On the Structure of Polynomial Time Reducibility

RICHARD E. LADNER

University of Washington, Seattle, Washington

ABSTRACT Two notions of polynomial time reducibility, denoted here by $\leq_T^{\mathbf{P}}$ and $\leq_m^{\mathbf{P}}$, were defined by Cook and Karp, respectively The abstract properties of these two relations on the domain of computable sets are investigated. Both relations prove to be dense and to have minimal pairs. Further, there is a strictly ascending sequence with a minimal pair of upper bounds to the sequence. Our method of showing density yields the result that if $\mathbf{P} \neq \mathbf{NP}$ then there are members of $\mathbf{NP} - \mathbf{P}$ that are not polynomial complete

KEY WORDS AND PHRASES polynomial time computation, Turing reducibility, many-one reducibility

CR CATEGORIES 5 25

1. *Introduction*

Cook [3] and Karp [6] have introduced two notions of polynomial time reducibility. They show quite effectively that the notion of reducibility is a useful tool in classifying the complexity of problems. They show that a wide class of important problems all have the same time complexity (modulo a polynomial) by showing that all the problems are reducible to each other in polynomial time We propose to study the abstract properties of their two reducibilities thought of just as relations between problems. We pay particular attention to properties that might shed some light on the question of whether or not every problem computable in nondeterministic polynomial time is also computable in deterministic polynomial time. We notice further that the properties we show are true of polynomial time reducibility hold true also of a wide variety of subrecursive reducibilities, including log space, elementary, and primitive recursive. We fix the alphabet $\Sigma = \{0, 1\}$ as the alphabet in which all problems are encoded, so that a *problem* is simply a subset of Σ^*. We let $<$ be the natural order on Σ^* ($\lambda < 0 < 1 < 00 < 01 < \cdots$), where λ represents the empty string. In general we consider only solvable problems, that is, computable subsets of Σ^*. If $x \in \Sigma^*$ we let $|x|$ denote the length of x When confusion will not arise we adopt the habit of identifying a problem with its characteristic function, namely, if $A \subseteq \Sigma^*$ then $A(x) = 1$ if $x \in A$ and $A(x) = 0$ if $x \notin A$. Our basic model of computation is the multitape Turing machine. All such machines are assumed to be deterministic unless otherwise specified. A Turing machine T (deterministic or nondeterministic) *runs in polynomial time* if there is a polynomial function q such that for every input of length n any computation sequence of T halts in $q(n)$ or fewer moves. Define $\mathbf{P}$ ($\mathbf{NP}$) to be the class of problems recognized by deterministic (nondeterministic) Turing machines which run in polynomial time.

This research was supported in part by the National Science Foundation under Grant GJ-34745x.

Many of the results presented here are found in a different form in "Polynomial time reducibility," Proc Fifth Annual ACM Symp on Theory of Computing, 1973, pp 122–129, and in "Subrecursive reducibilities," Tech Rep #73-03-13, Department of Computer Science, U. of Washington, Seattle, Wash.

Author's address· Department of Computer Science, University of Washington, Seattle, WA 98195.

The two definitions of polynomial time reducibility of Karp and Cook are just time bounded versions of many-one reducibility ($\leq_m$) and Turing reducibility ($\leq_T$) defined by Post [18]. A set A is many-one reducible to a set B if there is a computable function f such that $x \in A$ if and only if $f(x) \in B$. To obtain Karp's notion of polynomial time reducibility we simply require f to be computable in polynomial time. To be more precise *A is many-one reducible to B in polynomial time* ($A \leq_m^{\mathbf{P}} B$) if there is a function f which is computable by a Turing machine running in polynomial time such that $x \in A$ if and only if $f(x) \in B$. The notion of Turing reducibility can be obtained by considering *oracle Turing machines*. An oracle Turing machine is a multitape Turing machine with an oracle tape. There are special states Q, YES, and NO such that if the machine enters state Q then the machine enters state YES if the current contents of the oracle tape is in the oracle set and enters state NO otherwise. Strictly speaking, an oracle Turing machine has two "inputs"—the input written on the input tape and the oracle set (which is not written anywhere). Such a machine runs in polynomial time if there exists a polynomial q such that for any input of length n and any oracle set X the machine halts within $q(n)$ steps. A set A is Turing reducible to B if and only if there is an oracle Turing machine which recognizes A when presented with the set B as its oracle. To obtain Cook's notion of polynomial time reducibility we just require that the oracle Turing machine run in polynomial time. Hence *A is Turing reducible to B in polynomial time* ($A \leq_T^{\mathbf{P}} B$) if and only if there is an oracle Turing machine $\mathbf{M}$ which runs in polynomial time such that x is in A exactly when $\mathbf{M}$ halts in an accepting state with input x and oracle B.

The work of Post and others in recursive function theory has demonstrated that the notions of many-one reducibility and Turing reducibility are effective tools in classifying unsolvable problems. If we think of the problems in $\mathbf{P}$ as the "realistically computable problems," then the polynomial time bounded versions of these reducibilities may serve as useful tools in classifying the computable yet not realistically computable problems. Indeed Cook [3] has shown that the set, S, of satisfiable formulas has the property that $S \in \mathbf{NP}$ and if $A \in \mathbf{NP}$ then $A \leq_T^{\mathbf{P}} S$. Karp [6] noticed that Cook's proof actually yields the fact that every member of $\mathbf{NP}$ is $\leq_m^{\mathbf{P}} S$. In some sense S is as hard to compute (modulo polynomial time) as any member of $\mathbf{NP}$, albeit if $\mathbf{P} = \mathbf{NP}$ then S is realistically computable. Karp [6] calls sets with the properties of S polynomial complete. Calling such sets polynomial complete is quite fitting because it parallels the definition of complete set in recursive function theory (see Rogers [21]). A set B is m-complete (T-complete) if it is recursively enumerable and if A is any recursively enumerable set then $A \leq_m B$ ($A \leq_T B$). This analogy is strengthened if we notice that the characterization of $\mathbf{NP}$ as the class of all sets A such that there is a polynomial time computable relation R and a polynomial q where $A = \{x : \exists y \text{ of length } \leq q(|x|) \text{ where } (x, y) \in R\}$ (see Cook [3] or Karp [6]) parallels the characterization of the recursively enumerable sets as the class of all sets A such that there is a computable relation R where $A = \{x : \exists y \text{ where } (x, y) \in R\}$ (see Rogers [21]). We say that a set B is *polynomial m-complete* (*T-complete*) if $B \in \mathbf{NP}$ and if $A \in \mathbf{NP}$ then $A \leq_m^{\mathbf{P}} B$ ($A \leq_T^{\mathbf{P}} B$). Cook [3], Karp [6], Sethi [22], Stockmeyer [23], Ullman [25], and others have exhibited a wide variety of polynomial m-complete problems including satisfiability, 0-1 integer programming, register allocation, planar 3-colorability, and scheduling problems.

Define $A <_m^{\mathbf{P}} B$ ($A <_T^{\mathbf{P}} B$) if $A \leq_m^{\mathbf{P}} B$ and $B \not\leq_m^{\mathbf{P}} A$ ($A \leq_T^{\mathbf{P}} B$ and $B \not\leq_T^{\mathbf{P}} A$). Further $A \equiv_m^{\mathbf{P}} B$ ($A \equiv_T^{\mathbf{P}} B$) if $A \leq_m^{\mathbf{P}} B$ and $B \leq_m^{\mathbf{P}} A$ ($A \leq_T^{\mathbf{P}} B$ and $B \leq_T^{\mathbf{P}} A$). Since both $\leq_m^{\mathbf{P}}$ and $\leq_T^{\mathbf{P}}$ are reflexive and transitive relations, the relations $\equiv_m^{\mathbf{P}}$ and $\equiv_T^{\mathbf{P}}$ are equivalence relations. The equivalence classes are called the *polynomial m-degrees* and *polynomial T-degrees* respectively. The polynomial m-degrees (T-degrees) are partially ordered by the order induced by $\leq_m^{\mathbf{P}}$ ($\leq_T^{\mathbf{P}}$) on sets. Structural questions about the relations $\leq_m^{\mathbf{P}}$ and $\leq_T^{\mathbf{P}}$ reduce to structural questions about the respective induced ordering on the polynomial m-degrees and polynomial T-degrees.

In Section 2 we discuss the basic facts about m and T reducibility in polynomial time and examine the work of others that exposes properties of these relations. In Section 3 we show that the polynomial m-degrees and T-degrees are dense. Our method shows that if $\mathbf{P} \neq \mathbf{NP}$ then there must exist members of $\mathbf{NP}$ that are neither in $\mathbf{P}$ nor polynomial T-complete. Indeed, it turns out that if $\mathbf{P} \neq \mathbf{NP}$ then there exist $\leq_T^{\mathbf{P}}$-incomparable members of $\mathbf{NP}$, that is, problems A and B both in $\mathbf{NP}$ such that $A \nleq_T^{\mathbf{P}} B$ and $B \nleq_T^{\mathbf{P}} A$. In Section 4 we construct computable sets A and B neither of which is computable in polynomial time but if $C \leq_T^{\mathbf{P}} A$ and $C \leq_T^{\mathbf{P}} B$ then C is computable in polynomial time. Hence the polynomial T-degrees of A and B form a minimal pair. In Section 5 we show that neither the polynomial m-degrees nor polynomial T-degrees form a lattice by constructing a sequence $C_0, C_1, \cdots$ of computable sets and a pair A, B of computable sets such that: (i) $C_i \leq_m^{\mathbf{P}} C_{i+1}$ for all i, (ii) $C_{i+1} \nleq_T^{\mathbf{P}} C_i$ for all i, (iii) $C_i \leq_m^{\mathbf{P}} A$ and $C_i \leq_m^{\mathbf{P}} B$ for all i, (iv) if $D \leq_T^{\mathbf{P}} A$ and $D \leq_T^{\mathbf{P}} B$ then $D \leq_m^{\mathbf{P}} C_i$ for some i. As a consequence neither the polynomial m-degrees nor T-degrees of A and B can have a greatest lower bound. In Section 6 we show how to conclude these same sorts of results for other subrecursive reducibilities.

2. *Basics*

From this point on we restrict our attention to just the computable sets. When we speak of the polynomial m- or T-degrees we speak only of the polynomial m- or T-degrees of computable sets. We should first notice that Turing reducibility in polynomial time is a generalization of many-one reducibility in polynomial time; that is, if $A \leq_m^{\mathbf{P}} B$ then $A \leq_T^{\mathbf{P}} B$. Except for a certain pathology, neither kind of reducibility can distinguish members of $\mathbf{P}$; namely, if $A, B \in \mathbf{P}$ then $A \equiv_T^{\mathbf{P}} B$ and if $A, B \in \mathbf{P} - \{\emptyset, \Sigma^*\}$ then $A \equiv_m^{\mathbf{P}} B$. The polynomial m-degrees of $\emptyset$ and Σ^* are $\leq_m^{\mathbf{P}}$-incomparable. Because of this pathology we shall systematically ignore the existence of the sets $\emptyset$ and Σ^* when we speak of polynomial m-degrees. Hence we can say without hesitation that there is a least polynomial m-degree and, in fact, that its $\equiv_m^{\mathbf{P}}$-equivalence class is exactly $\mathbf{P} - \{\emptyset, \Sigma^*\}$. The set $\mathbf{P}$ is the least polynomial T-degree.

Every two polynomial m-degrees or T-degrees have a least upper bound in their respective orderings. Indeed, if A and B are problems then we can code both into a single problem $A \oplus B = \{0x : x \in A\} \cup \{1x : x \in B\}$ so that the polynomial m-degree (T-degree) of $A \oplus B$ is the least upper bound of the polynomial m-degrees (T-degrees) of A and B. We call the least upper bound of two polynomial m- or T-degrees their *join*. Hence the partial ordering of m-degrees and T-degrees are upper semilattices. The class $\mathbf{NP}$ is closed under many-one reducibility in polynomial time in the sense that if $B \in \mathbf{NP}$ and $A \leq_m^{\mathbf{P}} B$ then $A \in \mathbf{NP}$. This result seems to fail for $\leq_T^{\mathbf{P}}$. On the other hand $A \leq_T^{\mathbf{P}} \bar{A}$ for all A, which fails for $\leq_m^{\mathbf{P}}$ (Ladner et al. [9]).

Let $P_0, P_1, \cdots, T_0, T_1, \cdots$, and $\mathbf{M}_0, \mathbf{M}_1, \cdots$ be, respectively, effective enumerations of the Turing machine recognizers that run in polynomial time, the Turing machine transducers that run in polynomial time, and the oracle Turing machines that run in polynomial time. Such enumerations exist by attaching for each k to each appropriate kind of Turing machine a $(k + n^k)$-counter which will not allow the Turing machine to exceed making $k + n^k$ moves on any input of length n. Let P_i denote the set recognized by P_i as well as the Turing machine P_i. Similarly, let T_i denote the function computed by T_i and $\mathbf{M}_i(A)$ the set recognized by $\mathbf{M}_i$ with oracle A. We are not specific about the number of tapes, heads, and alphabet symbols we allow in our Turing machines because polynomial time computation is independent of these kinds of Turing machine specifications.

Define $\delta(A) = \{1^i : 1^i \notin \mathbf{M}_i(A)\}$. The set $\delta(A)$ represents sort of a diagonal set. Clearly $\delta(A)$ is computable if A is. Furthermore, $\delta(A) \nleq_T^{\mathbf{P}} A$ for if $\delta(A) = \mathbf{M}_e(A)$ then $1^e \in \delta(A)$ if and only if $1^e \notin \delta(A)$ We conclude that there is no maximum polynomial m-degree or T-degree, since $A \leq_m^{\mathbf{P}} A \oplus \delta(A)$ and $A \oplus \delta(A) \nleq_T^{\mathbf{P}} A$.

Some of the work of Axt [1] concerning primitive recursive reducibility can be generalized to yield the result that for any n there are n computable sets such that no one is polynomial T-reducible to the join of the remaining $n - 1$. More significantly, Machtey [12] shows that any countable partial ordering is embeddable in the primitive recursive degrees of recursive sets. His method can be applied to get the result that any countable partial ordering is embeddable in both the polynomial m-degrees and T-degrees of computable sets.

The work of Machtey [11–14] and Meyer and Ritchie [15] on honest primitive recursive or elementary recursive classes is related to the work here.

Machtey's method for showing that the honest subrecursive classes are dense is similar to our method of showing that the polynomial m- or T-degrees of computable sets are dense.

The existence of minimal pairs of subrecursive degrees of computable sets was first noticed by Machtey; in particular [11, Ths. 4.4, 4.7] directly imply that there exist minimal pairs of primitive recursive degrees of computable sets. These theorems are proved in detail as [12, Cor. 4.7] and [13, Th. 3.7]. Our method of constructing minimal pairs of polynomial T-degrees seems to be different from Machtey's method of constructing minimal pairs.

Lynch [10] defines and proves several interesting properties about an extremely general notion, "complexity determined reducibility," of which polynomial T-reducibility is a special case.

Let $N = \{0, 1, 2, \cdots\}$ and let $\langle \cdot, \cdot \rangle$ be a standard pairing function mapping $N \times N$ one-to-one onto N. For instance let $\langle x, y \rangle = \frac{1}{2}[(x + y)(x + y + 1)] + x$.

3. *Density Results*

Rather than going directly to the most general density result we prove a special case which illustrates the central ideas of these kinds of arguments. The method of proof in this section was inspired by the proof of Borodin et al. [2] of the fact that the Turing machine space complexity classes are dense.

THEOREM 1. *If B is computable and not in* $\mathbf{P}$ *then there exists a computable A such that* $A \notin \mathbf{P}$, $A \leq_m^{\mathbf{P}} B$, *and* $B \not\leq_T^{\mathbf{P}} A$.

PROOF. We construct a polynomial time bounded transducer T with range $\subseteq \{1\}^*$ in such a way that if we let $A = \{x \in B : |T(x)| \text{ is even}\}$ then $A \notin \mathbf{P}$ and $B \not\leq_T^{\mathbf{P}} A$. Since B cannot equal Σ^* we let $x_0 \in \Sigma^* - B$. The fact that $A \leq_m^{\mathbf{P}} B$ is witnessed by the transducer that prints x if $|T(x)|$ is even and x_0 if $|T(x)|$ is odd.

In some sense the machine T is "trying to satisfy" the following list of conditions: (0) $A \neq P_0$, (1) $B \neq \mathbf{M}_0(A)$, (2) $A \neq P_1$, (3) $B \neq \mathbf{M}_1(A)$, $\cdot$. For those x for which $T(x) = 1^i$, T is trying to satisfy the ith condition listed above. Think of T as processing successive inputs in the natural ordering of Σ^* To satisfy $A \neq P_i$, after some point T begins to output 1^{2i}. Hence A begins to look like B. Since $B \neq P_i$ then eventually some z must witness $A \neq P_i$, that is, eventually some z must exist such that $A(z) \neq P_i(z)$. In polynomial time, T tries to find that z and when it does T goes on to another task by printing 1^{2i+1}. In printing 1^{2i+1}, T is trying to satisfy the condition $B \neq \mathbf{M}_i(A)$ by forcing A to look empty beyond a certain point. Eventually some z must witness that $B \neq \mathbf{M}_i(A)$; otherwise we could conclude that $B \in \mathbf{P}$, which is not the case. In polynomial time, T tries to find that z and when it does, T may go on to another task by printing 1^{2i+2}. Thus, A in the end may be described as in Figure 1, where the entire line represents the natural ordering of Σ^*, beginning at the left with λ, and where the wavy lines represent A looking like B and the straight lines represent A looking like $\varnothing$.

We now give an informal description of T and rely on the reader's intuition to verify that T could be programmed on a Turing machine that runs in polynomial time. On input λ, T prints λ. On an input x of length n where $x \neq 0^n$, T prints $T(0^n)$. It remains to say what T does on inputs of the form 0^n where $n \geq 1$.

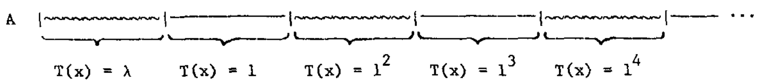

FIG. 1

On input 0^n where $n \geq 1$, T does the following:

1. For n moves, try to reconstruct the sequence $T(\lambda)$, $T(0)$, $T(0^2)$, $\cdots$. Let $T(0^m)$ be the last number of this sequence that is computed.

2. Case (i): $|T(0^m)|$ is even. Let $i = |T(0^m)|/2$. For n moves, try to find a z such that $A(z) \neq P_i(z)$. Do this by successively simulating T (to see what A is) and P_i on inputs $\lambda, 0, 1, 00, 01, \cdots$. If no such z is found, print 1^{2i}; otherwise, print 1^{2i+1}.

Case (ii): $|T(0^m)|$ is odd. Let $i = (|T(0^m)| - 1)/2$. For n moves try to find a z such that $B(z) \neq \mathbf{M}_i(A)(z)$. Do this by simulating an algorithm for B and the procedure $\mathbf{M}_i$ successively on inputs $\lambda, 0, 1, 00, 01, \cdots$. In simulating $\mathbf{M}_i$ on some input, should $\mathbf{M}_i$ enter state Q then T must be simulated with the contents of the oracle tape of $\mathbf{M}_i$ as its input to see if the string on the oracle tape is in A or not. The moves in this side calculation are counted among the n. If no such z is found, print 1^{2i+1}; otherwise, print 1^{2i+2}.

It should be noticed that $|T(0^n)| \leq |T(0^{n+1})| \leq |T(0^n)| + 1$ for all n. Furthermore, if $|T(0^n)| = i + 1$ for some n then the ith condition in the sequence above is satisfied as witnessed by the z found when computing $T(0^m)$ where m is the least number such that $|T(0^m)| = i + 1$. Hence the theorem is proved if $\{1\}^* = \text{Range } T$. This we show by induction on the length of elements of $\{1\}^*$. Clearly $\lambda \in \text{Range } T$. Assume $1^j \in \text{Range } T$. Let q be the least number such that $T(0^q) = 1^j$. Either $T(0^n) = T(0^q)$ for all $n \geq q$ or $1^{j+1} \in \text{Range } T$. We eliminate the former possibility. Assume $T(0^n) = T(0^q)$ for all $n \geq q$.

Case 1. j is even. Let $i = j/2$. In this case $A(x) = B(x)$ for all x of length greater than or equal to q. Since $B \notin \mathbf{P}$ then $A \neq P_i$. Let z be the least string in the natural ordering of Σ^* that witnesses this inequality. Let n be large enough so that on input 0^n, T can recompute the sequence $T(\lambda)$, $T(0)$, $\cdots$, $T(0^q)$ in n or fewer moves and T can also run its simulations of T and P_i through the input z in n or fewer moves. We must then have $T(0^n) = 1^{2i+1} = 1^{j+1}$, which was supposedly impossible.

Case 2. j is odd. Let $i = (j - 1)/2$. In this case, $x \notin A$ for all x of length greater than or equal to q. Hence $\mathbf{M}_i(A) \in \mathbf{P}$. Since $B \notin \mathbf{P}$ then $B \neq \mathbf{M}_i(A)$ and again there is a smallest z that witnesses this difference. By an argument similar to that of case 1 there is some $n \geq q$ such that T outputs 1^{j+1} on input 0^n. □

We can conclude immediately:

COROLLARY 1.1. *If* $\mathbf{P} \neq \mathbf{NP}$ *then there exist members of* $\mathbf{NP} - \mathbf{P}$ *that are not polynomial T-complete.*

PROOF. Let $B \in \mathbf{NP} - \mathbf{P}$ and let A be as in the theorem. Since $A \leq_m^{\mathbf{P}} B$ then $A \in \mathbf{NP}$ and since $B \nleq_T^{\mathbf{P}} A$ then A is not polynomial T-complete. □

Corollary 1.1 is analogous to the solution to "Post's problem" in recursive function theory. Post [18] posed the problem: Is every nonrecursive, recursively enumerable set Turing complete? A negative solution was discovered independently by Friedberg [4] and Muchnik [17]. A polynomial time bounded version of Post's problem would be: Is every member of $\mathbf{NP} - \mathbf{P}$ polynomial T-complete? In a way the problem only makes sense if $\mathbf{P} \neq \mathbf{NP}$.

Karp [6] mentions three problems in $\mathbf{NP}$ which are not known to be in $\mathbf{P}$ or to be polynomial m-complete; namely GRAPH ISOMORPHISM (the problem of whether two graphs are isomorphic), NONPRIMES (the problem of whether a binary string is a nonprime), and LINEAR INEQUALITIES (given an integer matrix M and integer vector d, whether there is a rational vector x such that $Mx \geq d$.) Corollary 1.1 allows the possibility that these problems may indeed be nonpolynomial T-complete problems in $\mathbf{NP}$ −

P. Assuming these three problems are not in **P** there is some evidence that the latter two problems are not polynomial m-complete. Pratt [19] has noticed that the complement of NONPRIMES is also in **NP**. Likewise, Karp [7] has noticed that the complement of LINEAR INEQUALITIES is in **NP**. On the other hand, Meyer [16] noticed that **NP** is closed under complement if and only if some polynomial m-complete problem has its complement in **NP**. Now, no complement of a known polynomial m-complete problem has yet been shown to be in **NP**. Thus it is not unlikely that both NONPRIMES and LINEAR INEQUALITIES are nonpolynomial m-complete members of $\mathbf{NP} - \mathbf{P}$.

One interesting property of the set A constructed in Theorem 1 is that it is the intersection of B with a set which can be computed in polynomial time, namely the set $\{x : | T(x) | \text{ is even}\}$. So if $\mathbf{P} \neq \mathbf{NP}$ then we could consider a polynomial T-complete problem like HAMILTON CIRCUIT (the problem of whether or not a graph has a cycle which includes each node exactly once; see Karp [6]). By our remarks there must be a set of graphs which can be determined in polynomial time such that the HAMILTON CIRCUIT problem restricted to that set would be in $\mathbf{NP} - \mathbf{P}$ yet not be polynomial T-complete. As we shall see from the next theorem, we are able to partition any problem not in **P** into two $\leq_T^{\mathbf{P}}$-incomparable problems.

Theorem 2. *If A and B are computable with $B \not\leq_T^{\mathbf{P}} A$ then there exists a set $D \in \mathbf{P}$ such that if $B_0 = B \cap D$ and $B_1 = B \cap \bar{D}$ then (i) $B \not\leq_T^{\mathbf{P}} A \oplus B_i$ for $i = 0, 1$, (ii) $B_i \not\leq_T^{\mathbf{P}} A$ for $i = 0, 1$.*

Proof. Let A and B be computable with $B \not\leq_T^{\mathbf{P}} A$. As in the proof of Theorem 1 we shall define a polynomial time computable function T with range $\subseteq \{1\}^*$. We set $D = \{x : | T(x) | \text{ is even}\}$.

With $B_0 = B \cap D$ and $B_1 = B \cap \bar{D}$ the machine T is "trying to satisfy" the following list of conditions: (0) $B \neq \mathbf{M}_0(A \oplus B_1)$, (1) $B \neq \mathbf{M}_0(A \oplus B_0)$, (2) $B_0 \neq \mathbf{M}_0(A)$, (3) $B_1 \neq \mathbf{M}_0(A)$, (4) $B \neq \mathbf{M}_1(A \oplus B_1)$, (5) $B \neq \mathbf{M}_1(A \oplus B_0)$, (6) $B_0 \neq \mathbf{M}_1(A)$, $\cdots$.

As before, imagine that T processes inputs in the natural ordering of Σ^*. When $T(x) = 1^i$ then T is trying to satisfy the ith condition in the above sequence. To satisfy an even numbered condition we make B_0 look like B and B_1 look like $\varnothing$, while to satisfy an odd numbered condition we make B_1 look like B and B_0 like $\varnothing$. Using algorithms for computing A and B, the construction of T follows lines exactly as in the construction of T in the proof of Theorem 1. The proof hinges on showing that T does not eventually get "stuck" generating some word in $\{1\}^*$. Should T get stuck generating 1^{4i} then for x sufficiently long $B_0(x) = B(x)$ and $B_1(x) = 0$. Since $B \not\leq_T^{\mathbf{P}} A$ then $B_0 \not\leq_T^{\mathbf{P}} A \oplus B_1$ and in particular $B_0 \neq \mathbf{M}_i(A \oplus B_1)$. Eventually T must realize this difference and try to satisfy the next condition on the list. Should T get stuck generating 1^{4i+2} then for all x sufficiently long $B_0(x) = B(x)$ and $B_1(x) = 0$. Since $B \not\leq_T^{\mathbf{P}} A$ then $B_0 \not\leq_T^{\mathbf{P}} A$ and in particular $B_0 \neq \mathbf{M}_i(A)$. Again, T must eventually realize this difference. The other two cases are symmetric. □

Instead of diagonalizing over the Turing reduction procedures that run in polynomial time we could have diagonalized over the many-one reduction procedures that run in polynomial time to obtain the following.

Corollary 2.1. *If A and B are computable with $A \neq \varnothing$, $A \neq \Sigma^*$, and $B \not\leq_m^{\mathbf{P}} A$ then there exists a set $D \in \mathbf{P}$ such that if $B_0 = B \cap D$ and $B_1 = B \cap \bar{D}$ then (i) $B \not\leq_m^{\mathbf{P}} A \oplus B_i$ for $i = 0, 1$, (ii) $B_i \not\leq_m^{\mathbf{P}} A$ for $i = 0, 1$.*

Proof. From the remarks above. □

Theorem 2 enables us to conclude that any polynomial T-degree "splits" over all smaller ones·

Corollary 2.2. *If $A <_T^{\mathbf{P}} B$ and A and B are computable then there exist computable sets C_0 and C_1 such that $A <_T^{\mathbf{P}} C_i <_T^{\mathbf{P}} B$ for $i = 0, 1$ and $B \equiv_T C_0 \oplus C_1$.*

Proof. We have $A \leq_T^{\mathbf{P}} B$ and $B \not\leq_T^{\mathbf{P}} A$. Let $i = 0$ or 1 and let $C_i = A \oplus B_i$ where B_i is as in Theorem 2. Obviously $A \leq_T^{\mathbf{P}} A \oplus B_i$. Since $B_i \not\leq_T^{\mathbf{P}} A$ then $A \oplus B_i \not\leq_T A$.

Since D is computable in polynomial time then certainly $B_i \leq_T^{\mathbf{P}} B$. Because $A \leq_T^{\mathbf{P}} B$ then $A \oplus B_i \leq_T^{\mathbf{P}} B$. Of course condition (i) of the theorem guarantees that $B \nleq_T^{\mathbf{P}} A \oplus B_i$. It is a simple matter to check that $C_0 \oplus C_1 \equiv_T B$ because $B_0 \cup B_1 = B$. □

Using Corollary 2.1 we can conclude that Corollary 2.2 also holds for many-one reducibility in polynomial time. The only question is showing $B \leq_m^{\mathbf{P}} C_0 \oplus C_1$. It would suffice to show that $B \leq_m^{\mathbf{P}} B_0 \oplus B_1$. This does not directly follow from the fact that $B_0 \cup B_1 = B$. However, $B \leq_m^{\mathbf{P}} B_0 \oplus B_1$ via the following many-one reduction procedure. On input x see if $x \in D$. If $x \in D$ print $0x$ and if $x \in \bar{D}$ print $1x$.

The density of the polynomial m- and T-degrees follows immediately:

COROLLARY 2.3. *The polynomial m- and T-degrees of computable sets are dense.*

PROOF. By the above. □

Under the assumption that $\mathbf{P} \neq \mathbf{NP}$ we can show that not only does $\mathbf{NP}$ contain $\leq_T^{\mathbf{P}}$-incomparable problems but that there is an infinite dense hierarchy of problems in $\mathbf{NP}$ based on Turing reducibility in polynomial time:

COROLLARY 2.4. *If $\mathbf{P} \neq \mathbf{NP}$ then there exist $\leq_T^{\mathbf{P}}$-incomparable members of $\mathbf{NP}$; that is, there exist problems B_0 and B_1 in $\mathbf{NP}$ such that $B_0 \nleq_T^{\mathbf{P}} B_1$ and $B_1 \nleq_T^{\mathbf{P}} B_0$.*

PROOF. Let $B \in \mathbf{NP} - \mathbf{P}$ and apply Theorem 2 to $\emptyset$ and B to obtain the polynomial time computable set D and the sets $B_0 = B \cap D$ and $B_1 = B \cap \bar{D}$. The sets B_0 and B_1 are $\leq_T^{\mathbf{P}}$-incomparable because B is not Turing reducible to either set in polynomial time yet B is Turing reducible to $B_0 \oplus B_1$ in polynomial time. Both B_0 and B_1 are in $\mathbf{NP}$ since $B \in \mathbf{NP}$ and $B_i \leq_m^{\mathbf{P}} B$ for $i = 0, 1$. □

It is interesting to note that no one has yet shown NONPRIMES and LINEAR INEQUALITIES to be $\leq_T^{\mathbf{P}}$-comparable problems. Perhaps they satisfy Corollary 2.4.

Let $\mathbf{Q}$ be the set of rational numbers in the closed interval $[0, 1]$.

COROLLARY 2.5. *If $\mathbf{P} \neq \mathbf{NP}$ then there is a family of problems $\{A_r : r \in \mathbf{Q}\} \subseteq \mathbf{NP}$ such that (i) $A_0 = \emptyset$ and A_1 is a polynomial m-complete problem, (ii) $p \leq q$ implies $A_p \leq_m^{\mathbf{P}} A_q$, (iii) $p \nleq q$ implies $A_p \nleq_T^{\mathbf{P}} A_q$.*

PROOF. Suppose $\mathbf{P} \neq \mathbf{NP}$. Let $A_0 = \emptyset$ and $A_1 =$ any polynomial m-complete set. Let $r_0, r_1, \cdots$ be an effective one-to-one enumeration of the rationals in $\mathbf{Q} - \{0, 1\}$. Let $R_i = \{r_j : j < i\} \cup \{0, 1\}$. We define by induction on i a family $\{A_r : r \in R_i\}$ in such a way that (i)–(iii) hold of the family when the p and q are restricted to R_i. The family $\{A_0, A_1\} = \{A_r : r \in R_0\}$ satisfies (i)–(iii). Assume $\{A_r : r \in R_i\}$ satisfies (i)–(iii). Let p and q be the unique members of R_i such that $p < r_i < q$ and no other pair p', q' from R_i satisfies the inequality, $p' < r_i < q'$, unless $p' \leq p$ and $q \leq q'$. Apply Theorem 2 to the sets A_p and A_q to get the polynomial time computable set D. Let $A_{r_i} = (A_q \cap D) \oplus A_p$. □

4. *A Minimal Pair*

Because of density we know that there can be no computable set $B \notin \mathbf{P}$ such that if $A \leq_T^{\mathbf{P}} B$ then either $A \in \mathbf{P}$ or $B \leq_T^{\mathbf{P}} A$. To put it another way, there can be no *minimal* polynomial T-degree of a computable set. For the same reason there can be no minimal polynomial m-degree of a computable set. The next question one might ask is whether or not there is a pair of computable sets neither of which are computable in polynomial time, but with the property that any set T-reducible to both of them in polynomial time is itself in $\mathbf{P}$. The polynomial T-degrees of two such sets are called a *minimal pair*. As we noted in Section 2, M. Machtey has observed already that other subrecursive degrees do have minimal pairs. Machtey has announced (unpublished) an independent proof of Theorem 3 based on his "honesty" methods. Of course any pair of sets whose polynomial T-degrees form a minimal pair must also have the property that their polynomial m-degrees form a minimal pair.

THEOREM 3. *There exist computable problems A and B neither of which is computable in polynomial time, but if $D \leq_T^{\mathbf{P}} A$ and $D \leq_T^{\mathbf{P}} B$ then D is computable in polynomial time.*

PROOF. We would like to construct sets A and B satisfying the following: (i) $A \neq P_i$ for all i, (ii) $B \neq P_i$ for all i, (iii) if $\mathbf{M}_i(A) = \mathbf{M}_j(B)$ then $\mathbf{M}_i(A)$ is computable in polynomial time, for all i and j (minimal pair conditions).

If we are not interested in making A and B computable then the theorem is quite easy to prove using methods found in [8]. We satisfy conditions (i) and (ii) by diagonalizing. To satisfy the $\langle i, j\rangle$-th minimal pair condition we assume that A and B are permanently defined at all arguments of length less than l. See if there exist an input x and a finite set $F \subseteq \{y : |y| \geq l\}$ such that $\mathbf{M}_i(A \cup F)(x) \neq \mathbf{M}_j(B)(x)$. If no such x and F exist then do nothing about the $\langle i, j\rangle$-th minimal pair condition. Otherwise, choose x and F to be minimal. Set $A = A \cup F$ and $l = l +$ sum of run times of $\mathbf{M}_i$ and $\mathbf{M}_j$ on input x. If $\mathbf{M}_i(A) = \mathbf{M}_j(B)$ then $\mathbf{M}_i(A) = \mathbf{M}_i(C)$ for some finite set C, namely, $C = \{y \in A : |y| < l'\}$ where l' is the value of l after considering the $\langle i, j\rangle$-th minimal pair condition. Hence $\mathbf{M}_i(A) \in \mathbf{P}$. It is clear that A and B are highly noncomputable, for in the construction of A and B we are able to satisfy a minimal pair condition in one large nonrecursive step. If we are to make A and B computable we must forsake the luxury of nonrecursive steps.

Let t be a strictly increasing recursive function such that for each i, if $n \geq i$ then $\mathbf{M}_i$ runs in time less than $t(n)$ on all input of length less than or equal to n. We construct A and B in stages with A^s and B^s being the respective values of A and B at the end of stage s. Together with A and B we construct a sequence $l_0, l_1, \cdots$ of integers and a polynomial time computable function $T : \{0, 1\}^* \to \{0, 1, c\}^*$. At stage $s + 1$ of the construction we determine $A(x)$ and $B(x)$ for all x where $l_s \leq |x| < l_{s+1}$. For convenience define $\bar{n}$ to be 1^n. The construction may be visualized partially as in Figure 2. In Figure 2 the two lines represent A and B as presented in the natural ordering of Σ^*. Segments that are straight lines are empty while segments that are wavy lines are possibly nonempty.

The sets A and B, the Turing machine T, and the sequence $l_0 < l_1 < \cdots$ have the following properties:

(a) $x \notin A$ whenever $l_{4s+1} \leq |x| < l_{4s+4}$ and $x \notin B$ whenever $l_{4s+3} \leq |x| < l_{4s+6}$ or $|x| < l_2$,

(b) $A(1^n) \neq P_s(1^n)$ if $n = l_{4s+1} - 1$, $B(1^n) \neq P_s(1^n)$ if $n = l_{4s+3} - 1$,

(c)
$$T(x) = \begin{cases} 0 & \cdots \text{ if } |x| < l_2, \\ 0c\bar{l}_{4s+1}cx_1c \cdots cx_kc & \cdots \text{ if } l_{4s+2} \leq |x| < l_{4s+4} \text{ and } x_1 < x_2 < \cdots < x_k \\ & \quad \text{are the members of } A \text{ of length less than } l_{4s+1}, \\ 1c\bar{l}_{4s+3}cx_1c \cdots cx_kc & \cdots \text{ if } l_{4s+4} \leq |x| < l_{4s+6} \text{ and } x_1 < x_2 < \cdots < x_k \\ & \quad \text{are the members of } B \text{ of length less than } l_{4s+3}, \end{cases}$$

(d) if $\mathbf{M}_i(A) = \mathbf{M}_j(B)$ then there exists s_0 such that if $s \geq s_0$ then. (1) if $s \equiv 0 \pmod 4$ and A' is any set with $A'(x) = A(x)$ for all x with $|x| < l_s$ then $\mathbf{M}_i(A')(x) = \mathbf{M}_i(A)(x)$ for all x of length less than l_s, (2) if $s \equiv 2 \pmod 4$ and B' is any set with $B'(x) = B(x)$ for all x with $|x| < l_s$ then $\mathbf{M}_j(B')(x) = \mathbf{M}_j(B)(x)$ for all x of length less than l_s.

Assuming we can construct A, B, T, and $\{l_i\}$ as described in (a)–(d), we explain why A and B satisfy the theorem. First, A and B are not polynomial time computable because

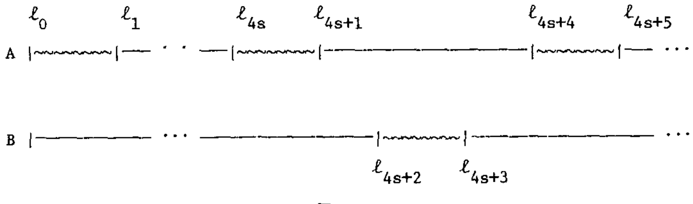

FIG. 2

of (b). Suppose $D \le_T^P A$ and $D \le_T^P B$. There are i and j such that $D = \mathbf{M}_i(A) = \mathbf{M}_j(B)$. Using T with the property described in (c) and properties (a) and (d) we can compute D in polynomial time. Consider the following algorithm. Let $|x| \ge l_{s_0}$. On input x: Compute $T(x)$. Now, $T(x) = ec\bar{l}cx_1c \cdots cx_kc$ for some $e, l, x_1, \cdots, x_k$. There are two cases:

Case 1. $e = 0$. Run $\mathbf{M}_i$ on input x. Should $\mathbf{M}_i$ enter state Q with y on the oracle tape then take the YES branch if $|y| < l$ and $y = x_i$ for some $i \le k$ and take the NO branch otherwise. Accept if and only if $\mathbf{M}_i$ does.

Case 2. $e = 1$. In this case run $\mathbf{M}_j$ on input x with exactly the same restrictions as described in case 1.

This algorithm does indeed compute $D(x)$ on all inputs x with $|x| > l_{s_0}$. Should $T(x) = 0c\bar{l}cx_1c \cdots cx_kc$ for some $l, x_1, \cdots, x_k$ then $l = l_{4p+1}$ and $|x| < l_{4p+4}$ for some p. Define A' by $A'(y) = A(y)$ for all y of length less than l_{4p+1} and $A'(y) = 0$ otherwise. By (a), $A(y) = 0$ for $l_{4p+1} \le y < l_{4p+4}$ whence $A'(y) = A(y)$ for all y of length less than l_{4p+4}. Now $4p + 4 > s_0$ and so by (d), $\mathbf{M}_i(A')(x) = \mathbf{M}_i(A)(x)$. But $\mathbf{M}_i(A')(x)$ is what is actually computed by the algorithm on input x. If $T(x) = 1c\bar{l}cx_1c \cdots cx_kc$ then $l = l_{4p+3}$ and $|x| < l_{4p+6}$ for some p. In this case the algorithm computes $\mathbf{M}_j(B')(x)$ where B' is defined by $B'(y) = B(y)$ if $y < l_{4p+3}$ and $B'(y) = 0$ otherwise. Again by (d) this is a computation of $D(x)$. Thus D is computable in polynomial time.

Roughly speaking, we can construct A, B, T, and $l_0, l_1, \cdots$ satisyfing (a)–(d) as follows. If $s \equiv 1$ or $3 \pmod 4$ then in l_{s+1} moves of a Turing machine that simulates the construction we can recover the entire construction on arguments of length less than l_s. This enables T to be computed in polynomial time. At each even numbered stage we try to force $\mathbf{M}_i(A)(x) \ne \mathbf{M}_j(B)(x)$ for some x with $\max\{i, j\} \le |x| < l_s$ provided we have not already done so. There are two ways we do this: if $s \equiv 0 \pmod 4$ then we try to extend A in such a way as to force $\mathbf{M}_i(A)(x) \ne \mathbf{M}_j(B)(x)$ for some x with $\max\{i, j\} \le |x| < l_s$, while if $s \equiv 2 \pmod 4$ then we try to extend B to do the same. Such extensions would amount to putting a subset of $\{y : l_s \le |y| < l_{s+1}\}$ into A or B. If no such extensions exist then we know the conditions of (d) must hold for i, j, and s. If we do force $\mathbf{M}_i(A) \ne \mathbf{M}_j(B)$ then the number $\langle i, j\rangle$ is canceled. Suppose $\mathbf{M}_i(A) = \mathbf{M}_j(B)$ in the limit. Let s be large enough so that $l_s > t(\max\{i, j\})$ (recall that $\mathbf{M}_k$ runs in time $t(n)$ on inputs of length less than or equal to n whenever $n \ge k$) and all numbers less than $\langle i, j\rangle$ that are ever canceled are canceled before stage s. Since $l_s > t(\max\{i, j\})$ then any changes in A or B at stage s do not alter the computations of $\mathbf{M}_i$ or $\mathbf{M}_j$ on inputs of length less than $\max\{i, j\}$. The conditions of (d) must hold for i, j, and s, or else we would have been able to force $\mathbf{M}_i(A) \ne \mathbf{M}_j(B)$ and would have canceled $\langle i, j\rangle$ at stage s. Finally, we use the strings $\overline{l_{2s+1} - 1}$ to diagonalize over the polynomial time computable sets. We now proceed with the construction of A, B, T, and $l_0, l_1, \cdots$ satisfying (a)–(d). Let U be a Turing machine that simulates the construction in the following way. On an input of length $n \ge l_1$, U (by simulating the Construction below) eventually writes down a string of the form $ec\bar{l}cx_1c \cdots cx_kc$ and halts, where: (1) l is the largest number less than or equal to n such that $l = l_s$ for some s and $s \equiv 1$ or $3 \pmod 4$; (2) if $l = l_s$ then $e = 0$ if $s \equiv 1 \pmod 4$ and $e = 1$ if $s \equiv 3 \pmod 4$; (3) if $e = 0$ then $x_1 < \cdots < x_k$ are the members of A of length less than l and if $e = 1$ then $x_1 < \cdots < x_k$ are the members of B of length less than l. If $n < l_1$ the U writes 0 on any input of length n. The computation of U on input x really only depends on the length of x. We may presume that if $|x| < |y|$ then U runs in less time on input x than it does on input y. Let u be the function such that on each input of length n, U makes exactly $u(n)$ moves. We may define the sequence $l_0, l_1, \cdots$ as follows:

$$l_0 = 0,$$

$$l_{s+1} = \begin{cases} t(l_s) + 1 & \text{if } s \text{ is even,} \\ u(l_s) & \text{if } s \text{ is odd.} \end{cases}$$

THE CONSTRUCTION

Stage s. [We define A and B at all x where $l_s \leq |x| < l_{s+1}$]. There are three cases to consider

Case 1. $s = 4k$. There are two steps

Step 1. Find the least uncanceled $d < k$, if any, such that there exist a string x with $\max\{i, j\} \leq |x| < l_s$ and a finite set $F \subseteq \{y : l_s \leq |y| < l_{s+1} - 1\}$ such that $\mathbf{M}_i(A \cup F)(x) \neq \mathbf{M}_j(B)(x)$ where $d = \langle i, j\rangle$. If no such d exists, go to step 2. Otherwise, choose x and F to be minimal, set $A = A \cup F$, and cancel d.

Step 2 If $1^l \notin P_k$ where $l = l_{s+1} - 1$ then set $A = A \cup \{1^l\}$.

Case 2. $s = 4k + 2$ Do exactly as in case 1, except reversing the roles of A and B.

Case 3. $s = 4k + 1$ or $4k + 3$. Do nothing.

End of the Construction

There is a seeming circularity to the simultaneous definition of $\{l_s\}$, A, and B in that A and B are defined in the Construction from $\{l_s\}$ and $\{l_s\}$ is defined from U which "is" the Construction. What we have done really is make implicit use of the recursion theorem.

We define T as follows: $T(x) = U(1^m)$ where m is the largest number such that $|x| \geq u(m)$ if there is such an m; otherwise $T(x) = 0$. By successively trying to compute $U(\lambda)$, $U(1)$, $U(1^2)$, $\cdots$, each within time bound $|x|$, we can certainly compute $T(x)$ in time bounded by $c|x|^2$ for some constant c.

The theorem is proved once we verify (a)–(d). The condition (a) is easily checked since we add nothing to A at stages $s \equiv 1, 2$, and $3 \pmod 4$ and nothing to B at stages $s \equiv 0, 1$, and $3 \pmod 4$. Condition (b) follows directly from the Construction. To check condition (c) we just examine the definition of T. Clearly $T(x) = 0$ for all x of length less than l_2 since $U(x) = 0$ for all x of length less than l_1 and $l_2 = u(l_1)$. If $l_{4s+2} \leq |x| < l_{4s+4}$ then $T(x) = U(\bar{l}_{4s+1})$ since $u(l_{4s+1}) = l_{4s+2}$ and $|x| < u(l_{4s+3})$, and $U(1^m) = U(\bar{l}_{4s+1})$ for all m such that $l_{4s+1} \leq m < l_{4s+3}$. Hence $T(x) = 0c\bar{l}_{4s+1}cx_1c \cdots cx_kc$ where $x_1 < \cdots < x_k$ are the members of A of length less than l_{4s+1}. If $l_{4s+4} \leq |x| < l_{4s+6}$ then we must have $T(x) = U(\bar{l}_{4s+3})$ and hence $T(x) = 1c\bar{l}_{4s+3}cx_1c \cdots cx_kc$ where $x_1 < \cdots < x_k$ are the members of B of length less than l_{4s+3}.

Finally we verify (d). Suppose $\mathbf{M}_i(A) = \mathbf{M}_j(B)$. To begin with, $\langle i, j\rangle$ is never canceled. If it were, say at stage s, then there is an x with $\max\{i, j\} \leq |x| < l_s$ such that $\mathbf{M}_i(A^s)(x) \neq \mathbf{M}_j(B^s)(x)$. Since $|x| > \max\{i, j\}$ then $\mathbf{M}_i$ and $\mathbf{M}_j$ run in time less than $t(|x|)$ on input x. Now, $t(|x|) \leq t(l_s) < l_{s+1}$. After stage s, strings added to A or B must be greater than or equal to l_{s+1} so that $\mathbf{M}_i(A^s)(x) = \mathbf{M}_i(A^{s'})(x)$ and $\mathbf{M}_j(B^s)(x) = \mathbf{M}_j(B^{s'})(x)$ for all $s' \geq s$. Whence $\mathbf{M}_i(A)(x) \neq \mathbf{M}_j(B)(x)$, which is a contradiction. Let s_0 be a stage such that $l_{s_0} > t(\max\{i, j\})$ and no number less than or equal to $\langle i, j\rangle$ is canceled at a stage greater than or equal to s_0. Suppose (d) fails at some stage $s \geq s_0$. We examine the case when $s \equiv 0 \pmod 4$. The other case is analogous. Let A' and x be such that $A'(y) = A(y)$ if $|y| < l_s$ and $\mathbf{M}_i(A')(x) \neq \mathbf{M}_i(A)(x)$ with $|x| < l_s$. If $|x| < \max\{i, j\}$ then by the definition of t, $\mathbf{M}_i$ runs in time less than $t(\max\{i, j\}) < l_{s_0} \leq l_s$ on input x. Since no word written on the oracle tape by $\mathbf{M}_i$ on input x can have length exceeding its run time, we must have $\mathbf{M}_i(A')(x) = \mathbf{M}_i(A)(x)$. Hence $|x| \geq \max\{i, j\}$. Let $F = \{y : l_s \leq |y| < l_{s+1} - 1$ and $y \in A'\}$. We must have $\mathbf{M}_i(A')(x) = \mathbf{M}_i(A^{s-1} \cup F)(x)$ since $\mathbf{M}_i$ runs in time less than $t(|x|) < l_{s+1} - 1$ on input x. Now $\mathbf{M}_j(B^{s-1})(x) = \mathbf{M}_j(B)(x)$ since $B(y) = 0$ for all y where $l_s \leq y < l_{s+1}$. Hence, $\mathbf{M}_i(A^{s-1} \cup F)(x) \neq \mathbf{M}_j(B^{s-1})(x)$ which means that $\langle i, j\rangle$ should be canceled at stage s, which is again a contradiction. □

5. *An Ascending Sequence with a Minimal Pair of Upper Bounds*

In this section we show that neither the polynomial m-degrees nor the polynomial T-degrees form a lattice. The method of proof is inspired by the initial segments arguments of Kleene and Post [8].

THEOREM 4. *There exist computable sets $C_0, C_1, \cdots, A$ and B such that (i) $C_i \leq_m^{\mathrm{P}} C_{i+1}$ for all i, (ii) $C_{i+1} \nleq_T^{\mathrm{P}} C_i$ for all i, (iii) $C_i \leq_m^{\mathrm{P}} A$ and $C_i \leq_m^{\mathrm{P}} B$ for all i, (iv) if $D \leq_T^{\mathrm{P}} A$ and $D \leq_T^{\mathrm{P}} B$ then $D \leq_m^{\mathrm{P}} C_i$ for some i.*

PROOF. If $E \subset \Sigma^*$ we define the ith *row* of E to be $R_i(E) = \{x : 1^i0x \in E\}$ and the

ith *section* of E to be $S_i(E) = \{y \in E \cdot y = 1^j0x$ for some x and $j < i\}$. It is quite easy to see that $R_i(E) \leq_m^{\mathbf{P}} S_{i+1}(E)$, $S_i(E) \leq_m^{\mathbf{P}} S_{i+1}(E)$, and $S_i(E) \leq_m^{\mathbf{P}} E$ for all i

We shall construct computable sets A and B with the following properties.

(a) for each i, $R_i(A)(x) = R_i(B)(x)$ for all but finitely many $x \in \Sigma^*$,

(b) $R_{2i+1}(A) \neq \mathbf{M}_j(S_{2i+1}(A))$ for all i and j,

(c) if $D = \mathbf{M}_i(A) = \mathbf{M}_j(B)$ then $D(x) = R_{2\langle i,j\rangle}(A)(x)$ for all but finitely many x.

If we set $C_i = S_{2i}(A)$ then $C_0, C_1, \cdots, A$ and B satisfy the theorem.

(i) Clearly $S_{2i}(A) \leq_m^{\mathbf{P}} S_{2i+2}(A)$.

(ii) $S_{2i+2}(A) \nleq_T^{\mathbf{P}} S_{2i}(A)$; otherwise $R_{2i+1}(A) \leq_T^{\mathbf{P}} S_{2i+1}(A)$, contrary to (b).

(iii) By (a), $S_{2i}(A) \leq_m^{\mathbf{P}} B$ as well as $\leq_m^{\mathbf{P}} A$.

(iv) Condition (c) guarantees that if $D \leq_T^{\mathbf{P}} A$ and $D \leq_T^{\mathbf{P}} B$ then $D \leq_m^{\mathbf{P}} S_{2i}(A)$ for some i.

We construct A and B in stages. Let A^s and B^s be the values of A and B at the end of stage s. Each member of A or B will have the form 1^i0x where $i \geq 0$ and $x \in \{0, 1\}^*$. Let $\langle x, y, z\rangle = \langle\langle x, y\rangle, z\rangle$.

If $s = \langle e, f, g\rangle$ then stage $s + 1$ of the construction is primarily devoted to diagonalizing to make $R_{2e+1}(A) \neq \mathbf{M}_f(S_{2e+1}(A))$. This diagonalization will succeed unless there is some $d \leq e$ such that we can force $\mathbf{M}_i(A) \neq \mathbf{M}_j(B)$ where $d = \langle i, j\rangle$. The parameter g allows us infinitely many attempts at diagonalizing to make $R_{2e+1}(A) \neq \mathbf{M}_f(S_{2e+1}(A))$. At least one of these attempts is successful because for each pair (i, j) there is at most one stage at which we forsake diagonalizing to force $\mathbf{M}_i(A) \neq \mathbf{M}_j(B)$. Should we force $\mathbf{M}_i(A) \neq \mathbf{M}_j(B)$ at stage $s + 1$ then the number $\langle i, j\rangle$ is canceled at stage $s + 1$.

From stage to stage we maintain a list $l_0, l_1, \cdots$. Let l_k^s be the value of l_k at the end of stage s. We shall have $l_k^s \leq l_k^{s+1}$ and $\lim_s l_k^s = \infty$. The value of l_k marks the amount of the kth row of A and B that is currently determined, that is, if $|x| \leq l_k^s$ then $R_k(A^t)(x) = R_k(A^s)(x)$ and $R_k(B^t)(x) = R_k(B^s)(x)$ for all $t \geq s$.

Let $s = \langle e, f, g\rangle$ and $d = \langle i, j\rangle$. If $d \leq e$ and no number less than or equal to d is canceled at stage $s + 1$ then we can "embed" a piece of $\mathbf{M}_i(A)$ into the $2d$th row of A and B by putting $1^{2d}0x$ into A and B at stage $s + 1$ provided $x \in \mathbf{M}_i(A)$ and $l_{2d}^s < |x| \leq l_{2d}^{s+1}$. If $d > e$ then $l_{2d}^s = l_{2d}^{s+1}$ unless some number less than or equal to e is canceled at stage $s + 1$. Since each number is canceled at most once we conclude that if we never cancel the number d then $\mathbf{M}_i(A)$ is embedded into the $2d$th row of both A and B, that is, $R_{2d}(A)(x) = R_{2d}(B)(x) = \mathbf{M}_i(A)(x)$ for all but finitely many x. (The exceptions would occur at stages when numbers less than d were canceled.) The construction works because, if we cannot extend A in such a way at stage $s + 1$ as to force $\mathbf{M}_i(A)(x) \neq \mathbf{M}_j(B)(x)$ for some x of length less than or equal to l_{2d}^{s+1}, then the value of $\mathbf{M}_i(A)(x)$ for x of length less than or equal to l_{2d}^{s+1} does not really depend on what is added to A or B at stage $s + 1$; in particular, we can safely add to A and B all $1^{2d}0x$ such that $x \in \mathbf{M}_i(A)$ and $l_{2d}^s < |x| \leq l_{2d}^{s+1}$.

Let t be an increasing recursive function such that for all i, $\mathbf{M}_i$ runs in time bounded by $t(n)$ on all inputs of length $n \geq i$.

THE CONSTRUCTION

Stage 0. Set $A = B = \emptyset$ and $l_i = 0$ for all i.

Stage $s + 1$. The number s equals $\langle e, f, g\rangle$ for some e, f, and g. Set $m_{2e+1} = l_{2e+1} + 1$ and $m_k = l_k + t(m_{k+1})$ for $0 \leq k < 2e + 1$. Set $d = 0$.

1. [Can d be canceled?] If $d > e$ go to 3. If d is uncanceled and there exist an x with $d \leq |x| \leq m_{2d}$ and a finite set $F \subseteq \{y : y$ has the form 1^k0z where $k \leq t(m_{2d})$ and $l_k < |z| \leq t(m_{2d})\}$ such that $\mathbf{M}_i(A \cup F)(x) \neq \mathbf{M}_j(B)(x)$ where $d = \langle i, j\rangle$, then go to 2 If d is already canceled or there is no such x and F then let $G = \{1^{2d}0y : l_{2d} < |y| \leq m_{2d}$ and $y \in \mathbf{M}_i(A)\}$. Set $A = A \cup G$, $B = B \cup G$, $l_{2d} = m_{2d}$, $l_{2d+1} = m_{2d+1}$, and $d = d + 1$ Return to 1

2. [d is canceled] In this case select x and F satisfying the conditions in 1. Set $A = A \cup F$ and $l_k = l_k + t(m_{2d})$ for $2d \leq k \leq t(m_{2d})$. Cancel d and go to the next stage

3. [Diagonalization] Let $m = m_{2e+1}$. If $1^m \notin \mathbf{M}_f(S_{2e+1}(A))$ then set $A = A \cup \{1^{2e+1}01^m\}$ and $B = B \cup \{1^{2e+1}01^m\}$. Go to the next stage

End of the Construction

Each construction stage ends in a finite amount of time because at most e iterations of 1 can occur at stage $\langle e, f, g\rangle + 1$. We must have $\lim_s l_k^s = \infty$ for each k because $l_k^{s+1} > l_k^s$ whenever $s = \langle k, f, g\rangle$ for some f and g and no number less than or equal to k is canceled at stage $s + 1$. The sets A and B are recursive, for to determine if 1^i0x is in A or B we wait for the least stage s such that $|x| \leq l_i^s$. Now $1^i0x \in A$ if and only if $1^i0x \in A^s$, and similarly for B.

The theorem is proved if we can verify (a)–(c) mentioned above.

Claim (a) For each i, $R_i(A)(x) = R_i(B)(x)$ for all but finitely many x.

We always add the same strings to A and B except in step 2 of the construction. In step 2 of stage $s + 1$ a string of the form 1^i0x is put into A only if $l_i < |x| \leq t(m_{2d})$ where d is canceled at stage $s + 1$. In the d iterations of step 1 before proceeding to step 2, the value of l_j for $j < 2d$ was increased to m_j. By the definition of m_j we must have $m_j \geq t(m_{2d})$ for all $j < 2d$. Hence 1^i0z enters A only if $i \geq 2d$ (since $l_j \geq t(m_{2d})$ for $j < 2d$). We conclude that $R_i(A)(x) = R_i(B)(x)$ for all x of length $> l_i^t$, where t is the least stage such that no number less than or equal to $i/2$ is canceled at a stage greater than or equal to t.

Claim (b) $R_{2i+1}(A) \neq \mathbf{M}_j(S_{2i+1}(A))$ for all i and j.

Let s be a number of the form $\langle i, j, k\rangle$ such that no number less than or equal to i is canceled after stage s and $l_{2i+1}^s > j$. At stage $s + 1$ we execute step 3 since no number less than or equal to i is canceled at stage $s + 1$. Let m be as in step 3. We certainly have $R_{2i+1}(A^{s+1})(1^m) \neq \mathbf{M}_j(S_{2i+1}(A^{s+1}))(1^m)$. Now, $m \leq l_{2i+1}^{s+1}$ so that $R_{2i+1}(A^{s'})(1^m) = R_{2i+1}(A^{s+1})(1^m)$ for all $s' \geq s + 1$. By cycling through step 1 i times we succeed in making $l_h^{s+1} \geq t(m)$ for all $h < 2i + 1$. Since $m > i$ then $\mathbf{M}_j$ runs in time bounded by $t(m)$ on input 1^m. Hence no question of the oracle can have length exceeding $t(m)$. Thus the computation on the right hand is fixed. That is,

$$\mathbf{M}_j(S_{2i+1}(A^{s+1}))(1^m) = \mathbf{M}_j(S_{2i+1}(A^{s'}))(1^m) \quad \text{for all} \quad s' \geq s + 1.$$

Claim (c) If $D = \mathbf{M}_i(A) = \mathbf{M}_j(B)$ then $D(x) = R_{2\langle i,j\rangle}(A)(x)$ for all but finitely many x.

Let $d = \langle i, j\rangle$ and let u be a stage such that no number less than or equal to d is canceled at a stage greater than or equal to u. Should d ever be canceled, say at stage $s + 1$, then we can show that $\mathbf{M}_i(A) \neq \mathbf{M}_j(B)$. We certainly have $\mathbf{M}_i(A^{s+1})(x) \neq \mathbf{M}_j(B^{s+1})(x)$ for the x selected in step 2. We have $l_k^{s+1} \geq t(|x|)$ for all $k \leq t(|x|)$ since l_k is already that large for $k < 2d$ and l_k is set that large for k with $2d \leq k \leq t(|x|)$. Now, $|x| \geq d \geq \max\{i, j\}$ so that both $\mathbf{M}_i$ and $\mathbf{M}_j$ run in time bounded by $t(|x|)$ on input x. Since no question of $\mathbf{M}_i$ or $\mathbf{M}_j$ can have length exceeding $t(|x|)$ on input x then we must have $\mathbf{M}_i(A^{s'})(x) \neq \mathbf{M}_j(B^{s'})(x)$ for all $s' \geq s + 1$.

Assume then that d is never canceled and that $D = \mathbf{M}_i(A) = \mathbf{M}_j(B)$. We show that $D(x) = R_{2d}(A)(x)$ for all x of length greater than l_{2d}^u. Let s be the unique stage greater than or equal to u such that $l_{2d}^s < |x| \leq l_{2d}^{s+1}$. At stage $s + 1$, step 1 is iterated at least $d + 1$ times. Let A' and A'' be the values of A at the beginning and at the end of the $(d + 1)$-st iteration respectively. Adding the set G to A guarantees that $A''(1^{2d}0x) = \mathbf{M}_i(A')(x)$. The left side of this equation equals $R_{2d}(A)(x)$ since $|x| \leq l_{2d}^{s+1}$. It suffices to show that $\mathbf{M}_i(A')(x) = \mathbf{M}_i(A)(x)$. Suppose not. Let $l_0', l_1', \cdots$ be the values of $l_0, l_1, \cdots$, respectively, at the beginning of the $(d + 1)$-st iteration of step 1. Since $A(1^k0z)$ is determined for z of length less than or equal to l_k' and $\{1\}^* \cap A = \emptyset$ then there exists a finite set $F \subseteq \{y . \,|\, |y| \leq t(|x|)$ and y has the form 1^k0z with $|z| > l_k'\}$ such that $\mathbf{M}_i(A' \cup F)(x) \neq \mathbf{M}_i(A')(x)$. Let B' be the value of B at the beginning of the $(d + 1)$-st iteration of step 1. Either $\mathbf{M}_i(A' \cup F)(x) \neq \mathbf{M}_j(B')(x)$ or $\mathbf{M}_i(A')(x) \neq \mathbf{M}_j(B')(x)$ so that x and either F or $\emptyset$ satisfy the condition of step 1, which forces d to be canceled at stage $s + 1$. □

The theorem directly implies·

Corollary 4.1. *Neither the polynomial m- nor T-degrees of computable sets are a lattice.*

PROOF. The m- or T-degrees of A and B do not have a greatest lower bound in their respective orderings. □

6. *Generalizations*

The results of the previous sections can be generalized to a wide variety of subrecursive reducibilities. We shall restrict ourselves to subrecursive reducibilities definable by space and time bounded Turing machines.

Let us first consider *time definable reducibilities.* Let $\mathbf{C}$ be any class of total functions from the natural numbers into itself. We say that A is *Turing reducible to B in* $\mathbf{C}$*-time* ($A \leq_T^{\mathbf{C}^t} B$) if there is an oracle Turing machine $\mathbf{M}$ and a function $f \in \mathbf{C}$ such that $\mathbf{M}$ runs in time f and $x \in A$ if and only if $\mathbf{M}$ accepts x when presented with B as its oracle. A is *many-one reducible to B in* $\mathbf{C}$*-time* ($A \leq_m^{\mathbf{C}^t} B$) if there is a Turing machine transducer T and a function $f \in \mathbf{C}$ such that T runs in time f and $x \in A$ if and only if $T(x) \in B$. Turing machine acceptors and transducers and oracle Turing machines may have any finite number of tapes and tape symbols.

To obtain a reasonable notion of time definable reducibility there are several restrictions we should put on $\mathbf{C}$. To begin with if $\leq_m^{\mathbf{C}^t}$ and $\leq_T^{\mathbf{C}^t}$ are to be reflexive relations then there should be a function $f \in \mathbf{C}$ such that $f(n) \geq n$ so that the oracle Turing machine can at least copy the input onto the oracle tape. For $\leq_m^{\mathbf{C}^t}$ and $\leq_T^{\mathbf{C}^t}$ to be transitive $\mathbf{C}$ should have the property that if f and $g \in \mathbf{C}$ then there exist $h \in \mathbf{C}$ such that

$$h(n) \geq f(n)[1 + 2 \max \{g(m) . m \leq f(n)\}].$$

If $A \leq_T^{\mathbf{C}^t} B$ via an $f(n)$-time bounded oracle Turing machine $\mathbf{M}$ and $B \leq_T^{\mathbf{C}^t} C$ via a $g(n)$-time bounded oracle Turing machine $\mathbf{N}$ then $A \leq_T^{\mathbf{C}^t} C$ via the $f(n)[1 + 2 \max \{g(m) : m \leq f(n)\}]$-time bounded oracle Turing machine $\mathbf{K}$. The machine $\mathbf{K}$ simulates $\mathbf{M}$ until $\mathbf{M}$ would enter state Q, then it simulates $\mathbf{N}$ or the contents of $\mathbf{M}$'s oracle tape and when $\mathbf{N}$ would halt $\mathbf{K}$ returns to the simulation of $\mathbf{M}$ knowing which of YES or NO it should enter. While $\mathbf{K}$ is simulating $\mathbf{N}$ on the contents of $\mathbf{M}$'s oracle tape should $\mathbf{N}$ enter state Q then so does $\mathbf{K}$ and $\mathbf{K}$'s oracle tape is actually $\mathbf{N}$'s. To avoid confusion, after simulating $\mathbf{N}$ and getting an answer, $\mathbf{K}$ should erase any trace of this side computation. On an input of length n, $\mathbf{M}$ cannot ask a question of the oracle of length greater than $f(n)$ so that $\mathbf{K}$ cannot make more than $\max \{g(m) : m \leq f(n)\}$ moves in a single simulation of $\mathbf{N}$ on $\mathbf{M}$'s oracle tape. The erasing of such a side computation requires no more than $\max \{g(m) : m \leq f(n)\}$ moves. Finally no more than $f(n)$ simulations of $\mathbf{N}$ on $\mathbf{M}$'s oracle tape are made by $\mathbf{K}$. Hence $\mathbf{K}$ runs in time bounded by $f(n) + 2f(n) \max \{g(m) : m \leq f(n)\}$. These restrictions on $\mathbf{C}$ which allow $\leq_T^{\mathbf{C}^t}$ to be reflexive and transitive force $\mathbf{C}$ to have the property that for every polynomial function p there is an $f \in \mathbf{C}$ such that $f(n) \geq p(n)$ for all n. It would seem then that any reasonable notion of time definable reducibility must be at least as powerful as polynomial time reducibility. If we are only interested in many-one reducibility then the restrictions on $\mathbf{C}$ to make $\leq_m^{\mathbf{C}^t}$ reflexive and transitive can be lessened. If $\mathbf{C}$ equals the set of linear functions then $\leq_m^{\mathbf{C}^t}$ is certainly transitive.

Let $\mathbf{C}^t$ be the $\mathbf{C}$*-time computable sets*, that is, the class of sets computable in time f for some $f \in \mathbf{C}$. A function q is *time measurable* if there is a multitape Turing machine T such that on each input of length n, T halts in exactly $q(n)$ moves. One thing that seemed to help make our earlier constructions work was that for each k we could "attach" a $(k + n^k)$-counter to each Turing machine in order to make effective enumerations of the polynomial time bounded Turing machine recognizers, Turing machine transducers, and oracle Turing machine recognizers. If $\mathbf{C}$ has the property that $\mathbf{C}$ is cofinal with a recursively enumerable class $\{q_i\}$ of time measurable functions (that is, there is an effective enumeration $q_0, q_1, \cdots$ of time measurable functions such that for all $f \in \mathbf{C}$ there is an i such that $q_i(n) \geq f(n)$ for all n and for all i there is an $f \in \mathbf{C}$ such that $f(n) \geq q_i(n)$), then we see for each i we could "attach" a $q_i(n)$-counter to each appropriate kind of Turing machine in order to obtain effective enumerations of the $\mathbf{C}$-time bounded recognizers, transducers, and oracle machines. If **C** contains enough functions to force $\leq_m^{\mathbf{C}^t}$ and $\leq_T^{\mathbf{C}^t}$ to be reflexive and transitive and there is a recursively enumerable sequence $\{q_i\}$ of time measurable functions cofinal with **C** then Theorems 1–4 hold where **P** is replaced by $\mathbf{C}^t$. The proofs of these "new" theorems would be essentially the same as already given for **P**. If we modify the proofs of Theorem 1–4 sufficiently we can remove the requirement that **C** be cofinal with a recursively enumerable class of time measurable functions and simply require **C** to be cofinal with a recursively enumerable class of total recursive functions.

Define **C** to be a *time class* if **C** is cofinal with a recursively enumerable set of total recursive functions, there is some $f \in \mathbf{C}$ such that $f(n) \geq n$ for all n, and for all f and g in **C** there is an $h \in \mathbf{C}$ such that $h(n) \geq f(n)[1 + 2 \max\{g(m) : m \leq f(n)\}]$ for all n.

THEOREM 5. *If* **C** *is a time class then* $\leq_m^{\mathbf{C}^t}$ *and* $\leq_T^{\mathbf{C}^t}$ *are reflexive and transitive relations and Theorems* 1–4 *hold with* **P** *replaced by* $\mathbf{C}^t$.

PROOF. We shall examine the modifications needed to obtain Theorem 1. Similar modifications can be made to get the other three theorems. Let $g_0, g_1, \cdots$ be an effective enumeration of total recursive functions cofinal with **C**. Let $R_0, R_1, \cdots$ be an effective enumeration of all Turing machine recognizers and let $\mathbf{R}_0, \mathbf{R}_1$ be an effective enumeration of all oracle Turing machines.

Just as in the original proof we construct a polynomial time bounded transducer T and $A = \{x : x \in B \text{ and } |T(x)| \text{ is even}\}$. Since **C** is a time class then **C**-time reducibility includes polynomial time reducibility. Hence the reduction procedure that witnesses $A \leq_m^{\mathbf{P}} B$ also witnesses $A \leq_m^{\mathbf{C}^t} B$. In this case T is "trying to satisfy" the following conditions for each pair (i, j):

(a) $A \neq R_i$ or on some input x, R_i runs in time greater than $g_j(|x|)$,

(b) $B \neq \mathbf{R}_i(A)$ or on some input x and with oracle X, $\mathbf{R}_i$ runs in time greater than $g_j(|x|)$.

If $T(x) = 1^k$ where $k = 2\langle i, j\rangle$ then T is trying to satisfy condition (a) for the pair (i, j) by trying to find in $|x|$ or fewer moves the least z in the natural ordering of Σ^* such that $A(z) \neq R_i(z)$ or R_i runs in time greater than $g_j(|z|)$ on input z. If $T(x) = 1^k$ where $k = 2\langle i, j\rangle + 1$, then T is trying to satisfy condition (b) for the pair (i, j) by trying to find in $|x|$ or fewer moves the least z in the natural ordering of Σ^* such that $B(z) \neq \mathbf{R}_i(A)(z)$ or there is a finite set $F \subseteq \{y : |y| \leq g_j(|z|)\}$ such that $\mathbf{R}_i$ runs in time greater than $g_j(|z|)$ on input z and with oracle F. From this intuitive description of what T is trying to do we could piece together an actual definition of the action of T so that $A = \{x : x \in B \text{ and } |T(x)| \text{ is even}\}$ satisfies (a) and (b) for all i and j.

Now, $A \notin \mathbf{C}^t$ since otherwise for some i and j, $A = R_i$ and R_i runs in time bounded by g_j, which implies condition (a) is not satisfied for i and j. Further $B \not\leq_T^{\mathbf{C}^t} A$; otherwise for some i and j, $B = \mathbf{R}_i(A)$ and R_i runs in time bounded by g_j, which implies condition (b) fails for i and j.

We leave it to the reader to supply modified proofs of Theorems 2–4. □

We now consider *space definable reducibilities*. To allow the possibility of space bounds less than linear we modify our Turing machines as follows. All Turing machines have a read only input tape and one work tape. Recognizers need no additional tapes. Transducers have a write only output tape. Oracle Turing machines have a write only oracle tape. Output tapes and oracle tapes do not move right. We adopt the convention that the oracle tape is erased after each question of the oracle. When we speak of a machine that runs in $q(n)$ space then we mean that on any input of length n the machine halts without scanning more than $q(n)$ different tape cells on the *work* tape. More than $q(n)$ different tape cells can be scanned on the input tape, output tape, or oracle tape.

Let **C** be a class of total functions. The class $\mathbf{C}^s$ is the class of sets computable in space bounded by a function in **C**. We say that *A is Turing reducible to B in* **C**-*space* ($A \leq_T^{\mathbf{C}^s} B$) if there is an oracle Turing machine **M** and a function $f \in \mathbf{C}$ such that **M** runs in $f(n)$-space and $x \in A$ if and only if **M** accepts x when presented with the oracle B. Further, A

is *many-one reducible to* B in **C**-*space* ($A \leq_T^{\mathbf{C}^s} B$) if there is a Turing machine transducer T and a function $f \in \mathbf{C}$ such that T runs in $f(n)$-space and $x \in A$ if and only if $T(x) \in B$.

In order to make $\leq_m^{\mathbf{C}^s}$ and $\leq_T^{\mathbf{C}^s}$ reflexive and transitive there are some restrictions we should make on **C**. To make each reflexive **C** should contain a function f such that $f(n) \geq 1$ for all n. Before specifying any restrictions on **C** let us imagine how we might reduce A to C in **C**-space if we already have $A \leq_T^{\mathbf{C}^s} B$ and $B \leq_T^{\mathbf{C}^s} C$. Let f and $g \in \mathbf{C}$ and let **M** be an $f(n)$-space bounded oracle machine and **N** be a $g(n)$-space bounded oracle machine such that **M** reduces A to B and **N** reduces B to C. The obvious reduction procedure reducing A to C is max $(\{cnf(n)c^{f(n)}\} \cup \{g(m) : m \leq cnf(n)c^{f(n)}\})$ space bounded for some c. The procedure would simulate **M** using a track of its work tape to simulate **M**'s oracle tape. Should **M** enter state Q then the machine begins simulating **N** on the contents of the track which simulates **M**'s oracle tape returning to the simulation of **M** when **N** halts. Unfortunately there is a constant c (dependent on **M**) such that a word of length $cnf(n)c^{f(n)}$ could be written by **M** on its oracle tape. Since **M**'s oracle tape is one-way read only there is no real need to write down all of **M**'s oracle tape onto the work tape. All logs are base 2.

LEMMA 6. *If* **M** *is* $f(n)$*-space bounded and reduces* A *to* B *and* **N** *is* $g(n)$*-space bounded and reduces* B *to* C *then there is an oracle Turing machine* **K** *reducing* A *to* C *that is max* $(\{f(n), \log n\} \cup \{g(m) : m \leq cnf(n)c^{f(n)}\})$*-space bounded for some constant* c.

PROOF. This proof generalizes a proof of Stockmeyer and Meyer [24, Lem. 2.1].

The machine **K** has two modes: mode 1 where **K** is directly simulating **M** and mode 2 where **K** is simulating **N**. When **K** is in mode 2 it may need to do an aside simulation of **M** as well. The machine **K** has 8 tracks.

Track 1 holds the current contents of **M**'s work tape.

Track 2 holds the binary count of the current read head position of **M**.

Track 3 holds the contents of **M**'s work tape as it was immediately after the last time **M** entered state Q.

Track 4 holds a binary count of **M**'s input head position as it was after the last time **M** entered state Q.

Track 5 holds **M**'s oracle head position when **K** is in mode 2.

Track 6 holds the contents of **N**'s work tape when **K** is in mode 2.

Track 7 holds an aside work space.

Track 8 holds an aside count.

In finite control **K** remembers whether **M** entered state YES or NO after the last time it entered state Q.

On input x, **K** operates as follows:

Begin in mode 1.

Mode 1. Simulate **M** directly updating tracks 1 and 2 but writing nothing on what would be **M**'s oracle tape. Should **M** enter state Q, **K** enters mode 2. **K** accepts if and only if **M** does.

Mode 2. Simulate **N** by updating tracks 5 and 6 and using **K**'s oracle tape to simulate **N**'s. Since **M**'s oracle tape is unavailable for scanning we must do a side computation to discover what the ith letter of **M**'s oracle tape is where i is the binary count on track 5. This is done by copying track 3 onto track 7, resetting the input head of **K** to the count on track 4, and simulating **M** until it has outputed i symbols (which would have normally been written on **M**'s oracle tape). Track 8 is used to keep count of how many symbols have been outputed so far. Once the ith output symbol is known then tracks 5 and 6 can be accurately updated. Should this simulation of **N** halt then **K** returns to mode 1, updating tracks 3 and 4 and remembering in finite control whether **M** entered state YES or state NO.

On an input of length n tracks 1, 3, and 7 are $f(n)$-space bounded while tracks 2 and 4 are log n-space bounded. There is a constant c such that no word longer than $cnf(n)c^{f(n)}$ can be written on **M**'s oracle tape so that track 6 is max $\{g(m) : m \leq cnf(n)c^{f(n)}\}$-space bounded and tracks 5 and 8 are log $(cnf(n)c^{f(n)})$ space bounded. Hence the bound of **K** holds. It can be easily verified that **K** reduces A to C. □

An important application of this lemma is in log space reducibility. Define $\leq_T^{\log}$ to be $\leq_T^{\mathbf{C}^s}$ where **C** contains the single function $\log n$. Hence $\leq_T^{\log}$ is the relation of Turing reducibility in $\log n$ space. Clearly $\leq_T^{\log}$ is reflexive and Lemma 6 shows that it is transitive. By the lemma if $A \leq_T^{\log} B$ and $B \leq_T^{\log} C$ then A is Turing reducible to C in space bounded by $\max(\{\log n\} \cup \{\log m : m \leq cn \log nc^{\log n}\})$ for some constant c. But this amounts to a $\log n$ space bound.

In general if **C** is a class of total functions and $\leq_T^{\mathbf{C}^s}$ is to be reflexive and transitive then **C** should contain a function f such that $f(n) \geq \log n$ for all n and if f and g are in **C** then for each constant c there is a function $h \in \mathbf{C}$ and constant d such that $dh(n) \geq \max(\{f(n)\} \cup \{g(m)\} : m \leq cnf(n)c^{f(n)}\})$ for all n. We allow the constant d because any work tape can be compressed by a constant factor.

Define **C** to be a *space class* if **C** is cofinal with a recursively enumerable set of total recursive function, **C** contains a function f with $f(n) \geq \log n$, and if $f, g \in \mathbf{C}$ and c is a constant then there exists $h \in \mathbf{C}$ and a constant d such that $dh(n) \geq \max(\{f(n)\} \cup \{g(m) . m \leq cnf(n)c^{f(n)}\})$ for all n.

THEOREM 7. *If* **C** *is a space class then* $\leq_m^{\mathbf{C}^s}$ *and* $\leq_T^{\mathbf{C}^s}$ *are reflexive and transitive relations and Theorems* 1–4 *hold with* **P** *replaced by* $\mathbf{C}^s$.

PROOF. We leave the proof to the reader. We only note that the function T defined in each of the proofs of Theorems 1, 2, and 3 will be $\log n$-space bounded rather than polynomial time bounded.

Other space and time definable reducibility notions are possible. Stockmeyer and Meyer [24] define an interesting notion of reducibility. They define $A \leq_{\log\text{-lin}} B$ if there is a $\log n$-space bounded transducer T and constant c such that $x \in A$ iff $T(x) \in B$ and $|T(x)| \leq c|x|$. Their notion could be generalized to Turing reducibility by requiring log space bound on the work tape and linear space bound on the oracle tape. In general there seem to be three basic complexity parameters in reduction procedures: (i) time of the procedure, (ii) storage space used by the procedure, and (iii) space used by the procedure in writing its questions. It would be interesting to make a comparison of different notions of reducibility obtained by varying the bounds on (i)–(iii).

It appears to us that our Theorems 1–4 are true of almost any reasonable notion of space and time definable reducibility obtained by varying the bounds on (i)–(iii).

There is also the possibility of abstractly defining notions of subrecursive reducibility. For instance, we could define primitive recursive reducibility by A is *primitive recursive* in B if the characteristic function of A is in the smallest class of functions containing the constant zero function, the successor function, the projection functions, and the characteristic function of B, and closed under composition and recursion. (We are identifying Σ^* with numbers written in diatic notation, that is, λ represents 0 and $a_1a_2\cdots a_n \in \Sigma^*$ represents $2^n - 1 + \sum a_i 2^{i-1}$). This is really a Turing reducibility. Define primitive recursive many-one reducibility by A is *many-one primitive recursive in* B if there is a primitive recursive function f such that $x \in A$ if and only if $f(x) \in B$.

The work of Ritchie [20] and others in showing that many abstractly definable classes of recursive functions are just Turing machine complexity classes can be extended to show that many interesting abstractly definable subrecursive reducibility notions are just space and/or time definable reducibilities. From the schemata for defining Grzegorczyk classes $\xi^n (n \geq 0)$ [5] we could define abstractly, as we did for primitive recursive reducibility, ξ^n-reducibility (both Turing and many-one). Both ξ^0 and ξ^1 seem somewhat pathological. The notion of ξ^2-reducibility is equivalent to reducibility with a linear space bound both on the work tape and oracle tape (output tape in the case of many-one reducibility). The notion of ξ^n-reducibility for $n \geq 3$ is equivalent to reducibility in ξ^n-space. Likewise the notion of primitive recursive reducibility is equivalent to reducibility in primitive recursive space. By our observations above Theorems 1–4 (interpreted correctly) hold for ξ^n-reducibility for ($n \geq 2$) and primitive recursive reducibility.

ACKNOWLEDGMENTS. We would like to thank A. H. Lachlan, M. Machtey, and R. W. Ritchie for several enlightening conversations concerning this research. We also appreciate the careful reading of the manuscript by Paul Frank.

REFERENCES

1 Axt, P On a subrecursive hierarchy and primitive recursive degrees *Trans. AMS 92* (1959), 85–105
2. Borodin, A, Constable, R L, and Hopcroft, J E Dense and nondense families of complexity classes IEEE Conf Record Tenth Annual Symp on Switching and Automata Theory, 1969, pp. 7–19
3 Cook, S A. The complexity of theorem proving procedures. Proc Third Annual ACM Symp on Theory of Computing, 1971, pp 151–158
4. Friedberg, R M Two recursively enumerable sets of incomparable degrees of unsolvability. *Proc Nat Acad Sci U S A. 43* (1957), 236–238
5 Grzegorczyk, A. *Some Classes of Recursive Functions* Rozprawy Matematyczhe, Warsaw, 1953
6 Karp, R M Reducibility among combinatorial problems In *Complexity of Computer Computations*, R. E Miller and J. W Thatcher, Eds., Plenum, New York, 1972, pp. 85–103.
7 Karp, R M Private communication.
8 Kleene, S C, and Post, E L. The upper semi-lattice of degrees of unsolvability *Ann. Math.*, Ser. 2, *59* (1954), 379–407
9. Ladner, R E, Lynch, N A, and Selman, A. Comparison of polynomial-time reducibilities. Proc Sixth Annual ACM Symp on Theory of Computing, 1974, pp 110–121
10 Lynch, N A. Relativization of the theory of computational complexity Tech Rep. TR-99, Project MAC, Ph D. Th, M I.T, Cambridge, Mass., 1972.
11 Machtey, M. Classification of computable functions by primitive recursive classes Proc Third Annual ACM Symp on Theory of Computing, 1971, pp. 251–257
12 Machtey, M Augmented loop languages and classes of computable functions *J Comput. Syst Sci. 6*, 6 (1972), 603–624
13 Machtey, M The honest subrecursive classes are a lattice. *Inform and Contr. 24*, 3 (1974), 247–263.
14. Machtey, M On the density of honest subrecursive classes. Tech Rep. CSD TR 92, Comput. Sci. Dep, Purdue U., Lafayette, Ind, 1973
15 Meyer, A R, and Ritchie, D M A classification of the recursive functions *Z Math Logik Grundlagen Math 18* (1972), 71–82
16 Meyer, A R Private communication
17 Muchnik, A A On the unsolvability of the problem of reducibility in the theory of algorithms. *Doklady Akademii Nauk SSSR*, n s, *108* (1956), 194–197
18. Post, E. L Recursively enumerable sets of positive integers and their decision problems. *Bull. AMS 50* (1944), 284–316
19 Pratt, V Every prime has a succinct certificate (In preparation)
20 Ritchie, R W Classes of predictably computable functions *Trans AMS 106* (1963), 139–173.
21 Rogers Jr, H *Theory of Recursive Functions and Effective Computability* McGraw-Hill, New York, 1967
22 Sethi, R Complete register allocation problems Proc Fifth Annual ACM Symp on Theory of Computing, 1973, pp 183–195
23 Stockmeyer, L J Planar 3-colorability is polynomial complete. *SIGACT News 5*, 3 (1973), 19–25
24 Stockmeyer, L. J, and Meyer, A R Word problems requiring exponential time preliminary report Proc Fifth Annual ACM Symp on Theory of Computing, 1973, pp 1–9
25 Ullman, J D Polynomial complete scheduling problems *ACM Operating Syst Rev 7*, 4 (1973), Fourth Symposium on Operating System Principles, 96–101

RECEIVED JANUARY 1974, REVISED JULY 1974