NL-printable sets and Nondeterministic Kolmogorov Complexity

Eric Allender [1,2]

Dept. of Computer Science
Rutgers University
New Brunswick, NJ, USA

Abstract

This paper introduces *nondeterministic* space-bounded Kolmogorov complexity, and we show that it has some nice properties not shared by some other resource-bounded notions of K-complexity.

P-printable sets were defined by Hartmanis and Yesha and have been investigated by several researchers. The analogous notion of L-printable sets was defined by Fortnow et al; both P-printability and L-printability were shown to be related to notions of resource-bounded Kolmogorov complexity. NL-printability was defined by Jenner and Kirsig, but some basic questions regarding this notion were left open. In this paper we answer a question of Jenner and Kirsig by providing a machine-based characterization of the NL-printable sets.

1 Introduction

By definition, machines with small space bounds have limited memory. In particular, they cannot remember where they have been, in the sense that a (nondeterministic) logspace-bounded machine that is searching a graph cannot in general remember the nodes that have been visited, and it cannot always reproduce the exact path that led it to the current node.

In this paper we present a simple trick that sometimes allows NL machines to perform feats of memory. Stated another way, we show that short descriptions are often sufficient for NL machines to reproduce large objects of interest. Although the technique is not really new – it is nearly two decades old, and was used again recently to prove results about time-bounded Kolmogorov complexity [BFL02] – it seems that its usefulness in NL is not as widely known as it should be.

[1] Supported in part by NSF grant CCR-0104823.
[2] allender@cs.rutgers.edu

A more general goal of this paper is to examine different notions of space-bounded Kolmogorov complexity and present some applications of these notions.

The original goal of this work was to improve our understanding of nondeterministic logspace (NL). Thus, before we introduce space-bounded Kolmogorov complexity, let us review the relevant background about NL.

2 Preliminaries, and some Motivation

Many of the observations in this paper are motivated by the desire to prove a collapse of some complexity classes between NL and UL. (UL is "unambiguous" logspace; more formal definitions appear below.) It was observed in [ARZ99] that the nonuniform collapse NL/poly = UL/poly of [RA00] holds also in the uniform case under a very plausible hypothesis. Namely, NL = UL if there is a set in DSPACE(n) that has exponential "hardness" in the sense of [NW94]. More recently, it has been pointed out by [KvM02] that this same conclusion can be weakened to a worst-case circuit lower bound. That is, NL = UL if there is a set in DSPACE(n) (such as SAT, for example) that requires circuits (or even branching programs) of size $2^{\epsilon n}$, for some $\epsilon > 0$.

So almost certainly it is the case that NL and UL are equal, and thus all of the various complexity classes *between* NL and UL are certainly equal, and thus *surely* it should be possible to actually *prove* (unconditionally) that some of these classes coincide in the uniform setting. There are *several* classes that were defined in [BJLR91] that lie between NL and UL, but unfortunately this paper cannot present any new collapse among these classes. Nonetheless, it will be necessary for the reader to know what some of these classes are, and thus we have the following list of definitions.

For a nondeterministic Turing machine M, the function $\#acc_M : \{0,1\}^* \to \mathbb{N}$ is defined so that $\#acc_M(x)$ is the number of accepting computations of M on input x. The reader is assumed to be familiar with deterministic and nondeterministic logspace (L and NL, respectively). UL is the class of languages accepted by NL machines M that satisfy the restriction that, for all x, $\#acc_M(x) \leq 1$. FewL is the class of languages[3] accepted by NL machines M that satisfy the restriction that, for all x, $\#acc_M(x) = |x|^{O(1)}$.

We will also need to consider space bounds other than logarithmic; in particular we will be interested in linear space bounds. The reader should be familiar with DSPACE(n) and NSPACE(n), and can surely guess what USPACE(n) is. FewSPACE(n) is the class of languages in NSPACE(n) ac-

[3] Here we are using the name that was used by [BJLR91] to refer to this class. A possible point of confusion is that this same class was called FewNL in [AR98]. The name FewNL was originally used by [BDHM91] to refer to a related class that is called FewUL by [BJLR91]. The *interested* reader is referred to [BJLR91] for definitions; we will not need to refer further to those classes here, and hence we omit the definitions. (The *disinterested* reader can simply remember that all of these classes are almost certainly just different names for NL.)

cepted by machines M that satisfy the restriction that, for all x, $\#acc_M(x) = 2^{O(|x|)}$. In the likely case that NL = UL, it follows that USPACE(n) = FewSPACE(n) = NSPACE(n). Conceivably, proving equality at the linear-space level could be *easier* than proving equality of the corresponding logspace classes.

One other subclass of NL that needs to be mentioned is RL (randomized logspace); a language A is in RL if and only if there is a nondeterministic logspace machine accepting A and making a nondeterministic choice on each step, with the additional property that if $x \in A$ then *at least half* of the sequences of nondeterministic choices lead to an accepting state. The class RSPACE(n) is defined analogously. Just as it is conjectured that UL = NL, there is a popular conjecture that RL = L. (For example, see [Sak96].) This would imply RSPACE(n) = DSPACE(n).

We also need a logspace-analog of the complexity class Few of [CH90]: the class LFew (which was called LogFew in [AR98]) is the set of all languages A such that there is an NL machine M with the property that for all x, $\#acc_M(x) = |x|^{O(1)}$, and there is a language $B \in$ L such that $x \in A$ if and only if $(x, \#acc_M(x)) \in B$. It is not immediately obvious that LFew is contained in NL. This containment was shown first in the nonuniform setting in [AR98], and then in [AZ98] a derandomization argument was used to show LFew$\subseteq$ NL. Shortly thereafter, a very simple hashing argument was used in [ARZ99] to prove this same inclusion. It is this same simple hashing argument that will be used over and over again in this note. It relies on the following fact:

Theorem 2.1 *([FKS82][Lemma 2], [Meh82][Theorem B]) Let S be a set of $n^{O(1)}$ n-bit strings (viewed as n-bit numbers). There is some prime number p with $O(\log n)$ bits such that for any $x \neq y$ in S, $x \not\equiv y (\mathrm{mod} p)$.*

3 Nondeterministic Kolmogorov Complexity

The basic theory of Kolmogorov complexity (see, for example [LV97]) yields a very nice measure of the "randomness" of a string x, but it suffers from the defect that this measure is not computable. This has motivated several different approaches to the task of defining resource-bounded versions of Kolmogorov complexity. (Again, a good survey of this material can be found in [LV97].) The approach that we will follow is based on a definition of Levin [Lev84] as extended and adapted to other complexity measures in [All01,ABK$^+$02,AKRR03].

First, we present (an equivalent restatement of) Levin's Kt measure, along with the deterministic time- and space-bounded Kolmogorov measures KT and KS of [All01,ABK$^+$02], as reformulated in [AKRR03].

Definition 3.1 Let U be a deterministic Turing machine.

$$\mathrm{Kt}_U(x) = \min\{|d| + \log t \;:\; \forall b \in \{0,1,*\} \forall i \leq n+1 \; U(d,i,b) \text{ runs in}$$

$$\text{time } t \text{ and accepts iff } x_i = b\}$$

$$\mathrm{KS}_U(x) = \min\{|d| + s : \forall b \in \{0,1,*\} \forall i \leq n+1 \; U(d,i,b) \text{ runs in space } s \text{ and accepts iff } x_i = b\}$$

$$\mathrm{KT}_U(x) = \min\{|d| + t : \forall b \in \{0,1,*\} \forall i \leq n+1 \; U(d,i,b) \text{ runs in time } t \text{ and accepts iff } x_i = b\}$$

Here, we say that $x_i = *$ if $i > |x|$.

As usual, we will choose a fixed "optimal" Turing machine U and use the notation Kt, KS, and KT to refer to $\mathrm{Kt}_U, \mathrm{KS}_U$, and KT_U. However, the definition of "optimal" Turing machine depends on the measure under consideration. For instance, U is Kt-*optimal* if for any Turing machine U' there exists a constant $c \geq 0$ such that for all x, $\mathrm{Kt}_U(x) \leq \mathrm{Kt}_{U'}(x) + c \log |x|$. Notice that there is an additive logarithmic term instead of the "usual" additive constant. This comes from the slight slow-down that is incurred in the simulation of U' by U. Similarly, U is KS-*optimal* if for any Turing machine U' there exists a constant $c > 0$ such that for all x, $\mathrm{KS}_U(x) \leq c\mathrm{KT}_{U'}(x)$, and U is KT-*optimal* if for any Turing machine U' there exists a constant $c > 0$ such that for all x, $\mathrm{KT}_U(x) \leq c\mathrm{KT}_{U'}(x) \log \mathrm{KT}_{U'}(x)$. The existence of optimal machines for Kt, KS and KT complexity follows via standard arguments. The definition of KT can be relativized to yield a measure KT^A by providing U with access to oracle A. Part of the motivation for the KT measure comes from the fact that if x is a string encoding the truth-table of a Boolean function f, then the minimum circuit size of f (on circuits with oracle A) is polynomially-related to $\mathrm{KT}(x)$ (respectively to $\mathrm{KT}^A(x)$). Also, there are optimal machines such that, for any languages A and B complete for $\mathrm{DTIME}(2^n)$ and $\mathrm{DSPACE}(n)$, respectively, it holds that

- $\mathrm{Kt}(x) + \log |x| = \Theta(\mathrm{KT}^A(x) + \log |x|)$.
- $\mathrm{KS}(x) + \log |x| = \Theta(\mathrm{KT}^B(x) + \log |x|)$.

Now, following the model of [AKRR03], let us introduce a nondeterministic analog of KS complexity.

Definition 3.2 Let U be a fixed nondeterministic Turing machine.

$$\mathrm{KNS}_U(x) = \min\{|d| + s : \forall b \in \{0,1,*\} \forall i \leq n+1 \; U(d,i,b) \text{ runs in space } s \text{ and accepts iff } x_i = b\}$$

As above, we define KNS as KNS_U, such that for all U', we have $\mathrm{KNS}_U(x) \leq c \cdot \mathrm{KNS}_{U'}(x)$ for some constant c.

One of the first types of resource-bounded Kolmogorov complexity to be studied was "distinguishing" complexity. For more on the history of this notion, see [BFL02]. In [AKRR03] a version of distinguishing complexity was introduced that has the same flavor as Levin's Kt measure:

Definition 3.3 Let U be a deterministic Turing machine. Define $\mathrm{KDt}_U(x)$ to be min $\{|d| + \log t : \forall y \in \Sigma^{|x|} \; U(d,y) \text{ runs in time } t \text{ and accepts iff } x = y\}$

Again, we have to be careful about the properties we require of the optimal Turing machine. We define KDt as KDt_U, such that for all U', we have $\mathrm{KDt}_U(x) \le \mathrm{KDt}_{U'}(x) + c\log|x|$ for some constant c. Note that in fact we can assume without loss of generality that this machine U *has only one-way access to its input* y. For our space-bounded versions of distinguishing complexity, we will need to impose this restriction. We emphasize this restriction on the way we access our input by adding an "arrow" to our notation.

Definition 3.4 Let U_1 be a fixed nondeterministic Turing machine, and let U_2 be a fixed deterministic Turing machine. We consider only Turing machines with two input tapes (one containing d and one containing y), where the machines *have only one-way access to the tape containing* y.

$$\begin{aligned}\mathrm{KN\vec{D}S}_{U_1}(x) = \min\{|d| + s \,:\, \forall y \in \Sigma^{|x|}\ & U_1(d,y) \\ & \text{runs in space } s \text{ and accepts iff } x = y\} \\ \mathrm{K\vec{D}S}_{U_2}(x) = \min\{|d| + s \,:\, \forall y \in \Sigma^{|x|}\ & U_2(d,y) \\ & \text{runs in space } s \text{ and accepts iff } x = y\}\end{aligned}$$

The first important observation is that several of these definitions are essentially equivalent to each other.

Proposition 3.5 *The following functions are in the same Θ-equivalence class. Thus they are more-or-less interchangeable (and in the rest of the paper we will refer primarily to* KNS*).*

- $\mathrm{KT}^A(x) + \log|x|$ *where A is any set complete for* NSPACE(n) *under linear-time reductions.*[4]
- $\mathrm{KNS}(x) + \log|x|$.
- $\mathrm{KN\vec{D}S}(x) + \log|x|$.

Although this proposition is quite easy to prove, it is worth observing that none of the other resource-bounded Kolmogorov complexity measures studied in [All01,ABK$^+$02,AKRR03] are known to enjoy similar properties. For instance, although Kt is roughly the same thing as KT^A for a language A complete for E, it is observed in [AKRR03] that Kt and KDt are likely to be quite different. Similarly, although [AKRR03] observes that distinguishing complexity coincides with time-bounded K-complexity in the nondeterministic setting, it is not known how to capture this notion in terms of KT^A relative to any oracle A (primarily because nondeterministic time classes are not known to be closed under complement).

[4] Although we do *not* know how to guarantee that there is a universal machine U for KT complexity that can simulate all other machines U' with at most linear slow-down, it is easy to show that, for *any* machine U' and any set complete for NTIME(n) under linear-time reductions, $\mathrm{KT}^A_{U'}(x)$ can be bounded by $\mathrm{KNS}(x) + \log|x|$, and there *exist* machines U such that $\mathrm{KNS}(x)$ can be bounded by $\mathrm{KT}^A_U(x) + \log|x|$; hence linear slow-down *can* be achieved with such an oracle A; without loss of generality we use such a machine U in defining KT^A.

It follows easily from Savitch's theorem that KS and KNS are polynomially related.

Proposition 3.6 $\mathrm{KNS}(x) = O(\mathrm{KS}(x))$ *and* $\mathrm{KS}(x) = O((\mathrm{KNS}(x) + \log|x|)^2)$.

On the other hand, the question of whether $\mathrm{DSPACE}(s(n))$ is equal to $\mathrm{NSPACE}(s(n))$ is essentially the question of how close KNS and KS are.

To make the connection between Kolmogorov complexity and the DSPACE vs. NSPACE question more explicit, we introduce 1-L and 1-NL computation, and some measures of the Kolmogorov complexity of a *language*.

Definition 3.7 1-L (1-NL) is the class of languages accepted by (nondeterministic) logspace machines where the input head moves only from left to right. (That is, the machine has a *one-way* input head.)

Proposition 3.8 *Let A be a language in* $\mathrm{NSPACE}(n)$ *accepted by a nondeterministic machine M running in time c^n. Let CompM be the language $\{w : |w| = c^x$ such that M accepts x along the path given by the sequence of nondeterministic choices $w\}$. Then CompM is in* 1-L.

Definition 3.9 Let A be a language and let $\mathrm{K}\mu$ be a Kolmogorov complexity measure. We define two measures of the Kolmogorov complexity of A:

$$\mathrm{K}\mu_A(n) = \min\{\mathrm{K}\mu(x) \;:\; |x| = n \text{ and } x \in A\}$$
$$\mathrm{K}\mu^A(n) = \max\{\mathrm{K}\mu(x) \;:\; |x| = n \text{ and } x \in A\}$$

If $A \cap \Sigma^n = \emptyset$ then $\mathrm{K}\mu_A(n)$ and $\mathrm{K}\mu^A(n)$ are undefined.

The following observations are easy to prove. They are stated here merely to provide some motivation for the preceding definitions. Later in the paper we will add some more conditions to these lists of equivalent statements.

Proposition 3.10 $\mathrm{NSPACE}(n) = \mathrm{DSPACE}(n)$ *if and only if for every* $A \in$ 1-L, $\mathrm{KS}_A(n) = O(\log n)$.

Proposition 3.11 $\mathrm{DSPACE}(n) = \mathrm{USPACE}(n)$ *if and only if for all 1-sparse sets*[5] $A \in$ 1-L, $\mathrm{KS}_A(n) = O(\log n)$.

Note that it is immediate that for every 1-sparse set $A \in$ 1-L, $\mathrm{K}\vec{\mathrm{D}}\mathrm{S}_A(n) = O(\log n)$. Recall also that the conjectured equality NL = UL implies that all of the preceding conditions are equivalent.

Let us mention one additional preliminary observation.

Proposition 3.12 *If* $\mathrm{KS}_A(n) = O(\log n)$ *for every dense*[6] $A \in$ 1-L, *then* $\mathrm{RSPACE}(n) = \mathrm{DSPACE}(n)$.

The hypothesis of Proposition 3.12 is *very* likely to be true; as already mentioned, [KvM02] presents a likely condition (that there is a set in $\mathrm{DSPACE}(n)$

[5] A set is 1-sparse if it contains at most one string of any given length.

[6] A language is *dense* if, for each n, A contains at least half of the strings of length n or no strings of length n.

that requires branching programs of size $2^{\epsilon n}$) that implies that every dense language in $A \in$ L/poly has $\mathrm{KS}_A(n) = O(\log n)$. This is *much* stronger than the hypothesis of Proposition 3.12, allowing *nonuniform* computations and *two-way* access to the input.

Sets in 1-L and 1-NL are simple enough that we *are* able to say something nontrivial about their Kolmogorov complexity. This is where we use the hashing lemma.

Theorem 3.13 *Let $A \in$ 1-NL. Then $\mathrm{KNS}^A(n) = O(\log |A^{=n}| + \log n)$ and $\mathrm{KNS}_A(n) = O(\log n)$.*

Observe that these bounds are essentially optimal (up to constant factors).

Proof. Let $A \in$ 1-NL, accepted by machine M. Let $m = |A^{=n}|$. Let $B = \{x0^{m-n} : x \in A\}$. By Theorem 2.1 there is a prime p of $O(m)$ bits such that all of the strings in B (and hence all of the strings in $A^{=n}$) are equivalent to different values mod p. Given as a description (p, j, m, n, M) (of length $O(\log |A^{=n}| + \log n)$) and given access to a string y on a one-way input tape, a nondeterministic machine can simulate the computation of the 1-NL machine M on input y, simultaneously computing y mod p, and accepting if and only $M(y)$ accepts and y is equivalent to j mod p. Thus for any string $x \in A^{=n}$, $\mathrm{KN}\vec{\mathrm{D}}\mathrm{S}(x) = O(\log |A^{=n}| + \log n)$. The first claim now follows by Proposition 3.5.

For the second claim, observe first that the language $\{(n, C) :$ configuration C appears on the lexicographically first accepting computation path of M on an input of length $n\}$ can be accepted by a nondeterministic machine in space linear in $|(n, C)|$. (That is, starting at the initial configuration, check for each successor configuration in turn if it is the *first* such configuration that appears on an accepting path; use the fact that NSPACE(n) is closed under complementation.) Now observe that the language $\{(n, i, b) :$ along the lexicographically first accepting configuration on an input of length n, the ith input symbol that is consumed is a $b\}$ is also in NSPACE(n). This clearly shows that $\mathrm{KT}^B(x) = O(\log n)$ for some $x \in A^{=n}$ and some $B \in$ NSPACE(n). The second claim now follows by Proposition 3.5. $\square$

The proof of the first assertion in Theorem 3.13 does not make essential use of nondeterminism. A similar proof shows:

Proposition 3.14 *Let $A \in$ 1-L. Then $\mathrm{K}\vec{\mathrm{D}}\mathrm{S}^A(n) = O(\log |A^{=n}| + \log n)$.*

4 NL-Printability

NL-printability was defined and studied in [JK89] as a generalization of the P-printable sets that were defined in [HY84] and further studied in [AR88] and elsewhere. The related notion of L-printability has also been studied [JK89,FGLM99]. In general, for a complexity class $\mathcal{C}$, a language A is $\mathcal{C}$-

printable if there is a function f computable in $\mathcal{C}$ (blurring temporarily the distinction between a class of languages and a class of functions) with the property that $f(0^n)$ is a list of all of the strings in A that have length at most n. For the cases $\mathcal{C} \in \{\mathrm{P}, \mathrm{L}, \mathrm{NL}\}$, this notion is fairly robust to minor changes in the definition (such as having the function f list only the strings of length *exactly* n, listing the elements in lexicographical order, etc.)

Certainly all P-printable sets are sparse, but it seems as if not all sparse sets in P are P-printable. Indeed, there are sparse sets in AC^0 that are not P-printable if and only if FewE $\neq$ E [AR88,RRW94].

When $\mathcal{C}$ is one of {L,P}, it is fairly obvious what is meant by "f is computable in $\mathcal{C}$". However, the reader might be less clear as to what is meant by "f is computable in NL". As it turns out, essentially all of the reasonable possibilities are equivalent, including:

(i) f is computed by a logspace machine with an oracle from NL.

(ii) f is computed by an NC^1 circuit with oracle gates for a language in NL.

(iii) The set $\{(x, i, b) :$ the i^{th} bit of $f(x)$ is $b\}$ is in NL.

Hence NL-printability is the same as $\mathrm{L}^{\mathrm{GAP}}$-printability, where GAP (the Graph Accessibility Problem) is the standard NL-complete set, and L^A-printability is the notion that was studied in [FGLM99], relativized to oracle A.

P-printability and L-printability can be characterized in terms of small time- and space-bounded Kolmogorov complexity. For instance, although it is not stated this way in [FGLM99], A is L-printable if and only if $A \in \mathrm{L}$ and $\mathrm{KS}^A(n) = O(\log n)$. Later in this section we give a similar characterization of NL-printability in terms of KNS-complexity.

A machine-based characterization of the P-printable sets was presented in [AR88]; A is P-printable if and only if A is sparse and is accepted by a one-way (deterministic or nondeterministic) logspace-bounded AuxPDA. (See [AR88] for definitions.) No machine-based characterization of the L-printable sets was presented in [FGLM99], and the results of this section partially explain why. A machine-based characterization of the NL-printable sets was attempted in [JK89], but only a partial characterization was acheived. (It was shown in [JK89] that all NL-printable sets are accepted by 1-NL machines, but it was left open if all sparse sets accepted by 1-NL machines are NL-printable. It was shown only that such sets accepted by 1-UL machines are NL-printable.) The main result of this section is the presentation of a machine-based characterization of the NL-printable sets.

Theorem 4.1 *The following are equivalent:*

- *A is* NL-*printable.*
- *A is* NL-*isomorphic to a tally set in* NL.
- *$A \in \mathrm{NL}$ and $\mathrm{KNS}^A(n) = O(\log n)$.*
- *A is sparse and is accepted by a* 1-NL *machine.*

Proof. The first two conditions can be shown to be equivalent using the related proof in [FGLM99]. If $A \in$ NL and $\mathrm{KNS}^A(n) = O(\log n)$, then A is NL-printable because we can try all of the small descriptions d and check that the description really is a valid description (i.e., for each i there is exactly one b such that $U(d, i, b)$ accepts), and then determine what string is described by d. Similarly, if A is NL-printable, then (n, j) is a short description of the j-th string of length n produced by the printing routine, and hence $\mathrm{KNS}^A(n) = O(\log n)$. As stated above, one of the remaining implications was shown in [JK89]. Thus it suffices to show that if A is sparse and is accepted by a 1-NL machine M, then A is NL-printable. However, this is immediate from Theorem 3.13. □

Theorem 4.1 causes us to pose three simple questions:

(1) Can the second condition be improved to show that NL-printable sets are L-isomorphic to tally sets in NL? This seems unlikely, since it implies that the elements have small KS complexity, and (as in the proof of Theorem 4.2 below) it follows that $\mathrm{DSPACE}(n) = \mathrm{FewSPACE}(n)$.

(2) Can the second condition be improved to show that NL-printable sets are NL-isomorphic to a tally set in L? This seems unlikely, although certainly for "dense enough" NL-printable sets, they are NL-isomorphic to 0^* (which certainly qualifies as a tally set in L). This can be shown via a straightforward modification of a theorem in [FGLM99], to show that if two NL-printable sets have "similar density" (as defined in [FGLM99]), then they are NL-isomorphic. However, if we consider a tally set $A \in \mathrm{NSPACE}(2^{2^n})$ accepted by a machine M running in time, say, $2^{2^{2^n}}$), and consider the related set $A' = \{y : |y| = 2^{2^{2^n}}$ and y encodes a sequence of guesses of M encoding an acceping computation on input $0^n\}$ then note that A' is in 1-NL, and thus is NL-printable. If there were a tally set T in L isomorphic to A', then A would be in $\mathrm{DSPACE}(2^{2^n})$, since a deterministic machine on input 0^n could look to see if there is any element of T having length between $2^{(2^{2^n})/k}$ and $2^{k2^{2^n}}$. Thus any such improvement would imply an unlikely collapse of very large complexity classes.

(3) It is natural to wonder if perhaps all sparse sets in 1-L are L-printable. This also seems unlikely:

Theorem 4.2 *The following are equivalent:*

(i) *All sparse sets $A \in$ 1-L are L-printable (i.e., $\mathrm{KS}^A(n) = O(\log n)$).*

(ii) *All sparse sets in 1-FewL are L-printable.*

(iii) *All sparse sets in 1-FewL are in L.*

(iv) $\mathrm{DSPACE}(n) = \mathrm{FewSPACE}(n)$.

Remark: The condition that $\mathrm{KS}(x) = O(\mathrm{K\vec{D}S}(x) + \log |x|)$ implies all of the conditions in this theorem, but appears to be slightly stronger. It is equivalent to the condition that for *every* language $A \in \mathrm{NSPACE}(n)$ there is a deterministic linear-space procedure that finds an accepting computation for those inputs on which there are few (or even only *one*) accepting paths.

Proof. (ii) trivially implies (i) and (iii). Let us show (i) $\Rightarrow$ (ii), (iii) $\Rightarrow$ (iv), and (iv) $\Rightarrow$ (i).

(i) $\Rightarrow$ (ii): Let A be a sparse set in 1-FewL, accepted by M. Let B be the set of all strings encoding sequences of configurations of an accepting computation of M. By assumption, B is sparse, and is in 1-L, and thus by hypothesis B is L-printable. Now A is L-printable via a routine that first prints the elements of B, and then extracts, from the sequence of configurations, the strings of A that are accepted by M.

(iii) $\Rightarrow$ (iv): This is immediate from standard padding techniques [Boo74].

(iv) $\Rightarrow$ (i): Here again we use the hashing technique. Let A be a sparse set in 1-L, let B be the set $\{1^n0^p1^j$: there are at least j numbers $i_1, \ldots, i_j$ such that there exist words $x_1 \equiv i_1(\text{mod}p), \ldots, x_j \equiv i_j(\text{mod}p)$ of length n in $A\}$, and let C be the set $\{0^n1^p0^i1^kb :$ there is a string x in $A^{=n}$ with $x \equiv i(\bmod p)$, where the k^{th} bit of x is $b\}$. It is easy to see that B and C are tally sets in FewL, and by hypothesis all such sets are in L. Now we can L-print A by, on input 0^n, finding a "good" p, and then cycling through all i's until each x has been printed. □

5 Upward Separation

Theorem 4.2 has the same general flavor of the "upward separation" results of [Har83,HIS85] (see also [Gla01,RRW94]). Upward separation results are of the form "$\mathcal{C}_1 - \mathcal{C}_2$ has no tally sets" if and only if "$\mathcal{C}_1 - \mathcal{C}_2$ has no sparse sets".

Here are a couple more results with a similar flavor to Theorem 4.2. The proofs follow along similar lines.

Theorem 5.1 *The following are equivalent:*

(i) DSPACE(n) = NSPACE(n).

(ii) *All sparse sets in* 1-NL *are in* L.

(iii) *All sparse sets in* 1-NL *are* L-*printable*

(iv) *For all* $A \in$ 1-L, $\text{KS}_A(n) = O(\log n)$.

Theorem 5.2 *The following are equivalent:*

(i) DSPACE(n) = USPACE(n).

(ii) *All 1-sparse sets in* 1-UL *are in* L.

(iii) *All 1-sparse sets in* 1-UL *are* L-*printable.*

(iv) *All 1-sparse sets in* 1-L *are* L-*printable.*

(v) *For all 1-sparse* $A \in$ 1-L, $\text{KS}_A(n) = \text{KS}^A(n) = O(\text{K}\vec{\text{D}}\text{S}_A(n) + \log n)$.

Again, please note that, in the likely case that NL = UL, *all* of the conditions in the preceding three theorems are equivalent.

6 OptL

The class OptL was defined in [AJ93] to be the class of functions f such that there is an NL-transducer M with the property that $f(x)$ is the lexicographically largest string produced by M along any accepting computation path on input x. It is known that OptL is contained in AC^1 [ÀJ95], and the question is raised in [RA00] if perhaps OptL is equal to FNL (the class of functions computable in NL). The following takes care of an easy special case.

Theorem 6.1 *Let f be a function in* OptL *with the property that there is an* NL *transducer realizing f that produces at most $n^{O(1)}$ distinct outputs for any string x of length n. Then f is in* FNL.

Proof. Again, we use the hashing technique. The set $\{(x,p,i) :$ there is an output of $M(x)$ that is equivalent to i mod $p\}$ is easily seen to be in NL. An NL machine can, on input x, find a "good" prime p, and then compare, for given i and j, the individual bits of output strings y_i and y_j that are produced by $M(x)$ that are equivalent to i and j (mod p). In this way, it can determine the lexicographically largest output of M on input x. □

7 Promise Problems

Lacking a proof of NL = UL, we have considered the "easier" problem of DSPACE(n) = USPACE(n), and as well as the problem of whether L = NL is equivalent to L = UL, or even whether L = FewL is equivalent to L = UL. Although we lack even a proof of this latter (modest) conjecture, we can prove that if L contains a solution to the Unique-GAP problem, then L = FewL (and in fact L = LFew). This is a direct logspace analogue to the fact (proved in [BG92]) that if P contains a solution to the Unique-SAT promise problem, then P = Few. Again, we use the hashing technique.

A solution to the Unique-GAP promise problem is a language A that:

- contains all instances (G, s, t) such that G is a directed acyclic graph with exactly one path from s to t, and
- contains no instances (G, s, t) such that G is a directed acyclic graph with no path from s to t.

If G contains more than one path from s to t, then A may or may not contain (G, s, t).

Observe that the "minimal" solution to the Unique-GAP promise problem (i.e., the language consisting of all triples (G, s, t) such that there is exactly one path from s to t in G) is complete for NL [Lan97]. Of course, there are also nonrecursive solutions to the Unique-GAP promise problem. Although the Unique-GAP problem is the obvious graph-theoretic characterization of UL, it is not known if UL contains any language that is a solution to the Unique-GAP promise problem. Even if UL has a complete set (and we cannot prove that it has a complete set), the existence of such a complete set is not known to imply the existence of a set in UL that is a solution to the Unique-GAP promise problem.

Although it is not known if LFew is contained in L^{UL}, something similar *is* known to happen. Let $\mathrm{L}^{\mathrm{PromiseUL}}$ denote the class of languages A with the property that there is a logspace-bounded oracle Turing machine M such that for *any* solution B to the Unique-GAP promise problem, M^B accepts A.

Theorem 7.1 *LFew is contained in* $L^{\mathrm{PromiseUL}}$.

Proof. Let A be a language in LFew. (That is, there is an NL machine M with the property that for all x, $\#acc_M(x) = |x|^{O(1)}$, and there is a language $B \in \mathrm{L}$ such that $x \in A$ if and only if $(x, \#acc_M(x)) \in B$. Let C be a solution to the Unique-GAP promise problem. We define a machine accepting A that uses C as its oracle (and that will also accept A given any other solution C').

On input x, search through all primes p of $O(\log n)$ bits (where the constant in the "big Oh" depends on the language A) to find a prime p that maximizes the value i for which the following is true:

> There are at least i values $j_1 < \ldots < j_i$ such that there exists an accepting computation of $M(x)$ that is equivalent to each of these i residues mod p, and furthermore, for each configuration α of M and for each j, if α is on an accepting path of $M(x)$ that is equivalent to j mod p, then there is a successor of α that lies on such a path.

Note that for a "good" prime p, there is a unique way to guess these i residues and a unique path for each residue, and thus once our logspace oracle machine locates a "good" p it will be able to verify that p is good using only queries to the part of C that satisfy the promise. (That is, since the condition above can be tested in NL, the standard reduction to GAP allows us to test the condition using queries to GAP. Since, for a "good" p the condition can be tested by an NL machine with a unique accepting path, this can be tested using queries to GAP that satisfy the promise.)

Once a good prime p has been found, it is clear that $\#acc_M(x)$ can be computed, and thus membership in A can be determined. $\square$

The preceding theorem has somewhat the same flavor as the result of [BF99] regarding "promise RP" – although the analogy is not strong. Although we are unable to show that L = UL implies L = LFew, this does seem like a small step in that direction.

8 Conclusion

For any NL machine M and input x, the lexicographically largest (or smallest) accepting path of M on x can be found and computed by an NL machine, using only $O(1)$ additional bits of description. On the other hand, it is not known if there are $n^{O(1)}$ paths that can be found and computed by an NL machine, using only $O(\log n)$ additional bits of description. The hashing technique that is used in this paper *does* provide for a short description of each such path, if there are no more than $n^{O(1)}$ paths in total.

It might be interesting to find if there is some machine-based characterization of $\mathcal{C}$-printable sets, for other small classes $\mathcal{C}$. It is not too hard to show that every sparse set that is accepted by a uniform read-once bounded-width branching program is L-printable. (Sketch: for each of the $O(1)$ nodes v at level i, compute the number of paths from s to v and from v to t. This enables a logspace machine to take a number j and compute the j^{th} accepting path in the branching program, and to output the input variables that cause this path to be followed.) It is not clear if this computation can be performed in Boolean NC^1, and it is even less clear that every NC^1-printable set (or even every AC^0-printable set) can be accepted by read-once bounded-width branching programs.

Is OptL = FNL (at least in the nonuniform setting)? Can new relationship be proved among the classes {UL, FewUL, FewL, LFew, NL} in the uniform setting?

Acknowledgement

I thank Vladimir Glasnak, Sunny Daniels, Michal Koucký, Detlef Ronneburger, Sambuddha Roy, and Samir Datta for helpful conversations.

References

[ABK+02] E. Allender, H. Buhrman, M. Koucký, D. van Melkebeek, and D. Ronneburger. Power from random strings. In *Proc. IEEE FOCS*, pages 669–678, 2002.

[AJ93] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[ÀJ95] C. Àlvarez and B. Jenner. A note on logspace optimization. *Computational Complexity*, 5:155–166, 1995.

[AKRR03] E. Allender, M. Koucký, D. Ronneburger, and S. Roy. Derandomization and distinguishing complexity. In *Proc. IEEE Conf. on Comput. Complexity*, 2003.

[All01] E. Allender. When worlds collide: Derandomization, lower bounds, and Kolmogorov complexity. In *Proc. FST&TCS*, volume 2245 of *Lecture Notes in Computer Science*, pages 1–15, 2001.

[AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM J. Comput.*, 17:1193–1202, 1988.

[AR98] E. Allender and K. Reinhardt. Isolation, matching, and counting. In *Proc. IEEE Conf. on Comput. Complexity*, pages 92–100, 1998. This material was incorporated into [ARZ99].

[ARZ99] E. Allender, K. Reinhardt, and S. Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

[AZ98] E. Allender and S. Zhou. Uniform inclusions in nondeterministic logspace. In R. Freivalds, editor, *Randomized Algorithms*, pages 35–41, 1998. MFCS Satellite Workshop, Brno, Czech Republic. A revised version was incorporated into [ARZ99].

[BDHM91] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of logspace-MOD class. *Math. Systems Theory*, 25:223–237, 1991.

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided randomness. In *Proc. STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 100–109, 1999.

[BFL02] H. Buhrman, L. Fortnow, and S. Laplante. Resource-bounded Kolmogorov complexity revisited. *SIAM J. Comput.*, 31(3):887–905, 2002.

[BG92] R. Beigel and J. Gill. Counting classes: thresholds, parity, mods, and fewness. *Theoretical Computer Science*, 103:3–23, 1992.

[BJLR91] G. Buntrock, B. Jenner, K.-J. Lange, and P. Rossmanith. Unambiguity and fewness for logarithmic space. In *Proc. 8th FCT*, volume 529 of *Lecture Notes in Computer Science*, pages 168–179, 1991.

[Boo74] R. V. Book. Tally languages and complexity classes. *Information and Control*, 26:186–193, 1974.

[CH90] Jin-Yi Cai and Lane A. Hemachandra. On the power of parity polynomial time. *Mathematical Systems Theory*, 23:95–106, 1990.

[FGLM99] L. Fortnow, J. Goldsmith, M. A. Levy, and S. Mahaney. L-printable sets. *SIAM J. Comput.*, 28:137–151, 1999.

[FKS82] M. Fredman, J. Kómlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. In *Proc. IEEE FOCS*, pages 165–169, 1982.

[Gla01] V. Glasnak. Sparse sets and collapse of complexity classes. *Information and Computation*, 170:26–48, 2001.

[Har83] J. Hartmanis. On sparse sets in $\mathrm{NP}-\mathrm{P}$. *Information Processing Letters*, 16:55–60, 1983.

[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.

[JK89] B. Jenner and B. Kirsig. Alternierung und Logarithmischer Platz. Dissertation, Universität Hamburg, 1989.

[KvM02] A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31:1501–1526, 2002.

[Lan97] K.-J. Lange. An unambiguous class possessing a complete set. In *Proc. STACS*, volume 1200 of *Lecture Notes in Computer Science*, pages 339–350, 1997.

[Lev84] L. A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.

[LV97] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications, Second Edition.* Springer, 1997.

[Meh82] K. Mehlhorn. On the program size of perfect and universal hash functions. In *Proc. IEEE FOCS*, pages 170–175, 1982.

[NW94] N. Nisan and A. Wigderson. Hardness vs. randomness. *Journal of Computer and System Sciences*, 49:149–167, 1994.

[RA00] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. *SIAM J. Comput.*, 29:1118–1131, 2000.

[RRW94] R. P. N. Rao, J. Rothe, and O. Watanabe. Upward separation for FewP and related classes. *Information Processing Letters*, 52:175–180, 1994.

[Sak96] M. Saks. Randomization and derandomization in space-bounded computation. In *Proc. IEEE Conf. on Comput. Complexity*, pages 128–149, 1996.