) is infinite, then there exist easy sets whose serial query hierarchies are as “tall” as \( QH \).

**5 Proof of Lemma 1**

**Lemma 1** If \( A, B \in \text{NP} \) and \( BL_A(k) \leq^p co-BL_B(k) \), then there exists a self-p-printable set \( S \in \Delta^p_3 \) such that \( A \in \text{NP}^S \).

**Proof**

We prove this by producing a \( \Delta^p_3 \) program which on input \( 1^n \) generates a finite set \( T_n \) with \( \leq kn \) elements. Furthermore, every string in \( T_n \) will have length \( n \). The set produced, \( S = \bigcup_{n \geq 1} T_n \), will have the specified properties. The main part of the program is a loop which is iterated for values of \( i \) from 0 up to \( k - 1 \). Each iteration produces either the desired \( T_n \) or a “reduction” from \( BL_A(k - i) \) to \( co-BL_B(k - i) \) for strings of length \( n \).

In the following discussion, let \( g \) be the polynomial time function that reduces \( BL_A(k) \) to \( co-BL_B(k) \) and let \( j = k - i \). We maintain the following loop invariant to assist our proof. Before each iteration, we have \( z_1, \ldots, z_i \in \{0, 1\}^n \) such that
\[
\forall x_1, \ldots, x_j \in \{0, 1\}^n, \quad g((z_1, \ldots, z_i, x_j)) = \langle u_1, \ldots, u_j, v_1, \ldots, v_l \rangle
\]
where \( u_j \) is the projection onto the \( j \)th element.

This loop invariant holds trivially for \( i = 0 \), since \( g \) is a reduction from \( BL_A(k) \) to \( co-BL_B(j) \). The body of the loop is given below. Note that the loop terminates either at step 1 or at step 4.

1. if \( i = k - 1 \), then write down \( z_1, \ldots, z_i \) and exit the loop.

2. Compile a function \( h : \{0, 1\}^{n+i} \rightarrow \{0, 1\}^i \) defined
\[
h((x_1, \ldots, x_i, z_i)) = \pi_j g((z_1, \ldots, z_i, x_i, \ldots, x_i))
\]
where \( \pi_j \) is the projection onto the \( j \)th element.

3. Ask the \( \text{NP}^{\text{NP}} \) oracle:
\[
\forall x \in A^n \exists x_1, \ldots, x_{j-1} \in \{0, 1\}^n, \quad h((x_1, \ldots, x_{j-1}, x)) \in B
\]

4. If the oracle answers “yes”, then write down \( z_1, \ldots, z_i \), and exit the loop.

5. If the oracle answers “no”, then we know there exists an \( x \in A^{\pi_n} \) such that
\[
\forall x_1, \ldots, x_{j-1} \in \{0, 1\}^n, \quad h((x_1, \ldots, x_{j-1}, x)) \notin B.
\]

Using binary search and the \( \text{NP}^{\text{NP}} \) oracle, we can find the lexically smallest such \( x \) and write it down. Let \( z_{i+1} = x \) and advance to the next iteration.

Now we show how to construct \( T_n \). There are two cases. If \( A^{\pi_n} = \emptyset \), then we put \( \emptyset \) in \( T_n \). Note that we can easily check if \( A^{\pi_n} \neq \emptyset \) with a \( P^I \) question. If \( A^{\pi_n} \neq \emptyset \), then we start the loop described above with \( i = 0 \). If the loop terminates, we put \( z_1, \ldots, z_i \) and all their padded prefixes in \( T_n \).

We still have to prove two claims. First, we must show that the loop invariant holds from iteration to iteration. Also, we need to show that \( A^{\pi_n} \in \text{NP}^{\text{NP}} \).

**Claim 1:** Suppose that in some iteration \( i \) we reach step 1. From the loop invariant of the current loop iteration, we know that
∀x1, ..., xj ∈ {0, 1}n,

\[ g((x_1, ..., x_j, z_j, z)) = (u_1, ..., u_j, v_1, ..., v_j) \]

\[ \iff (x_1, ..., x_j, z_j) \in BL_a(j) \]

\[ \iff (u_1, ..., u_j) \in co-BL_b(j) \].

If \( j \) is even, then by the definition of \( BL_a(j) \) and \( co-BL_b(j) \) we have

\[ (x_1, ..., x_j-1) \in BL_a(j-1) \] \[ \iff (u_1, ..., u_j-1) \in co-BL_b(j-1) \] \[ or (by negating both sides of the iff) \]

\[ (x_1, ..., x_j-1) \notin BL_a(j-1) \] \[ \iff (u_1, ..., u_j-1) \notin co-BL_b(j-1) \] \[ or (by negating both sides of the iff) \]

\[ (x_1, ..., x_j-1) \in BL_a(j-1) \] \[ \iff (u_1, ..., u_j-1) \in co-BL_b(j-1) \].

Fixing \( x_j \) to be the \( x \) found in step 5, we get

\[ (x_1, ..., x_j-1) \in BL_a(j-1) \] \[ \iff (u_1, ..., u_j-1) \in co-BL_b(j-1) \] \[ or (by negating both sides of the iff) \]

\[ (x_1, ..., x_j-1) \notin BL_a(j-1) \] \[ \iff (u_1, ..., u_j-1) \notin co-BL_b(j-1) \] \[ or (by negating both sides of the iff) \]

\[ (x_1, ..., x_j-1) \in BL_a(j-1) \] \[ \iff (u_1, ..., u_j-1) \in co-BL_b(j-1) \].

So, in both cases we manage to maintain the loop invariant.

Claim 2: We need to show that \( \overline{A}^{=n} \in \text{NP}^T_n \). There are two cases to consider.

Case 1: If the loop terminated with \( i = k - 1 \), then from the loop invariant we know that \( ∀x \in {0, 1}^n \),

\[ g((x_1, ..., x_k, z_k, z)) = (u_1, ..., u_k, v_1, ..., v_k) \]

\[ \iff (x \in BL_a(1) \iff u \in co-BL_b(1)) \].

However, \( BL_a(1) = A \) and \( co-BL_b(1) = B \), so we really have

\[ x \in A \iff u \in \overline{B} \]

or (by negating both sides of the iff)

\[ x \in \overline{A} \iff u \in B. \]

To check if \( x \in \overline{A}^{=n} \) an \( \text{NP}^T_n \) machine simply queries \( T_n \) to find \( z_1, ..., z_{k-1} \), computes \( u = \pi_{xy}(x, z_{k-1}, ..., z) \) and accepts iff \( u \in B \).

Case 2: If the loop terminated with \( i < k - 1 \), we want to show that

\[ x \in \overline{A}^{=n} \iff \exists x_1, ..., x_j \in \{0, 1\}^n, h((x_1, ..., x_j, x)) \in B, \]

where \( j = k - i \) and

\[ h((x_1, ..., x_j)) = \pi_{xy}((x_1, ..., x_j, z_1, ..., z_j)). \]

Since the loop terminated at step 4, the \( \text{NP}^B \) oracle must have answered “yes” to “∀x \in \overline{A}^{=n} \exists x_1, ..., x_{j-1} \in \{0, 1\}^n, h((x_1, ..., x_{j-1}, x)) \in B?” So, we obtain one direction of the iff

\[ x \in \overline{A}^{=n} \iff \exists x_1, ..., x_{j-1} \in \{0, 1\}^n, h((x_1, ..., x_{j-1}, x)) \in B. \]

Moreover, we know from the loop invariant that

\[ (x_1, ..., x_{j-1}, x_1) \in BL_a(j-1) \] \[ \iff (u_1, ..., u_{j-1}) \in co-BL_b(j-1) \] \[ or (by negating both sides of the iff) \]

\[ (x_1, ..., x_{j-1}) \notin BL_a(j-1) \] \[ \iff (u_1, ..., u_{j-1}) \notin co-BL_b(j-1) \] \[ or (by negating both sides of the iff) \]

\[ (x_1, ..., x_{j-1}) \in BL_a(j-1) \] \[ \iff (u_1, ..., u_{j-1}) \in co-BL_b(j-1) \].

Fixing \( x_j \) to be the \( x \) found in step 5, we get

\[ (x_1, ..., x_{j-1}) \in BL_a(j-1) \] \[ \iff (u_1, ..., u_{j-1}) \in co-BL_b(j-1) \] \[ or (by negating both sides of the iff) \]

\[ (x_1, ..., x_{j-1}) \notin BL_a(j-1) \] \[ \iff (u_1, ..., u_{j-1}) \notin co-BL_b(j-1) \] \[ or (by negating both sides of the iff) \]

\[ (x_1, ..., x_{j-1}) \in BL_a(j-1) \] \[ \iff (u_1, ..., u_{j-1}) \in co-BL_b(j-1) \].

Note that \( u_j = h((x_1, ..., z_{j-1}, x)) \) and that in either case \( u_j \in B \) implies \( x \in \overline{A} \). So, we obtain the other direction of the iff

\[ \exists x_1, ..., x_{j-1} \in \{0, 1\}^n, h((x_1, ..., x_{j-1}, x)) \in B \]

\[ \iff x \in \overline{A}. \]

Combining the implications in (1) and (2), we have

\[ x \in \overline{A}^{=n} \iff \exists x_1, ..., x_{j-1} \in \{0, 1\}^n, h((x_1, ..., x_{j-1}, x)) \in B. \]

This relationship allows us to compute \( \overline{A}^{=n} \) with an \( \text{NP}^B \) machine. To check if \( x \in \overline{A}^{=n} \) an \( \text{NP}^B \) machine queries \( T_n \) to find \( z_1, ..., z_n \), guesses \( x_1, ..., x_{j-1} \), and accepts iff \( h((x_1, ..., x_{j-1}, x)) \in B \).

In summary, we constructed \( S \subseteq \Delta^P_n \) length by length (i.e., \( S = \bigcup_{n \geq 1} T_n \)). Each \( T_n \) has at most \( kn \) strings and all the strings in \( T_n \) are of length \( n \). Also, each \( T_n \) is closed under prefixes, so \( S \) is self-p-printable. Finally, \( \overline{A} \in \text{NP}^S \) because the following \( \text{NP}^S \) program determines if \( x \in \overline{A} \):

256
1. Let \( |x| = n \).
2. If \( \emptyset^n \in S \), then \( \overline{A}^m = \emptyset \). Reject \( x \).
3. Print out the strings \( z_1, \ldots, z_i \) in \( S^m \). Let \( j = k - i \).
4. If \( i = k - 1 \), compute \( u = \pi_3g((x, z_1, \ldots, z_i)) \) and accept iff \( u \in B \).
5. If \( i < k - 1 \) accept \( x \) iff
   \[
   \exists x_1, \ldots, x_{j-1} \in \{0,1\}^n, \\
   \pi_3g((x_1, \ldots, x_{j-1}, x, z_1, \ldots, z_i)) \in B.
   \]

6 Discussion

We have been saying that a proper hierarchy is the result of the oracle access mechanism rather than the hardness of the oracle. In this section we elaborate on this point. This is an intuitive discussion.

One can describe an oracle machine’s computation in three parts: the base machine, the oracle and the oracle access mechanism. In bounded query computations, the oracle access is the resource bottleneck of the computation. Suppose \( A \in \text{NP} - \text{low}_3 \), then we know that \( \text{BLA}(k+1) \) is in \( P^{4l[k+1]} \) but not in \( P^{4l[k]} \). If we replace the oracle in the \( P^{4l[k]} \) machine with \( \text{SAT} \), the \( P^{\text{SAT}[l]} \) computation still cannot recognize \( \text{BLA}(k+1) \). So, a more powerful \( \text{NP} \) oracle does not help the base machine recognize \( \text{BLA}(k+1) \), if the base machine is allowed only \( k \) parallel queries. Also, note that a \( P^{4l[k+1]} \) machine can compute \( \text{BLA}(k+1) \) in linear time and zero work space. On the other hand, a more powerful polynomial time base machine still cannot compute \( \text{BLA}(k+1) \) if it is allowed only \( k \) parallel queries to an \( \text{NP} \) oracle. In contrast, once we allow the communication channel between the oracle and the base machine to widen to \( k + 1 \) bits, the oracle computation can recognize \( \text{BLA}(k+1) \). We think of the bounded oracle queries as a very fragile resource. (Resources like space and time are robust and have speed up theorems.) So, we conclude that the oracle access mechanism plays the crucial role in building a proper query hierarchy.

7 Conclusion

Many open questions remain. For example, we know that query hierarchies built from sets in \( \text{P} \) always collapse. We also know that for any \( A \in \text{NP} \cap \text{co-NP} \) there exists \( B \in \text{NP} \cap \text{co-NP} \) such that \( A \leq^p \text{co-NP} \) and the query hierarchy built from \( B \) collapses. Are there sets in \( \text{NP} \cap \text{co-NP} \) that have proper query hierarchies? What about sets between \( \text{low}_3 \) and \( \text{low}_1 = \text{NP} \cap \text{co-NP} \)? Are their query hierarchies proper? Many interesting language classes live in this region, including sparse sets in \( \text{NP} \), \( \text{NP} \cap \text{P/poly} \), \( \text{R} \) and \( \text{BPP} \). Can we show that any of these sets have proper or collapsing hierarchies? Also, we would like to strengthen the results for serial query hierarchies. We know that

\[ \text{AND}^2_A \leq^p \text{A}, \quad \text{and} \quad \text{OR}^2_A \leq^p \text{A} \]

is a sufficient condition. We know that this condition holds for all \( \leq^p \)-complete sets, all sets in \( \text{P} \), and \( \text{Graph Isomorphism} \). Primes remains the only candidate for natural language that does not have this property. We would like to see either a construction for an \( \text{NP} \) set without this property, or a proof that all \( \text{NP} \) sets have this property.

Now that we know more about query hierarchies, we see that having a proper query hierarchy is not entirely dependent on the “hardness” of a set. In fact, sets in \( \text{low}_3 \), \( \text{low}_2 \), \( \text{low}_1 \), if they exist, have proper query hierarchies. Instead we believe that having a proper query hierarchy is a result of the meagerness of the access mechanism. When you are not told very much, every bit of extra information helps. However, we should not consider this an artifact of the definition, because we get a glimpse of the structure of the Polynomial Hierarchy in these “mini” hierarchies within \( \Delta^P_2 \).

8 Acknowledgements

The author would like to thank his advisor Juris Hartmanis for guidance and support, Jim Kadin for invaluable discussions and for suggesting nice open questions, Wei Li and Desh Ranjan for keeping him on his toes, Georges Lauri for proofreading this paper, and Christine Piatko for lending him a red pen during a time of great need.

References


