Computational complexity of small descriptions

Ricard Gavaldà
Osamu Watanabe

Report LSI–90–43

Computational Complexity of Small Descriptions*

Ricard GAVALDÀ

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

Osamu WATANABE

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

November 19th, 1990

ABSTRACT

For a set L that is polynomial time reducible (in short, $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible) to some sparse set, we investigate the computational complexity of such sparse sets relative to L. We construct sets A and B such that both of them are $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *some* sparse set, but A (resp., B) is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *no* sparse set in P^A (resp., $\mathrm{NP}^B \cap \text{co-NP}^B$); that is, the complexity of sparse sets to which A (resp., B) is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible is more than P^A (resp., $\mathrm{NP}^B \cap \text{co-NP}^B$). Some consequences of these results and applications of our proof technique are also discussed.

1. Introduction

In computational complexity theory or, in more 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.

Relative complexity of small descriptions is asked in several contexts. Let us take for example "concept learning" studied in, e.g., [An87, An88]. Roughly speaking, concept

*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 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).

learning is to obtain a (small) description of an unknown set X. In particular, "query learning", i.e., the type of learning discussed in [An87, An88], achieves this task by asking queries on X to a teacher; thus, one can formalize "query learning" as one type of oracle computation, where an unknown set is used as an oracle [Wa90]. 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. $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$[1] is the class of sets that are polynomial time reducible (i.e., $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible) to some sparse set. Recall that a *sparse set* is a set S such that for some polynomial p and for all $n \geq 0$, S has at most $p(n)$ elements of length $\leq p(n)$. Hence, one can encode a set S up to a given length n in a string of polynomial length. Thus, $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$ sets, i.e., sets in $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$, are often regarded as sets with a small description. There is another interpretation to sets in $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$. It is well-known (see, e.g., [BDG88]) that a set belongs to $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$ if and only if it has polynomial size circuits. Since a family of polynomial size circuits is considered as a small description, $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$ sets are, in this sense also, sets with a small description. Therefore, an $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$ set L is regarded as a set with a small description, and a sparse set S to which L is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible is called a *sparse set description* for L.

Let L be any $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$ set. 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, which is a nonrecursive $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$ set because any sparse set is by definition in $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$.) Then all sparse set descriptions for such L should be nonrecursive. Thus, for such L, it does not make sense to discuss the "absolute" complexity of sparse set descriptions. Here, we consider the "relative" complexity; that is, we investigate the complexity of sparse set descriptions for L relative to L. Secondly 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 $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$.

In order to state our main results, let us introduce some notion and notation. For any complexity class C and any set $L \in \mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$, we say that L has a *C-self-recognizable sparse set description* if L is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to some S that is in C^L. Define C-SELF to be

[1]This class was denoted as $\mathrm{P}_{\mathrm{T}}(\mathrm{SPARSE})$ in [BK88, TB88]. We are following more recent notation [AHOW90].

the class of R_T(SPARSE) sets with a $\mathcal{C}$-self-recognizable sparse 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 R_T(SPARSE) sets. By extending [Sc86; Lemma 5.6], one can easily show that every set L in R_T(SPARSE) has a sparse set description in $\Delta_3^{P,L}$ $(= P^{NP^{NP^L}})$. In other words, R_T(SPARSE) $\subseteq \Delta_3^P$-SELF. (Indeed, R_T(SPARSE) $= \Delta_3^P$-SELF since by definition we have Δ_3^P-SELF $\subseteq R_T$(SPARSE).) However, we do not know whether this is the optimal upper bound for some R_T(SPARSE) set. In fact, no nontrivial lower bound has been known for any R_T(SPARSE) set; for example, it has been asked [TB91] whether there exists an R_T(SPARSE) set that is not in P-SELF. In our main results, we construct such a set A, i.e., $A \in R_T$(SPARSE) $-$ P-SELF, and also construct a set B in R_T(SPARSE) $-$ (NP $\cap$ co-NP)-SELF. Furthermore, it is shown that $A \in$ (NP $\cap$ co-NP)-SELF and $B \in$ NP-SELF. Thus, our results yield a $\mathcal{C}$-SELF hierarchy in R_T(SPARSE); that is, P-SELF $\subsetneq$ (NP $\cap$ co-NP)-SELF $\subsetneq$ NP-SELF $\subseteq \Delta_3^P$-SELF $(=$ R_T(SPARSE)).

Note that a sparse set S has a trivial sparse set description, namely S itself, that is in P^S. Thus, a set in R_T(SPARSE) $-$ P-SELF must be the one that is $\leq_T^P$-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, $L \leq_T^P S$, and $S \notin P^L$, this does not necessarily mean that $L \in R_T$(SPARSE) $-$ P-SELF; L might have another sparse set description in P^L. That is, a set L in R_T(SPARSE) $-$ P-SELF must be such that *every* sparse set description for L does not belong 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 R_T(SPARSE) if and only if it has polynomial size circuits; furthermore, there is a close relation between sparse set descriptions and polynomial size circuits. Thus, one may expect that the complexity of sparse set descriptions has something to do with that of finding polynomial size circuits. The answer is "yes and no": both the complexity of sparse set descriptions that of finding polynomial size circuits are closely related, but not exactly the same. (The difference is mainly due to the difference between "recognition" and "generation"; see Section 4.1 for more explanation.) Thus, 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 [TB88] proposed investigating $\mathrm{E}_r(\mathrm{TALLY})$ classes and $\mathrm{E}_r(\mathrm{SPARSE})$ classes, 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 [TB88] and later in [AW90], many important questions concerning $\mathrm{E}_r(\mathrm{SPARSE})$ classes have been left open. Here we solve all such questions. It has been asked [TB91] whether P/poly has a set not in $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE})$, which is the same as asking whether P-SELF $\subsetneq$ $\mathrm{R}_{\mathrm{T}}(\mathrm{SPARSE})$; thus, the first main result answers to this question affirmatively. We also prove various separations between $\mathrm{E}_r(\mathrm{SPARSE})$ classes that have been asked in [TB88]. From our main results, we immediately obtain $\mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE}) \subsetneq \mathrm{E}_{\mathrm{T}}^{\mathrm{SN}}(\mathrm{SPARSE}) \subsetneq \mathrm{E}_{\mathrm{T}}^{\mathrm{NP}}(\mathrm{SPARSE})$. By applying the proof technique we used to obtain our main results, we show that $\mathrm{E}_{\mathrm{tt}}^{\mathrm{P}}(\mathrm{SPARSE}) \subsetneq \mathrm{E}_{\mathrm{T}}^{\mathrm{P}}(\mathrm{SPARSE})$.

2. Preliminaries

In this paper we follow standard definitions and notations in computational complexity theory (see, e.g., [BDG88]).

We use the alphabet $\Sigma = \{0, 1\}$. By a *string* 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 $X^{\leq n}$ to denote the complement of X, i.e., $\Sigma^* - X$, and the set $\{x \in X : |x| \leq n\}$ 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 $\mathcal{D}$. In this paper, 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 $\langle \cdot, \cdot \rangle_{\mathbf{N}}$ denote a pairing function from $\mathbf{N} \times \mathbf{N}$ *onto* $\mathbf{N}$; for example, we can define $\langle n, m \rangle_{\mathbf{N}} = (n+m)(n+m+1)/2+n$. Let $\langle \cdot, \cdot \rangle_{\mathrm{T}}$ denote a pairing function from $0^* \times 0^*$ *onto* 0^*, which can be defined by, e.g., $\langle 0^n, 0^m \rangle_{\mathrm{T}} = 0^{\langle n,m \rangle_{\mathbf{N}}}$. We generalize these pairing functions to any k-tupling function by applying them $k-1$ times; e.g., $\langle x, y, z \rangle = \langle x, \langle y, z \rangle \rangle$.

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, ..., x_n, we use $x_1 \# \cdots \# x_n$ to denote the string

$\overline{x_1} \cdot \cdots \cdot \overline{x_n}$. With our convention, we have that $|x_1 \# \cdots \# x_n| \leq |x_1| + \cdots + |x_n| + 2n(\lceil \log(\max\{|x_1|, ..., |x_n|\}) \rceil + 1)$. The quantity $2n(\lceil \log(\max\{|x_1|, ..., |x_n|\}) \rceil + 1)$ will be referred as a *tupling factor*.

Our computation model is the standard multi-tape (oracle) Turing machine. A machine is either deterministic or nondeterministic; a deterministic machine is either an acceptor or a transducer while a nondeterministic one is always an acceptor. 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.

In this paper, we use several polynomial time reducibilities. For (deterministic) polynomial time reducibilities, we consider $\leq^{\mathrm{P}}_{f(n)\text{-tt}}$, $\leq^{\mathrm{P}}_{\mathrm{tt}}$ and $\leq^{\mathrm{P}}_{\mathrm{T}}$. (See [LLS75] for the definition of $\leq^{\mathrm{P}}_{\mathrm{tt}}$ and $\leq^{\mathrm{P}}_{\mathrm{T}}$.) Note that $\leq^{\mathrm{P}}_{f(n)\text{-tt}}$ is a special case of $\leq^{\mathrm{P}}_{\mathrm{tt}}$-reducibility, which is defined as follows: for any function $f(n)$, $X \leq^{\mathrm{P}}_{f(n)\text{-tt}} Y$ if X is reducible to Y via a $\leq^{\mathrm{P}}_{\mathrm{tt}}$-reduction whose truth-table size is bounded by $f(n)$. We also consider reducibilities that are achieved by polynomial time nondeterministic computation: $\leq^{\mathrm{SN}}_{\mathrm{T}}$ and $\leq^{\mathrm{NP}}_{\mathrm{T}}$. For any set A and B, $A \leq^{\mathrm{SN}}_{\mathrm{T}} B$ (resp., $A \leq^{\mathrm{NP}}_{\mathrm{T}} B$) if A is in $\mathrm{NP}^B \cap \text{co-NP}^B$ (resp., NP^B).

In our discussions, we use the notion of "resource bounded Kolmogorov complexity". (See [LV88] for the 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 transducer that expects as input a pair of a transducer T and an input string x to it, and simulates T on 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 $|d| \cdot t^2$ steps.

The existence of such universal machine is shown, e.g., in [Ha83].

Now we define the following sets. For any positive integers n and m, define $\mathrm{KT}[n,m]$ to be the set of strings that can be generated from an input of length $\leq n$ within m steps by our universal Turing transducer U. For any string x, $\mathrm{KT}[n,m \mid x]$ is defined similarly; i.e., it is the set of strings that can be generated by U from an input $d \cdot x$, where $|d| \leq n$, within m steps. Notice that for each n, m, and x, both $\mathrm{KT}[n,m]$ and $\mathrm{KT}[n,m \mid x]$ contain only finite number of elements.

3. Main Results

In this section we will construct 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 will show that A is in (NP $\cap$ co-NP)-SELF and B is in NP-SELF. In other words, A is a set $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *some* sparse set in $\mathrm{NP}^A \cap \text{co-NP}^A$ but $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *no* sparse set in P^A, and B is a set $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *some* sparse set in NP^B but $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to *no* sparse set in $\mathrm{NP}^B \cap \text{co-NP}^B$. Therefore, we have that P-SELF $\subsetneq$ (NP $\cap$ co-NP)-SELF $\subsetneq$ 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 set A that meets the following two requirements:

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

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

Note that the second requirement is equivalent to the following: for every set X, if $A \leq^{\mathrm{P}}_{\mathrm{T}} X \leq^{\mathrm{P}}_{\mathrm{T}} A$, 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 some Kolmogorov complex strings. (We can define such A so that A is $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to some sparse set S_A.) Then every string in A is hard to describe relative to any other string in A. Using this property, we are able to show that every $\leq^{\mathrm{P}}_{\mathrm{T}}$-reduction from A to A must very similar to the identity function (in a way to be made precise later). Now consider any set X that is $\leq^{\mathrm{P}}_{\mathrm{T}}$-equivalent to A, i.e., $A \leq^{\mathrm{P}}_{\mathrm{T}} X \leq^{\mathrm{P}}_{\mathrm{T}} A$. Then the composition of the $\leq^{\mathrm{P}}_{\mathrm{T}}$-reduction from A to X and the one from X to A is regarded as a $\leq^{\mathrm{P}}_{\mathrm{T}}$-reduction from A to A; thus, such a reduction is very similar to the identity function. Hence, X must have distinguished information about almost all strings in A (which is non-sparse) and thus cannot be sparse.

Now we define the set A precisely. Recall that $\{p_l\}_{l\geq 1}$ denotes an enumeration of polynomials. In the following, let $\{M_i\}_{i\geq 1}$ denote an enumeration of polynomial time deterministic Turing reductions. We can assume that each M_i is p_i time bounded. Also, let us say that a set X is p_l-sparse iff for each n, $\|X^{\leq n}\| \leq p_l(n)$. In the construction, we will 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 such that $m(n) \cdot \log n = o(n)$.

We build our set A in stages. (Recall that the function $\lambda i,j,l.\langle i,j,l\rangle_{\mathbf{N}}$ is a one-to-one and onto function from $\mathbf{N}\times\mathbf{N}\times\mathbf{N}$ to $\mathbf{N}$.) At each stage $k = \langle i,j,l\rangle_{\mathbf{N}}$, 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, the above conditions (a) and (b) 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 $k = \langle i, j, l\rangle_{\mathbf{N}}$.

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

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

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

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

(v) n is sufficiently large so that no string of length $\geq n$ is used or queried before 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.

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

Note that L_1 and L_2 are disjoint, and that 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 each 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) $L(M_i; L(M_j; A_1)) \neq A_1$, or (ii) $L(M_i; L(M_j; A_2)) \neq A_2$ 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) $L(M_j; A_1)$ is not p_l-sparse, or (ii) $L(M_j; A_2)$ 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 above construction 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 above construction 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 as were 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 of y that is not a block of 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 $l_0 \# l_1 \# \ldots \# l_m$ 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 τ.

Finally, let d_0 be the description of some program such that U generates τ from $\overline{d_0} \cdot (d_1 \# x \# l \# (\tau - x - w))$. Intuitively, d_0, on input $d_1 \# x \# l \# (\tau - x - w)$, 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 only on d_0. Hence, U with $\overline{d_0} \cdot (d_1 \# x \# l \# (\tau - x - w))$ as an input halts within $|d_0| \cdot b^2 \cdot (t(n) + 2n^2)^2$ steps. Choose $c \geq |d_0| \cdot b^2$. Then, by condition (iii) on n this is less than $t(2n^2)$.

We now estimate the size of $\overline{d_0} \cdot (d_1 \# x \# l \# (\tau - x - w))$. (Recall that $\overline{d_0} \cdot (d_1 \# x \# l \# (\tau - x - w)) = d_0 \# d_1 \# x \# l \# (\tau - x - w)$.) Let e be $|d_0| + |d_1| + |x| + |l_0| + \cdots + |l_m| + |\tau - x - w|$. Adding up the size of each string, we have that $e < c + (c \log n + c) + mn + (m+1) \log n + (2n^2 - mn - n)$; hence, by condition (i) on n, $e < 2n^2 - n/2$. The tupling factor for $d_0 \# d_1 \# x \# l \tau - x - w$ 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. □

Here we define a new oracle machine M_s by composing machines M_i and M_j. On an input x ($|x| \leq n \cdot m$) and with an 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 iff M_i accepts x.

Recalling condition (v) on n, it is clear that there exists some polynomial p_s (depending only on i and j) that bounds the running time of M_s. Furthermore, if (Case a) does not hold, we have that $L(M_s; A_1) = A_1$ and $L(M_s; A_2) = A_2$. The next Lemma says that, to perform these reductions, M_s can expect very little help from oracle A_1 or 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 (i) $x \in L_1$, (ii) the first $l-1$ queries of $M_s^{A_1}(x)$ are either x or not in $L_1 \cup L_2$, but (iii) 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 $\overline{l} \cdot x$ as an input, simulates $M_s(x)$ with oracle $\{x\}$ and prints its lth query. Then, on input $\overline{d_2} \cdot (\overline{l} \cdot x)$ $(= \overline{d_2 \# l} \cdot x)$, 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. As $l \leq p_s(n)$, there is a constant c', depending only on i and j, such that $|l| \leq c' \cdot \log n$. Choosing $c \geq 3 \cdot (|d_2| + c')$, we have that $|d_2 \# l| \leq c \cdot \log n + c$. This contradicts Lemma 3.2.

If $x \in L_2$ instead, we just 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 or X_2 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 answer from oracles X_1 and X_2. Note that $C(x)$ is always defined on 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 a moment that also $C(x') = y$ and, for example, 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

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

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

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

(by condition (iv) on n).

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

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

We show that the sparse set S_A constructed in Theorem 3.1 is indeed 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 a *census function* of L: For each $n \geq 0$, $\mathrm{cens}_L(n) = \|L^{\leq n}\|$. Note that S_A has n elements at all the lengths n used in its construction. 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 $c = \mathrm{cens}_X(|u|)$.

(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 at Theorem 3.1 is in $(\mathrm{NP} \cap \text{co-NP})\text{-SELF} - \text{P-SELF}$.

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

A *three-state 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-state 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 [Lo82], where it is shown that

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 nondeterministic machine M that is strong under oracle Y and that recognizes L relative to Y, i.e., $X = L(M;Y)$.

Let us state the second main result, i.e., showing a set B in $\mathrm{R_T}(\mathrm{SPARSE})$ that does not belong to $(\mathrm{NP} \cap \text{co-NP})\text{-SELF}$.

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

Remark. For proving $B \notin (\mathrm{NP} \cap \text{co-NP})\text{-SELF}$, it suffices to show that for any sparse set S, if $B \leq_{\mathrm{T}}^{\mathrm{P}} S$, then $S \notin \mathrm{NP}^B \cap \text{co-NP}^B$. The above theorem 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. Indeed, our construction technique does not seem to yield a set L in $\mathrm{R_T}(\mathrm{SPARSE}) - \text{NP-SELF}$; that is, a set L constructed by our technique always has a sparse description that is NP^L-recognizable. Then from Proposition 3.5, if such a sparse description has an easy census function, its complexity goes down to $\mathrm{NP}^L \cap \text{co-NP}^L$. Thus, the key to construct the desired set B is to make B so that B is not reducible to any 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. 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, some extension B' exists such that $M_s^{B'}$ always has a path ending in a REJ state for every input. We choose one such path for every input thereby fixing a nondeterministic path of M_s. Then we can regard M_s (on such a path) as a deterministic machine; thus, using an argument similar to the previous proof, we can prove that the intermediate set X should be non-sparse.

To define the sets B and S_B, let $\{M_i\}_{i\geq 1}$ denote an enumeration of polynomial time three-state nondeterministic machines. We can assume that each M_i is p_i time bounded. 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 $k = \langle i, j, l\rangle_{\mathbf{N}}$, 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 $L(M_j; B^{(k)})$.

(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 $k = \langle i, j, l\rangle_{\mathbf{N}}$, 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, 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 $B^{(k-1)}$, (ii) M_i is not strong under $L(M_j; B^{(k-1)})$, or (iii) $L(M_i; L(M_j; B^{(k-1)})) \neq B^{(k-1)}$. Then set $B^{(k)}$ to $B^{(k-1)}$ and $S^{(k)}$ to $\emptyset$.

(Case b)

Either (i) $L(M_i; L(M_j; B_1)) \neq B_1$, or (ii) $L(M_i; L(M_j; B_2)) \neq B_2$. 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) $L(M_j; B_1)$ is not p_l-sparse, or (ii) $L(M_j; B_2)$ 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 above construction 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 first computation path of $M_s(x)$ with oracle $\emptyset$ 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 in time 2^{n^d} for some constant d that depends only on M_i and M_j. The following properties of r_x are 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. Assume that (Case a) is false; this means that M_i and M_j are strong under $L(M_j; B^{(k-1)})$ and $B^{(k-1)}$ respectively.

Suppose that no path of $M_s^{\emptyset}(x)$ ends in state REJ. Then since M_i is strong under $L(M_j; B^{(k-1)})$, some path of $M_s^{\emptyset}(x)$ ends in ACC. That is, M_i accepts x with oracle $L(M_j; B^{(k-1)})$. But this means that $L(M_i; L(M_j; B^{(k-1)})) \neq B^{(k-1)}$ because $x \notin B^{(k-1)}$. A contradiction. □

The next lemma corresponds precisely 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 $B_1 \leq_T^{SN} X_1 \leq_T^{SN} B_1$ and $B_2 \leq_T^{SN} X_2 \leq_T^{SN} B_2$ via M_i and M_j. We will show that at least one of X_1 or X_2 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.

Using repeatedly Lemma 3.10, one can prove that

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

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

The proofs are very similar to the ones in Lemma 3.4, considering paths r_x and t_x to be the only relevant paths of $M_s(x)$ and $M_i(x)$, respectively. Then, the same counting argument of Lemma 3.4 shows that at least one of $\|X_1^{\leq p_i(n)}\|$ and $\|X_2^{\leq p_i(n)}\|$ must be greater than $p_l(p_i(n))$. □

The separation of NP-SELF from (NP ∩ co-NP)-SELF follows easily.

Corollary 3.12. The set B defined at Theorem 3.8 is in NP-SELF − (NP∩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

In this section we explain concequences of our results and applications of our proof techniques to other but closely related topics.

4.1. Complexity of Finding Polynomial Size Circuits

We have investigated the relative complexity of sparse set descriptions. Roughly speaking, such investigation can be regarded as the study of the "complexity" of finding (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 $L \leq_T^P S$ is not exactly representing the complexity of finding circuits for L. Here we will first make such difference clear and then discuss the consequences of the results in the previous section. We 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 above definition does not guarantee that a set L with polynomial size circuits should be 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 [KL80] 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 [Pi79]. The reader will find a good and complete proof in [BDG88].) Thus, now 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

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

Then the class P/poly is defined as follows.

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

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 each 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, ..., d_n on its input gates.

Now we are ready for discussing the "complexity" of finding circuits formally. For any $L \in$ 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 $L \in$ P/poly has a generator g that is polynomial time computable relative to L, i.e., $g \in \text{PF}^L$. 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 $H \in$ P/poly such that no circuit generator for H is in PF^{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 $L \in$ 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 $\mathcal{CF}$-SELF$_{\rm gen}$ is defined to be the class of sets with $\mathcal{CF}$-self generable circuits. Ko [Ko85] 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., [BDG88]) that P/poly = $\mathrm{R_T}$(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 $\mathcal{CF}$-SELF$_{\rm gen}$ are not the same in general. This difference is mainly due to the difference 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 [HY84] 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 [AW90].) The best known relation is that every sparse set in P is $\mathrm{PF}^{\mathrm{NP}}$-printable. Here we have similar relationship.

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

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

The relationship between $\mathcal{C}$-SELF, $\mathcal{C}$-TALLY-SELF, and $\mathcal{CF}$-SELF$_{\rm gen}$ is summarized as follows. (The following relations are either immediate by definition or proved easily by applying well-known techniques; thus, we omit their proof.)

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) $\mathcal{CF}$-SELF$_{\rm gen} \subseteq \mathcal{C}$-TALLY-SELF $\subseteq \mathcal{C}$-SELF.

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

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

(4) If $\mathcal{C}$ is closed under application of polynomial existential quantifiers, then $\mathcal{C}$-SELF = $\mathcal{C}$-TALLY-SELF.

Remark.

(1) By using more fine structure below $\mathrm{PF}^{\mathcal{C}}$, we can state the following closer relation: ($\mathcal{C}$/bit)-SELF$_{\rm gen}$ = ($\mathcal{C} \cap$ co-$\mathcal{C}$)-TALLY-SELF, where $\mathcal{C}$/bit is a subclass of $\mathrm{PF}^{\mathcal{C}}$ intro-

duced in [WT90]. Intuitively, $\mathcal{CF}$/bit is the class of functions f such that each bit of f is computable in $\mathcal{C}$. Formally, it is defined as follows.

$$\mathcal{C}/\text{bit} = \{\ f : \textit{Bit-}f \text{ is in } \mathcal{C}\ \},\ \text{where}$$
$$\textit{Bit-}f = \{\ \langle x, 0^i, d\rangle : \text{the } i\text{th bit of } f(x) \text{ is } d\ \} \cup \{\ \langle x, \epsilon, l\rangle : |f(x)| = l\ \}.$$

(2) It follows from (2) and (3) that $\mathcal{C}$-SELF $\subseteq$ PF$^{\text{NP}^{\mathcal{C}}}$-SELF$_{\text{gen}}$.

In particular, we have the following relations.

Corollary 4.2.
(1) PF-SELF$_{\text{gen}}$ = P-TALLY-SELF.
(2) PF$^{\text{NP}}$-SELF$_{\text{gen}}$ = Δ_2^{P}-TALLY-SELF = Δ_2^{P}-SELF.
(3) PF$^{\Sigma_2^{\text{P}}}$-SELF$_{\text{gen}}$ = Δ_3^{P}-TALLY-SELF = Δ_3^{P}-SELF. Thus, P/poly = PF$^{\Sigma_2^{\text{P}}}$-SELF$_{\text{gen}}$.

Notice that PF$^{\Sigma_2^{\text{P}}}$ is a general upper bound for the complexity of finding circuits, which is essentially the same as the one for the complexity of a sparse set description.

Now let us discuss a lower bound for the complexity of finding circuits and the consequence of our main results. Recall that recognizing a sparse 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$_{\text{gen}}$.

Proof. Note that P/poly = R$_{\text{T}}$(SPARSE) and that PF-SELF$_{\text{gen}}$ $\subseteq$ P-SELF. Thus, the set A defined at Theorem 3.1 clearly satisfies the theorem. □

We should note here that the same lower bound is obtained from observations in [BB86] and [AH89]. Balcázar and Book [BB86] showed that if L is in PF-SELF$_{\text{gen}}$ (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 [AH89] 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 above theorem because $S_0 \in$ P-SELF $-$ PF-SELF$_{\text{gen}}$ while A does so because $A \in$ P/poly $-$ P-SELF.

Next we improve a lower bound by using Theorem 3.8. In order to state our result, we need one complexity class of functions — NPSV — which is often used in the literature (see, e.g., [GS88]).

NPSV is the class of functions computed by polynomial time nondeterministic single valued transducers. What follows is a precise definition of NPSV (resp., NPSV^L). In our computation model, we assume that a transducer yields an output (and halts) whenever it enters an accepting state; thus, a nondeterministic transducer may yield more than one output for a given input. A nondeterministic (oracle) transducer N is called *single-valued* (relative to L) if N (resp., N^L) on input x outputs the same value on every accepting computation. The class NPSV (resp., NPSV^L) is the class of functions f that is computed by some polynomial time nondeterministic single-valued (oracle) transducer (relative to L).

Now we state a new lower bound.

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

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

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

Proof. The proof is immediate from the following relations: (i) $(\mathrm{NP/bit})\text{-}\mathrm{SELF}_{\mathrm{gen}} = (\mathrm{NP} \cap \text{co-NP})\text{-TALLY-SELF}$ (Proposition 4.1: Remark (1)), and (ii) NPSV = NP/bit [WT90]. But here we give a direct proof.

First we prove $\mathrm{NPSV\text{-}SELF}_{\mathrm{gen}} \subseteq (\mathrm{NP} \cap \text{co-NP})\text{-TALLY-SELF}$. Let L be any set in $\mathrm{NPSV\text{-}SELF}_{\mathrm{gen}}$. Then L has a circuit generator $g \in \mathrm{NPSV}^L$. Here define a tally set T by $T = \{\, \langle 0^n, 0^i, 0^d \rangle_{\mathrm{T}} : \text{the } i\text{th bit of } g(0^n) \text{ is } d \in \{0,1\} \,\}$. 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 any 0^n, N^L on 0^n has an accepting computation, and N^L outputs $g(0^n)$ on every such accepting computation. Once $g(0^n)$ is obtained, one can easily determine whether a given $\langle 0^n, 0^i, 0^d \rangle_{\mathrm{T}}$ 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 $(\mathrm{NP} \cap \text{co-NP})\text{-TALLY-SELF} \subseteq \mathrm{NPSV\text{-}SELF}_{\mathrm{gen}}$. Let L be any set in $(\mathrm{NP} \cap \text{co-NP})\text{-TALLY-SELF}$; 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 $> p_M(|x|)$.) Here 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 for completing 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 guessed all the σ_i correctly and that verified all the correctness. Clearly, N^L is single valued and computes g. □

4.2. Classes E_r(SPARSE)

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 $\leq^{\mathrm{P}}_{\mathrm{T}}$ reduction between L and its descriptions. 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 [TB88] and [AW90].

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 [TB91], this is not a proper name for non-transitive reduction types such as $\leq^{\mathrm{NP}}_{\mathrm{T}}$ and $\leq^{\mathrm{P}}_{f(n)\text{-tt}}$, because these reducibilities do not even define equivalence relations.

It follows directly from the definitions that P-SELF $= \mathrm{E}^{\mathrm{P}}_{\mathrm{T}}$(SPARSE); in particular, all sets in P-SELF are $\leq^{\mathrm{P}}_{\mathrm{T}}$-equivalent to its easiest sparse set description. Also, since $\leq^{\mathrm{P}}_{\mathrm{T}}$ implies $\leq^{\mathrm{SN}}_{\mathrm{T}}$ and $\leq^{\mathrm{NP}}_{\mathrm{T}}$ reducibilities, clearly (NP $\cap$ co-NP)-SELF $\subseteq \mathrm{E}^{\mathrm{SN}}_{\mathrm{T}}$(SPARSE) and NP-SELF $\subseteq \mathrm{E}^{\mathrm{NP}}_{\mathrm{T}}$(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 $= \mathrm{E}^{\mathrm{P}}_{\mathrm{T}}$(TALLY). Therefore, from Corollary 4.2 (1), we have that PF-SELF$_{\mathrm{gen}} = \mathrm{E}^{\mathrm{P}}_{\mathrm{T}}$(TALLY). This fact was already

in [BB86; Theorem 3.1], where it is shown that sets with self-producible circuits are exactly those $\leq^{\mathrm{P}}_{\mathrm{T}}$-equivalent to tally sets.

Concerning this class, Allender and Watanabe [AW90] propose the question of whether $\mathrm{E}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{TALLY}) = \mathrm{E}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SPARSE})$. This question arises from the fact that any $\leq^{\mathrm{P}}_{\mathrm{T}}$-reduction to a tally set can be rewritten as a $\leq^{\mathrm{P}}_{\mathrm{tt}}$-reduction to the same tally set. Then, when considering equivalence to a tally set, one might ask whether it is possible to trade the power of $\leq^{\mathrm{P}}_{\mathrm{T}}$ reduction for the access to a slightly more complex oracle, namely, a sparse set. This insight is even more tempting because it is known [BK88] that the reduction classes of the tally and the sparse sets are equal for reducibilities $\leq^{\mathrm{P}}_{\mathrm{T}}$ and $\leq^{\mathrm{P}}_{\mathrm{tt}}$ (and that all of them equal P/poly).

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 function $m(n)$ to give a set A that is not in P-SELF $= \mathrm{E}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SPARSE})$. The reader will easily verify that A is $\leq^{\mathrm{P}}_{m(n)\text{-tt}}$-reducible to the sparse set S_A given in that theorem. Therefore, we have

Theorem 4.6. For any unbounded function $m(n)$, $\mathrm{R}^{\mathrm{P}}_{m(n)-\mathrm{tt}}(\mathrm{SPARSE}) \not\subset \mathrm{E}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SPARSE})$.

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

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

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

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

We next answer the last question in [TB88] concerning $\mathrm{E}_r(\mathrm{SPARSE})$ classes, that is, we show that $\mathrm{E}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SPARSE}) \subsetneq \mathrm{E}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SPARSE})$. 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 $\leq^{\mathrm{P}}_{\mathrm{T}}$ and $\leq^{\mathrm{P}}_{\mathrm{tt}}$, we exploit the adaptiveness of $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducibility, namely, its ability to do prefix search.

Theorem 4.9. There is a set D in $\mathrm{E}^{\mathrm{P}}_{\mathrm{T}}(\mathrm{SPARSE})$ that is not in $\mathrm{E}^{\mathrm{P}}_{\mathrm{tt}}(\mathrm{SPARSE})$.

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

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

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

Let $\{M_i\}_{i\geq 1}$ denote an enumeration of polynomial time deterministic Turing tt-reductions, where each M_i is p_i time bounded. The construction proceeds in stages, and at each stage $k = \langle i,j,l\rangle_{\mathbf{N}}$, 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 $k = \langle i,j,l\rangle_{\mathbf{N}}$ are defined as follows.

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

(2) Define also the following sets:

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

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

(3) Define $D^{(k)}$ and $S^{(k)}$ as follows.

$D_1 = D^{(k-1)} \cup (L_1 \oplus S_1)$ and $D_2 = D^{(k-1)} \cup (L_2 \oplus S_2)$.

(Case a)

Either (i) $L(M_i; L(M_j; D_1)) \neq D_1$, or (ii) $L(M_i; L(M_j; D_2)) \neq D_2$ occurs. In this case, if (i) occurs, then set $D^{(k)}$ to D_1 and $S^{(k)}$ to S_1; otherwise, set $D^{(k)}$ to D_2 and $S^{(k)}$ to S_2.

(Case b)

Either (i) $L(M_j; D_1)$ is not p_l-sparse, or (ii) $L(M_j; D_2)$ is not p_l-sparse. In this case, if (i) occurs, then set $D^{(k)}$ to D_1 and $S^{(k)}$ to S_1; otherwise, set $D^{(k)}$ to D_2 and $S^{(k)}$ to S_2.

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

Notice that D is $\leq_{T}^{P}$-reducible to S_D by the following reduction: On inputs of the form $1x$, accept iff $x \in S_D$. 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 $1\langle 0^n, \tau 0\rangle \in S_D$, accept iff all blocks of x are in W_1; otherwise accept iff 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 $S_D \leq_{T}^{P} D$.

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 framework. 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 $1\langle 0^n, \tau 0\rangle$ nor $1\langle 0^n, \tau 1\rangle$.

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

Define d_3 as the description of a program that, being given $\overline{l} \cdot x$ as an input, simulates $M_s(x)$ and prints its lth query. Then, on input $\overline{d_3} \cdot (\overline{l} \cdot x)$ $(= \overline{d_3 \# l} \cdot x)$, 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 $|d_3 \# l| \leq c + c \cdot \log n$ leads to the contradiction.

Part (ii) is proved similarly by using the following argument: if $M_s(0x)$ produces $1\langle 0^n, \tau 0\rangle$ or $1\langle 0^n, \tau 1\rangle$, 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 $D_1 \leq_{tt}^{P} X_1 \leq_{tt}^{P} D_1$ and $D_2 \leq_{tt}^{P} X_2 \leq_{tt}^{P} D_2$ via M_i and M_j. We will show that at least one of X_1 or X_2 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 $1\langle 0^n, \tau 0\rangle$ and $1\langle 0^n, \tau 1\rangle$. 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 that $M_i(x')$ queries y, for some $x' \in L_1 \cup L_2$ that is not x. We have now that $M_s(x')$ queries x when simulating $M_j(y)$. But this is impossible by the previous lemma. □ Claim

Following the same argument 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. The interested reader can verify that there is a deterministic polynomial time machine such that, relative to S_D, prints all the elements of S_D up to a given n. This means that $S_D \in \text{PF-SELF}_{\text{gen}}$ $(= \text{E}_{\text{T}}^{\text{P}}(\text{TALLY}))$, and, because $\leq_{\text{T}}^{\text{P}}$ is transitive, so is D. Thus we have the following corollary.

Corollary 4.12.
(1) $\text{E}_{\text{T}}^{\text{P}}(\text{TALLY}) \not\subset \text{E}_{tt}^{\text{P}}(\text{SPARSE})$.
(2) $\text{E}_{tt}^{\text{P}}(\text{SPARSE}) \subsetneq \text{E}_{\text{T}}^{\text{P}}(\text{SPARSE})$.

References

[AH89] E. Allender and L. Hemachandra, Lower bounds for the low hierarchy, in "Proc. 16th International Colloquium on Automata, Languages and Programming", Lecture Notes in Computer Science 372 (1989), 31-45; the final version will appear in J. ACM.

[AHOW90] E. Allender, L. Hemachandra, M. Ogiwara, and O. Watanabe, Relating equivalence and reducibility to sparse sets, manuscript.

[AW90] E. Allender and O. Watanabe, Kolmogorov complexity and the degrees of tally sets, Information and Computation 86 (1990), 160-178.

[An87] D. Angluin, Learning regular sets from queries and counterexamples, Inform. and Comput. 75 (1987), 87-106.

[An88] D. Angluin, Queries and concept learning, Machine Learning 2 (1988), 319-342.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró, "Structural Complexity I", EATCS Monographs on Theoretical Computer Science, Springer-Verlag (1988).

[BB86] J. Balcázar and R. Book, Sets with small generalized Kolmogorov complexity, Acta Informatica 23 (1986), 679-688.

[BK88] R. Book and K. Ko, On sets truth-table reducible to sparse sets, SIAM J. Comput. 17 (1988), 903-919.

[GS88] J. Grollmann and A. Selman, Complexity measures for public-key cryptosystems, SIAM J. Comput. 17 (1988), 309-335.

[Ha83] J. Hartmanis, Generalized Kolmogorov complexity and the structure of feasible computations, in "Proc. 24th IEEE Sympos. Foundations of Computer Sciences", IEEE (1983), 439-445.

[HY84] J. Hartmanis and Y. Yesha, Computation times of NP sets of different densities, Theoret. Comput. Sci. 34 (1984), 17-32.

[KL80] R. Karp and R. Lipton, Some connections between nonuniform and uniform complexity classes, in "Proc. 12th ACM Sympos. Theory of Computing", ACM (1980), 302-309.

[Ko85] K. Ko, Continuous optimization problems and a polynomial hierarchy of real functions, J. Complexity 1 (1985), 210-231.

[LLS75] R. Ladner, N. Lynch and A. Selman, A comparison of polynomial time reducibilities, Theoret. Comput. Sci. 1 (1975), 103-123.

[LV88] M. Li and P. Vitányi, Two decades of applied Kolmogorov complexity: in memoriam Andrei Nikolaevich Kolmogorov 1903-1987, in "Proc. 3rd Structure in Complexity Theory Conference", IEEE (1988), 80-101; the final version appeared in "Complexity Theory Retrospective, A. Selman (ed.)", Springer-Verlag (1990), 147-203.

[Lo82] T. Long, Strong nondeterministic polynomial-time reducibilities, Theoret. Comput. Sci. 21 (1982), 1-25.

[Pi79] N. Pippenger, On simultaneous resource bounds, in "Proc. 20th IEEE Sympos. Foundation of Comput. Sci.", IEEE (1979), 307-311.

[Sc86] U. Schöning, "Complexity and structure", Lecture Notes in Computer Science 211 (1986).

[TB88] 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 (1988), 591-599; the final version [TB91], Reducibilities on tally and sparse sets, will appear in RAIRO.

[Wa90] 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 (1990), 139-152.

[WT90] O. Watanabe and S. Toda, Structural Analyses on the complexity of inverting functions, in "Proc. International Symposium SIGAL '90", Lecture Notes in Computer Science 450, Springer-Verlag (1990), 31-38.