USING SELF-REDUCIBILITIES TO CHARACTERIZE POLYNOMIAL TIME

by

Judy Goldsmith

Deborah Joseph

Paul Young

Computer Sciences Technical Report #749

February 1988

USING SELF-REDUCIBILITIES TO CHARACTERIZE POLYNOMIAL TIME*

JUDY GOLDSMITH
University of Wisconsin - Madison

DEBORAH JOSEPH
University of Wisconsin - Madison

PAUL YOUNG
University of Washington

In this paper we study the effect that the *self-reducibility* properties of a set have on its time complexity. In particular, we show that the extent to which a set is self-reducible can determine whether or not the set is polynomially decidable. Our results concern three variations on the notion of Turing self-reducibility: *near-testability, word decreasing query self-reducibility,* and *p-cheatability.* Our first pair of results provide characterizations of polynomial time by coupling the notion, first of Turing self-reducibility, then of near-testability, with the notion of p-cheatability. In contrast, our final theorems show that if specific nondeterministic and deterministic time classes do not collapse, then 1) we cannot similarly characterize polynomial time by coupling word decreasing query reducibility with a variant of p-cheatability, and 2) we cannot characterize polynomial time by coupling *parity-P* and p-cheatability.

Contents

Categories and Subject Descriptors: F.1.3 [Theory of Computation]: complexity classes
General Terms: Structural complexity theory
Additional Key Words and Phrases: self-reducible, wdq self-reducible, near-testable, p-cheatable, polynomially decidable

Date: November 30, 1987

---

* This work was supported in part by the National Science Foundation under grant DCR-8402375, as well as by a grant from AT&T Bell Laboratories. This report has also appeared as University of Washington Technical Report 87-11-11 under the same title.

Authors' addresses:

J. Goldsmith and D. Joseph: Computer Sciences Department, University of Wisconsin, 1210 West Dayton St., Madison, WI 53706.

P. Young: Computer Science Department FR-35, University of Washington, Seattle, WA 98195.

1. INTRODUCTION.

In this paper we study the effect that various *self-reducibility* properties have on the complexity of a set. In particular, we show that the extent to which a set exhibits certain combinations of these properties can determine whether the set is polynomially decidable.

A set A is said to be *Turing self-reducible* if there is a polynomial time oracle machine SR^A that recognizes A, and on inputs of length n all of SR^A 's queries have *length* less than n. Obviously, all *polynomially* decidable sets are Turing self-reducible, and all sets that are Turing self-reducible are decidable in polynomial space. In this paper we consider three variations on the notion of Turing self-reducibility: *near-testability, word decreasing query self-reducibility,* and *p-cheatability.* We investigate when these notions can be combined to give an exact characterization of P, the *polynomial time decidable* sets. We prove four results.

A set A is called *p-cheatable* if there is a constant $k \geq 1$ such that, for any 2^k elements, the question of membership in A for all 2^k elements can be reduced in polynomial time to only k questions about membership in A.

Result 1: $A \in P$ if and only if A is both *p-cheatable* and *Turing self-reducible.*

After proving Result 1, we investigate whether Result 1 can be extended to other classes of self-reducible sets.

Balcázar introduced a class of sets called *word decreasing query (wdq) self-reducible* sets. This class is defined by weakening the definition of Turing self-reducible to allow that on input w, SR^A may query A about any word that is *less than* w in the lexicographical ordering. In [GJY-87a] we studied a special subclass of the wdq self-reducible sets in which the only query that SR^A needs to ask is membership of $w-1$ (the lexicographic predecessor of w). These sets are called *near-testable.* Our second result extends the scope of Result 1.

Result 2: $A \in P$ if and only if A is both *p-cheatable* and *near-testable.*

In [GHJY-87], Goldsmith, Hemachandra, Joseph and Young show that the class of near-testable sets is polynomially many-one equivalent to the class *parity-P*, ($\oplus P$), defined by Papadimitriou and Zachos, ([PZ-82]). Interestingly, in spite of the many-one equivalence of NT and *parity-P*, Result 2 cannot be extended to *parity-P* unless certain large nondeterministic time classes collapse. Specifically, if we define $f(0) = 2$ and $f(n+1) = 2^{f(n)}$, then we obtain the following.

Result 3: If $\bigcup_{c>0} ParityTIME[(2^{f(n)})^c] \neq \bigcup_{c>0} TIME[(2^{f(n)})^c]$, then there are sets in *parity-P* that are *p-cheatable* but are not in P.

As our final result, we show that, just as Result 2 is unlikely to extend to *parity-P*, Results 1 and 2 are unlikely to extend to their common generalization, the class of wdq self-reducible sets.

Result 4: If $\bigcup_{c>0} NTIME[(2^{f(n)})^c] \neq \bigcup_{c>0} TIME[(2^{f(n)})^c]$, then there are sets that are word decreasing query self-reducible and are ($2^k - 1$ for k) p-cheatable but are not in P.

In summary, the results in this paper give fairly tight bounds on the strength of self-reducibility needed to guarantee that a set is polynomially decidable. In addition, two of our results provide new characterizations of the class of polynomially decidable sets.

In Section 2 we present basic definitions for these classes of self-reducible sets. Because many readers may be unfamiliar with some of these classes, we have enumerated some recent results about these sets in an Appendix.

2. BASIC DEFINITIONS.

Our starting point in this study of *self-reducible sets* and polynomial time behavior will be the class of *Turing self-reducible sets.* As mentioned above, a set A is *Turing self-reducible* if there is a polynomial time oracle machine SR^A that recognizes A, and on inputs of length n all of SR^A 's queries have length less than n. These sets have been extensively studied by Balcázar, Berman, Book, Fortune, Ko, Mahaney, Meyer, Paterson, Schöning, Trakhtenbrot and others. Each of the other classes that we will consider can be obtained by relaxing this definition of Turing self-reducible. The classes of *word decreasing query self-reducible sets* and *near-testable sets* are obtained by relaxing the condition that all queries have length less than the length of the input. The class of *p-cheatable sets* is obtained by further relaxing the condition on the size of the queries but compensating by requiring that only a few queries are needed to answer membership questions for many elements.

The class of *word decreasing query (wdq) self-reducible sets* was introduced by Balcázar to unify results relating hierarchy collapses to the existence of sets that are reducible to sparse sets. (See the Appendix for a summary of these results.)

Definition [Ba-87]. *A set A is wdq self-reducible if there is a polynomial time oracle Turing machine SR^A that decides membership in A, and for each input w, all of SR^A 's queries lexicographically precede w.*

Obviously, any set that is wdq self-reducible is decidable in *exponential time.* Independently of Balcázar, in [GJY-87a] we studied a particularly simple subclass of the wdq self-reducible sets called the *near-testable sets.*

Definition [GJY-87a]. *A set A is near-testable ($A \in NT$) if there is a polynomially computable function which, given w, decides whether exactly one of w and $w+1$ is in A, where $w+1$ is the lexicographic successor of w. That is, the function*

$$f(w) = \chi_A(w) + \chi_A(w+1) \ (mod\ 2)$$

is computable in polynomial time.

Thus if a set is near-testable, then it is wdq self-reducible with $w-1$ being the only query necessary to answer membership for input w. Obviously, all wdq self-reducible sets that require only one query to the set, including all near-testable sets, are decidable in polynomial space.

The *p-cheatable* sets were first studied by Amir, Beigel, Gasarch, Gill, Hay and Owings, ([AG-87], [Be-87a], [Be-87b], [BGGO-86], [BGH-87], [BGO-87]). The definition of the class is based on the following scenario: one is given a large collection of inputs, $w_1, w_2, ..., w_n$, and one would like to determine whether or not each input is in a known set A. It is assumed that the membership problem for A is difficult. Therefore, if knowing whether some w_i is in A helps in determining whether another w_j is in A, then one would like to use this information. These authors use this idea to motivate the following machine model. To determine membership in A, one designs an oracle machine CH that, when given A as oracle will determine membership in A for n different inputs by asking the oracle as few questions and doing as little computation as possible. For example, one might hope to design machines that run in polynomial time and for a fixed k decide membership for 2^k inputs by asking only k questions. The definition can be formalized as follows.

Definition [AG-87], [Be-87a]. *A set A is (n for k) p-cheatable if there exists a polynomial time oracle machine CH using A as oracle such that, if CH^A is given inputs $(w_1, ..., w_n)$, then, with k or fewer queries to the oracle, CH^A determines membership in A for each of $w_1, ..., w_n$. We will be primarily interested in the case where $n = 2^k$ and in this paper we call (2^k for k) p-cheatable sets simply p-cheatable.*

Note that the queries that CH asks in the course of deciding membership for $w_1, ..., w_{2^k}$ are not restricted to come from the set $w_1, ..., w_{2^k}$. In fact, for the purpose of our results the exact questions asked and the oracle to which they are addressed turn out not to be relevant.[1] It will turn out that the only thing that matters is the *number* of questions that are asked. Also, notice that k is a constant that will depend on the set A.

1. Beigel's definition of p-cheatable sets (what we call (2^k for k) p-cheatable) allows CH access to *any* fixed oracle. However, the definitions given by Amir and Gasarch for ($2^k - 1$ for k) p-cheatable sets require that CH use A as the oracle. (These sets are called *verbose* by Amir and Gasarch.)

To put the results that follow in context it is useful to observe the following.

- SAT is *2-disjunctive truth-table self-reducible* and hence Turing self-reducible. Thus, if $P \neq NP$, then not all Turing self-reducible sets are in P.
- Balcázar has observed that it is not hard to code the successive instantaneous descriptions of any polynomial space computation into a wdq self-reducible set. Thus, there are wdq self-reducible sets that are complete for $Pspace$.
- Amir and Gasarch ([AG-87]) have shown that there are p-cheatable sets of arbitrarily high time complexity. Thus, there are p-cheatable sets that are not wdq self-reducible.
- Finally, in [GJY-87a] we have shown that if one-way functions exist, then $NT \neq P$. What's more, a variant of SAT is polynomial many-one complete for NT, ([GHJY-87]). (See Appendix.)

3. RESULTS.

Any set that is polynomially decidable is self-reducible with respect to each of the definitions that we have given, although, as we have just seen, the converse is almost surely false. Our purpose here is to investigate when various *combinations* of self-reducibility *do* imply the converse. Our first result shows that if a set is both Turing self-reducible and p-cheatable, then it is polynomially decidable.

Before proving this theorem, we want to make two comments about this theorem and its proof.

First, this result is a strengthening of a result of Beigel ([Be-87b]) which shows that, if a set is self-truth-table-reducible (a variant of truth-table self-reducible) and p-cheatable, then it is polynomially decidable.[2] Our extension is important because it shows that it is *not* the combination of a *nonadaptive* reducibility (truth-table reducibility) with an adaptive reducibility (p-cheatability) that causes the set to be polynomially decidable. The polynomial decision algorithm instead results from the fact that Turing self-reducibility is *length decreasing*, and hence is polynomially well-founded. It is this well-foundedness that is needed to force the p-cheatable set to be polynomially decidable.

Second, the requirement that the set be (2^k *for* k) p-cheatable is perhaps a little misleading. What is really needed is a special form of polynomial cheatability which says that there is some fixed integer n and a polynomial time algorithm that, given any sequence $a_1, a_2, ..., a_n$, can find some initial subsequence

2. Since SAT is self-reducible via a particularly simple truth-table reduction, our first result can also be consider an extension of work by Amir and Gasarch, who, using a result of Krentel, have observed that no set that is complete for NP under many-one polynomial time reductions is p-cheatable, unless $P = NP$. (Krentel proved that if $P \neq NP$, then there are functions computable in polynomial time using $k+1$ queries to SAT that can not be computed in polynomial time using only k queries to SAT, ([Kr-86]). Since any set polynomially many-one equivalent to a p-cheatable set is easily seen to be p-cheatable, it follows that no set that is many-one complete for NP can be even ($k+1$ for k) p-cheatable, unless $P = NP$.)

$a_1, ..., a_j$ such that, if answers to the membership questions $a_1 \in A?, ..., a_j \in A?$, are given, the answer to $a_{j+1} \in A?$ can be derived.

We begin our proof by showing that if a set is (2^k *for* k) p-cheatable, then this condition is satisfied. In the proof, we will use a combinatorial lemma to help establish this condition. The condition in turn is used to prune the self-reducibility tree (a depth-first search tree) for A that results from the Turing self-reduction on A.

Theorem 1. *If A is both Turing self-reducible and p-cheatable, then A is in P.*

Proof. Suppose that A is p-cheatable and Turing self-reducible. Let CH^A be an oracle Turing machine which witnesses that A is p-cheatable and let SR^A be an oracle Turing machine which witnesses that A is Turing self-reducible. We will describe a polynomial time algorithm for A based on CH^A and SR^A.

An important idea in this proof, and in the proof of Theorem 2, is to *simulate* CH^A with a polynomial time machine that does not actually ask any questions of an oracle. We will call this machine $\widehat{CH}$ and begin by describing its behavior.

Suppose that CH^A on input $(a_1, a_2, ..., a_{2^k})$ correctly decides membership in A for each of the a_i's and does so by asking at most k questions of the oracle. The machine $\widehat{CH}$ on the same input simulates CH^A, and when CH^A makes a query to the oracle, $\widehat{CH}$ simulates both possible answers. Since CH^A asks at most k questions, $\widehat{CH}$ produces (at most) 2^k possible Boolean valued output vectors, $\overline{v_i}$, as CH^A 's possible answers to membership of $a_1 \in A?$, $a_2 \in A?$, ..., $a_{2^k} \in A?$[3] Since k is a fixed constant, $\widehat{CH}$ runs in time polynomial in $|(a_1, a_2, ..., a_{2^k})|$; if we assume that $a_i < a_j$ for all $i < j$, then $\widehat{CH}$ runs in time polynomial in $|a_{2^k}|$.

Thus the questions $a_1 \in A?$, $a_2 \in A?$, ..., $a_{2^k} \in A?$ form a vector of Boolean variables, which by an abuse of notation we can abbreviate to $a_1, ..., a_{2^k}$, while the simulation of all possible answers to k queries forms at most 2^k sequences, $\overline{v_i}$, of possible Boolean output values for these questions.

We next develop some notation and prove an easy lemma which enable us to conclude that for any such sequence, in time polynomial in $|a_{2^k}|$, we can always find a $j < 2^k$ such that answers to the j questions $a_1 \in A?$, ..., $a_j \in A?$ will always allow us to infer the answer to $a_{j+1} \in A?$

3. While it is true that CH^A is not really guaranteed to ask only k questions or run in polynomial time if it is not given correct answers to membership about A, if a sequence of simulated queries to A either generates too many queries or fails to run within the required polynomial bound, the resulting output vector can safely be ignored since *a prior* the output vector is not generated by correct information.

Notations. Let $a_1, ..., a_n$ be n boolean variables. Let $\overline{v_i}$, $(1 \leq i \leq m)$ be m fixed truth assignments to $a_1, ..., a_n$. (So $\overline{v_i} = v_{i,1}, ..., v_{i,n}$, is a sequence of n boolean values.) For $1 \leq j \leq n$ we say that $\overline{v_i} \equiv_j \overline{v_{i'}}$ if $\overline{v_i}$ and $\overline{v_{i'}}$ agree on their first j values; i.e., if for all t with $1 \leq t \leq j$, $v_{i,t} = v_{i',t}$. For $0 \leq j < n$, we say that $a_1, ..., a_j$ imply a_{j+1}, and we write $(a_1, ..., a_j) \rightarrow (a_{j+1})$, if for all i, i', $\overline{v_i} \equiv_j \overline{v_{i'}}$ implies that $v_{i,j+1} = v_{i',j+1}$. Clearly, if $(a_1, ..., a_j) \rightarrow (a_{j+1})$ then, no matter what truth values are assigned to $a_1, ..., a_j$ from any of the permissible initial assignments in $\overline{v_1}, ..., \overline{v_m}$, we can predict the truth assignment to a_{j+1} from the truth assignments to $a_1, ..., a_j$. Finally, for each j, $(1 \leq j \leq n)$, we let $\#_j$ denote the *index* of the the equivalence relation $\equiv_j$; i.e., $\#_j$ is the number of *distinct* initial Boolean sequences $v_{i,1}, ..., v_{i,j}$ from among all $\overline{v_i}$, $(1 \leq i \leq m)$. Clearly for all j, $\#_j \leq \#_{j+1}$ and $\#_j \leq m$.

Lemma 1. *Let $a_1, ..., a_n$ be n boolean variables. Let $\overline{v_i}, (1 \leq i \leq m)$ be m fixed truth assignments to $a_1, ..., a_n$. Then*
a) $\#_j = \#_{j+1}$ implies that $(a_1, ..., a_j) \rightarrow (a_{j+1})$.
b) If $m \leq n$, then there exists $j < m$ such that $(a_1, ..., a_j) \rightarrow (a_{j+1})$.

Proof. a) Follows directly from the observation that the equivalence relation $\equiv_{j+1}$ is always a refinement of the equivalence relation $\equiv_j$. Thus, if the indices of the equivalence relations are the same, then it must be true that for all i, i', $\overline{v_i} \equiv_j \overline{v_{i'}}$ implies that $v_{i,j+1} = v_{i',j+1}$.

b) Note that if $\#_1 = 1$, then all values $v_{i,1}$ are the same, so trivially $(\ ) \rightarrow (a_1)$. Thus if (b) fails, $\#_1 = 2$. But by (a), if (b) fails, then $\#_{j+1} > \#_j$, so $\#_j > j$ for all j. But this implies $\#_n > n$, which is impossible since for all j, $\#_j$ cannot exceed the number of distinct Boolean vectors $\overline{v_i}$, which is m. ∎

The next thing that is important for the proof of Theorem 1 is to *simulate* SR^A; again without actually querying an oracle. The result of this simulation is a *self-reduction tree.* The *self-reduction tree* for a is a *depth-first* search tree defined as follows: a is the root of the tree, and each child of a is a string queried by $SR^A(a)$, (in the order queried, from left to right). Each element of the tree other than a is a query made in the self-reduction of its parent node. The leaves are strings of length one. $SR^A(a)$ queries at most $p(n)$ strings, where $n = |a|$. The self-reductions of each of these strings query at most $p(n-1)$ strings, and so forth. Therefore, the self-reduction tree for a has at most $p(n)$ elements at depth one, $p(n)p(n-1)$ elements at depth two, and so forth, for a total of at most $\sum_{i=0}^{n-2} \prod_{j=0}^{i} p(n-j)$ elements. Although this is exponential in n, it can be used to give a decision procedure since the self-reduction tree systematically reduces the membership question for a to questions about the leaves of the tree, all of which are among the finite set of elements of length one. We will use the simulation of CH^A to prune this tree, so that, to decide whether a is in A, we will only need to discover membership information for polynomially many elements in the tree.

Our pruning of the self-reduction tree will remove most of the right subtrees. In particular, on *any* branch of the tree from any leaf a_1 to the root a, if there are more than 2^k nodes on the branch, then for *any* subsequence of (not necessarily contiguous) nodes $a_{i_1}, ..., a_{i_{2^k}}$ from the leaf to the root, we can find some $j < 2^k$ such that $(a_{i_1}, ..., a_{i_j}) \rightarrow (a_{i_{j+1}})$. But this means that as we traverse back up this branch of the self-reduction tree, once the memberships of $a_{i_1}, ..., a_{i_j}$ are all known we can infer the membership of $a_{i_{j+1}}$ *without* traversing down any subtree of $a_{i_{j+1}}$ that lies strictly to the right of the branch from a_1 to a. Thus, in *any* collection of at least 2^k nodes on a single branch of the tree, all of the right subtrees of at least one node can be pruned away. It follows that on any branch of the tree, all but at most $2^k - 1$ right subtrees can be pruned away. By careful programming this pruned tree can be generated in time polynomial in the number of its nodes.

Having seen that the self-reduction tree, as pruned by the machine $\widehat{CH}$, always has at most $2^k - 1$ nodes that branch to the right as we traverse the tree from any leaf to the root, we now analyze the size of such trees. For simplicity, we consider only trees, as above, that have nodes with size at most equal to their maximum distance to a leaf.

Claim. Any tree of this type with a constant bound on the number of right-branching nodes along any path, and with a polynomial bound both on the depth of the tree and on the number of children of each node, has total size that is polynomial in the size of the root.

We call a tree with constant bound c on the number of right-branching nodes a *c-tree*. Let $S(n, c)$ be the maximum number of elements in such a c-tree of depth n with polynomial bound $p(n)$ on the number of children a node of length n may have. We will make the worst-case assumption that the branching nodes are as close to the root as possible, i.e. that there is no non-branching node on a path *between* two branching nodes. (Suppose there were: we could draw a c-tree with more nodes by adding a subtree as a *left* child of the non-branching node. This would not affect its original children, but would increase the size of the tree.) We assume without loss of generality that p is non-decreasing, and for all n, $p(n) \geq 2$. Our goal is to prove that $S(n, c) \leq [n * p(n)]^{c+2}$.

If a is the root of the tree, consider a's children. The rightmost child of a lies along a path that has no right-branching nodes. Therefore, it is the root of a c-tree of height $n - 1$. The other $p(n) - 1$ children lie below a right-branching node, (namely, the root), so they are each the root of a $(c - 1)$-tree of height $n - 1$. In other words, we have the recurrence relation

$$S(n, c) \;=\; S(n-1, c) \;+\; (p(n)-1) * S(n-1, c-1) \;+\; 1.$$

To establish the base case for an induction, we note that any tree of depth 1 has at most two nodes on any path, so $S(1, c) = p(1) + 1$. A straightforward induction on n using the preceding recurrence relation then establishes that $S(n, c) \leq [n * p(n)]^{c+2}$.

Therefore, in fewer than $[n * p(n)]^{c+2}$ steps, we can determine a sufficient number of elements in the self-reduction tree for input a to determine whether $a \in A$. Thus $A \in P$, as claimed. ∎

Notice that in the proof of Theorem 1, we have used the fact that SR^A 's queries are length-decreasing to bound the height of the self-reduction tree. This is the only reason that we need this restriction. As we discuss in the Appendix, Meyer, Paterson, and Ko have generalized the notion of Turing self-reducibility to allow not just size reducing queries to A, but polynomially bounded queries along arbitrary polynomially bounded well-founded partial orderings. It is clear that our proof of Theorem 1 works equally well for this more general notion of Turing self-reducible.

As an obvious corollary to Theorem 1 we have the following.

Corollary. *A is polynomially decidable if and only if A is both Turing self-reducible and p-cheatable.*

A natural question to ask at this point is to what extent can we relax the notion of self-reducibility and still obtain a result like Theorem 1. For instance, if we consider sets that are wdq self-reducible and p-cheatable, are they still polynomially decidable? In an attempt to answer this question we will next consider sets that are near-testable. (Recall that the near-testable sets form a very simple subclass of the wdq self-reducible sets.) In this case the answer to our question is "yes;" all sets that are p-cheatable and near-testable are polynomially decidable.

Theorem 2. *If A is both near-testable and p-cheatable, then $A \in P$.*

Proof. In the proof of the previous result the polynomial time decision procedure for A used the self-reduction procedure SR^A to partially build a self-reduction tree, and then used the simulation $\widehat{CH}$ to prune that tree. In that case we could think of the self-reduction algorithm as *driving* the algorithm and the p-cheatability algorithm as a subprocess to keep the time bound down. In the proof of Theorem 2 the simulation $\widehat{CH}$ will *drive* the algorithm and we will use the near-testability algorithm to *finish-off* the process.

The key idea is much as before; the simulation $\widehat{CH}$ can be used to give a partial decision procedure for A. For any sequence $a_1, ..., a_{2^k}$, by analyzing the output of $\widehat{CH}$, we can always obtain a relationship of the form $(a_1, a_2, ..., a_j) \rightarrow (a_{j+1})$.

In this construction, we use $\widehat{CH}$ to prune subintervals out of the interval $[0, a]$, continuing until we are left with only intervals of length one. On input a, the idea is to split the interval $[0, a]$ into 2^k equal length subintervals: $[0, a_1]$, $(a_1, a_2]$, ..., $(a_{2^k-1}, a_{2^k} = a]$. We will try to *reduce* the problem of deciding membership for a to the problem of deciding membership for $a_1, ..., a_{2^k-1}$. For example, suppose we run the simulation $\widehat{CH}$ on input $(a_1, a_2, ..., a_{2^k})$ and analyze the output. *If* we can deduce that $(a_1, ..., a_{2^k-1}) \rightarrow (a_{2^k})$, then to decide whether $a \in A$ we simply need to know which of $a_1, ..., a_{2^k-1}$ are in A. Thus, as long as we remember the relationship $(a_1, ..., a_{2^k-1}) \rightarrow (a_{2^k})$, we can ignore all elements in the interval $(a_{2^k-1}, a_{2^k}]$. Since the a_i's are evenly spaced this allows us to omit from further consideration $\frac{a}{2^k}$ elements. If this process could be repeated, removing $\frac{1}{2^k}$ of the remaining elements at

each step, then we would have a polynomial time algorithm. In fact, although the process is not nearly so uniform, we can give a polynomial time algorithm using this general idea.

Before formally describing the algorithm we need one more piece of terminology. At each step of the algorithm we will try to eliminate an interval of elements. Each of these intervals will be associated with one of the elements that we use as input to $\widehat{CH}$, (the a_i's in the above example). In particular, at any step in the algorithm the terminology a_i's *interval* will refer to those elements between a_{i-1} (or 0 in the case of a_1) and a_i that have *not yet* been eliminated. We now describe the full algorithm.

The algorithm for deciding a $\in$ A.

Step 0: Let $a_{1,1} = \lfloor \frac{a}{2^k} \rfloor$, $a_{1,2} = \lfloor \frac{2a}{2^k} \rfloor$, ..., $a_{1,2^k-1} = \lfloor \frac{2^{k-1}a}{2^k} \rfloor$, and $a_{1,2^k} = a$.

Initially, the intervals associated with each point are, $a_{1,1} : [0, a_{1,1}]$, ..., $a_{1,2^k} : (a_{1,2^k-1}, a_{1,2^k}]$.

Step i: **If** not all of the intervals have length 1,

then run $\widehat{CH}$ on $(a_{i,1}, a_{i,2}, ..., a_{i,2^k})$ and deduce a membership relation of the form

$$(*) \qquad (a_{i,1}, ..., a_{i,j}) \rightarrow (a_{i,j+1}), \quad 0 \leq j < 2^k;$$

a) *eliminate* $a_{i,j+1}$*'s interval* from further consideration and
store the relation $(*)$ for determining $a_{i,j+1}$'s membership question;
b) *pick a new endpoint* to use in the next step by
picking the midpoint of the longest interval currently associated with any a_i; and
c) *renumber the* a_i*'s* so that, $a_{i+1,1} < a_{i+1,2} < ... < a_{i+1,2^k}$;

else *(finishing up)* When all of the intervals have length 1, it is time to *put the pieces back together* and discover whether $a \in A$. It is easy to see that the single element intervals that remain, together with the eliminated intervals, exactly cover the entire interval $[0, a]$ in a disjoint fashion. Also every $a_{i,j}$ that was used in the above process must appear as a right endpoint of one of these intervals. For simplicity let us number the remaining intervals of length one together with the eliminated intervals, $I_1, I_2, ..., I_n$, from left to right. We want to determine membership for the right endpoint of each interval, since the right endpoint of I_n is a. If I_1 has unit length then its right endpoint is 1, and we know whether 1 is in A. If I_1 does not have unit length, then it must have been eliminated at some step of the algorithm, and the only way to have eliminated an interval of the form $[0, a_{i,1}]$ was by having $\widehat{CH}$ directly determine the membership of $a_{i,1}$ with a relation $(\ ) \rightarrow (a_{i,1})$. Thus, whatever form I_1 has, the algorithm directly tells us whether or not its right endpoint is in A.

Suppose now that we have determined membership for the right endpoints of intervals $I_1, ..., I_j$ and we want to determine membership for I_{j+1}'s endpoint. If the length of I_{j+1} is 1, then we simply use the near-testability algorithm to figure out the answer to the membership question using the right endpoint of I_j. Otherwise, there must have been some *Step i* at which I_{j+1} was eliminated using a relation $(a_{i,1}, ..., a_{i,j}) \rightarrow (a_{i,j+1})$. In this case the answer to $a_{i,j+1}$'s membership question

can be deduced by knowing the answers to the membership questions for smaller elements. Since each of these elements is the endpoint of some interval and we now know the answers for all smaller endpoints, we can now deduce the answer for I_{j+1}'s endpoint.

Complexity analysis. Notice that, except in the case that an interval has length 1, the *putting the pieces together* step at the end of the algorithm, simply involves *looking-up* information that was generated in the earlier steps of the algorithm. If an interval has length 1, then we must use the near-testability algorithm for A. This algorithm runs in polynomial time. Therefore, if the number of intervals generated is polynomial in the length of a, then the algorithm will be polynomial in $|a|$. Showing that the number of intervals is polynomial amounts to showing that the number of *Steps* in the algorithm is polynomial. To show that this is the case we need to consider the number of elements in the intervals that are eliminated at each step.

In *Step 1* we are guaranteed to eliminate an interval containing $1/2^k$ of the points less than a. If at each step we eliminated $1/2^k$ of the remaining elements, then after the i^{th} step we would have $a \cdot (2^k - 1)^i/(2^k)^i$ elements remaining. Thus in $log_{(2^k-1)/2^k} a$ steps we would have reduced all of the intervals to size one, and would have a polynomial time algorithm. Unfortunately, at each step we do not remove $1/2^k$ of the remaining elements. However, at any step there are 2^k intervals, and at each step we remove one interval and cut the remaining longest interval in half. Thus in going from *Step* $(2^k - 1) \cdot (i)$ to *Step* $(2^k - 1) \cdot (i + 1)$ we eliminate one half of the elements active at *Step* $(2^k - 1) \cdot (i)$. Since k is a constant, this is sufficient to guarantee that the algorithm runs in polynomial time. ∎

As an obvious corollary to Theorem 2 we have the following.

Corollary. *A is polynomially decidable if and only if A is both near-testable and p-cheatable.*

Before returning to the problem of extending Theorems 1 and 2 to the entire class of wdq self-reducible sets, we will look briefly at a class very closely related to the near-testable sets. In [GHJY-87] it is shown that the class of near-testable sets is polynomially many-one equivalent to the class *parity-P*.[4,5] It is thus natural to ask whether Theorem 2 holds for this class. Interestingly, in spite of the polynomial many-one equivalence of NT and *parity-P*, the theorem probably does not hold for *parity-P*.

4. A language, L, is in *parity-P* if there is a nondeterministic Turing machine, M, such that $w \in L$ if and only if $M(w)$ has an *odd* number of accepting paths; *parity-P* is sometimes denoted by $\oplus P$. If C is any deterministic time class, we will use the notation *parity-C* to stand for the obvious parity class defined with respect to the time bounds for the corresponding nondeterministic machines.

5. A thorough discussion of the relation between near-testability, *parity-P*, and p-cheatability, including the construction of sets complete for *parity-P* and for NT may be found in [GHJY-87]. Much of the work in [GHJY-87] is based on work in [GJY-87a] and in [He-87], and a discussion of some of this work may be found in the Appendix to this paper.

To see one reason why this is so, we consider log*-sparse sets. While standard definitions of sparse sets require that, for any value of n, the set have relatively few elements of size less than or equal to n, we require not only this, but that individual elements of the set are very widely separated.

Definition. *We say that a set S is log*-sparse if for x and y in S, $x < y$ implies $2^{|x|} < |y|$.*

Sparse sets have played an important role in complexity theory. Any polynomially sparse set that is Turing self-reducible or near-testable is in P. Similarly, any set that is word decreasing query self-reducible and is contained in a polynomially decidable, polynomially sparse, set (such as a tally set) is itself polynomially decidable. Although these sets are in P and it is known that no sparse sets can be complete for NP or for $coNP$ unless $P = NP$, it is nevertheless believed that that there exist very sparse sets in $NP - P$. In explaining why this should be so, Hartmanis, Immerman, and Sewelson ([HIS-85]) point out that the analysis of nondeterministic Turing machines is difficult precisely because their behavior often seems to be very isolated. In general, the behavior of a nondeterministic machine on any given input is not likely to tell much about its behavior on other elements of the set. They use this intuition to motivate their results showing that separation of higher nondeterministic and deterministic time classes implies that there are very sparse sets in $NP - P$. Specifically, they prove that for any sufficiently rapidly growing, well-behaved function T,

$$\bigcup_{c>0} NTIME[T(n)^c] \neq \bigcup_{c>0} TIME[T(n)^c]$$

if and only if there is a set of density $2^{T^{-1}(n)}$ in $NP - P$, ([HIS-85]; Thm 6). In the case where T is a simple exponential, they use the proof of this theorem to prove that there exist sparse sets in $NP - P$ if and only if there exist tally sets in $NP - P$, ([HIS-85]; Cor 4).

For the results that follow, we deal with log*-sparse sets. To apply the Hartmanis, Immerman, Sewelson results we define $f(0) = 2$ and $f(n+1) = 2^{f(n)}$ and we take $T(n) = 2^{f(n)}$. Theorem 6 of [HIS-85] immediately applies, and the analogue of their Corollary 4 shows that the higher deterministic and nondeterministic classes are separated if and only if there are well-spaced log*-sparse tally sets in $NP - P$. Furthermore by careful modifications of all of these proofs, we can prove these same results with NP replaced by *parity-P*. We summarize these facts in the following lemma.

Lemma 2, a). *There is a log*-sparse tally set, S, in $NP - P$ if and only if*

$$\bigcup_{c>0} NTIME[(2^{f(n)})^c] \neq \bigcup_{c>0} TIME[(2^{f(n)})^c].$$

Furthermore, $S \subset T =_{def} \{ 1^{f(n)} \mid$ n is a natural number$\}$.

b). *There is a log*-sparse tally set, S, in parity-P - P if and only if*

$$\bigcup_{c>0} ParityTIME[(2^{f(n)})^c] \neq \bigcup_{c>0} TIME[(2^{f(n)})^c].$$

Furthermore, $S \subset T =_{def} \{ 1^{f(n)} \mid$ n is a natural number$\}$.

Since the separation of the higher complexity classes implies the existence of log*-sparse tally sets in $NP - P$ and in parity-P - P, we take such separations as hypotheses in our final two theorems, which explain why extensions of Theorems 1 and 2 to *parity-P* and to word decreasing query self-reducible sets are unlikely.

Theorem 3. *If* $\bigcup_{c>0} ParityTIME[(2^{f(n)})^c] \neq \bigcup_{c>0} TIME[(2^{f(n)})^c]$, *then there are sets that are in parity-P* $\cap$ *p-cheatable but are not in P.*

Proof. The proof is simple, and in fact yields a stronger result. Let S and T be the sets guaranteed by part (b) of Lemma 2. Note that T is both polynomially decidable and log*-sparse, and that S itself is in *parity-P* but not in P. We prove that S is (n for 1) p-cheatable *for every* n.

Note that for any string y, deterministically simulating the nondeterministic parity machine used to define S takes time $2^{p(|y|)}$, which is less than $2^{2^{|y|}}$, where p is a polynomial bound on the runtime of the parity machine. Therefore, given any n-tuple $(x_1, x_2, ..., x_n)$, in increasing order, we can calculate membership in S for all of the x_i's as follows. First discard all x_i's which are not in T. Since T is polynomially decidable this takes time at most polynomial in the sum of the lengths of all x_i. Call the largest remaining string x_m. Next discard all strings x_i such that $2^{|x_i|} \geq |x_m|$, and finally test all remaining strings $x_j < x_m$ for membership in S by *brute force*, by simulating the nondeterministic algorithm which defines S. Since such x_j's all satisfy $2^{|x_j|} < |x_m|$, and since there are at most $|x_m|$ such strings, the latter work can *all* be done in time polynomial in $|x_m|$. Finally, to decide whether $x_m \in S$, query the oracle. ∎

We now turn to the question of whether we can extend Theorems 1 and 2 to show that all wdq self-reducible sets that are p-cheatable are in P. We believe that the answer is "no," but we also expect that a proof will be difficult. To disprove the result one would need to construct a set that is wdq self-reducible and p-cheatable, but not in P. All known p-cheatable sets are, as above, polynomially (many-one) reducible to sparse or co-sparse sets. But, as shown by Meyer and by Balcázar, any wdq self-reducible set that is polynomially Turing reducible to a sparse or co-sparse set is in Σ_2^p. Thus, to disprove the result, one would presumably have either to construct a new type of p-cheatable set or to prove $\Sigma_2^p \neq P$. As a partial step in this direction, under an hypothesis similar to that of Theorem 3, we can construct wdq self-reducible sets that are ($2^k - 1$ for k) p-cheatable, but not in P.

In [GJY-87b], we showed how to construct sets that are ($2^k - 1$ for k) p-cheatable, not sparse, and not *close* to any polynomially decidable set. We now show that similar sets can be made wdq self-reducible and yet not polynomially decidable.

Theorem 4. *If* $\bigcup_{c>0} NTIME[(2^{f(n)})^c] \neq \bigcup_{c>0} TIME[(2^{f(n)})^c]$, *then there is a set* A *that is*

i) wdq self-reducible,

ii) ($2^k - 1$ for k) p-cheatable,

iii) in P/poly,

iv) in Σ_2^p, *but*

v) not in P.

Proof. Let S and T be the sets guaranteed by part (a) of Lemma 2. As in the proof of Theorem 3, we observe that T is both log*-sparse and decidable in polynomial time. Let M be a nondeterministic polynomial time Turing machine that decides membership in S in time $p(|x|)$. We assume without loss of generality that p is strictly increasing and that all computation paths on input x have length $p(|x|)$. We will use M to describe a set, A, that is wdq self-reducible and ($2^k - 1$ for k) p-cheatable, but is not polynomially decidable.

The set A will consist of strings of the form $x\#c_x$ where $\#$ is a marker and c_x is a computation path of M on input x. We define

$$A = \{x\#c : \exists \tilde{c} \leq c \text{ and } \tilde{c} \text{ is an accepting computation of } M \text{ on input } x \},$$

where $\leq$ denotes the lexicographic ordering.

Claim 1: A is wdq self-reducible. To see this, observe that to decide whether $a = x\#c \in A$ we can check whether $x\#\tilde{c} \in A$ where $\tilde{c}$ is the lexicographic predecessor of c. If $x\#\tilde{c}$ is in A, then accept a. If $x\#\tilde{c}$ is not in A or if $x\#c$ has no such lexicographic predecessor, then check whether c is an accepting computation of M on input x. If it is, then accept a. If it is not, then reject a.

Claim 2: A is ($2^k - 1$ for k) p-cheatable. In fact, we make a stronger claim: there is an oracle Turing machine that, given *any* $2^k - 1$ elements (k a *variable*), determines in time polynomial in their cumulative length whether or not each is in A, and asks no more than k queries. Assume that we are given inputs $a_1, a_2, ..., a_{2^k-1}$ in increasing order. First find the corresponding x values, and let a_m be the largest of the a_i's for which the corresponding x value is in T. Those a_i's bigger than a_m can be rejected, so next consider those of length $|a_m|$. Any of these for which the corresponding x value is not in T can be rejected. The remaining ones must all have the same x value since T is log*-sparse. Sort the remaining elements of length $|a_m|$ lexicographically and perform a binary search, asking membership questions in A. This will require at most k queries to an oracle for A and will determine membership in A for all a_i of length $|a_m|$.

Next consider the a_i's of shorter lengths. These have different x values than a_m. For any x not in T the corresponding a_i can be rejected. Recalling that T is $\log^*$ sparse, the x values for the remaining a_i's must be widely spaced. In fact, T is so uniformly sparse that if $x_1 < x_2$ are both in T, then *all* of x_1's computation paths can be examined in time polynomial in $|x_2\#c|$. Therefore, for the remaining a_i's of some fixed length less than $|a_m|$, we can test whether the largest is in A by a *brute force* computation to see if it's x value is in S, and if it is in S, just as for the a_i's of length a_i, we can determine membership of the remaining a_i's by binary search.

Claim 3: A is in $P/poly$. To show this, we need to show that A is polynomially Turing reducible to a sparse set. This is easily seen to be the case because, to determine whether a is in A for $a = x\#c$ and $x \in T$, we simply need to know which, if any, of $M(x)$'s computation paths is the least accepting path. This information can be coded into a sparse oracle.

Claim 4: A is *not* polynomially decidable. If A were polynomially decidable, then the following algorithm would determine membership in S in polynomial time. To decide whether $s \in S$ we test whether the string $s\#c$, where c is the lexicographically greatest computation path of $M(s)$, is in A. ∎

One last observation should be made about this theorem. Since all *known* wdq self-reducible sets are in $Pspace$, we might try to replace the current hypothesis by the simpler hypothesis that $Pspace \neq P$. If this could be done, we would have as a corollary that

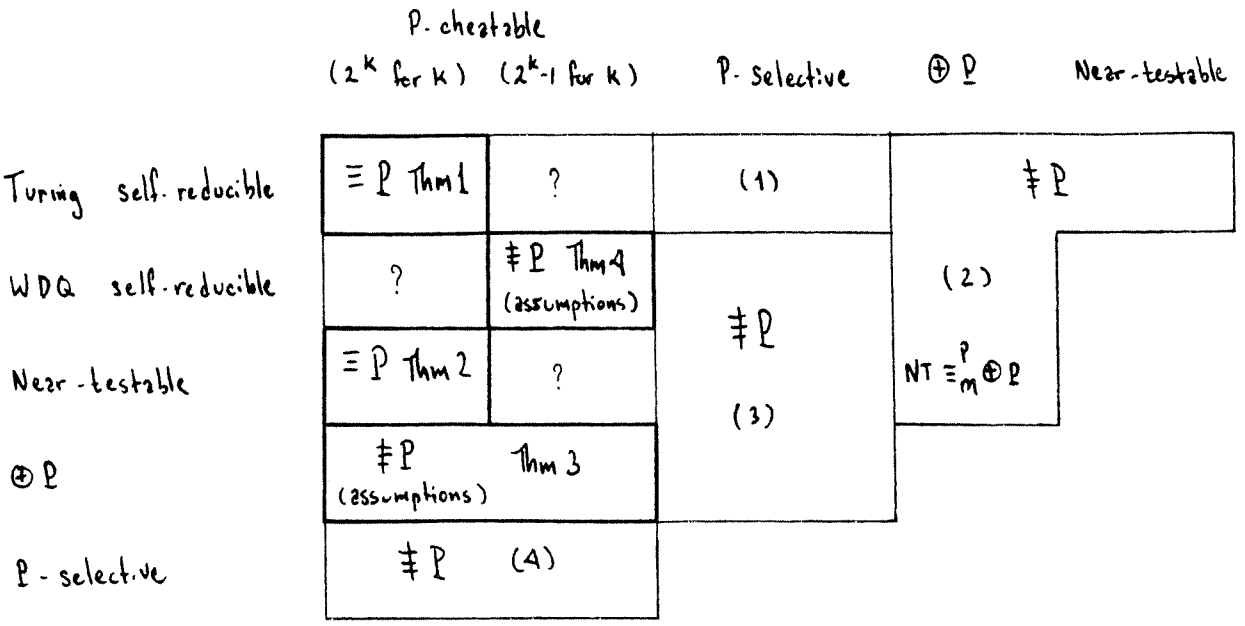
$$Pspace \neq P \text{ implies } \Sigma_2^P \neq P,$$

since any wdq set that is in $P/poly$ is in Σ_2^P. To make our job easier we would be willing to drop the $P/poly$ condition from the theorem, but again we hit a stumbling block. All *known* ($2^k - 1$ for k) p-cheatable sets are in $P/poly$.

4. SUMMARY AND CONCLUSIONS.

In the past, most notions of self-reducibility that have been studied have been refinements of Turing self-reducibility. Many natural sets have been shown to be self-reducible, and in general it is not believed that polynomially self-reducible sets must be polynomially decidable. Recently, several new notions of self-reduction have been introduced with the study of wdq self-reducible sets, near-testable sets, and p-cheatable sets. Again, examples can be found of sets that are self-reducible via these types of reductions and yet are probably not polynomially decidable. However, an interesting situation arises when one considers sets that exhibit more than one type of self-reducibility; often the combination of self-reducibilities causes the set to be polynomially decidable. The four results presented in this paper explore the possibility of using combinations of self-reducibility properties to characterize polynomial computation. They add to the knowledge of the interplay of these reducibilities that has come directly from the study of near-testable and p-cheatable sets. We combine the results of this paper with previously known results in the table below.

Table 1



(1) Selman [Se-82b] has shown that any p-selective set that is positive truth-table self-reducible is polynomially decidable.

(2) The classes NT and $\oplus P$ are known to be $\leq_m^P$-equivalent. The set $\oplus SAT$ is easily seen to be $\leq_m^P$-complete for $\oplus P$ and, like SAT, it is Turing self-reducible. Therefore, unless $P = \oplus P$, $\oplus P \cap \{\textit{Turing self-reducible sets}\} \neq P$. If we consider the image of $\oplus SAT$ under the uniform many-one reduction that exists between NT and $\oplus P$, then this set is also Turing self-reducible. Thus, unless $NT = P = \oplus P$, $NT \cap \{\textit{Turing self-reducible sets}\} \neq P$. See [GHJY-87] for details.

(3) If there is a log*-sparse set in P that is not P-printable, then there is a near-testable p-selective set that is not polynomially decidable. This set is also wdq self-reducible and is in $\oplus P$ because of the trivial containments. Again, the details of this construction are contained in [GHJY-87].

(4) A straight forward diagonalization can be used to construct a log*-sparse set that is p-cheatable and p-selective, but not polynomially decidable.

Combinations of Reducibilities that Characterize Polynomial Time

In conclusion, our results show that the notions of wdq self-reducibility, p-cheatability and membership in P and $P/poly$ are tightly intertwined. As shown in Table 1, questions about these connections remain open. We believe that solutions to the following questions would be particularly useful in providing a better understanding of polynomial computations and self-reducibility.

Question 1: Are there sets that are wdq self-reducible, p-cheatable, but not polynomially decidable? A positive answer would give us a version of Theorem 4 that does not rely on the existence of log*-sparse sets.

Question 2: When are p-cheatable sets polynomially reducible to sparse sets? We expect a complete characterization to be difficult.

Acknowledments: We would like to thank Richard Beigel and Bill Gasarch for introducing us to the notions of p-cheatable sets. Thanks go to Jose Balcázar and to Alan Seman for discussions on self-reducibilities, and also to Larry Ruzzo for his observations concerning the existence of log*-sparse sets.

5. BIBLIOGRAPHY.

[AR-87] E. Allender and R. Rubinstein, "P-printable sets," *Manuscript*, submitted for publication, (1987), 1-20.

[AG-87] A. Amir and W. Gasarch, "Polynomially terse sets," *Proc Second Annual Structure in Complexity Conference*, IEEE Computer Society (1987), 22-27.

[Ba-87] J. Balcázar, "Self-reducibility," *Symp on the Theory of Automata and Computing*, Springer Verlag Lecture Notes in Computer Science (1987).

[BBS-86] J. Balcázar, R. Book, and U. Schöning, "The polynomial time hierarchy and sparse oracles," *J. ACM* **33** (1986), 603-617.

[Be-87a] R. Beigel, "Bi-immunity and separation results for cheatable sets," *Manuscript* (June, 1987), 1-15.

[Be-87b] R. Beigel, "Query-limited Reducibilities," *Stanford PhD Dissertation* (September, 1987).

[BGGO-86] R. Beigel, W. Gasarch, J. Gill and J. Owings, "Terse, super-terse and verbose sets," *University of Maryland Technical Report*, TR-1806 (1986), 1-25.

[BGH-87] R. Beigel, W. Gasarch, L. Hay, "Bounded Query Classes and the Difference Hierarchy," *University of Maryland Technical Report*, TR-1847 (1987), 1-26.

[BGO-87] R. Beigel, W. Gasarch, and J. Owings, "Terse sets and verbose sets," *Recursive Function Theory: Newsletter* **36** (1987), 13-14.

[Be-78] P. Berman, "Relationship between density and deterministic complexity of NP-complete languages," *Symposium on the Math. Found. of Comput. Sci., Springer Verlag Lecture Notes in Computer Science* **62** (1978), 63-71.

[BH-77] L. Berman and J. Hartmanis, "On isomorphisms and density of NP and other complete sets," *SIAM J Comput,* **6** (1977), 305-322.

[Bo-74] R. Book, "Tally languages and complexity classes," *Inform and Control,* **26** (1974), 186-193.

[Fo-79] S. Fortune, "A note on sparse complete sets," *SIAM J Comput* **8** (1979), 431-433.

[GJY-87a] J. Goldsmith, D. Joseph, and P. Young, "Self-reducible, p-selective, near-testable, and p-cheatable sets: the effect of internal structure on the complexity of a set," (abstract), *Proc Second Annual Structure in Complexity Conference* IEEE Computer Society (1987), 50-59. Full paper distributed as *University of Washington Technical Report #* 87-06-02, 1-22.

[GJY-87b] J. Goldsmith, D. Joseph, and P. Young, "A note on bi-immunity and p-closeness of p-cheatable sets in *P/poly*," *University of Washington Technical Report # 87-11-05* (1987), 1-13.

[GHJY-87] J. Goldsmith, L. Hemachandra, D. Joseph, and P. Young, "Near-testable sets," *University of Washington Technical Report # 87-11-06* (1987), 1-22.

[HIS-85] J. Hartmanis, N. Immerman, V. Sewelson, "Sparse sets in $NP - P$: Exptime versus Nexptime," *Inform and Control* **65** (1985), 158-181.

[He-87] L. Hemachandra, "On parity and near-testability: $P^A \neq NT^A$ with probability 1," *Cornell University Technical Report* 87-852 (July) 1987.

[Ko-83] K. Ko, "On self-reducibility and weak P-selectivity," *J Computer System Sci* **26** (1983), 209-221.

[Kr-86] M. Krentel, "The complexity of optimization problems," *Proc 18th Ann ACM Symposium Theory Computing* (1986), 60-76.

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

[MP-79] A. Meyer and M. Paterson, "With what frequency are apparently intractable problems difficult?" *MIT/LCS/TM-126* (1979).

[PZ-82] C. H. Papadimitriou, and S. T. Zachos, "Two Remarks on the Power of Counting," *Proc 6th GI Conference Theoretical Computer Science,* Springer Verlag Lecture Notes Computer Science **145** (1983), 269-275.

[Sc-86] U. Schöning, *Complexity and Structure*, Springer Verlag Lecture Notes Computer Science **211** (1986).

[Se-79] A. Selman, "P-selective sets, tally languages, and the behavior of polynomial reducibilities on NP," *Math Systems Theory* **13** (1979), 55-65.

[Se-82a] A. Selman, "Analogues of semi-recursive sets and effective reducibilities to the study of NP complexity," *Inform and Control* **52** (1982), 36-51.

[Se-82b] A. Selman, "Reductions on NP and P-selective sets," *Theoretical Computer Science* **19** (1982), 287-304.

[Tr-70] B. Trakhtenbrot, "On autoreducibility," *Dokl Akad Nauk SSSR* **11** (1970).

[VV-85] L. Valiant and V. Vazirani, "NP is as easy as detecting unique solutions," Proc 17th ACM Symp Theory Computing 1985, 458-463.

6. APPENDIX: SOME PROPERTIES OF NEAR-TESTABLE, P-CHEATABLE AND WDQ SELF-REDUCIBLE SETS

A. Some Properties of Turing Self-reducible Sets

Theorem. *If A is self-reducible, then $A \in PSPACE$.*

Theorem [Se-82b]. *If A is positive truth-table self-reducible and p-selective, then $A \in P$.*

Meyer, Paterson [MP-79] and Ko [Ko-83] have given more general definitions of self-reducibility.

Definition [Ko-83]. *A partial ordering $\prec$ on Σ^* is polynomially well-founded and length related (for short: polynomially related) if there is a polynomial p such that*

(i) $x \prec y$? is polynomially testable,

(ii) $x \prec y$ implies that $|x| \leq p(|y|)$, and

(iii) the length of a $\prec$-descending chain is shorter than p of the length of its maximal element.

Definition [Ko-83]. *A set $A \subseteq \Sigma^*$ is tt self-reducible if there is a polynomially related ordering $\prec$ on Σ^*, a polynomially computable function f and a polynomial p such that, for all $x \in \Sigma^*$*

(i) $f(x)$ is a tt-condition $< \alpha, < x_1, ..., x_k >>$ where α is a k-ary boolean formula that can be evaluated in time $p(|x|)$ and $x_i \prec x$ for all $i \leq k$, and

(ii) $x \in A$ iff $\alpha(\chi_A(x_1), ..., \chi_A(x_k)) = 1$, where χ_A is the characteristic function for A.

If the truth-table condition α is a simple disjunction or conjunction, then A is said to be *disjunctive (or conjunctive) self-reducible*. SAT is obviously disjunctive self-reducible, and it has been conjectured that all NP-complete sets are disjunctive self-reducible. This, however, remains an open question.

Ko said that a set is *Turing self-reducible* if there is a polynomial time oracle machine CH such that (i) M^A is a recognizer for A, and (ii) on input x all of M's queries are to elements that precede x in the $\prec$-ordering. Obviously, any set that is tt self-reducible is Turing self-reducible.

B. Some Properties of Wdq Self-reducible Sets

Proposition [Ba-87]. *If A is wdq self-reducible, then $A \in EXPTIME$.*

Theorem [Ba-87]. *If $A \in PSPACE/poly$ and A is wdq self-reducible, then $A \in PSPACE$.*

Theorem [Ba-87]. *If $A \in \Sigma_i^P/poly$ and A is wdq self-reducible, then $\Sigma_2^P(A) \subseteq \Sigma_{i+2}^P$.*

C. Some Properties of P-cheatable Sets

Theorem [BGGO-87]. *All p-cheatable sets are decidable.*

Theorem [Be-87a]. *$P \neq NP$ implies that no NP-complete set is p-cheatable.*

Theorem [AG-87]. *Let $Dtime(T(n))$ be any deterministic time class. There is a 2-pcheatable set A that is bi-immune w.r.t. $Dtime(T(n))$. That is neither A nor $\overline{A}$ has a $Dtime(T(n))$ decidable subset.*

Theorem [Be-87]. *1-pcheatable sets cannot be polynomially bi-immune.*

Theorem [GJY-87b]. *There are 1-pcheatable sets that are not p-close.*

Theorem [GJY-87b]. *There are 2-pcheatable sets that are bi-immune for arbitrary time classes, in $P/poly$, yet not p-close.*

Theorem [GJY-87a]. *There are ($2^k - 1$ for k) p-cheatable sets that are recursively bi-immune, in $P/poly$, yet not p-close.*

D. Some Properties of Near-testable Sets

Observations [GJY-87a].

i) $P \subseteq NT = coNT \subseteq EXPTIME \cap PSPACE$,

ii) *if $A \in NT$ and A is polynomially sparse, then $A \in P$,*

iii) *there is $A \in EXPTIME$, such that $A \notin NT$, and*

iv) *if A is sparse and $A \in NT$, then $A \in P$.*

Theorem [GHJY-87], [He-87]. *$NT \subseteq \oplus P$ and every set in $\oplus P$ is $\leq_m^P$-reducible to a set in NT.*

Corollary [GHJY-87], [He-87]. *$NT \equiv_m^P \oplus P$.*

Corollary [GHJY-87], [He-87]. *$\oplus P \neq P$ if and only if $NT \neq P$.*

Observation [GHJY-87]. *The same proof that shows that SAT is $\leq_m^P$-complete for $\#P$ can be used to show that parity-SAT is $\leq_m^P$-complete for $\oplus P$. Parity-SAT has received a lot of attention recently. In particular, Valiant and Vazirani ([VV-85]) have shown that parity-SAT is NP-hard under randomized reductions. Furthermore, if f is a reduction from parity-SAT to a set in NT (as in the Theorem above), then $f(parity{-}SAT)$ is $\leq_m^p$-complete for NT.*

Theorem [GHJY-87], [He-87]. *If $S \in NT \cap P/poly$, then there is a tally set $T \in 2^{O(n^k)}$ such that $S \in P^T$.*

Theorem [GHJY-87], [He-87]. *Relative to a random oracle A, $\oplus P^A - PP^A \neq \emptyset$ with probability one.*

Corollary [GHJY-87], [He-87]. *Relative to a random oracle A, $NT - PP^A \neq \emptyset$ with probability one.*