ON THE COMPUTATIONAL COMPLEXITY OF SMALL DESCRIPTIONS*

RICARD GAVALDÀ† AND OSAMU WATANABE‡

Abstract. For a set L that is polynomial-time reducible (in short, $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible) to some sparse set, we investigate the computational complexity of such sparse sets relative to L. We prove the first lower bounds on the complexity of recognizing such sets. We construct sets A and B such that both of them are $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to *some* sparse set, but A (resp., B) is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to *no* sparse set in P^A (resp., $\mathrm{NP}^B \cap \text{co-}\mathrm{NP}^B$); that is, the complexity of sparse sets to which A (resp., B) is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible is more than P^A (resp., $\mathrm{NP}^B \cap \text{co-}\mathrm{NP}^B$). From these results or application of our proof technique we obtain: 1) lower bounds for the relative complexity of generating polynomial-size circuits for some sets in P/poly, and 2) separations of the classes of sets equivalent or reducible to sparse sets under various polynomial-time reducibilities.

Key words. computational complexity, sparse sets, tally sets, polynomial-time reducibility, polynomial-time equivalence, Kolmogorov complexity

1. Introduction. In computational complexity theory or, more in general, in the theory of computation, sets with a small description have often been the subject of investigation. In this paper we study the computational complexity of (recognizing or generating) such small descriptions relative to an original set. More specifically, for a set L that is polynomial-time reducible to some sparse set, we investigate the complexity of such sparse sets relative to L.

The relative complexity of small descriptions is important in several contexts. Let us take for example "concept learning" as studied in [4, 5]. Roughly speaking, the goal of concept learning is to obtain a (small) description of an unknown set X. In particular, "query learning," the type of learning discussed in [4, 5], achieves this task by asking queries on X to a teacher; thus, one can formalize query learning as a type of oracle computation, where an unknown set is used as an oracle [24]. Therefore, the relative complexity of finding a description of an unknown set is a key factor in order to discuss whether a given concept class is, say, polynomial-time learnable.

Now we explain how the problem is formalized in this paper. $\mathrm{R_T}(\mathrm{SPARSE})$[1] is the class of sets that are polynomial-time reducible (in short, $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible) to some sparse set. Recall that a set S is *sparse* if for some polynomial p and for all $n \geq 0$,

* This research was performed while the second author was visiting Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, and was supported in part by ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM). The paper was prepared in part while the first author was visiting the University of California at Santa Barbara, supported by the National Science Foundation under grant CCR-89-13584. The second author was also supported in part by Grant in Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan under Grant-in-Aid for Co-operative Research (A) 02302047 (1990).

† Department of Software (L.S.I.), Universitat Politècnica de Catalunya. Pau Gargallo 5, 08028 Barcelona, SPAIN. gavalda@lsi.upc.es

‡ Department of Computer Science, Tokyo Institute of Technology. Meguro-ku, Ookayama 1-12-1, Tokyo 152, JAPAN. watanabe@cs.titech.ac.jp

[1] This class was denoted as $\mathrm{P_T}(\mathrm{SPARSE})$ in [8, 22]. We are following more recent notation [2].

S has at most $p(n)$ elements of length $\leq n$. Hence, one can encode a sparse set up to a given length n in a string of length polynomial in n. Thus, $\mathrm{R_T}(\mathrm{SPARSE})$ sets can be regarded as sets with small descriptions. There is another interpretation of sets in $\mathrm{R_T}(\mathrm{SPARSE})$. It is well known (see, e.g., [7]) that a set belongs to $\mathrm{R_T}(\mathrm{SPARSE})$ if and only if it has polynomial-size circuits. Since a family of polynomial-size circuits can be considered as a small description, $\mathrm{R_T}(\mathrm{SPARSE})$ sets are, also in this sense, sets with small descriptions.

Let L be any $\mathrm{R_T}(\mathrm{SPARSE})$ set. A sparse set S to which L is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible is called a *sparse set description* for L. We investigate the computational complexity of sparse set descriptions for L. First, notice that L can be nonrecursive. (For example, there is clearly some sparse set that is not recursive, although all sparse sets are by definition in $\mathrm{R_T}(\mathrm{SPARSE})$.) Then all sparse set descriptions for such L should be nonrecursive, so it does not make sense to discuss the "absolute" complexity of sparse set descriptions for L. Thus, we consider the "relative" complexity; that is, we investigate the complexity of sparse set descriptions for L relative to L. Second, notice that one can easily construct a sparse set description for L that is not recursive relative to L. That is, the problem of interest is the complexity of an "easiest" sparse set description for L. We study this type of description complexity for sets in $\mathrm{R_T}(\mathrm{SPARSE})$.

In order to state our main results, let us introduce some notions and notation. For any complexity class $\mathcal{C}$ and any set $L \in \mathrm{R_T}(\mathrm{SPARSE})$, we say that L has a *$\mathcal{C}$-self-recognizable sparse set description* if L is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some sparse set that is in $\mathcal{C}^L$. Define $\mathcal{C}$-SELF to be the class of $\mathrm{R_T}(\mathrm{SPARSE})$ sets with a $\mathcal{C}$-self-recognizable sparse set description. Intuitively, $\mathcal{C}$ represents an upper bound for the relative complexity of sparse set descriptions.

There is a general upper bound for the complexity of sparse set descriptions for $\mathrm{R_T}(\mathrm{SPARSE})$ sets. By extending [21, Lemma 5.6], one can easily show that every set L in $\mathrm{R_T}(\mathrm{SPARSE})$ has a sparse set description in $\Delta_3^{\mathrm{P},L}$ $(= \mathrm{P}^{\mathrm{NP}^{\mathrm{NP}^L}})$. In other words, $\mathrm{R_T}(\mathrm{SPARSE}) \subseteq \Delta_3^{\mathrm{P}}$-SELF. (Indeed, $\mathrm{R_T}(\mathrm{SPARSE}) = \Delta_3^{\mathrm{P}}$-SELF since by definition we have Δ_3^{P}-SELF $\subseteq \mathrm{R_T}(\mathrm{SPARSE})$.) However, we do not know whether this is the optimal upper bound for some $\mathrm{R_T}(\mathrm{SPARSE})$ set. In fact, no nontrivial lower bound has been known for any $\mathrm{R_T}(\mathrm{SPARSE})$ set; for example, it has been asked [23] whether there exists an $\mathrm{R_T}(\mathrm{SPARSE})$ set that is not in P-SELF. In our main results, we construct such a set A, i.e., $A \in \mathrm{R_T}(\mathrm{SPARSE}) -$ P-SELF, and also another set B in $\mathrm{R_T}(\mathrm{SPARSE}) -$ (NP $\cap$ co-NP)-SELF. Furthermore, we show that $A \in$ (NP $\cap$ co-NP)-SELF and $B \in$ NP-SELF. Thus, our results yield a $\mathcal{C}$-SELF hierarchy in $\mathrm{R_T}(\mathrm{SPARSE})$; that is, P-SELF $\subsetneq$ (NP $\cap$ co-NP)-SELF $\subsetneq$ NP-SELF $\subseteq \Delta_3^{\mathrm{P}}$-SELF ($= \mathrm{R_T}(\mathrm{SPARSE})$).

Note that every sparse set S has a trivial sparse set description, namely S itself, that is in P^S. Thus, a set in $\mathrm{R_T}(\mathrm{SPARSE}) -$ P-SELF must be one that is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some sparse set but that looks quite different from sparse sets. Also note that although one can easily think of a pair of L and S such that S is sparse, $L \leq_{\mathrm{T}}^{\mathrm{P}} S$, and $S \notin \mathrm{P}^L$, this does not necessarily mean that $L \in \mathrm{R_T}(\mathrm{SPARSE}) -$ P-SELF; L might have another sparse set description in P^L. That is, a set L in $\mathrm{R_T}(\mathrm{SPARSE}) -$ P-SELF must be such that *no* sparse set description for L belongs to P^L.

There are some topics closely related to our problem; our main results and/or our proof technique provide interesting observations in these topics.

Recall that a set is in $\mathrm{R_T}(\mathrm{SPARSE})$ if and only if it has polynomial-size circuits. Furthermore, from the proof of this fact it is clear that there should be a relation between the complexity of sparse set descriptions and that of finding polynomial-size circuits. Indeed, these two measures are closely related, but not exactly the same. (The difference is mainly that between "recognition" and "generation;" see Section 4.1 for more explanation.) Hence, our main results yield similar (but not the same) lower bound results for the problem of finding polynomial-size circuits. We show that the set A constructed in the first main result has no polynomial-size circuits that are polynomial-time generable relative to A and that the set B constructed in the second main result has no polynomial-size circuits that are NPSV-generable relative to B.

In order to study the difference between various reduction types, Tang and Book [22] proposed investigating classes $\mathrm{E}_r(\mathrm{TALLY})$ and $\mathrm{E}_r(\mathrm{SPARSE})$, where for any reduction type r, $\mathrm{E}_r(\mathrm{TALLY})$ (resp., $\mathrm{E}_r(\mathrm{SPARSE})$) is the class of sets that are equivalent to some tally set (resp., some sparse set) under reducibility r. (A *tally set* is any subset of 0^*.) While the structure of $\mathrm{E}_r(\mathrm{TALLY})$ classes was studied well in [22] and later in [3], many important questions concerning $\mathrm{E}_r(\mathrm{SPARSE})$ classes have been left open. Here we solve all such questions. It has been asked [23] whether P/poly has a set not in $\mathrm{E^P_T}(\mathrm{SPARSE})$, which is the same as asking whether P-SELF $\subsetneq \mathrm{R_T}(\mathrm{SPARSE})$; thus, the first main result answers this question affirmatively. We also prove various separations between $\mathrm{E}_r(\mathrm{SPARSE})$ classes that have been asked in [22]. From our main results, we immediately obtain $\mathrm{E^P_T}(\mathrm{SPARSE}) \subsetneq \mathrm{E^{SN}_T}(\mathrm{SPARSE}) \subsetneq \mathrm{E^{NP}_T}(\mathrm{SPARSE})$. By applying the proof technique we used to obtain our main results, we show that $\mathrm{E^P_{tt}}(\mathrm{SPARSE}) \subsetneq \mathrm{E^P_T}(\mathrm{SPARSE})$.

With similar motivations, Book and Ko [8] and Ko [15, 16] studied the inclusion relations between classes of sets *reducible* to sparse sets under several polynomial-time reducibilities r, $\mathrm{R}^{\mathrm{P}}_r(\mathrm{SPARSE})$. Most of the inclusion relations between these classes were well determined, except for a few ones that were formulated as conjectures in [15]. We apply again the technique used to prove our main results, and show that two of the conjectures in [15] are true. More precisely, we show that not every set $\le^{\mathrm{P}}_{\mathrm{c}}$-reducible to a sparse set is $\le^{\mathrm{P}}_{\mathrm{d}}$-reducible to a sparse set, and therefore that $\mathrm{R^P_d}(\mathrm{SPARSE}) \subsetneq \mathrm{R_T}(\mathrm{SPARSE})$.

2. Preliminaries. In this paper we follow standard definitions and notations in computational complexity theory (see [7, 21]).

We use the alphabet $\Sigma = \{0,1\}$. By *string* or *word* we mean an element of Σ^*. Let x and y be any strings, and let X be any set of strings. We denote by $x \cdot y$ the concatenation of strings x and y, by $|x|$ the length of a string x, and by $\|X\|$ the cardinality of X. We use $\overline{X}$ and $X^{\le n}$ to denote the complement of X, i.e., $\Sigma^* - X$, and the set $\{x \in X : |x| \le n\}$ respectively.

A pairing (or tupling) function on a domain $\mathcal{D}$ is a one-to-one function from $\mathcal{D} \times \mathcal{D}$ (or $\mathcal{D} \times \cdots \times \mathcal{D}$) to $\mathcal{D}$. We use various pairing (or tupling) functions depending on our purpose. Let $\langle \cdot, \cdot \rangle$ be a standard pairing function on Σ^*; we assume that this pairing function is polynomial-time computable and invertible. Let $\langle\cdot,\cdot\rangle_{\mathrm{N}}$ denote a pairing function from $\mathbf{N}\times\mathbf{N}$ *onto* $\mathbf{N}$; for example, we can define $\langle n,m\rangle_{\mathrm{N}}=(n+m)(n+m+1)/2+n$. Let $\langle\cdot,\cdot\rangle_{\mathrm{T}}$ denote a pairing function from $0^*\times 0^*$ *onto* 0^*, which can be defined, for example, as $\langle 0^n,0^m\rangle_{\mathrm{T}}=0^{\langle n,m\rangle_{\mathrm{N}}}$. We generalize these pairing functions to k-tupling functions by applying them $k-1$ times; thus, $\langle x,y,z\rangle=\langle x,\langle y,z\rangle\rangle$.

For any string x, a *self-delimiting code* of x, denoted as $\overline{x}$, is the string $u10x$, where u is obtained by doubling each bit of the binary representation of $|x|$; for example, $\overline{0}=11\cdot 10\cdot 0$, $\overline{0110}=110000\cdot 10\cdot 0110$, etc. Note that from $\overline{x}\cdot y$, one can compute both x and y in linear time. For any strings $x_1,\ldots,x_n$, we use $x_1\#\cdots\#x_n$ to denote the string $\overline{x_1}\cdots\overline{x_n}$. With our convention, we have that $|x_1\#\cdots\#x_n|\le|x_1|+\cdots+|x_n|+2n(\lceil\log(\max\{|x_1|,\ldots,|x_n|\})\rceil+1)$. The quantity $2n(\lceil\log(\max\{|x_1|,\ldots,|x_n|\})\rceil+1)$ will be referred as a *tupling overhead.*

Our computation model is the standard multi-tape (oracle) Turing machine. We consider deterministic and nondeterministic machines, all of which are either acceptors or transducers. Note that nondeterministic transducers compute, in general, multivalued functions. For any (oracle) machine M, the execution of M on an input x (relative to an oracle X) is denoted as $M(x)$ (resp., $M^X(x)$). Let $L(M)$ (resp., $L(M;X)$) denote the set of strings accepted by M (relative to X). For any complexity class $\mathcal{C}$, let $\mathcal{C}^X$ denote the complexity class $\mathcal{C}$ defined relative to X.

We use several polynomial-time reducibilities. For deterministic ones we consider $\le^{\mathrm{P}}_{f(n)\text{-tt}}$, $\le^{\mathrm{P}}_{\mathrm{c}}$, $\le^{\mathrm{P}}_{\mathrm{d}}$, $\le^{\mathrm{P}}_{\mathrm{tt}}$, and $\le^{\mathrm{P}}_{\mathrm{T}}$. See [17] for the definitions of $\le^{\mathrm{P}}_{\mathrm{tt}}$ and $\le^{\mathrm{P}}_{\mathrm{T}}$. $\le^{\mathrm{P}}_{\mathrm{c}}$ and $\le^{\mathrm{P}}_{\mathrm{d}}$ are conjunctive and disjunctive polynomial-time reducibilities; see again [17]. Note that truth-table and Turing versions of, say, $\le^{\mathrm{P}}_{\mathrm{c}}$ are equivalent in power. Therefore we make no distinction between truth-table and Turing when talking about conjunctive or disjunctive reducibilities. $\le^{\mathrm{P}}_{f(n)\text{-tt}}$ is a special case of $\le^{\mathrm{P}}_{\mathrm{tt}}$-reducibility that is defined as follows: for any function $f(n)$, $X\le^{\mathrm{P}}_{f(n)\text{-tt}}Y$ if X is reducible to Y via a $\le^{\mathrm{P}}_{\mathrm{tt}}$-reduction that makes at most $f(n)$ queries on an input of length n. We also consider two nondeterministic polynomial-time reducibilities, $\le^{\mathrm{SN}}_{\mathrm{T}}$ and $\le^{\mathrm{NP}}_{\mathrm{T}}$. For any sets A and B, $A\le^{\mathrm{SN}}_{\mathrm{T}}B$ (resp., $A\le^{\mathrm{NP}}_{\mathrm{T}}B$) if A is in $\mathrm{NP}^B\cap\text{co-}\mathrm{NP}^B$ (resp., NP^B).

In our discussions we use the notion of "resource-bounded Kolmogorov complexity." (See [18] for a survey on this notion and related topics.) Here we prepare some useful notation concerning this complexity measure.

We first fix one *universal Turing transducer* U. Roughly speaking, U is a deterministic transducer that expects as input a pair of a transducer T and a string x, and simulates T on input x. More precisely, for every transducer T (whose "program" is encoded as d) and every string x, U on input $\overline{d}\cdot x$ executes in the following way:

(i) $U(\overline{d}\cdot x)$ halts and outputs y if and only if $T(x)$ halts and outputs y, and
(ii) if $T(x)$ halts within t steps, then $U(\overline{d}\cdot x)$ halts within $|d|^2\cdot t^2$ steps.

The existence of such universal machine is shown in [11].

Now we define the following sets: For any string x and any $n,m\ge 0$, $\mathrm{KT}[n,m\,|\,x]$ is the set of strings that are generated by U from some input $y\cdot x$ within m steps, where $|y|\le n$. Define $\mathrm{KT}[n,m]$ to be $\mathrm{KT}[n,m\,|\,\epsilon]$, where ϵ denotes the empty string. Notice that for each n, m, and x, $\mathrm{KT}[n,m\,|\,x]$ contains only finitely many strings.

3. Main Results. In this section we construct two sets A and B in $\mathrm{R_T}$(SPARSE) such that A is not in P-SELF and B is not in (NP $\cap$ co-NP)-SELF. Furthermore, we show that A is in (NP $\cap$ co-NP)-SELF and B is in NP-SELF. In other words, A is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to *some* sparse set in $\mathrm{NP}^A \cap \text{co-NP}^A$ but $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to *no* sparse set in P^A, and B is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to *some* sparse set in NP^B but $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to *no* sparse set in $\mathrm{NP}^B \cap \text{co-NP}^B$. Therefore, we have that P-SELF $\subsetneqq$ (NP $\cap$ co-NP)-SELF $\subsetneqq$ NP-SELF.

THEOREM 3.1. *There is a set A in* $\mathrm{R_T}$(SPARSE) *that is not in* P-SELF.

Proof. We construct A that meets the following two requirements:

(I) For some sparse set S_A, $A \leq_{\mathrm{T}}^{\mathrm{P}} S_A$.

(II) For every sparse set S, if $A \leq_{\mathrm{T}}^{\mathrm{P}} S$, then $S \notin \mathrm{P}^A$.

Note that the second requirement is equivalent to the following: for every set X, if $A \leq_{\mathrm{T}}^{\mathrm{P}} X \leq_{\mathrm{T}}^{\mathrm{P}} A$, then X is not sparse.

We first explain the rough idea of our construction. The set A is built to be a non-sparse collection of Kolmogorov complex strings such that every string in A is hard to describe relative to any other string in A (Lemma 3.2; we can define such A so that it is $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to some sparse set S_A). Using this property, we are able to show that every $\leq_{\mathrm{T}}^{\mathrm{P}}$-reduction from A to A must be very similar to the identity function (in a way to be made precise in Lemma 3.3). Now consider any set X that is $\leq_{\mathrm{T}}^{\mathrm{P}}$-equivalent to A, that is, $A \leq_{\mathrm{T}}^{\mathrm{P}} X \leq_{\mathrm{T}}^{\mathrm{P}} A$. Then the composition of the $\leq_{\mathrm{T}}^{\mathrm{P}}$-reduction from A to X and that from X to A is regarded as a $\leq_{\mathrm{T}}^{\mathrm{P}}$-reduction from A to A; thus, such a reduction is very similar to the identity function. Hence, X must have distinguished information about most strings in A (which is non-sparse) and thus cannot be sparse.

Now we define the set A precisely. Let $\{p_l\}_{l\geq 1}$ denote an enumeration of polynomials and $\{M_i\}_{i\geq 1}$ denote an enumeration of polynomial-time deterministic Turing reductions. We can assume without loss of generality that the running time of each M_i is bounded by p_i. Also, we say that a set X is p_l-sparse if $\|X^{\leq n}\| \leq p_l(n)$ for every n. In the construction we use two fixed functions $t(n)$ and $m(n)$. Function $t(n)$ must grow faster than any polynomial; choosing $t(n) = 2^n$ is enough for this proof. For $m(n)$, we can use any unbounded function that can be computed in time polynomial in n and for which $m(n) \cdot \log n = o(n)$.

We build our set A in stages. (Recall that the function $\lambda i,j,l.\langle i,j,l\rangle_{\mathrm{N}}$ is a one-to-one and onto function from $\mathbf{N} \times \mathbf{N} \times \mathbf{N}$ to $\mathbf{N}$.) At each stage $k = \langle i,j,l\rangle_{\mathrm{N}}$, we define $A^{(k)}$ so that one of the following conditions holds:

(a) $A^{(k)} \neq L(M_i; L(M_j; A^{(k)}))$.

(b) $L(M_j; A^{(k)})$ is not p_l-sparse.

Then define $A = \cup_{k\geq 1} A^{(k)}$. We also build $S^{(k)}$ at stage k and define $S_A = \cup_{k\geq 1} S^{(k)}$ so that $A \leq_{\mathrm{T}}^{\mathrm{P}} S_A$; thereby A satisfies requirement (I).

Note that for each stage k, conditions (a) and (b) above have the following meaning: (a) guarantees that no X exists such that $A^{(k)} \leq_{\mathrm{T}}^{\mathrm{P}} X \leq_{\mathrm{T}}^{\mathrm{P}} A^{(k)}$ via M_i and M_j, and (b) guarantees that if $X \leq_{\mathrm{T}}^{\mathrm{P}} A^{(k)}$ via M_j, then X is not p_l-sparse. Furthermore, we will construct $A^{(k)}$ in such a way that the construction does not affect the previously established conditions. Thus, the defined set A satisfies requirement (II).

What follows is how we define the sets $A^{(k)}$ and $S^{(k)}$ at stage $k = \langle i,j,l\rangle_{\mathrm{N}}$.

(1) Choose $c > 0$ that is larger than the constants and the size of programs that appear during the following construction. These constants and program sizes depend only on M_i, M_j, and p_l; thus, c can be determined at this point of stage k.

(2) Define $n > 0$ to be the smallest integer such that

  (i) $c \log n + (m(n)+1)\log(2n) + 2c < n/2$,

  (ii) $2(5 + m(n))\log 2n^2 < n/4$,

  (iii) $c \cdot (t(n) + 2n^2)^2 < t(2n^2)$,

  (iv) $\binom{n}{m(n)} > p_l(p_i(n \cdot m(n)))$,

  (v) $n^c + c < t(n)$, and

  (vi) n is sufficiently large so that no string of length $\geq n$ is used or queried up to stage $k-1$ and this construction up to stage $k-1$ can be done in time n.

With the conditions placed on $m(n)$, clearly we can choose such n.

(3) Let τ be the lexicographically first string of length $2n^2$ such that $\tau \notin \mathrm{KT}[2n^2 - 1, t(2n^2)]$. Divide τ into $2n$ blocks of n bits, $w_1, \ldots, w_{2n}$, and define $W_1 = \{w_1, \ldots, w_n\}$ and $W_2 = \{w_{n+1}, \ldots, w_{2n}\}$. Let m stand for the value of $m(n)$ during this stage. Then define L_1 and L_2 as follows.

$$L_1 = \{\ w_{i_1} \cdots w_{i_m} : \text{all } w_{i_j} \text{ are different, sorted, and in } W_1\ \} \text{ and}$$
$$L_2 = \{\ w_{i_1} \cdots w_{i_m} : \text{all } w_{i_j} \text{ are different, sorted, and in } W_2\ \}$$

(where "sorted" means in lexicographical order).

It is easy to show that all blocks $w_1, \ldots, w_{2n}$ must be distinct unless we contradict the definition of τ. Hence, L_1 and L_2 are disjoint, and each of them contains $\binom{n}{m}$ strings of length $n \cdot m$. We should notice that L_1 (resp., L_2) is reducible to W_1 (resp., W_2) by a simple and uniform reduction that makes m queries; that is, to decide whether a given string $u_1 \cdots u_m$ is in L_1 (where all u_i are of length n, different, and sorted), it is enough to check whether every u_i is in W_1.

(4) Finally define $A^{(k)}$ and $S^{(k)}$ as follows.

  Let $A_1 = A^{(k-1)} \cup L_1$ and $A_2 = A^{(k-1)} \cup L_2$.

  (Case a)

  Either (i) $A_1 \neq L(M_i; L(M_j; A_1))^{\leq nm}$, or (ii) $A_2 \neq L(M_i; L(M_j; A_2))^{\leq nm}$ occurs. In this case, if (i) occurs, then set $A^{(k)}$ to A_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $A^{(k)}$ to A_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

  (Case b)

  Either (i) $L(M_j; A_1)^{\leq p_i(nm)}$ is not p_l-sparse, or (ii) $L(M_j; A_2)^{\leq p_i(nm)}$ is not p_l-sparse. In this case, if (i) occurs, then set $A^{(k)}$ to A_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $A^{(k)}$ to A_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

If the construction above is completed, then the defined sets A $(= \cup_{k \geq 1} A^{(k)})$ and S_A $(= \cup_{k \geq 1} S^{(k)})$ clearly satisfy the requirements. Thus, it suffices to show that the construction can be completed, which is proved in the following lemmas. □

We show that the construction above can be completed. For this purpose, it is enough to show that either (Case a) or (Case b) holds at every stage; in other words, at every stage, if (Case a) fails, then (Case b) certainly holds. We will prove this in Lemma 3.4.

The following lemmas refer to any stage k of the construction and use all the symbols defined in that stage. For a word $x = w_{i_1} \cdots w_{i_m}$ in $L_1 \cup L_2$, we say that the words $w_{i_1}, \ldots, w_{i_m}$ are the *blocks* of x.

We first show the following property.

LEMMA 3.2. *For every x and y in $L_1 \cup L_2$, if $x \neq y$, then $y \notin \mathrm{KT}[c\log n + c, t(n) \mid x]$.*

Proof. For sake of contradiction, assume that there is a description d_1 of size at most $c\log n + c$ such that our universal Turing machine U generates y from $d_1 \cdot x$ in time $t(n)$. We will show that this assumption implies $\tau \in \mathrm{KT}[2n^2 - 1, t(2n^2)]$, contradicting the choice of τ.

Let w be the first block in y that is not a block in x, and let $\tau - x - w$ be the string that is obtained by deleting w and all blocks of x from the word τ. Also, let ℓ be the list $\ell_0 \# \ell_1 \# \ldots \# \ell_m$ such that

- w is the ℓ_0-th block in τ, and
- for every $i \in \{1, \ldots, m\}$, the ith block in x is the ℓ_i-th block in τ.

That is, ℓ gives the information needed to insert w and x again into $\tau - x - w$.

Finally, let d_0 be the description of some program such that U generates τ from $\overline{d_0} \cdot (d_1 \# x \# \ell \# (\tau - x - w))$. Intuitively, d_0, on input $d_1 \# x \# \ell \# (\tau - x - w)$, computes y from $d_1 \cdot x$, compares x and y in order to extract w, and inserts x and w into $\tau - x - w$ from the information in ℓ, thereby obtaining τ. Note that such a program can be chosen independently of n, τ, x, y, and d_1; thus, we can regard $|d_0|$ as a fixed constant. Furthermore, the program can do this task in time $\leq b \cdot (t(n) + 2n^2)$ for some constant b that depends on d_0 only. Hence, U with $\overline{d_0} \cdot (d_1 \# x \# \ell \# (\tau - x - w))$ as input halts within $|d_0|^2 \cdot b^2 \cdot (t(n) + 2n^2)^2$ steps. Choose $c \geq |d_0|^2 \cdot b^2$. By condition (iii) on n this is less than $t(2n^2)$.

We now estimate the size of $\overline{d_0} \cdot (d_1 \# x \# \ell \# (\tau - x - w)) = d_0 \# d_1 \# x \# \ell \# (\tau - x - w)$. Note first that the size of each ℓ_i is at most $\log(2n)$. Let e be $|d_0| + |d_1| + |x| + |\ell_0| + \cdots + |\ell_m| + |\tau - x - w|$. Adding up the size of each string, we have that $e < c + (c\log n + c) + mn + (m+1)\log(2n) + (2n^2 - mn - n)$; hence, by condition (i) on n, $e < 2n^2 - n/2$. The tupling overhead for $d_0 \# d_1 \# x \# \ell \# (\tau - x - w)$ is smaller than $2(5+m)(\log e + 1) \leq 2(5+m)\log 2n^2 < n/4$ (by condition (ii)). Thus, the total size of this string is less than $2n^2 - n/2 + n/4 < 2n^2$, giving the desired contradiction. $\square$

Now we define a new oracle machine M_s by composing machines M_i and M_j. On input x ($|x| \leq n \cdot m$) and with oracle Y, M_s executes as follows (if $|x| > n \cdot m$, then M_s rejects x): (1) It first builds a table t that encodes $A^{(k-1)}$. (2) Then M_s simulates $M_i(x)$, where each query y asked by $M_i(x)$ is solved by simulating $M_j(y)$, and each query asked by $M_j(y)$ is solved by asking oracle $t \cup Y$. (3) M_s accepts x if and only if M_i accepts x.

Recalling condition (vi) on n, it is clear that there exists some polynomial p_s (depending on M_i and M_j only) that bounds the running time of M_s. Furthermore, if (Case a) does not hold, we have that $L(M_s; A_1)^{\leq nm} = A_1$ and $L(M_s; A_2)^{\leq nm} = A_2$. The next lemma says that, to perform these reductions, M_s can expect very little help from oracles A_1 and A_2.

LEMMA 3.3. *For all strings x and y in $L_1 \cup L_2$, if either $M_s^{A_1}(x)$ or $M_s^{A_2}(x)$ query y, then $x = y$.*

Proof. We will proceed by contradiction. Suppose for example that x is in L_1, that the first $\ell - 1$ queries of $M_s^{A_1}(x)$ are either x or not in $L_1 \cup L_2$, but that the ℓth one is a word $y \in L_1 \cup L_2$ such that $y \neq x$.

Define d_2 as the description of a program that, being given $\overline{\ell} \cdot x$ as an input, simulates $M_s(x)$ with oracle $\{x\}$ and prints its ℓth query. Then, on input $\overline{d_2} \cdot (\overline{\ell} \cdot x)$ $(= (d_2 \# \ell) \cdot x)$, our universal machine prints a word in $L_1 \cup L_2$ other than x.

Since the program d_2 depends only on M_s, we can regard $|d_2|$ as a constant in this stage. Because M_s is p_s-time-bounded, there must be a constant c_1, depending on M_i and M_j only, such that $|\ell| \leq c_1 \cdot \log n$. Choosing $c \geq 3 \cdot (|d_2| + c_1)$, some routine calculations show that $|d_2 \# \ell| \leq c \cdot \log n + c$. Furthermore, there must be another constant c_2, depending on M_i, M_j, and $|d_2|$ only, such that U on input $\overline{d_2} \cdot (\overline{\ell} \cdot x)$ runs in time $n^{c_2} + c_2$. Choosing $c \geq c_2$, this is less than $t(n)$ by condition (v) on n. This contradicts Lemma 3.2.

If $x \in L_2$ instead, we use oracle $\emptyset$ in the simulation of M_s. The other cases are symmetric. □

The previous lemma roughly states that the composition of reductions M_i and M_j must have a very simple structure. We can use this to show that intermediate sets in these reductions cannot be sparse.

LEMMA 3.4. *If (Case a) fails, then (Case b) holds.*

Proof. Suppose that (Case a) fails, and let X_1 and X_2 be two sets such that $A_1 \leq_T^P X_1 \leq_T^P A_1$ and $A_2 \leq_T^P X_2 \leq_T^P A_2$ via M_i and M_j. We will show that at least one of $X_1^{\leq p_i(nm)}$ or $X_2^{\leq p_i(nm)}$ is not p_l-sparse.

Define function C as follows. For any $x \in L_1 \cup L_2$, $C(x)$ is the first query made by $M_i^{X_1}(x)$ that has different answers from oracles X_1 and X_2. Note that $C(x)$ is always defined for every x in $L_1 \cup L_2$. Otherwise, $M_i(x)$ gives the same answer with oracles X_1 and X_2; but this answer cannot be NO because x is in one of L_1 and L_2, and cannot be YES because L_1 and L_2 are disjoint.

Since $C(x)$ is always an element of either X_1 or X_2, we can regard C as a mapping from $L_1 \cup L_2$ into $X_1 \cup X_2$. Furthermore, we show in the following that C yields a different element in $X_1 \cup X_2$ for each one in $L_1 \cup L_2$.

CLAIM. The function C is one-to-one on the domain $L_1 \cup L_2$.

Proof of the Claim. Take any $x \in L_1 \cup L_2$ and name $y = C(x)$. Let us show first the following fact: both $M_j^{A_1}(y)$ and $M_j^{A_2}(y)$ query x.

We know that y must have different answers with respect to oracles X_1 and X_2. With our assumptions, this means that $M_j(y)$ must answer differently with oracles A_1 and A_2. But these oracles differ only in the words in L_1 and L_2, and neither $M_j^{A_1}(y)$ nor $M_j^{A_2}(y)$ can query any of these words except for x; otherwise, so does $M_s(x)$ and we have a contradiction to Lemma 3.3. Therefore, to give a different answer, both $M_j^{A_1}(y)$ and $M_j^{A_2}(y)$ must query x.

To prove that C is one-to-one, assume for example that also $C(x') = y$ and that

$M_i^{X_1}(x')$ queries y, for some $x' \in L_1 \cup L_2$ different from x. We have now that $M_s^{A_1}(x')$ queries x when simulating $M_j(y)$. This contradicts again Lemma 3.3. □ *Claim*

Define now $C_1 = \{ x \in L_1 \cup L_2 : C(x) \in X_1 \}$ and $C_2 = \{ x \in L_1 \cup L_2 : C(x) \in X_2 \}$. Since C is one-to-one on $L_1 \cup L_2$, we have

$$\|X_1^{\leq p_i(nm)}\| + \|X_2^{\leq p_i(nm)}\| \geq \|C_1\| + \|C_2\| = \|C_1 \cup C_2\|.$$

But C_1 and C_2 cover all of $L_1 \cup L_2$, so

$$\|C_1 \cup C_2\| = \|L_1 \cup L_2\| = 2\binom{n}{m} > 2p_l(p_i(nm)).$$

(by condition (iv) on n).

Thus, $\|X_1^{\leq p_i(nm)}\| + \|X_2^{\leq p_i(nm)}\| > 2 \cdot p_l(p_i(nm))$. Therefore, at least one of $\|X_1^{\leq p_i(nm)}\|$ and $\|X_2^{\leq p_i(nm)}\|$ must be greater than $p_l(p_i(nm))$, and hence, either $X_1^{\leq p_i(nm)}$ or $X_2^{\leq p_i(nm)}$ is not p_l-sparse. □

From Theorem 3.1, we know that A has no sparse set description that is P^A-recognizable. Here we give an upper bound; that is, we show that A has a sparse set description in $\mathrm{NP}^A \cap \text{co-NP}^A$.

Indeed, the sparse set S_A constructed in Theorem 3.1 is in $\mathrm{NP}^A \cap \text{co-NP}^A$. It is easy to see that S_A is in NP^A by considering the following procedure. To check that a word w is in S_A, guess a word x that has w as a block and verify that $x \in A$. On the other hand, because of the fact that S_A has an easy "census function," we can prove that $S_A \in \text{co-NP}^A$.

For any set L, the following function cens_L is called the *census function* of L: For each $n \geq 0$, $\mathrm{cens}_L(0^n) = \|L^{\leq n}\|$. Note that S_A has n elements at all the lengths n used in its construction, and is empty at all others. Together with the following proposition, this shows that $S_A \in \mathrm{NP}^A \cap \text{co-NP}^A$.

PROPOSITION 3.5. *For every pair of sets X and Y, if X is sparse and in NP^Y, and its census function is polynomial-time computable, then X is in $\mathrm{NP}^Y \cap \text{co-NP}^Y$.*

Proof. It is enough to show that $\overline{X} \in \mathrm{NP}^Y$. Since $X \in \mathrm{NP}^Y$, there is a polynomial-time nondeterministic machine M that recognizes X relative to Y. For a given u, we can check that u is not in X by the following procedure:

(1) Compute $c = \mathrm{cens}_X(0^{|u|})$.
(2) Guess c different words v such that $v \neq u$, $|v| \leq |u|$, and $v \in X$. (The last condition is verified by M relative to Y.)
(3) Conclude that u is not in X if and only if c different words are found (and verified) at (2).

Clearly, this procedure witnesses that $\overline{X} \in \mathrm{NP}^Y$. □

COROLLARY 3.6. *The set A defined in Theorem 3.1 is in* $(\mathrm{NP} \cap \text{co-NP})$-SELF $-$ P-SELF.

Next we construct a set having only sparse set descriptions of higher complexity; we will show a set B that is in $\mathrm{R_T}(\text{SPARSE})$ but not in $(\mathrm{NP} \cap \text{co-NP})$-SELF. To simplify the following discussion, let us introduce the following notions.

A *three-output machine* is a nondeterministic Turing machine with three final states: ACC, REJ, and ? (standing for "accept," "reject," and "don't know"). Let M be any three-output oracle machine, and let Y be any oracle set. M is called *strong under oracle* Y if for all x, $M^Y(x)$ has either at least one computation path ending in ACC, or at least one ending in REJ, but not both. When M is strong under oracle Y, we define $L(M;Y)$ to be the set of strings x for which $M^Y(x)$ has at least one computation path ending in ACC. $L(M;Y)$ can be regarded as the set recognized by M relative to Y.

These are taken from the original definition of $\leq_{\mathrm{T}}^{\mathrm{SN}}$-reducibility in [19], where the following fact is proved.

PROPOSITION 3.7. *For any sets X and Y, $X \leq_{\mathrm{T}}^{\mathrm{SN}} Y$ if and only if there is a polynomial-time three-output machine M that is strong under oracle Y and that recognizes X relative to Y, i.e., $X = L(M;Y)$.*

Our second main result is as follows.

THEOREM 3.8. *There is a set B in* $\mathrm{R_T(SPARSE)}$ *such that for every sparse set S, if $B \in \mathrm{NP}^S \cap \text{co-}\mathrm{NP}^S$, then $S \notin \mathrm{NP}^B \cap \text{co-}\mathrm{NP}^B$. Thus, B is not in* $(\mathrm{NP} \cap \text{co-NP})$-SELF.

REMARK. For proving $B \notin (\mathrm{NP} \cap \text{co-NP})$-SELF, it suffices to show that for any sparse set S, if $B \leq_{\mathrm{T}}^{\mathrm{P}} S$, then $S \notin \mathrm{NP}^B \cap \text{co-}\mathrm{NP}^B$. The theorem above is slightly stronger than what we need for the moment. We will use this stronger version in Section 4.2.

Proof. We construct B that meets:

(I) For some sparse set S_B, $B \leq_{\mathrm{T}}^{\mathrm{P}} S_B$.

(II) For every set X, if $B \leq_{\mathrm{T}}^{\mathrm{SN}} X \leq_{\mathrm{T}}^{\mathrm{SN}} B$, then X is not sparse.

The construction is quite similar to that of Theorem 3.1, but now we must attack nondeterministic machines that may have many computation paths. Note that our construction technique does not seem to yield a set L in $\mathrm{R_T(SPARSE)} - \text{NP-SELF}$; that is, a set L constructed by our technique always has a NP^L-recognizable sparse set description. Then from Proposition 3.5, if such a sparse set description has an easy census function, its complexity goes down to $\mathrm{NP}^L \cap \text{co-}\mathrm{NP}^L$. Thus, the key to construct the desired set B is to make sure that it is reducible to no sparse set with an easy census function.

In order to explain how we use this point, let us consider any pair of machines that may reduce $B \leq_{\mathrm{T}}^{\mathrm{SN}} X \leq_{\mathrm{T}}^{\mathrm{SN}} B$ for some set X. Similarly to the previous proof, define M_s to be a machine obtained by composing these two machines; i.e., M_s is a $\leq_{\mathrm{T}}^{\mathrm{SN}}$-reduction from B to B. The requirement we consider first is that M_s must be strong under B. But during the construction at stage k, we can try M_s with various extensions of $B^{(k-1)}$ that have different census values. If M_s fails to be strong under some extension, we can define the set $B^{(k)}$ so that the pair of machines does not make a correct $\leq_{\mathrm{T}}^{\mathrm{SN}}$-reduction from B to B. Otherwise, M_s is strong under any extension; furthermore, the extension $B' = B^{(k)}$ is such that $M_s^{B'}$ always has a path ending in REJ state for every sufficiently large input. We choose one such path for every input thereby fixing a nondeterministic path of M_s. Then we regard M_s (on such a path) as a deterministic machine and, using an argument similar to the previous proof, we can prove that the intermediate set X must be non-sparse.

To define the sets B and S_B, let $\{M_i\}_{i\geq 1}$ denote an enumeration of polynomial-time nondeterministic three-output machines. We can assume that the running time of each M_i is bounded by p_i. Choose $m(n)$ as before, but choose the time bound $t(n) = 2^{2^n}$, much larger than before.

We build our set B in stages. At each stage $k = \langle i, j, l \rangle_{\mathrm{N}}$, we define $B^{(k)}$ so that one of the following conditions holds:

(a) Either M_j is not strong under oracle $B^{(k)}$, or M_i is not strong under oracle $L(M_j; B^{(k)})$.

(b) $B^{(k)} \neq L(M_i; L(M_j; B^{(k)}))$.

(c) $L(M_j; B^{(k)})$ is not p_l-sparse.

Then define $B = \cup_{k \geq 1} B^{(k)}$. We also build $S^{(k)}$ at stage k and define $S_B = \cup_{k \geq 1} S^{(k)}$ so that $B \leq_{\mathrm{T}}^{\mathrm{P}} S_B$; thereby B satisfies requirement (I). Also, if we always achieve (a), (b), or (c) then B clearly satisfies requirement (II).

To define $B^{(k)}$ and $S^{(k)}$ at stage $k = \langle i, j, l \rangle_{\mathrm{N}}$, we follow steps (1) to (3) in the proof of Theorem 3.1; that is, choose c, n, m, τ, and define W_1, W_2, L_1, and L_2. Then do step (4) below.

(4) Define $B^{(k)}$ and $S^{(k)}$ as follows.

Let $B_1 = B^{(k-1)} \cup L_1$ and $B_2 = B^{(k-1)} \cup L_2$.

(Case a)

One of the following holds: (i) M_j is not strong under $B^{(k-1)}$ on some input of length at most $p_i(nm)$, (ii) M_i is not strong under $L(M_j; B^{(k-1)})$ on some input of length at most nm, or (iii) $L(M_i; L(M_j; B^{(k-1)}))^{\leq nm} \neq B^{(k-1)}$. Then set $B^{(k)}$ to $B^{(k-1)}$ and $S^{(k)}$ to $\emptyset$.

(Case b)

Either (i) $L(M_i; L(M_j; B_1))^{\leq nm} \neq B_1$, or (ii) $L(M_i; L(M_j; B_2))^{\leq nm} \neq B_2$.[2] In this case, if (i) occurs, then set $B^{(k)}$ to B_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $B^{(k)}$ to B_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

(Case c)

Either (i) $L(M_j; B_1)^{\leq p_i(nm)}$ is not p_l-sparse, or (ii) $L(M_j; B_2)^{\leq p_i(nm)}$ is not p_l-sparse. In this case, if (i) occurs, then set $B^{(k)}$ to B_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $B^{(k)}$ to B_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

If the construction above is completed, then the defined sets B $(= \cup_{k \geq 1} B^{(k)})$ and S_B $(= \cup_{k \geq 1} S^{(k)})$ clearly satisfy the requirements. The following lemmas show that it can always be completed. □

Consider again any stage k. Lemma 3.2 is still true here. M_s is also defined similarly by using M_i and M_j. Now, for each word x of length $n \cdot m$, define r_x to be the lexicographically least computation path of $M_s(x)$ relative to $\emptyset$ that ends in state REJ (if such a path exists).

It is important to notice that, if r_x exists, it can be found recursively from x by brute force search of the computation tree of $M_s(x)$. More precisely, r_x can be found

[2] (Case b) also holds if any of the computations involved fails to be strong, as in (Case a). We omit the details.

in time $2^{p(n)}$ for some polynomial p that depends on M_i and M_j only. The following property of r_x is crucial in our proof.

LEMMA 3.9. *If (Case a) does not hold, then r_x exists for every $x \in L_1 \cup L_2$.*

Proof. If (Case a) does not hold, then $M_i(x)$ is strong under $L(M_j; B^{(k-1)})$ because x has length nm. Furthermore, $M_j(z)$ is strong under $B^{(k-1)}$ for every possible query z of $M_i(x)$, because z must have length at most $p_i(nm)$. Suppose that no path of $M_s^{\emptyset}(x)$ ends in state REJ. Because M_s is the composition of M_i and M_j and the composition of strong computations is strong, some path of $M_s^{\emptyset}(x)$ must end in ACC. That is, M_i accepts x with oracle $L(M_j; B^{(k-1)})$. But this means that $L(M_i; L(M_j; B^{(k-1)}))^{\leq nm} \neq B^{(k-1)}$ because $x \notin B^{(k-1)}$. A contradiction. □

The next lemma corresponds to Lemma 3.3.

LEMMA 3.10. *Assume that (Case a) does not hold. For all strings x and y in $L_1 \cup L_2$, if y is queried in r_x, then $x = y$.*

Proof. If (Case a) does not hold, then, by the previous lemma, r_x exists for every $x \in L_1 \cup L_2$, and moreover can be found in time smaller than $t(n)$. Hence, one can easily modify the proof of Lemma 3.3; the detail is omitted. □

The following lemma is the counterpart of Lemma 3.4. Its proof is almost the same when considering only the path r_x for every $x \in L_1 \cup L_2$.

LEMMA 3.11. *If both (Case a) and (Case b) fail, then (Case c) holds.*

Proof. Suppose that (Case a) and (Case b) fail, and let X_1 and X_2 be two sets such that $B_1 \leq_{\mathrm{T}}^{\mathrm{SN}} X_1 \leq_{\mathrm{T}}^{\mathrm{SN}} B_1$ and $B_2 \leq_{\mathrm{T}}^{\mathrm{SN}} X_2 \leq_{\mathrm{T}}^{\mathrm{SN}} B_2$ via M_i and M_j. We will show that at least one of $X_1^{\leq p_i(nm)}$ or $X_2^{\leq p_i(nm)}$ is not p_l-sparse.

For any $x \in L_1 \cup L_2$, let t_x be the computation path of $M_i(x)$ that $M_s^{\emptyset}(x)$ is simulating along path r_x. It is clear that t_x is uniquely determined by r_x. Now define function C as follows. For each x in $L_1 \cup L_2$, $C(x)$ is the first query along t_x that has different answer from oracles X_1 and X_2. In a way similar to Lemma 3.4 we can show that

(i) function C is defined on $L_1 \cup L_2$, and

(ii) it is one-to-one on $L_1 \cup L_2$.

To prove (i), assume that $x \in L_1$. As L_1 and L_2 are disjoint, $x \notin L_2$. By Lemma 3.10, no query along r_x is in L_2, so all queries in r_x have the same answer from oracles $\emptyset$ and L_2. Hence, t_x is a correct rejecting path of $M_i(x)$ relative to X_2. On the other hand, because $x \in L_1$, $M_i(x)$ has no rejecting path relative to X_1, so some query in t_x must have different answers with oracles X_1 and X_2.

Part (ii) is proved as the Claim in Lemma 3.4, using repeatedly Lemma 3.10. Then, the same counting argument of Lemma 3.4 shows that at least one of $\|X_1^{\leq p_i(nm)}\|$ and $\|X_2^{\leq p_i(nm)}\|$ must be greater than $p_l(p_i(nm))$. □

The separation of NP-SELF from (NP $\cap$ co-NP)-SELF follows easily.

COROLLARY 3.12. *The set B defined in Theorem 3.8 is in* NP-SELF $-$ (NP $\cap$ co-NP)-SELF.

Proof. Recall the algorithm that is given before Proposition 3.5 for showing $S_A \in \mathrm{NP}^A$. Clearly, a similar idea proves that $S_B \in \mathrm{NP}^B$. Thus, S_B witnesses that B is in NP-SELF. □

4. Related Topics. This section presents some consequences of our results and applications of our proof techniques to other but closely related topics.

4.1. Complexity of Finding Polynomial-Size Circuits. In the previous section we have investigated the the problem of recognizing sparse set descriptions. Recall that sets having sparse set descriptions are exactly those accepted by polynomial-size circuits. (Because the term "polynomial-size" is always assumed in the following discussion, it is often omitted.) However, the relative complexity of a sparse set S such that $L \leq_{\mathrm{T}}^{\mathrm{P}} S$ is not exactly representing the relative complexity of finding circuits for L. Here we will first make such difference clear and then use the results in the previous section to show some lower bounds for the complexity of finding circuits.

First we review the notion of "having polynomial-size circuits." For any set L, we say that L has *polynomial-size circuits* if for some polynomial p and for every $n \geq 0$, there exists a boolean circuit consisting of $p(n)$ (or less) gates that determines whether $x \in L$ for every $x \in \Sigma^n$. Notice that the definition above does not guarantee that a set L with polynomial-size circuits is recognized by some single machinery; we just know that L is recognized by a collection of circuits. Thus, the class of sets with polynomial-size circuits does not fit in conventional complexity classes. Karp and Lipton [13] introduced a general framework for such nonuniform complexity classes, which is useful for studying the class of sets having polynomial-size circuits. By using one of their nonuniform complexity classes — namely, P/poly — one can completely characterize the class of sets with polynomial-size circuits. (This fact is essentially proved by Pippenger [20]. A good and complete proof can be found in [7].) Thus, we can consider that P/poly is identical to the class of sets with polynomial-size circuits.

Let us review the definition of P/poly. Define a class of functions, poly, by

$$\mathrm{poly} = \{\ g : 0^* \to \Sigma^* \ :\ \exists p : \text{polynomial}, \forall n \geq 0\ [\ |g(0^n)| \leq p(n)\ ]\ \}.$$

Then the class P/poly is defined as follows.

$$\begin{array}{lll} L \in \mathrm{P/poly} & \leftrightarrow & \text{there exist } g \in \mathrm{poly} \text{ and } E \in \mathrm{P} \text{ such that} \\ & & \forall n \geq 0, \forall x \in \Sigma^n\ [\ x \in L \ \leftrightarrow\ \langle g(0^n), x\rangle \in E\ ]. \end{array}$$

From the proof showing the equivalence between P/poly and the class of sets with polynomial-size circuits, one can easily fix some set E_0 so that for every L, a function g satisfying the above with E_0 is regarded as a *circuit generator*; that is, $g(0^n)$ denotes the circuit for $L^{=n}$ and the computation of E_0 on $\langle g(0^n), d_1 \cdots d_n\rangle$ (where $d_i \in \{0,1\}$) is the evaluation of the circuit (denoted as) $g(0^n)$ with $d_1, \ldots, d_n$ on its input gates.

Now we are ready to discuss the "complexity" of finding circuits formally. For any $L \in \mathrm{P/poly}$, we regard the relative complexity of a circuit generator for L as the "complexity" of finding circuits for L. For example, suppose that some set $L \in \mathrm{P/poly}$ has a generator g that is polynomial-time computable relative to L, i.e., $g \in \mathrm{PF}^L$. Then we consider that L has circuits that are easy to find, i.e., polynomial-time computable, relative to L. On the other hand, a set $H \in \mathrm{P/poly}$ such that no circuit generator for H is in $\mathrm{PF}^{\mathrm{NP}^H}$ is considered as a set with no easy-to-find circuits. Let us introduce the following notion: For any complexity class $\mathcal{CF}$ of functions and any set $L \in \mathrm{P/poly}$, we say that a set L has *$\mathcal{CF}$-self-generable circuits* if there is a circuit generator for L that is in $\mathcal{CF}^L$. The class $\mathcal{CF}$-$\mathrm{SELF_{gen}}$ is defined to be the class of sets with $\mathcal{CF}$-self-generable circuits. Ko [14] introduced the notion of "having self-p-producible circuits," which is the same as "having PF-self-generable circuits." Thus, ours is a generalization of his notion.

It is well known (see, e.g., [7]) that P/poly equals $\mathrm{R_T(SPARSE)}$, that is, L has polynomial-size circuits if and only if L is reducible to some sparse set. Thus, the complexity of finding circuits for L seems to correspond to the complexity of a sparse set description for L. Indeed, the two notions "having a $\mathcal{C}$-self-recognizable sparse set description" and "having $\mathcal{CF}$-self-generable circuits" are closely related (where $\mathcal{CF}$ is a function class corresponding to $\mathcal{C}$). However, they are not the same; in other words, classes $\mathcal{C}$-SELF and $\mathcal{CF}$-$\mathrm{SELF_{gen}}$ are not the same in general. This difference is mainly the one between "recognition" and "generation," which is often seen in computational complexity theory. Usually "recognition" is easier than "generation." For example, while every PF-printable set [12] is clearly P-recognizable (and sparse), we do not know whether every sparse set in P is PF-printable. (As a matter of fact, there is some evidence for the existence of a sparse set in P that is not PF-printable [3].) The best known relation is that every sparse set in P is $\mathrm{PF^{NP}}$-printable. Here we have similar relationships.

In order to discuss the relation between $\mathcal{C}$-SELF and $\mathcal{CF}$-$\mathrm{SELF_{gen}}$ classes, we introduce one more class which is located between $\mathcal{C}$-SELF and $\mathcal{CF}$-$\mathrm{SELF_{gen}}$.

$$\mathcal{C}\text{-TALLY-SELF} = \{\ L\ :\ \exists T \text{ tally } [L \leq_{\mathrm{T}}^{\mathrm{P}} T\ \wedge\ T \in \mathcal{C}^L]\ \}.$$

The relationship between $\mathcal{C}$-SELF, $\mathcal{C}$-TALLY-SELF, and $\mathcal{CF}$-$\mathrm{SELF_{gen}}$ is summarized as follows.

Proposition 4.1. *Let $\mathcal{C}$ be any "standard" complexity class of languages, and let $\mathcal{CF}$ be the corresponding complexity class of functions.*

(1) $\mathcal{CF}$-$\mathrm{SELF_{gen}} \subseteq \mathcal{C}$-TALLY-SELF $\subseteq \mathcal{C}$-SELF.

(2) $\mathcal{C}$-TALLY-SELF $\subseteq \mathrm{PF}^{\mathcal{C}}$-$\mathrm{SELF_{gen}}$.

(3) $\mathcal{C}$-SELF $\subseteq \mathrm{NP}^{\mathcal{C}}$-TALLY-SELF.

(4) If $\mathcal{C}$ is closed under $\leq_{\mathrm{c}}^{\mathrm{NP}}$ reducibility, then $\mathcal{C}$-SELF $= \mathcal{C}$-TALLY-SELF.

Remark.

(1) For any class $\mathcal{C}$ containing Δ_3^{P}, $\mathcal{CF}$-$\mathrm{SELF_{gen}} = \mathcal{C}$-TALLY-SELF $= \mathcal{C}$-SELF (*Cf.* Part (1) of the proposition). This is because $\mathrm{P/poly} \subseteq \Delta_3^{\mathrm{P}}$-$\mathrm{SELF_{gen}}$ [21]. However, the analog relation is not known for lower complexity classes.

(2) From (2) and (3), we have $\mathcal{C}$-SELF $\subseteq \mathrm{PF}^{\mathrm{NP}^{\mathcal{C}}}$-$\mathrm{SELF_{gen}}$.

(3) From (4), we have Σ_k^{P}-SELF $= \Sigma_k^{\mathrm{P}}$-TALLY-SELF for any $k \geq 1$.

In the following, we only state the proof of part (3) of the proposition and omit the other proofs. (Inclusions in parts (1) and (2) are either immediate from the definition or easy to prove, and part (4) is proved similarly as part (3).)

Proof of part (3). Let L be any set in $\mathcal{C}$-SELF. Thus, there exists some sparse set S such that $L \leq_{\mathrm{T}}^{\mathrm{P}} S$ and $S \in \mathcal{C}^L$.

Define a tally set T_S as follows:

$$T_S = \{\langle 0^n, 0^m, 0^i, 0^j, 0^b\rangle_{\mathrm{T}} \ : \ n,m,i,j \geq 0, \ b \in \{0,1\}, \\ \exists s_1 < \ldots < s_m \in S^{\leq n} \text{ [the } i\text{th bit of } s_j \text{ is } b \text{] }\}.$$

Then it is easy to see that $S^{\leq n}$ can be enumerated in polynomial time by using T_S as an oracle. (Without loss of generality, we are assuming that S does not contain the empty string.) Thus, $L \leq_{\mathrm{T}}^{\mathrm{P}} T_S$. On the other hand, clearly $T_S \in \mathrm{NP}^S$; thus, $T_S \in \mathrm{NP}^{\mathcal{C}^L}$. ◻

Among many relations generated from Proposition 4.1 we are interested, in particular, in the following ones.

COROLLARY 4.2.

(1) PF-SELF$_{\mathrm{gen}}$ = P-TALLY-SELF $\subseteq$ P-SELF.

(2) PF$^{\mathrm{NP}}$-SELF$_{\mathrm{gen}}$ = Δ_2^{P}-TALLY-SELF $\subseteq$ Δ_2^{P}-SELF.

(3) PF$^{\Sigma_2^{\mathrm{P}}}$-SELF$_{\mathrm{gen}}$ = Δ_3^{P}-TALLY-SELF = Δ_3^{P}-SELF. *Thus,* P/poly = PF$^{\Sigma_2^{\mathrm{P}}}$-SELF$_{\mathrm{gen}}$.

Note that PF$^{\Sigma_2^{\mathrm{P}}}$ is a general upper bound for the complexity of finding circuits, essentially the same one we know for the complexity of a sparse set description.

Now let us discuss two lower bounds for the complexity of finding circuits that are consequences of our main results. Recall that recognizing a sparse set description is easier than finding circuits. Thus, the lower bound results we had in the previous section yield at least the same lower bounds for the complexity of finding circuits.

The first lower bound is from Theorem 3.1.

THEOREM 4.3. *There is a set in* P/poly *that is not in* PF-SELF$_{\mathrm{gen}}$.

Proof. Note that P/poly = $\mathrm{R_T}$(SPARSE) and that PF-SELF$_{\mathrm{gen}}$ $\subseteq$ P-SELF. Thus, the set A defined in Theorem 3.1 clearly satisfies the theorem. ◻

We should note here that the same lower bound is obtained from observations in [1, 6]. Balcázar and Book [6] showed that if L is in PF-SELF$_{\mathrm{gen}}$ (in their notation, L has a self-p-producible circuit), then L has a certain lowness property (more specifically, L has EL_1-lowness). On the other hand, Allender and Hemachandra [1] constructed a sparse set S_0 that does not have such lowness. Thus, S_0 satisfies the theorem. Note that every sparse set is trivially in P-SELF; hence, S_0 satisfies the theorem above because $S_0 \in$ P-SELF $-$ PF-SELF$_{\mathrm{gen}}$ while A does so because $A \in$ P/poly $-$ P-SELF.

Next we improve the lower bound by using Theorem 3.8. In order to state our result, we need a complexity class of functions — NPSV — which is often used in the literature (see, e.g., [10]). Let NPSV (resp., NPSV^L) be the class of functions computed by polynomial-time nondeterministic single-valued transducers (relative to L).

THEOREM 4.4. *There is a set in* P/poly *that is not in* NPSV-SELF$_{\mathrm{gen}}$.

Proof. Note that (NP $\cap$ co-NP)-TALLY-SELF $\subseteq$ (NP $\cap$ co-NP)-SELF; then it immediately follows from the next lemma that the set B defined in Theorem 3.8 satisfies the theorem. ◻

LEMMA 4.5. NPSV-SELF$_{\mathrm{gen}}$ = (NP $\cap$ co-NP)-TALLY-SELF.

Proof. First we prove NPSV-SELF$_{\mathrm{gen}}$ $\subseteq$ (NP $\cap$ co-NP)-TALLY-SELF. Let L be any set in NPSV-SELF$_{\mathrm{gen}}$. Then L has a circuit generator $g \in \mathrm{NPSV}^L$. Define a tally set T by $T = \{ \langle 0^n, 0^i, 0^b\rangle_{\mathrm{T}}$: the ith bit of $g(0^n)$ is $b \in \{0,1\}$ $\}$. Then one can easily produce $g(0^n)$ using T as an oracle; hence, $L \leq_{\mathrm{T}}^{\mathrm{P}} T$.

Let N be a polynomial-time nondeterministic single-valued oracle transducer that computes g relative to L. Note that $g(0^n)$ is defined for all $n \geq 0$; hence, for every n, N^L on 0^n has an accepting computation, and N^L outputs $g(0^n)$ on every accepting computation. Once $g(0^n)$ is obtained, one can easily determine whether a given $\langle 0^n, 0^i, 0^b \rangle_{\mathrm{T}}$ is in T. Thus, by modifying N, we can obtain polynomial-time nondeterministic oracle acceptors M_1 and M_2 that respectively accept T and $\overline{T}$ relative to L.

Next we prove (NP $\cap$ co-NP)-TALLY-SELF $\subseteq$ NPSV-SELF$_{\mathrm{gen}}$. Let L be any set in (NP $\cap$ co-NP)-TALLY-SELF; then there exists a tally set T such that $L \in \mathrm{P}^T$ and $T \in \mathrm{NP}^L \cap \text{co-NP}^L$. Let M and p_M respectively denote a polynomial-time deterministic acceptor that accepts L relative to T and a polynomial time bound for M. (Note that M on x cannot query strings of length $> p_M(|x|)$.) Define $g : 0^* \to \Sigma^*$ as follows. For each $n \geq 0$, $g(0^n) = \sigma_1 \cdots \sigma_{p_M(n)}$, where $\sigma_i = 1 \leftrightarrow 0^i \in T$. Clearly, g is a circuit generator for L (when using an appropriate circuit evaluator).

Now to complete the proof we only need to construct a polynomial-time nondeterministic single-valued oracle transducer N that computes g relative to L, thereby showing $g \in \mathrm{NPSV}^L$. Let M_1 and M_2 be polynomial-time nondeterministic oracle acceptors that respectively accept T and $\overline{T}$ relative to L. The computation of N^L on input 0^n proceeds as follows: For each i, $1 \leq i \leq p_M(n)$, N^L guesses the value of σ_i (i.e., $\sigma_i = 1$ if $0^i \in T$, and $\sigma_i = 0$ if $0^i \notin T$) and verifies it by using either M_1 or M_2. Then N^L outputs $\sigma_1 \cdots \sigma_{p_M(n)}$ on each nondeterministic computation that guesses all the σ_i correctly and that verifies their correctness. Clearly, N^L is single-valued and computes g. $\Box$

4.2. Equivalence and Reducibility to Sparse Sets. Intuitively, the (relative) complexity of an easiest sparse set description for a language L gives a lower bound on the power of the reduction that we must use to recognize it (relative to L). Similarly, we have seen that the complexity of its easiest tally set description gives lower bounds on the resources used to generate any sparse set description for L.

Note that, up to now, we have insisted on keeping a $\leq_{\mathrm{T}}^{\mathrm{P}}$ reduction between L and its descriptions because we wanted to study sets in P/poly. If we relax this condition, we can then ask what is the minimum reduction type that allows us to *simultaneously* recognize L and its easiest sparse set description relative to each other. This leads naturally to the definition of classes E_r(SPARSE) and E_r(TALLY), that are extensively studied in [2, 3, 22].

For a reduction type r, two sets A and B are $\leq_r$-equivalent if $A \leq_r B$ and $B \leq_r A$. We define E_r(SPARSE) as the class of sets L for which there is a sparse set S such that L and S are $\leq_r$-equivalent. E_r(TALLY) is defined similarly. If reduction r is transitive, the class E_r(SPARSE) is also called the *equivalence degree* of the sparse sets under r. As explained in [23], this is not a proper name for non-transitive reduction types such as $\leq_{\mathrm{T}}^{\mathrm{NP}}$ and $\leq_{f(n)\text{-tt}}^{\mathrm{P}}$, because these reducibilities do not define equivalence relations.

It follows directly from the definitions that P-SELF $= \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}$(SPARSE); in particular, every set in P-SELF is $\leq_{\mathrm{T}}^{\mathrm{P}}$-equivalent to its easiest sparse set description. Also, since $\leq_{\mathrm{T}}^{\mathrm{P}}$ implies $\leq_{\mathrm{T}}^{\mathrm{SN}}$ and $\leq_{\mathrm{T}}^{\mathrm{NP}}$ reducibilities, clearly (NP $\cap$ co-NP)-SELF $\subseteq \mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}$(SPARSE) and NP-SELF $\subseteq \mathrm{E}_{\mathrm{T}}^{\mathrm{NP}}$(SPARSE).

Let us consider now a class defined by equivalence to tally sets. Recalling the definition in Section 4.1, it is immediate that P-TALLY-SELF = $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}$(TALLY). Therefore, from Corollary 4.2 (1), we have that PF-SELF$_{\mathrm{gen}}$ = $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}$(TALLY). This fact was already in [6, Theorem 3.1], where it is shown that sets with self-p-producible circuits are exactly those $\leq_{\mathrm{T}}^{\mathrm{P}}$-equivalent to tally sets.

Concerning this class, Allender and Watanabe ask in [3] whether $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}$(TALLY) = $\mathrm{E}_{\mathrm{tt}}^{\mathrm{P}}$(SPARSE). This question arises from the fact that any set $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible to a sparse set is also $\leq_{\mathrm{T}}^{\mathrm{P}}$-reducible (and thus, $\leq_{\mathrm{tt}}^{\mathrm{P}}$-reducible) to some tally set [8]. Therefore, the reduction classes of tally and sparse sets are equal for reducibilities $\leq_{\mathrm{T}}^{\mathrm{P}}$ and $\leq_{\mathrm{tt}}^{\mathrm{P}}$ (and all of them equal P/poly). Similarly, when considering equivalence to a tally set, we may ask whether it is possible to trade the power of $\leq_{\mathrm{T}}^{\mathrm{P}}$ reductions for the access to a slightly more complex oracle, namely, a sparse set.

We can now formulate again the results presented in Section 3, to give various separations between the equivalence classes mentioned above. For example, we have pointed out that the construction in Theorem 3.1 can use any unbounded and easily computable function $m(n)$ to give a set A that is not in P-SELF = $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}$(SPARSE). The reader will easily verify that A is $\leq_{m(n)\text{-tt}}^{\mathrm{P}}$-reducible to the sparse set S_A given in that theorem. Therefore, we have

THEOREM 4.6. *For any function $m(n)$ that is unbounded and computable in time polynomial in n,* $\mathrm{R}_{m(n)\text{-tt}}^{\mathrm{P}}(\mathrm{SPARSE}) \not\subseteq \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE})$.

Furthermore, the reduction to S_A is conjunctive, so the result above holds in fact for $\leq_{m(n)\text{-c}}^{\mathrm{P}}$ reducibility.

Because the set A is in (NP$\cap$co-NP)-SELF $\subseteq \mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}$(SPARSE), we have the following corollary.

COROLLARY 4.7. $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE}) \subsetneqq \mathrm{P/poly} \cap \mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}(\mathrm{SPARSE})$.

Similarly, from Theorem 3.8 and Corollary 3.12, we know that the set $B \in$ NP-SELF is not in $\mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}$(SPARSE). As a consequence we have

COROLLARY 4.8. $\mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}(\mathrm{SPARSE}) \cap \mathrm{P/poly} \subsetneqq \mathrm{E}_{\mathrm{T}}^{\mathrm{NP}}(\mathrm{SPARSE}) \cap \mathrm{P/poly}$.

We next answer the last question in [22] concerning E_r(SPARSE) classes, that is, we show that $\mathrm{E}_{\mathrm{tt}}^{\mathrm{P}}(\mathrm{SPARSE}) \subsetneqq \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE})$. The separation does not seem to follow directly from the sets in Section 3, but we can use the same technique here. To distinguish between $\leq_{\mathrm{T}}^{\mathrm{P}}$ and $\leq_{\mathrm{tt}}^{\mathrm{P}}$, we exploit the adaptiveness of $\leq_{\mathrm{T}}^{\mathrm{P}}$ reducibility, namely, its ability to do prefix search.

THEOREM 4.9. *There is a set D in* $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE}) - \mathrm{E}_{\mathrm{tt}}^{\mathrm{P}}(\mathrm{SPARSE})$.

Proof. We construct D that meets the following two requirements:

(I) For some sparse set S_D, $D \leq_{\mathrm{T}}^{\mathrm{P}} S_D$ and $S_D \leq_{\mathrm{T}}^{\mathrm{P}} D$.

(II) For every set X, if $D \leq_{\mathrm{tt}}^{\mathrm{P}} X \leq_{\mathrm{tt}}^{\mathrm{P}} D$, then X is not sparse.

Let $\{M_i\}_{i\geq 1}$ denote an enumeration of polynomial-time deterministic tt-reductions, where the running time of each M_i is bounded by p_i. The construction proceeds in stages, and at each stage $k = \langle i, j, l\rangle_{\mathrm{N}}$, we define $D^{(k)}$ and $S^{(k)}$ that meet the same requirements as in the proof of Theorem 3.1.

Sets $D^{(k)}$ and $S^{(k)}$ at stage $k = \langle i, j, l\rangle_{\mathrm{N}}$ are defined as follows.

(1) Execute steps (1), (2), and (3) as in Theorem 3.1, defining τ, W_1, W_2, L_1, and L_2.

(2) Define also the following sets:

$$
\begin{aligned}
&Pref(\tau) = \{\ \langle 0^n, u\rangle\ :\ u \text{ is a prefix of } \tau\ \},\\
&S_1 = Pref(\tau) \oplus \{\ 0\tau\ \}, \quad\text{and}\quad S_2 = Pref(\tau) \oplus \{\ 1\tau\ \}\\
&(\text{where } X \oplus Y \text{ is the set } 0X \cup 1Y).
\end{aligned}
$$

Intuitively, S_1 and S_2 contain all the information about τ, plus an indication of whether W_1 or W_2 is used. Moreover, we will see that this information can only be accessed if some kind of prefix search can be used.

(3) Define $D_1 = D^{(k-1)} \cup (L_1 \oplus S_1)$ and $D_2 = D^{(k-1)} \cup (L_2 \oplus S_2)$. Then, set $D^{(k)}$ and $S^{(k)}$ as in step (4) of Theorem 3.1, using D_1, D_2, S_1, and S_2 in place of A_1, A_2, W_1, and W_2.

Finally define $D = \cup_{k\geq 1} D^{(k)}$ and $S_D = \cup_{k\geq 1} S^{(k)}$.

Notice that D is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to S_D by the following reduction: On inputs of the form $1x$, accept if and only if $x \in S_D$. On an input of the form $0x$, with $|x| = n \cdot m$ for some n used in the construction, find the word τ (of length $2n^2$) used in that stage by prefix search on S_D. Compute the sets W_1 and W_2 from τ. Then, if $10\tau \in S_D$, accept if and only if all blocks of x are in W_1; otherwise accept if and only if all blocks of x are in W_2.

Since D is of the form $L \oplus S_D$ for some set L, it is immediate that $S_D \leq^{\mathrm{P}}_{\mathrm{T}} D$.

Now to finish the proof of the theorem, we must only show that either (Case a) or (Case b) holds at every stage. We do this in the next lemmas. □

Consider again a fixed stage k and all the definitions made at that stage. Of course, Lemma 3.2 is also valid in this context. Compose machines M_i and M_j to give a new oracle machine M_s, which will also be a tt-machine whose running time is bounded by a polynomial p_s.

The next Lemma is the analog of Lemma 3.3.

LEMMA 4.10.

(i) For all strings x and y in $L_1 \cup L_2$, if $M_s(0x)$ queries $0y$, then $x = y$.

(ii) For all strings x in $L_1 \cup L_2$, $M_s(0x)$ queries neither 110τ nor 111τ.

Proof. For part (i), suppose that $x \in L_1 \cup L_2$ and that $0y$ is the ℓth word in the list of queries made by $M_s(0x)$ such that $y \in L_1 \cup L_2$ and $y \neq x$. Notice that we can write simply $M_s(0x)$ because this list is independent of the oracle used by M_s.

Define d_3 as the description of a program that, being given $\overline{\ell} \cdot x$ as an input, simulates $M_s(0x)$ and prints its ℓth query minus the first symbol, which is y. Then, on input $\overline{d_3}\cdot(\overline{\ell}\cdot x)$ $(= (d_3\#\ell)\cdot x)$, our universal machine prints a word in $L_1 \cup L_2$ other than x. By the same argument as in Lemma 3.3, the choice of c such that $|d_3\#\ell| \leq c \log n + c$ leads to the contradiction.

Part (ii) is proved similarly by using the following argument: if $M_s(0x)$ produces 110τ or 111τ, then many words in $L_1 \cup L_2$ can be described too easily from x plus a short program. □

LEMMA 4.11. *If (Case a) fails, then (Case b) holds.*

Proof. Suppose that (Case a) fails, and let X_1 and X_2 be two sets such that $D_1 \leq^{\mathrm{P}}_{\mathrm{tt}} X_1 \leq^{\mathrm{P}}_{\mathrm{tt}} D_1$ and $D_2 \leq^{\mathrm{P}}_{\mathrm{tt}} X_2 \leq^{\mathrm{P}}_{\mathrm{tt}} D_2$ via M_i and M_j. We will show that at least one of $X_1^{\leq p_i(nm)+1}$ or $X_2^{\leq p_i(nm)+1}$ is not p_l-sparse.

Define function C as follows. For any $x \in L_1 \cup L_2$, $C(x)$ is the first query in the list produced by $M_i(0x)$ that has different answer from oracles X_1 and X_2.

As before, it is easy to show that the function C is defined on $L_1 \cup L_2$. We also make the following claim.

CLAIM. The function C is one-to-one on $L_1 \cup L_2$.

Proof of the Claim. Take any $x \in L_1 \cup L_2$ and name $y = C(x)$.

We show first that $M_j(y)$ must query $0x$. This is because D_1 and D_2 differ only in the words of $0L_1$ and $0L_2$ and in the two words 110τ and 111τ. But by the previous lemma, $M_j(y)$ cannot query any of these words except $0x$. Therefore, if $M_j(y)$ does not query $0x$, it gets the same list of answers from oracles D_1 and D_2. But then y is in both X_1 and X_2 or in none of them. This contradicts the definition of $y = C(x)$.

To prove that C is one-to-one, assume for a moment that also $C(x') = y$ and hence $M_i(0x')$ queries y, for some $x' \in L_1 \cup L_2$ that is not x. We have now that $M_s(0x')$ queries $0x$ when simulating $M_j(y)$. But this is impossible by the previous lemma. □ *Claim*

Following the same argument as in Lemma 3.4, we can finish the proof. □

Notice now that the sparse set S_D is defined, essentially, by adding at every stage the set of prefixes of a single word. It is easy to see that there is a deterministic polynomial-time machine that, relative to S_D, prints all the elements of S_D up to a given length n. This means that S_D is in PF-SELF$_{\mathrm{gen}}$ ($= \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{TALLY})$) and, because $\leq_{\mathrm{T}}^{\mathrm{P}}$ is transitive, so is D. Thus we have the following corollary.

COROLLARY 4.12. $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{TALLY}) \not\subseteq \mathrm{E}_{\mathrm{tt}}^{\mathrm{P}}(\mathrm{SPARSE})$.

The last questions we address deal with the relations between classes of sets reducible to sparse sets under various reducibilities r, $\mathrm{R}_r(\mathrm{SPARSE})$. The study of these classes was initiated by Book and Ko [8] and continued later by Ko [15], who focused on conjunctive and disjunctive reducibilities. In [15], four conjectures concerning these classes were formulated. The first one was refuted by Allender, Hemachandra, Ogiwara, and Watanabe:

THEOREM 4.13 ([2]). $\mathrm{R}_{\mathrm{btt}}^{\mathrm{P}}(\mathrm{SPARSE}) \subseteq \mathrm{R}_{\mathrm{d}}^{\mathrm{P}}(\mathrm{SPARSE})$.

Very recently, the second of the conjectures has been refuted by Buhrman, Longpré, and Spaan:

THEOREM 4.14 ([9]). $\mathrm{R}_{\mathrm{bdtt}}^{\mathrm{P}}(\mathrm{SPARSE}) \subseteq \mathrm{R}_{\mathrm{c}}^{\mathrm{P}}(\mathrm{SPARSE})$.

By using the constructions in Section 3, we prove the remaining two of Ko's conjectures. The first one states that disjunctive reducibility does not have all the power of conjunctive reducibility when applied to sparse sets.

THEOREM 4.15. $\mathrm{R}_{\mathrm{c}}^{\mathrm{P}}(\mathrm{SPARSE}) \not\subseteq \mathrm{R}_{\mathrm{d}}^{\mathrm{P}}(\mathrm{SPARSE})$.

Proof. Build a set A as in Theorem 3.1, but using function $t(n) = 2^{3n}$ instead of $t(n) = 2^n$. We only use the following properties of A:

(i) A is $\leq_{\mathrm{c}}^{\mathrm{P}}$-reducible to some sparse set,
(ii) A is not sparse, and
(iii) for every $c \geq 0$, for all sufficiently large n, and for every two different words x and y in $A^{=n}$, $y \notin \mathrm{KT}[c \log n + c, t(n) \mid x]$.

Part (iii) follows by an easy extension of Lemma 3.2. Note also that the construction in Theorem 3.1 can be made much simpler if we are only interested in obtaining these properties.

Now assume that A is $\leq_{\mathrm{d}}^{\mathrm{P}}$-reducible to some sparse set S via a function f and we will derive a contradiction. For every x, we view $f(x)$ as a set so that x is in A if and only if $f(x) \cap S$ is not empty. Assume that f is computable in time $p_i(n)$ and that S has at most $p_j(n)$ elements of length at most n, for two polynomials p_i and p_j.

Since A is not sparse, there are infinitely many n such that $A^{=n}$ has more than $p_j(p_i(n))$ elements. For any such n, there are two strings x and y in $A^{=n}$ such that $x > y$ and $S \cap f(x) \cap f(y) \neq \emptyset$; that is, x and y share a string witnessing that they are in $A^{=n}$.

Let $z \in S \cap f(x) \cap f(y)$ be the ℓth query in $f(x)$. Let d be a program that, on input $(n\#\ell) \cdot x$, computes the ℓth element in $f(x)$, which is z, and finds the smallest string u of length n such that $z \in f(u)$. Observe that u must be in $A^{=n}$ and that it must be different from x (by the assumption $x > y$).

The time used by d to do this task can be bounded by $p_k(n) \cdot 2^n$, for some polynomial p_k. Hence $U((d\#n\#\ell) \cdot x)$ prints u in time at most $|d|^2 \cdot (p_k(n) \cdot 2^n)^2$. If n is sufficiently large, this is less than $t(n)$. Furthermore, because f is computable in time bounded by p_i, there is a constant c independent of n such that $|n\#\ell| \leq c \log n + c$. Therefore, for these particular x, u, and c, $u \in \mathrm{KT}[c \log n + c, t(n) \mid x]$.

With the assumption $A \leq_{\mathrm{d}}^{\mathrm{P}} S$, this must happen for infinitely many n. But this contradicts (iii). $\square$

As in Theorem 4.6, this result holds not only for $\leq_{\mathrm{c}}^{\mathrm{P}}$ but also for $\leq_{m(n)\text{-c}}^{\mathrm{P}}$ reducibility, where $m(n)$ is any reasonable unbounded function. And Theorem 4.13 states precisely that it cannot be extended to bounded $m(n)$.

Note also that Buhrman, Longpré, and Spaan [9] have obtained an improvement of this theorem. They use an extension of our techniques to show that the set witnessing the separation can be made $\leq_{m(n)\text{-c}}^{\mathrm{P}}$-reducible to some *tally* set, so $\mathrm{R}_{m(n)\text{-c}}^{\mathrm{P}}(\mathrm{TALLY}) \not\subseteq \mathrm{R}_{\mathrm{d}}^{\mathrm{P}}(\mathrm{SPARSE})$.

The last of Ko's conjectures is an immediate corollary of Theorem 4.15.

COROLLARY 4.16. $\mathrm{R}_{\mathrm{d}}^{\mathrm{P}}(\mathrm{SPARSE}) \subsetneq \mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$.

5. Final Remarks. Two aspects of our main results are worth noting. The first one is that sets A and B in Section 3 are recursive and that we can give good bounds for their time complexity. The second one is that the constructions can be used for sparseness bounds much larger than polynomial.

Let us consider first the time complexity of set A in Theorem 3.1. A small modification of its construction will make A decidable in exponential time. The only hard steps in the construction are steps (3) and (4). Step (3) requires finding the string $\tau \notin \mathrm{KT}[2n^2 - 1, t(2n^2)]$, and this can be done by exhaustive search in time $2^{O(n^2)}$. Step (4), (Case a), can be detected by cycling through all words of length at most nm and using each one to simulate machines M_i and M_j; since M_i and M_j run in polynomial time, this simulation takes time less than 2^n if n is chosen sufficiently large with respect to p_i and p_j. Thus, the total time in this step is $2^{O(nm)} \cdot 2^n$, which is $2^{O(n^2)}$.

A similar exponential bound for step (4), (Case b), fails because the polynomial p_i can be arbitrarily large and thus the number of strings to test grows faster than any fixed exponential. However, for the rest of the proof it is enough to test only the strings that are actually queried in the reduction from A to $L(M_j, A)$. There are at most $p_i(nm) \cdot 2^{nm+1}$ such strings, which is less than 2^{n^2} for n sufficiently large. Thus, the total running time of this modified construction is exponential in n. It is an open question whether there exists some set satisfying Theorem 3.1 that is decidable in subexponential time.

A similar argument shows that set B in Theorem 3.8 can be built in time $2^{f(n)}$, where f is any function that majorizes every polynomial. Our technique requires a superexponential bound because we have to simulate arbitrary $\leq_{\mathrm{T}}^{\mathrm{SN}}$-reductions in a deterministic way.

As for the sparseness bound, we can use the construction in Theorem 3.1 to prove the following: Say that a function $f(n)$ is subexponential if for every $\epsilon > 0$ and for all but finitely many n, $f(n) < 2^{n^\epsilon}$. Then there is some $A \in \mathrm{P/poly}$ that is $\leq_{\mathrm{T}}^{\mathrm{P}}$-equivalent to no set whose census is bounded by a subexponential function.

Indeed, take the function $m(n)$ in the proof of Theorem 3.1 to be $\sqrt{n}$ and modify the construction of A as follows:

Step (4), (Case b):
If $\|L(M_j; A_1)^{\leq p_i(nm)}\| > \|L(M_j; A_2)^{\leq p_i(nm)}\|$, then set $A^{(k)}$ to A_1 and $S^{(k)}$ to $S^{(k-1)} \cup W_1$; otherwise, set $A^{(k)}$ to A_2 and $S^{(k)}$ to $S^{(k-1)} \cup W_2$.

Informally, we choose the extension of A that maximizes the number of elements in $L(M_j, A)$ at that stage.

Suppose that set X is $\leq_{\mathrm{T}}^{\mathrm{P}}$-equivalent to the resulting set A, and let M_i be any $\leq_{\mathrm{T}}^{\mathrm{P}}$-reduction from A to X. Recall that polynomial p_i bounds the running time of M_i. With the argument in the proof of Lemma 3.4 we can show that

$$\|X^{\leq p_i(n\sqrt{n})}\| \geq \|A^{=n\sqrt{n}}\| = \binom{n}{\sqrt{n}}$$

for infinitely many n — otherwise at some stage of the construction of A we diagonalize against the reductions making A and X equivalent. Since $\binom{n}{\sqrt{n}}$ is $2^{\Omega(\sqrt{n})}$, it is easy to see that in these conditions

$$\|X^{\leq n}\| \geq 2^{n^\epsilon}$$

for some $\epsilon > 0$ depending on p_i and infinitely many n. In other words, the census of X is not bounded by any subexponential function.

Analogous bounds can be achieved for sets B and D in Theorems 3.8 and 4.9 respectively. Note that these bounds are essentially optimal: a standard padding argument shows that, for any set L and any $\epsilon > 0$, there is some set $\leq_{\mathrm{m}}^{\mathrm{P}}$-equivalent to L whose census is bounded by 2^{n^ϵ}.

6. Acknowledgements. We thank the two referees for their thorough reading and many helpful suggestions. We thank Harry Buhrman and Luc Longpré for providing us with their work [9]. The first author thanks José Balcázar for many discussions on this subject and Ronald Book for inviting him to UCSB.

REFERENCES

[1] E. ALLENDER AND L. HEMACHANDRA, *Lower bounds for the low hierarchy,* J. Assoc. Comput. Machin., 39 (1992), pp. 234–251.

[2] E. ALLENDER, L. HEMACHANDRA, M. OGIWARA, AND O. WATANABE, *Relating equivalence and reducibility to sparse sets,* SIAM J. Comput., 21 (1992), pp. 521–539.

[3] E. ALLENDER AND O. WATANABE, *Kolmogorov complexity and degrees of tally sets,* Inform. Comput., 86 (1990), pp. 160–178.

[4] D. ANGLUIN, *Learning regular sets from queries and counterexamples,* Inform. Comput., 75 (1987), pp. 87–106.

[5] D. ANGLUIN, *Queries and concept learning,* Machine Learning, 2 (1988), pp. 319–342.

[6] J. BALCÁZAR AND R. BOOK, *Sets with small generalized Kolmogorov complexity,* Acta Informatica, 23 (1986), pp. 679–688.

[7] J. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity I,* EATCS Monographs on Theoretical Computer Science 11, Springer-Verlag, 1988.

[8] R. BOOK AND K. KO, *On sets truth-table reducible to sparse sets,* SIAM J. Comput., 17 (1988), pp. 903–919.

[9] H. BUHRMAN, L. LONGPRÉ, AND E. SPAAN, *SPARSE reduces conjunctively to TALLY,* to be presented at the Structure in Complexity Theory 8th Annual IEEE Conference, San Diego (California), may 1993.

[10] J. GROLLMANN AND A. SELMAN, *Complexity measures for public-key cryptosystems,* SIAM J. Comput., 17 (1988), pp. 309–335.

[11] J. HARTMANIS, *Generalized Kolmogorov complexity and the structure of feasible computations,* in Proc. 24th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1983, pp. 439–445.

[12] J. HARTMANIS AND Y. YESHA, *Computation times of NP sets of different densities,* Theoret. Comput. Sci., 34 (1984), pp. 17–32.

[13] R. KARP AND R. LIPTON, *Some connections between nonuniform and uniform complexity classes,* in Proc. 12th ACM Symposium on Theory of Computing, ACM Press, 1980, pp. 302–309.

[14] K. KO, *Continuous optimization problems and a polynomial hierarchy of real functions,* J. Complexity, 1 (1985), pp. 210–231.

[15] K. KO, *Distinguishing conjunctive and disjunctive reducibilities by sparse sets,* Inform. Comput., 81, 1989, pp. 62–87.

[16] K. KO, *On adaptive versus nonadaptive bounded query machines,* Theoret. Comput. Sci., 82 (1991), pp. 51–69.

[17] R. LADNER, N. LYNCH, AND A. SELMAN, *A comparison of polynomial time reducibilities,* Theoret. Comput. Sci., 1 (1975), pp. 103–123.

[18] M. LI AND P. VITÁNYI, *Kolmogorov complexity and its applications,* in Handbook of Theoretical Computer Science, vol. A, J. van Leeuwen, ed., Elsevier, 1990, pp. 187–254.

[19] T. LONG, *Strong nondeterministic polynomial-time reducibilities,* Theoret. Comput. Sci., 21 (1982), pp. 1–25.

[20] N. PIPPENGER, *On simultaneous resource bounds,* in Proc. 20th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1979, pp. 307–311.

[21] U. SCHÖNING, *Complexity and Structure,* Lecture Notes in Computer Science 211, Springer-Verlag, 1986.

[22] S. TANG AND R. BOOK, *Separating polynomial-time Turing and truth-table reductions by tally*

sets, in Proc. 15th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 317, Springer-Verlag, 1988, pp. 591–599.

[23] S. Tang and R. Book, *Reducibilities on tally and sparse sets,* RAIRO Informatique Théorique et Applications, 25 (1991), pp. 293–302. Extended version of [22].

[24] O. Watanabe, *A formal study of learning via queries,* in Proc. 17th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 443, Springer-Verlag, 1990, pp. 139–152.