

L-Printable Sets

Lance Fortnow*
Dept. of Computer Science
University of Chicago

Judy Goldsmith†
Matthew A. Levy†
Computer Science Dept.
University of Kentucky

Stephen Mahaney‡
DIMACS Center
Rutgers University

September 9, 1996

Abstract

A language is L -printable if there is a logspace algorithm which, on input 1^n prints all members in the language of length n . Following the work of Allender and Rubinfeld [AR88] for P -printable sets, we present some simple properties of the L -printable sets. This definition of “ L -printable” is robust, and allows us to give alternate characterizations of the L -printable sets in terms of tally sets and Kolmogorov complexity. In addition, we show that a regular or context-free language is L -printable if and only if it is sparse, and we investigate the relationship between L -printable sets, L -rankable sets (i.e., sets A having a logspace algorithm that on input x outputs the number of elements of A that precede x in the standard lexicographic ordering of strings) and the sparse sets in L . We prove that under reasonable complexity theoretic assumptions, these three classes of sets are all different. We also show that the class of sets of small generalized Kolmogorov space complexity is exactly the class of sets which are L -isomorphic to tally languages.

*Supported in part by NSF grant CCR-9253582.

†Supported in part by NSF grant CCR-9315354.

‡Supported by NSF cooperative agreement CCR-9119999 and a grant from the New Jersey Commission on Science and Technology.

1 Introduction

What is an easy set? Typically, complexity theorists view easy sets as those with easy membership tests. An even stronger requirement might be that there is an easy algorithm to print all the elements of a given length. These “printable” sets are easy enough that we can efficiently retrieve all of the information we might need about them.

Hartmanis and Yesha first defined P-printable sets in 1984 [HY84]. A set A is P-printable if there is a polynomial-time algorithm that on input 1^n outputs all of the elements of A of length n . Any P-printable set must lie in P and be sparse, i.e., the number of strings of each length is bounded by a fixed polynomial of that length. Allender and Rubinfeld [AR88] give an in-depth analysis of the complexity of the P-printable sets.

Once P-printability has been defined, it is natural to consider the analogous notion of logspace-printability. Since it is not known whether or not $L = P$, an obvious question to ask is: do the L-printable sets behave differently than the P-printable sets? In this paper, we are able to answer this question in the affirmative, at least under plausible complexity theoretic assumptions. Jenner and Kirsig [JK89] define L-printability as the logspace computable version of P-printability. Because L-printability implies P-printability, every L-printable set must be sparse and lie in L. In this paper we give the first in-depth analysis of the complexity of L-printable sets. (Jenner and Kirsig focused only one chapter on printability, and most of their printability results concern NL-printable sets.)

Whenever a new class of sets is analyzed, it is natural to wonder about the structure of those sets. Hence, we examine the regular and context-free L-printable sets. Using characterizations of the sparse regular and context-free languages, we show in Section 4 that every sparse regular or context-free language is L-printable. (Although the regular sets are a special case of the context-free sets, we include the results for the regular languages because our characterization of the sparse regular languages is simple and intuitive.)

We might expect many of the properties of P-printable sets to have logspace analogues, and, in fact, this is the case. In Section 5 we show that L-printable sets (like their polynomial-time counterparts) are closely related to tally sets in L, and to sets in L with low generalized *space-bounded* Kolmogorov complexity.

A set is said to have *small generalized Kolmogorov complexity* if all of its strings are highly compressible and easily restorable. Generalized time-bounded Kolmogorov complexity and generalized space-bounded Kolmogorov complexity are introduced in [Ha83] and [S83]. Several researchers [R86, BB86, HH88] show that P-printable sets are exactly the sets in P with small generalized time-bounded Kolmogorov complexity. [AR88] show that a set has small generalized time-bounded Kolmogorov complexity if and only if it is P-isomorphic to a tally set. Using similar techniques, we show in Section 5 that the L-printable sets are exactly the sets in L with small generalized space-bounded Kolmogorov complexity. We also prove that a set has small generalized space-bounded Kolmogorov complexity if and only if it is L-isomorphic to a tally set.

In Section 6, we note that sets which can be ranked in logspace (i.e., given a string x , a logspace algorithm can determine the number of elements in the set $\leq x$) seem different from the L-printable sets. For sparse sets, P-rankability is equivalent to P-printability. We show a somewhat surprising result in Section 6, namely that the sparse L-rankable sets and the L-printable sets are the same if and only if there are no sparse sets in P - L if and only if $\text{linearSPACE} = \text{E}$.

Are all sparse sets in L either L-printable or L-rankable? Allender and Rubinfeld [AR88] show that every sparse set in P is P-printable if and only if there are no sparse sets in $\text{FewP} - \text{P}$. In Section 6, we similarly show a stronger collapse: specifically, that every sparse set in L is L-printable if and only if every sparse set in L is L-rankable if and only if there are no sparse sets in $\text{FewP} - \text{L}$ if and only if $\text{linearSPACE} = \text{FewE}$.

Unlike L-printable sets, L-rankable sets may have exponential density. Blum (see [GS91]) shows that every set in P is P-rankable if and only if every $\#P$ function is computable in polynomial time. In Section 6, we also show that every set in L is L-rankable if and only if every $\#P$ function is computable in logarithmic space.

2 Definitions

We assume a basic familiarity with Turing machines and Turing machine complexity. For more information on complexity theory, we suggest either [BDG88] or [P94]. We also assume a

familiarity with regular languages and expressions and context-free languages as found in [M91]. We will denote the characteristic function of A by χ_A . We will use the standard lexicographic ordering on strings and let $|w|$ be the length of the string w . (Recall that $w \leq_{lex} v$ iff $|w| < |v|$ or $|w| = |v|$ and, if i is position of the leftmost bit where w and v differ, $w_i < v_i$.) The alphabet $\Sigma = \{0, 1\}$, and all strings are elements of Σ^* . We denote the complement of A by \bar{A} .

The class P is deterministic polynomial time, and L is deterministic logarithmic space; remember that in calculating space complexity, the machine is assumed to have separate tapes for input, computation, and output. The space restriction applies only to the work tape. It is known that $L \subseteq P$, but it is not known whether the two classes are equal. The class E is deterministic time $2^{O(n)}$, and LinearSPACE is deterministic space $O(n)$.

Definition 1 *A set A is in the class PP if there is a polynomial time nondeterministic Turing machine that, on input x , accepts with more than half its computations iff $x \in A$. A function f is in #P if there is a polynomial time nondeterministic Turing machine M such that for all x , $f(x)$ is the number of accepting computations of $M(x)$.*

Allender[A86] defined the class FewP. FewE is defined analogously.

Definition 2 [A86] *A set A is in the class FewP if there is a polynomial time nondeterministic Turing machine M and a polynomial p such that on all inputs x , M accepts x on at most $p(|x|)$ paths. A set A is in the class FewE if there is an exponential time nondeterministic Turing machine M and a constant c such that on all inputs x , M accepts x on at most 2^{cn} paths. (Note that this is small compared to the double exponential number of paths of an exponential-time nondeterministic Turing machine.)*

Definition 3 *A set S is sparse if there is some polynomial $p(n)$ such that for all n , the number of strings in S of length n is bounded by $p(n)$ (i.e., $|S^n| \leq p(n)$).*

A set T over alphabet Σ is a tally set if $T \subseteq \{\sigma\}^$, for any character $\sigma \in \Sigma$.*

The work here describes certain enumeration properties of sparse sets in L. There are two notions of enumeration that are considered: rankability and printability.

Definition 4 *If \mathcal{C} is a complexity class, then a set, A , is \mathcal{C} -printable if and only if there is a function computable in \mathcal{C} that, on any input of length n , outputs all the strings of length n in A .*

Note that P-printable sets are necessarily in P, and are sparse, since all of the strings of length n must be printed in time polynomial in n . Since every logspace computable function is also computable in polynomial time, L-printable sets are also P-printable, and thus are also sparse.

Definition 5 *If \mathcal{C} is a complexity class, then a set, A , is \mathcal{C} -rankable if and only if there is a function r_A computable in \mathcal{C} such that $r_A(x) = |\{y \leq_{lex} x : y \in A\}|$. (In other words, $r_A(x)$ gives the lexicographic rank of x in A .) The function r_A is called the ranking function for A .*

Note that P-rankable sets are necessarily in P but are not necessarily sparse. Furthermore, a set is P-rankable if and only if its complement is P-rankable. Finally, note that any P-printable set is P-rankable.

Definition 6 *If \mathcal{C} is a complexity class, then two sets, A and B , are \mathcal{C} -isomorphic ($A \cong_{\mathcal{C}} B$) if there are total functions f and g computable in \mathcal{C} that are both one-one and onto, such that $f(g(x)) = x$ and $g(f(y)) = y$, and f is a reduction from A to B , and g is a reduction from B to A .*

In order for two sets to be P-isomorphic, they must have similar densities: if one set is sparse and the other is not, then any one-one reduction from the sparse set to the dense set must have super-polynomial growth rate. By the same argument, if one has a super-polynomial gap, the other must have a similar gap.

A lexicographic (or order-preserving) isomorphism from A to B is, informally, a bijection which maps the i th element of A to the i th element of B and maps the i th element of \overline{A} to \overline{B} . Note that in the definition of similar densities, the isomorphisms need not be computable in any particular complexity class. This merely provides the necessary condition on densities, in order for the two sets to be P-isomorphic or L-isomorphic.

Definition 7 *Two sets, A and B , have similar densities if the lexicographic (order-preserving) isomorphisms from A to B and from B to A are polynomial size bounded.*

The notion of printability, or of ranking on sparse sets, can be considered a form of compression. Another approach to compression is found in the study of Kolmogorov complexity; a string is said to have “low information content” if it has low Kolmogorov complexity. We are interested in the space-bounded Kolmogorov complexity class defined by Hartmanis [Ha83].

Definition 8 *Let M_v be a Turing machine, and let f and s be functions on the natural numbers. Then we define*

$$KS_v[f(n), s(n)] = \{w : |w| = n \text{ and } \exists y (|y| \leq f(n) \text{ and } M_v(y) = w \text{ and } M_v \text{ uses } s(n) \text{ space})\}.$$

Following the notation of [AR88], we refer to y as the compressed string, $f(n)$ as the compression, and $s(n)$ as the restoration space. Hartmanis [Ha83] shows that there exists a universal machine M_u such that for all v , $KS_v[f(n), s(n)] \subseteq KS_u[f(n) + c, cs(n) + c]$. We will drop the subscript and let $KS[f(n), s(n)] = KS_u[f(n), s(n)]$.

3 Basic Results

We begin by formalizing some observations from the previous section.

Proposition 9 *If A is L-printable, then A has polynomially bounded density, i.e., A is sparse.*

This follows immediately from the fact that logspace computable functions are P-time computable, so L-printability implies P-printability, and from the observations on P-printable sets.

Proposition 10 ([JK89]) *If A is L-printable, then $A \in L$.*

Proof: To decide $x \in A$, simulate the L-printing function for A with input $1^{|x|}$. As each $y \in A$ is “printed,” compare it, bit by bit, with x . If $y = x$, accept. Because the comparisons can be done using $O(1)$ space, and the L-printing function takes $O(\log |x|)$ space, this is a logspace procedure. \square

Proposition 11 *If A is L-rankable, then $A \in L$.*

Proof: Note that the function $x - 1$ (the lexicographic predecessor of x) can be computed (though not written) in space logarithmic in $|x|$. Since logspace computable functions are closed under composition, $r_A(x - 1)$ can be computed in logspace, as can $r_A(x) - r_A(x - 1) = \chi_A(x)$.
□

Proposition 12 *If A is L-printable, then A is L-rankable.*

Proof: To compute the rank of x , we print the strings of A up to $|x|$ and count the ones which are lexicographically smaller than x . Since A is sparse, by Proposition 9, we can store this counter in logspace. □

Note that we give our own proof of the following theorem from [JK89].

Proposition 13 ([JK89]) *If A is L-printable, then A is L-printable in lexicographically increasing order.*

Proof:

To prove this, we use a variation on selection sort. Suppose the logspace machine M L-prints A . Then we can construct another machine, N , to L-print A in lexicographically increasing order. Note that it is possible to store an instantaneous description of a logspace machine, i.e., the position of the input head, the state, the contents of the worktape, and the character just output, in $O(\log |x|)$ space.

The basic idea is that we store, during the computation, enough information to produce three strings: the most recently printed string (in the lexicographically ordered printing), the current candidate for the next string to be printed, and the current contender. We can certainly store three IDs for M in logspace. Each ID describes the state of M immediately prior to printing the desired string.

In addition to storing the IDs, we must simulate M on these three computations in parallel, so that we can compare the resulting strings bit by bit. If the contender string is greater than the last string output (so it has not already been output) and less than the candidate,

it becomes the new candidate. Otherwise, the final ID of the computation becomes the new contender. These simulated computations do not produce output for N ; when the next string is found for N to print, its initial ID is available, and the simulation is repeated, with output. \square

Using the same technique as in the previous proof, one can easily show the following.

Proposition 14 *If A is L-printable, and $A \cong_{\log} B$, then B is L-printable as well.*

4 L-Printable Sets

We begin this section with a very simple example of a class of L-printable sets.

Proposition 15 ([JK89]) *The tally sets in L are L-printable.*

Proof: Decide whether $1^n \in A$, and if so, print it. \square

One may ask, are all of the L-printable sets as trivial as Proposition 15? We demonstrate in the following sections that every regular language or context-free language which is sparse is also L-printable (see Theorem 19 and Corollary 24.2). We also give an L-printable set which is neither regular nor context-free (see Proposition 25).

4.1 Sparse Regular Languages

We show that the sparse regular languages are L-printable. In order to do so, we give some preliminary results about regular expressions.

Definition 16 ([BEGO71]) *Let r be a regular expression. We say r is unambiguous if every string has at most one derivation from r .*

Theorem 17 ([BEGO71]) *For every regular language L , there exists an unambiguous regular expression r such that $L(r) = L$.*

We should note that even though removal of ambiguity from a regular expression is, in general, PSPACE-complete [SH85], this does not concern us. Theorem 17 guarantees the existence of an unambiguous regular expression corresponding to every regular language, which is sufficient for our needs.

We now define a restricted form of regular expression, which will generate precisely the sparse regular languages. (Note that a similar, although more involved, characterization was given in [SSYZ92].)

Definition 18 *We define a static regular expression (SRE) on an alphabet Σ inductively, as follows:*

1. *The empty expression is an SRE, and defines \emptyset , the empty set.*
2. *If $x \in \Sigma$ or $x = \lambda$ (the empty string), then x is an SRE.*
3. *If s and t are SREs, then st , the concatenation of s and t , is an SRE.*
4. *If s and t are SREs, then $s + t$, the union of s and t , is an SRE.*
5. *If s is an SRE, then s^* is an SRE iff:*
 - a) *s does not contain a union of two SREs; and,*
 - b) *s does not contain any use of the $*$ operator.*

Note the restriction of the $*$ operator in the above definition. I.e., $*$ can only be applied to a string. This is the only difference between SREs and standard regular expressions.

We can alternately define an SRE as a regular expression which is the sum of terms, each of which is a concatenation of letters and starred strings.

Theorem 19 *Let R be an unambiguous regular expression. Then $L(R)$ is sparse iff R is static.*

Proof: We first prove two lemmas about “forbidden” subexpressions.

Lemma 19.1 *Let α, β be non-empty regular expressions such that $S = (\alpha + \beta)^*$ is an unambiguous regular expression. Then there is a constant $k > 0$ such that, for infinitely many n , $L(S)$ contains $2^{\frac{n}{k}}$ strings of length n .*

Proof: Let $u, v \in \Sigma^*$ such that $u \in L(\alpha)$ and $v \in L(\beta)$. Let $k = |u| \cdot |v|$. Because S is unambiguous, there must be at least two strings of length k in $L(S)$, namely $u^{|v|}$ and $v^{|u|}$. So, for any length n such that $n = ik$, $i \geq 1$, there are at least $2^i = 2^{i \cdot \frac{k}{k}} = 2^{\frac{n}{k}}$ strings of length n in $L(S)$. \square

Lemma 19.2 *Let α, β be non-empty regular expressions such that S is an unambiguous regular expression, where S is either of the form $(\alpha^*\beta)^*$ or of the form $(\alpha\beta^*)^*$. Then, there is a constant k such that, for infinitely many n , $L(S)$ contains $2^{\frac{n}{k}}$ strings of length n .*

Proof: Let $u, v \in \Sigma^*$ such that $u \in L(\alpha)$ and $v \in L(\beta)$. Suppose $S = (\alpha^*\beta)^*$. Let $k = |u| \cdot |v| + |v|$. If S is unambiguous, there are at least two distinct strings of length k in $L(S)$, namely, $u^{|v|}v$ and $v^{|u|+1}$. So, for any length n such that $n = ik$, $i \geq 1$, there are at least $2^i = 2^{\frac{n}{k}}$ strings of length n in $L(S)$.

The proof is very similar if $S = (\alpha\beta^*)^*$ is unambiguous. \square

It is clear that unambiguity is necessary for both lemmas. For example, the expression $(a+a)^*$ is not static, but $L((a+a)^*) = L(a^*)$, which is sparse.

Note that if R is the empty expression, the theorem is true, since R is static, and $L(R) = \emptyset$, which is certainly sparse. So, for the rest of the proof, we will assume that R is non-empty.

To show one direction of the theorem, suppose R is not static. Then it contains a subexpression which is either of the form $(\gamma_0(\alpha + \beta)\gamma_1)^*$ or of the form $(\gamma_0\alpha^*\gamma_1)^*$. In the first case, by a small modification to the proof of lemma 19.1, $L(R)$ is not sparse. In the second case, by a similar modification to the proof of lemma 19.2, $L(R)$ cannot be sparse.

Now, suppose R is static. If $R = x$, $x \in \Sigma$, $L(R)$ contains only the string x . If $R = r^*$, where r is either a string of characters or a single character, $L(R)$ can have at most one string of any length.

Suppose $R = r + s$, where r and s are SREs. Let $p_r(n)$ and $p_s(n)$ bound the number of strings in $L(r)$ and $L(s)$, respectively. Then there are at most $p_r(n) + p_s(n)$ strings of length n .

Finally, suppose $R = rs$, where r and s are SREs. Let $p_r(n)$ and $p_s(n)$ bound the number of strings in $L(r)$ and $L(s)$, respectively. Then, the number of strings of length n is:

$$q(n) \leq \sum_{i=0}^n p_r(i) \cdot p_s(n-i).$$

The degree of q is bounded by $1 + \text{degree}(p_r(n)) + \text{degree}(p_s(n))$. By induction on the complexity of R , $L(R)$ is sparse. \square

Note that the second half of the proof does not use unambiguity. Hence, any static regular expression generates a sparse regular language.

Theorem 20 *Let R be an SRE. Then $L(R)$ is L-printable.*

Proof: The idea of the proof is not hard. Effectively, we divide R into terms which are either starred expressions or non-starred expressions. For example, we would divide $0(1 + 0)10(11)^*00(0 + 11)$ into three parts: $0(1 + 0)10$, $(11)^*$, and $00(0 + 11)$. Then, we essentially L-print each term independently, and check to see if the strings generated have the correct length. In our example, to print strings of length 9, we might generate 0110, 11, and 0011, respectively, and check that the combined string is in fact 9 characters long. (In this case, the string is too long, and is not printed.)

Let k be the number of stars that appear in R . Partition R into at most $2k+1$ subexpressions, k with stars, and the others containing no stars.

The machine to L-print $L(R)$ has two types of counters. For each starred subexpression, the machine counts how many times that subexpression has been used. For a string of length n , no starred subexpression can be used more than n times. Each counter for a starred subexpression only needs to count up to n .

Each non-starred subexpression generates only a constant number of strings. Thus, up to $k + 1$ additional counters, each with a constant bound, are needed. (Note that the production may intermix the two types of counters, for instance if $(x^* + y^*)$ occurs.)

The machine uses two passes for each potential string. First, the machine generates a current string, counting its length. If the string is the correct length, it regenerates the string and prints it out. Otherwise, it increments the set of counters, and continues. In this way, all strings of lengths $\leq n$ are generated, and all strings of length n are printed.

Lastly, we need to argue that this is a logspace machine. Each of the at most $2k + 1$ counters must count up to n (for n sufficiently large, say, larger than $|R|$). Thus, the counting can be done in $\log n$ space. In addition, the actual production of a string requires an additional

counter, to store a loop variable. The rest of the computation can be handled in $O(1)$ space, using the states of the machine. Thus, $L(R)$ is L-printable.

Note that this L-printing algorithm may generate some strings in $L(R)$ more than once. To get a non-redundant L-printer, simply modify the program to output the strings in lexicographic order, as in Proposition 13, or use an unambiguous SRE for $L(R)$. \square

Theorem 20 does not characterize the L-printable sets, as we see below.

Proposition 21 *There exists a set S such that S is L-printable and not regular.*

Proof: The language $S = \{0^k 1^k : k \in \mathcal{N}\}$ is L-printable (for any n , we print out $0^{\frac{n}{2}} 1^{\frac{n}{2}}$ only if n is even), but not regular. \square

4.2 Sparse Context-Free Languages

Using the theory of bounded context-free languages we can also show that every sparse context-free language is L-printable.

Definition 22 *A set A is bounded if there exist strings w_1, \dots, w_k such that*

$$A \subseteq (w_1)^* \cdots (w_k)^*$$

Note the similarity between bounded languages and languages generated by SRE's. Note also that every bounded language is sparse.

Ibarra and Ravikumar [IR86] prove the following.

Theorem 23 ([IR86]) *If A is a context-free language then A is sparse if and only if A is bounded.*

Ginsburg [G66, p. 158] gives the following characterization of bounded context-free languages.

Theorem 24 ([G66]) *The class of bounded context-free languages is the smallest class consisting of the finite sets and fulfilling the following properties.*

1. If A and B are bounded context-free languages then $A \cup B$ is also a bounded context-free language.
2. If A and B are bounded context-free languages then $AB = \{xy \mid x \in A \text{ and } y \in B\}$ is also a bounded context-free language.
3. If A is a bounded context-free language and x and y are fixed strings then the following set is also a bounded context-free language.

$$\{x^n a y^n : a \in A \text{ and } n \in \mathcal{N}\}$$

Corollary 24.1 *Every bounded context-free language is L-printable.*

Proof: Every finite set is L-printable. The L-printable sets are closed under the three properties in Theorem 24. \square

Corollary 24.2 *Every sparse context-free language is L-printable.*

This completely characterizes the L-printable context-free languages. However the sparse context-free languages do not characterize the L-printable languages.

Proposition 25 *There exists an L-printable set S such that S is not context-free.*

Proof: The language $S = \{0^n 1^n 0^n : n \in \mathcal{N}\}$ is L-printable, but is not context-free. \square

5 L-Isomorphisms

It is easy to show that two P-printable sets, or P-rankable sets, of similar densities are P-isomorphic. Since the usual proof relies on binary search, it does not immediately extend to L-rankable sets. However, we are able to exploit the sparseness of L-printable sets to show the following.

Theorem 26 *If A and B are L-printable and have similar densities, then A and B are L-isomorphic (i.e., $A \cong_{\log} B$).*

Proof:

For each x , define y_x to be the image of x in the lexicographic isomorphism from A to B . Since A and B are L-printable, they are both sparse. Let $p(n)$ be a strictly increasing polynomial that bounds the densities of both sets. If $x \notin A$, then x is “close” to y_x , in the sense that there are at most $p(|x|)$ strings between them in the lexicographic ordering. (Recall Definition 7.) In fact, for all x , $|y_x| \leq p(|x| + 1)$.

Let $r_A(x)$ be the rank of x in A . If $x \notin A$, then the rank of x in \overline{A} is $x - r_A(x)$. Furthermore, $x - r_A(x) = y_x - r_B(y_x)$, and y_x is the unique element of \overline{B} for which this holds. Note that both $r_A(x)$ and $r_B(y_x)$ can be written in space $O(\log |x|)$. Thus, to compute y_x , we need to compute $x - r_A(x) + r_B(y_x)$. We do so by maintaining a variable d , which is initialized to $r_A(x)$. Counter c is initialized to 0. The following loop is iterated until a counter, c , reaches $p(|x| + 1)$:

1. L-print (in lexicographic order) the elements of B of length c ; for each string that is lexicographically smaller than $(x - d)$, decrement d ;
2. increment c .

Output $x - d$.

Note that, if d is written on the work tape, each bit of $x - d$ can be computed in logspace as needed, and the output of the L-printing function can be compared to $x - d$ in a bit-by-bit manner.

If $x \in A$, since the L-printing function outputs strings in lexicographic order, computing y_x is easy: compute $r_A(x)$, then “L-print” B internally, actually outputting the $r_A(x)^{th}$ string.

Without loss of generality, we can assume that the simulated L-printer for B prints B in lexicographic order. Thus, as soon as the $r_A(x) - 1st$ element of B is printed internally, the simulation switches to output mode.

The following is an overview of the logspace algorithm computing the desired isomorphism.

1. Compute $A(x)$.
2. Compute $r_A(x)$, and write it on a work tape.
3. If $x \in A$, find the $r_A(x)^{th}$ element of B , and output it.

4. If $x \notin A$, find the unique string $y_x \notin B$ such that $x - r_A(x) = y_x - r_B(y_x)$, and output y_x .

□

Using this theorem, we can now characterize the L-printable sets in terms of isomorphisms to tally sets, and in terms of sets of low Kolmogorov space complexity.

Theorem 27 *The following are equivalent:*

1. S is L-printable.
2. S is L-isomorphic to some tally set in L.
3. There exists a constant k such that $S \subseteq KS[k \log n, k \log n]$ and $S \in L$.

Although it is not known whether or not every sparse L-rankable set is L-isomorphic to a tally set (see Theorem 30), we can prove the following lemma, which will be of use in the proof of Theorem 27.

Lemma 27.1 *Let A be sparse and L-rankable. Then there exists a tally set $T \in L$ such that A and T have similar density.*

Proof: Let $A^{\leq n}$ denote the strings of length at most n in A . Let $p(n)$ be an everywhere positive monotonic increasing polynomial such that $|A^{\leq n}| \leq p(n)$ for all n , and such that $p(n) - p(n-1)$ is greater than the number of strings of length n in A . Let $r(x)$ be the ranking function of A . We define the following tally set:

$$T = \{1^{p(|x|-1)+r(x)-r(1^{|x|-1})} : x \in A\}.$$

To show that $T \in L$, notice that of the tally strings 1^i , $p(n-1) < i \leq p(n)$, $1^i \in T$ iff $p(n-1) < i \leq p(n-1) + r(1^i) - r(1^{i-1})$. So, to decide $T(1^m)$, we first find the largest n such that $p(n-1) < m \leq p(n)$. (Note that n can be written in binary in space $O(\log m)$.) Then compute $d_1 = m - p(n-1)$. This difference is bounded by $p(n)$, and thus can be written in logspace. Finally, compute $d_2 = r(1^n) - r(1^{n-1})$ and compare to d_1 . Accept iff $d_1 \leq d_2$.

Finally, we show that T and A have similar density. Let $f : A \rightarrow T$ be the lexicographic isomorphism between T and A . Note that f maps strings of length n to strings of length at

most $p(n)$, so f is polynomially bounded. Note that p is always positive, which implies that f is length-increasing. So, f^{-1} must also be polynomially bounded. Thus, T and A have similar density. \square

The following proof of Theorem 27 is very similar to the proof of the analogous theorem in [AR88].

Proof:

[1 \Rightarrow 2] Let S be L-printable. Then it is sparse and L-rankable. Let T be the tally set guaranteed by Lemma 27.1. By Proposition 15, T is L-printable. Thus, T and S are L-printable, and T and S have similar density. So by Theorem 26, $S \cong_{\log} T$.

[2 \Rightarrow 3] Let S be L-isomorphic to a tally set T , and let f be the L-isomorphism from S to T . Let $x \in S$ be a string of length n . Let $f(x) = 0^r$. Since f is logspace-computable, there exists a constant c such that $r \leq n^c$, i.e., $|r| \leq c \log n$. In order to recover x from r , we only have to compute $f^{-1}(0^r)$. Computing 0^r given r requires $\log n$ space for one counter. Further, there exists a constant l such that computing $f^{-1}(0^r)$ requires at most $lc \log n$ space, since $r \leq n^c$. So, the total space needed to compute x given r is less than or equal to $\log n + lc \log n \leq k \log n$ for some k . Hence, $S \subseteq KS[k \log n, k \log n]$. If $T \in L$, then $S \in L$, since $S \cong_{\log} T$.

[3 \Rightarrow 1] Assume $S \subseteq KS[k \log n, k \log n]$ for some k , and $S \in L$. On input 0^n , we simulate M_u for each string of length $k \log n$. For a given string x , $|x| = k \log n$, we first simulate $M_u(x)$ and check whether it completes in space $k \log n$. If it does, we recompute $M_u(x)$, this time checking whether the output is in S . If it is, we recompute $M_u(x)$, and print out the result. The entire computation only needs $O(\log n)$ space, so S is L-printable.

\square

It was shown in [AR88] that a set has small generalized Kolmogorov complexity if and only if it is P-isomorphic to a tally set. (Note: this was improvement of the result in [BB86], which showed that a set has small generalized Kolmogorov complexity if and only if it is “semi-isomorphic” to a tally set.) Using a similar argument and Theorem 27 we can show an analogous result for sets with small generalized Kolmogorov space complexity. First, we prove the following result.

Proposition 28 *For all M_v and k , $KS_v[k \log n, k \log n]$ is L-printable.*

Proof: To L -print for length n , simulate M_v on each string of length less than or equal to $k \log n$, and output the results. \square

Corollary 29 *There exists a k such that $A \subseteq KS[k \log n, k \log n]$ if and only if A is L -isomorphic to a tally set.*

Proof: Suppose A is L -isomorphic to a tally set. Then, by the argument given in the proof of [2 \Rightarrow 3] in Theorem 27, $A \subseteq KS[k \log n, k \log n]$.

Now, suppose $A \subseteq KS[k \log n, k \log n]$. By Proposition 28 and Theorem 27, $KS[k \log n, k \log n]$ is L -isomorphic to a tally set in L via some L -isomorphism f . It is clear that A is L -isomorphic to $f(A)$. Since $f(A)$ is a subset of a tally set, $f(A)$ must also be a tally set. \square

6 Printability, Rankability and Decision

In this section we examine the relationship among L -printable sets, L -rankable sets and L -decidable sets. We show that any collapses of these classes, even for sparse sets, are equivalent to unlikely complexity class collapses.

Theorem 30 *The following are equivalent:*

1. *Every sparse L -rankable set is L -printable.*
2. *There are no tally sets in $P - L$.*
3. *$E = LinearSPACE$.*

Proof:

[2 \Leftrightarrow 3] This equivalence follows from techniques similar to those of Hartmanis, Immerman and Sewelson [HIS85].

[2 \Rightarrow 1] Suppose A is a sparse L -rankable set. Note that $A \in L$.

Let

$$T = \{1^{(i,j)} : \text{The } i\text{th bit of the } j\text{th string in } A \text{ is } 1\},$$

where

$$\langle i, j \rangle = \frac{(i+j)(i+j+1)}{2} + i.$$

Note that $\langle i, j \rangle$ can be computed in space linear in $|i| + |j|$. Since A is sparse, i and j are bounded by a polynomial in the length of the j th string. Hence, $\langle i, j \rangle$ can be computed using logarithmic space with respect to the length of the j th string.

Given $\langle i, j \rangle$, we can determine i and j in polynomial time, and we can find the j th string of A by using binary search with the ranking function of A . Hence, $T \in \text{P}$. So, by assumption, $T \in \text{L}$.

Next we give a method for printing A in logspace. Given a length n , we compute (and store) the ranks of 0^n and 1^n in A . Let r_{start} and r_{end} be the ranks of 0^n and 1^n , respectively. If $0^n \notin A$, the string with rank r_{start} has length less than n . First, we check to see if $0^n \in A$, and if so, print it. Then, for each j , $r_{start} < j \leq r_{end}$, we output the j th string by computing and printing $T(1^{\langle i, j \rangle})$ for each bit i . This procedure prints the strings of A of length n .

Note that since A is sparse, we can store r_{start} and r_{end} in $O(\log n)$ space. Since $i \leq n$, we can store and increment the current value of i in $\log n$ space.

[1 \Rightarrow 2] Let $T \in \text{P}$ be a tally set. Since the monotone circuit value problem is P-complete (see [GHR95]), there exists a logspace-computable function f and a nondecreasing polynomial p such that $f(n)$ produces a circuit C_n with the following properties.

1. C_n is monotone (i.e., C_n uses only AND and OR gates).
2. C_n has $p(n)$ gates.
3. The only inputs to C_n are 0 and 1.
4. C_n outputs 1 iff 1^n is in T .

We can assume that the reduction orders the gates of C_n so that the value of gate g_i depends only on the constants 0 and 1 and the values of gates g_j for $j < i$ ([GHR95]). Let x_n be the string of length $p(n)$ such that the i th bit of x_n is the value of gate g_i .

Let $A = \{x_n : n \in \mathcal{N}\}$. Then A contains exactly one string of length $p(n)$ for all n , and no strings of any other lengths.

Claim 30.1 *The set A is L -rankable.*

Proof: To prove this claim, let w be any string. In logspace, we can find the greatest n such that $p(n) \leq |w|$. If $p(n) \neq |w|$ then $w \notin A$, and the rank of w is n . Suppose $|w| = p(n)$. Since x_n is the only string of length $p(n)$ in A , the rank of w is $n - 1$ if $w < x_n$, and n otherwise.

Consider the i th bit of w as a potential value for gate g_i in C_n . Let j be the smallest value such that w_j is not the value of g_j . In order to find the value of a gate g_i , we first use $f(n)$ (our original reduction) to determine the inputs to g_i . By the time we consider the i^{th} bit of w , we know that w is a correct encoding of all of the gates g_k such that $k < i$, so we can use those bits of w as the values for the gates. Thus, we can determine the value of g_i and compare it to the i th bit of w . If they differ, we are done. If they are the same, we continue with the next gate. We can count up to $p(n)$ in logspace, so this whole process needs only $O(\log p(n))$ space to compute.

Once j is found, there are three cases to consider.

1. If j doesn't exist then $w = x_n$.
2. If the j th bit of w is 0 then $w < x_n$.
3. If the j th bit of w is 1 then $w > x_n$.

These follow since the i th bit of x_n matches the i th bit of w for all $i < j$. \square

Thus A is L -rankable and, by assumption, L -printable.

So, to determine if 1^n is in T , L -print A for length $p(n)$ to get x_n . The bit of x_n which encodes the output gate of C_n is 1 iff $1^n \in T$. Since every step of this algorithm is computable in logspace, $T \in L$.

This completes the proof of Theorem 30. \square

Corollary 30.2 *There exist two non- L -isomorphic L -rankable sets of the same density unless there are no tally sets in $P - L$.*

Proof: Consider the sets T and A from the second part of the proof of Theorem 30. The set $B = \{1^{p(n)} : n \in \mathcal{N}\}$ has the same density as A . By Proposition 15, B is L -printable. If A and

B were L-isomorphic then by Proposition 14, A would also be L-printable and T would be in L. \square

One may wonder whether every sparse set in L is L-printable or L-rankable. We show that either case would lead to the unlikely collapse of FewP and L. Recall that FewP consists of the languages in NP accepted by nondeterministic polynomial-time Turing machines with at most a polynomial number of accepting paths.

Fix a nondeterministic Turing machine M and an input x . Let p specify an accepting path of $M(x)$ represented as a list of configurations of each computation step along that path. Note that in logarithmic space we can verify whether p is such an accepting computation since if one configuration follows another only a constant number of bits of the configuration change.

We can assume without loss of generality that all paths have the same length and that no accepting path consists of all zero or all ones.

Define the set P_M by

$$P_M = \{x\#p : p \text{ is an accepting path of } M \text{ on } x\}.$$

From the above discussion we have the following proposition that we will use in the proofs of Theorems 34 and 35.

Proposition 31 *For any nondeterministic machine M , P_M is in L.*

Allender and Rubinstein [AR88] showed the following about P-printable sets.

Theorem 32 ([AR88]) *Every sparse set in P is P-printable if and only if there are no sparse sets in FewP - P.*

Allender [A86] also relates this question to inverting functions.

Definition 33 *A function f is strongly L-invertible on a set S if there exists a logspace computable function g such that for every $x \in S$, $g(x)$ prints out all of the strings y such that $f(y) = x$.*

We extend the techniques of Allender [A86] and Allender and Rubinstein [AR88] to show the following.

Theorem 34 *The following are equivalent.*

1. *There are no sparse sets in FewP - L.*
2. *Every sparse set in L is L-printable.*
3. *Every sparse set in L is L-rankable.*
4. *Every L-computable, polynomial-to-one, length-preserving function is strongly L-invertible on $\{1\}^*$,*
5. $\text{FewE} = \text{linearSPACE}$.

Proof:

[1 \Rightarrow 2] Let A be a sparse set in L. Then A is in P. By (1) we have that there are no sparse sets in FewP - P. By Theorem 32, A is P-printable.

Consider the following set B .

$$B = \{1^{(n,i,j,b)} : \text{The } i\text{th bit of the } j\text{th element of } A \text{ of length } n \text{ is } b\}$$

Since A is P-printable then B is in P. By (1) (as B is sparse and in $P \subseteq \text{FewP}$), we have B is in L. Then A is L-printable by reading the bits off from B .

[2 \Rightarrow 3] Follows immediately from Proposition 12.

[3 \Rightarrow 1] Let A be a sparse set in FewP accepted by a nondeterministic machine M with computation paths of length $q(n)$ for inputs of length n .

Consider the set P_M defined as above. Note that P_M is sparse since for any length n , M only accepts a polynomial number of strings with at most a polynomial number of accepting paths each. Also by Proposition 31 we have P_M in L.

By (3) we have that P_M is L-rankable. We can then determine in logarithmic space whether $M(x)$ accepts (and thus x is in A) by checking whether

$$r_{P_M}(x\#0^{q(|x|)}) < r_{P_M}(x\#1^{q(|x|)}).$$

[2 \Rightarrow 4] Let f be a L-computable, polynomial-to-one, length-preserving function. Consider $S = \{y : f(y) \in 1^*\}$. Since S is in L, S is L-printable.

[4 \Rightarrow 2] Let A be a sparse set in L . Define $f(x) = 1^{|x|}$ if x is in A and x otherwise. If g is a strong L -inverse of f on 1^* then $g(1^n)$ will print out the strings of length n of A and 1^n . We can then print out the strings of length n in logspace by printing the strings output by $g(1^n)$, except we print 1^n only if 1^n is in A .

[1 \Leftrightarrow 5] In [RRW94], Rao et al. show that there are no sparse sets in $\text{FewP} - P$ if and only if $\text{FewE} = E$. A straightforward modification of their proofs is sufficient to show that there are no sparse sets in $\text{FewP} - L$ if and only if $\text{FewE} = \text{linearSPACE}$.

□

Unlike L -printability, L -rankability does not imply sparseness. One may ask whether every set computable in logarithmic space may be rankable. We show this equivalent to the extremely unlikely collapse of PP and L .

Theorem 35 *The following are equivalent.*

1. *Every $\#P$ function is computable in logarithmic space.*
2. $L = PP$.
3. *Every set in L is L -rankable.*

Our proof uses ideas from Blum (see [GS91]), who shows that every set in P is P -rankable if and only if every $\#P$ function is computable in polynomial-time. Note that Hemachandra and Rudich [HR90] proved results similar to Blum's.

Proof:

[1 \Rightarrow 2] If A is in PP then there is a $\#P$ function f such that x is in A if the high-order bit of $f(x)$ is one.

[2 \Rightarrow 1] Note that $L = PP$ implies that $P = PP$ implies that $P = P^{PP}$ implies that $P = P^{\#P}$. Thus we have $L = P^{\#P}$ and we can compute every bit of a $\#P$ function in logarithmic space.

[1 \Rightarrow 3] Let A be in L . Consider the nondeterministic polynomial-time machine M that on input x guesses a $y \leq_{lex} x$ and accepts if y is in A . The number of accepting paths of $M(x)$ is a $\#P$ function equal to $r_A(x)$.

[3 \Rightarrow 1] Let f be a #P function. Let M be a nondeterministic polynomial-time machine such that $f(x)$ is the number of accepting computations of $M(x)$. Let $q(n)$ be the polynomial-sized bound on the length of the computation paths of M . Consider P_M as defined above. By Proposition 31 we have that P_M is in L so by (3) P_M is L-rankable. We then can compute $f(x)$ in logarithmic space by noticing

$$f(x) = r_{P_M}(x\#1^{q(|x|)}) - r_{P_M}(x\#0^{q(|x|)}).$$

□

7 Conclusions

The class of L-printable sets has many analogous properties to its polynomial-time counterpart. For example, even without the ability to do binary searching, one can show two L-printable sets of the same density are isomorphic. However, some properties do not appear to carry over: it is very unlikely that every sparse L-rankable set is L-printable.

Despite the strict computational limits on L-printable, this class still has some bite: every tally set in L, every sparse regular and context-free language and every L-computable set of low space-bounded Kolmogorov complexity strings is L-printable.

Acknowledgments

The authors want to thank David Mix Barrington for a counter-example to a conjecture about sparse regular sets, Alan Selman for suggesting the tally set characterization of L-printable sets and Corollary 29, Chris Lusena for proofreading, and Amy Levy, John Rogers and Duke Whang for helpful discussions. The last equivalence of Theorem 34 was suggested by an anonymous referee. The authors would like to thank both anonymous referees for many helpful suggestions and comments.

References

- [AR88] E. Allender and R. Rubinfeld. P-Printable Sets. *SIAM Journal on Computing*, 17(6):1193–1202, 1988.
- [A86] E. Allender. The Complexity of Sparse Sets in P. *Proceedings 1st Conf. on Structural Complexity Theory*, pages 1-11, Springer-Verlag, 1986.
- [BB86] J. Balcazar and R. Book. Sets with Small Generalized Kolmogorov Complexity. *Acta Informatica*, 23:679–688, 1986.
- [BDG88] J. Balcázar, J. Díaz and J. Gabarró. **Structural Complexity I**, Springer, 1988.
- [BEGO71] R. Book, S. Even, S. Greibach and G. Ott. Ambiguity in Graphs and Expressions. *IEEE Transactions on Computers*, Vol. C-20, No. 2, 1971.
- [G66] S. Ginsburg. **The Mathematical Theory of Context-Free Languages**. McGraw Hill, 1966.
- [GS91] A. Goldberg and M. Sipser. Compression and Ranking. *SIAM Journal on Computing*, 20(3):524-536, 1991.
- [GHR95] R. Greenlaw, H.J. Hoover, and W. Ruzzo. **Limits to Parallel Computation: P-Completeness Theory**, Oxford University Press, 1995.
- [Ha83] J. Hartmanis. Generalized Kolmogorov Complexity and the Structure of Feasible Computations. *IEEE Proceedings of 24th Symposium Foundations of Computer Science*, 439–445, 1983.
- [HH88] J. Hartmanis and L. Hemachandra. On Sparse Oracles Separating Feasible Complexity Classes. *Information Processing Letters*, 28:291–295, 1988.
- [HIS85] J. Hartmanis, N. Immerman, and V. Sewelson. Sparse Sets in $NP - P : EXPTIME$ versus $NEXPTIME$. *Information and Control*, 65:158–181, 1985.
- [HY84] J. Hartmanis and Y. Yesha. Computation Times of NP Sets of Different Densities. *Theoretical Computer Science*, 34:17–32, 1984.

- [HR90] L. Hemachandra and S. Rudich. On the Complexity of Ranking. *Journal of Computer and System Sciences*, 41(2):251–271, 1990.
- [IR86] O. Ibarra and B. Ravikumar. On Sparseness, Ambiguity and Other Decision Problems for Acceptors and Transducers. In *Proceedings of the 3rd Symposium on Theoretical Aspects of Computer Science*, volume 210 of *Lecture Notes in Computer Science*, pages 171–179, 1986.
- [JK89] B. Jenner and B. Kirsig, **Alternierung und Logarithmischer Platz.** PhD Thesis, Universität Hamburg, 1989.
- [M91] J. Martin. **Introduction to Languages and the Theory of Computation,** McGraw-Hill, 1991.
- [P94] C. Papadimitriou. **Computational Complexity,** Addison-Wesley, 1994.
- [RRW94] R.P.N. Rao, J. Rothe and O. Watanabe. Upward separation for FewP and related classes. *Information Processing letters*, Vol. 52, 1994, pages 175–180.
- [R86] R. Rubinfeld. *A note on sets with small generalized Kolmogorov complexity,* Tech. Report TR 86-4, Iowa State University, Ames, IA, March 1986.
- [SSYZ92] J. Shallit, A. Szilard, S. Yu and K. Zhang. Characterizing Regular Languages with Polynomial Densities, *Proceedings of the 17th International Symposium on Mathematical Foundations of Computer Science*, pages 494–503, 1992.
- [S83] M. Sipser. A complexity theoretic approach to randomness, *Proceedings of the 15th ACM Symposium on Theory of Computing*, 1983, pages 330–335.
- [SH85] R.E. Stearns and H.B. Hunt III. On the Equivalence and Containment Problems for Unambiguous Regular Expressions, Regular Grammars and Finite Automata. *SIAM J. Comput.* 14(3):598-611, 1985.