Theoretical Computer Science 80 (1991) 203-225
Elsevier

On sets polynomially enumerable by iteration*

Lane A. Hemachandra**

Department of Computer Science, University of Rochester, Rochester, NY 14627, U.S.A.

Albrecht Hoene and Dirk Siefkes

Fachbereich Informatik, Technische Universität Berlin, D-1000 Berlin 10, Fed. Rep. Germany

Paul Young***

Department of Computer Science and Engineering, FR-35, University of Washington, Seattle, WA 98195, U.S.A.

Abstract

Hemachandra, L.A., A. Hoene, D. Siefkes and P. Young, On sets polynomially enumerable by iteration, Theoretical Computer Science 80 (1991) 203-225.

Sets whose members are enumerated by some Turing machine are called recursively enumerable. We define a set to be *polynomially enumerable by iteration* if its members are *efficiently* enumerated by iterated application of some Turing machine. We prove that many complex sets—including all exponential-time complete sets, all NP-complete sets yet obtained by direct construction, and the complements of all such sets—are polynomially enumerable by iteration. These results follow from more general results. In fact, we show that all recursively enumerable sets that are $\leq_{1-tt}^{P}$-*self-reducible* are polynomially enumerable by iteration, and that all recursive sets that are $\leq_{1-tt}^{P}$-*self-reducible* are *bi*-enumerable. We also show that when the $\leq_{1-tt}^{P}$-self-reduction is via a function whose inverse is computable in polynomial time, then the above results hold with the polynomial enumeration given by a function whose inverse is computable in polynomial time. In the final section of the paper we show that no NP-complete set can be iteratively enumerated in lexicographically increasing order unless the polynomial time hierarchy collapses to NP. We also show that the sets that are monotonically bi-enumerable are "essentially" the same as the sets in parity polynomial time.

1. Introduction to enumerability by iteration

Computational complexity theory has conventionally concentrated on the complexity of the *membership* problem of a particular set or class of sets. This paper

* This paper extends results of a preliminary version presented at the Fourteenth Symposium on Mathematical Foundations of Computer Science, Porąbka Kozubnik, Poland, 1989. This work was done in part while the first three authors visited Gerd Wechsung in Jena.

** Supported in part by a Hewlett-Packard Corporation equipment grant and the National Science Foundation under grant CCR-8809174/CCR-8996198 and a Presidential Young Investigator Award.

*** Supported in part by a Brittingham Visiting Professorship in the Computer Sciences Department, University of Wisconsin-Madison.

0304-3975/91/$03.50

follows an alternate approach; we study the complexity of *enumerating* the elements of possibly complex sets. This approach reflects a newly emergent theme in computational complexity theory: sets with high membership complexity do not perforce lack efficient algorithms for other fundamental operations in the set [14, 19].

In this paper, we show that many provably complex sets are enumerable by iteration in polynomial time. Other papers following the same paradigm show that complex sets often have efficient algorithms to perfectly hash [12], to near-test membership [11, 14], to pad [6, 36], to judge likelihood of set membership [45], and to implicitly check membership [24]. Efficient approximation algorithms can also be viewed as falling within this paradigm [13, 7].

This paper defines polynomial enumerability by iteration in analogy with a recursion-theoretic characterization of recursive enumerability. By definition, a set is recursively enumerable if there is a partial recursive function that "generates" or "enumerates" the set (see, e.g., [42, Chap. 5, 28, Section 7.7]). One way of formulating this is to do the enumeration iteratively [20, 51]. A set S is recursively enumerable if $S = \emptyset$ or if there is a Turing machine F (computing partial recursive function f) such that:

(1) if $x \in S$ then $F(x)$ halts, and

(2) there is an $x_0 \in S$ such that $S = \{x_0, f(x_0), f(f(x_0)), \ldots\}$.

By analogy we say that a set S is *polynomially enumerable by iteration* (*i-enumerable*) if there is a Turing machine F (computing function f) such that:

(1) if $x \in S$ then $F(x)$ halts in polynomial time (that is, in less than $p(|x|)$ steps), and

(2) there is an $x_0 \in S$ such that $S = \{x_0, f(x_0), f(f(x_0)), \ldots\}$.

This gives a uniform, efficient way to *iteratively* enumerate the elements of S.[1]

If one adds to the iterative characterization just given of the recursively enumerable sets the requirement that the iterating function be *total*, one obtains the recursion-theoretic notion of a *splinter*. That is, a set S is a splinter [51] (see also [37]) if:

> there is an initial element x_0 and a total recursive function f (the splinter function), such that $S = \{x_0, f(x_0), f(f(x_0)), \ldots\}$.

Splinters were widely used in classical recursive function theory twenty to thirty years ago, and they proved particularly helpful in analyzing reducibilities (see, e.g., [56–58]). What we shall see in this paper is the converse. In a polynomial setting, reducibilities, and in particular self-reducibilities, give powerful tools for showing that sets are iteratively enumerable in polynomial time.

Before proceeding, we should note that in the recursion-theoretic setting, splinters do *not* characterize the class of recursively enumerable sets. By requiring that the iterating, or enumerating, function be *total*, we obtain only a subset of the recursively

[1] A related but distinct research stream studies the complexity of generating single elements from complex sets. In particular, studies have been made of the complexity of generating random elements [31], of generating "solved" hard elements [1], and of finding test data [39, 43].

enumerable sets.[2] Furthermore, if we require that the enumerating function, f, be one-one and total, then, as shown by the following classic result of Myhill, we obtain a characterization of the *paddable* sets (which are often called *cylinders*): a nonempty set is a one-one splinter if and only if it is a recursively enumerable cylinder [37].

We should note that analogies between recursion-theoretic concepts and complexity-theoretic notions are notoriously slippery, and in a polynomial setting there is no difference between a total function computable in polynomial time and a partial function computable in polynomial time (since one can add a clock to the function). Thus, our i enumerable sets might be regarded *either* as an analog of the recursively enumerable sets or of splinters. What we shall see is that in a polynomial setting something intermediate happens. For example, although the "if" direction of Myhill's theorem fails in a polynomial setting, in Section 4 we prove that the "only if" direction not only holds, but can be substantially strengthened: all recursively enumerable sets having a one-one, polynomial-time computable function that both preserves membership and is size-increasing are i-enumerable. It follows that all exponential-time complete sets are bi-enumerable (i.e., there is a single polynomial enumerator that enumerates both the set and its complement), as are the k-creative sets with one-one honest productive functions, the recursive sets with honest padding, and all standard NP-complete sets. Furthermore, all recursively enumerable sets with honest padding functions are i-enumerable.

For sets that are self-reducible via a one-one function that is length-increasing *and invertible* in polynomial time, Section 5 proves a stronger and more difficult result: Recursively enumerable sets that are self-reducible via such a reduction are i-enumerable via a polynomial-time invertible enumeration function. As a consequence, all recursively enumerable polynomial-time cylinders are invertibly i-enumerable. Similarly, we obtain bi-enumerability results via invertible iterating functions for recursive sets.

Beyond invertibility, one might ask if i-enumerable sets can be enumerated via lexicographically monotonically increasing functions. Section 6 presents evidence that Σ_k^p-complete sets lack monotonic enumerators: if *any* Σ_k^p-complete set is *monotonically* i-enumerable, then $\Sigma_k^p = \Pi_k^p$. Moreover, we show that the class of sets that are monotonically bi-enumerable is "essentially" (in this case, up to one-one reductions) the same as parity polynomial time.

2. Relation to other notions of enumeration

This section discusses and compares i-enumerability to other definitions that have been proposed to capture the intuitive notion of a "polynomially enumerable set."

[2] It is easy to see that when the iterating function, f, is required to be *total*, any splinter of f must either be cofinite or its complement must have an infinite r.e. subset; i.e., a splinter cannot be a *simple* set, so not every r.e. set is the splinter of a total recursive function.

In view of the characterization of the recursively enumerable sets by iterated applications of Turing machines, the notion of i-enumerable sets, which we introduce in this paper, can be viewed as one reasonable time-bounded analog of the recursively enumerable sets—one that has its roots embedded in a particularly clear, simple, and intuitively appealing concept of enumerability. However, as pointed out earlier, polynomial-time analogs of recursion theoretic concepts are notoriously slippery. For example, it is well-known that both P and NP$\cap$coNP can be viewed as an appropriate time-bounded analog of the recursive sets. Similarly, there are several formulations of polynomial enumerability that may appropriately be viewed as an analog of recursive enumerability.

Perhaps the most commonly cited polynomial-time analog of the r.e. sets is NP. Because of their well-known quantifier characterization [47, 55], sets in NP are often appropriately thought of as an analog of the recursively enumerable sets. But this characterization gives no clear method for giving a polynomial-time enumeration of sets in NP, and so, for our purposes, the quantifier characterization of NP does *not* give an analog of *enumerability*.

In fact, efficient *enumeration* of sets has not been much studied in the literature of complexity theory. One common notion of polynomial-time enumeration, defined by Hartmanis and Yesha [30], is P-printability.[3] A set A is ***P-printable*** if there is a polynomial-time Turing maching M that on input 0^n prints all elements of length at most n. This definition has several deficiencies that make one hesitate to crown it the "right" notion of polynomial enumerability, particularly if one is looking for an analog of the notion of recursively enumerable. The definition of P-printability allows only polynomial-time computable sets to be enumerable, and, equally seriously, allows only sparse sets to be enumerable. Though the sparseness restriction can be removed by considering the larger class of polynomial-time rankable sets [16, 29, 27], this larger class still has the limitation of being a subset of deterministic polynomial time (unlike i-enumerability, which clearly is not).

Earlier, Young and others defined very general models for enumerations and investigated their control structures, orderings, and behaviors in a quite general complexity-theoretic setting. In doing so, for any monotonically increasing total recursive function t, Young defined an infinite set to be enumerable in time t if some Turing machine enumerates the set, and for all n, at least n distinct elements of the set are always enumerated within at most $t(n)$ steps [59]; letting $t(n)$ vary over all polynomials gives a notion of polynomial enumeration. With this definition, for example, one can prove that any translator from one programming system to another can change the underlying orders of the sets being enumerated in only certain obvious, and trivial, ways [48].

Young's definition [59] has neither of the deficiencies we have cited for P-printable sets, but it does have some limitations not shared by i-enumerations. For example,

[3] The notion of P-printability is closely connected to notions of time-bounded Kolmogorov complexity due to Adleman, Hartmanis, and Sipser [2, 18, 46], and has been studied in many papers [26, 4, 3, 23].

Young's definition excludes all sub-polynomially dense sets (even trivial ones, like $\{1^{2^n} \mid n \in N\}$, that some people feel are, intuitively, "polynomially enumerable"), and it includes all recursively enumerable sets that have a uniformly polynomially dense P-printable subset—even though some such sets have such high membership-testing complexity that it appears that any polynomial enumerator (in Young's sense) must consecutively output elements of vastly differing sizes. Thus the enumeration is perhaps not always as "smooth" as one might like.

In contrast, i-enumerability allows many sub-polynomially dense sets. For example, the set $\{1^{2^n} \mid n \in N\}$ is i-enumerable, although no sub-logarithmically dense sets can be i-enumerable. And, in Theorem 5.1 we show that many sets can be i-enumerated in such a way that the enumerator never outputs an element whose size is much different than the size of its input. On the other hand, there are also sets that are enumerable via Young's definition but not i-enumerable.[4]

Finally, Selman [44] has proposed that exactly the sets in NP be considered "polynomial-enumerable."

Definition 2.1 (Selman [44]). A set B is *polynomial-enumerable*$_{SEL}$ if $B = \emptyset$ or if there is a function f such that:

(1) $B = range(f)$,

(2) f is computable in polynomial time, and

(3) there is a polynomial q such that

$$(\forall y)[y \in range(f) \Rightarrow (\exists x)[|x| \leq q(|y|) \text{ and } y = f(x)]].$$

Selman proves that a set is polynomial-enumerable$_{SEL}$ if and only if it is in NP. Polynomial-enumerability$_{SEL}$ corresponds to natural enumeration in the following sense: one can run the enumeration function for a set B on every string in Σ^* in lexicographical order, and the outputs will be a list enumerating the elements of B. Also, because the honesty condition requires just *one* honest witness for each element of the range, Selman's notion is freed from the density restrictions that plague the enumerability notions of both this paper and [59]. However, Selman's enumerability has some undesirable features. The enumeration functions are not necessarily one-one—elements may be repeated tremendously often. Enumerations that are not one-one give the enumerator an artificial ability to "pause" and thus may go for extremely long stretches without producing any *new* element. In our opinion, this violates a reasonable intuition about what polynomial enumeration should mean. Furthermore, although this ability to "pause" is exactly the feature that frees Selman's notion from density woes, it also prevents his enumeration from achieving the extremely uniform production of output that iterative enumerability possesses. It is easy to see that the class of sets enumerated in Selman's sense by functions

[4] An example is the set $L = \{x \mid a_{|x|} = |x|\}$ with $a_0 = 0$ and $a_{i+1} = 2^{a_i}$ for $i \geq 0$. Since L has superpolynomial gaps, it is certainly not i-enumerable. On the other hand L is easily seen to be polynomially enumerable in Young's sense.

that are partial *and one-one* on their ranges is exactly the class UP[5] (essentially [17]). Thus, unless NP = UP, many-to-one enumerations cannot in general be converted to one-one enumerations.

Many open problems exist on the way to better understanding the connections between i-enumerability and other notions of polynomial enumerability. For example, we noted above that there are i-enumerable sets that are sub-polynomially sparse and hence are not polynomially enumerable in the sense defined in [59]. On the other hand, any i-enumerable set for which the iteration function f always produces a polynomial number of elements in polynomial time *is* polynomially enumerable in the sense of [59]. What is the relationship, especially in light of Theorem 5.1, between Young's polynomial enumerability and i-enumerability on sets that are uniformly dense in some sense?

3. Notations and definitions

When speaking of complexity classes, we use standard terminology such as P, NP, PSPACE, and so on [28]. We will switch frequently and implicitly between the natural numbers and words over the alphabet $\{0, 1\}$ by identifying $n \in N$ with the strings in Σ^*, using the natural bijection: the integer i is identified with the ith string in lexicographical order. Consequently, algorithms having natural numbers as input or output in fact operate on the strings representing the numbers. For $x \in \Sigma^*$ we will denote by $|x|$ the length of x. A function f is *honest* if the length of its input is bounded by a polynomial in the length of its output, that is, if there is a polynomial q such that for all x, $q(|f(x)|) \geq |x|$.

Throughout the paper we will employ a pairing function $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ defined (using the above-mentioned correspondence between Σ^* and N) by $\langle x, y \rangle = \frac{1}{2}(x+y)(x+y+1)+x$ [9]. Clearly $\langle \cdot, \cdot \rangle$ is polynomial-time computable, one-one, honest, length-nondecreasing, onto, and polynomial-time invertible. To pair sets, we define $A \otimes B = \{\langle x, y \rangle \mid x \in A \text{ and } y \in B\}$.

A *P-cylinder* is a set A that is polynomially isomorphic to some set $B \otimes N$; i.e., there is a polynomial-time computable function π that is one-one, onto, and invertible in polynomial time with [36]

$$x \in A \Leftrightarrow \pi(x) = \langle y, n \rangle \text{ and } \langle y, n \rangle \in B \otimes N.$$

We denote this by $A \equiv^{p}_{1\text{-}1} B \otimes N$. It is easy to show that in this case even $A \equiv^{p}_{1\text{-}1} A \otimes N$ holds [36, pp. 211–212]. For a set A, a one-one function $pad : \Sigma^* \times N \to \Sigma^*$ that satisfies $pad(x, n) \in A \Leftrightarrow x \in A$ is called a *padding function* for A. We will call a set *honestly paddable* if it has a padding function that is honest in both arguments.

[5] UP (defined by Valiant [52], see also [41, 17, 22]) is unambiguous polynomial time, the class of languages accepted by nondeterministic polynomial-time Turing machines that on *no* input have more than one accepting path.

Berman and Hartmanis [6] and Mahaney and Young [36] showed that a set A is a P-cylinder if and only if A has a polynomial-time computable padding function that is polynomial-time invertible. (The invertibility requirement usually refers to the second argument, but this is equivalent to demanding invertibility in both arguments [36].)

We now formally define the types of enumerability that this paper studies. A set is bi-enumerable if both the set and its complement are enumerated by a single enumeration function. A set is monotonically i-enumerable if it is i-enumerable by an enumeration function that is monotonically increasing. A set is invertibly i-enumerable if it is i-enumerable via an enumeration function whose inverse can be computed in polynomial time.

Definition 3.1. (1) A set $L \subseteq \Sigma^*$ is *polynomially enumerable by iteration* (*i-enumerable*) if a polynomial-time computable function $f: \Sigma^* \to \Sigma^*$ and an element $x \in \Sigma^*$ exist such that: $L = \{x, f(x), f(f(x)), \ldots\}$. The function f is called the *enumeration function* or the *iteration function* for L.

(2) A set $L \subseteq \Sigma^*$ is *polynomially bi-enumerable by iteration* (*bi-enumerable*) if there are elements $x \in \Sigma^*$ and $y \in \Sigma^*$, and a polynomial-time computable function $f: \Sigma^* \to \Sigma^*$, such that: $L = \{x, f(x), \ldots\}$ and $\bar{L} = \{y, f(y), \ldots\}$.

(3) A function f is *monotonic* if for all x, $f(x) >_{lex} x$, i.e., $f(x)$ is lexicographically greater than x. A set $L \subseteq \Sigma^*$ is *monotonically* (*bi-*)*enumerable by iteration* ((*bi*)*-i-enumerable*) if it is (bi)-i-enumerable via a monotonic enumeration function.

(4) A function f is *invertible* if it is one-one and f^{-1} is polynomial-time computable on *range*(f).[6] A set $L \subseteq \Sigma^*$ is *invertibly i-enumerable* (*invertibly bi-enumerable*) if it is an i-enumerable set (a bi-enumerable set) with an invertible enumeration function.

In our analysis of i-enumerations we will be particularly interested in a special kind of self-reducibility.[7]

Definition 3.2. (1) $A \leq^p_{1,w} B$ if there is a one-one polynomial-time computable function f with $x \in A \Leftrightarrow f(x) \in B$ and $|f(x)| > |x|$ for all x.

(2) We will call a set $A \leq^p_{1,w}$-*self-reducible* if $A \leq^p_{1,w} A$.

Intuitively, sets that are $\leq^p_{1,w}$-self-reducible may be viewed as having weak padding functions.

[6] For such a function, *range*(f) is polynomial-time decidable by checking whether $f(f^{-1}(x)) = x$ holds; thus the two common definitions of f^{-1}—one detecting values not in the range and one failing arbitrarily on such values—are equivalent for the purposes of this paper.

[7] Traditionally, a *self-reduction* is expected to reduce the membership question for an arbitrary element, x, to membership questions for some number of other elements, where the latter elements are *smaller* in size (or at least lexicographically smaller) than x [33]. Thus, defining a set to be self-reducible if $A \leq^p_{1,w} A$ is perhaps something of a misnomer. On the other hand, $\leq^p_{1,w}$ is a well-studied reduction, and in this sense $A \leq^p_{1,w} A$ is obviously a self-reduction of A. Thus, we hope that the reader will forgive our abuse of traditional terminology in the following definition.

4. Iteratively-enumerable sets

This section gives a sufficient condition for a set to be i-enumerable. Theorem 4.2 shows that a recursively enumerable set is i-enumerable if it is $\leq^p_{1\text{-}tt}$-self-reducible, with a similar result for bi-enumerability for the recursive sets. This is sufficient to show that many common sets are bi-enumerable.

Before we present our main theorems, we illustrate our basic technique with the simple example of the well-known NP-complete set *SAT*.

Example 4.1. *SAT is bi-enumerable.*

Proof of Example 4.1. *SAT* is a P-cylinder [6], hence $SAT \equiv^p_{1\text{-}tt} SAT \otimes N$. Since the property of being bi-enumerable is invariant under polynomial isomorphisms, it suffices to show that we can cycle through $SAT \otimes N$ and $\overline{SAT \otimes N}$ with an appropriate enumeration function, and since our pairing function is onto (thus $\overline{SAT \otimes N} = \overline{SAT} \otimes N$), it suffices to show that we can cycle through $SAT \otimes N$ and $\overline{SAT} \otimes N$ with an appropriate enumeration function. If g is an enumeration function for $SAT \otimes N$ and π a polynomial isomorphism from *SAT* to $SAT \otimes N$ then $\pi^{-1}(g(\pi(x)))$ is an enumeration function for *SAT*. For each formula F we will call the set $\{F\} \otimes N$ the *column* of F.

The idea of the proof is that the enumeration function climbs up one column until there is enough time to recompute the entire history of the enumeration function from the initial element up to the point the column could have been last left. Thereafter, the enumeration function determines, by deterministic search, an appropriate pair that has not been reached before and jumps to that pair.

The initial elements x and y in our construction will be $\langle T, 0\rangle$ and $\langle F, 0\rangle$, respectively. For the whole procedure we will fix a polynomial $p(n)$ of degree greater than 1, e.g., $p(n) = n^2 + 2$. Furthermore let the function e be defined by

$$e(0) = 0 \quad \text{and} \quad e(k) = 2^{2^{e(k-1)} \cdot p(|\langle F_0, e(k-1)\rangle|)} \quad \text{for } k > 0.$$

Note that for a given n it is easy to determine in linear time whether n is of the form $e(k)$ for some k. The enumeration function f is defined by the following algorithm.

On input $\langle F_0, n\rangle$ do:

- Check if n is of the form $e(k)$ for some k. If this is not the case output $\langle F_0, n+1\rangle$.
- If n is of the form $e(k)$ for some k, then try to recompute in time $p(|\langle F_0, n\rangle|)$ the sequences $x, f(x), f(f(x)), \ldots$ and $y, f(y), f(f(y)), \ldots$ in parallel until one reaches the pair $\langle F_0, e(k-1)+1\rangle$ (recomputation part); call the elements found in the former sequence H_x and in the latter H_y. Thereafter try to determine in time $p(|\langle F_0, n\rangle|)$—by lexicographically enumerating all strings that do not occur in

one of the two sequences so far and deterministically checking—the smallest pair $\langle E, j\rangle$ with the properties:

(1) $\langle E, j\rangle \notin H_0$ and $\langle E, j\rangle \notin H_1$, i.e., $\langle E, j\rangle$ has not been met by either of the sequences before,

(2) $E \leq e(k-1)$, and

(3) $E \in SAT \Leftrightarrow F_0 \in SAT$ (generation part).

- If either the recomputation part or the generation part runs out of time (i.e., takes more than $p(|\langle F_0, n\rangle|)$ steps) before it accomplishes its goal, then output $\langle F_0, n+1\rangle$.

Why is the function f defined above an enumeration function for $SAT \otimes N$ and for $\overline{SAT} \otimes N$? Clearly, f maintains membership in the set and its complement, i.e., $f(\langle F_0, n\rangle) \in SAT \otimes N$ if and only if $\langle F_0, n\rangle \in SAT \otimes N$, since a new element is generated only in two ways—either we output an element of the form $\langle F_0, n+1\rangle$, or the output is determined by brute force deterministic search. In the latter case (by construction) one of the sequences H_0 or H_1 must contain the pair $\langle F_0, e(k-1)+1\rangle$, where $e(k) = n$. Thus we know, whether $\langle F_0, e(k)\rangle$ is in $SAT \otimes N$ or not, and we are able to satisfy condition (3).

It remains to show that each element of $SAT \otimes N$ and $\overline{SAT} \otimes N$ eventually occurs in one of the chains H_0 or H_1, i.e., the chains do not cycle or omit any pair. First note that if we leave a column, say at $\langle F_0, e(k)\rangle$, we know all the elements the chain has reached up to the pair $\langle F_0, e(k-1)+1\rangle$. By construction we know that the column was not left between the pairs $\langle F_0, e(k-1)+1\rangle$ and $\langle F_0, e(k)\rangle$. An element of the form $\langle F_0, l\rangle$ with $l > e(k-1)$ will not be accessed because of condition (2). Hence the enumeration function will never map a pair to an element that was already reached by one of the sequences.

On the other hand, since each $\langle E, j\rangle$ eventually appears as the lexicographically smallest unreached pair with E being a satisfiable (or unsatisfiable) formula, we will have proved our claim if we can show that each column will be left after a finite number of steps. Therefore suppose we are in a column over a formula F_0, and this column was entered by one of the chains at an element $\langle F_0, e(k)+1\rangle$ the last time. For $\langle F_0, e(k+1)\rangle$ let T_0 be the time that is necessary to both (a) recompute the sequences up to $\langle F_0, e(k)+1\rangle$, and (b) to determine the smallest pair $\langle E, j\rangle$ satisfying the conditions (1), (2), and (3). For input $\langle F_0, e(l)\rangle$ with $l > k$ the time necessary to recompute the chains up to $\langle F_0, e(l-1)+1\rangle$ and find the pair $\langle E, j\rangle$ is smaller than the sum of T_0 and the time the algorithm spends on the elements in the column of F_0. The maximal number of elements that have to be checked in this column at point $\langle F_0, e(l)\rangle$ is bounded by $e(l-1)+1$, on each of which the algorithm does not run longer than $r(|\langle F_0, e(l-1)+1\rangle|)$ steps, where r is a polynomial majorizing its run time. An easy calculation now yields that for l large enough

$$p(|\langle F_0, e(l)\rangle|) \geq T_0 + (e(l-1)+1) \cdot r(|\langle F_0, e(l-1)+1\rangle|)$$

holds. Thus there will be time to recompute the history up to the desired point, and the column can be left after a finite number of steps. □

Looking at the proof of this theorem, it is easy to see that only two properties of *SAT* were exploited—namely that *SAT* is polynomially isomorphic to its cylindrification and that *SAT* has a deterministic (exponential-time) decision procedure. In fact a closer analysis suggests that the proof will work no matter how long the latter decision procedure takes. This suggests that all recursive P-cylinders may be bi-enumerable. Furthermore, the proof did not use the full power of the cylinder. It mostly used the fact that the padding function could continuously produce new elements (i.e., that it is cycle-free), that the columns could be periodically "tested" for membership, and that it could be efficiently checked whether two elements are in the same column. This all suggests that the full power of cylinders is also not needed. Thus, Theorem 4.2 and its corollaries show that a considerably broader class of sets is bi-enumerable.

Theorem 4.2. (1) *Every nonempty r.e. set that is* $\leq^{p}_{1,\omega}$*-self-reducible is i-enumerable.*
(2) *Every recursive set* $\emptyset \neq L \neq \Sigma^*$ *that is* $\leq^{p}_{1,\omega}$*-self-reducible is bi-enumerable.*

Proof. The proof that the self-reduction implies the enumerability is essentially the same as for the example with *SAT*. For the first part let L be r.e. and h be a $\leq^{p}_{1,\omega}$-self-reduction[8] for L. Let q be a polynomial such that

- the distance function[9]

$$d(y, z) = \begin{cases} \text{the unique } n \text{ such that } h^n(y) = z & \text{if } n \text{ exists,} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

  is computable in time $q(|y|+|z|)$, and
- the function that yields the value of multiple h-steps $st(y, n) \stackrel{\text{def}}{=} h^n(y)$ is computable in time $q(|st(y, n)|)$.

Clearly, by the size-increasing property of h such a polynomial can be found. Since L is r.e., the set $R = \{x \in L \mid (\forall y \in L)[h(y) \neq x]\}$—the *roots* of L (relative to h)—is r.e. as well. Let M be a machine that enumerates R, and let x_0 be the smallest element in R (equivalently, the smallest element in L). x_0 will be the starting element in our construction. The function $e(n)$ remains as defined in the proof of Example 4.1. Again we fix a working polynomial p that majorizes q and n^2+2, say $p(n) = (q(n)+n^2+2)^2$.

The following algorithm defines an enumeration function f for L:

Input x:

(a) Recompute in time $p(|x|)$ the sequence $x_0, f(x_0), \ldots$ as far as possible. Call the resulting set $X = \{x_0, x_1, x_2, \ldots, x_k\}$ where $f(x_i) = x_{i+1}$ for $0 \leq i < k$. Let x_r be

[8] The proof will show that for Part 1 of Theorem 4.2 the condition that h is a $\leq^{p}_{1,\omega}$-self-reduction could even be weakened to a one-sided version of $\leq^{p}_{1,\omega}$: h must merely be a polynomial-time computable function such that, for all $x \in L$, (1) $h(x) \in L$, (2) $|h(x)| > |x|$, and (3) $\forall y \in L: x \neq y \Leftrightarrow h(x) \neq h(y)$.

[9] As usual, $h^n(\cdot)$ denotes the n-fold application of h on y.

the root with the highest index occurring in X and $V = \{x_i \in X \mid h(x_i) \notin X \text{ and } 0 \leq i < k\}$.

(b) If $d(x_r, x) = e(i)$ for some i and $z = st(x_r, e(i-1)+1) \in X$ then run M for $p(|x|)$ steps and call the generated set R_x. Determine the set $h(V) = \{y \mid (\exists x_i \in V)[h(x_i) = y]\}$ and the lexicographically smallest element w in $(R_x - X) \cup h(V)$ (if the set is not empty). If $|w| \leq |z|$ then output w.

(c) If part (b) does not output an element, then output $h(x)$.

Why does this work? Clearly the procedure runs in polynomial time. Moreover, membership is maintained, since new elements are accessed exclusively in two ways: either via h where membership in L is maintained by definition or via the enumerator of the roots of L.

Again it remains to show that the generated chain $x_0, f(x_0), \ldots$ does not cycle or omit any formula. First note that, whenever on a certain input x an element v appears in the chain $x_0, f(x_0), \ldots, x$ when the invariant (*) holds that either $h^{-1}(v)$ is undefined or $h^{-1}(v)$ was reached by that chain before v. In particular, if an element v is accessed by a step of type (b) then $h^{-1}(v)$ is undefined or it is in X, which contains the elements of the chain we currently can recompute.

For any $v \in \Sigma^*$ let us call the set $\{r, h(r), \ldots\}$ with r being the uniquely determined element in R, for which $h^i(r) = v$ for some i holds, the *splinter of* v. Now V consists of those elements of X, on which a splinter was left and not re-entered within X. If for x now $d(x_r, x) = e(i)$ with x_r as in case (b) and $z = st(x_r, e(i-1)+1) \in X$ holds, then no element in $(R_x - X) \cup h(V)$ of length $\leq |z|$ has been reached by the chain up to x, since by construction the splinter of x_r was not left between z and x, and h is length-increasing. Thus if $(R_x - X) \cup h(V)$ is nonempty, part (b) returns an element that has not been accessed before. It follows readily that part (c) cannot step into a cycle either, since otherwise (*) would be hurt, and therefore f is cycle-free on L.

The fact that the chain does not omit any element follows from the same arguments as in the proof of Example 4.1. It suffices to show that every splinter can be left after a finite number of steps each time it is entered, since every word of L must appear at some point either in the roots of L or in the splinter of an element in $h(V)$. The proof that every splinter can be left after a finite number of steps is identical to the one in the proof of Example 4.1.

The second part of the theorem is obtained by a straightforward modification of the proof of the first part. One simply performs a dovetailing in the recomputation of the two chains as in Example 4.1. □

An immediate consequence of Theorem 4.2 is that every r.e. P-cylinder is an i-enumerable set and that every nontrivial recursive P-cylinder is bi-enumerable. But Corollary 5.2 will give a much stronger result about P-cylinders.

The converse direction of the recursion-theoretic equivalence between recursively enumerable cylinders and one-one i-enumerable sets provably does not hold in a

polynomial-time environment. This is easy to see by density arguments, since, e.g., the tally set $L = \{0^n \mid n \in N\}$ certainly is bi-enumerable but is not a P-cylinder. In general, it is easy to construct sparse sets that are i-enumerable, and sparse sets clearly cannot be P-cylinders since they cannot have padding functions. But it is also straightforward to construct dense and co-dense sets that are i-enumerable but not P-cylinders. All of these sets can be constructed to be computationally fairly easy, but Corollaries 4.4 and 4.7 will show that there are also hard sets that are not known to be P-cylinders, but that are nevertheless bi-enumerable.

It is an open problem (the Berman–Hartmanis Conjecture) whether all NP-complete sets are P-isomorphic. This is the case if and only if every NP-complete set is a P-cylinder [6, 36]. An immediate consequence of the truth of this conjecture would be that $P \neq NP$. Joseph and Young provided evidence that there are NP-complete sets that are not P-cylinders [32] by directly constructing a class of NP-complete sets—the k-creative sets—that do not seem to be P-cylinders. To understand these sets, let $\{M_i\}_{i \in N}$ be any standard enumeration of *non*deterministic polynomial-time Turing machines.

Definition 4.3 (Joseph and Young [32]). A set L is *k-creative* if $L \in NP$ and there exists a polynomial-time computable function f such that for all i: if M_i runs in time $|i|n^k + |i|$ then $f(i) \in L \Leftrightarrow f(i) \in L(M_i)$. The function f is called the *productive function* for $\bar{L}$. (All k-creative sets are known to be NP-complete [32].)

Corollary 4.4. (1) *Every set L that is k-creative with an honest, one-one productive function for $\bar{L}$ is bi-enumerable.*

(2) *Every set in* NP *that is honestly paddable is bi-enumerable.*

Proof. For k-creative sets with one-one honest productive functions, Joseph and Young [32, p. 233] construct a one-one polynomial-time padding function. By direct examination of their proof [32, p. 233] this padding function is size-increasing (and therefore honest). Furthermore, sets that are honestly paddable are easily seen to be $\leq^p_{1,1}$-self-reducible: Let *pad* be a padding function of L, and let q be a polynomial witnessing the honesty of *pad*. Then the function $h(x) = pad(x, 0^{q(|x|)+1})$ is easily seen to provide a $\leq^p_{1,1}$-self-reduction of L. This proves both parts of the corollary. □

Watanabe and others have observed that for any one-one polynomial-time computable and honest function f and any NP-complete set S, the set $f(S)$ must be NP-complete, and it is not clear that $f(S)$ is polynomially isomorphic to S *unless* the function f is invertible in polynomial time [54]. Thus, even when S is polynomially i-enumerable, if f is a candidate for a one-way function, it is not clear that $f(SAT)$ will be polynomially i-enumerable. Sets like $f(S)$, which first require a separate *a priori construction* of an NP-complete set S, are not *standard* in the

sense that, unlike the k-creative sets and standard NP-complete sets like *SAT*, they do not arise by direct construction, but instead rely on the a priori existence of known NP-complete sets.

Since the only known NP-complete sets that are obtained by direct construction are the common ones, such as *SAT* and *CLIQUE*, which are known to be polynomial-time isomorphic to *SAT*, and certain k-creative sets with one-one and honest productive functions, we can state Observation 4.5.

Observation 4.5. All currently known *standard* NP-complete sets are bi-enumerable.

From this observation, a question naturally arises. Under the assumption that $P \neq NP$, does the notion of bi-enumerability subsume NP-completeness? Is every NP-complete set bi-enumerable? We show that this is not the case in relativized worlds, and not even the case in relativized worlds where inverting functions is easy

Theorem 4.6. *There is a recursive oracle A that is reasonable ($P^A \neq NP^A$) relative to which one-way functions do not exist and there is NP^A-complete set that is not i-enumerable by any P^A computable enumeration function (and thus not bi-enumerable by any P^A computable enumeration function).*

Proof. Since any set polynomial-time isomorphic to an i-enumerable set is itself i-enumerable, the oracle A must (as a necessary but not sufficient condition) ensure that there exist non-P^A-isomorphic NP^A-complete sets. Kurtz [35] proves that there is an oracle relative to which $P^A \neq NP^A$ and the Berman-Hartmanis conjecture fails so badly that there exists an NP^A-complete set B that, though dense and co-dense, contains enormous gaps; Hartmanis and Hemachandra [21] extend this construction to ensure that, in addition, no one-way functions exist. It is easy to see that these gaps in the oracle (i.e., empty segments of the oracle) are so large that they preclude any possibility of B being bi-enumerable by a P^A computable enumeration function (as the enumeration functions have no way of bridging the gaps). □

Another consequence of Theorem 4.2 is the following.

Corollary 4.7. *All sets complete for exponential time[10] under polynomial-time many-one reductions are bi-enumerable.*

Proof. Berman [5] (see also [53, 10]) has shown that the complete many-one degree for exponential time collapses to a single $\leq^P_{1,1}$-degree. □

5. Invertibly i-enumerable sets

From our construction in Theorem 4.2, it is clear that in general, enumeration functions—even assuming that they are one-one—will not be polynomial-time

[10] The claim holds for DTIME($2^{(2n+1)}$) as well as for DTIME($2^{n^{(2k)}}$).

invertible, since extremely "nonhonest" jumps occur at the points where we plunge down to some very small string. This section asks what conditions a set must satisfy in order to be invertibly i-enumerable (Definition 3.1).

Theorem 5.1 will show that sets that are $\leq^P_{1,si}$-self-reducible via polynomially invertible functions are perforce invertibly i-enumerable. Note that the class of sets that are $\leq^P_{1,si}$-self-reducible via polynomially invertible functions properly contains the class of r.e. P-cylinders, since there are tally and sparse sets that are $\leq^P_{1,si}$-self-reducible via polynomially invertible functions and, as remarked earlier, sparse sets cannot be P-cylinders.

Theorem 5.1. (1) *Every nonempty r.e. set that is $\leq^P_{1,si}$-self-reducible via a polynomially invertible function is invertibly i-enumerable.*

(2) *Every recursive set $\emptyset \neq L \neq \Sigma^*$ that is $\leq^P_{1,si}$-self-reducible via a polynomially invertible function is invertibly bi-enumerable.*

Before giving the formal proof let us describe intuitively what happens in our construction (see also Figs. 1 and 2). Let x_0 be the lexicographically smallest element of L. Let the roots of L be as defined in the proof of Theorem 4.2. Again, a set $\{r, h(r), \ldots\}$ containing a word r with r being a root will be called *the splinter of* x. If $h^i(r) = x$ then x will be said to have height i in this splinter.

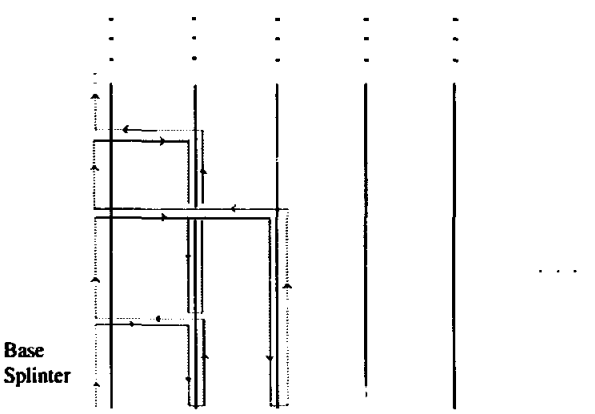

Fig. 1. Cycling scheme of the enumeration function.

The algorithm that performs this enumeration gives—unlike in the proof of Theorem 4.2—the splinter of x_0, which we will call the *base splinter*, special treatment (see Fig. 1). If the input element is in this set, we perform the same technique as before; that is we try to recompute our history up to a certain height in the base splinter and determine a new element in another splinter. If we have enough time to find this element, then it will not be accessed directly, but instead we enter the splinter of the new element at approximately the same height as the current element of the base splinter.

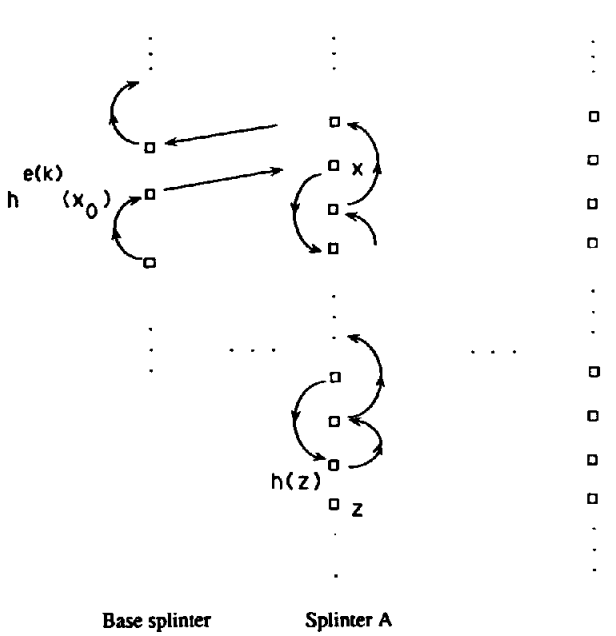

Fig. 2. Interleaving the steps of the chain of the enumeration function on non-base splinters.

On inputs that are in a nonbase splinter the algorithm acts differently. The chain has entered such a splinter—let us call it A—from the base splinter at an element x of height much greater than that of the element z where it was left the last time (recall that when A was entered we were able to recompute our history in polynomial time in the length of the input). Therefore we will organize the cycling scheme in the following way. To fill the interval between x and z we walk down in A, using the invertibility of h until we detect the last point the chain has reached before, and then we walk up again. Since we cannot allow any element to be accessed twice on that course, the up- and the down-trail have to be interleaved. We walk the down-trail on the even heights of A, and the up-trail on the odd ones. As soon as we have reached the point where A was entered the last time we go back to the base splinter and iterate the algorithm (see Fig. 2). How these critical points can be detected is shown below.

Proof. We restrict ourselves to the proof for r.e. sets. The other case is again obtained in a straightforward way as in Theorem 4.2.

Let L be r.e. and h be a one-one function that reduces L size-increasing to itself and is invertible in polynomial time. For every x let $root(x)$ be the uniquely determined element $r \in \Sigma^*$, for which $h^i(r) = x$ for some i holds and for which, for no $y \in \Sigma^*$ does $h(y) = r$. Furthermore let $splinter(x) = \{root(x), h(root(x)), \ldots\}$. Let q be a polynomial such that:

- the function $rt(x) \stackrel{\text{def}}{=} (r, n)$ with $r = root(x)$ and $h^n(r) = x$ is computable in time $q(|x|)$,
- the function $st(y, n) \stackrel{\text{def}}{=} h^n(y)$ is computable in time $q(|st(y, n)|)$,
- $h^{-1}(x)$ is computable in time $q(|x|)$.
- the function $g(x, n) = \min\{z \mid n \leq |z|, z \in splinter(x), \text{and } st(root(x), i) = z \text{ for some even } i\}$ is computable in time $q(|g(x, n)| + |x| + |n|)$, where the min indicates the lexicographical minimum.

It should be clear that these requirements can be met, since h is size-increasing and invertible in polynomial time.

Now let x_0 be the lexicographically smallest element in L. Let $R = \{x \in L \mid x \neq x_0 \text{ and } (\forall y \in \Sigma^*)[h(y) \neq x]\}$ be the r.e. set of the roots of L without x_0, and let M be an enumerator for R. e remains as defined in the proof of Theorem 4.2. Again let $p(n) = (q(n) + n^2 + 2)^2$.

The following algorithm defines an enumeration function f for L:

Input x:

Compute $rt(x) = (r, n)$.

Case (a): $r = x_0$. Recompute in time $p(|x|)$ the sequence $x_0, f(x_0), \ldots$ as far as possible and call the resulting set $X = \{x_0, x_1, x_2, \ldots, x_k\}$ where $f(x_i) = x_{i+1}$ for $0 \leq i < k$. Let $V = \{x_i \in X \mid h(x_i) \notin X \text{ and } 0 \leq i < k\}$. If $n = e(i)$ for some i and $st(x_0, e(i-1)+1) \in X$ then construct the set $R_x = \{z \mid |z| \leq |x| \text{ and } M \text{ enumerates } z \text{ within } p(|x|) \text{ steps}\}$. If $(R_x - X) \cup V$ is not empty determine the lexicographically smallest element w of $(R_x - X) \cup V$ and output $t = g(root(w), |x|)$. If no element is output before, output $h(x)$.

Case (b): $r \neq x_0$. *Case* (b1): there are k', z, w such that $st(x_0, e(k')) = z$, $f(z) = w$, $w \in splinter(r)$, and $|w| \leq |x|$. Let k be the largest such k'. If n is even, output $h(x)$ if $h^{-1}(h^{-1}(x)) = w$ and otherwise output $h^{-1}(h^{-1}(x))$. If n is odd, output $h(z)$ if $h^{-1}(x) = w$ and otherwise output $h(h(x))$.

Case (b2): there are no k', z, w such that $st(x_0, e(k')) = z$, $f(z) = w$, $w \in splinter(r)$, and $|w| \leq |x|$. If n is even, output $h(x)$ if $n = 0$ and output $h^{-1}(h^{-1}(x))$ otherwise. If n is odd, output $h(h(x))$.

We must show that f is polynomial-time computable. This certainly holds if the input element x is in the base splinter. Otherwise case (b) will be chosen. To determine the required element $z = st(x_0, e(k))$ in the base splinter at most a polynomial number of elements of this form have to be checked, since h is length-increasing. For each of them $f(st(x_0, e(i)))$ has to be computed. Running the algorithm on such an input, option (a) will be chosen, and thus the algorithm runs in polynomial time.

For the correctness of the algorithm first note that on the base splinter, i.e., when option (a) is chosen, the algorithm acts similar to the one in the proof of Theorem 4.2. As soon as there is enough time to replay the history (i.e., $st(x_0, e(i-1)+1) \in X$) a new element w is generated. Therefore either the generator for the roots of L (i.e., the set $R_x - X$) is used or an element y in a nonbase splinter. In the latter case the

chain must have entered that splinter before and left it at $h^{-1}(y)$ (those are the elements of V). In both cases the new element will not be accessed directly. Instead $g(root(w), |x|)$ yields the lexicographically smallest word in the splinter of w that has an even height and is as least as long as x. This will make sure that f is invertible on $f(x)$. Since at this time all the elements touched by the chain before are known (see the proof of Theorem 5.1) the chain will not run into a cycle here.

Once the f-chain has entered a nonbase splinter A at an element t from some element s in the base splinter, it walks down on elements of the form $h^{2i}(root(t))$—thus, unlike the base splinter case, leaving the elements of odd height untouched—until it finds the point where A has been entered the last time by the chain. Thereafter it walks up in A on elements of the form $h^{2i+1}(root(t))$ until it reaches $h(t)$. From there it returns to $h(s)$ in the base spinter. The crucial point we have to prove is that the points where the chain has to change its direction will in fact be detected in the recomputation in case (b). That is, we have to show, that if a nonbase splinter A was entered, during the visit previous to the current one, at element $w = h^{2j}(root(t))$, then $f(h^{2j+2}(root(t))) = h^{2j+3}(root(t))$, and $f(h(t)) = h(s)$.

Let $w' = h^{2j+2}(root(t))$. By way of contradiction suppose the chain enters a cycle in A at the element $w = h^{2j}(root(t))$ and $f(w') = w$ for $w' = h^{2j+2}(root(t))$. That is, suppose that as we hop down a nonbase column we fail to stop before hitting already-visited territory. By construction, w was accessed the last time from a base splinter element z. At z the algorithm had enough time to recompute its history and determine a new element. Since $|w'| > |z|$ it is clear that the element w will be found. Thus at $|w'|$ we detect that we have been at w before and can change our direction.

By the same reason, when we climb up A if we arrive at the point $h(t)$—t being the element where A was entered—we will have time to find the base splinter point s from which t was accessed from the base splinter, since $|h(t)| > |t| > |s|$. This proves that f is cycle-free.

A straightforward consideration of the different cases yields that f is invertible in polynomial time, using the method of this proof. □

As in the general case of sets with a (not necessarily P-invertible) honest padding function, it follows that any r.e. set with a P-invertible padding function is invertibly i-enumerable. This yields the following consequence for P-cylinders.

Corollary 5.2. *If a nonempty set L is a r.e. (recursive) P-cylinder then it is invertibly i-enumerable (invertibly bi-enumerable).*

6. On the monotonicity of enumeration functions

It follows from Section 5 that all NP-complete P-cylinders are honestly[11] bi-enumerable. A natural question arising along this line is: which sets can be generated

[11] An i-enumerable set is said to be *honestly i-enumerable* if its enumeration function is honest.

by monotonically increasing enumeration functions (see Definition 3.1)? Is it possible that some NP-complete sets are monotonically i-enumerable or monotonically bi-enumerable? In this section, we provide evidence that this is not the case.

In this section, we can no longer use the pairing function $\langle \cdot, \cdot \rangle$, as it does not preserve the natural lexicographic ordering between pairs of strings. Let p be some monotonically increasing polynomial (specified appropriately in the following proofs); we now define $\langle\!\langle \cdot, \cdot \rangle\!\rangle_p$, which for our purposes will serve as an adequate form of lexicographically order-preserving pairing function. Let $rank_A(x)$ indicate the position of string x within set A, i.e., $rank_A(x) = \|\{z \mid z \leq_{lex} x \text{ and } z \in A\}\|$ [16, 27]. Let $h(x, y)$ be the concatenation of x and y. Let $R_p = \{z \mid (\exists x, y)[|y| = p(|x|) \text{ and } z = h(x, y)]\}$. Let $D_p = \{(x, y) \mid |y| = p(|x|)\}$. Finally, $\langle\!\langle x, y \rangle\!\rangle_p = \{z \mid rank_{\Sigma^*}(z) = rank_{R_p}(h(x, y))\}$. The pairing function $\langle\!\langle \cdot, \cdot \rangle\!\rangle_p$ is onto, polynomial-time computable, and on inputs from D_p preserves lexicographic ordering in a very natural way. Though this function is not one-one, it does provide a one-one mapping between D_p and Σ^*, and furthermore, there is a polynomial-time algorithm that, given z, finds the unique element of D_p whose image under $\langle\!\langle \cdot, \cdot \rangle\!\rangle_p$ is z.

Definition 6.1. (1) A set A is *uniformly dense* if there is a polynomial $p(n)$ such that for any string x, there is an element of A whose lexicographical position within Σ^* is at most $p(|x|)$ different from the position of x [14].

(2) A set B is a *revealer* of S if B is uniformly dense, $B \in P$, and $B \cap S \in P$.

Note that having a revealer is a bit more general than saying that "either the set or its complement has an infinite P-subset." In [14] it was proven that near-testable sets with revealers are all in P. Since—as shown in Theorem 6.5—all monotonically bi-enumerable sets are near-testable, it follows immediately from their result that all monotonically bi-enumerable sets with revealers are in P and (since any reasonable encoding of $3SAT$ will have a revealer) that there is a reasonable encoding of $3SAT$ that is monotonically bi-enumerable if and only if P = NP [14].

Thus, the [14] encoding of $3SAT$ is unlikely to be a monotonically bi-enumerable NP-complete set. Can *any* NP-complete (or more generally any Σ_k^p-complete) set be monotonically enumerable? We give evidence that none can.

Theorem 6.2. *For each $k \geq 1$, if there is a Σ_k^p-complete set that is monotonically i-enumerable, then* $PH = \Sigma_k^p$.

Proof. Let L be Σ_k^p-complete and let f be its monotonic enumeration function. Then the following equivalence holds:

$$x \notin L \Leftrightarrow \text{there is a } y <_{lex} x \text{ such that } y \in L \text{ and } f(y) >_{lex} x.$$

Clearly, the right-hand side is a Σ_k^p statement. Since $\bar{L}$ is complete for Π_k^p, this proves that $\Pi_k^p \subseteq \Sigma_k^p$ and thus $PH = \Sigma_k^p$.

On the other hand, there are coNP-complete sets that are monotonically i-enumerable.

Theorem 6.3. *There are monotonically i-enumerable* coNP*-complete sets.*

Proof. Let L be an NP-complete set, p be a polynomial, and N be an NP machine accepting L such that every computation path of N on an input x has length exactly $p(|x|)$ and $N(x)$ never accepts on the path $0^{p(|x|)}$. Without loss of generality, we may assume that p is monotonically increasing. Then the set

$$L' = \{z \mid (\forall y')[(|y'| = |y| \text{ and } y' \leq_{lex} y) \Rightarrow (y' \text{ is not an accepting path of } N(x))] \text{ and } |y| = p(|x|), \text{ where } (x, y) \text{ is the unique element of } D_p \text{ such that } \langle\!\langle x, y \rangle\!\rangle_p = z\}$$

is also complete for coNP, since $x \notin L \Leftrightarrow \langle\!\langle x, 1^{p(|x|)} \rangle\!\rangle_p \in L'$. Moreover, L' is monotonically i-enumerable. On input z, find x and y such that $z = \langle\!\langle x, y \rangle\!\rangle_p$ and $|y| = p(|x|)$. Then output $\langle\!\langle x^+, 0^{p(|x^+|)} \rangle\!\rangle_p$ if (1) $y = 1^{p(|x|)}$ or (2) y^+ is an accepting path of N on input x, and output $\langle\!\langle x, y^+ \rangle\!\rangle_p$ otherwise (z^+ denotes the successor of the string z in the lexicographical ordering). □

How are monotonically i-enumerable sets related to PSPACE and the polynomial-time hierarchy? Certainly every monotonically i-enumerable set is contained in PSPACE, but is it possible that, for example, all such sets are contained in the polynomial-time hierarchy? In the following, we suggest a negative answer to this question.

In particular, we show that the class of monotonically bi-enumerable sets is "essentially" the same (in this case up to polynomial-time one-one reductions) as the complexity class $\oplus$P, parity polynomial time, which was defined by Papadimitriou and Zachos [40] and Goldschlager and Parberry [15].

Definition 6.4. (1) $L \in \oplus$P if there is a nondeterministic polynomial-time Turing machine N such that, for all x, $N(x)$ has an odd number of accepting paths if and only if $x \in L$ [40, 15].

(2) L is *near-testable* if the function

$$h(x) = \begin{cases} 1 & \text{if exactly one of } x \text{ and the lexicographical predecessor of } x \text{ is in } L, \\ 0 & \text{otherwise,} \end{cases}$$

is polynomial-time computable [14].

Goldsmith, Hemachandra, Joseph and Young [11] proved—again up to polynomial-time one-one reductions—that $\oplus$P coincides with the class of near-testable sets. Though $\oplus$P contains certain relatively large subsets of NP, such as UP and FewP [8] (see also [34]), it is not known whether NP$\subseteq\oplus$P; indeed, there is a relativized world in which this fails [50].

Theorem 6.5. (1) *Every monotonically bi-enumerable set is near-testable.*

(2) *Every set in* $\oplus\mathrm{P}$ *is polynomial-time one-one reducible to a set that is monotonically bi-enumerable.*

Proof. (1) Let f be the monotonic function that enumerates L and $\bar{L}$, and let x_- denote the lexicographical predecessor of x. Then $h(x) \stackrel{\text{def}}{=} 0$ if $f(x_-) = x$ and $h(x) \stackrel{\text{def}}{=} 1$ otherwise witnesses the near-testability of L.

(2) The proof uses the methods of Goldsmith et al. [11]. For $L \in \oplus\mathrm{P}$ let p be a monotonically increasing polynomial and $R(x, y)$ be a polynomial-time computable predicate such that

$$x \in L \Leftrightarrow \|\{y \mid R(x, y) \text{ and } |y| = p(|x|) - 2\}\| \text{ is odd.}$$

We define $R'(x, y)$ to be the polynomial-time computable predicate that fulfills, for $b, b' \in \{0, 1\}$:

$$R'(x, byb') \Leftrightarrow (b' = 1 \text{ or } R(x, y)) \text{ and } |y| = p(|x|) - 2.$$

Note that, for all x, $\|\{y \mid R'(x, y)\}\|$ is even, whereas $x \in L$ if and only if $\frac{1}{2}\|\{y \mid R'(x, y)\}\|$ is odd. Furthermore we define

$$L' = \{z \mid \text{there are an odd number of pairs } (x', y'), |y'| = p(|x'|) \text{ and } \langle\langle x', y'\rangle\rangle_p \leq_{lex} \langle\langle x, y\rangle\rangle_p, \text{ for which } R'(x', y') \text{ holds, where } (x, y) \text{ is the unique pair in } D_p \text{ such that } \langle\langle x, y\rangle\rangle_p = z\}.$$

Now $g(x) \stackrel{\text{def}}{=} \langle\langle x, 01^{p(|x|)-1}\rangle\rangle_p$ one-one reduces L to L', since

$$\begin{aligned} x \in L &\Leftrightarrow \tfrac{1}{2}\|\{y \mid R'(x, y)\}\| \text{ is odd} \\ &\Leftrightarrow \|\{z \mid y' \leq_{lex} 01^{p(|x|)-1} \text{ and } R'(x, y') \text{ holds, where } (x, y') \text{ is the unique pair in } D_p \text{ such that } \langle\langle x, y'\rangle\rangle_p = z\}\| \text{ is odd} \\ &\Leftrightarrow \langle\langle x, 01^{p(|x|)-1}\rangle\rangle_p \in L'. \end{aligned}$$

It remains to show that L' is monotonically bi-enumerable. For a given pair $\langle\langle x, y\rangle\rangle_p$ we have to find the next $\langle\langle x', y'\rangle\rangle_p$ such that for the pairs that lie lexicographically between $\langle\langle x, y\rangle\rangle_p$ and $\langle\langle x', y'\rangle\rangle_p$ an even number of times R' holds. Clearly, among the three lexicographical successors of $\langle\langle x, y\rangle\rangle_p$ we will find such a pair, since at least every second element in the lexicographical ordering belongs to L'. □

It follows immediately from the above proof that there are complete sets for $\oplus\mathrm{P}$ that are monotonically i-enumerable. If L is any complete set for $\oplus\mathrm{P}$ then the set L' built from L by the above proof is a monotonically i-enumerable $\oplus\mathrm{P}$-complete set.

Corollary 6.6. *There are monotonically i-enumerable* $\oplus\mathrm{P}$*-complete sets.*

Thus, monotonically bi-enumerable sets are "essentially" the same as the $\oplus\mathrm{P}$ sets and the near-testable sets. Parts (1) and (2) of Corollary 6.7 now follow from the downward closure of $\oplus\mathrm{P}$ under many-one reductions. Part 3 is a consequence of Toda's result that $\oplus\mathrm{P} \subseteq \Sigma_k^p$ implies $\mathrm{PH} = \Sigma_{k+1}^p$ [49].

Corollary 6.7. (1) *Every monotonically bi-enumerable set is contained in* $\oplus$P.

(2) *The downward-closures under polynomial-time many-one reductions of* $\oplus$P, *the class of near-testable sets, and the class of bi-enumerable sets are equal.*

(3) *If every monotonic bi-enumerable set is contained in the polynomial-time hierarchy then the polynomial hierarchy collapses. In fact, if a monotonically bi-enumerable set to which any* $\oplus$P*-complete set reduces is in* Σ_k^p, *then* $\mathrm{PH} = \Sigma_{k+1}^p$.

7. Conclusions

Most of classical complexity theory studies the complexity of language *recognition*, but recently there has been much interest in the complexity of operations other than set recognition. This paper studied the complexity of enumerating elements of complex sets, and related it to the complexity of self-reductions within the set. We investigated a broad class of sets that have certain kinds of weak self-reducibilities, or paddings, and proved that all such sets have methodical, polynomial-time computable, enumeration schemes—that is, they are i-enumerable. In particular, this paper proved that all exponential-time complete sets are i-enumerable, and that all NP-complete sets yet obtained by direct construction are i-enumerable. It remains an open question whether all NP-complete sets are i-enumerable. Since we constructed a nontrivial relativized world that has NP-complete sets that are not i-enumerable, this problem will be hard to resolve with current techniques. Another open problem is to characterize the i-enumerable sets in terms of some relatively simple form of self-reducibility. Finally, as discussed in Section 2, the relation of i-enumerability to other notions of polynomial-time enumeration is not yet fully understood.

Acknowledgment

We thank Gerd Wechsung, Eric Allender, Gerhard Buntrock, Judith Goldsmith, Dieter Hofbauer, Sanjay Jain, Ken Regan, and Hubert Wagener for helpful conversations. We are particularly grateful to José Balcázar for pointing out that the bi-enumerability results apply to all recursive P-cylinders, and to Richard Chang for simplifying the proof of Corollary 6.7. Moreover, we thank an anonymous Bleßhuhn for inspiration.

References

[1] M. Abadi, E. Allender, A. Broder, J. Feigenbaum and L. Hemachandra, On generating solved instances of computational problems, in: *Advances in Cryptology—CRYPTO '88*, Lecture Notes in Computer Science **403** (Springer, Berlin, 1990) 297–310.

[2] L. Adelman, Time, space, and randomness, Technical Report MIT/LCS/TM-131, MIT, Cambridge, MA, April 1979.

[3] E. Allender and R. Rubinstein, P-printable sets, *SIAM J. Comput.* 17(6) (1988) 1193–1202.
[4] J. Balcázar and R. Book, Sets with small generalized Kolmogorov complexity, *Acta Inform.* 23(6) (1986) 679–688.
[5] L. Berman, Polynomial reducibilities and complete sets, Ph.D. Thesis, Cornell University, Ithaca, NY, 1977.
[6] L. Berman and J. Hartmanis, On isomorphisms and density of NP and other complete sets, *SIAM J. Comput.* 6(2) (1977) 305–322.
[7] D. Bruschi, D. Joseph and P. Young, A structural overview of NP optimization problems, *Algorithms Review*, to appear.
[8] J. Cai and L. Hemachandra, On the power of parity polynomial time, *Math. Systems Theory*, 23 (1990) 95–106.
[9] M. Davis, *Computability and Unsolvability* (Dover, New York, 1958).
[10] K. Ganesan and S. Homer, Complete problems and strong polynomial reducibilities, in: *Proc. STACS 1989: 6th Ann. Symp. on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 349 (Springer, Berlin, 1989) 240–250.
[11] J. Goldsmith, L. Hemachandra, D. Joseph and P. Young, Near-testable sets, *SIAM J. Comput.*, to appear.
[12] J. Goldsmith, L. Hemachandra and K. Kunen, On the structure and complexity of infinite sets with minimal perfect hash functions, Tech. Report TR-399, Dept. of Computer Science, Univ. of Rochester, NY, 1990.
[13] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman, San Francisco, CA, 1979).
[14] J. Goldsmith, D. Joseph and P. Young, Self-reducible, P-selective, near-testable, and P-cheatable sets: the effect of internal structure on the complexity of a set, in: *Proc. 2nd Structure in Complexity Theory Conf.* (IEEE Computer Science Press, 1987) 50–59.
[15] L. Goldschlager and I. Parberry, On the construction of parallel computers from various bases of boolean functions, *Theoret. Comput. Sci.* 43 (1986) 43–58.
[16] A. Goldberg and M. Sipser, Compression and ranking, in: *Proc. 17th ACM Symp. on Theory of Computing* (1985) 440–448.
[17] J. Grollmann and A. Selman, Complexity measures for public-key cryptosystems, *SIAM J. Comput.* 17 (1988) 309–335.
[18] J. Hartmanis, Generalized Kolmogorov complexity and the structure of feasible computations, in: *Proc. 24th IEEE Symp. on Foundations of Computer Science* (1983) 439–445.
[19] L. Hemachandra, Algorithms from complexity theory: polynomial-time operations for complex sets, in: *Proc. SIGAL Internat. Symp. on Algorithms*, Lecture Notes in Computer Science 450 (Springer, Berlin, 1990) 221–231.
[20] H. Hermes, Zum Begriff der Axiomatisierbarkeit, *Math. Nachrichten* 4 (1950–1951) 343–347.
[21] J. Hartmanis and L. Hemachandra, One-way functions and the non-isomorphism of NP-complete sets, *Theoret. Comput. Sci.* to appear.
[22] J. Hartmanis and L. Hemachandra, Complexity classes without machines: on complete languages for UP, *Theoret. Comput. Sci.* 58 (1988) 129–142.
[23] J. Hartmanis and L. Hemachandra, On sparse oracles separating feasible complexity classes, *Inform. Process. Lett.* 28 (1988) 291–295.
[24] L. Hemachandra and A. Hoene, On sets with efficient implicit membership tests, *SIAM J. Comput.*, to appear.
[25] L. Hemachandra, A. Hoene and D. Siefkes, Polynomial-time functions generate SAT: On P-splinters, in: *Mathematical Foundations of Computer Science 1989, Proc. 14th Symp.* Lecture Notes in Computer Science 379 (Springer, Berlin, 1989) 259–269.
[26] J. Hartmanis, N. Immerman and V. Sewelson, Sparse sets in NP − P: EXPTIME versus NEXPTIME, *Inform. and Control* 65(2/3) (1985) 159–181.
[27] L. Hemachandra and S. Rudich, On ranking, *J. Comput. System Sci.* 41 (1990) 251–271.
[28] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
[29] D. Huynh, The complexity of ranking simple languages, *Math. Systems Theory* 23 (1990) 1–20.
[30] J. Hartmanis and Y. Yesha, Computation times of NP sets of different densities, *Theoret. Comput. Sci.* 34 (1984) 17–32.

[31] M. Jerrum, L. Valiant and V. Vazirani, Random generation of combinatorial structures from a uniform distribution, *Theoret. Comput. Sci.* **43**(2, 3) (1986) 169–188.
[32] D. Joseph and P. Young, Some remarks on witness functions for non-polynomial and non-complete sets in NP, *Theoret. Comput. Sci.* **39** (1985) 225–237.
[33] D. Joseph and P. Young, Self-reducibility: effects of internal structure on computational complexity, in: A. Selman, ed., *Complexity Theory Retrospective* (Springer, Berlin, 1990) 82–107.
[34] J. Köbler, U. Schöning, S. Toda and J. Torán, Turing machines with few accepting computations and low sets for PP, in: *Proc. 4th Structure in Complexity Theory Conf.* (IEEE Computer Society Press, 1989) 208–215.
[35] S. Kurtz, A relativized failure of the Berman-Hartmanis conjecture, Technical Report TR83-001, University of Chicago Department of Computer Science, Chicago, IL, 1983.
[36] S. Mahaney and P. Young, Reductions among polynomial isomorphism types, *Theoret. Comput. Sci.* **39** (1985) 207–224.
[37] J. Myhill, Recursive digraphs, splinters, and cylinders, *Math. Ann.* **138** (1959).
[38] P. Orponen, D. Russo and U. Schöning, Optimal approximations and polynomially levelable sets, *SIAM J. Comput.* **15**(2) (1986) 399–408.
[39] M. Pilchner, R. Rardin and C. Tovey, Polynomial constructibility and traveling salesman problems of intermediate complexity, Technical Report ONR-URI Computational Combinatoric Report CC-88-2, Purdue University, 1988.
[40] C. Papadimitriou and S. Zachos, Two remarks on the power of counting, in: *Proc. 6th GI Conf. on Theoretical Computer Science* Lecture Notes in Computer Science **145** (Springer, Berlin, 1983) 269–276.
[41] C. Rackoff, Relativized questions involving probabilistic algorithms, *J. ACM* **29**(1) (1982) 261–268.
[42] H. Rogers, Jr., *The Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York, 1967).
[43] L. Sanchis, Test case construction for NP-hard problems, in: *Proc. 26th Ann. Allerton Conf. on Communication, Control, and Computing* (1988).
[44] A. Selman, Polynomial time enumeration reducibility, *SIAM J. Comput.* **7** (1978) 440–457.
[45] A. Selman, Analogues of semirecursive sets and effective reducibilities to the study of NP complexity, *Inform. and Control* **52** (1982) 36–51.
[46] M. Sipser, A complexity theoretic approach to randomness, in: *Proc. 15th ACM Symp. on Theory of Computing* (1983) 330–335.
[47] L. Stockmeyer, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977) 1–22.
[48] M. Shay and P. Young, Characterizing the orders changed by program translators, *Pacific J. Math.* **76** (1978) 485–490.
[49] S. Toda, On the computational power of PP and ⊕P, in: *Proc. 30th IEEE Symp. on Foundations of Computer Science* (IEEE Computer Society Press, 1989) 514–519.
[50] L. Torenvliet, Structural concepts in relativized hierarchies, 1986, Ph.D. Thesis, Universiteit van Amsterdam.
[51] J. Ullian, Splinters of recursive functions, *J. Symbolic Logic* **25** (1960) 33–38.
[52] L. Valiant, The relative complexity of checking and evaluating, *Inform. Process. Lett.* **5** (1976) 20–23.
[53] O. Watanabe, On one-one P-equivalence relations, *Theoret. Comput. Sci.* **38** (1985) 157–165.
[54] O. Watanabe, A note on the P-isomorphism conjecture, *Theoret. Comput. Sci.* **84** (1991) to appear.
[55] C. Wrathall, Complete sets and the polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977) 23–33.
[56] P. Young, On semi-cylinders, splinters, and bounded-truth-table reducibility, *Trans. AMS* **115** (1965) 329–339.
[57] P. Young, A theorem on recursively enumerable classes and splinters, *Proc. AMS* **17** (1966) 1050–1056.
[58] P. Young, On pseudo-creative sets, splinters, and bounded-truth-table reducibility, *Z. Math. Logik Grundlagen Math.* **13** (1967) 25–31.
[59] P. Young, Toward a theory of enumerations, *J. ACM* **16** (1969) 328–348.