On Computing Boolean Connectives of Characteristic Functions

Richard Chang*
Computer Science Department
Cornell University
Ithaca, NY 14853

Jim Kadin
Computer Science Department
University of Maine
Orono, ME 04469

November 10, 1992

Abstract

This paper is a study of the existence of polynomial time Boolean connective functions for languages. A language L has an AND function if there is a polynomial time f such that $f(x,y) \in L \iff x \in L$ and $y \in L$. L has an OR function if there is a polynomial time g such that $g(x,y) \in L \iff x \in L$ or $y \in L$. While all NP complete sets have these functions, Graph Isomorphism, which is probably not complete, is also shown to have both AND and OR functions. The results in this paper characterize the complete sets for the classes D^{P} and $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ in terms of AND and OR, and relate these functions to the structure of the Boolean hierarchy and the query hierarchies. Also, this paper shows that the complete sets for the levels of the Boolean hierarchy above the second level cannot have AND or OR unless the polynomial hierarchy collapses. Finally, most of the structural properties of the Boolean hierarchy and query hierarchies are shown to depend only on the existence of AND and OR functions for the NP complete sets.

1 Introduction

In this paper we consider the existence of polynomial time Boolean combining functions for languages. We say that a language L has a binary AND function, i.e. an AND_2 function, if there is a polynomial time function f such that for all strings x and y, $f(x,y) \in L$ if and only if $x \in L$ and $y \in L$. Similarly, we say that a language L has a binary OR function, an OR_2 function, if there is a polynomial time function g such that for all strings x and y, $g(x,y) \in L$ if and only if $x \in L$ or $y \in L$. In addition, a language may have "any-ary" Boolean functions (AND_ω and OR_ω), polynomial time functions f and g such that for all n and strings $x_1, \cdots, x_n$, $f(x_1, \cdots, x_n) \in L$ if and only if $x_1, \cdots, x_n$ are all in L, and $g(x_1, \cdots, x_n) \in L$ if and only if at least one of $x_1, \cdots, x_n$ is in L.

*Supported in part by NSF research grants DCR-8520597 and CCR-88-23053, and by an IBM Graduate Fellowship.
Current address: Computer Science Department, University of Maryland Baltimore County, Baltimore, MD 21228.

The existence of these functions is intimately tied to questions about polynomial time reducibilities and structural properties of languages and complexity classes. They are also related to various notions of the internal structure such as self-reducibility and p-selectivity (c.f. [GJY87, Sel79]), but in our case, we do not restrict the length or output of the polynomial time functions. Our initial motivation for considering these functions was the observation that all NP complete languages have AND_ω and OR_ω functions. In fact any language that is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for even relativized versions of complexity classes such as P, NP, PSPACE have any-ary Boolean functions, because for these robust classes there exist universal machines which can run many computations simultaneously or in sequence. We will show that languages that are $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for D^{P} [PY84] have AND_ω but do not have OR_2 unless the polynomial time hierarchy collapses. Complete languages for the higher levels of the Boolean hierarchy [CGH$^+$88] do not have either AND_2 or OR_2 unless the polynomial time hierarchy collapses.

These Boolean functions are related to polynomial time conjunctive and disjunctive reducibilities and to the closure of complexity classes under union and intersection.

Definition [LLS75] Let A and B be any two languages. Then, A conjunctively reduces to B (written $A \leq^{\mathrm{P}}_{\mathrm{c}} B$) if there exists a polynomial time function f such that for all x

$$x \in A \iff f(x) = \langle y_1, \ldots, y_r \rangle \text{ and } \forall i,\ 1 \leq i \leq r,\ y_i \in B.$$

In addition, we write $A \leq^{\mathrm{P}}_{\text{2-c}} B$ if $A \leq^{\mathrm{P}}_{\mathrm{c}} B$ via a polynomial time function f which always outputs a 2-tuple. Similarly, A disjunctively reduces to B (written $A \leq^{\mathrm{P}}_{\mathrm{d}} B$) if there exists a polynomial time function f such that for all x

$$x \in A \iff f(x) = \langle y_1, \ldots, y_r \rangle \text{ and } \exists i,\ 1 \leq i \leq r,\ y_i \in B.$$

Again, if the output of f is always a 2-tuple, then $A \leq^{\mathrm{P}}_{\text{2-d}} B$.

Let m-1(L) be the class of languages $\leq^{\mathrm{P}}_{\mathrm{m}}$- reducible to a language L. Then,

$$\begin{aligned} L \text{ has } \mathrm{AND}_2 &\iff \text{m-1}(L) \text{ is closed under intersection} \\ &\iff \text{m-1}(L) \text{ is closed under } \leq^{\mathrm{P}}_{\text{2-c}}, \end{aligned}$$

$$L \text{ has } \mathrm{AND}_\omega \iff \text{m-1}(L) \text{ is closed under } \leq^{\mathrm{P}}_{\mathrm{c}}.$$

Similarly, OR is related to disjunctive reducibilities and union. Hence by looking at these concepts in terms of Boolean functions for languages, we are simply thinking of them more as structural properties of languages than as structural properties of complexity classes. An advantage of this approach is that it becomes convenient to study interesting languages such as Graph Isomorphism and USAT (the set of Boolean formulas that have exactly one satisfying assignment) that are not known to be $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for any standard classes.

The paper is organized as follows. Section 2 presents definitions and preliminary concepts. In Section 3 we discuss some languages that have AND and OR functions. Most notably, we show that Graph Isomorphism does have any-ary AND and OR functions even though it is not known to be NP complete. In Section 4 we characterize the $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete languages of D^{P} and $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ in terms of AND and OR functions. In Section 5 we use the above characterizations to show that the complete languages of the higher levels of the Boolean hierarchy and the related query hierarchies do not have AND and OR functions unless the Boolean hierarchy collapses (which implies that the polynomial time hierarchy collapses [Kad88, CK90]).

In Section 6, we observe that the existence of AND and OR functions for languages is a condition that makes many proof techniques work. For instance the mind-change technique [Bei91, WW85] which showed that $\mathrm{P}^{\mathrm{SAT}\|[2^k-1]} = \mathrm{P}^{\mathrm{SAT}[k]}$ works for any set A that has binary AND and OR functions. Similarly, most of the theorems concerning the basic structure and normal forms of the Boolean hierarchy and the intertwining of the Boolean and query hierarchies depend only on the fact that SAT has AND_2 and OR_2. The results of this section have been proven independently by Bertoni, Bruschi, Joseph, Sitharam, and Young [BBJ$^+$89].

Finally, in Section 7, we use diagonalization techniques to show that there exist languages which have any subset of AND_2, AND_ω, OR_2 and OR_ω so long as they do not violate the implications:

$$A \text{ has } \mathrm{AND}_\omega \Longrightarrow A \text{ has } \mathrm{AND}_2, \quad \text{and} \quad A \text{ has } \mathrm{OR}_\omega \Longrightarrow A \text{ has } \mathrm{OR}_2.$$

2 Definitions, Notations and Facts

We assume the reader is familiar with the classes P, NP, the polynomial time hierarchy (PH), the NP complete language SAT, the graph isomorphism problem, and polynomial time many-one reducibility ($\leq^{\mathrm{P}}_{\mathrm{m}}$). Although our original notion of AND and OR consists of polynomial time functions that operate on strings (c.f. [Kad89]), as a matter of notational convenience, we will define AND_2, AND_ω, OR_2, OR_ω as sets in this paper.

Definition For any set A, we define the sets:

$$\begin{aligned}
\mathrm{AND}_2(A) &= \{\, \langle x, y\rangle \mid x \in A \text{ and } y \in A \,\} \\
\mathrm{OR}_2(A) &= \{\, \langle x, y\rangle \mid x \in A \text{ or } y \in A \,\} \\
\mathrm{AND}_k(A) &= \{\, \langle x_1, \ldots, x_k\rangle \mid \forall i,\ 1 \leq i \leq k, x_i \in A \,\} \\
\mathrm{OR}_k(A) &= \{\, \langle x_1, \ldots, x_k\rangle \mid \exists i,\ 1 \leq i \leq k, x_i \in A \,\} \\
\mathrm{AND}_\omega(A) &= \bigcup_{i=1}^{\infty} \mathrm{AND}_i(A) \\
\mathrm{OR}_\omega(A) &= \bigcup_{i=1}^{\infty} \mathrm{OR}_i(A).
\end{aligned}$$

If $\mathrm{AND}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$ or $\mathrm{AND}_\omega(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$, then we say that A *has* AND_2 or AND_ω respectively. This corresponds to our initial motivation for AND, since a language L "has an AND_2 function" if and only if $\mathrm{AND}_2(L) \leq^{\mathrm{P}}_{\mathrm{m}} L$. Similarly, If $\mathrm{OR}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$ or $\mathrm{OR}_\omega(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$, then we say that A *has* OR_2 or OR_ω respectively.

Some elementary facts about $\mathrm{AND}_2(A)$ and $\mathrm{OR}_2(A)$ are:

1. $\mathrm{AND}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A \iff \mathrm{OR}_2(\overline{A}) \leq^{\mathrm{P}}_{\mathrm{m}} \overline{A}$.
2. if $A \leq^{\mathrm{P}}_{\mathrm{m}} B$ then for all $k \geq 2$, $\mathrm{OR}_k(A) \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{OR}_k(B)$ and $\mathrm{AND}_k(A) \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{AND}_k(B)$.
3. if $A \equiv^{\mathrm{P}}_{\mathrm{m}} B$ then $\mathrm{AND}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A \iff \mathrm{AND}_2(B) \leq^{\mathrm{P}}_{\mathrm{m}} B$.
4. if $A \equiv^{\mathrm{P}}_{\mathrm{m}} B$ then $\mathrm{OR}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A \iff \mathrm{OR}_2(B) \leq^{\mathrm{P}}_{\mathrm{m}} B$.

These facts hold for $\mathrm{AND}_\omega(A)$ and $\mathrm{OR}_\omega(A)$ as well.

Observe that if $\mathrm{AND}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$, then for all k, $\mathrm{AND}_k(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$ by repeated application of the reduction from $\mathrm{AND}_2(A)$ to A. However, it is possible for $\mathrm{AND}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$, but $\mathrm{AND}_\omega(A) \not\leq^{\mathrm{P}}_{\mathrm{m}} A$ (see Section 7). On the other hand, if $\mathrm{AND}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$ by a *linear time* reduction, then $\mathrm{AND}_\omega(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$.

Lemma 1 If $\mathrm{AND}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$ via a linear time reduction, then $\mathrm{AND}_\omega(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$. Analogously, if $\mathrm{OR}_2(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$ via a linear time reduction, then $\mathrm{OR}_\omega(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$.

Proof: Let f be a linear time $\leq^{\mathrm{P}}_{\mathrm{m}}$-reduction from $\mathrm{AND}_2(A)$ to A. For any r, we can take a tuple $\langle x_1, \ldots, x_r \rangle$ and apply f pairwise to $f(x_1, x_2)\ f(x_3, x_4) \cdots f(x_{r-1}, x_r)$. (If r is odd, then we add a trivial element of A to the list.) Then we can apply f pairwise to the outputs of the first applications of f. Repeating this process until we have a single string gives us a tree of applications of f. The height of the tree is $\lceil \log r \rceil$. If n is the total length of the tuple, $r \leq n$, and so the total running time is bounded by $c^{\log n} n$ for some constant c. This is polynomial in n. (Recall that n is the total length of the original tuple, not the length of each x_i.) □

Definition [CGH+88] We write BH_k and co-BH_k for the k^{th} levels of the Boolean hierarchy, defined as follows:

$$
\begin{aligned}
\mathrm{BH}_1 &= \mathrm{NP}, \\
\mathrm{BH}_2 &= \{\ L_1 \cap \overline{L_2} \mid L_1, L_2 \in \mathrm{NP}\ \}, \\
\mathrm{BH}_{2k} &= \{\ L_1 \cap \overline{L_2} \mid L_1 \in \mathrm{BH}_{2k-1} \text{ and } L_2 \in \mathrm{NP}\ \}, \\
\mathrm{BH}_{2k+1} &= \{\ L_1 \cup L_2 \mid L_1 \in \mathrm{BH}_{2k} \text{ and } L_2 \in \mathrm{NP}\ \}, \\
\text{co-}\mathrm{BH}_k &= \{\ L \mid \overline{L} \in \mathrm{BH}_k\ \}, \\
\mathrm{BH} &= \bigcup_{k=1}^{\infty} \mathrm{BH}_k.
\end{aligned}
$$

Definition We write BL_k for the canonical complete language for BH_k and co-BL_k for the complete language for co-BH_k:

$$\begin{aligned}
\mathrm{BL}_1 &= \mathrm{SAT}, \\
\mathrm{BL}_2 &= \{\ \langle x_1, x_2\rangle \mid x_1 \in \mathrm{SAT} \text{ and } x_2 \in \overline{\mathrm{SAT}}\ \}, \\
\mathrm{BL}_{2k} &= \{\ \langle x_1, \ldots, x_{2k}\rangle \mid \langle x_1, \ldots, x_{2k-1}\rangle \in \mathrm{BL}_{2k-1} \text{ and } x_{2k} \in \overline{\mathrm{SAT}}\ \}, \\
\mathrm{BL}_{2k+1} &= \{\ \langle x_1, \ldots, x_{2k+1}\rangle \mid \langle x_1, \ldots, x_{2k}\rangle \in \mathrm{BL}_{2k} \text{ or } x_{2k+1} \in \mathrm{SAT}\ \}, \\
\text{co-}\mathrm{BL}_1 &= \overline{\mathrm{SAT}}, \\
\text{co-}\mathrm{BL}_{2k} &= \{\ \langle x_1, \ldots, x_{2k}\rangle \mid \langle x_1, \ldots, x_{2k-1}\rangle \in \text{co-}\mathrm{BL}_{2k-1} \text{ or } x_{2k} \in \mathrm{SAT}\ \}, \\
\text{co-}\mathrm{BL}_{2k+1} &= \{\ \langle x_1, \ldots, x_{2k+1}\rangle \mid \langle x_1, \ldots, x_{2k}\rangle \in \text{co-}\mathrm{BL}_{2k} \text{ and } x_{2k+1} \in \overline{\mathrm{SAT}}\ \}.
\end{aligned}$$

Since BH_k is composed of k NP languages and since SAT is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for NP, a straightforward substitution will show that BL_k is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for BH_k. Alternatively, BH_2 is called $\mathrm{D}^{\mathrm{P}} = \{\ L_1 - L_2 \mid L_1, L_2 \in \mathrm{NP}\ \}$ (since set subtraction is equivalent to intersection with the complement). We will use the two terms interchangeably. Also, we will refer to BL_2 as $\mathrm{SAT} \wedge \overline{\mathrm{SAT}}$ the traditional $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete language for D^{P} [PY84]. An example of a language in D^{P} is USAT, the set of Boolean formulas with exactly one satisfying assignment. USAT is an element of D^{P} because

$$\mathrm{USAT} = \mathrm{SAT} - \{\ F \mid F \text{ is a Boolean formula with 2 or more satisfying assignments.}\ \}.$$

Definition For any set A, we write $\mathrm{P}^{A[f(n)]}$ for the set of languages recognized by polynomial time Turing machines that ask at most $f(n)$ queries to the oracle A on inputs of length n. We will be mainly concerned with the case where $f(n)$ is a constant k. So we write $\mathrm{P}^{A[k]}$ for languages recognized by polynomial time Turing machines that ask at most k queries on inputs of any length. Similarly, $\mathrm{P}^{A[O(\log n)]}$ is the set of languages recognized by polynomial time Turing machines that ask $O(\log n)$ queries to A.

Definition For any set A, we write $\mathrm{P}^{A\|[f(n)]}$ for the set of languages recognized by polynomial time Turing machines which ask at most $f(n)$ queries in *parallel* on inputs of length n. In the parallel (or *non-adaptive*) query model, the polynomial time machine is allowed to read the input, compute the $f(n)$ query strings $q_1, \ldots, q_{f(n)}$ (without the help of the oracle) and ask all $f(n)$ queries at once. The oracle answers with a bit vector $b_1, \ldots, b_{f(n)}$ such that $q_i \in A \iff b_i = 1$. It is important to note that in the parallel query model the polynomial time base machine cannot use the oracle to determine which string to query. In the serial query model the query strings can depend on the answers of previous queries. Again, we will mostly be concerned with the case where $f(n)$ is a constant. Also, we will use the shorthand $\mathrm{P}^{A\|}$ for $\bigcup_{k\geq 1}^{\infty} \mathrm{P}^{A\|[n^k+k]}$.

Definition The *query hierarchy* over A is denoted by $\mathrm{QH}(A) = \bigcup_{k=1}^{\infty} \mathrm{P}^{A[k]}$. Also, we define the *parallel query hierarchy* over A as $\mathrm{QH}_{\|}(A) = \bigcup_{k=1}^{\infty} \mathrm{P}^{A\|[k]}$. Finally, as a shorthand, we write QH for $\mathrm{QH}(\mathrm{SAT})$ and $\mathrm{QH}_{\|}$ for $\mathrm{QH}_{\|}(\mathrm{SAT})$.

Please note that the bounded query classes defined above are classes of languages and not of functions. The complexity of bounded query function classes have also been studied extensively [AG88, ABG90, Bei88, Kre88], but in this paper we will restrict our attention to bounded query language classes. The class $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ has also been studied in the context of sparse sets [Kad89]. By definition, the bounded query language class $\mathrm{P}^{A\|}$ is exactly the set of languages truth-table reducible to A, (see Ladner, Lynch and Selman for definitions [LLS75]). In fact, all of the bounded parallel classes defined above correspond to various notions of truth-table reducibilities. However, in order to keep the notation of the parallel and serial query classes consistent, we will use the parallel query terminology instead of truth-table reducibilities.

Much work has been done to show that the Boolean Hierarchy, the parallel query hierarchy and the serial query hierarchy are intertwined. We list some of the theorems below and refer the reader to the literature for the proofs [Bei91, Hem89, KSW87].

Fact

1. $\mathrm{P}^{\mathrm{SAT}\|[k]} \subseteq \mathrm{BH}_{k+1} \cap \text{co-}\mathrm{BH}_{k+1}$.
2. $\mathrm{BH}_k \cup \text{co-}\mathrm{BH}_k \subseteq \mathrm{P}^{\mathrm{SAT}\|[k]}$.
3. $\mathrm{P}^{\mathrm{SAT}[k]} = \mathrm{P}^{\mathrm{SAT}\|[2^k-1]}$.
4. $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} = \mathrm{P}^{\mathrm{SAT}\|}$.

3 Languages Which Do

In this section, we present some familiar languages which are known to have AND_ω and OR_ω.

Lemma 2

1. SAT has AND_ω and OR_ω.
2. $\mathrm{SAT} \wedge \overline{\mathrm{SAT}}$ has AND_ω.

Proof:

1. Given n formulas $f_1, \ldots, f_n$,

$$\langle f_1, \ldots, f_n \rangle \in \mathrm{AND}_\omega(\mathrm{SAT}) \iff f_1 \wedge \ldots \wedge f_n \in \mathrm{SAT}$$
$$\langle f_1, \ldots, f_n \rangle \in \mathrm{OR}_\omega(\mathrm{SAT}) \iff f_1 \vee \ldots \vee f_n \in \mathrm{SAT}.$$

2. Given n pairs $(f_1, g_1), \ldots, (f_n, g_n)$,

$$\langle (f_1, g_1), \ldots, (f_n, g_n) \rangle \in \mathrm{AND}_\omega(\mathrm{SAT} \wedge \overline{\mathrm{SAT}})$$
$$\iff (f_1 \wedge \ldots \wedge f_n, g_1 \vee \ldots \vee g_n) \in \mathrm{SAT} \wedge \overline{\mathrm{SAT}}.$$

□

Lemma 2 also implies that all NP complete languages have AND_ω and OR_ω. In fact any language that is $\leq_{\text{m}}^{\text{P}}$-complete for any relativized version of NP, P, or PSPACE has AND_ω and OR_ω. In addition, all languages in P also have AND_ω and OR_ω. One question is whether any of the incomplete languages in $\text{NP} - \text{P}$ have these Boolean functions or not. In our next theorem, we show that Graph Isomorphism, a natural language that is probably not $\leq_{\text{m}}^{\text{P}}$-complete for NP [Sch88], does have AND_ω and OR_ω. One open question is whether Primes has these Boolean functions.

Definition $\text{GI} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$.

The Labelled Graph Isomorphism problem is the problem of recognizing whether two graphs with labelled vertices are isomorphic by an isomorphism that preserves the labels.

Definition

$$\text{LGI} = \{\ \langle G, H\rangle \mid \begin{array}{l} G \text{ and } H \text{ are isomorphic graphs with labelled} \\ \text{nodes, and the isomorphism preserves labels } \}. \end{array}$$

We will show that LGI has AND_ω and OR_ω functions. The existence of AND_ω and OR_ω functions for GI follows from the fact that $\text{LGI} \equiv_{\text{m}}^{\text{P}} \text{GI}$ [Hof79].

Lemma 3 LGI has AND_ω and OR_ω.

Proof: Without loss of generality we can assume that graphs are represented as adjacency matrices paired with a table mapping vertices to integer labels.

We define an AND_ω function for LGI as follows. Given r pairs of graphs $\langle (G_1, H_1), \ldots, (G_r, H_r)\rangle$, we preprocess each G_i and H_i by:

1. for each G_i and H_i, add a new vertex and make it adjacent to every old vertex of the original graph.

2. define r new labels. For each i, label the new vertex of G_i and H_i with new label i.

Then let G be the disjoint union of all the altered G_i's (i.e. put them all together in one graph with no extra edges), and let H be the disjoint union of all the altered H_i's.

If for all i, the original G_i is isomorphic to the original H_i by a label preserving mapping, then G is isomorphic to H by mapping each G_i to H_i and the new vertex of G_i to the new vertex of H_i. Clearly, this is an isomorphism from G to H that preserves the labelling. If G is isomorphic (label preserving) to H, then the isomorphism must map the unique vertex in G with new label i (the new vertex added to G_i) to the unique vertex in H with new label i (the new vertex added to H_i). This induces an isomorphism between G_i and H_i (for all i).

To show that LGI has OR_ω, we first show that LGI has OR_2. Then, we note that the reduction can be done in linear time, which implies that LGI has OR_ω.

Given 2 pairs of labelled graphs, (G_1, H_1) and (G_2, H_2), we preprocess the graphs as described above (adding 2 new labels). Define a new labelled graph G containing all four graphs G_1, H_1, G_2, and H_2 as subgraphs with 2 new edges added connecting the new vertices of G_1 and G_2 and the new vertices of H_1 and H_2. H is produced similarly except the new edges connect G_1 with H_2 and H_1 with G_2 (see Figure 1).

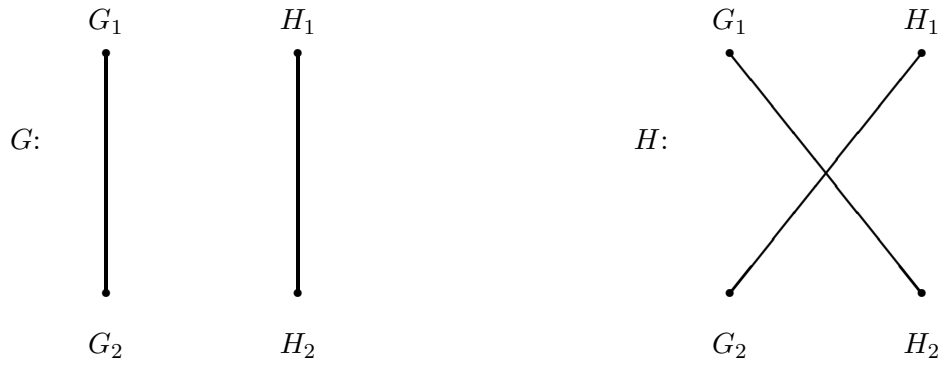

Figure 1: New labelled graphs G and H.

Suppose G_1 and H_1 are isomorphic by a label preserving mapping. Then, G and H are isomorphic by mapping G_1 in G to H_1 in H, H_1 to G_1, G_2 to G_2 and H_2 to H_2. Symmetrically, if G_2 and H_2 are isomorphic, then G and H are also isomorphic.

If G and H are isomorphic and G_1 is not isomorphic to H_1, then the new vertex of G_1 in G must be mapped to the new vertex of G_1 in H. Thus, the new vertex of G_2 in G must be mapped to the new vertex of H_2 in H. This induces an isomorphism between G_2 and H_2.

To see that the reduction from OR_2(LGI) to LGI is linear time, note that we only doubled the size of the input and added only 2 new labels. Lemma 1 then implies that LGI has OR_ω.

□

Corollary 4 GI has AND_ω and OR_ω.

4 Characterizations of Complete Languages

In this section, we show that the complete languages for D^P and $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ can be characterized using AND_2 and OR_ω. We show that the two characterizations are very similar. The only difference in the characterization is that a $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ complete set must have OR_ω, but a D^P complete set need not. Since $\mathrm{D}^\mathrm{P} \neq \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ unless PH collapses, D^P complete sets probably do not have OR_ω.

Lemma 5 Suppose that A has AND_2 and both SAT and $\overline{\mathrm{SAT}}$ $\leq^{\mathrm{P}}_{\mathrm{m}}$-reduce to A, then A is D^{P} hard under $\leq^{\mathrm{P}}_{\mathrm{m}}$ reductions.

Proof: Since $\mathrm{SAT} \wedge \overline{\mathrm{SAT}}$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for D^{P} and $\mathrm{SAT} \wedge \overline{\mathrm{SAT}} \leq^{\mathrm{P}}_{\mathrm{m}} A$, A must be D^{P} hard. □

Corollary 6 A set A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for D^{P} if and only if all of the following hold:

1. $A \in \mathrm{D}^{\mathrm{P}}$.
2. $\mathrm{SAT} \leq^{\mathrm{P}}_{\mathrm{m}} A$ and $\overline{\mathrm{SAT}} \leq^{\mathrm{P}}_{\mathrm{m}} A$.
3. A has AND_2.

Proof: If A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for D^{P} then $\mathrm{SAT} \wedge \overline{\mathrm{SAT}} \equiv^{\mathrm{P}}_{\mathrm{m}} A$. Since $\mathrm{SAT} \wedge \overline{\mathrm{SAT}}$ has all of the properties listed above, so must A. Conversely, if A has all of these properties, then $\mathrm{SAT} \wedge \overline{\mathrm{SAT}} \leq^{\mathrm{P}}_{\mathrm{m}} A$ and $A \in \mathrm{D}^{\mathrm{P}}$. Hence, A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for D^{P}. □

Now, we show that if a D^{P} hard set has OR_2 or OR_ω, then it is also hard for higher levels of the parallel query hierarchy.

Theorem 7 Let A be any set that is $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard for D^{P}.

1. If A has OR_2, then for all k, A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard for $\mathrm{P}^{\mathrm{SAT}[k]}$.
2. If A has OR_ω, then A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard for $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$.

Proof: Let C be any set in $\mathrm{P}^{\mathrm{SAT}[k]}$. To determine if $x \in C$, consider the query tree of the $\mathrm{P}^{\mathrm{SAT}[k]}$ computation. (The query tree is a full binary tree where the internal nodes are labelled by the oracle queries. The two subtrees below the node represent the computations that follow oracle reply. One branch assumes the oracle replied yes, the other no.) The query tree has height k and 2^k leaves. Only one path in the tree (from root to leaf) is the correct path, and $x \in C$ if and only if this path ends in an accepting configuration.

Now we show that a $\mathrm{SAT} \wedge \overline{\mathrm{SAT}}$ computation can determine if a given path is the correct path. Let $p_1, \ldots, p_i$ be the queries on the path assumed to be answered yes, and $q_1, \ldots, q_j$ be the queries assumed to be answered no. Then, the path is correct if and only if $p_1 \wedge \cdots \wedge p_i \in \mathrm{SAT}$ and $q_1 \vee \cdots \vee q_j \in \overline{\mathrm{SAT}}$; that is, if only if

$$(p_1 \wedge \cdots \wedge p_j, q_1 \vee \cdots \vee q_j) \in \mathrm{SAT} \wedge \overline{\mathrm{SAT}}.$$

Since the query tree is of constant depth, it is possible to generate the entire tree in polynomial time and write down all the paths that end in an accepting configuration. Note that $x \in C$ if and only if one of these accepting paths is the correct path. Let r be the number of accepting paths.

As described above, we can determine if any given path is the correct path using a $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ computation. That is, for the i^{th} accepting path P_i, we can construct two Boolean formulas, F_i and G_i such that

$$P_i \text{ is the correct path } \iff (F_i, G_i) \in \mathrm{SAT}\wedge\overline{\mathrm{SAT}}.$$

Now, to determine if at least one of the r accepting paths is the correct path, we can use an $\mathrm{OR}_r(\mathrm{SAT}\wedge\overline{\mathrm{SAT}})$ computation. Thus,

$$x \in C \iff \langle (F_1, G_1), \ldots, (F_r, G_r)\rangle \in \mathrm{OR}_r(\mathrm{SAT}\wedge\overline{\mathrm{SAT}}).$$

Since r is bounded by 2^k (a constant), we can reduce the language C to $\mathrm{OR}_{2^k}(\mathrm{SAT}\wedge\overline{\mathrm{SAT}})$ using the reduction described above. Finally, since we are assuming that A is D^{P} hard and has OR_2, we know that $\mathrm{OR}_{2^k}(\mathrm{SAT}\wedge\overline{\mathrm{SAT}}) \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{OR}_{2^k}(A)$ and that A has OR_{2^k}. Hence,

$$C \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{OR}_{2^k}(\mathrm{SAT}\wedge\overline{\mathrm{SAT}}) \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{OR}_{2^k}(A) \leq^{\mathrm{P}}_{\mathrm{m}} A.$$

Thus, A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard for $\mathrm{P}^{\mathrm{SAT}[k]}$.

The proof of the second case is similar. The only difference is that the query tree is polynomial in size instead of constant. □

Corollary 8 Suppose that A has AND_2 and OR_ω and both SAT and $\overline{\mathrm{SAT}}$ $\leq^{\mathrm{P}}_{\mathrm{m}}$-reduce to A, then A is $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ hard under $\leq^{\mathrm{P}}_{\mathrm{m}}$ reductions.

Theorem 9 A set A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ if and only if all of the following hold:

1. $A \in \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$.
2. $\mathrm{SAT} \leq^{\mathrm{P}}_{\mathrm{m}} A$ and $\overline{\mathrm{SAT}}$ $\leq^{\mathrm{P}}_{\mathrm{m}} A$.
3. A has AND_2.
4. A has OR_ω.

Proof: If a set A has all of the properties listed above, then A is $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ hard by Corollary 8 and $A \in \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. Hence, A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. Conversely, suppose that A is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. Clearly, $\mathrm{SAT} \leq^{\mathrm{P}}_{\mathrm{m}} A$ and $\overline{\mathrm{SAT}} \leq^{\mathrm{P}}_{\mathrm{m}} A$. To show that A has AND_2 simply observe that $\mathrm{AND}_2(A)$ must also be contained in $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. Showing that A has OR_ω is trickier. We need to cite the fact that $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} = \mathrm{P}^{\mathrm{SAT}\|}$ [Hem89]. Once we know that $A \in \mathrm{P}^{\mathrm{SAT}\|}$, it is not difficult to construct a $\mathrm{P}^{\mathrm{SAT}\|}$ machine to recognize $\mathrm{OR}_\omega(A)$. Then, $\mathrm{OR}_\omega(A) \in \mathrm{P}^{\mathrm{SAT}\|} = \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ and $\mathrm{OR}_\omega(A) \leq^{\mathrm{P}}_{\mathrm{m}} A$ by completeness of A. □

Notice that the only difference in the characterizations of D^{P} complete sets and $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ complete sets (Corollary 6 and Theorem 9) is that $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ complete sets must have OR_ω. Thus, D^{P} complete sets cannot have OR_ω unless $\mathrm{D}^{\mathrm{P}} \subseteq \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. In fact, we have the following corollary.

Corollary 10 $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ has $\mathrm{OR}_2 \iff \mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ has $\mathrm{OR}_\omega \iff \mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$.

Proof: First, note that $\mathrm{OR}_\omega(\mathrm{SAT}\wedge\overline{\mathrm{SAT}}) \in \mathrm{P}^{\mathrm{SAT}\|} = \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. Now, if $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$, then $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ would be $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$, so $\mathrm{OR}_\omega(\mathrm{SAT}\wedge\overline{\mathrm{SAT}}) \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{SAT}\wedge\overline{\mathrm{SAT}}$. Thus, $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ would have OR_ω. Also, if $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ has OR_ω, then it certainly has OR_2. Finally, we need to show that if $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ has OR_2, then $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$. So suppose that $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ has OR_2. Then by Theorem 7, $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ is hard for $\mathrm{P}^{\mathrm{SAT}[2]}$. However, the complement of $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ is $\overline{\mathrm{SAT}}\vee\mathrm{SAT}$ which is in $\mathrm{P}^{\mathrm{SAT}[2]}$. So, $\mathrm{SAT}\wedge\overline{\mathrm{SAT}} \equiv^{\mathrm{P}}_{\mathrm{m}} \overline{\mathrm{SAT}}\vee\mathrm{SAT}$. Since $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ has AND_ω (Lemma 2), $\overline{\mathrm{SAT}}\vee\mathrm{SAT}$ has OR_ω by DeMorgan's Law. Thus, $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ must also have OR_ω since the two sets are $\leq^{\mathrm{P}}_{\mathrm{m}}$-equivalent. Therefore, $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$ by Theorem 9 and $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$. □

Since $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$ implies that PH collapses to just above Σ^{P}_2 [CK90], Corollary 10 is evidence that $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ does not have OR_2 or OR_ω.

Corollary 11 If $\mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}}$ then PH collapses and $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$.

Proof: If $\mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}}$, then $\overline{\mathrm{SAT}}\vee\mathrm{SAT}$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for D^{P}. So, $\overline{\mathrm{SAT}}\vee\mathrm{SAT}$ has AND_2 by Corollary 6. Then, by DeMorgan's Law, $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ has OR_2 which implies $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$ by Theorem 10. The polynomial hierarchy then collapses by Kadin's proof [Kad88, CK90]. □

It was observed by Kadin [Kad89] that the collapse of the Boolean hierarchy at level 2 ($\mathrm{D}^{\mathrm{P}} = \text{co-}\mathrm{D}^{\mathrm{P}}$) immediately implies $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{D}^{\mathrm{P}}$. However, we cannot push the same theorem through for the collapse of the Boolean hierarchy at levels 3 or higher. We explain this phenomenon using AND_2 and OR_2. Observe that in the proof above we relied on the fact that $\mathrm{SAT}\wedge\overline{\mathrm{SAT}}$ has AND_2. We will show in the next section that the complete languages for the higher levels of the Boolean hierarchy cannot have AND_2 or OR_2, unless PH collapses.

5 Languages Which Don't

In this section, we show that the complete languages for the higher levels of the Boolean hierarchy and query hierarchies probably do not have AND_2 or OR_2. In the following theorems, keep in mind that $\mathrm{BH}_k \subseteq \mathrm{BH} \subseteq \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. Also recall that if $\mathrm{BH} \subseteq \mathrm{BH}_k$, $\mathrm{QH}_\| \subseteq \mathrm{P}^{\mathrm{SAT}\|[k]}$, $\mathrm{QH} \subseteq \mathrm{P}^{\mathrm{SAT}[k]}$, or $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{BH}_k$, then the polynomial hierarchy collapses [Kad88, CK90]. Of course Theorems 12 and 13 apply to any $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete sets for the various levels of the BH.

Theorem 12 For $k \geq 2$,

1. BL_k has $\mathrm{OR}_2 \iff \mathrm{BH} \subseteq \mathrm{BH}_k$.
2. BL_k has $\mathrm{OR}_\omega \iff \mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{BH}_k$.

Proof:

1. ($\Rightarrow$) For $k \geq 2$, BL_k is D^{P} hard. By Theorem 7, If BL_k has OR_2, then it is hard for $\mathrm{P}^{\mathrm{SAT}[k]}$. However, co-$\mathrm{BL}_k \in \mathrm{P}^{\mathrm{SAT}[k]}$, so co-$\mathrm{BL}_k \leq^{\mathrm{P}}_{\mathrm{m}} \mathrm{BL}_k$. Thus, BH_k is closed under complementation and $\mathrm{BH} \subseteq \mathrm{BH}_k$.

   ($\Leftarrow$) If $\mathrm{BH} \subseteq \mathrm{BH}_k$ then $\mathrm{P}^{\mathrm{SAT}\|[2k]} \subseteq \mathrm{BH}_k$, since $\mathrm{P}^{\mathrm{SAT}\|[2k]} \subseteq \mathrm{BH}_{2k+1}$. Clearly, $\mathrm{OR}_2(\mathrm{BL}_k) \in \mathrm{P}^{\mathrm{SAT}\|[2k]}$, so BL_k has OR_2.

2. ($\Rightarrow$) As above, BL_k is D^{P} hard. By Theorem 9, if BL_k has OR_ω, then BL_k is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. So, $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{BH}_k$.

   ($\Leftarrow$) $\mathrm{OR}_\omega(\mathrm{BL}_k) \in \mathrm{P}^{\mathrm{SAT}\|} = \mathrm{P}^{\mathrm{SAT}[O(\log n)]}$. So, if $\mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{BH}_k$ then BL_k has OR_ω. □

Theorem 13 For $k \geq 3$,

1. BL_k has $\mathrm{AND}_2 \iff \mathrm{BH} \subseteq \mathrm{BH}_k$.
2. BL_k has $\mathrm{AND}_\omega \iff \mathrm{P}^{\mathrm{SAT}[O(\log n)]} \subseteq \mathrm{BH}_k$.

Proof: DeMorgan's Law implies that co-BL_k has OR_2 or OR_ω. Then, the proof proceeds as above, because for $k \geq 3$, co-BL_k is D^{P} hard. □

If the polynomial hierarchy does not collapse, then the preceding theorems tell us that as we go up the Boolean hierarchy, the complete languages lose the Boolean connective functions. At the first level, $\mathrm{BL}_1 = \mathrm{SAT}$, so BL_1 has both AND_ω and OR_ω. At the second level, BL_2 is D^{P} complete, so BL_2 has AND_ω but does not have OR_2. From the third level up, BL_k has neither AND_2 nor OR_2. In conclusion, we see that when we walk up the Boolean hierarchy from level one, the complete languages lose pieces of "robustness".

The $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete languages for the different levels of query hierarchy and the parallel query hierarchy (over SAT) also do not have AND_2 or OR_2 unless the PH collapses.

Theorem 14 For any k, if C is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{\mathrm{SAT}\|[k]}$, then

C has AND_2 or $\mathrm{OR}_2 \Longrightarrow \mathrm{BH} \subseteq \mathrm{P}^{\mathrm{SAT}\|[k]}$.

Proof: First, note that since $\mathrm{P}^{\mathrm{SAT}\|[k]}$ is closed under complementation, $\overline{C} \equiv^{\mathrm{P}}_{\mathrm{m}} C$. Therefore C has AND_2 if and only if C has OR_2. We will show that if C has either Boolean function, then $\mathrm{BH}_{k+1} \subseteq \mathrm{P}^{\mathrm{SAT}\|[k]}$ which implies the collapse of the BH (and the PH).

Recall that if k is odd, then the $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete language for BH_{k+1} is

$$\mathrm{BL}_{k+1} = \{\ \langle x_1, \ldots, x_{k+1} \rangle \mid \langle x_1, \ldots, x_k \rangle \in \mathrm{BL}_k \text{ and } x_{k+1} \in \overline{\mathrm{SAT}}\ \}.$$

If C has AND_2, then $\mathrm{BL}_{k+1} \leq^{\mathrm{P}}_{\mathrm{m}} C$ since $\mathrm{BL}_k \leq^{\mathrm{P}}_{\mathrm{m}} C$ and $\overline{\mathrm{SAT}} \leq^{\mathrm{P}}_{\mathrm{m}} C$. But this implies $\mathrm{BH}_{k+1} \subseteq \mathrm{P}^{\mathrm{SAT}\|[k]}$.

If k is even, then the $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete language for BH_{k+1} is

$$\mathrm{BL}_{k+1} = \{\ \langle x_1, \ldots, x_{k+1}\rangle \mid \langle x_1, \ldots, x_k\rangle \in \mathrm{BL}_k \text{ or } x_{k+1} \in \mathrm{SAT}\ \}.$$

If C has OR_2, then $\mathrm{BL}_{k+1} \leq^{\mathrm{P}}_{\mathrm{m}} C$ since $\mathrm{BL}_k \leq^{\mathrm{P}}_{\mathrm{m}} C$ and $\mathrm{SAT} \leq^{\mathrm{P}}_{\mathrm{m}} C$. Again this implies $\mathrm{BH}_{k+1} \subseteq \mathrm{P}^{\mathrm{SAT}\|[k]}$. □

In related research, we have shown that USAT, the set of Boolean formulas that have exactly one satisfying assignment, does not have OR_ω unless the PH collapses [CKR91].

6 AND_2 and OR_2 and Hierarchies

Considerable research has focused on understanding the structure of the Boolean hierarchy and the query hierarchies [WW85, Bei91, CGH$^+$88, KSW87]. The basic structural properties of these hierarchies are as follows:

1. *Normal forms for* BH: there are many different but equivalent ways to define the levels of the BH. For example, BH_{2k} can be defined as any of the following [CH85, CGH$^+$88]:
   (a) $\{\ (\cdots(((L_1 \cap \overline{L_2}) \cup L_3) \cap \overline{L_4})\cdots) \cap \overline{L_{2k}} \mid L_i \in \mathrm{NP}\ \}$.
   (b) $\{\ (\cdots(((L_1 \cap \overline{L_2}) \cup L_3) \cap \overline{L_4})\cdots) \cap \overline{L_{2k}} \mid L_i \in \mathrm{NP},\ L_1 \supseteq L_2 \supseteq L_3 \supseteq \ldots \supseteq L_{2k}\ \}$.
   (c) $\{\ L_1 - (L_2 - (L_3 - \cdots - (L_{2k-1} - L_{2k})\cdots)) \mid L_i \in \mathrm{NP}\ \}$.
   (d) $\{\ L_1 - (L_2 - (L_3 - \cdots - (L_{2k-1} - L_{2k})\cdots)) \mid L_i \in \mathrm{NP},\ L_1 \supseteq L_2 \supseteq L_3 \supseteq \ldots \supseteq L_{2k}\ \}$.
   (e) $\{\ (L_1 \cap \overline{L_2}) \cup (L_3 \cap \overline{L_4}) \cup \cdots \cup (L_{2k-1} \cap \overline{L_{2k}}) \mid L_i \in \mathrm{NP}\ \}$.
   (f) $\{\ (L_1 \cap \overline{L_2}) \cup (L_3 \cap \overline{L_4}) \cup \cdots \cup (L_{2k-1} \cap \overline{L_{2k}}) \mid L_i \in \mathrm{NP},\ L_1 \supseteq L_2 \supseteq L_3 \supseteq \ldots \supseteq L_{2k}\ \}$.
2. *Complete languages*:
   (a) BL_k is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for BH_k.
   (b) $\mathrm{ODD}_k(\mathrm{SAT})$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for BH_k [Bei91, WW85]. ($\mathrm{ODD}_k(\mathrm{SAT})$ is defined below.)
   (c) $\mathrm{EVEN}_k(\mathrm{SAT}) \oplus \mathrm{ODD}_k(\mathrm{SAT})$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{\mathrm{SAT}\|[k]}$ [Bei91, WW85].
3. *Basic containments and intertwining* [Bei91, KSW87]:
   $$\mathrm{BH}_k \cup \text{co-BH}_k \subseteq \mathrm{P}^{\mathrm{SAT}\|[k]} \subseteq \mathrm{BH}_{k+1} \cap \text{co-BH}_{k+1}.$$
4. *Intertwining of the query hierarchies* [Bei91]:
   $$\mathrm{P}^{\mathrm{SAT}[k]} = \mathrm{P}^{\mathrm{SAT}\|[2^k-1]}.$$

5. *Upward collapse*:

$$\begin{aligned} \mathrm{BH}_k = \text{co-}\mathrm{BH}_k &\implies \mathrm{BH} \subseteq \mathrm{BH}_k \\ \mathrm{P}^{\mathrm{SAT}\|[k]} = \mathrm{P}^{\mathrm{SAT}\|[k+1]} &\implies \mathrm{QH}_{\|} \subseteq \mathrm{P}^{\mathrm{SAT}\|[k]} \\ \mathrm{P}^{\mathrm{SAT}[k]} = \mathrm{P}^{\mathrm{SAT}[k+1]} &\implies \mathrm{QH} \subseteq \mathrm{P}^{\mathrm{SAT}[k]}. \end{aligned}$$

Definition For any set A, we define the following languages:

$$\begin{aligned} \mathrm{EVEN}_k(A) &= \{\ \langle x_1, \ldots, x_k \rangle \mid \ \|\{\ x_1, \ldots, x_k\ \} \cap A\| \ \text{ is even }\}, \\ \mathrm{ODD}_k(A) &= \{\ \langle x_1, \ldots, x_k \rangle \mid \ \|\{\ x_1, \ldots, x_k\ \} \cap A\| \ \text{ is odd }\}, \\ \mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A) &= \{\ \langle 0, x_1, \ldots, x_k \rangle \mid \langle x_1, \ldots, x_k \rangle \in \mathrm{EVEN}_k(A)\ \} \\ &\quad \cup \{\ \langle 1, x_1, \ldots, x_k \rangle \mid \langle x_1, \ldots, x_k \rangle \in \mathrm{ODD}_k(A)\ \}. \end{aligned}$$

In this section, we show that all the above properties of SAT follow simply from the fact that SAT (or any NP complete set) has AND_2 and OR_2 (Lemma 2). That is, they do not depend on the fact that SAT is NP complete or even in NP at all. This observation follows from the fact that given any set A that has AND_2 and OR_2, the Boolean and query hierarchies based on A have all the above properties. The results of this section have been proven independently by Bertoni, Bruschi, Joseph, Sitharam, and Young [BBJ$^+$89].

The QH and $\mathrm{QH}_{\|}$ over A were defined in section 2. We define the Boolean hierarchy over A as the difference hierarchy over m-1(A), set of all languages $\leq^{\mathrm{P}}_{\mathrm{m}}$-reducible to A.

Definition

$$\begin{aligned} \mathrm{BH}_1(A) &= \text{m-1}(A), \\ \mathrm{BH}_{2k}(A) &= \{\ L_1 \cap \overline{L_2} \mid L_1 \in \mathrm{BH}_{2k-1}(A) \text{ and } L_2 \in \text{m-1}(A)\ \}, \\ \mathrm{BH}_{2k+1}(A) &= \{\ L_1 \cup L_2 \mid L_1 \in \mathrm{BH}_{2k}(A) \text{ and } L_2 \in \text{m-1}(A)\ \}, \\ \text{co-}\mathrm{BH}_k(A) &= \{\ L \mid \overline{L} \in \mathrm{BH}_k(A)\ \}, \\ \mathrm{BH}(A) &= \bigcup_{k=1}^{\infty} \mathrm{BH}_k(A). \end{aligned}$$

First, note that if $A = \mathrm{SAT}$, then $\mathrm{BH}_k(A) = \mathrm{BH}_k$. Also, we will leave it to the reader to show that if A has AND_2 and OR_2, then the various normal forms defined in [CGH$^+$88] hold for the different levels of $\mathrm{BH}(A)$ (see [CH85]). For example, a language L is in $\mathrm{BH}_k(A)$ iff there exist languages $L_1, \ldots, L_k \in \text{m-1}(A)$ such that

$$L = D(L_1, \ldots, L_k) \stackrel{\text{def}}{=} L_1 - (L_2 - (L_3 - (\cdots - L_k))).$$

Theorem 15 If A has AND_2 and OR_2, then $\mathrm{ODD}_k(A)$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{BH}_k(A)$.

Proof: First, we show that $\mathrm{ODD}_k(A)$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard for $\mathrm{BH}_k(A)$. If $L \in \mathrm{BH}_k(A)$, then $L = D(L_1, \ldots, L_k)$ for $L_1, \ldots, L_k \in \text{m-1}(A)$. Define $L'_i \stackrel{\mathrm{def}}{=} \bigcap_{j \leq i} L_j$. Then each $L'_i \in \text{m-1}(A)$, since m-1(A) is closed under intersection, and $L'_1 \supseteq L'_2 \supseteq \cdots \supseteq L'_k$. One can prove by induction that $D(L_1, \ldots, L_k) = D(L'_1, \ldots, L'_k)$. Then clearly, $x \in L$ iff the number of sets $L'_1, \ldots, L'_k$ that contain x is odd. Since each L'_i is in m-1(A), given a string x, we can reduce x to $\langle q_1, \ldots, q_k \rangle$ such that $x \in L'_i \iff q_i \in A$. Then $x \in L \iff \langle q_1, \ldots, q_k \rangle \in \mathrm{ODD}_k(A)$. Therefore $\mathrm{ODD}_k(A)$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard for $\mathrm{BH}_k(A)$.

To see that $\mathrm{ODD}_k(A) \in \mathrm{BH}_k(A)$, for each $i \leq k$, consider the set

$$L_i \stackrel{\mathrm{def}}{=} \{\ \langle x_1, \ldots, x_k \rangle \mid \ \| \{\ x_1, \ldots, x_k\ \} \cap A \| \geq i\ \}.$$

$L_i \in \text{m-1}(A)$ since given $\langle x_1, \ldots, x_k \rangle$, we can generate all $s = \binom{k}{i}$ subsets of i strings and use the reduction from $\mathrm{OR}_s(\mathrm{AND}_i(A))$ to A to generate a string that is in A iff all i strings in one of the subsets are in A. Then $\mathrm{ODD}_k(A) = D(L_1, \ldots, L_k)$, implying $\mathrm{ODD}_k(A) \in \mathrm{BH}_k(A)$. □

The most difficult proof in this section is the argument that $\mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A)$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{A\|[k]}$. We prove this by showing that the mind-change technique [Bei91, WW85] works provided A has AND_2 and OR_2.

Theorem 16 If A has AND_2 and OR_2, then $\mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A)$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{A\|[k]}$.

Proof: Both $\mathrm{EVEN}_k(A)$ and $\mathrm{ODD}_k(A)$ are obviously in $\mathrm{P}^{A\|[k]}$, so we need to show that every set in $\mathrm{P}^{A\|[k]}$ can be reduced to $\mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A)$.

Let L be a language in $\mathrm{P}^{A\|[k]}$ computed by a polynomial time oracle Turing machine M which asks at most k parallel queries. With each $M^A(x)$ computation, we associate *truth tables* that have the following syntax:

q_1	q_2	$\ldots$	q_k	result
0	0	$\ldots$	0	accept
0	0	$\ldots$	1	reject
$\vdots$	$\vdots$		$\vdots$	$\vdots$
1	1	$\ldots$	1	reject

The idea is that $q_1, \ldots, q_k$ are the queries that $M^A(x)$ makes. Each row of the truth table records whether the computation accepts or rejects assuming that the answer to the queries are as listed in the row. (A "1" in column q_i means we are assuming that $q_i \in A$ and a "0", $q_i \notin A$). The full truth table has 2^k lines, but consider truth tables with fewer lines. In particular, we call a truth table for $M^A(x)$ *valid* if

1. The strings $q_1, \ldots, q_k$ are the queries made by $M^A(x)$.

2. The first row of the truth table is all 0's.

3. If a "1" appears in column q_i of row j, then for all rows below row j a "1" appears in column q_i. (Think of each row as the set of q_i's assumed to be in A, represented as a bit vector. This condition implies that the rows are monotonic under the set containment relation.)

4. If a "1" appears in column q_i of any row, then q_i is in fact an element of A.

N.B. valid truth tables have between 1 and $k+1$ rows, so the full truth table is not a valid truth table. The following is an example of a valid truth table.

q_1	q_2	q_3	q_4	q_5	q_6	q_7	result
0	0	0	0	0	0	0	reject
0	0	1	0	0	0	0	accept
0	0	1	0	1	0	0	accept
0	0	1	0	1	1	0	reject
0	0	1	0	1	1	0	accept
1	0	1	0	1	1	0	reject

For each valid truth table T, we associate a number m_T—the mind changes of T—which is the number of times the result column changes from accept to reject or from reject to accept. In the example above, m_T is 4. Since valid truth tables have between 1 and $k+1$ rows, $0 \leq m_T \leq k$.

Now we define the set of valid truth tables labelled with the number of mind changes.

$$\mathcal{T} = \{\ \langle T, x, s\rangle \mid T \text{ is a valid truth table for } M^A(x) \text{ and } m_T = s\ \}.$$

Claim: $\mathcal{T} \leq_{\mathrm{m}}^{\mathrm{P}} A$. On input $\langle T, x, s\rangle$, a polynomial time machine can simulate $M^A(x)$ to determine which strings, $q_1, \ldots, q_k$, will be queried. Then, the machine can easily check the syntax of T to see if it meets conditions 2 and 3 in the definition of a valid truth table and to see if the number of mind changes is indeed equal to s. If any of these conditions is not met, $\langle T, x, s\rangle$ is reduced to a known string in $\overline{A}$. Otherwise, let $q_{i_1}, q_{i_2}, \ldots, q_{i_t}$ be the queries which have a 1 in the last row of T. Then, T is a valid truth table if and only if each $q_{i_j} \in A$. That is,

$$\langle T, x, s\rangle \in \mathcal{T} \iff \langle q_{i_1}, \ldots, q_{i_t}\rangle \in \mathrm{AND}_t(A)$$

Since $t \leq k$, we can use $\mathrm{AND}_k(A)$ instead of $\mathrm{AND}_t(A)$ (by adding the appropriate number of trivial strings known to be in A). Also, since k is a constant, $\mathrm{AND}_k(A) \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{AND}_2(A)$. Finally, since A has AND_2, we know that $\mathrm{AND}_2(A) \leq_{\mathrm{m}}^{\mathrm{P}} A$. Thus,

$$\mathcal{T} \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{AND}_k(A) \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{AND}_2(A) \leq_{\mathrm{m}}^{\mathrm{P}} A$$

and there is a $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduction from $\mathcal{T}$ to A.

Now define

$$\mathcal{ET}_r = \{\ x \mid \exists T, s \text{ such that } s \geq r \text{ and } \langle T, x, s\rangle \in \mathcal{T}\ \}.$$

Claim: $\mathcal{ET}_r \leq_{\mathrm{m}}^{\mathrm{P}} A$. Since k is constant, for fixed queries $q_1, \ldots, q_k$, there is only a constant number of truth tables with k columns. So simply generate all the tuples $\langle T, x, s\rangle$ such that T is a truth table with k columns and $r \leq s \leq k$. Let, $\langle T_1, x, s_1\rangle$, $\ldots$, $\langle T_\alpha, x, s_\alpha\rangle$ be the list of such tuples. Since $x \in \mathcal{ET}_r$ if and only if $\exists i,\ 1 \leq i \leq \alpha,\ \langle T_i, x, s_i\rangle \in \mathcal{T}$,

$$x \in \mathcal{ET}_r \iff \langle\langle T_1, x, s_1\rangle, \ldots, \langle T_\alpha, x, s_\alpha\rangle\rangle \in \mathrm{OR}_\alpha(\mathcal{T}).$$

Note that since k is a constant, α is bounded by a constant $\beta = k2^{2^k}$. Thus, $\mathcal{ET}_r \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{OR}_\beta(\mathcal{T})$. However, $\mathcal{T} \leq_{\mathrm{m}}^{\mathrm{P}} A$, so $\mathrm{OR}_\beta(\mathcal{T}) \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{OR}_\beta(A)$. Again, since β is a constant and A has OR_2, we know that $\mathrm{OR}_\beta(A) \leq_{\mathrm{m}}^{\mathrm{P}} A$. Thus,

$$\mathcal{ET}_r \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{OR}_\beta(\mathcal{T}) \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{OR}_\beta(A) \leq_{\mathrm{m}}^{\mathrm{P}} A$$

and there is a $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduction from $\mathcal{ET}_r$ to A. Call this $\leq_{\mathrm{m}}^{\mathrm{P}}$-reduction g_r.

Now, we will use $g_1, \ldots, g_k$ to produce a reduction from L to $\mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A)$. This reduction will simply print out a k-tuple $\langle z_1, \ldots, z_k\rangle$, where $z_i = g_i(x)$, and an extra bit to indicate if the reduction is to $\mathrm{EVEN}_k(A)$ or $\mathrm{ODD}_k(A)$. Each z_i has the property that $z_i \in A$ iff $x \in \mathcal{ET}_i$ iff there exists a valid truth table that makes at least i mind changes. Let γ be the maximum number of mind changes. Then, $z_1, \ldots, z_\gamma \in A$ and $z_{\gamma+1}, \ldots, z_k \notin A$. So, the parity of the number of z_i's in A is the same as the parity of γ.

Now, we claim that the parity of the maximum number of mind changes is enough to determine if M^A accepted the string x. First, note that there must exist a valid truth table with the maximum number of mind changes which has, in its last row, the actual oracle replies. (I.e., there is a 1 in column q_i of the last row if and only if $q_i \in A$.) To see this, simply confirm that adding the row of actual oracle replies to the bottom of a valid truth table results in another valid truth table. Thus, $M^A(x)$ accepts iff the result in the last row of this truth table is "accept".

Second, consider this valid truth table that makes γ mind changes. Note that if γ is odd (even) then the result in the last row is the opposite of (same as) the result in the first row. Suppose the result in the first row is "accept", then $x \in L$ iff γ is even iff $\langle z_1, \ldots, z_k\rangle \in \mathrm{EVEN}_k(A)$. Similarly, if the result in the first row is "reject", then $x \in L$ iff $\langle z_1, \ldots, z_k\rangle \in \mathrm{ODD}_k(A)$. Let b be a bit that is 0 if the result of the first row is "accept" and 1 if the result is "reject". This bit b can be computed in polynomial time and

$$x \in L \iff \langle b, z_1, \ldots, z_k\rangle \in \mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A).$$

Therefore, $\mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A)$ is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{P}^{A\|[k]}$. □

Theorem 17 If A has AND_2 and OR_2, then

$$\mathrm{BH}_k(A) \cup \text{co-}\mathrm{BH}_k(A) \subseteq \mathrm{P}^{A\|[k]} \subseteq \mathrm{BH}_{k+1}(A) \cap \text{co-}\mathrm{BH}_{k+1}(A).$$

Proof: The first containment follows straight from the definitions of the various classes. The second containment holds because

$$\mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A) \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{ODD}_{k+1}(A) \quad \text{and} \quad \mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A) \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{EVEN}_{k+1}(A).$$

Then, the completeness results of the previous two theorems imply that $\mathrm{P}^{A\|[k]} \subseteq \mathrm{BH}_{k+1}(A)$ and $\mathrm{P}^{A\|[k]} \subseteq \text{co-}\mathrm{BH}_{k+1}(A)$. To see how $\mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A) \leq_{\mathrm{m}}^{\mathrm{P}} \mathrm{ODD}_{k+1}(A)$, let a_0 and a_1 be two fixed strings such that $a_0 \in A$ and $a_1 \notin A$. Then,

$$\langle x_1, \ldots, x_k \rangle \in \mathrm{EVEN}_k(A) \iff \langle a_0, x_1, \ldots, x_k \rangle \in \mathrm{ODD}_{k+1}(A).$$

Also,

$$\langle x_1, \ldots, x_k \rangle \in \mathrm{ODD}_k(A) \iff \langle a_1, x_1, \ldots, x_k \rangle \in \mathrm{ODD}_{k+1}(A).$$

Thus,

$$\langle b, x_1, \ldots, x_k \rangle \in \mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A) \iff \langle a_b, x_1, \ldots, x_k \rangle \in \mathrm{ODD}_{k+1}(A),$$

where b is the 0-1 bit according to the definition of $\mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A)$. Similarly,

$$\langle b, x_1, \ldots, x_k \rangle \in \mathrm{EVEN}_k(A) \oplus \mathrm{ODD}_k(A) \iff \langle a_{1-b}, x_1, \ldots, x_k \rangle \in \mathrm{EVEN}_{k+1}(A).$$

□

Theorem 18 If a set A has AND_2 and OR_2, then $\mathrm{P}^{A[k]} = \mathrm{P}^{A\|[2^k-1]}$.

Proof: For any A, $\mathrm{P}^{A[k]} \subseteq \mathrm{P}^{A\|[2^k-1]}$, so all we need to show is $\mathrm{P}^{A\|[2^k-1]} \subseteq \mathrm{P}^{A[k]}$. We will do this by showing that $\mathrm{ODD}_{2^k-1}(A) \in \mathrm{P}^{A[k]}$ and then Theorem 16 implies that $\mathrm{P}^{A\|[2^k-1]} \subseteq \mathrm{P}^{A[k]}$.

The main idea of this proof is to use k serial queries to do binary search over $2^k - 1$ strings to determine how many are elements of A. To determine if at least r strings are in A, generate all $s = \binom{2^k-1}{r}$ subsets of the queries with r elements. Then, use the reduction from $\mathrm{OR}_s(\mathrm{AND}_r(A))$ to A to determine if one of the subsets contains only strings in A. If so, at least r of the query strings are elements of A. Since it takes exactly k steps of binary search to search over $2^k - 1$ elements, $\mathrm{P}^{A[k]} = \mathrm{P}^{A\|[2^k-1]}$. □

In [Cha89], Chang erroneously stated that for all A, if $\mathrm{P}^{A\|[k]} = \mathrm{P}^{A\|[k+1]}$, then the entire query hierarchy based on A collapses to $\mathrm{P}^{A\|[k]}$. This is only known to be true when A has AND_2 and OR_2.

Theorem 19 If A has AND_2 and OR_2 and $\mathrm{P}^{A\|[k]} = \mathrm{P}^{A\|[k+1]}$, then for all $j > k$, $\mathrm{P}^{A\|[k]} = \mathrm{P}^{A\|[j]}$.

Proof: It suffices to show that under these assumptions, $\mathrm{P}^{A\|[k+1]} = \mathrm{P}^{A\|[k+2]}$. Since A has AND_2 and OR_2, by Theorem 16, $\mathrm{EVEN}_{k+1}(A) \oplus \mathrm{ODD}_{k+1}(A)$ is $\leq_{\mathrm{m}}^{\mathrm{P}}$-complete for $\mathrm{P}^{A\|[k+1]}$. However, $\mathrm{P}^{A\|[k]} = \mathrm{P}^{A\|[k+1]}$, so there is some $\mathrm{P}^{A\|[k]}$ computation for $\mathrm{EVEN}_{k+1}(A) \oplus \mathrm{ODD}_{k+1}(A)$. Thus,

a $\mathrm{P}^{A\|[k+1]}$ computation can accept $\mathrm{EVEN}_{k+2}(A) \oplus \mathrm{ODD}_{k+2}(A)$ using k queries to compute the parity of the first $k+1$ strings and the last query to determine if the last string is in A. Since $\mathrm{EVEN}_{k+2}(A) \oplus \mathrm{ODD}_{k+2}(A)$ is $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete for $\mathrm{P}^{A\|[k+2]}$, $\mathrm{P}^{A\|[k+1]} = \mathrm{P}^{A\|[k+2]}$. □

The upward collapse properties of the $\mathrm{BH}(A)$ and $\mathrm{QH}(A)$ follow from Theorem 19.

7 Some Examples from Diagonalization

In the preceding sections, we have seen many natural languages which have various combinations of AND_2, AND_ω, OR_2 and OR_ω. In this section, we use diagonalization techniques to show that there exist languages which have any subset of these combining functions so long as they do not violate the implications:

$$A \text{ has } \mathrm{AND}_\omega \Longrightarrow A \text{ has } \mathrm{AND}_2, \quad \text{and} \quad A \text{ has } \mathrm{OR}_\omega \Longrightarrow A \text{ has } \mathrm{OR}_2.$$

At this point, we direct the reader's attention to some results on relativized Boolean Hierarchies. For any oracle A, let $\mathrm{NPC}(A)$ be the canonical $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete language for NP^A:

$$\mathrm{NPC}(A) \stackrel{\mathrm{def}}{=} \{\ \langle N_i, x, 1^r\rangle \mid N_i^A(x) \text{ accepts in some computation path in } r \text{ steps }\},$$

where $N_1, N_2, N_3, \ldots$ is the usual list of nondeterministic polynomial time Turing machines. It is known that for any constant k, there exist an oracle A where the Boolean Hierarchy over $\mathrm{NPC}(A)$ has exactly k distinct levels [CGH+88]. (That is, $\mathrm{BH}(\mathrm{NPC}(A)) = \mathrm{BH}_k(\mathrm{NPC}(A))$, but $\mathrm{BH}_{k-1}(\mathrm{NPC}(A)) \neq \mathrm{BH}_k(\mathrm{NPC}(A))$.) There is also an oracle A where the Boolean hierarchy over $\mathrm{NPC}(A)$ has infinitely many levels [CGH+88]. In fact, the latter case occurs with probability 1 for random oracles [Cai87]:

$$\mathrm{Prob}_A[\ \mathrm{BH}(\mathrm{NPC}(A)) \text{ is infinite }] = 1.$$

So, let us fix an oracle $\mathcal{O}$ and a language $I \stackrel{\mathrm{def}}{=} \mathrm{NPC}(\mathcal{O})$ such that $\mathrm{BH}(I)$ has infinitely many distinct levels. Since the proof techniques used in the previous sections work in presence of any oracle (i.e., the results relativize), the complete languages for the levels of $\mathrm{BH}(I)$ provide us with examples of languages which have various subsets of the Boolean combining functions:

1. $\mathrm{BL}_2(I)$ has AND_2 and AND_ω, but does not have either OR_2 or OR_ω.
2. co-$\mathrm{BL}_2(I)$ has OR_2 and OR_ω, but does not have either AND_2 or AND_ω.
3. $\mathrm{BL}_3(I)$ does not have AND_2, AND_ω, OR_2 or OR_ω.

These examples follow from the relativized versions of the results in the previous sections. For example, $\mathrm{BL}_3(I)$ cannot have OR_2 because if $\mathrm{BL}_3(I)$ had OR_2, then by Theorem 12, $\mathrm{BH}(I)$ would collapse to $\mathrm{BH}_3(I)$. This would contradict our assumption that $\mathrm{BH}(I)$ is an infinite hierarchy.

The examples above show that there exist languages which have some subset of AND_2, AND_ω, OR_2 and OR_ω. However, these examples do not exhaust all the possibilities. In particular, we have not come across an example of a language which has AND_2 but not AND_ω. In the remainder of the section we show how to construct such examples by diagonalization. Table 1 summarizes these examples.

AND_2	AND_ω	OR_2	OR_ω	Example
no	no	no	no	$\text{BL}_3(I)$
no	no	no	yes	not possible
no	no	yes	no	Corollary 22
no	no	yes	yes	co-$\text{BL}_2(I)$
no	yes	no	no	not possible
no	yes	no	yes	not possible
no	yes	yes	no	not possible
no	yes	yes	yes	not possible
yes	no	no	no	Theorem 21
yes	no	no	yes	not possible
yes	no	yes	no	Theorem 20
yes	no	yes	yes	Corollary 24
yes	yes	no	no	$\text{BL}_2(I)$
yes	yes	no	yes	not possible
yes	yes	yes	no	Theorem 23
yes	yes	yes	yes	SAT

Table 1: Examples of languages having various subsets of the Boolean combining functions.

In the proofs that follow, we will construct languages which will diagonalize against the polynomial time computable functions. So, let $D_1, D_2, D_3, \ldots$ be a list of all polynomial time Turing machines. We assume that D_i runs in time n^{k_i} where the exponent k_i can be easily determined from the encoding of D_i. Moreover, we assume that the size of the encoding of D_i, $|D_i|$, does not grow too quickly. The assumption that $|D_i| \leq \log i$ suffices. Also, we will assume that $k_i \leq \log\log i$. These additional assumptions are sufficient to show that the languages constructed can be recognized in exponential time. However, for the sake of brevity we do not include a running time analysis here.

Theorem 20

There exists a language A such that A has AND_2 and OR_2, but does not have AND_ω or OR_ω.

Proof: We construct A over the alphabet $\Sigma = \{\ 0, 1, (,), \#, \wedge, \vee\ \}$. Let $\mathcal{S}$ be the set of all the strings in Σ^* which have balanced parentheses. The language A that we construct will be a subset

of $\mathcal{S}$. Also, for any set X, we use $X^{\leq m}$ to denote the set of strings in X of length less than or equal to m. For notational convenience let us define two functions $\alpha_2, \beta_2 : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ by

$$\alpha_2(x,y) = (x) \wedge (y)\#^{|xy|^2} \quad \text{and} \quad \beta_2(x,y) = (x) \vee (y)\#^{|xy|^2}.$$

For any $B \subseteq \mathcal{S}^{\leq m}$ we define CLOSURE(B,m) to be the closure of B with respect to α_2 and β_2 up to length m. This closure is simply the smallest set C such that $B \subseteq C \subseteq \mathcal{S}^{\leq m}$ and the following conditions hold.

1. $x \in C$ and $y \in C$ and $|\alpha_2(x,y)| \leq m \Longrightarrow \alpha_2(x,y) \in C$.
2. $\alpha_2(x,y) \in C \Longrightarrow x \in C$ and $y \in C$.
3. ($x \in C$ or $y \in C$) and $|\beta_2(x,y)| \leq m \Longrightarrow \beta_2(x,y) \in C$.
4. $\beta_2(x,y) \in C \Longrightarrow x \in C$ or $y \in C$.

The closure property guarantees that CLOSURE(B,m) = CLOSURE(CLOSURE$(B,m),m)$.

In our construction we want to prove that A has AND_2 and OR_2. It suffices to show that $A = \bigcup_{m=0}^{\infty} \text{CLOSURE}(A^{\leq m}, m)$. If this conditions holds, then $\text{AND}_2(A) \leq_{\text{m}}^{\text{P}}$-reduces to A via α_2, and $\text{OR}_2(A) \leq_{\text{m}}^{\text{P}}$-reduces to A via β_2. Note that we need to use the #'s for non-linear padding in α_2 and β_2 because if $\text{AND}_2(A) \leq_{\text{m}}^{\text{P}} A$ via a linear time function, then A would have AND_ω as well.

Now we construct A in stages. During Stage s we construct A_s, an initial segment of A, so $A_0 \subseteq A_1 \subseteq A_2 \subseteq \cdots$ and $A \stackrel{\text{def}}{=} \bigcup_{s=0}^{\infty} A_s$. During the even stages $s = 2i$, we guarantee that the Turing machine D_i does not reduce $\text{OR}_\omega(A)$ to A. During the odd stages $s = 2i+1$, we make sure that D_i does not reduce $\text{AND}_\omega(A)$ to A. Also, all of the strings that we add in stage s (i.e., $A_s - A_{s-1}$) will have lengths between n_s and m_s inclusively. We start with $A_0 = \emptyset$ and $n_0 = m_0 = 0$.

Stage $s = 2i$:

1. Let $n_s = m_{s-1} + 1$ and $m_s = n_s^{3k_i}$.
2. $A_s :=$ CLOSURE(A_{s-1}, m_s).
3. Choose n_s new strings $x_1, \ldots, x_{n_s}$ from $\{0,1\}^{n_s}$. Compute $z = D_i(x_1, \ldots, x_{n_s})$. Note that $|z| \leq (n_s^2)^{k_i}$. If $z \in A_s$, then proceed to the next stage.
4. Otherwise, $z \notin A_s$. Pick an x_j such that $z \notin$ CLOSURE$(A_s \cup \{x_j\}, m_s)$. (We will show below that such an x_j always exists.) Then, let $A_s :=$ CLOSURE$(A_s \cup \{x_j\}, m_s)$ and proceed to the next stage.

Before we show the construction for Stage $2i+1$, we should show why the machine D_i cannot reduce $\text{OR}_\omega(A)$ to A and why x_j always exists. First, observe that the closure process in Step 2 will only add to A_s strings that are in the range of α_2 and β_2. Since the strings $x_1, \ldots, x_{n_s}$ chosen in Step 3 do not contain any parentheses, they are not in the range of α_2 and β_2 and will not be

placed in A by a closure procedure. Thus, none of $x_1, \ldots, x_{n_s}$ are in A_s at Step 3. Now, consider the string $z = D_i(x_1, \ldots, x_{n_s})$. If z is already an element of A_s, then at the end of the construction $z \in A$, but $\langle x_1, \ldots, x_{n_s} \rangle \notin \mathrm{OR}_\omega(A)$. So, D_i does not reduce $\mathrm{OR}_\omega(A)$ to A.

On the other hand, if $z \notin A_s$ and we find an x_j such that $z \notin \mathrm{CLOSURE}(A_s \cup \{x_j\}, m_s)$, then at the end of Stage s, $\langle x_1, \ldots, x_{n_s} \rangle \in \mathrm{OR}_\omega(A_s)$, but $z \notin A_s$. Furthermore, since $|z| \leq m_s$, z will not be placed in A during the subsequent stages of the construction. Thus, $\langle x_1, \ldots, x_{n_s} \rangle \in \mathrm{OR}_\omega(A)$, but $z \notin A$. So, D_i does not reduce $\mathrm{OR}_\omega(A)$ to A.

Finally, we need to show that in Step 4 an x_j such that $z \notin \mathrm{CLOSURE}(A \cup \{x_j\}, m_s)$ can always be found. To prove this, we define the *depends* relation on $\mathcal{S} \times \mathcal{S}$. Let u be any string in $\mathcal{S}$. If u is in the range of α_2 or β_2, that is $u = \alpha_2(v, w)$ or $u = \beta_2(v, w)$ for some $v, w \in \mathcal{S}$, then we say that u depends on v and u depends on w. (Note that u can only be one of $\alpha_2(v, w)$ or $\beta_2(v, w)$.) Otherwise, if u is not in the range of α_2 or β_2, then we call u an *atom*. We also make the depends relation transitive and reflexive. Observe that by the definition of α_2 and β_2, if u depends on v, then v is actually a substring of u. Since u contains mostly # for padding, u cannot depend on many strings. In fact, a string $u \in \mathcal{S}$ of length n can depend on at most $\log n$ many atoms (proof by induction).

Now, for all lengths n and for all $B \subseteq \mathcal{S}^{\leq n}$, let u and v be any two strings in $\mathcal{S}^{\leq n}$. Suppose that u does *not* depend on v, then whether u is in the closure of B is independent of whether v is in B. That is, $u \in \mathrm{CLOSURE}(B, n) \iff u \in \mathrm{CLOSURE}(B - \{v\}, n) \iff u \in \mathrm{CLOSURE}(B \cup \{v\}, n)$. This can also be shown by induction.

To spite D_i in Step 4, we need to find an x_j such that $z \notin \mathrm{CLOSURE}(A_s \cup \{x_j\}, m_s)$. By, our argument above, z can only depend on $3k_i \log n_s$ atoms. Since $i = s/2$ and $k_i \leq \log \log i$, we know that $3k_i \log n_s < n_s$, for all but the first few stages. Thus, z does not depend on one of the n_s atoms $x_1, \ldots, x_{n_s}$ that we have chosen. So, there always exist an appropriate x_j.

In Stage $s = 2i + 1$, we make sure that D_i does not reduce $\mathrm{AND}_\omega(A)$ to A. The correctness argument for the odd stages of the diagonalization is analogous to the even stages.

Stage $s = 2i + 1$*:*

1. Let $n_s = m_{s-1} + 1$ and $m_s = n_s^{3k_i}$.
2. $A_s := \mathrm{CLOSURE}(A_{s-1}, m_s)$.
3. Choose n_s strings $x_1, \ldots, x_{n_s}$ from $\{0,1\}^{n_s}$. Let X denote the set $\{x_1, \ldots, x_{n_s}\}$. Compute $z = D_i(x_1, \ldots, x_{n_s})$. If $z \notin \mathrm{CLOSURE}(A_s \cup X, m_s)$, let $A_s := \mathrm{CLOSURE}(A_s \cup X, m_s)$ and proceed to the next stage.
4. Otherwise, $z \in \mathrm{CLOSURE}(A_s \cup X, m_s)$. Pick an x_j such that z does not depend x_j. Then, $z \in \mathrm{CLOSURE}(A_s \cup (X - \{x_j\}), m_s)$. Let $A_s := \mathrm{CLOSURE}(A_s \cup (X - \{x_j\}), m_s)$ and proceed to the next stage. □

Theorem 21

There exists a language A such that A has AND_2 but does not have OR_2, AND_ω or OR_ω.

Proof: The construction of the set A for this theorem is similar to the one in Theorem 20. In this case, we are only concerned with closures with respect to α_2, so for any $B \subseteq \mathcal{S}^{\leq m}$, we define $\mathrm{CLOSURE}(B, m)$ to be the smallest set C such that $B \subseteq C \subseteq \mathcal{S}^{\leq m}$ and the following conditions hold.

1. $x \in C$ and $y \in C$ and $|\alpha_2(x, y)| \leq m \Longrightarrow \alpha_2(x, y) \in C$.
2. $\alpha_2(x, y) \in C \Longrightarrow x \in C$ and $y \in C$.

As before, our construction of A will guarantee that $A = \bigcup_{m=1}^{\infty} \mathrm{CLOSURE}(A^{\leq m}, m)$ which is sufficient to show that $\mathrm{AND}_2(A) \leq_{\mathrm{m}}^{\mathrm{P}} A$ via α_2. During Stage $2i$ of the construction, we make sure that the machine D_i does not reduce $\mathrm{OR}_2(A)$ to A.

Stage $s = 2i$:

1. Let $n_s = m_{s-1} + 1$ and $m_s = n_s^{3k_i}$.
2. $A_s := \mathrm{CLOSURE}(A_{s-1}, m_s)$.
3. Choose 2 strings x_1, x_2 from $\{0,1\}^{n_s}$. Compute $z = D_i(x_1, x_2)$. If $z \in A_s$, then proceed to the next stage.
4. Otherwise, $z \notin A_s$. Pick an x_j, $j = 1$ or 2, such that $z \notin \mathrm{CLOSURE}(A_s \cup \{x_j\}, m_s)$. Let $A_s := \mathrm{CLOSURE}(A_s \cup \{x_j\}, m_s)$ and proceed to the next stage.

Now, we need to argue that the appropriate x_j exists. If z does not depend on x_1 then we can simply let $x_j := x_1$ since z would not be in $\mathrm{CLOSURE}(A_s \cup \{x_1\}, m_s)$. The situation is analogous if z does not depend on x_2. However, since z may depend on as many as $\log m_s$ strings, it may be the case that z depends on both x_1 and x_2. Then, we spite D_i by placing x_1 in A and leaving x_2 out (i.e., let $x_j := x_1$). In this case, $z \notin \mathrm{CLOSURE}(A_s \cup \{x_j\}, m_s)$, because z will only be placed in the closure if all of the atoms on which z depends are in $A_s \cup \{x_j\}$ and $x_2 \notin A_s \cup \{x_j\}$. (This observation holds because we are not considering closure under β_2.) Thus, D_i does not reduce $\mathrm{OR}_2(A)$ to A.

During the odd stages, we make sure that D_i does not reduce $\mathrm{AND}_\omega(A)$ to A. This part of the construction is identical to the odd stages in the proof of Theorem 20. □

Let A be the language constructed in Theorem 21. Then, by DeMorgan's Law, $\overline{A}$ would have OR_2 but not AND_2, AND_ω or OR_ω. Thus, we have the following corollary.

Corollary 22

There exists a language B such that B has OR_2 but does not have AND_2, AND_ω or OR_ω.

Theorem 23

There exists a language A such that A has AND_2, OR_2 and AND_ω but does not have OR_ω.

Proof: We will construct a set A that is closed with respect to β_2 and α_ω, where α_ω is the any-ary function defined by:

$$\alpha_\omega(x_1, \ldots, x_r) = (x_1) \wedge \cdots \wedge (x_r) \#^{|x_1 \cdots x_r|^2}$$

Again, for any set $B \subseteq \mathcal{S}^{\leq m}$, we define $\mathrm{CLOSURE}(B, m)$ to be the smallest set C such that $B \subseteq C \subseteq \mathcal{S}^{\leq m}$ and the following conditions hold.

1. $\forall \ell,\ 1 \leq \ell \leq r,\ x_\ell \in C$ and $|\alpha_\omega(x_1, \ldots, x_r)| \leq m \Longrightarrow \alpha_\omega(x_1, \ldots, x_r) \in C$.
2. $\alpha_\omega(x_1, \ldots, x_r) \in C \Longrightarrow \forall \ell,\ 1 \leq \ell \leq r,\ x_\ell \in C$.
3. ($x \in C$ or $y \in C$) and $|\beta_2(x, y)| \leq m \Longrightarrow \beta_2(x, y) \in C$.
4. $\beta_2(x, y) \in C \Longrightarrow x \in C$ or $y \in C$.

The construction below will guarantee that $A = \bigcup_{m=1}^{\infty} \mathrm{CLOSURE}(A^{\leq m}, m)$. This is sufficient to show that $\mathrm{AND}_\omega(A) \leq_{\mathrm{m}}^{\mathrm{P}} A$ via α_ω and $\mathrm{OR}_2(A) \leq_{\mathrm{m}}^{\mathrm{P}} A$ via β_2. Moreover, at Stage s of the construction, we make sure that D_s does not reduce $\mathrm{OR}_\omega(A)$ to A.

Stage s:

1. Let $n_s = m_{s-1} + 1$ and $m_s = n_s^{3k_i}$.
2. $A_s := \mathrm{CLOSURE}(A_{s-1}, m_s)$.
3. Choose n_s strings $x_1, \ldots, x_{n_s}$ from $\{0,1\}^{n_s}$. Compute $z = D_s(x_1, \ldots, x_{n_s})$. If $z \in A_s$, then proceed to the next stage.
4. Otherwise, $z \notin A_s$. Then, pick an x_j such that $z \notin \mathrm{CLOSURE}(A_s \cup \{x_j\}, m_s)$. Let $A_s := \mathrm{CLOSURE}(A_s \cup \{x_j\}, m_s)$ and proceed to the next stage.

As before, we need to show that the appropriate x_j always exists. In the proof of Theorem 20, we used the fact that z depends only on $\log m_s$ many strings. In this proof, however, we want A to be closed with respect to α_ω, so the number of strings that z depends on may be polynomial in n_s. Fortunately, as we show next, not all of these strings can affect the closure of A_s.

For all n, let $B \subseteq \mathcal{S}^{\leq n}$ and $y \in \mathcal{S}^{\leq n}$ such that $y \notin \mathrm{CLOSURE}(B, n)$. We say that y is *affected* by a string $v \in \mathcal{S}^{\leq n}$ if $y \in \mathrm{CLOSURE}(B \cup \{v\}, n)$. We show by induction that y is affected by at most $\log n$ atoms. First, if y is an atom (i.e., y is not in the range of α_ω or β_2) then y is only affected by itself. Otherwise, if $y = \beta_2(v_1, v_2)$, then neither v_1 nor v_2 can be in $\mathrm{CLOSURE}(B, n)$. Thus, y is affected by v_1, v_2 and all of the atoms which affect v_1 and v_2. Now, $|v_1| \leq \sqrt{n}$ and $|v_2| \leq \sqrt{n}$. So, by the induction hypothesis, each of v_1 and v_2 is affected by only $\log(\sqrt{n})$ atoms.

Thus, y is affected by at most $2\log(\sqrt{n}) = \log n$ atoms. Finally, if $y = \alpha_\omega(v_1, \ldots, v_r)$ then for all $E \subseteq \mathcal{S}^{\leq n}$,

$$y \in \text{CLOSURE}(E, n) \iff \forall \ell,\ 1 \leq \ell \leq r,\ v_\ell \in \text{CLOSURE}(E, n).$$

In particular, since $y \notin \text{CLOSURE}(B, n)$, we know that for some v_t, $v_t \notin \text{CLOSURE}(B, n)$. Moreover, if y is affected by an atom u, then $y \in \text{CLOSURE}(B \cup \{u\}, n)$ which in turn implies that $v_t \in \text{CLOSURE}(B \cup \{u\}, n)$. Thus, any atom which affects y must also affect v_t. Then, by the induction hypothesis y is affected by at most $\log(|v_t|) \leq \log(\sqrt{|y|}) \leq 1/2 \log n$ atoms.

Using the claim above, we now show that in Step 4 the appropriate x_j can always be found. Since $z \notin A_s$, we also know that $z \notin \text{CLOSURE}(A_s, m_s)$ because A_s has already been closed in Step 2. Then, applying the above claim with $B = A_s$ and $n = m_s$, we know that z can only be affected by $\log m_s$ atoms. Since $\log m_s < n_s$ for all but the first few stages and since the x_ℓ's are atoms, z is not affected by one of the x_ℓ's. Thus, $z \notin \text{CLOSURE}(A \cup \{x_j\}, m_s)$ and D_s does not reduce $\text{OR}_\omega(A)$ to A. □

Let A be the language constructed in Theorem 23. Again, by DeMorgan's Law, $\overline{A}$ would have AND_2, OR_2 and AND_ω but not OR_ω. Thus, we have the following corollary.

Corollary 24

There exists a language B such that B has AND_2, OR_2 and OR_ω but does not have AND_ω.

References

[ABG90] A. Amir, R. Beigel, and W. I. Gasarch. Some connections between bounded query classes and non-uniform complexity. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 232–243, 1990.

[AG88] A. Amir and W. I. Gasarch. Polynomial terse sets. *Information and Computation*, 77:37–56, April 1988.

[BBJ+89] A. Bertoni, D. Bruschi, D. Joseph, M. Sitharam, and P. Young. Generalized Boolean hierarchies and the hierarchy over RP. Technical Report 809, Department of Computer Sciences, University of Wisconsin—Madison, 1989.

[Bei88] R. Beigel. NP-hard sets are p-superterse unless R = NP. Technical Report 4, Department of Computer Science, The Johns Hopkins University, 1988.

[Bei91] R. Beigel. Bounded queries to SAT and the Boolean hierarchy. *Theoretical Computer Science*, 84(2):199–223, July 1991.

[Cai87] J. Cai. Probability one separation of the Boolean hierarchy. In *4th Annual Symposium on Theoretical Aspects of Computer Science*, volume 247 of *Lecture Notes in Computer Science*, pages 148–158. Springer-Verlag, 1987.

[CGH+88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The Boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, December 1988.

[CH85] J. Cai and L. A. Hemachandra. The Boolean hierarchy: Hardware over NP. Technical Report TR 85-724, Cornell Department of Computer Science, December 1985.

[Cha89] R. Chang. On the structure of bounded queries to arbitrary NP sets. In *Proceedings of the 4th Structure in Complexity Theory Conference*, pages 250–258, June 1989. Revised paper in *SIAM Journal on Computing,* 21(4):743–754, August 1992.

[CK90] R. Chang and J. Kadin. The Boolean hierarchy and the polynomial hierarchy: a closer connection. In *Proceedings of the 5th Structure in Complexity Theory Conference*, pages 169–178, July 1990. To appear in *SIAM Journal on Computing*.

[CKR91] R. Chang, J. Kadin, and P. Rohatgi. Connections between the complexity of unique satisfiability and the threshold behavior of randomized reductions. In *Proceedings of the 6th Structure in Complexity Theory Conference*, pages 255–269, July 1991. Revised paper to appear in *Journal of Computer and System Sciences*.

[GJY87] J. Goldsmith, D. Joseph, and P. Young. Self-reducible, p-selective, near-testable, and p-cheatable sets: The effect of internal structure on the complexity of a set. In *Proceedings of the 2nd Structure in Complexity Theory Conference*, pages 50–59, 1987.

[Hem89] L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.

[Hof79] C. M. Hoffman. *Group-Theoretic Algorithms and Graph Isomorphism*, volume 136 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[Kad88] J. Kadin. The polynomial time hierarchy collapses if the Boolean hierarchy collapses. *SIAM Journal on Computing*, 17(6):1263–1282, December 1988.

[Kad89] J. Kadin. $P^{NP[\log n]}$ and sparse Turing complete sets for NP. *Journal of Computer and System Sciences*, 39:282–298, December 1989.

[Kre88] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[KSW87] J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *RAIRO Theoretical Informatics and Applications*, 21:419–435, 1987.

[LLS75] R. E. Ladner, N. A. Lynch, and A. L. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–123, 1975.

[PY84] C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, April 1984.

[Sch88] U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37:312–323, December 1988.

[Sel79] A. L. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979.

[WW85] K. Wagner and G. Wechsung. On the Boolean closure of NP. In *Proceedings of the 1985 International Conference on Fundamentals of Computation Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 485–493. Springer-Verlag, 1985.