# How Reductions to Sparse Sets Collapse the Polynomial-time Hierarchy: A Primer

Paul Young

Technical Report 93-03-07
March, 1993

Department of Computer Science and Engineering, FR-35
University of Washington
Seattle, WA 98195

# How Reductions to Sparse Sets Collapse the Polynomial-time Hierarchy: A Primer

Paul Young*

## 1 Introduction

In [KL-80] it is proved

**(KL-1)** If $SAT \leq_T^P S$ for some *sparse* set $S$, then the polynomial time hierarchy collapses to $\Sigma_2 \cap \Pi_2$.

**(KL-2)** If $SAT \leq_T^P S$ for some *sparse* set $S$, then for every set $A$ in the polynomial time hierarchy, there is *some* sparse set $S_A$ such that $A \leq_T^P S_A$.

Using the well-known (and easy) result that $A$ is $\leq_T^P$ reducible to a sparse set if and only if $A$ can be solved with polynomial size circuits,[1] (KL-2) may be read as saying that if $SAT$ has polynomial size circuits, then every set in the polynomial-time hierarchy has polynomial size circuits.

*Department of Computer Science and Engineering, University of Washington. This expository paper is based on lectures given while the author was a Visiting Professor in Dipartimento di Scienze dell'Informazione, Universitá degli Studi di Milano in November and December, 1991. The author is grateful for the support given by Consiglio Nazionale delle Ricerche. Presented also at the $4^{th}$ European Summer School in Logic, Language and Information, Essex, England, August 1992. *This revision of March, 1993*, unifies the two separate parts of the paper which appeared in SIGACT News [Yo-92a], [Yo-92b], and corrects a number of inaccurate statements involving Kadin's theorem which appeared in [Yo-92a].

[1] If we start with a deterministic Turing machine working in polynomial time, then standard proofs of Cook's Theorem (see e.g., [MY-78]) build Boolean formulas (i.e., circuits) for mimicking the computation. Aside from the inputs, the circuits are uniform for all inputs of any fixed length. Since these circuits are, *a priori*, of polynomial size, if the Turing machine also has access to a sparse oracle then those polynomially many oracle elements accessible from inputs of length $n$ can also be coded directly into the circuits, in effect making the circuits contain the relevant parts of the sparse oracle. Thus, any set reducible to a sparse set can be solved with polynomial size circuits. Conversely, if we have polynomial sized circuits each of which solve all membership questions "$x \in A$?" for inputs of lengths $n$, then the totality of all such circuits forms a sparse set, and a Turing machine with access to these circuits can solve $A$.

It has long been known that if $SAT$ is reducible to a sparse set by more restrictive polynomial-time reductions, then even more dramatic collapses of the polynomial-time hierarchy must occur. Recently, ([OW-91]), Ogiwara and Watanabe proved that if $SAT \leq^P_{btt} S$ for some sparse set $S$, then $P = NP$, a result which subsumed all earlier results on polynomial-time bounded truth-table and many-one reductions of $SAT$ to sparse sets.

It is the purpose of this paper to give simple proofs, in a uniform format, of the major known (pre-1992) results relating how polynomial-time reductions of $SAT$ to sparse sets collapse the polynomial-time hierarchy. To help the reader familiar with basic facts of complexity theory follow the main flow of ideas, while keeping the exposition self-contained, straight forward proofs from elementary complexity theory are relegated to footnotes. We treat polynomial-time Turing reductions (i.e., Cook reductions) in Section 2. Bounded truth-table reductions (and many-one reductions) are treated in Section 3. Sections 2 and 3 may be read independently of each other. Section 4 uses the definitions of Section 3 to give simple proofs of results on conjunctive and disjunctive reductions. A comprehensive discussion of early work on how reductions to sparse sets collapse the polynomial-time hierarchy may be found in [Ma-89]. Additional discussions of this topic, as well as extensive bibliographies, may be found in [JY-90] and in [Yo-90].

# 2 Polynomial-Time Turing Reductions

## 2.1 Introduction

In [Lo-82], Long *explicitly used* the result (KL-1) to prove:

**(Lo-1)** If $SAT \leq^P_T S$ for some *sparse* set $S$, then *every* set in the polynomial-time hierarchy reduces to *the very same* sparse set, $S$.

I.e., if $NP$ can be solved with only a sparse amount of "free advice," then that *same* small amount of advice is sufficient to solve the entire polynomial-time hierarchy. This result is not only important in its own right, but it has interesting consequences.

For example:

**(Lo-2)** If $SAT \leq_T^P S$ for a *sparse* set $S \in \Delta_2$, then the polynomial-time hierarchy collapses to $\Delta_2$.

In [Ka-87], Kadin proved a variation of (Lo-2):

**(Ka-1)** If $\overline{SAT} \leq_T^{NP} S$ for a *sparse* set $S \in NP$, then the polynomial-time hierarchy collapses to $\Theta_2$. [2]

We give a simple proof of Kadin's Theorem in somewhat sharper form:

**(Theorem 3)** If $\overline{SAT} \leq_T^{NP} S$ for a *sparse* set $S \in NP$, then the polynomial-time hierarchy collapses to a subclass of $\Theta_2$ which we call *simple parity,* and which we denote by $\oplus_2^{simple}$.

Simple parity is the class of sets, $A$, reducible to $SAT$ by a reduction which *nonadaptively* produces a polynomial list of questions $q_1?, q_2?, \ldots,$ $q_{p(|x|)}?$ to $SAT$. The answers are guaranteed to be a string of "yes" answers followed by a string of "no" answers, with $x \in A$ *iff* the number of "yes" answers to "$q_i \in SAT?$" is even. Since the queries must be produced before any of the answers are received, and because the string of answers is guaranteed to be of the form $11\ldots1100\ldots00$ and of polynomial length, the parity of the string can be checked in polynomial time with logarithmically many queries to $SAT$ by simply doing a binary search on the questions "$q_i \in SAT?$" without ever producing the complete string of answers. Clearly $\oplus_2^{simple} \subseteq \Theta_2$.[3]

Our intent in this section is to give a simple, uniform, exposition of the Karp-Lipton, Long, and Kadin results. Thus we intend that this section of the paper contain simple proofs of all known major results on how general polynomial-time Turing reductions of $SAT$ to sparse sets force collapses of the polynomial-time hierarchy.

---

[2] $\Theta_2$ is the collection of sets reducible to $SAT$ via polynomial-time Turing reductions which make only a *logarithmic* number of calls to $SAT$. We write $\leq_T^{Plog}$ for this reducibility.

[3] By [KSW-87], certain parity sets are complete for $\Theta_2$. Harry Buhrman has pointed out (personal communication) that it follows from these results that $\oplus_2^{1-simple} = \Theta_2$, and Richard Beigel has pointed out that this equality is explicitly proved in [BH-91]. (Buss and Hay [BH-91] call what we call *simple-parity reductions* "polynomial time normalized reductions" and Wagner, [Wa-88], calls them "truth-table Hausdorf reductions.") Thus, although we do not give the proof, Kadin's Theorem does imply Theorem 3.

## 2.2 Basic Definitions.

We assume the reader familiar with the notion of both deterministic and nondeterministic Turing reductions which operate in polynomial-time. We denote these reducibilities by $\leq_T^P$ and by $\leq_T^{NP}$, respectively. $\leq_T^{P_{log}}$ is exactly like $\leq_T^P$, except that only logarithmically many queries to the oracle are allowed in the reduction.

**Definition 1** *We say that a set $A$ is simple-parity reducible to a set $B$, and write $A \leq^P_{simple-\oplus} B$, if, given $x$, we can in polynomial time produce polynomially many queries $q_1?, q_2?, \ldots, q_{p(|x|)}?$, the answers to $q_1 \in B?, q_2 \in B?, \ldots, q_{p(|x|)} \in B?$, are guaranteed to be monotonically nonincreasing, and $x \in A$ if and only if the number of successful queries is even.*[4]

**Definition 2**

- $\Sigma_0 = \Pi_0 = \Delta_0 = P$, *(where $P$ is all polynomial-time sets)*
- $\Sigma_{i+1} = \{A \mid A \leq_T^{NP} B \text{ for some } B \in \Sigma_i\}$
- $\Delta_{i+1} = \{A \mid A \leq_T^{P} B \text{ for some } B \in \Sigma_i\}$
- $\Pi_{i+1} = \text{co}-\Sigma_{i+1} = \{\overline{A} \mid A \in \Sigma_{i+1}\}$
- $PH = \cup_i \Sigma_i$
- $\Theta_{i+1} = \{A \mid A \leq_T^{P_{log}} B \text{ for some } B \in \Sigma_i\}$
- $\oplus_{i+1}^{simple} = \{A \mid A \leq^P_{simple-\oplus} B \text{ for some } B \in \Sigma_i\}$

**Elementary Facts**

1. $\oplus_i^{simple}$, $\Delta_i$, and $\Theta_i$ are all closed under complement
2. $PH = \cup_i \Sigma_i = \cup_i \Pi_i = \cup_i \Delta_i \subseteq PSpace$
3. $\oplus_i^{simple} \subseteq \Theta_i \subseteq \Delta_i \subseteq \Sigma_i \cap \Pi_i \subseteq \Pi_i \subseteq \oplus_{i+1}^{simple}$
4. $\Sigma_i = \exists\forall\exists \ldots P$, and $\Pi_i = \forall\exists\forall \ldots P$, ($i$ quantifiers)[5]
5. $\exists\exists = \exists$ and $\forall\forall = \forall$[5]

[4] Note that $\leq^P_{simple-\oplus}$ is not obviously transitive, and so is not obviously a reducibility. But see [BH-91] and [Wa-88], as discussed in footnote 2. In any case, for our purposes, we do not need transitivity.

[5] In (4) and (5), $P$ is a polynomially-time-decidable predicate and the quantifiers "$\exists$" and "$\forall$" range over all elements whose length is bounded by some polynomial in the

**Definition 3** *A set $S$ is sparse if there is a nondecreasing polynomial bound $b$ such that for all $n$,*

$$|S_n| \ =_{def} \ |\{x \mid \ x \in S \text{ and } |x| \leq n\}| \ \leq \ b(n).$$

I.e., a set is sparse if, of all of the exponentially many elements of length less than or equal to $n$, at most a polynomial number are actually in $S$. In this sense, sparse sets don't have "very many" elements.

Finally, we take $N$ to be the universal set, either the set of all nonnegative integers or the set of all finite strings over some alphabet, as the reader pleases.

### 2.3 Turing Reductions of $SAT$ to Sparse Sets

**Theorem 1** [KL-80] If there exists a sparse set $S$ such that $SAT \leq_T^P S$, then $\Sigma_2 \subseteq \Pi_2$. Thus $PH \ = \ \Sigma_2 \cap \Pi_2$.

**Proof** We begin with an arbitrary set $A \in \Sigma_2$. Then by our definition of $\Sigma_2$ and by our hypothesis on $SAT$ and $S$,

$$A \ \leq_T^{NP} \ SAT \ \leq_T^P \ S.$$

Let $M_{SAT \to S}$ be the machine which deterministically decides $SAT$ when given $S$ as an oracle. Although $M_{SAT \to S}$ correctly decides $SAT$ given $S$ as oracle, we will be concerned about the behavior of $M_{SAT \to S}^F$ when $M_{SAT \to S}$ is given an oracle $F \neq S$. Standard techniques show that, without loss of generality, we can make several assumptions about the behavior of $M_{SAT \to S}$:

---

length of the instances in $P$. Thus the characterization of $\Sigma_1$ given by (4) is just the well-known characterization of $NP$ as sets definable from polynomial-time relations by one polynomially bounded existential quantifier. The characterization of $\Pi_1$ (i.e. of $co-NP$) then follows directly by complementation. Characterizations (4) and (5) are then proven simultaneously by induction on $i$, using essentially the same proof at the $i$-th level as is used to prove the base case $i = 1$.

The collapse of adjacent like quantifiers described in (5) follows in a standard, and straightforward, fashion by using polynomial-time computable pairing and projection functions.

- For *any* oracle $F$, if $M^F_{SAT\rightarrow S}(w)$ accepts, then $w \in SAT$.[6]

- There is some monotonically increasing polynomial bound $p_1(n)$ on the length of the elements of any $F$ which can be used in oracle appeals while computing $M^F_{SAT\rightarrow S}(w)$ on inputs $w$ of length $\leq n$.

- $M^{S_{p_1(|w|)}}_{SAT\rightarrow S}(w)$ rejects implies that $w \notin SAT$.[7]

Now let $M_{A\rightarrow SAT}$ be the machine which *non*deterministically decides $A$ when given $SAT$ as an oracle. Using standard techniques for dealing with nondeterministic computations, we can without loss of generality again make several assumptions about the behavior of $M_{A\rightarrow SAT}$:

- There is some monotonically increasing polynomial bound $p_0(n)$ on the length of the elements of any $F$ which can be used in oracle appeals while computing $M^F_{A\rightarrow SAT}(x)$ on inputs $x$ of length $\leq n$.

- For any oracle $F$, on any of the nondeterministic computation paths of $M^F_{A\rightarrow SAT}(x)$, the machine $M_{A\rightarrow SAT}$ makes exactly one query to the oracle, and that path *accepts* just if the answer to the query is "no."[8]

---

[6]Given the machine $M_{SAT\rightarrow S}$, we can use $M_{SAT\rightarrow S}$ in a more or less "equivalent" algorithm which partially "checks" its output in the following fashion: Given a formula $B(y_1, y_2, \ldots, y_n)$, see if $M^F_{SAT\rightarrow S}(B(y_1, y_2, \ldots, y_n))$ accepts. If it does not, "reject." If it does, see if $M^F_{SAT\rightarrow S}(B(0, y_2, \ldots, y_n))$ accepts. If it does, see if $M^F_{SAT\rightarrow S}(B(0, 0, \ldots, y_n))$ accepts, but if it does not, see if $M^F_{SAT\rightarrow S}(B(1, 0, \ldots, y_n))$ accepts. If $M^F_{SAT\rightarrow S}(B(1, 0, \ldots, y_n))$ does not accept, see if $M^F_{SAT\rightarrow S}(B(1, 1, 0 \ldots, y_n))$ accepts... Proceeding in this fashion, at most $n$ queries to $F$ lead us either to "reject" or to an assignment of "true" and "false" values to the variables in $B(y_1, y_2, \ldots, y_n)$. We can then check in polynomial time whether this assignment satisfies the formula $B$. If it does, "accept," else "reject". Note that if $F$ provides the same answers as does $S$, then this algorithm must be correct, but in any case it cannot lead to acceptance of a formula $B(y_1, y_2, \ldots, y_n)$ which is not satisfiable.

[7]Since the computation of $M^S_{SAT\rightarrow S}(w)$ cannot access any elements of the oracle $S$ of length $\geq p_1(|w|)$, obviously $M^{S_{p_1(|w|)}}_{SAT\rightarrow S}(w)$ and $M^S_{SAT\rightarrow S}(w)$ produce the same answers. (Note that if $M_0$ were a machine which decides $SAT$ given some "advice" $F$ which contains only logarithmically many bits, then the technique we have outlined here would enable us to actually solve $SAT$ in polynomial time, since in polynomial time we could cycle through all possible logarithmic advice sets, $F$, and we could be sure that if any such advice $F$ led us to accept some formula $w$, then we would really have $w \in SAT$, ([KL-80]).)

[8]Since this is a nondeterministic reduction, the machine $M_{A\rightarrow SAT}$ can, along each of its computation paths, guess its oracle queries and their answers. At the end of the path, it then needs to verify the correctness of these guesses, which are of the form "$< w_1, \ldots, w_k > \in SAT^k$?" and "$< y_1, \ldots, y_{k'} > \in \overline{SAT}^{k'}$?", with the path accepting

We now let $M_{A\to S}$ be the obvious composition of machines $M_{A\to SAT}$ and $M_{SAT\to S}$, which nondeterministically decides $x \in A$ by running machine $M_{A\to SAT}$ nondeterministically on input $x$ but replacing any oracle call "$w \in SAT$?" by a deterministic calculation of $M^S_{SAT\to S}(w)$, using $S$ as oracle. We now make some straightforward observations:

1. $p_1(p_0(|x|))$ serves as a polynomial bound on the length of the elements of $S$ which can be used in oracle appeals while computing $M^S_{A\to S}(x)$.

2. For any input $x$, $M^S_{A\to S}(x)$ and $M^{S_{p_1(p_0(|x|))}}_{A\to S}(x)$ produce the same answers. Therefore:

3. If $M^{S_{p_1(p_0(|x|))}}_{A\to S}(x)$ accepts, then $x \in A$.[9]

4. $|S_{p_1(p_0(|x|))}| \leq b(p_1(p_0(|x|))$

5. For *any* set $F$, $M^F_{A\to S}(x)$ can produce a wrong answer to "$x \in A$?" *only* if for some query $w$, $M^F_{SAT\to S}(w)$ produces a wrong answer to "$w \in SAT$?" This can occur *only* if $M^F_{SAT\to S}(w)$ falsely rejects, and the only error this can produce is for the computation of $M^F_{A\to SAT}(x)$ to falsely accept. Therefore:

6. For any $F$, if $M^F_{A\to S}(x)$ rejects then $x \notin A$.

Putting (3) and (6) together we see that

$$x \in A \iff (\forall F)_{|F|\leq b(p_1(p_0(|x|)) \ \&\ F\subseteq N_{p_1(p_0(|x|))}}[M^F_{A\to S}(x) \text{ accepts}].$$

But this describes the predicate $x \in A$ in $\forall\exists$ format. Since $A$ was an arbitrary set in $\Sigma_2$, we have thus shown that $\Sigma_2 \subseteq \Pi_2$.

---

just if both answers are "yes." Because $SAT$ and $\overline{SAT}$ are reducible to their conjunctive closure, each of these questions can be regarded as a single query, first to $SAT$, and then to $\overline{SAT}$. But since this is a nondeterministic reduction, the appeal to $SAT$ can be verified by further nondeterministic extensions of the computation path, carrying the query to $\overline{SAT}$ to the end of the extended paths. If the extended path accepts or rejects without appeal to $\overline{SAT}$ as oracle, we can always insert some trivial appeal which produces the "right" answer, (provided the oracle really is $SAT$). Putting all of this together, we see that we can assume without loss of generality that the nondeterministic machine $M_{A\to SAT}$ on each path of its computation tree makes exactly one oracle query and accepts along that path if and only if the reply to the query is "no."

[9] Observation 6 below shows that a very strong form of the converse of this observation is also true.

Standard quantifier reduction techniques now yield $PH \subseteq \Sigma_2 \cap \Pi_2$.[10] $\Box$

We now wish to prove Long's Theorem, which asserts that if $SAT \leq_T^P S$ with $S$ sparse, then *every* set in the polynomial-time hierarchy is reducible to $S$. In doing so, we adopt all of the definitions and results from the proof of the Karp-Lipton Theorem. Obviously, from the Karp-Lipton Theorem, it suffices to prove that $A \in \Sigma_2$ implies $A \leq_T^P S$. One more definition is useful:

**Definition 4** *We say that $BAD(x, F, w)$ if the computation $M_{A \to S}^F(x)$ calls $w$ and $w \in SAT$, but when $M_{SAT \to S}^F(w)$ is called it rejects $w$.*

Note that the *only* way that $M_{A \to S}^F(x)$ can fail to give the same answer as $M_{A \to S}^S(x)$ is for $M_{SAT \to S}^F(w)$ to fail to correctly decide $SAT$ and the *only* way this can happen is for $M_{SAT \to S}^F(w)$ to reject some $w \in SAT$. I.e. the *only* way that $M_{A \to S}^F(x)$ fails to correctly decide "$x \in A$?" is if $(\exists w) BAD(x, F, w)$. Note also that $(\exists w) BAD(x, F, w)$ is an $NP$ predicate, and so can be decided in polynomial time by any set which is $NP$ hard.

**Lemma 1** *There is a polynomial-time procedure $FIND(x, F)$, which, given $S$ as oracle, decides whether $(\exists w) BAD(x, F, w)$ and, if such a $w$ exists, finds an example of such a $w$ and then returns with the set of all oracle queries made to $F$ in the execution of $M_{SAT \to S}^F(w)$.*

**Proof** The predicate $R(x, w, F)$ defined by $(\exists w' \neq \lambda) BAD(x, F, ww')$ is in $NP$, and hence can be decided in polynomial time, given $S$ as oracle. Furthermore, there is a polynomial bound on the length of any $w$ satisfying $BAD(x, F, w)$.

Hence, given an oracle $S$ for $SAT$ and given $x$ and $F$, if $(\exists w) BAD(x, F, w)$ we can (using essentially the same technique as outlined in footnote 5), by starting with $w$ as the empty string, $\lambda$, in polynomial time iteratively extend $w$ to a (nonextendable) string $w$ satisfying $BAD(x, F, w)$. Given such a $w$, we can then run $M_{SAT \to S}^F(w)$ to find the set of all elements queried in this computation. $\Box$

Note for future use that if $BAD(x, F, w)$ holds and if $F \subseteq S$ then some elements queried in the computation of $M_{SAT \to S}^F(w)$ must be in $S - F$.

[10] By complementation of $\Sigma_2 \subseteq \Pi_2$, one obviously gets $\Pi_2 \subseteq \Sigma_2$. From this we see that if we have an arbitrary set in the polynomial-time hierarchy described (Elementary Fact 4) by more than two alternating quantifiers, the inner-most two quantifiers can be interchanged. This produces two adjacent quantifiers of the same type, which by Elementary Fact 5 can be collapsed, producing a predicate with one fewer quantifier. Continuing in this fashion eventually reduces the total number of alternating quantifiers to two, which as we've just seen can be written in either order.

**Theorem 2 [Lo-82]** *Suppose that SAT* $\leq_T^P$ *S with S sparse. Then every set A in the polynomial-time hierarchy is* $\leq_T^P$ *reducible to S.*

**Proof** Our objective is, given any input $x$ to $M_{A\rightarrow S}(x)$, to systematically, in polynomial time, build up a subset $F \subseteq S_{p_1(p_0(|x|))}$ such that the computations of $M_{A\rightarrow S}^{F}(x)$ and of $M_{A\rightarrow S}^{S_{p_1(p_0(|x|))}}(x)$ are the same. But as we have just seen at the end of the last proof, this is guaranteed for $F$ if there is no $w$ such that $BAD(x, F, w)$. Consider now the following algorithm, which assumes $S$ as oracle:

Begin by initializing $F$ to be the empty set.
While $(\exists w)BAD(x, F, w)$
$F := F \cup [FIND(F, x) \cap S]$.

Obviously, $F \subseteq S_{p(|x|)}$ is an invariant of the loop, so that $F$ remains polynomially bounded in size. But we have already seen that $(\exists w)BAD(x, F, w)$ implies that $FIND(x, F)$ must produce some query in $S - F$. Consequently, after a polynomial number of steps, the algorithm *must* stop with some $F \subseteq S_{p(|x|)}$ for which there is no query $w$ for which $BAD(x, F, w)$.

But this means that for *this* $F$ we have that $x \in A$ if and only if $M_{A\rightarrow S}^{F}(x)$ accepts. Since

$$\{< x', F' > \mid M_{A\rightarrow S}^{F'}(x') \text{ accepts}\}$$

is an $NP$ set, $S$, (which is an oracle for $SAT$) can decide whether $M_{A\rightarrow S}^{F}(x)$ accepts. □

**Corollary 1** [Implicit in **Lo-82**] *If there is a sparse set S in* $\Delta_2$ *with SAT* $\leq_T^P$ *S, then PH* $\subseteq \Delta_2$.

**Proof.** By Long's theorem, every set $A \in PH$ is $\leq_T^P$ to the fixed set $S \in \Delta_2$. Since $\Delta_2$ is closed under $\leq_T^P$, the corollary is immediate. □

**Corollary 2 [Ma-82]** *If there is a sparse set in NP with SAT* $\leq_T^P$ *S, then PH* $\subseteq \Delta_2$.

**Proof** $NP \subseteq \Delta_2$. □

A key feature of the proof of Long's Theorem is that, while most of the predicates are in $NP$, $S$ is not, and so we still need to use $S$ as oracle to test $F \subseteq S$. However if $S$ itself is in $NP$, then the construction simplifies, and $F$ need not even be explicitly found, giving us Kadin's improvement of Mahaney's Theorem:

**Theorem 3 [Ka-87]** *If there is a sparse set $S$ in $NP$ with $\overline{SAT} \leq_T^{NP} S$, then $PH \subseteq \oplus_2^{simple} \subseteq \ominus_2$.*

**Proof** As we shall later see, given any set $A$ in $\Sigma_2$, our assumptions on $\overline{SAT}$ guarantee that there is a nondeterministic machine $M_{A \to S}$ which, given $S$ as oracle, nondeterministically accepts $A$. Define

$$\begin{aligned} SMALL(x,m) &=_{def} (\exists F)[F \subseteq S_{p(|x|)} \ \&\ |F| \geq m], \quad \text{and define} \\ ACCEPT(x,m) &=_{def} [SMALL(x,m+1) \ \vee \\ &\qquad (\exists F)[F \subseteq S_{p(|x|)} \ \&\ |F| = m \ \&\ M_{A \to S}^F(x) \text{ accepts}]. \end{aligned}$$

Since $S$ is sparse and in $NP$, the predicates $SMALL$ and $ACCEPT$ are also in $NP$, and therefore by Cook's Theorem there are polynomially computable functions $g_0$ and $g_1$ such that

$$\begin{aligned} ACCEPT(x,m) &\iff g_0(x,m) \in SAT, \quad \text{and} \\ SMALL(x,m) &\iff g_1(x,m) \in SAT. \end{aligned}$$

Now notice that if $m_{census} = |S_{p(|x|)}|$, then

- $SMALL(x,m) \iff m \leq m_{census}$.
- The *only* way to satisfy $ACCEPT(x, m_{census})$ is by taking $F = S_{p(|x|)}$ and having $M_{A \to S}^F(x)$ accept.

Thus the answers to $g_0(x,0) \in SAT?$, $g_1(x,0) \in SAT?$, $g_0(x,1) \in SAT?$, $g_1(x,1) \in SAT?$, $\ldots, g_0(x, b(p(|x|)) \in SAT?$, $g_1(x, b(p(|x|))) \in SAT?$, form a monotonically nondecreasing sequence, and we have that $x \in A$ if and only if the number of "yes" answers is even. Thus $A \in \oplus_2^{simple}$, so

$$\Sigma_2 \quad \subseteq \quad \oplus_2^{simple} \quad \subseteq \quad \ominus_2 \quad \subseteq \quad \Pi_2.$$

By Theorem 1, it thus follows that $PH \subseteq \oplus_2^{simple} \subseteq \ominus_2$. [11]

[11] Of course, from footnote 2, we know that $\oplus_2^{simple} = \ominus_2$.

To complete the proof, we must merely verify that given any set $A$ in $\Sigma_2$, there is a nondeterministic machine $M_{A \to S}$ which, given $S$ as oracle, nondeterministically accepts $A$.

Since $A \in \Sigma_2$, there is a nondeterministic Turing machine, $M_{A \to SAT}$ which, given $SAT$ as oracle accepts $A$. By our hypothesis, there is also a nondeterministic machine $M_{\overline{SAT} \to S}$ which, given $S$ as oracle, accepts $\overline{SAT}$. If we replace all oracle calls of $M_{A \to SAT}$ asking whether some $w \in SAT$, by, concurrent nondeterministic calls to $SAT$ with no oracle, and to $M_{\overline{SAT} \to S}$ with $S$ as oracle, then some nondeterministic path of *one* of these concurrent calls must successfully tell us whether $w \in SAT$, enabling us to proceed with the nondeterministic calculation of $M_{A \to SAT}$. Clearly, this gives us a nondeterministic calculation accepting $A$, given $S$ as oracle. □

There is a pleasing symmetry to Long's Corollary: the collapse of $PH$ to $\Delta_2$ can be accomplished by requiring only that the sparse set $S$ be in $\Delta_2$. On the other hand, for Theorem 3 to get a collapse to $\oplus_2^{simple}$, we required that the sparse set be all the way down in $NP$. It is therefore tempting to conjecture that there should be some reasonable class $C$ someplace between $NP$ and $\Delta_2$ such that $PH$ collapses to $C$ provided the sparse set is in $C$. To get this result it would suffice to have $NP \subseteq C$, $C$ contains a set which is $\leq_m^P$-complete for $C$, and "$x \in S$?" in $C$ implies that "$F_x \subseteq S$?" is in $C$, where $F_x$ ranges over all sets which are polynomial in $|x|$. All of the classes $\Sigma_i$, $\Pi_i$, and $\Delta_i$, satisfy these conditions, but the class $\oplus_2^{simple}(\Theta_2)$ seems not to, and we know of no *interesting* class between $NP$ and $\Delta_2$ which satisfies the last of these conditions.

# 3 Bounded Truth-Table Reductions

**Definition 5**

- *A set $A$ is $k$-bounded truth-table reducible to a set $B$ ($A \leq^P_{k-tt} B$) if, given $x$, in polynomial time we can compute a truth-table $T_x(z_1, \ldots, z_k)$ of $k$ boolean variables, and a list $< x_1, \ldots, x_k >$ and then know that*

$$x \in A \iff T_x(C_B(x_1), \ldots, C_B(x_k)).^{12}$$

- *The truth-table reduction is said to be via a* fixed *truth-table if the truth-table $T_x$ is independent of $x$. I.e., the same truth-table is used for all queries $< x_1, \ldots, x_k >$.* [13]

- *A set $A$ is bounded truth-table reducible to a set $B$ ($A \leq^P_{btt} B$) if $A \leq^P_{k-tt} B$ for some $k$.*[14]

Despite considerable research on restricted polynomial-time truth-table reductions ([Uk-83], [Ya-83], [Ye-83], [BK-87], [Wa-87], [Ko-88], [TB-88], [Wa-88]), prior to the recent Ogiwara-Watanabe results [OW-91], only very limited results have been obtained concerning the implications of restricted truth-table reductions of sets in *NP* and *coNP* to sparse sets. The following theorem summarizes most earlier (pre 1992) known results on such collapses within the polynomial-time hierarchy.

**Theorem 4** *Restricted truth-table reductions to sparse sets.*[15]

[12] $C_B$ is the characterisitic function of the set $B$.

[13] $SAT \leq^P_m S$ iff $SAT \leq^P_{1-tt} S$ via a *fixed positive* one-truth-table reduction. $\overline{SAT} \leq^P_m S$ iff $SAT^P_{1-tt}S$ via a *fixed negative* one-truth-table reduction.

[14] Bounded truth-table reductions are the basic reductions used to define Boolean hierarchies, which have recently been investigated by surprisingly large groups of authors, ([CGHHSWW-88], [CGHHSWW-89], [BBJSY-89],

[15] Conjunctive truth-tables, ($\leq^P_{conj}$), are truth-tables which are given by a simple conjunction of the variables (with no negations). Similarly, disjunctive truth-tables use only disjuncts of the variables. Conjunctive and disjunctive truth-tables are each special cases of *positive*, ($\leq^P_{pos-tt}$), truth-tables, which are monotonic in the sense that the output of the truth-table can never be changed from "true" to "false" by changing one of the variable's answers from "false" to "true." Positive bounded truth-tables, ($\leq^P_{pos-btt}$), are simply positive truth-tables which are bounded. Thus, if Yesha's results held for $\leq^P_{btt}$ instead of just for $\leq^P_{pos-btt}$, they would strictly imply the results of Ukkonen and Yap.

- **Fortune-79**
  *If coSAT $\leq^P_m$-reducible to a sparse set, then P = NP.*

- **Mahaney-82**
  *If SAT $\leq^P_m$-reducible to a sparse set, then P = NP.*

- **Yesha-83**
  *If coSAT $\leq^P_{pos-btt}$-reducible to a sparse set, then P = NP.*

- **Yesha-83**
  *If SAT $\leq^P_{pos-btt}$-reducible to a sparse set in NP then P = NP.*

- **Watanabe-89**
  *If every set in $D^P$ is $\leq^P_{1-tt}$-reducible to a sparse set, then P = NP.*

- **Ukkonen-Yap-83**
  *If coSAT $\leq^P_{conj}$-reducible to a sparse set, then P = NP.*

- **Yap-83**
  *If SAT $\leq^P_{conj}$-reducible and $\leq^P_{disj}$-reducible to a sparse set in NP, then P = NP.*

It is interesting to note that Watanabe's result is the only result in the above group to allow nonpositive reductions. Nevertheless, his proof is still a fairly direct adaptation of Fortune's tree pruning argument.

In [OW-91], Ogiwara and Watanabe proved a result which superceded all of the above results on how many-one and *bounded* truth-table reductions of *SAT* to sparse sets force collapses of the polynomial-time hierarchy. They proved that if $SAT \leq_{btt} S$ for some sparse set $S$, then $P = NP$.

It is our intent to give a simple proof of the Ogiwara-Watanabe Theorem.[16] After we give this proof, in Section 4 of this paper we shall explain how an even simpler version of this method can be adapted to also prove both the

A *one-truth-table*, ($\leq^P_{1-tt}$), or more generally a *k-truth-table*, ($\leq^P_{k-tt}$), is one in which a maximum of $k$ questions can be asked. Thus, $\leq^P_m$ is just the general form of a *positive* $\leq^P_{1-tt}$-reduction. As introduced by Papadimitriou and Yannakakis, $D^P$ is the class of sets which can be described as the intersection of a set in *NP* and a set in *coNP*. Thus, it sits just above *NP* and *coNP* in the Boolean hierarchy over *NP*.

[16] In [HL-91], Homer and Longpré give a proof which is simpler than the original proof in [OW-91]. While our proof seems even simpler than the Homer-Longpré proof, both the Homer-Longpré and the Ogiwara-Watanabe algorithms yield sharper time bounds than the algorithm we give.

Ukkonen-Yap and the Yap results on unbounded truth-tables, and in fact give a stronger version of the Yap-83 result.

The proofs we give are best visualized as breadth-first prunings of self-reduction trees for $SAT$, and we begin with the necessary elementary definitions of these concepts.

**Definition 6** *Let $B(z_1, \ldots, z_k)$ be a a boolean formula, with $k$ boolean variables $z_1, \ldots, z_k$. The daughters of $B(z_1, \ldots, z_k)$ are defined to be the two formulas $B(0, z_2, \ldots, z_k)$ and $B(1, z_2, \ldots, z_k)$. $B(0, z_2, \ldots, z_k)$ lies to the left of $B(1, z_2, \ldots, z_k)$. Note that any formula with at least one free variable is satisfiable if and only if at least one of its daughters is satisfiable.*

*The self-reduction tree for the formula $B(z_1, \ldots, z_k)$, begins with the formula $B(z_1, \ldots, z_k)$ at the root, and successively adds daughters until no formulas with free variables remain. The leaves of the tree thus have no free variables, and the instantiation of the variables along the leaves give all possible assignments which might satisfy the root, $B(z_1, \ldots, z_k)$.*

*If node $n$ has daughters $n_0$ and $n_1$ with $n_0$ to the left of $n_1$, then all descendants of $n_0$ are said to be to the left of all descendants of $n_1$. If we have two nodes, $n$ and $n'$ at the same level of the tree, we write $n \prec n'$ if $n$ lies to the left of $n'$.*

Our goal in the proofs which follow will always be to do a breadth-first search of the self-reduction tree for a boolean formula $B(z_1, \ldots, z_k)$, but to prune the tree as we go along so that every level of the tree has at most polynomially many elements. Note that in the full, unpruned tree there is a doubling of nodes as we proceed from one level of the tree to the next. As we proceed down the tree pruning each level, once a given level, $l$, is pruned, then for all remaining nodes we next add all daughter nodes to the tree, doubling the nodes at the next level, $l'$, of the tree before pruning them back to our polynomial bound. As we do this, we may well read and process the entire (doubled) set of nodes in level $l'$ before cutting level $l'$ back to our original polynomial bound. This enables the entire process to remain polynomial.

**Definition 7** *A tree-pruning algorithm for the self-reduction tree of a boolean formula $B(z_1, \ldots, z_k)$ is* satisfactory *if, at any step at which a node $n'$ is eliminated, there is a node $n$ at the same level and to the left of $n'$ for which we know that:*

- *node $n$ is not yet eliminated, and*
- *if $B(z_1, \ldots, z_k)$ has a satisfying assignment, then the left-most satisfying assignment lies on or to the left of $n'$ if and only if it lies on or to the left of node $n$.*

Clearly, if we use a satisfactory tree pruning algorithm to prune the self-reduction tree for a Boolean formula, then the formula has a satisfying assignment just if there is a satisfying assignment in one of the leaves remaining in the pruned tree. If the resulting tree has polynomial size, then all leaves can be checked in polynomial time to see if a satisfying assignment exists.

We obtain satisfactory self-reduction trees *for SAT* by considering the following variation of *SAT*:

**Definition 8** $L(SAT) = \{n \mid n$ *is a node in a self-reduction tree for some formula $B(z_1, \ldots, z_k)$ and a path to some satisfying assignment passes through or to the left of node $n$*$\}$.

Obviously $L(SAT)$ is in $NP$ so that $SAT \leq_{k-tt} S$ implies that

$$L(SAT) \leq_m SAT \leq_{k-tt} S$$

so

$$L(SAT) \leq_{k-tt} S.$$

It is *this* reduction which will be used to obtain satisfactory prunings of the self-reduction tree *for SAT*. $L(SAT)$ is a simple concept, but its introduction in [OW 90] provided a key new tool for studying self-reduction trees and reducibilities. (Note also for Section 4 that $SAT \leq_{conj} A$ implies $L(SAT) \leq_{conj} A$ and that $SAT \leq_{disj} A$ implies $L(SAT) \leq_{disj} A$.)

Our proof of the Ogiwara-Watanabe Theorem will be by induction on the number of variables of the truth-table, so we begin with the special case of one-truth-tables. The following points are worth noting for one-truth-tables:

- The proof seems much easier than any of the standard proofs of Mahaney's result that $SAT \leq_m^P$-reducible to a sparse set implies $P = NP$.

- The proof is a *uniform, simultaneous* proof of both the Mahaney and the Fortune Theorems.

**Proposition 1** *If $SAT \leq_{1-tt} S$ for some sparse set $S$ via a* fixed *one-truth-table, then there is a satisfactory tree pruning algorithm which prunes the self-reducibility trees for $SAT$ and solves $SAT$ in polynomial time. Furthermore:*

- *The algorithm is independent of the particular one-truth-table which accomplishes the reduction of $SAT$ to $S$.*

- *The algorithm prunes each level of the self-reducibility tree for a formula $w$ to have at most $2q(|w|) + 1$ nodes, where $q(|w|)$ bounds the maximum number of elements of $S$ that can be queried using truth-tables of size $|w|$.*

- *The time required for the algorithm is $O(P(|w|) * |w| * (2q(|w|) + 1))$, where $P(|w|)$ bounds the time required to compute the truth-table reductions on inputs of length $|w|$ and $q(|w|)$ bounds the maximum number of elements of $S$ that can be queried using truth-tables of size $|w|$.*

**Proof** For an input formula of length $m = |w|$, we prune the self-reduction tree for $SAT$ doing a breadth-first search, guaranteeing that at each level of the tree at most $2q(m) + 1$ nodes of the tree are retained.

Since we are assuming a one-truth-table reduction, at any node, $n$ of the tree we calculate a value $T(n)$ which indicates some query "$y \in S$?" in our reduction of $L(SAT)$ to $S$. As we do the breadth-first search from left-to-right, we label each node we reach with this value, $T(n)$. If two nodes, $n$ and $n'$ with $n \prec n'$ have the same label, the associated truth-tables must either both evaluate to "true" or must both evaluate to "false". In either case, we can clearly eliminate all nodes between $n$ and $n'$, and also eliminate the right most node, $n'$, since the left most satisfying assignment (if any) is at or to the left of node $n$ if and only if the left-most satisfying assignment is at or to the left of node $n'$. If at any point in this left to right breadth-first search for a level of the self-reduction tree we reach a node $n_1$ which gives us an accumulation of $q(m) + 1$ distinct labels, we stop the left-to-right breadth-first search of this level. Clearly at least one of the remaining

labeled nodes, say node $n_0$, with $n_0 \preceq n_1$, belongs to $\overline{S}$, although we may not know which one, and in this case further pruning is necessary. (If the process does *not* terminate before we traverse this level of the tree, then we have pruned this level of the tree to at most $q(m)+1$ nodes, and no further pruning is necessary.)

If the left-to right breadth-first search terminates because we accumulate $q(m)+1$ nodes, then we begin a right-to-left traversal of the tree at this level, labeling and eliminating nodes just as in the left-to-right traversal of the level. Again we stop if we reach a node $n_2$ which gives us an accumulation of $q(m)+1$ distinct labels in this right-to-left traversal of the level, or if we reach node $n_1$. If we reach node $n_1$, we will have accumulated in both our left-to-right and in our right-to-left searches at most $2q(m)+1$ nodes, and no further pruning is necessary. If we stop before reaching node $n_1$, we will have again found $q(m)+1$ distinct labels in this right-to-left traversal of the level. And again, clearly one of the nodes, $n_3$, with $n_2 \preceq n_3$, which we've labeled belongs to $\overline{S}$, although we cannot know which one. In this case we eliminate the nodes between $n_1$ and $n_2$. The justification for doing this is as follows:

Since $n_0$ and $n_3$ both belong to $\overline{S}$, the left most satisfying assignment (if any) lies to the left of $n_0$ if and only if it lies to the left of $n_3$. Thus, we can clearly safely eliminate all nodes between $n_0$ and $n_3$, including node $n_3$. Unfortunately, we do not know which of the labeled nodes are $n_0$ and $n_3$. But we do know that $n_0 \preceq n_1 \prec n_2 \preceq n_3$. Furthermore, both $n_1$ and $n_2$ *are* known since the left-to-right search stopped at $n_1$ and the right-to-left search stopped at $n_2$. Thus we can safely eliminate all nodes between $n_1$ and $n_2$, including $n_2$.

This procedure for eliminating nodes gives a self-reduction tree which has at most $2q(m)+1$ nodes at each level. Furthermore, the invariant for node elimination is that a node $n'$ is eliminated only if there is an identified node $n$ to the left of $n'$ which has the property that a left most satisfying assignment lies to the left of $n'$ if and only if a left most satisfying assignment lies to the left of node $n$. Therefore the procedure cannot eliminate the left-most satisfying assignment.

Since the pruned self-reduction tree has at most $m$ levels, it has at most $m*(2q(m)+1)$ nodes, so we can build the pruned tree and test whether any of the $2q(m)+1$ leaves is a satisfying assignment of the original formula in time which is approximately $P(m)*m*(2q(m)+1)$, where $P(m)$ is the time required to compute the truth-table tests on formulas of length $m$.

Obviously, the above algorithm is independent of the particular one-truth-table used to reduce $SAT$ to $S$. $\Box$

To prove the full Ogiwara-Watanabe Theorem, we assume inductively that we have an algorithm which, given any fixed truth-table of $k$ variables successfully prunes the self-reduction tree for $SAT$. Then, given a fixed truth-table of $k+1$ variables, we will prune the self-reduction tree for $SAT$ by successively trying all possible prunings obtained by fixing *each* of the $k+1$ variables to obtain *at least* $k+1$ different $k$-truth-tables, In doing this, we use first left-to-right and then right-to-left prunings of each level of the self-reduction tree. This process works essentially the way the left-to-right and right-to-left pruning does for the pruning given above for one-truth-tables. In the case of one-truth-tables, when the number of distinct labels got out of hand, we knew that we were making some query to a node ($n_0$ or $n_3$) in $\overline{S}$. In the case of $k+1$-truth-tables, when the number of distinct labels gets out of hand, we need to know that we were making some query to a node ($n_0$ or $n_3$) in $\overline{S}^{k+1}$. The following definitions and lemma set up the necessary simple combinatorial machinery. Throughout, we now let $S_q \subseteq N$ be a finite set, with $|S_q| \leq q$. Intuitively $S_q$ will be the set of elements of $S$ which can be queried using our presumed truth-table reduction of $SAT$ to $S$ starting with elements of $SAT$ of length $m$.

**Definition 9**

$$B(k) \quad = \quad k! * (2q+1)^k$$

(For later use, note that $B(0) = 1$, $B(1) = 2q+1$, and that for $n \geq 2$, $B(k)$ is always even.)

**Definition 10** *Let* $Y \subseteq N^k$, *and let* $y = < y_1, \ldots, y_{i-1}, y_i, y_{i+1}, \ldots, y_k > \in Y$.

- $y$ *is* safe *if* $(\exists i)[y_i \in S_q]$.
- $Y$ *is* safe *if* $(\forall y)[y \in Y \ \Rightarrow y$ is safe$]$.
- $Y_{\{y_i\}} \ =_{def} \ \{< z_1, \ldots, z_{i_1}, z_{i+1}, \ldots, z_k > \ | \ < z_1, \ldots, z_{i_1}, y_i, z_{i+1}, \ldots, z_k > \in Y \ \}$.

I.e., $Y$ is safe if all members of $Y$ have some element which has some component in $S_q$. For each individual coordinate point $y_i$, $Y_{\{y_i\}}$ may be viewed as the $k-1$ dimensional projection of $Y$ from $y_i$.

**Lemma 2** *For $k \geq 1$, if*

- $Y \subseteq N^k$
- $|Y| \geq \frac{B(k)}{2}$
- $(\forall y \in Y)(\forall 1 \leq i \leq k)[|Y_{\{y_i\}}| \ \leq \ B(k-1)]$

*then $Y$ is not safe.*

**Proof** The third hypothesis of the lemma quarantees that for each coordinate $i$ $(1 \leq i \leq k)$, there are at most $q * B(k-1)$ elements which can be safe *because* the $i^{th}$ coordinate projects into $S$. Therefore there are at most

$$\begin{aligned} k * q * B(k-1) &= k * q * (k-1)! * (2q+1)^{k-1} \\ &< k! * (\frac{2q+1}{2}) * (2q+1)^{k-1} \\ &= \frac{k! * (2q+1)^k}{2} \\ &= \frac{B(k)}{2} \end{aligned}$$

safe elements. Therefore any subset of $N^k$ which satisfies the third condition of the hypothesis and has at least $\frac{B(k)}{2}$ members must have at least one element which is not safe, i.e. which is also a member of $\overline{S}^k$. □

**Theorem 5 [OW-91]** *Suppose SAT $\leq_{k-tt}$ S with S sparse. Then P = NP.*

We will prove this theorem by first proving a result which gives a successful breadth-first tree pruning algorithm in the event that the truth-table is a *fixed* k-truth-table. This is adequate, because for a general $k$-truth-table reduction, on any input the algorithm calls at most one of $2^{2^k}$ fixed truth-tables, so if we label each node in the breadth-first search of the self-reduction tree by the truth-table called, we will only have a finite number of such labels, and if we can prune each of these $2^{2^k}$ independent sets of labeled nodes down to polynomial size, we will have a breadth-first search tree with only polynomial width at each level. Obviously, the running time will be "only" increased by at most the rather large *constant* factor, $2^{2^k}$.

Thus to prove the result, it suffices to prove the weaker result:

**Theorem 6** *Suppose SAT* $\leq_{k-tt}$ *S with S sparse, via a* fixed $k-truth-table$. *Then there is a tree pruning algorithm which uses L(SAT) to prune the self-reducibility tree for SAT and solves SAT in polynomial-time. Furthermore,*

- *The algorithm is independent of the particular k-truth-table which accomplishes the reduction of SAT to S.*
- *The tree pruning algorithm successfully prunes the self-reduction tree for SAT so that each level of the tree has at most B(k) nodes.*
- *The time required for the algorithm is polynomial in m, the size of input formulas to the algorithm.*

**Proof** We have already observed that $B(1) = 2q + 1$, and $B(0) = 1$, where $q$ is polynomial in $m$, so Lemma 1 gives us the basis for an inductive proof of the correctness of the theorem. So we assume now that we have proven the result for truth-tables of size $k$, and give a tree-pruning algorithm for truth-tables with $k + 1$ variables.

Our strategy is as follows. We will prune any given level of the tree from left-to-right, and then from right-to-left, much as we did for the case of 1-truth-tables. Suppose in this pruning we are at a node, $n$ and we calculate the value $T(n) = < y_1, y_2, \ldots, y_i, \ldots, y_{k+1} >$ . If this same $k + 1$-truth-table has occurred earlier, obviously, just as was the case for 1-truth-tables, we may eliminate both this node and all intervening nodes. But in addition, for any fixed $i$, if we hold $y_i$ fixed, while we do not know whether $y_i \in S$, we nevertheless know that if $n$ and $n'$ are two distinct nodes with $T(n) = < y_1, y_2, \ldots, y_i, \ldots, y_{k+1} >$ and $T(n') = < y'_1, y'_2, \ldots, y'_i, \ldots, y'_{k+1} >$ for which it happens that $y_i = y'_i$, then we can regard the reductions given by $T(n)$ and $T(n')$ as reductions given, not by a $k + 1$-truth-table, but as reductions given by the same (but unknown) $k$-truth-table, obtained by fixing the $i^{th}$ variable at the truth value given by the unknown answer to the question "$y_i \in S$?".

For each fixed position $i$, all of the nodes which agree when the $i^{th}$ variable is fixed can (by our induction hypothesis) be pruned so that at most $B(k)$ nodes remain. We do this, systematically moving left-to-right in a breadth-first search of a level of our self-reduction tree for $SAT$. Whenever we pick up *new* values $y_i$ located in positions $i$ where $y_i$ has not occured in position $i$ to the left, we use generic $k$-truth-table pruning *across the*

*entire level of the tree* for this new position and value. If we do this for *every new variable position and node* $n$ visited in the left-to-right search, then we will have guaranteed that in the *entire* set $Y$ of distinct node labels, $T(n) = < y_1, y_2, \ldots, y_i, \ldots, y_{k+1} >$, remaining after all of these prunings, we will have satisfied the condition

$$(\forall y \in Y)(\forall 1 \leq i \leq k+1)[|Y_{\{y_i\}}| \leq B(k)].^{17}$$

For each $y_i$ visited this will hold across the entire level of the tree, and therefore will certainly hold across all nodes *to the left* of our current position. But by our basic combinatorial lemma, this guarantees that if the set $Y$ of distinct labels remaining at, and to the left of, our current position ever gets at least $B(k+1)/2$ distinct members, then at least one of these labels to the left of our current position is not safe. I.e., *all* $k+1$ of its elements queried are really in $\overline{S}$. I.e., at some unknown place to the left of our current position in the left-to-right search of this level of the self-reduction tree for $SAT$ we must have been evaluating the truth-table $T(f, f, \ldots, f)$.

This situation is now exactly as for the proof in the case for a 1-truth-table: If the left-to-right pruning accumulates at least $B(k+1)/2$ distinct members, then we stop the left-to-right pruning, and instead begin a right-to-left pruning, both collapsing between nodes with identical truth-table labels and trying at all nodes all possible $k$-truth-table prunings, obtained by fixing any $i$ (one of the $k+1$ variable positions) and identifiying any two nodes, $n$ and $n'$ as using the same $k$-truth-table if $T(n) = < y_1, y_2, \ldots, y_i, \ldots, y_{k+1} >$ and $T(n') = < y'_1, y'_2, \ldots, y'_i, \ldots, y'_{k+1} >$ with $y_i = y'_i$.

Again, if the number of distinct nodes remaining during this right-to-left pruning ever reaches at least $B(k+1)/2$ distinct members, then we can be sure that we have encountered *some* node for which the truth-table reduces to $T(f, f, \ldots, f)$. In this case, just as in the case for 1-truth-tables, all nodes between those reached in the left-to-right search and those reached in the right-to-left search, can be eliminated, leaving us with just $B(k+1) - 1$ nodes.

Because $k$ is fixed and independent of the size of the input, the steps involved in going recursively from a $k+1$-truth-table to a polynomial num-

[17] In simplist terms, what this amounts to is taking the $k+1$-truth-table, and at each level of the tree, taking every instance of every coordinate of the truth-table which appears in the reduction, regarding those nodes which have this value at this coordinate as being labeled by the same unknown $k$-truth-table and inductively pruning these nodes using the generic $k$-truth-table pruning algorithm.

ber of $k$-truth-tables cannot turn the process, which is polynomial without the recursion, into a nonpolynomial process, (although it certainly can enormously blow up the polynomial). More specifically, if we let $W(k)$ denote the maximum time required to process any level of the tree while processing a $k$-truth-table, then we see that

$$W(k+1) \leq O(2B(k+1)P(m) + (\#\text{ of distinct } k-\text{truth}-\text{tables processed})\, W(k)),$$

where $2B(k+1)$ serves as a maximum bound on the width that any level of the tree ever reaches, and $P(m)$ bounds the difficulty of computing the various truth-table labels of size $k+1$. Since each node visited yields a maximum of $k+1$ new $k$-truth-tables requiring a recursion, this yields

$$\begin{aligned} W(k+1) &\leq 2B(k+1)P(m) + 2B(k+1)(k+1)W(k) \\ &\leq 2B(k+1)(k+1)[P(m) + W(k)] \\ &\leq 2B(k+1)(k+1)[P(m) + [2B(k+1)(k+1)[P(m) + W(k-1)]]] \\ &\leq \cdots \\ &\leq [2B(k+1)(k+1)]^{k+1}[P(m) + W(1)]. \end{aligned}$$

We saw in Proposition 1 that $W(1)$ has a polynomial bound, and of course $B(k+1)$ already contains a rather large polynomial, but since $k+1$ is fixed, the time required to process any level of the tree remains bounded by a (very large) polynomial. The proof is now completed by observing that there are only linearly many levels to the tree, so that the entire tree can still be processed in a polynomial number of steps. □

As mentioned earlier, the algorithms given in [OW-91] and in [HL-91], have much better polynomial bounds, with the better of the two in [HL-91]. As explained in [HL-91], the smaller bounds are better suited to obtain extensions of the Ogiwara-Watanabe results to "modestly unbounded" truth-tables. Furthermore, just as in [HL-91], the techniques used in our proofs can be applied not just at $NP$, but at any level of the polynomial-time hierarchy.

# 4 Conjunctive and Disjunctive Reductions

Closing the gap between what is known for polynomial-time Turing reductions to sparse sets (Section 2) and what is known for bounded truth-table reductions (Section 3), is an intriguing open problem. The results of this section are a small step in this direction.

Using simpler versions of the tree-pruning arguments of Section 3, we now prove extensions of the Ukkonen and Yap results stated in Theorem 4. Note that the statement $coSAT \leq^P_{conj}$-reducible to a sparse set, $S$, is equivalent to saying that $SAT \leq^P_{disj} \overline{S}$.

**Theorem 7** *If $S$ is sparse and either*

- $SAT \leq^P_{disj} \overline{S}$ (Ukkonen-Yap-83), *or*
- $SAT \leq^P_{conj} S$ [18]

*then $P = NP$.*

**Proof** Suppose first that $SAT \leq^P_{disj} \overline{S}$. This implies that $L(SAT) \leq^P_{disj} \overline{S}$. We give a simple pruning using a left to right traversal of the self-reduction tree for $SAT$.

Suppose that we are looking at a level of the tree in which an input generates a disjunctive truth-table of size $k$, (so $k$ is now polynomial in the size of the input).

Suppose that we are at some node $n$ of the self-reduction tree for $SAT$ while doing a left-to-right breadth-first pruning of the self-reduction tree for $SAT$. Let $y_1, y_2, \ldots y_k$ be the elements queried by the truth-table for $L(SAT)$ for this node, and assume for the moment that all of these elements have also been queried some place earlier in the left-to-right search (not necessarily all at the same node). Now if the truth-table evaluates to "true," then one of the $y_i$ must be in $\overline{S}$, and since this same $y_i$ occured at some earlier node, $n' \prec n$, node $n'$ must also have evaluated to "true." Thus there is a satisfying assignment someplace at, or to the left of node $n'$. In this case, node $n$ can be safely eliminated. But the other possibility is that node $n$ evaluates to "false." This means that there is no satisfying assignment either through node $n$ nor to the left of $n$, so again node $n$ can be safely eliminated.

[18] Added June, 1992: This extension of Yap's 83 result has been obtained independently in several very recent papers on reductions to sparse sets, ([AHHKLMOSST-92], [RR-92]).

In either case, if the the elements queried by the truth-table at node $n$ are a subset of the elements queried to the left of node $n$, then node $n$ can be safely eliminated.

Thus we may assume that as we do the left-to-right search, we accumulate at least one new query $y_i$ at each node that is retained. But then, once we have retained $k+1$ nodes, one of the nodes we have retained must have at least one element of $\overline{S}$ as part of its query, and so it evaluates to "true." Thus some satisfying assignment lies to the left, or at this node, so the formula is satisfiable.

So in any case, we may prune the tree to have at most a polynomial number of nodes, enabling us to decide $SAT$, either because we terminated knowing that a satisfying assignment exists, or by pruning the entire tree to polynomial size and then testing all of the leaves of the tree.

In the case that $SAT \leq^P_{conj} S$, we proceed similarly, but this time we do a right-to-left search.

Suppose that in this right-to-left search we are at a node $n$ for which the queries for $L(SAT)$ query only elements $y_1, y_2, \ldots y_k$ which have already been queried in our right-to-left search, and let $n'$ be the node remaining *immediately to the right* of node $n$.

If some $y_i$ is in $\overline{S}$ then node $n$ evaluates to "false." Since $y_i$ also occurs in some node to the right of $n$, this other node, and hence certainly also node $n'$, must also evaluate to "false." In this case, there is no satisfying assignment through node $n'$ nor to the left of $n'$, so node $n'$ may be safely eliminated. The other possibility is that for $L(SAT)$ the node $n$ evaluates to "true." But in this case, there is a satisfying assignment either through, or to the left of node $n$, so again node $n'$ may be safely eliminated.

In either case, if the elements queried by the truth-table at node $n$ are a subset of the elements queried to the right of node $n$, then node $n'$, the node immediately to the right of node $n$, can be safely eliminated.

Thus we may assume that as we do the right-to-left search, we accumulate at least one new query $y_i$ at each node that is retained. But then, if we retain $k+1$ nodes, one of the nodes we have retained must have at least one element in $\overline{S}$ as part of its query, and so it evaluates to "false." Thus no satisfying assignment lies at this node or to its left, so the right-to-left search may be safely terminated at this node.

So in any case, we can prune the tree to have at most a polynomial number of nodes, and this enables us to decide $SAT$ by testing all of its leaves. □

## 5 Acknowledgements

I would like to thank Deborah Joseph and Tim Long for teaching me much of what I know about reductions to sparse sets. Giovanni Faglia exhibited much patience in listening to false proofs of extensions of Theorem 3. Special thanks go to Harry Buhrman and Richard Beigel for pointing out the work in [KSW-87] and [BH-91] equating parity and log reductions and to Tim Long for especially useful comments on a near final draft of this paper.

## 6 References

[BBJSY-89] Bertoni, A., D. Bruschi, D. Joseph, M. Sitharam, and P. Young, "Generalized Boolean hierarchies and Boolean hierarchies over *RP,*" final version prior to publication available as Univ Wisc CS Tech Report; short abstract in *Proc* $\mathbf{7}^{th}$ *Symp Foundations Computing Theory*, (FCT-1989), *Springer-Verlag, LNCS* **380**, 35-46.

[BH-91] S. Buss and L. Hay, "On Truth-Table Reducibility to SAT," *Information and Computation* **91** (1991), 86-102.

[CGHHSWW-88] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung, "The Boolean hierarchy I: structural properties," *SIAM J Comput,* **6** (1988), 1232-1252.

[CGHHSWW-89] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung, "The Boolean hierarchy II: applications," *SIAM J Comput,* **7** (1989), 95-111.

[Fo-79] S. Fortune, "A note on sparse complete sets," *SIAM J Comput,*, (1979) 431-433.

[JY-90] D. Joseph and P. Young, "Self-reducibility: Effects of internal structure on computational complexity," in "Complexity Theory Retrospective," edited by A. Selman, Springer Verlag, 1990, 82-107.

[Ka-87] J. Kadin, "$P^{NP^{[logn]}}$ and sparse Turing complete sets for NP," *Proc Structure in Complexity Conference,* **2** (1987), 33-40.

[KL-80] R. Karp and R. Lipton, "Some connections between nonuniform and uniform complexity classes," *Proc of the* $\mathbf{12}^{th}$ *ACM STOC*, 1980, 302-309. (Also appears as "Turing machines that take advice," *L'Ensignement Mathématique* **82** (1982), 191-210.)

[KSW-77] J. Köbler, U. Schöning, and K. Wagner, "The difference and the truth-table hierarchies for NP," *R.A.I.R.O.* **21** (1987), 419-435.

[HL-91] S. Homer and L. Longpré, "On reductions of NP sets to sparse sets," *Structure in Complexity Conference,* **6** (1991), 79-88, *JCSS*, to appear.

[Lo-82] T. Long, "A note on sparse oracles for NP," *JCSS,* **24** (1982), 224-232.

[Ma-82] S. Mahaney, "Sparse complete sets for NP: solution of a conjecture of Berman and Hartmanis," *JCSS,* **25** (1982), 130-143.

[Ma-89] S. Mahaney, "The isomorphism conjecture and sparse sets," in "Computational Complexity Theory," edited by J. Hartmanis in *AMS Proc Symp Applied Math Series,* (1989).

[MY-78] M. Machtey and P. Young, "An introduction to the general theory of algorithms," Elsevier-North Holland, (1978).

[OW-91] M. Ogiwara and O. Watanabe. "On polynomial-time bounded truth-table reducibility of NP sets to sparse sets," *SIAM Journal on Computing,* **20(3)** (1991), 471-483.

[Uk-83] E. Ukkonen, "Two results on polynomial time truth-table reductions to sparse sets," *SIAM J Comput,* **12** (1983), 580-587.

[Wa-87] O. Watanabe, "On intractability of the class up," *Mathematical Systems Theory,* **24** (1991), 1-10.

[Wa-88] O. Watanabe, "On polynomial-time one-truth-table reducibility to sparse sets," *JCSS,* **44** (1992), 500-516.

[Ya-83] C. Yap, "Some consequences of non-uniform conditions on uniform classes," *Theor Comput Sci* **26** (1983), 287-300.

[Ye-83] Y. Yesha, "On certain polynomial time truth-table reductions to sparse sets," *SIAM J Comput,* **12** (1983), 411-425.

[Yo-90] P. Young, "Juris Hartmanis: Fundamental contributions to isomorphism problems," in "Complexity Theory Retrospective," edited by A. Selman, Springer Verlag, (1990), 28-58.

A Bibliography of Recent Papers, mostly added Summer 1992:

[AHHKLMOSST-92] V. Arvind, et al., "Reductions to sets of low information content," in "Complexity Theory," edited by K. Ambos-Spies, S. Homer, U. Schöning, Cambridge University Press (1993), to appear.

[AKM-92] V. Arvind, J. Köbler, M. Mundhenk, "Bounded truth-table and conjunctive reductions to sparse and tally sets," *Technical Report, Universität Ulm Fakultät für Informatik,* **92-01** (April, 1992), 1-22.

[Bi-92] Bin Fu, "With quasi-linear queries, EXP is not polynomial time Turing reducible to sparse sets," *Proc Structure in Complexity Theory Conference* **8**, (1993), to appear.

[BLS-93] H. Buhrman, Luc Longpre, Edith Spaan, "Sparse reduces conjunctively to tally," *Proc Structure in Complexity Theory Conference* **8** (1993), to appear.

[RR-92] D. Ranjan and P. Rohatgi, "Randomized reductions to sparse sets," *Proc Structure in Complexity Theory Conference* **7** (1992), 239-42.

[Yo-92a] P. Young, "How Reductions to Sparse Sets Collapse the Polynomial Time Hierarchy: A Primer. Part I: Polynomial-time Turing Reductions." *SIGACT News,* **84** (1992), 107-117.

[Yo-92b] "How Reductions to Sparse Sets Collapse the Polynomial Time Hierarchy: A Primer. Part II: Restricted Polynomial-time Reductions." *SIGACT News,* **85** (1992), 83-94.