On Computing Boolean Connectives of Characteristic Functions

Richard Chang*
Jim Kadin

TR 90-1118
May 1990

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*Supported in part by NSF Research Grant DCR-8520597 and CCR-88-23053, and by an IBM Graduate Fellowship.
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

May 9, 1990

Abstract

We study 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, we show that Graph Isomorphism, which is probably not complete, also has them. We characterize the complete sets for the classes $\text{D}^p$ and $\text{P}^{\text{NP}[O(\log n)]}$ in terms of AND and OR, and we relate these functions to the structure of the Boolean hierarchy and the query hierarchies. We show that the sets that are complete for levels above the second level of the Boolean hierarchy do not have AND and OR unless the polynomial hierarchy collapses. We show that most of the structural properties of the Boolean hierarchy and query hierarchies depend only on the existence of AND and OR functions for 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 $\text{AND}_2$ 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 $\text{OR}_2$ 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 ($\text{AND}_\omega$ and $\text{OR}_\omega$), polynomial time functions $f$ and $g$ such that for all $n$ and strings $x_1, \ldots, x_n$, $f(x_1, \ldots, x_n) \in L$ if and only if $x_1, \ldots, x_n$ are all in $L$, and $g(x_1, \ldots, x_n) \in L$ if and only if at least one of $x_1, \ldots, x_n$ is in $L$.

The existence of these functions is intimately tied to questions about polynomial time reducibilities and structural properties of languages and complexity classes. Our initial motivation for considering these functions was the observation that all NP-complete languages have $\text{AND}_\omega$ and $\text{OR}_\omega$ functions. In fact any language that is $\leq^p_m$-complete for even relativized versions of complexity classes such as P, NP, PSPACE have any-ary Boolean functions by virtue of the fact that such robust complexity classes are represented by machine models that can be run on the different strings in the input tuple. We will show that languages that are $\leq^p_m$-complete for $\text{D}^p$ [PY84] have $\text{AND}_\omega$ but do not have $\text{OR}_2$ unless the polynomial time hierarchy collapses. Complete languages for the higher levels of the Boolean hierarchy [CGH+88] do not have either $\text{AND}_2$ or $\text{OR}_2$ unless the polynomial time hierarchy collapses.

*Supported in part by NSF research grants DCR-8520597 and CCR-88-23053, and by an IBM Graduate Fellowship.
These Boolean functions are related to polynomial time conjunctive and disjunctive reducibilities (defined by Ladner, Lynch, and Selman [LLS75]) and to closure of complexity classes under union and intersection. Let m-1(L) be the class of languages \( \leq^p_m \)-reducible to a language \( L \). It is easy to show that:

\[
L \text{ has AND}_2 \iff m-1(L) \text{ is closed under intersection} \\
\iff m-1(L) \text{ is closed under } \leq^p_2,
\]

\[
L \text{ has AND}_\omega \iff m-1(L) \text{ is closed under } \leq^p_
\]

(\( \leq^p_2 \) and \( \leq^p_\omega \) are defined in [LLS75]). 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 \( \leq^p_m \)-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 \( \leq^p_m \)-complete languages of \( D^P \) and \( P^{NP[O(\log n)]} \) in terms of AND and OR functions. In Section 5 we use the above characterizations to show that the complete languages of the different 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]).

Finally, 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 used by Beigel to show that \( P^{SAT}[2^{k-1}] = P^{SAT}[k] \) 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 \( \text{AND}_2 \) and \( \text{OR}_2 \). The results of this section have been proven independently by Bertoni, Bruschi, Joseph, Sitharam, and Young [BBJ+89].

## 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 (\( \leq^p_m \)).

Although our intuitive notion of AND and OR consists of polynomial time functions that operate on strings, it is convenient to define \( \text{AND}_2, \text{AND}_\omega, \text{OR}_2, \text{OR}_\omega \) as sets:

**Definition** For any set \( A \), we define the sets

- \( \text{AND}_2(A) = \{(x, y) \mid x \in A \text{ and } y \in A\} \)
- \( \text{OR}_2(A) = \{(x, y) \mid x \in A \text{ or } y \in A\} \)
- \( \text{AND}_k(A) = \{(x_1, \ldots, x_k) \mid \forall i, 1 \leq i \leq k, x_i \in A\} \)
- \( \text{OR}_k(A) = \{(x_1, \ldots, x_k) \mid \exists i, 1 \leq i \leq k, x_i \in A\} \)
- \( \text{AND}_\omega(A) = \bigcup_{i=1}^{\infty} \text{AND}_i(A) \)
\[ \text{OR}_\omega(A) = \bigcup_{i=1}^{\infty} \text{OR}_i(A). \]

If \(\text{AND}_2(A) \leq^P_m A\) or \(\text{AND}_\omega(A) \leq^P_m A\), then we say that \(A\) has \(\text{AND}_2\) or \(\text{AND}_\omega\) respectively. If \(\text{OR}_2(A) \leq^P_m A\) or \(\text{OR}_\omega(A) \leq^P_m A\), then we say that \(A\) has \(\text{OR}_2\) or \(\text{OR}_\omega\) respectively. Some obvious facts about \(\text{AND}_2(A)\) and \(\text{OR}_2(A)\) are:

1. \(\text{AND}_2(A) \leq^P_m A \iff \text{OR}_2(\overline{A}) \leq^P_m \overline{A}.\)
2. \(\text{if } A \equiv^P_m B \text{ then } \text{AND}_2(A) \leq^P_m A \iff \text{AND}_2(B) \leq^P_m B.\)
3. \(\text{if } A \equiv^P_m B \text{ then } \text{OR}_2(A) \leq^P_m A \iff \text{OR}_2(B) \leq^P_m B.\)

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

Note that if \(\text{AND}_2(A) \leq^P_m A\), then for all \(k\), \(\text{AND}_k(A) \leq^P_m A\). It is possible that \(\text{AND}_2(A) \leq^P_m A\), but \(\text{AND}_\omega(A) \not\leq^P_m A\). However if \(\text{AND}_2(A) \leq^P_m A\) by a linear time reduction, then \(\text{AND}_\omega(A) \leq^P_m A\).

**Lemma 1** If \(\text{AND}_2(A) \leq^P_m A\) by a linear time reduction, then \(\text{AND}_\omega(A) \leq^P_m A\). Similarly, if \(\text{OR}_2(A) \leq^P_m A\) by a linear time reduction, then \(\text{OR}_\omega(A) \leq^P_m A\).

**Proof:** Let \(f\) be a linear time reduction from \(\text{AND}_2(A)\) to \(A\). For any \(r\), we can take a tuple \((x_1, \ldots, x_r)\) and apply \(f\) pairwise to \(f(x_1, x_2) f(x_3, x_4) \cdots f(x_{r-1}, x_r)\). 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 \(\log r\). If \(n\) is the total length of the tuple, \(r \leq n\), and so the total running time is bounded by \(e^{\log^* n}\) for some constant \(c\). This is polynomial in \(n\). \(\square\)

**Definition** We write BH\((k)\) and co-BH\((k)\) for the \(k\)th levels of the Boolean hierarchy, defined as follows:

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

**Definition** We write L_{BH\((k)\)} for the canonical complete language for BH\((k)\) and L_{co-BH\((k)\)} for the complete language for co-BH\((k)\):

\[
\begin{align*}
L_{BH}(1) &= \text{SAT}, \\
L_{BH}(2k) &= \{\langle x_1, \ldots, x_{2k} \rangle \mid \langle x_1, \ldots, x_{2k-1} \rangle \in L_{BH}(2k-1) \text{ and } x_{2k} \in \overline{\text{SAT}}\}, \\
L_{BH}(2k+1) &= \{\langle x_1, \ldots, x_{2k+1} \rangle \mid \langle x_1, \ldots, x_{2k} \rangle \in L_{BH}(2k) \text{ or } x_{2k+1} \in \text{SAT}\}, \\
L_{\text{co-BH}}(1) &= \overline{\text{SAT}}, \\
L_{\text{co-BH}}(2k) &= \{\langle x_1, \ldots, x_{2k} \rangle \mid \langle x_1, \ldots, x_{2k-1} \rangle \in L_{\text{co-BH}}(2k-1) \text{ or } x_{2k} \in \text{SAT}\}, \\
L_{\text{co-BH}}(2k+1) &= \{\langle x_1, \ldots, x_{2k+1} \rangle \mid \langle x_1, \ldots, x_{2k} \rangle \in L_{\text{co-BH}}(2k) \text{ and } x_{2k+1} \in \overline{\text{SAT}}\}.
\end{align*}
\]

Alternatively, BH\((2)\) is called D^P = \{L_1 \cap L_2 \mid L_1, L_2 \in \text{NP}\}. We will use the two terms interchangeably. Also, we will refer to L_{BH\((2)\)} as SAT \land \overline{\text{SAT}} the traditional \leq^P_m-complete language for D^P.
Definition For any set $A$, we write $P^A[k]$ for the set of languages recognized by polynomial time Turing machines that ask at most $k$ queries to the oracle $A$ on any input. The query hierarchy relative to $A$ is $\text{QH}_A \overset{\text{def}}{=} \bigcup_{k=1}^{\infty} P^A[k]$. Similarly, $P^{A[O(\log n)]}$ is the set of languages recognized by polynomial time Turing machines that ask $O(\log n)$ queries to $A$. Also, we write $P^{A[||k]}$ for the set of languages recognized by polynomial time Turing machines which ask at most $k$ queries in parallel on any input. Finally, we define the parallel query hierarchy relative to $A$ as $\text{QH}_{||A} \overset{\text{def}}{=} \bigcup_{k=1}^{\infty} P^{A[||k]}$.

Fact The Boolean, parallel query and the query hierarchies (relative to SAT) are intertwined [Bei87,Hem87]

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

3 Languages Which Do

In this section, we present some familiar languages which are known to have $\text{AND}_\omega$ and $\text{OR}_\omega$.

Lemma 2

1. SAT has $\text{AND}_\omega$ and $\text{OR}_\omega$.
2. SAT $\land \overline{\text{SAT}}$ has $\text{AND}_\omega$.

Proof:

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

   $\langle f_1, \ldots, f_n \rangle \in \text{AND}_\omega(\text{SAT}) \iff f_1 \land \ldots \land f_n \in \text{SAT}$

   $\langle f_1, \ldots, f_n \rangle \in \text{OR}_\omega(\text{SAT}) \iff f_1 \lor \ldots \lor f_n \in \text{SAT}$.

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

   $\langle (f_1, g_1), \ldots, (f_n, g_n) \rangle \in \text{AND}_\omega(\text{SAT} \land \overline{\text{SAT}})$

   $\iff (f_1 \land \ldots \land f_n, g_1 \lor \ldots \lor g_n) \in \text{SAT} \land \overline{\text{SAT}}$. 

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

Definition $\text{GI} \overset{\text{def}}{=} \{(G, H) \mid G \text{ and } H \text{ are isomorphic graphs}\}$. 
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**

\[ \text{LGI} \overset{\text{def}}{=} \{ (G, H) \mid G \text{ and } H \text{ are isomorphic graphs with labelled nodes, and the isomorphism preserves labels } \} . \]

We will show that LGI has AND_\omega and OR_\omega functions. The existence of AND_\omega and OR_\omega functions for GI follows from the fact that LGI \equiv_p GI [Hof79].

**Lemma 3** LGI has AND_\omega and OR_\omega.

**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_\omega function for LGI as follows. Given \( r \) pairs of graphs \( \{(G_1, H_1), \ldots, (G_r, H_r)\} \), 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_\omega, 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_\omega.

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 \( H_2 \). \( G \) is produced similarly except the new edges connect \( G_1 \) with \( H_2 \) and \( H_1 \) with \( G_2 \) (see Figure 1).

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 \( H \), \( G_2 \) 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 \( H \). This means that 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_\omega. \( \square \)

**Corollary 4** GI has AND_\omega and OR_\omega.
4 Characterizations of Complete Languages

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

Lemma 5 If $SAT, \overline{SAT} \leq_m A$ and $A$ has $AND_2$, then $A$ is $D^P$ hard under $\leq_m$ reductions.

Proof: $SAT \land \overline{SAT}$ is $\leq_m$-complete for $D^P$. Clearly, $SAT \land \overline{SAT} \leq_m A$. So, $A$ must be $D^P$ hard. \square

Theorem 6 A set $A$ is $\leq_m$-complete for $D^P$ if and only if

1. $A \in D^P$.
2. $SAT, \overline{SAT} \leq_m A$.
3. $A$ has $AND_2$.

Now, we show that if a $D^P$-hard set has $OR_2$ or $OR_\omega$, then it is also hard for higher levels of the parallel query hierarchy.

Theorem 7 Let $A$ be any set that is $\leq_m$-hard for $D^P$.

1. If $A$ has $OR_2$, then for all $k$, $A$ is $\leq_m$-hard for $P^{SAT[k]}$.
2. If $A$ has $OR_\omega$, then $A$ is $\leq_m$-hard for $P^{NP[O(\log n)]}$.

Proof: Let $C$ be any set in $P^{SAT[k]}$. To determine if $x \in C$, consider the query tree of the $P^{SAT[k]}$ 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 $D^P$ 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 $p_1 \land \ldots \land p_i \in SAT$ and $q_1 \lor \ldots \lor q_j \in SAT$. This is exactly a $D^P$ computation.

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. We can use $r$ separate $D^P$ computations to check if any of the accepting paths is the correct path; and since we are assuming that $A$ is $D^P$-hard and has OR$_2$, we can use one $A$ computation instead of the $r$ $D^P$ computations.

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

\[ \square \]

Corollary 8 If $SAT, \overline{SAT} \leq_m^P A$ and $A$ has AND$_2$ and OR$_\omega$, then $A$ is $P^{NP[O(\log n)]}$ hard under $\leq_m^P$ reductions.

**Proof:** By Lemma 5 $A$ is a $D^P$-hard set. By the preceding theorem, $A$ is also hard for $P^{NP[O(\log n)]}$.

\[ \square \]

Theorem 9 A set $A$ is $\leq_m^P$-complete for $P^{NP[O(\log n)]}$ if and only if

1. $A \in P^{NP[O(\log n)]}$,
2. $SAT, \overline{SAT} \leq_m^P A$.
3. $A$ has AND$_2$.
4. $A$ has OR$_\omega$.

Notice that the only difference in the characterizations of $D^P$ complete sets and $P^{NP[O(\log n)]}$ complete sets (Theorems 6 and 9) is that $P^{NP[O(\log n)]}$ complete sets have OR$_\omega$. So, we have the following corollary.

Corollary 10 If $SAT \land \overline{SAT}$ has OR$_\omega$, then $P^{NP[O(\log n)]} \subseteq D^P$.

Since $P^{NP[O(\log n)]} \subseteq D^P$ implies that PH collapses to just above $\Sigma^P_2$ [CK89], Corollary 10 is evidence that $SAT \land \overline{SAT}$ does not have OR$_\omega$. In fact, it probably does not have OR$_2$, either.

Corollary 11 If $SAT \land \overline{SAT}$ has OR$_2$, then $P^{NP[O(\log n)]} \subseteq D^P$.

**Proof:** If $SAT \land \overline{SAT}$ has OR$_2$, then by Theorem 7, $SAT \land \overline{SAT}$ is hard for $P^{SAT[2]}$. However, the complement of $SAT \land \overline{SAT}$ is $SAT \lor SAT$ which is in $P^{SAT[2]}$. So, $SAT \land \overline{SAT} \leq_m^P SAT \lor SAT$. Since $SAT \land \overline{SAT}$ has AND$_\omega$ (Lemma 2), $SAT \lor SAT$ has OR$_\omega$ by DeMorgan's Law. Thus, $SAT \land \overline{SAT}$ must also have OR$_\omega$ since the two sets are $\leq_m^P$-equivalent. Then, by Corollary 10 $P^{NP[O(\log n)]} \subseteq D^P$.

\[ \square \]

Corollary 12 If $D^P = co-D^P$ then PH collapses and $P^{NP[O(\log n)]} \subseteq D^P$.

7
Proof: If $D^P = \text{co-D}^P$, then $\overline{\text{SAT}} \lor \text{SAT}$ is $\leq_m^P$-complete for $D^P$. So, $\overline{\text{SAT}} \lor \text{SAT}$ has $\text{AND}_2$ by Theorem 6. Then, by DeMorgan’s Law, SAT $\land \overline{\text{SAT}}$ has OR$_2$ which implies $\text{P}^{\text{NP}[O(\log n)]} \subseteq D^P$ by Theorem 11. The polynomial hierarchy collapses by Kadin’s proof [Kad88].

It was observed by Kadin [Kad87] that the collapse of the Boolean hierarchy at level 2 ($D^P = \text{co-D}^P$) immediately implies $\text{P}^{\text{NP}[O(\log n)]} \subseteq D^P$. 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 SAT $\land \overline{\text{SAT}}$ has AND$_2$. We will show in the next section that the complete languages for the 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 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 $\text{BH}(k) \subseteq \text{BH} \subseteq \text{P}^{\text{NP}[O(\log n)]}$. Also recall that if $\text{BH} \subseteq \text{BH}(k)$, then $\text{BH}(k) \subseteq \text{P}^{\text{NP}[O(\log n)]}$, or $\text{P}^{\text{NP}[O(\log n)]} \subseteq \text{BH}(k)$, then the polynomial hierarchy collapses. Of course Theorems 13 and 14 apply to any $\leq_m^P$-complete sets for the various levels of the BH.

Theorem 13 For $k \geq 2$,

1. $L_{\text{BH}(k)}$ has OR$_2$ $\iff$ BH $\subseteq \text{BH}(k)$.

2. $L_{\text{BH}(k)}$ has OR$_\omega$ $\iff$ $\text{P}^{\text{NP}[O(\log n)]} \subseteq \text{BH}(k)$.

Proof:

1. ($\Rightarrow$) For $k \geq 2$, $L_{\text{BH}(k)}$ is $D^P$-hard, so by Theorem 7, $L_{\text{BH}(k)}$ has OR$_2$ implies that it is hard for $\text{P}^{\text{SAT}[k]}$. However, $L_{\text{co-BH}(k)} \in \text{P}^{\text{SAT}[k]}$, so $L_{\text{co-BH}(k)} \leq_m^P L_{\text{BH}(k)}$. This implies that $\text{BH}(k)$ is closed under complementation, which implies $\text{BH} \subseteq \text{BH}(k)$.

   ($\Leftarrow$) If BH $\subseteq \text{BH}(k)$ then $\text{P}^{\text{SAT}[2k]} \subseteq \text{BH}(k)$, since $\text{P}^{\text{SAT}[2k]} \subseteq \text{BH}(2k + 1)$. Clearly, OR$_2(L_{\text{BH}(k)}) \in \text{P}^{\text{SAT}[2k]}$, so $L_{\text{BH}(k)}$ has OR$_2$.

2. ($\Rightarrow$) As above, $L_{\text{BH}(k)}$ is $D^P$-hard. By Theorem 9, $L_{\text{BH}(k)}$ has OR$_\omega$ implies that $L_{\text{BH}(k)}$ is $\leq_m^P$-complete for $\text{P}^{\text{NP}[O(\log n)]}$. So, $\text{P}^{\text{NP}[O(\log n)]} \subseteq \text{BH}(k)$.

   ($\Leftarrow$) OR$_\omega(L_{\text{BH}(k)}) \in \text{P}^{\text{SAT}[\|]} = \text{P}^{\text{NP}[O(\log n)]}$. So, if $\text{P}^{\text{NP}[O(\log n)]} \subseteq \text{BH}(k)$ then $L_{\text{BH}(k)}$ has OR$_\omega$.

Theorem 14 For $k \geq 3$,

1. $L_{\text{BH}(k)}$ has AND$_2$; $\iff$ BH $\subseteq \text{BH}(k)$.

2. $L_{\text{BH}(k)}$ has AND$_\omega$ $\iff$ $\text{P}^{\text{NP}[O(\log n)]} \subseteq \text{BH}(k)$.

Proof: DeMorgan’s Law implies that $L_{\text{co-BH}(k)}$ has OR$_2$ or OR$_\omega$. Then, the proof proceeds as above, because for $k \geq 3$, $L_{\text{co-BH}(k)}$ is $D^P$-hard.
functions. At the first level, \(L_{BH(1)} = SAT\), so \(L_{BH(1)}\) has both \(AND_\omega\) and \(OR_\omega\). At the second level, \(L_{BH(2)}\) is \(\mathcal{D}^P\)-complete, so \(L_{BH(2)}\) has \(AND_\omega\) but does not have \(OR_2\). From the third level up, \(L_{BH(k)}\) 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 \(\leq_m^P\)-complete languages for the different levels of query hierarchy and the parallel query hierarchy (relative to SAT) also do not have \(AND_2\) or \(OR_2\) unless the PH collapses.

**Theorem 15** For any \(k\), if \(C\) is \(\leq_m^P\)-complete for \(P^{SAT}[\|k\|]\), then

\[
C \text{ has } AND_2 \text{ or } OR_2 \implies BH \subseteq P^{SAT}[\|k\|].
\]

**Proof:** First, note that since \(P^{SAT}[\|k\|]\) is closed under complementation, \(\overline{C} \equiv_m^P C\). Therefore \(C\) has \(AND_2\) if and only if \(C\) has \(OR_2\). We will show that if \(C\) has either Boolean function, then \(BH(k+1) \subseteq P^{SAT}[\|k\|]\) which implies the collapse of the BH (and the PH). If \(k\) is odd, then the \(\leq_m^P\)-complete language for \(BH(k+1)\) is

\[
L_{BH(k+1)} = \{(x_1, \ldots, x_{k+1}) \mid (x_1, \ldots, x_k) \in L_{BH(k)} \text{ and } x_{k+1} \in \overline{SAT}\}.
\]

If \(C\) has \(AND_2\), then \(L_{BH(k+1)} \leq_m C\) since \(L_{BH(k)} \leq_m C\) and \(\overline{SAT} \leq_m C\). But this implies \(BH(k+1) \subseteq P^{SAT}[\|k\|]\). If \(k\) is even, then the \(\leq_m^P\)-complete language for \(BH(k+1)\) is

\[
L_{BH(k+1)} = \{(x_1, \ldots, x_{k+1}) \mid (x_1, \ldots, x_k) \in L_{BH(k)} \text{ or } x_{k+1} \in SAT\}.
\]

If \(C\) has \(OR_2\), then \(L_{BH(k+1)} \leq_m C\) since \(L_{BH(k)} \leq_m C\) and \(SAT \leq_m C\). Again this implies \(BH(k+1) \subseteq P^{SAT}[\|k\|]\). 

In related research, we have shown that USAT, the set of Boolean formulas that have exactly one satisfying assignment, does not have \(OR_\omega\) unless the PH collapses [CK90].

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,Bei87,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 [CGH+88].

2. **Complete languages:**

   (a) \(L_{BH(k)}\) is \(\leq_m^P\)-complete for \(BH(k)\).

   (b) \(ODD_k(SAT)\) is \(\leq_m^P\)-complete for \(BH(k)\) [Bei87,WW85]. (\(ODD_k(SAT)\) is defined below)

   (c) \(EVEN_k(SAT) \oplus ODD_k(SAT)\) is \(\leq_m^P\)-complete for \(P^{SAT}[\|k\|]\) [Bei87,WW85].

3. **Basic containments and intertwining** [Bei87,KSW87]:

\[
BH(k) \cup \text{co-BH}(k) \subseteq P^{SAT}[\|k\|] \subseteq BH(k+1) \cap \text{co-BH}(k+1).
\]
4. Intertwining of the query hierarchies [Bei87]:
\[ \text{pSAT}[k] = \text{pSAT}[2^k-1]. \]

5. Upward collapse:
\[
\begin{align*}
\text{BH}(k) = \text{co-BH}(k) & \quad \implies \quad \text{BH} \subseteq \text{BH}(k) \\
\text{pSAT}[k] = \text{pSAT}[k+1] & \quad \implies \quad \text{QH} \subseteq \text{pSAT}[k] \\
\text{pSAT}[k] = \text{pSAT}[k+1] & \quad \implies \quad \text{QH} \subseteq \text{pSAT}[k].
\end{align*}
\]

**Definition** For any set $A$, we define the following languages:
\[
\begin{align*}
\text{ODD}_k(A) & \overset{\text{def}}{=} \{ (x_1, \ldots, x_k) \mid \| \{x_1, \ldots, x_k \} \cap A \| \text{ is odd} \} \\
\text{EVEN}_k(A) & \overset{\text{def}}{=} \{ (x_1, \ldots, x_k) \mid \| \{x_1, \ldots, x_k \} \cap A \| \text{ is even} \}.
\end{align*}
\]

In this section, we show that all the above properties follow simply from the fact that SAT (or any NP-complete set) has AND$_2$ or OR$_2$. 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 QH$_\parallel$ relative to a set $A$ were defined in section 2. We define the Boolean hierarchy relative to $A$ as the difference hierarchy of $m$-1($A) \overset{\text{def}}{=} \text{set of all languages } \leq^P_m \text{-reducible to } A$.

**Definition**
\[
\begin{align*}
\text{BH}_A(1) & = m$-1$(A), \\
\text{BH}_A(2k) & = \{ L \mid L = L' \cap \overline{L_2} \text{ where } L' \in \text{BH}_A(2k-1) \text{ and } L_2 \in m$-1$(A) \}, \\
\text{BH}_A(2k+1) & = \{ L \mid L = L' \cup L_2 \text{ where } L' \in \text{BH}_A(2k) \text{ and } L_2 \in m$-1$(A) \}, \\
\text{co-BH}_A(k) & = \{ L \mid \overline{L} \in \text{BH}_A(k) \}, \\
\text{BH}_A & = \bigcup_{k=1}^{\infty} \text{BH}_A(k).
\end{align*}
\]

First, 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 $\text{BH}_A$ (see [CH85]). For example, a language $L$ is in $\text{BH}_A(k)$ iff there exist languages $L_1, \ldots, L_k \in m$-1($A$) such that
\[
L = D(L_1, \ldots, L_k) \overset{\text{def}}{=} L_1 - (L_2 - (L_3 - (\cdots - L_k))).
\]

**Theorem 16** If $A$ has AND$_2$ and OR$_2$, then $\text{ODD}_k(A)$ is $\leq^P_m$-complete for $\text{BH}_A(k)$.

**Proof:** First, we show that $\text{ODD}_k(A)$ is $\leq^P_m$-hard for $\text{BH}_A(k)$. If $L \in \text{BH}_A(k)$, then $L = D(L_1, \ldots, L_k)$ for $L_1, \ldots, L_k \in m$-1($A$). Define $L'_{i,j} \overset{\text{def}}{=} \bigcap_{j \leq i} L_j$. Then each $L'_{i,j} \in m$-1($A$), since $m$-1($A$) is closed under intersection, and $L'_i \supseteq L'_j \supseteq \ldots \supseteq L'_k$. 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 contains $x$ is odd. Since each $L'_i$ is in m-1($A$), given a string x, we can reduce x to $(q_1, \ldots, q_k)$ such that $x \in L'_{i,j} \iff q_i \in A$. Then $x \in L \iff (q_1, \ldots, q_k) \in \text{ODD}_k(A)$. Therefore $\text{ODD}_k(A)$ is $\leq^P_m$-hard for $\text{BH}_A(k)$.

To see that $\text{ODD}_k(A) \in \text{BH}_A(k)$, for each $i \leq k$, consider the set
\[ L_i = \{(x_1, \ldots, x_k) \mid \|\{x_1, \ldots, x_k\} \cap A\| \geq i\}. \]

\(L_i \in m(A)\) since given \((x_1, \ldots, x_k)\), we can generate all \(s = \binom{k}{i}\) subsets of \(i\) strings and use the reduction from \(OR, AND\) to \(A\) to generate a string that is in \(A\) iff all \(i\) strings in one of the subsets are in \(A\). Then \(ODD_k(A) = D(L_1, \ldots, L_k)\), implying \(ODD_k(A) \in BH_A(k)\). \(\square\)

The most difficult proof in this section is the argument that \(EVEN_k(A) \oplus ODD_k(A)\) is \(\leq^P_m\)-complete for \(P^{A[k]}\). We prove this by showing that the mind-change technique used by Beigel [Bei87] works provided \(A\) has \(AND_2\) and \(OR_2\).

**Theorem 17** If \(A\) has \(AND_2\) and \(OR_2\), then \(EVEN_k(A) \oplus ODD_k(A)\) is \(\leq^P_m\)-complete for \(P^{A[k]}\).

**Proof:** Both \(EVEN_k(A)\) and \(ODD_k(A)\) are obviously in \(P^{A[k]}\), so we need to show that every set in \(P^{A[k]}\) can be reduced to \(EVEN_k(A) \oplus ODD_k(A)\).

Let \(L\) be a language in \(P^{A[k]}\) 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:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>0</td>
<td>acc</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>\ldots</td>
<td>1</td>
<td>rej</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>\ldots</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>\ldots</td>
<td>1</td>
<td>rej</td>
</tr>
</tbody>
</table>

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 *valid* if

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

2. 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.)

3. 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. The following is an example of a valid truth table:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>rej</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>acc</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>acc</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>rej</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>acc</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>rej</td>
</tr>
</tbody>
</table>

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 acc. 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.
\[ T = \{(T, x, s) \mid T \text{ is a valid truth table for } M^A(x) \text{ and } m_T = s\}. \]

Claim: \( T_{\leq m}^A \). 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 1 and 2 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 \( A \). Otherwise, \( T \) is a valid truth table if and only if all the \( q_i \)'s with a 1 in the last row are actually in \( A \). This last condition can be reduced to \( A \) via the reduction from \( \text{AND}_k(A) \) to \( A \).

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

Claim: \( \mathcal{ET}_{\leq m}^r \). 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 truth tables \( T \) and all values of \( s \) between 0 and \( r \). Since \( x \in \mathcal{ET}_r \) if and only if \( \langle T, x, s \rangle \in T \) for one of the \( (T, s) \) pairs generated and since \( T_{\leq m}^A \), \( \mathcal{ET}_r \) can be reduced to \( A \) by using \( \text{OR}_2(A) \) to combine all the reductions from \( T \) to \( A \) into one reduction.

We use the reduction from \( \mathcal{ET}_r \) to \( A \) to produce a reduction from \( L \) to \( \text{EVEN}_k(A) \oplus \text{ODD}_k(A) \). This reduction will simply print out a \( k \)-tuple \( (z_1, \ldots, z_k) \) and an extra bit to indicate if the reduction is to \( \text{EVEN}_k(A) \) or \( \text{ODD}_k(A) \). 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 \( i \) mind changes. Let \( t \) be the maximum number of mind changes. Then, \( z_1, \ldots, z_t \in A \) and \( z_{t+1}, \ldots, z_k \notin A \). So, the parity of the number of \( z_i \)'s in \( A \) is the same as the parity of \( t \).

Now, we claim that the parity of the maximum number of mind changes is enough to determine if \( M^A(x) \) accepted. 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 \( t \) mind changes. Note that if \( t \) 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 \( t \) is even iff \( (z_1, \ldots, z_k) \in \text{EVEN}_k(A) \). Similarly, if the result in the first row is "reject", then \( x \in L \) iff \( (z_1, \ldots, z_k) \in \text{ODD}_k(A) \). Since the result in the first row can be computed in polynomial time, a polynomial time function can reduce \( L \) to \( \text{EVEN}_k(A) \oplus \text{ODD}_k(A) \).

\[ \square \]

**Theorem 18** If \( A \) has \( \text{AND}_2 \) and \( \text{OR}_2 \), then
\[ \text{BH}_A(k) \cup \text{co-BH}_A(k) \subseteq P^{A[[k]]} \subseteq \text{BH}_A(k + 1) \cap \text{co-BH}_A(k + 1). \]

**Proof:** The first containment follows straight from the definitions of the various classes.

The second containment holds because \( \text{EVEN}_k(A) \oplus \text{ODD}_k(A) \leq_m \text{ODD}_{k+1}(A) \) and the completeness results of the previous two theorems.

\[ \square \]

**Theorem 19** If a set \( A \) has \( \text{AND}_2 \) and \( \text{OR}_2 \), then \( P^{A[[k]]} = P^{A[[2^k-1]]} \).
Proof: For any $A$, $P^{A[k]} \subseteq P^{A[2^k-1]}$, so all we need to show is $P^{A[2^k-1]} \subseteq P^{A[k]}$. We will do this by showing that $ODD_{2^k-1}(A) \in P^{A[k]}$ and then Theorem 17 implies that $P^{A[2^k-1]} \subseteq P^{A[k]}$.

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 $OR_r(AND_r(A))$ 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, $P^{A[k]} = P^{A[2^k-1]}$. □

In [Cha89], Chang erroneously stated that for all $A$, if $P^{A[2]} = P^{A[2^2]}$, then the entire query hierarchy based on $A$ collapses to $P^{A[2]}$. This is only known to be true when $A$ has AND$_2$ and OR$_2$.

Theorem 20 If $A$ has AND$_2$ and OR$_2$ and $P^{A[2]} = P^{A[2^2]}$, then for all $j > k$, $P^{A[k]} = P^{A[j]}$.

Proof: It suffices to show that under these assumptions, $P^{A[2^2]} = P^{A[2^2]}$. Since $A$ has AND$_2$ and OR$_2$, by Theorem 17, $EVEN_{2^k+1}(A) \oplus ODD_{2^k+1}(A)$ is $\leq_m$-complete for $P^{A[2^2]}$. However, $P^{A[2]} = P^{A[2^2]}$, so there is some $P^{A[2]}$ computation for $EVEN_{2^k+1}(A) \oplus ODD_{2^k+1}(A)$. This means that a $P^{A[2]}$ computation can accept $EVEN_{k+2}(A) \oplus ODD_{k+2}(A)$ by 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 $EVEN_{k+2}(A) \oplus ODD_{k+2}(A)$ is $\leq_m$-complete for $P^{A[2]}$, $P^{A[2]} = P^{A[2^2]}$. □

The upward collapse properties of the BH$_A$ and QH$_A$ follow from Theorem 20.

References


