Electronic Colloquium on Computational Complexity, Report No. 57 (1998)

Characterizing Small Depth and Small Space Classes by Operators of Higher Types

Manindra Agrawal*
Dept. of Computer Science
Indian Institute of Technology
Kanpur, India
manindra@iitk.ac.in

Eric Allender†
Department of Computer Science, Rutgers University
Piscataway, NJ 08855, USA
e-mail: allender@cs.rutgers.edu

Samir Datta‡
Department of Computer Science, Rutgers University
Piscataway, NJ 08855, USA
e-mail: sdatta@paul.rutgers.edu

Heribert Vollmer
Lehrstuhl für Theoretische Informatik, Universität Würzburg
Am Exerzierplatz 3, D-97072 Würzburg, Germany
e-mail: vollmer@informatik.uni-wuerzburg.de

Klaus W. Wagner
Lehrstuhl für Theoretische Informatik, Universität Würzburg
Am Exerzierplatz 3, D-97072 Würzburg, Germany
e-mail: wagner@informatik.uni-wuerzburg.de

*Part of this research was done while visiting the University of Ulm under an Alexander von Humboldt Fellowship.
†Supported in part by NSF grant CCR-9734918.
‡Supported in part by NSF grant CCR-9734918.

ISSN 1433-8092

Abstract

Motivated by the question of how to define an analog of interactive proofs in the setting of logarithmic time- and space-bounded computation, we study complexity classes defined in terms of operators quantifying over oracles. We obtain new characterizations of NC^1, L, NL, NP, and NSC (the nondeterministic version of SC). In some cases, we prove that our simulations are optimal (for instance, in bounding the number of queries to the oracle).

1 Introduction

Interactive proofs motivate complexity theorists to study new modes of computation. These modes have been studied to great effect in the setting of polynomial time (e.g. [Sha92, LFKN92, BFL90]) and small space-bounded classes (e.g. [FL93, CL95]). Is it possible to study interactive proofs in the context of even smaller complexity classes? Would such a study be useful or interesting?

It has often proved very useful to study modes of computation on very small complexity classes, although it has not always been clear at first, just how these modes should be modeled. For instance, the definition of alternating Turing machine given in [CKS81] does not allow an interesting notion of sublinear time complexity, whereas augmenting this model with random access to the input provides a useful model for studying circuit complexity classes such as NC^1 and AC^0 [Ruz81, Sip83]. How can one define a useful notion of interactive proof system for deterministic log time?

In attempting to answer this and related questions, we take as our starting point the work of Baier and Wagner [BW98a], where it was shown that (single-prover and multi-prover) interactive proof systems can be modeled by quantifying over oracles applied to P. This framework is defined quite elegantly in terms of operators acting on complexity classes, generalizing the framework initially presented by Schöning [Sch89]. We present the formal definitions below in Section 2.1.

After we present our definitions, we quickly review in Section 3 the main results that were known previously, regarding characterizations of complexity classes in terms of operators applied to P. The most important results regarding interactive proofs are stated there, as well as some additional characterizations due to Baier and Wagner.

Then, in Section 4, we present the notion of "scaling up" and "scaling down" characterizations in this framework, and we present some instances where "scaling down" does hold, as well as some instances where "scaling down" does not hold, and we discuss some of the subtleties involved. We are able to successfully give scaled-down versions of the known characterizations involving nondeterministic time-bounded complexity classes, and the characterizations we give are essentially optimal. In this way, we obtain new characterizations of NP and NL.

Next, in Section 5, we consider how to "scale down" the known characterizations of space-bounded classes. In this way, we obtain new characterizations of NC^1 and NSC. However, a number of open questions remain, regarding whether it is possible to obtain true "scaled down" versions of the characterization of PSPACE in terms of interactive proofs, as given in [Sha92].

2 Definitions

2.1 Oracle Operators

Let us start by formally defining the operators we will use in this paper. They are operators acting on oracles, and since later we are going to apply sequences of oracle quantifiers to some base class, we will need oracle machines with more than one oracle tape.

Definition 2.1. A relativized class $\mathcal{K}$ of type $\sigma_1 \cdots \sigma_k$ is given by a recursive enumeration $M_0, M_1, M_2, \ldots$ of oracle machines with k oracle tapes each where $\sigma_j \in \{0, 1, 2\}$ is the type of M_i's jth oracle ($i \geq 0$, $1 \leq j \leq k$). Here, an oracle of type 2 is a usual oracle with no restriction. An oracle of type 1 is an oracle where after every query the oracle tape is not erased (hence every query is an extension of the previous query). An oracle of type 0 is simply a word. Access to this word is by using the oracle tape as an index tape to address bits at specified positions. Now $L(M_i)$ consists of exactly those tuples $(x, X_1, \ldots, X_k)$ where M halts accepting on the "actual" input x with oracles $X_1, \ldots, X_k$ where X_i is of type σ_i.

For sake of clarity we explicitly remark that resource bounds are measured in the length of the actual input. **In the case of space bounds, all oracle/index tapes are subject to the space restriction.**

Now we define the following operators [BW98a]:

Definition 2.2. Let $\mathcal{K}$ be a relativized class of type $\sigma_1 \cdots \sigma_k \sigma$. Then we define the following classes of type $\sigma_1 \cdots \sigma_k$:

For $\sigma = 0$ and $h\colon \mathbb{N} \to \mathbb{N}$, we have $L \in \exists^h \cdot \mathcal{K}$ iff there is a $B \in \mathcal{K}$ such that

$$(x, X_1, \ldots, X_k) \in L \iff (\exists y, |y| = h(|x|))(x, X_1, \ldots, X_k, y) \in B,$$

and $L \in \forall^h \cdot \mathcal{K}$ iff there is a $B \in \mathcal{K}$ such that

$$(x, X_1, \ldots, X_k) \in L \iff (\forall y, |y| = h(|x|))(x, X_1, \ldots, X_k, y) \in B.$$

We use the superscript log to denote the union of all classes obtained by choosing $h \in O(\log n)$, p to denote the union of all classes obtained by choosing $h \in n^{O(1)}$, and exp to denote the union of all classes obtained by choosing $h \in 2^{n^{O(1)}}$.

For $\sigma = 1$ or $\sigma = 2$, $L \in \exists^\sigma \cdot \mathcal{K}$ iff there is a $B \in \mathcal{K}$ such that

$$(x, X_1, \ldots, X_k) \in L \iff (\exists Y)(x, X_1, \ldots, X_k, Y) \in B,$$

and $L \in \forall^\sigma \cdot \mathcal{K}$ iff there is a $B \in \mathcal{K}$ such that

$$(x, X_1, \ldots, X_k) \in L \iff (\forall Y)(x, X_1, \ldots, X_k, Y) \in B.$$

Book, Vollmer, and Wagner in [BVW96] examined the bounded-error probabilistic operator of type 2. We repeat their definition together with the definitions of the corresponding type 0 operator (see e.g. [Sch89]). We also consider a one-sided error operator in the case of type 0.

Definition 2.3. Let $\mathcal{K}$ be a relativized class of type $\sigma_1 \cdots \sigma_k 2$. Then $\mathrm{BP}^2\mathcal{K}$ is a class of type $\sigma_1 \cdots \sigma_k$, and $L \in \mathrm{BP}^2\mathcal{K}$ iff there exists a set $B \in \mathcal{K}$ such that

$$\mu_A\left[(x, X_1, \ldots, X_k) \in L \leftrightarrow (x, X_1, \ldots, X_k, A) \in B\right] \geq \frac{2}{3}.$$

The measure $\mu\colon 2^{\{0,1\}^\omega} \to [0,1]$ is the product measure based on $\mu_0 : 2^{\{0.1\}} \to [0,1]$, where $\mu_0(\{0\}) = \mu_0(\{1\}) = \frac{1}{2}$. Here we identify as usual languages over $\{0,1\}$ as infinitely long sequences from $\{0,1\}^\omega$.

Definition 2.4. Let $h\colon \mathbb{N} \to \mathbb{N}$. Let $\mathcal{K}$ be a relativized class of type $\sigma_1 \cdots \sigma_k 0$. Then we say that

1. $L \in \mathrm{BP}^h\mathcal{K}$ iff there exists a $B \in \mathcal{K}$ such that

$$\mathrm{prob}_{|y|=h(|x|)}\left[(x, X_1, \ldots, X_k) \in L \leftrightarrow (x, X_1, \ldots, X_k, y) \in B\right] \geq \frac{2}{3}.$$

2. $L \in \overline{\mathrm{R}}^h \mathcal{K}$ iff there is a $B \in \mathcal{K}$ such that

$$\begin{array}{l}(x, X_1, \ldots, X_k) \in L \rightarrow \mathrm{prob}_{|y|=h(|x|)}\big[(x, X_1, \ldots, X_k, y) \in B\big] = 1 \\ (x, X_1, \ldots, X_k) \notin L \rightarrow \mathrm{prob}_{|y|=h(|x|)}\big[(x, X_1, \ldots, X_k, y) \in B\big] \leq \frac{1}{2}.\end{array}$$

In the following we will apply sequences of operators to base classes such that the resulting class is of type ϵ (without any oracle). That is, in the terminology of Definition 2.1 we have $k = 0$ and we deal with a usual class of languages of simple words, not of tuples consisting of words and sets. The type of the base class will always be implicitly given from the context. If we write e.g. $Q_1^{\sigma_1} \cdots Q_k^{\sigma_k} \cdot \mathrm{P}$, then P is a class of languages of $k+1$ tuples $(x, X_1, \ldots, X_k)$ where for $1 \leq j \leq k$, X_j is of type σ_j if $\sigma_j \in \{1, 2\}$ and X_j is of type 0 if $\sigma_j \in \{\exp, \mathrm{p}, \log\}$.

2.2 Restricting the number of queries

Let us now consider restrictions to the number of oracle queries or the number of spot checks on a random access input tape.

Definition 2.5. Let $k \geq 0$, let $\sigma_1, \sigma_2, \ldots, \sigma_k \in \{1, 2, \exp, \mathrm{p}, \log\}$, and let $r_1, r_2, \ldots, r_k \colon \mathbb{N} \rightarrow \mathbb{N}$. Let

$$L \in Q_1^{\sigma_1}[r_1] Q_2^{\sigma_2}[r_2] \cdots Q_k^{\sigma_k}[r_k] \cdot \mathcal{K}$$

iff $L \in Q_1^{\sigma_1} Q_2^{\sigma_2} \cdots Q_k^{\sigma_k} \cdot \mathcal{K}$ via a $\mathcal{K}$ machine M which on input $(x, X_1, \ldots, X_k)$ makes at most $r_i(|x|)$ queries to the i-th oracle or random access input tape (for $1 \leq i \leq k$). We omit $[r_i]$ if it does not constitute a real restriction (for example if it is greater than the runtime of the underlying machine).

3 Known Results

The following relation between type 2 oracle operators and word operators is known: If $\mathcal{K}$ is a class defined by nondeterministic polynomial time machines (technically, $\mathcal{K}$ is *leaf language definable*, i.e, $\mathcal{K} = \mathrm{Leaf}^{\mathrm{P}}(B)$ for some set B [HLS+93, JMT96]), then $Q^2 \cdot \mathcal{K} = Q^{\exp} \cdot \mathcal{K}$ (where Q can be any one of the above operators) [BVW96].

We remark that a connection between the BP^2 operator and ALMOST-classes has been established in [BVW96], see also [VW97a]. There it was shown that for a great number of classes $\mathcal{K}$, the identities

$$\mathrm{BP}^2 \cdot \mathcal{K} = \mathrm{BP}^{\exp} \cdot \mathcal{K} = \text{ALMOST-}\mathcal{K}$$

hold.

However the main reason for studying the above formalism stems from the fact that many results in the area of interactive protocols and probabilistically checkable proofs can be formulated conveniently in terms of the operators defined in Section 2.

Fortnow, Rompel, and Sipser in 1988 characterized the power of multiprover interactive proofs by an existential operator ranging over oracles. Baier and Wagner in 1996 obtained the corresponding result for single-prover interactive proofs.

Theorem 3.1. *1.* $\mathrm{NEXPTIME} = \mathrm{MIP} = \exists^2 \cdot \mathrm{BP}^{\mathrm{p}} \cdot \mathrm{P} = \exists^2 . \overline{\mathrm{R}}^{\mathrm{p}} \cdot \mathrm{P}$ *[FRS94].*

2. $\mathrm{PSPACE} = \mathrm{IP} = \exists^1 \cdot \mathrm{BP}^{\mathrm{p}} \cdot \mathrm{P} = \exists^1 \cdot \overline{\mathrm{R}}^{\mathrm{p}} \cdot \mathrm{P}$ *[BW98a].*

Because the first equality in both statements is *not* relativizable, this yields non-relativizable operator characterizations of NEXPTIME and PSPACE. In contrast to this, the following characterizations (obtained in [BW98a], see also [VW97b]) are relativizable.

Theorem 3.2. *1.* $\mathrm{NEXPTIME} = \exists^2 \cdot \forall^{\mathrm{p}} \cdot \mathrm{P} = \exists^2[3] \cdot \forall^{\mathrm{p}} \cdot \mathrm{P}$.

2. $\mathrm{PSPACE} = \exists^1 \cdot \forall^{\mathrm{p}} \cdot \mathrm{P} = \exists^1[2] \cdot \forall^{\mathrm{p}} \cdot \mathrm{P}$.

In [BW98b] it has been proved that this presentation of PSPACE and NEXPTIME cannot be improved with respect to the number of queries.

The main question of this paper is if characterizations of small space classes or small depth circuit classes similar to those above can be given. Before we turn to this point, we first address the class NP.

4 Scaling down and up: NEXPTIME vs. NP

A known inclusion relating NP with another class can usually easily be scaled up to obtain a similar result for NEXPTIME. This is done by translational methods (exponential padding). However, here we are interested in going the other direction: "scaling down". We have a result for NEXPTIME such as

$$\begin{aligned} \mathrm{NEXPTIME} &= \exists^2 \cdot \forall^{\mathrm{p}} \cdot \mathrm{P} = \exists^2[3] \cdot \forall^{\mathrm{p}} \cdot \mathrm{P} && (1) \\ &= \exists^2 \cdot \mathrm{BP}^{\mathrm{p}} \cdot \mathrm{P} = \exists^2 \cdot \overline{\mathrm{R}}^{\mathrm{p}} \cdot \mathrm{P} && (2) \end{aligned}$$

Can we obtain similar results for NP? What would a "similar result" be in this context? One's intuition is to try to reduce logarithmically all of the resource bounds and operator ranges that are involved, and to obtain an equality with the property, that the original result can obtained from the "scaled down" result via padding ("scaling up").

However it is clear that this will not work in each case. (We give a concrete example below, showing how some attempts at "scaling down" are doomed to failure.) If there is a genuine "scaled-down" version of a result, then it cannot be derived mechanically. Rather, it requires a separate proof.

In this section we want to exemplify what we mean by comparing some previously-known characterizations of NP and NEXPTIME. Let us start with an easy example and show how to scale down equation (1).

Theorem 4.1. $\mathrm{NP} = \exists^2 \cdot \forall^{\log} \cdot \mathrm{P} = \exists^2[3] \cdot \forall^{\log} \cdot \mathrm{DLOGTIME}$.

Proof sketch. For the inclusion $\exists^2 \cdot \forall^{\log} \cdot \mathrm{P} \subseteq \mathrm{NP}$, note that, although a polynomial-time machine M can write queries having a polynomial number of bits (and thus ranging over an exponentially-large address space), for each given oracle and for each given logarithmically-long string z, the polynomial-time machine queries at most a polynomial number of oracle positions. Thus, in order to determine if x is in a language in $\exists^2 \cdot \forall^{\log} \cdot \mathrm{P}$, it suffices to guess a polynomial number of answers in some oracle A, and then execute a poly-time machine $M^A(x, z)$ for all of the polynomially-many choices for z.

For the inclusion $\mathrm{NP} \subseteq \exists^2[3] \cdot \forall^{\log} \cdot \mathrm{DLOGTIME}$, the proof relies on the fact that 3-SAT is NP-complete under DLOGTIME-uniform projections [Imm87]. To accept a 3-SAT formula, the existential quantifier guesses an assignment. The universal quantifier guesses a clause. The DLOGTIME computation checks that the clause is satisfied by the assignment. Since we have 3 literals per clause, we meet the restriction that every DLOGTIME-computation looks at at most 3 bits in the assignment. ❑

Sitting inside of Theorem 4.1 is a perfect scaled-down version of equation (1). That is, start with the equality

$$\mathrm{NEXPTIME} = \exists^2 \cdot \forall^{\mathrm{p}} \cdot \mathrm{P} = \exists^2[3] \cdot \forall^{\mathrm{p}} \cdot \mathrm{P}.$$

When we logarithmically reduce all of the bounding functions, NEXPTIME becomes NP, $\forall^{\mathrm{p}}$ becomes $\forall^{\log}$, and P becomes POLYLOGTIME, to yield the equality

$$\mathrm{NP} = \exists^2 \cdot \forall^{\log} \cdot \mathrm{POLYLOGTIME} = \exists^2[3] \cdot \forall^{\log} \cdot \mathrm{DLOGTIME}.$$

It is natural to wonder if the constant "3" in Theorem 4.1 can be reduced. As the next theorem shows, this is equivalent to the NL = NP question.

Theorem 4.2. $\mathrm{NL} = \exists^2[2] \cdot \forall^{\log} \cdot \mathrm{DLOGTIME}$.

Proof. The proof of $\mathrm{NL} \subseteq \exists^2[2] \cdot \forall^{\log} \cdot \mathrm{DLOGTIME}$ relies on the NL-completeness of 2-SAT under DLOGTIME reductions [Jon75, Imm88, Sze87]. The proof is exactly analogous to that showing $\text{3-SAT} \in \exists^2[3] \cdot \forall^{\log} \cdot \mathrm{DLOGTIME}$.

For the other direction, let the values on the oracle tape be z_i (where $1 \leq i \leq p(n)$ for some polynomial p) and let the universally-quantified value be y ($|y| = \log n$). Then depending on the input x and on y, the machine first looks at one bit on the oracle tape z_{i_1} and depending on whether it is 0 or 1 looks at either z_{i_2} or at z_{i_3} and then accepts or rejects depending on the value of this second bit. All this is done within time $O(\log n)$. Thus the acceptance criterion can be written as $(z_{i_1} \rightarrow l_2) \wedge (\overline{z_{i_1}} \rightarrow l_3)$, where each $l_j \in \{z_{i_j}, \overline{z_{i_j}}\}$ $(j = 2, 3)$, which is a 2-SAT formula with 2 clauses. Taking a conjunction over all possible y's we get a 2-SAT formula of polynomial length that is satisfiable iff the machine accepts. Satisfiability of this formula can be checked in NL, completing the proof. ❑

The obvious next step is to ask how to scale down the $\mathrm{NEXPTIME} = \exists^2 \cdot \mathrm{BPP} = \exists^2 \cdot \mathrm{coRP}$ result. Observe that it was just this question which was part of the motivation for the PCP-characterization of NP [AS98, ALM+92], see also [BFLS91, FGL+91, Bab93]. The PCP result can be formulated in terms of our operators as follows:

Theorem 4.3 [AS98, ALM+92].

$$\mathrm{NP} = \exists^2 \cdot \mathrm{BP}^{\log} \cdot \mathrm{P} = \exists^2[O(1)] \cdot \overline{\mathrm{R}}^{\log} \cdot \mathrm{P}.$$

Formally scaling down the NEXPTIME characterization given in equation (2) would yield something like

$$\mathrm{NP} \stackrel{?}{=} \exists^2 \cdot \overline{\mathrm{R}}^{\log} \cdot \mathrm{DLOGTIME}$$

or

$$\mathrm{NP} \stackrel{?}{=} \exists^2 \cdot \overline{\mathrm{R}}^{\log} \cdot \mathrm{POLYLOGTIME}.$$

However, *neither* of these equalities hold, as the following proposition demonstrates.

Proposition 4.4. $1^* \notin \exists^2 \cdot \overline{\mathrm{R}}^{\log} \cdot \mathrm{POLYLOGTIME}$.

Proof. Assume otherwise. Let M be the POLYLOGTIME machine such that, on input 1^n, there exists an oracle A such that for most y, $M(x, y, A) = 1$, and such that for any string x that contains any zeros, for any oracle A, for most strings y, $M(x, y, A) = 0$.

Let n be large enough, and consider some "good" oracle A that causes $M(x, y, A)$ to accept for most y.

Given a number i such that $1 \leq i \leq n$, let us say that *i is queried by y* if the computation of $M(x, y, A)$ reads bit i of the input.

An easy counting argument shows that there is some i such that fewer than one-third of the strings y query i. (To see this, note that there are at most n^k possible strings y, for some k. Each such y queries at most $\log^j n$ of the i's. Thus, on average, each i is queried by at most $n^{k-1}/\log^j n < n^k/3$ strings y.)

Thus, M does not have the desired behavior on input $1^{i-1}01^{n-i}$. ❑

Remark 4.5. *It is not hard to show that* $\exists^2 \cdot \overline{\mathrm{R}}^{\log} \cdot \mathrm{DLOGTIME}$ *does contain* $\mathrm{NTIME}(\log n)$.

Thus, it seems at first as if no good options remain to us, in our attempt to scale down equation (2). Theorem 4.3 is not adequate, since in scaling up this result, the polynomial time bound becomes an exponential time bound. Instead, we are in need of a stronger PCP-like characterization of NP. As we shall see, this is possible.

Fortunately, one can reduce the time bound in Theorem 4.3 to polylogarithmic time if the input is encoded in an error-correcting code, see e.g. [BFLS91], [Bab93, Theorem 2] or [HPS95, Section 4.8]. The encoding is a polynomial time procedure, but one can hope that such an encoding is no more time consuming if an additional padding has to be encoded.

To make this precise we define *polynomial time strong many-one reducibility* between sets as follows: Let $A \in \{0,1\}^*\#^*$. (Later, # will be our padding symbol.) Then $A \leq_m^{\mathrm{p,s}} B$ iff there exists a function f with the following properties: First, f is polynomially length bounded. Second, given (x, m) (m in binary) we can compute every bit of $f(x\#^m)$ in polynomial time. Finally, f reduces A to B, i.e. $x\#^m \in A \iff f(x\#^m) \in B$.

We will use this notion of strong many-one reducibility to define an equivalence relation on complexity classes. Let us write $\mathcal{K}_1 \simeq \mathcal{K}_2$ if every set in $\mathcal{K}_1$ is reducible to a set in $\mathcal{K}_2$ by strong many-one reductions, and vice-versa. (In most cases of interest to us, we will have $\mathcal{K}_2 \subseteq \mathcal{K}_1$ or vice-versa.)

As was pointed out to us by S. Safra, the proof of the PCP-theorem [ALM+92] even yields the following stronger characterization of NP:

Theorem 4.6 [ALM$^+$92]. NP $\simeq \exists^2[O(1)] \cdot \overline{\mathrm{R}}^{\log} \cdot$ POLYLOGTIME.

From this statement one can easily conclude (by translation) the MIP theorem NEXPTIME $= \exists^2 \cdot \overline{\mathrm{R}}^{\mathrm{p}} \cdot \mathrm{P}$ (equation (2)). The reason for this is the following: Applying standard translational (padding) techniques, given a language L in NEXPTIME, the padded version of L, which is in NP, will reduce to a language in $\exists^2 \cdot \overline{\mathrm{R}}^{\log} \cdot$ POLYLOGTIME under strong many-one reductions. The reduction can actually be performed in time polynomial in the length of the prefix (without padding symbols) which is polylogarithmic in the input length. Translating this up shows NEXPTIME $\subseteq \exists^2 \cdot \overline{\mathrm{R}}^{\mathrm{p}} \cdot \mathrm{P}$. In fact since we can restrict the number of queries to a constant in Theorem 4.6 the same applies for NEXPTIME, giving the following improvement of equation (2):

Corollary 4.7 [Gol97]. NEXPTIME $= \exists^2[O(1)] \cdot \overline{\mathrm{R}}^{\mathrm{p}} \cdot \mathrm{P}$.

5 Scaling down and up: PSPACE vs. ?

As done for NEXPTIME in the previous section we want to examine in this section scaled-down versions of the PSPACE characterizations from Theorem 3.1 and 3.2:

$$\mathrm{PSPACE} = \exists^1[2] \cdot \forall^{\mathrm{p}} \cdot \mathrm{P} = \exists^1 \cdot \forall^{\mathrm{p}} \cdot \mathrm{P} \tag{3}$$

$$\mathrm{PSPACE} = \exists^1 \cdot \mathrm{BP}^{\mathrm{p}} \cdot \mathrm{P} = \exists^1 \cdot \overline{\mathrm{R}}^{\mathrm{p}} \cdot \mathrm{P} \tag{4}$$

As in the previous section we have to reduce all resource bounds and operator ranges logarithmically. But what is the scaled-down counterpart $\mathcal{K}$ of PSPACE? When we scale up $\mathcal{K}$ we should arrive at PSPACE, i.e. for every language A, $A \in$ PSPACE iff there is some $k \in \mathbb{N}$ such that $A_k \in \mathcal{K}$, where $A_k =_{\mathrm{def}} \left\{ x\#^{2^{|x|^k}-|x|} \mid x \in A \right\}$. It is obvious that every class $\mathcal{K}$ such that $\mathrm{NC}^1 \subseteq \mathcal{K} \subseteq$ POLYLOGSPACE fulfills this condition and hence is a scaled-down counterpart of PSPACE. In fact, we can prove several scaled down versions of equation (3).

Let us start by trying to replace (as in the previous section) the class P by POLYLOGTIME, i.e. the class under consideration is $\exists^1[O(1)] \cdot \forall^{\log} \cdot$ POLYLOGTIME. It turns out that this coincides with a nondeterministic analogue of Steve's class SC($=_{\mathrm{def}}$ DTIME-SPACE($n^{O(1)}, \log^{O(1)} n$)). That is, it coincides with the class NSC $=_{\mathrm{def}}$ NTIME-SPACE($n^{O(1)}, \log^{O(1)} n$).

Theorem 5.1.

$$\begin{aligned} \mathrm{NSC} &= \exists^1[3] \cdot \forall^{\log} \cdot \mathrm{POLYLOGTIME} \\ &= \exists^1 \cdot \forall^{\log} \cdot \mathrm{POLYLOGTIME} \\ &= \exists^1 \cdot \forall^{\log} \cdot \mathrm{SC}. \end{aligned}$$

The proof follows immediately from the following two lemmas.

Lemma 5.2. *For every s such that $\log n \le s(n) \le n^{O(1)}$, we have*

$$\exists^1 \cdot \forall^{\log} \cdot \text{DTIME-SPACE}(n^{O(1)}, s) \subseteq \text{NTIME-SPACE}(n^{O(1)}, s).$$

Proof. The proof is similar to the proof of IP $\subseteq$ PSPACE. We simulate a computation of the form $\exists^1 \cdot \forall^{\log} \cdot \text{DTIME-SPACE}(n^{O(1)}, s)$ by making a depth first search in the query tree. Note that, since the length of a query must respect the space bound, a query consists of $O(s)$ bits. Furthermore, since the oracle's access is of type 1, there are at most $O(s)$ queries made on any execution path. Thus a query-answer sequence can be stored in $O(s)$ space. In order to search the query tree, we first guess a query-answer sequence and check that it is the leftmost one in the tree of all possible such sequences. (That is, it is the sequence generated if all of the oracle queries are answered negatively.) Then we check that if this is a sequence that actually occurs on one path of the $\forall^{\log} \cdot \text{DTIME-SPACE}(n^{O(1)}, s)$ computation, then this computation path is accepting. That is, we cycle through all possible strings y of length $\log n$, and verify that *either* $M(x, y)$ does *not* ask the queries in that sequence, *or* (it *does* ask that sequence, *and* it accepts).

Next, we consider the lexicographically next queries-answer-sequence, check again that if it actually occurs it leads to an accepting computation. In this way we search the whole query tree. All in all we guess an oracle, but since access to this oracle is type 1, we only have to store one path in the tree in order to be able to be consistent with previous answers (we do not have to store the whole oracle up to the maximal query length). Thus the simulation is space bounded by $s(n)$ and since s is bounded by a polynomial and we have only polynomially many $\forall^{\log} \cdot \text{DTIME-SPACE}(n^{O(1)}, s)$ computations, the overall time is also polynomial. ❑

Lemma 5.3. *Let $\log n \le s(n) \le n$ be such that $s(n)$ is computable (in binary) in time $\log^2 n$. Then*

$$\text{NTIME-SPACE}(n^{O(1)}, s) \subseteq \exists^1[3] \cdot \forall^{\log} \cdot \text{DTIME}(s \cdot \log).$$

Proof. The proof consists of three main ingredients: First we show that NTIME-SPACE($n^{O(1)}, s$) can without loss of generality be assumed to be oblivious in a certain weak form. Using this we can then describe the behavior of such machines by certain 3-CNF formulae. Third, we show how to evaluate this formula in $\exists^1[3] \cdot \forall^{\log}$DTIME($s \cdot \log$). The main point here is that we have to force type 1 behavior while evaluating the formula.

Let $A \in$ NTIME-SPACE($n^{O(1)}, s$) be recognized by machine M. We can assume that M is oblivious in the following sense: The input head movements only depend on the length of the input and not the actual input itself, and the function mapping (in binary) (n, t) into $e(n, t)$, the position of the input head after t time steps of M on an input of length n, is computable in quadratic time. This can be achieved as follows:

Starting with an arbitrary NTIME-SPACE($n^{O(1)}, s$) machine N for A, M starts a simulation as follows: Given an input x of length n, first the binary representation of n is computed in an oblivious way (in time $O(n \log n)$). Then $s(n)$ is computed in binary (in time $O((\log n)^2)$) and then $s(n)$ tape cells are marked (in time $O(n \log n)$). In this second phase the input head does not move.

Now M simulates N in an oblivious way: The worktape head makes one complete right scan over the marked region of $s(n)$ cells, then a complete left scan followed again by a complete right scan, and so on. The input head rests during such right-left-scan phases and moves one step when the worktape head returns to the left end of the marked region. In this way the input head performs complete scans across the input. This behavior allows M to keep track of the input head position by a binary counter on a track of the worktape. One step of N can be simulated during a complete right-left-scan of the input head, i.e. in $O(n \cdot s(n))$ steps of M.

Next, we describe the behavior of M by a certain formula. Consider an input x, $|x| = n$. Let the runtime of M be bounded by n^k for $k \in \mathbb{N}$. Let $x(l)$ be the l-th symbol in x. Let $a(t, i)$ denote the contents of M's worktape at position i in step t, where we assume that M's state is attached to that tape symbol that is scanned by the worktape head; i.e. every $a(t, i)$ is either an alphabet symbol or a pair consisting of an alphabet symbol and a state of M. Let H be a predicate such that $H(a, b, c, d, e)$ iff according to M's transition table, if the tape contents in a given time step in three neighboring cells is b, c, and d, and the input head scans a symbol e, then in the next time step the c will be replaced by an a. Similarly, let $H_1(a)$ iff a is the blank symbol, let $H_2(a)$ iff a is an arbitrary symbol to which the accepting state is attached, and finally $H_3(a)$ iff a is a marker symbol for the right and left end of the used part of the tape. Now,

$$\begin{aligned} x \in A \iff \bigvee_a \Big(\bigwedge_{t=1}^{n^k} \bigwedge_{i=1}^{s(n)} & H\big(a(t,i), a(t-1,i-1), \\ & \quad a(t-1,i), a(t-1,i+1), \\ & \quad x(e(n,t))\big) \\ \wedge \bigwedge_{i=0}^{s(n)} & H_1\big(a(0,i)\big) \wedge H_2\big(a(n^k,1)\big) \\ \wedge \bigwedge_{t=0}^{n^k} & \big(H_3\big(a(t,0)\big) \wedge H_3\big(a(t,s(n)+1)\big)\big)\Big) \end{aligned}$$

Next, we encode the values $a(t,i)$ by Boolean vectors $b(t,i,1)\cdots b(t,i,m)$ for suitable m (depending on the size of M's alphabet). The above formulae H, H_1, H_2, and H_3 can then be transformed into 3-CNFs of constant size, say with r clauses. Define $0^0 = 1^1 = 1$ and $0^1 = 1^0 = 0$. Then there are functions h and a such that

$$x \in A \iff \bigvee_{b'} \bigwedge_{t=0}^{n^k} \bigwedge_{i=0}^{s(n)+1} \bigwedge_{j=1}^{r} \bigvee_{l=1}^{3} b'\big(h(t,i,j,l)\big)^{a(t,i,j,l)}$$

Here, b' is a binary string encoding both the Boolean vectors $b(t,i,1)\cdots b(t,i,m)$ plus the (constantly many) new variables that have to be introduced when transforming the formulas into 3-CNF (see [GJ79, p. 48].) New variables introduced in clauses refering to particular choices of t,i will never be used in clauses responsible for other values of t,i. We refer to these variables as the (t,i)-variables. Observe that $b'(h(t,i,j,l))$ is one of the values $b(t,i,\mu), b(t-1,i-1,\mu), b(t-1,i,\mu), b(t-1,i+1,\mu)$ for $1 \le \mu \le m$ and $a(t,i,j,l) \in \{0,1\}$, or a (t,i)-variable. Observe that by the assumption of the theorem and by obliviousness, $x(e(n,t))$ is computable in time $\log^2 n \le s(n)\cdot \log n$; hence, since the transition function is fixed, h and a are computable with the same resources, hence the above formula can be evaluated in $\exists^2[3]\cdot\forall^{\log}\mathrm{DTIME}(s\cdot\log)$

The only problem that remains to be fixed now is that during the evaluation of the formula, the access to b' is not of type 1. To achieve this we encode b' in a special form. We subsume $b(t,i,1)\cdots b(t,i,m)$ and the values of the (t,i)-variables in $b'(t,i)$ and abbreviate $b'(t,0)\cdots b'(t,s(n)+1)$ by $b'(t)$. The length of every $b'(t)$ is L, a value linear in $s(n)$. Observe that for every $t > 0$ and for every i and j, the bits $b'(h(t,i,j,1))$, $b'(h(t,i,j,2))$, and $b'(h(t,i,j,3))$ can be found in $b'(t-1)$ and $b'(t)$. Thus what we have to accomplish is that we are able to ask for $b'(t-1)$ and $b'(t)$ (for $t > 1$) in a type 1 manner. Let 2^v be the smallest power of 2 greater than n^k. The string b' contains $b'(t)$ for $t \leq n^k$; for completeness define $b'(t) =_{\text{def}} 0...000$ for $n^k < t < 2^v$. Construct an oracle B as follows: Given t, $1 \leq t < 2^v$, we want to encode $b'(t)$ into B by putting one string into the oracle for every bit in $b'(t)$ that is 1. First, let the length v binary representation of t be $u_1 \cdots u_{\ell-1}u_\ell 10^q$. Let $b'(t) = b'_1 b'_2 \cdots b'_L$. Then for every $1 \leq i \leq L$, set $0^L u_1 0^L u_2 0^L \cdots 0^L u_\ell 0^i \in B$ iff $b'_i = 1$. Then we can recover $b'(t)$ by asking the oracle for all $0^L u_1 0^L u_2 0^L \cdots 0^L u_\ell 0^i$, $1 \leq i \leq L$. Moreover, it can be seen that if we want to recover any $b'(t-1)$ and $b'(t)$ we only have to query B in a type 1 manner. The length of the oracle queries is now bounded by $O(\log n \cdot s(n))$. ❏

Defining $\mathrm{SC}^k =_{\text{def}} \text{DTIME-SPACE}(n^{O(1)}, \log^k n)$ and $\mathrm{NSC}^k =_{\text{def}} \text{NTIME-SPACE}(n^{O(1)}, \log^k n)$ we thus have:

Corollary 5.4. $\mathrm{NSC}^k \subseteq \exists^1[3] \cdot \forall^{\log} \cdot \mathrm{DTIME}(\log^{k+1} n) \subseteq \exists^1 \cdot \forall^{\log} \cdot \mathrm{SC}^{k+1} \subseteq \mathrm{NSC}^{k+1}$.

Since NSC is a scaled-down counterpart of PSPACE, the result NSC $= \exists^1[3] \cdot \forall^{\log} \cdot$ POLYLOGTIME (Theorem 5.1) is a perfect scaled-down analogue of PSPACE $= \exists^1[2] \cdot \forall^{\mathrm{p}} \cdot \mathrm{P}$ (equation (3)), except that we need one oracle query more in our NSC characterization. It is now natural to ask what happens if in the above we replace POLYLOGTIME by DLOGTIME. This leads us to a second scaled-down analogue of equation (3) as follows:

Theorem 5.5.

$$\begin{aligned} \mathrm{NC}^1 &= \exists^1[2] \cdot \forall^{\log} \cdot \mathrm{DLOGTIME} \\ &= \exists^1 \cdot \forall^{\log} \cdot \mathrm{DLOGTIME} \end{aligned}$$

Proof. The inclusion $\mathrm{NC}^1 \subseteq \exists^1[2] \cdot \forall^{\log} \cdot \mathrm{DLOGTIME}$ is essentially Barrington's Theorem [Bar89]. Barrington shows that for every $A \in \mathrm{NC}^1$ there is a function $f\colon \{0,1\}^* \times \{0,1\}^* \to A_5$ computable in logarithmic time (A_5 is the group of all even permutations on 5 positions) and a polynomial p such that for all x,

$$x \in A \iff f(x,0) \circ f(x,1) \circ \cdots \circ f(x, p(|x|)) = 1,$$

where $\circ$ denotes multiplication in A_5. Now the sequence of intermediate results of the evaluation of this term can be encoded as an oracle. For all $i = 0, 1, \ldots, p(|x|)$ it has to be checked whether $f(x,i)$ transforms the previous intermediate result into the next one. This is a $\forall^{\log} \cdot$ DLOGTIME predicate with two oracle queries. Without further care however, the two necessary queries for every i are not of type 1, but one can overcome this difficulty by choosing an appropriate encoding of the queries and making use of the tree rearranging techniques employed in Lemma 5.3.

For the inclusion $\exists^1 \cdot \forall^{\log} \cdot$ DLOGTIME $\subseteq$ NC^1 we prove a somewhat more general result in the following lemma. ❑

Lemma 5.6. *Let $\mathcal{C}$ be either* L *or* DLOGTIME. *Then each language A in $\exists^1 \cdot \forall^{\log} \cdot \mathcal{C}$ is accepted by a family of* NC^1 *circuits with oracle gates for a language from $\mathcal{C}$.*

Proof. Let A be in $\exists^1 \cdot \forall^{\log} \cdot \mathcal{C}$, as witnessed by some machine M. That is, for a string x of length n, x is in A if and only if $\exists^1 B$ such that, for all strings z of length $\log n$, $M^B(x, z)$ accepts.

Recall that the oracle queries that are asked must be short enough to fit on M's tape (which is subject to the logarithmic space bound). Since M is a deterministic machine, for any given computation of $M^B(x, z)$, there is a string i of length $\log n$ such that the set of queries that are asked by machine M during the computation are a subset of $\{j : j \text{ is a prefix of } i\}$. Thus the condition "$M^B(x, z)$ accepts" can be rewritten as follows: for all strings i of length $\log n$, either some query asked by $M^B(x, z)$ is *not* a prefix of i, or all queries asked by $M^B(x, z)$ *are* prefices of i, and $M^B(x, z)$ accepts.

For an oracle B and a string i, let $\overrightarrow{B(i)}$ denote the sequence of length $|i| + 1$ consisting of the answers to all queries of the form "is j in B?" where j is a prefix of i (including the empty prefix). Let $[M, x, z, i, \overrightarrow{B(i)}]$ denote the Boolean value of the condition "either some query asked by $M^B(x, z)$ is *not* a prefix of i, or all queries asked by $M^B(x, z)$ *are* prefices of i, and $M^B(x, z)$ accepts". (Note that this condition depends only on the bits that are present in $\overrightarrow{B(i)}$.)

Restating, note that x is in A if and only if

$$\exists^1 B \forall^{\log} z \forall^{\log} i [M, x, z, i, \overrightarrow{B(i)}]$$

which is equivalent to

$$\exists^1 B \forall^{\log} i \forall^{\log} z [M, x, z, i, \overrightarrow{B(i)}].$$

Let us pick the value of $B(\epsilon)$ nondeterministically. Thus

$$\exists^1 B \forall^{\log} i \forall^{\log} z [M, x, z, i, \overrightarrow{B(i)}]$$

is equivalent to "there exists a value $b \in \{0,1\}$ for $B(\epsilon)$ such that $\exists^1 B' \forall^{\log} i \forall^{\log} z[M, x, z, i, b\overrightarrow{B'(i)}]$" (where $[M, x, z, i, b\overrightarrow{B'(i)}]$ means the same thing as $[M, x, z, i, \overrightarrow{B'(i)}]$, except the value b is used in place of the first bit of $\overrightarrow{B'(i)}$ when answering the oracle queries asked by M.

Next note that, if the first bit of i is 0, then the value of $[M, x, z, i, b\overrightarrow{B'(i)}]$ is completely independent of the value of $B(1)$, and similarly if the first bit of i is 1, then the value of $[M, x, z, i, b\overrightarrow{B'(i)}]$ is completely independent of the value of $B(0)$. Thus we can pick the answers to these queries to B independently. That is, the expression

$$\exists^1 B \forall^{\log} i \forall^{\log} z[M, x, z, i, \overrightarrow{B(i)}]$$

is equivalent to

$$\exists b \in \{0,1\} \; \forall c \in \{0,1\} \; \exists b' \in \{0,1\} \exists^1 B' \forall i \in c\{0,1\}^{\log n - 1} \forall^{\log} z[M, x, z, i, bb'\overrightarrow{B(i)}]$$

This process can be extended for $\log n$ steps, where we first existentially guess a value for $B(j)$ (for some prefix j of our current i) and then universally check that the guess is good for both extensions $j0$ and $j1$ of j. This gives a (DLOGTIME-uniform) formula for this expression, where the atomic predicates of the formula are of the form $[M, x, z, \alpha]$ for some logarithmic-length string α. The condition $[M, x, z, \alpha]$ can be answered by an oracle gate for a language in $\mathcal{C}$. This completes the proof. ❑

Theorem 5.1 showed that applied to SC the quantifier sequences $\exists^1 \cdot \forall^{\log}$ and $\exists^1[O(1)] \cdot \forall^{\log}$ add exactly the power of nondeterminism. For the case of logarithmic space computations however the situation is different:

Theorem 5.7.

$$\begin{aligned} \mathrm{L} &= \exists^1[O(1)] \cdot \forall^{\log} \cdot \mathrm{L} \\ &= \exists^1 \cdot \forall^{\log} \cdot \mathrm{L} \end{aligned}$$

Proof. The inclusions from left to right are obvious. The other direction follows from Lemma 5.6 above. ❑

Comparing Theorems 5.1 and 5.7, the reader may be tempted to conclude that if L = SC, then L = NSC, reasoning as follows: $\mathrm{L} = \exists^1 \cdot \forall^{\log} \cdot \mathrm{L} = \exists^1 \cdot \forall^{\log} \cdot \mathrm{SC} = \mathrm{NSC}$. In fact, no such implication is known to hold. The flaw in the one-line "proof" lies in the fact that the hypothesized equality L = SC (concerning two unrelativized classes) does *not* imply that the "relativized classes of type 10" L and SC are equal. (See Definition 2.1 regarding relativized classes of type $\sigma_1\sigma_2$.) In fact, a simple diagonalization argument shows that these relativized classes are *not* equal.

Finally we want to discuss the question about a scaled-down version of equation (4). From the discussion at the beginning of this section, it is clear that to show $\mathrm{NC}^1 \subseteq \exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{DLOGTIME} \subseteq \exists^1 \cdot \mathrm{BP}^{\log}\mathrm{POLYLOGTIME} \subseteq$ POLYLOGSPACE is sufficient to obtain (4) by translation. However the power of the operator sequence $\exists^1 \cdot \mathrm{BP}^{\log}$ is not clear to us. Concerning upper bounds, $\exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{SC} \subseteq \mathrm{NSC}$ can be shown as in the proof of Lemma 5.2. In fact one can even observe that the simulation given there constitutes a symmetric algorithm, showing that $\exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{L} \subseteq \mathrm{SymL}$. Concerning lower bounds, inclusions as $\mathrm{NC}^1 \subseteq \exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{POLYLOGTIME}$ are not very likely for the same reasons as explained in Section 4 (see the discussion preceding Proposition 4.4). However one might hope for $\mathrm{NC}^1 \leq_m^{\mathrm{p,s}} \exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{POLYLOGTIME}$, but this is open. Let us therefore conclude with the question if one of the classes $\exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{SC}$, $\exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{L}$, $\exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{POLYLOGTIME}$, or $\exists^1 \cdot \mathrm{BP}^{\log} \cdot \mathrm{DLOGTIME}$ coincides with a well-known class.

Acknowledgment. We thank S. Safra for pointing out that the proof of the PCP-theorem from [ALM+92] yields Theorem 4.6.

References

[ALM+92] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the intractability of approximation problems. In *Proceedings 33rd Symposium on the Foundations of Computer Science*, pages 14–23. IEEE Computer Society Press, 1992.

[AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.

[Bab93] L. Babai. Transparent (holographic) proofs. In *Proceedings 10th Symposium on Theoretical Aspects of Computer Science*, volume 665 of *Lecture Notes in Computer Science*, pages 525–534. Springer Verlag, 1993.

[Bar89] D. A. Mix Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[BFL90] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. In *Proceedings 31st Symposium on Foundations of Computer Science*, pages 16–25. IEEE Computer Society Press, 1990.

[BFLS91] L. Babai, L Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings 23rd Symposium on Theory of Computing*, pages 21–32. ACM Press, 1991.

[BVW96] R. V. Book, H. Vollmer, and K. W. Wagner. On type-2 probabilistic quantifiers. In *Proceedings 23rd International Colloquium on Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 369–380. Springer Verlag, 1996.

[BW98a] H. Baier and K. W. Wagner. The analytic polynomial-time hierarchy. *Mathematical Logic Quaterly*, 1998. To appear.

[BW98b] H. Baier and K. W. Wagner. Bounding queries in the analytical polynomial-time hierarchy. *Theoretical Computer Science*, 1998. To appear.

[CKS81] A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28:114–133, 1981.

[CL95] Anne Condon and Richard Ladner. Interactive proof systems with polynomially bounded strategies. *Journal of Computer and System Sciences*, 50(3):506–518, June 1995.

[FGL$^+$91] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings 32nd Symposium on Foundations of Computer Science*, pages 2–12. IEEE Computer Society Press, 1991.

[FL93] Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time–space complexity. *Theoretical Computer Science*, 113(1):55–73, 24 May 1993.

[FRS94] Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 21 November 1994.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[Gol97] O. Goldreich. A taxonomy of proof systems. In L. A. Hemaspaandra and A. L. Selman, editors, *Complexity Theory Retrospective II*, pages 109–134. Springer Verlag, 1997.

[HLS$^+$93] U. Hertrampf, C. Lautemann, T. Schwentick, H. Vollmer, and K. W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207, 1993.

[HPS95] S. Hougardy, H.-J. Prömel, and A. Steger. Probabilistically checkable proofs and their consequences for approximation algorithms. In W. Deubner, H.-J. Prömel, and B. Voigt, editors, *Trends in Discrete Mathematics*, volume 9 of *Topics in Discrete Mathematics*, pages 175–223. North Holland, 1995.

[Imm87] N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16:760–778, 1987.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17:935–938, 1988.

[JMT96] B. Jenner, P. McKenzie, and D. Thérien. Logspace and logtime leaf languages. *Information & Computation*, 129:21–33, 1996.

[Jon75] N. D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 15:68–85, 1975.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, October 1992.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and Systems Sciences*, 21:365–383, 1981.

[Sch89] U. Schöning. Probabilistic complexity classes and lowness. *Journal of Computer and System Sciences*, 39:84–100, 1989.

[Sha92] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39:869–877, 1992.

[Sip83] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th Symposium on Theory of Computing*, pages 61–69. ACM Press, 1983.

[Sze87] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Bulletin of the European Association for Theoretical Computer Science*, 33:96–100, 1987.

[VW97a] H. Vollmer and K. W. Wagner. Measure one results in computational complexity theory. In D.-Z. Du and K.-I. Ko, editors, *Advances in Algorithms, Languages, and Complexity.* Kluwer Academic Publishers, 1997.

[VW97b] H. Vollmer and K. W. Wagner. On operators of higher types. In *Proceedings 12th Conference on Computational Complexity*, pages 174–184. IEEE Computer Society Press, 1997.