Relationships Among PL, #L, and the Determinant*

Eric Allender†
Department of Computer Science
Hill Center, Busch Campus, P.O. Box 1179
Piscataway, NJ 08855-1179, USA
allender@cs.rutgers.edu

Mitsunori Ogihara‡
Department of Computer Science
University of Rochester
Rochester, NY 14627
ogihara@cs.rochester.edu

Abstract

Recent results by Toda, Vinay, Damm, and Valiant have shown that the complexity of the determinant is characterized by the complexity of counting the number of accepting computations of a nondeterministic logspace-bounded machine. (This class of functions is known as #L.) By using that characterization and by establishing a few elementary closure properties, we give a very simple proof of a theorem of Jung, showing that probabilistic logspace-bounded (PL) machines lose none of their computational power if they are restricted to run in polynomial time.

We also present new results comparing and contrasting the classes of functions reducible to PL, #L, and the determinant, using various notions of reducibility.

1 Introduction

One of the most important and influential early results of complexity theory is the theorem of [47] showing that the complexity of computing the permanent of an integer matrix is characterized by the complexity class #P of functions that count the number of accepting computation paths of a nondeterministic polynomial-time machine. It is perhaps surprising that well over a decade passed before it was discovered that an equally-close connection exists between the complexity of computing the determinant of a matrix and the class #L (defined in [5]) of functions that count the number of accepting computation paths of a nondeterministic logspace-bounded machine. In order to state this connection more precisely, let us define GapL to be $\{f : f(x) = g(x) - h(x)$ for some g and h in #L$\}$. (This definition is by analogy to the class GapP, consisting of the difference of #P functions, which is defined and studied in [17, 36, 22].)

*A preliminary version of this paper appeared in Proc. 9th IEEE Structure in Complexity Theory Conference, 1994

†Supported in part by NSF grants CCR-9204874 and CCR-9509603. Part of this work was done while on sabbatical at Princeton University.

‡Supported in part by the JSPS under grant NSF-INT-9116781/JSPS-ENG-207, and by the NSF under grant CCR-9002292. Work done in part while at University of Electro-Communications, Tokyo.

Although stated in different ways, the results of [49, Theorem 6.5], [44, Theorem 2.1], [16], and [48, Theorem 2] show the following fact:

Theorem 1 *[44, 16, 49, 48] A function f is in* GapL *iff f is logspace many-one reducible to the determinant.*

(A function f is logspace many-one reducible to the determinant if there is a function g computable in logspace such that $f(x)$ (viewed as a number written in binary) is equal to the determinant of matrix $g(x)$.)[1] The proof given in [44] is particularly clear and direct: Toda shows that the determinant (of integer matrices) is reducible to iterated matrix multiplication over the integers (using [7]), which in turn is reducible to iterated matrix multiplication over $\{0, 1, -1\}$, which in turn is reducible to a canonical GapL-complete problem, which in turn is reducible to the determinant. We remark that the logspace many-one reductions presented in [44], can in fact be computed by uniform AC^0 circuits; we will use this fact later.

In this paper we use Theorem 1 to give a very simple proof of a theorem of Jung [27], concerning the complexity class PL. PL is defined to be the class of languages A for which there exists a probabilistic Turing machine (in the sense of [19]; that is, a Turing machine with access to a source of unbiased random bits), such that on input x the machine never uses more than $\log |x|$ space, and $x \in A$ if and only if the probability that the machine reaches an accepting configuration is greater than one half. Although the definition of PL is straightforward, it turns out that the complexity of PL is rather difficult to analyze, in part because probabilistic logspace machines can perform useful work after exponentially many computation steps. An easy way to see this is to note that the s-t connectivity problem (a standard NL-complete set) can be accepted with *zero error* by a PL machine that follows a randomly-chosen path from vertex s, accepting if vertex t is reached, and otherwise trying another randomly-chosen path, and so on; if a path exists, it will eventually be found (with probability one).

Gill showed in [19] that PL is contained in PSPACE. This was improved to $\mathrm{DSPACE}(\log^6 n)$ in [41], but it was not until the appearance of [8] that it was even known that PL is contained in P; [8] shows that PL is contained in NC^2, and this can be improved slightly to TC^1 (the

[1] This definition of "functional many-one reducibility" is from [5]. It should be noted that the notion of one function being "many-one reducible" to another is sometimes defined in a much less restrictive way. For example, it is shown in [53, 11] that the permanent of zero-one matrices is "many-one complete" for #P, using a less restrictive version of "many-one reductions." On the other hand, it follows from [47] that the permanent of integer matrices cannot be complete for #P or GapP using our definition of "many-one reduction" unless $\mathrm{P} = \oplus\mathrm{P}$ (since the permanent and the determinant are equal mod 2). That is, the relationship between the determinant and #L is even *closer* than the relationship that is known to exist between the permanent and #P.

class computed by logarithmic-depth threshold circuits).[2] Similarly, PL was not known to be closed under complementation until a complicated proof was given in [42]; a much simpler proof subsequently appeared in [39]. The source of all this difficulty was the fact, already alluded to, that probabilistic logspace machines cannot obviously be restricted to run in polynomial time without loss of computational power. However, Jung showed in [27] that, at least in the *unbounded error* model (which defines the class PL), the polynomial-time restriction causes no loss of power. The proof presented in [27] is complicated; we present an easy proof in Section 3.

Note that Theorem 1, combined with our easy proof of Jung's theorem, give an alternative proof of the result of [8] concerning the complexity of PL, as well as a making closure of PL under complement completely obvious.

Note that it would be remarkable if a theorem analogous to Jung's theorem could also be proved in the *bounded error* case. We have already seen that NL can be accepted with bounded (in fact zero) error; thus this would in some sense provide a non-uniform version of L=NL (i.e., L/poly = NL/poly). Furthermore, Nisan [33] has shown that bounded-error, polynomial-time logspace can be simulated in polynomial time and space $\log^2 n$; hence extending Jung's theorem to bounded error would provide a small-space polynomial-time algorithm for transitive closure. All of these consequences would be surprising. We will not discuss bounded-error probabilistic logspace in the remainder of the paper; for more information on the bounded-error classes, see [9].

Let DET denote the determinant function. (Do not confuse this with the definition of [15], where DET is used to denote the class of functions NC^1-reducible to the determinant.) Theorem 1 allows one to show that $L^{DET} = L^{\#L}$, as well as $AC^0(DET) = AC^0(\#L)$ and $NC^1(DET) = NC^1(\#L)$, etc. However, no logspace-analog of the theorem $P^{PP} = P^{\#P}$ is known, and we suspect that no such analog exists. For example, although we show that any of the high-order $O(\log n)$ bits of a #L function can be computed in L^{PL}, we know of no reason to suspect that the middle or low-order bit of a #L function can be computed in this way.

For most natural problems A, the class of things reducible to A remains the same regardless of the notion of reducibility that is used. For instance, NL is the class of languages reducible to s-t-connectivity under logspace many-one or Turing reductions, or under 1-L reductions or quantifier-free projections or NC^1 or AC^0 reductions or under any of the many other low-level reductions that have been studied in the literature. However, DET does not seem to have this property. In Section 6 we present characterizations of the classes reducible to PL and #L under various notions

[2]The complete history of the complexity of PL is even more complicated. The interested reader may wish to consult [8, p. 115], [25], [43, p. 111], and [20]. The related notion of probabilistic finite automata has also been studied in depth; for recent results and references, see [31, 32].

of reducibility. For example, $\mathrm{AC}^0(\mathrm{PL})$ is equal to the hierarchy $\mathrm{PL}^{\mathrm{PL}^{\cdot^{\cdot^{\cdot^{\mathrm{PL}}}}}}$.

2 Basic Facts about GapL

The class #L was defined and studied in [5]; #L is the class of functions f such that, for some nondeterministic logspace-bounded machine M, $f(x)$ is the number of accepting computation paths of M on x. As in [5] we restrict our definition to those machines M that halt on all computation paths on all inputs; clearly the running time is polynomial. (Otherwise, the number of accepting computation paths can be infinite.) Let FL denote the class of functions computed in (deterministic) logspace. It is noted in [5] that $\mathrm{FL} \subseteq \#\mathrm{L}$.

The following definitions are adapted from [17]. Given a nondeterministic Turing machine M that halts on all computation paths on all inputs, let $\mathrm{gap}_M(x)$ be [number of accepting paths of M on input x] $-$ [number of rejecting paths of M on input x]. Let GapL be $\{\mathrm{gap}_M : M \text{ is a nondeterministic logspace-bounded Turing machine}\}$.

We have defined #L and GapL functions to be functions mapping Σ^* to the integers; however, it is convenient to assume certain conventions concerning how these values are encoded. (For instance, if leading zeros were *not* written, then it would be pointless to talk about the "high-order bit" of a #L function.) Thus we assume that there is some constant k such that a #L function is always represented in binary, using exactly n^k bits. One could devise other conventions, but they would not affect the theorems in any interesting way. GapL functions will be encoded similarly, where the high-order bit records the sign.

Let PL(poly) denote the class of sets accepted by PL machines with the restriction that the machines run in polynomial time. In the next section, we will show that this class is equal to PL.

Given function classes $\mathcal{C}$ and $\mathcal{D}$, let $\mathcal{C} - \mathcal{D}$ be the class of functions $\{f - g : f \in \mathcal{C} \text{ and } g \in \mathcal{D}\}$.

The following elementary results relating #L, GapL, and PL(poly) are easy to establish via trivial modifications to the proofs of the analogous results in [17]

Proposition 2 $\mathrm{GapL} = \#\mathrm{L} - \#\mathrm{L} = \#\mathrm{L} - \mathrm{FL} = \mathrm{FL} - \#\mathrm{L}$.

Proposition 3 *The following are equivalent:*

1. $A \in \mathrm{PL(poly)}$.
2. $\exists f \in \#\mathrm{L}$ *such that* $x \in A$ *iff the high order bit of* $f(x)$ *is 1.*
3. $\exists f \in \mathrm{GapL}$ *such that* $x \in A$ *iff* $f(x) > 0$.

Corollary 4 *Pos.DET* $\in$ PL(poly), *where Pos.DET is the set of all integer matrices with determinant* > 0.

Proof. Immediate from Theorem 1 and Proposition 3. ∎

It is clear that PL(poly) is closed under complement; the usual proof that PP is closed under complement carries over to this case. Thus Corollary 4 and Theorem 1 show immediately that the following sets are complete for PL(poly) under $\leq_m^{\log}$ reductions [26]:

- The set of integer matrices with determinant > 0.
- The set of integer matrices with determinant ≥ 0.
- $\{(A, m) : \mathrm{DET}(A) > m\}$
- $\{(A, m) : \mathrm{DET}(A) \geq m\}$
- $\{(A, B) : \mathrm{DET}(A) > \mathrm{DET}(B)\}$
- $\{(A, B) : \mathrm{DET}(A) \geq \mathrm{DET}(B)\}$

3 A simple proof of Jung's Theorem

In this section we define nondeterministic logspace many-one reducibility $\leq_m^{\mathrm{FNL}}$. We show that PL(poly) is closed under $\leq_m^{\mathrm{FNL}}$, and we show that every set in PL is $\leq_m^{\mathrm{FNL}}$-reducible to a set in PL(poly), and hence that PL = PL(poly).

The class FNL of functions computable via NC^1 circuits with oracles for NL was defined and studied in [4], where it was noted that FNL admits many alternative characterizations, including the following:

- f is in FNL iff f is computed by a logspace-bounded oracle Turing machine with an oracle for NL.
- f is in FNL iff $|f(x)| = |x|^{O(1)}$ and the language $\{(x, i, b) : b \in \Sigma \cup \{\$\}$ and the i-th symbol of $f(x) = b\} \in$ NL, where the presence of $(x, i, \$)$ indicates that $|f(x)| < i$.

Definition 1 *Let* A *and* B *be languages. We say that* $A \leq_m^{\mathrm{FNL}} B$ *if there is a function* $f \in FNL$ *such that for all* $x, x \in A \iff f(x) \in B$.

Theorem 5 *If* $A \leq_m^{\mathrm{FNL}} B \in$ PL(poly), *then* $A \in$ PL(poly).

Proof. Let M be a PL(poly) machine accepting B, and let f be the FNL function reducing A to B. Let N be the NL machine (with polynomial run time) deciding membership in the set $\{(x, i, b) :$ symbol i of $f(x)$ is $b\}$.

Let M' be a machine that, on input x, begins simulating the behavior of M on input $f(x)$. Each time the i-th symbol of $f(x)$ is needed, M' will probabilistically choose a symbol $b \in \Sigma \cup \{\$\}$ and simulate N on input (x, i, b), using coin flips to simulate the nondeterminism of N. If the simulation of N accepts, then M' will continue, using symbol b. If the simulation of N rejects, then M' will flip one more coin and halt, accepting iff the outcome of the coin flip is "1".

It is easy to verify that M' accepts x with probability greater than one half iff $x \in A$. ∎

Theorem 6 PL = PL(poly).

Proof. Let M be a PL machine accepting language A, and let input x be given. As in the proof of Theorem 6.4 of [19], we will construct a Markov chain modeling the behavior of M on x, where state 1 of the chain is the initial configuration, state r is the (unique) accepting configuration, state $r-1$ is a rejecting, halting configuration, and there is a state of the Markov chain for each configuration of M, so that each probability $p_{i,j} \in \{0, \frac{1}{2}, 1\}$ records the probability of going to configuration j from i.

Using an oracle from NL, it is easy to modify this Markov chain by removing all states with no path to the accepting configuration. Any transition in the original chain to one of these removed states is replaced by a transition to a unique rejecting state. The unique accepting and rejecting states are made "absorbing" states, by having the chain loop once it reaches either of those states. Call this new chain B. We have observed that transition matrix B can be constructed from x via an FNL computation. Let m be the number of states in B, other than the two absorbing states.

Let x_i be the probability of ending in the accepting state, starting from state i of B. Observe that

$$x_i = B_{i,m+2} + \sum_k B_{i,k} x_k.$$

As in [19], note that this can be rewritten as

$$(D - I)X = -C$$

where X is the column vector $(x_1, \ldots, x_m)$, D is the upper left $m \times m$ submatrix of B, and C is the column vector consisting of the top m elements of column $m+2$ of B. Furthermore, since B is a Markov chain with two absorbing states, $D - I$ is invertible (see, e.g., [23, Lemma III.4.1]). Let

$E_1 = 2(D - I)$, and let E_2 be equal to E_1, except with column 1 replaced by $2C$. Then E_1 and E_2 are integer matrices constructible from x via an FNL reduction, and $(D - I)X = -C$ if and only if $E_1X = -2C$. By Cramer's rule, x_1 is equal to $\mathrm{DET}(E_2)/\mathrm{DET}(E_1)$.

That is, x is in A iff $x_1 > \frac{1}{2}$ iff 2DET(E_2) - DET(E_1) > 0.

Since DET is in GapL, the function f taking matrices M_1, M_2 as input, such that $f(M_1, M_2) =$ 2DET(M_2)$-$ DET(M_1) is clearly also in GapL. Thus the set $\{(M_1, M_2) : f(M_1, M_2) > 0\}$ is in PL(poly). We have shown that every set in PL is $\leq_m^{\mathrm{FNL}}$-reducible to this set. The result now follows from Theorem 5. ∎

The proof given above makes it clear that the result "PL = PL(poly)" *relativizes* using the appropriate notion of "relativized PL." (This was not so clear from the proof in [27].) We will need this fact in later sections. First, however, we must define what is meant by "relativized PL."

The question of what is an appropriate (or even meaningful) way to provide space-bounded machines with an oracle has been the subject of some debate. For a discussion of some of the issues involved and a list of references, see [1]. It turns out that a simple and useful notion of relativization is the so-called Ruzzo-Simon-Tompa relativization [39]; briefly, using this notion of relativization, a set is in PL^A if there is a probabilistic logspace machine M with an oracle tape and query states (as is usual in the definition of oracle Turing machines) along with the restriction that the machine act *deterministically* when it is writing on its oracle tape. Note that this forces the property that each query that can be asked during a computation on input x is described completely by x and by M's configuration when the query starts to be written. In particular, only a polynomial number of queries can possibly be asked over all of M's computations on x.

Corollary 7 *For any set S,* $\mathrm{PL}^S = \mathrm{PL(poly)}^S$.

Proof. In the proof of Theorem 6, the FNL reduction to a set in PL(poly) is replaced by a FNL^S reduction to the same set; namely, the transitions in the Markov chain denote the transition probabilities in the *relativized* computation. These can clearly be computed using an oracle for S.

The proof of Theorem 5 also clearly carries over to the relativized setting. That is, if A is reducible to $B \in \mathrm{PL(poly)}^S$ via an FNL^S reduction, then $A \in \mathrm{PL(poly)}^S$.

∎

It is worth observing that a proof essentially identical to the proof of Theorem 5 shows the following:

Theorem 8 $\mathrm{PL}^{\mathrm{NL}} = \mathrm{PL}$.

4 Closure Properties of GapL and PL

One way of interpreting the results of the previous section is that PL is the class of languages that are reducible to the high-order bit of a #L function. In this section, we will improve this, to show that PL is the class of languages that are reducible to any of the high-order $O(\log n)$ bits of a #L function. Along the way, we will establish some closure properties of PL and of GapL.

The following theorem and its proof are logspace-analogues of results concerning GapP that were proved in [17].

Theorem 9 *Let f be any function in* GapL.

1. *Let g be a function in* FL. *Then $f(g(\cdot))$ is in* GapL.
2. $\sum_{i=0}^{|x|^c} f(x,i)$ *is in* GapL.
3. $\prod_{i=0}^{|x|^c} f(x,i)$ *is in* GapL.
4. *Let g be a function in* FL *such that $g(x) = O(1)$. Then $\binom{f(x)}{g(x)}$ is in* GapL.

Proof. The proofs of the first three closure properties may be taken essentially word-for-word from [17].

For the fourth closure property, observe, as in [17] that if f is the difference of two #L functions h and h', then

$$\binom{f(x)}{g(x)} = \sum_{i=0}^{g(x)} (-1)^i \binom{h(x)+i}{i} \binom{h'(x)+1}{g(x)-i}.$$

The result now follows from the first three closure properties and from [14, Lemma 2], where it is shown that if a is in #L and b is in FL with $b(x) = O(1)$, then $\binom{a(x)}{b(x)}$ is in GapL. ∎

A *logspace conjunctive truth-table reduction* of one set A to another set B (denoted $A \leq_{\mathrm{ctt}}^{\log} B$) is a logspace-computable function $f(x) = y_1, \ldots, y_r$ for some r such that x is in A if and only if *all* of the y_i are in B. Logspace disjunctive truth-table reductions ($\leq_{\mathrm{dtt}}^{\log}$) are defined similarly, with "all" replaced by "at least one." For more formal definitions, see [30].

Corollary 10 PL *is closed under $\leq_{\mathrm{ctt}}^{\log}$ and $\leq_{\mathrm{dtt}}^{\log}$ reductions. (In particular,* PL *is closed under intersection and union.)*

Proof. We present the proof for $\leq_{\mathrm{ctt}}^{\log}$ reductions. The proof of the other claim is analogous. Let $C \leq_{\mathrm{ctt}}^{\log} B$ via a reduction g, so that $x \in C$ iff $\forall i \leq n^c \;\; g(x,i) \in B$. Let h be the GapL function such that $h(y) > 0$ if $y \in B$, and $h(y) < 0$ otherwise. Let $H(x,i)$ denote $h(g(x,i))$.

The definition of the following sequence of polynomials follows the presentation in Section 2 of [13]. Define

$$\begin{aligned}
P^+(x,i) &= \\
&[(H(x,i)-1)\textstyle\prod_{j=1}^{n^k}(H(x,i)-2^j)^2]^{2^{\lceil c\log n\rceil+1}} \\
P^-(x,i) &= \\
&[(-H(x,i)-1)\textstyle\prod_{j=1}^{n^k}(-H(x,i)-2^j)^2]^{2^{\lceil c\log n\rceil+1}} \\
N(x,i) &= P^-(x,i)-P^+(x,i) \\
D(x,i) &= P^-(x,i)+P^+(x,i) \\
S(x,i) &= \tfrac{N(x,i)}{D(x,i)} \\
A(x) &= (\textstyle\sum_{i=1}^{n^c} S(x,i))-(n^c-1) \\
A'(x) &= \\
&(\prod_{i=1}^{n^c}(D(x,i)))(\sum_{i=1}^{n^c}(N(x,i)\prod_{l\neq i}D(x,l))) \\
&-(\prod_{i=1}^{n^c}(D(x,i)))^2(n^c-1)
\end{aligned}$$

$S(x,i)$ is equivalent to the function that is called $S_{n^k}^{(n^c)}(H(x,i))$ in [13]; it is proved there that $A(x)$ is positive if all of the $H(x,i)$ are positive, and $A(x)$ is negative otherwise. Observe that $A'(x)$ is positive if and only if $A(x)$ is, and note also that A' may be seen to be in GapL because of the closure properties established above. This shows that $C \in$ PL. ∎

Corollary 10 can be strengthened by defining $\leq_{ctt}^{\mathrm{FNL}}$and $\leq_{dtt}^{\mathrm{FNL}}$reductions in the obvious way. It suffices to notice that, by Theorem 5, the function $H(x,i) = h(g(x,i))$ is in GapL, if $g \in$ FNL. This shows:

Corollary 11 PL *is closed under* $\leq_{ctt}^{\mathrm{FNL}}$ *and* $\leq_{dtt}^{\mathrm{FNL}}$ *reductions.*

Corollary 12 *Language A is in* PL *if and only if there is a logspace-computable function $b(x) = O(\log|x|)$ and a* GapL *function f such that [$x \in A$ if and only if the high-order $b(x)$-th bit of $f(x)$ is 1].*

Proof. The forward direction is obvious from the foregoing, with $b(x) = 1$.

For the converse, note that the language $B = \{(x,y) : y \leq f(x)\}$ is in PL (since $(x,y) \in B$ iff the GapL function $f(x) - y \geq 0$). Thus the language $C = \{(z,x,y) : z \leq f(x) \leq y\}$ is also in PL, by Corollary 10. Let $u_1, u_2, \ldots, u_r$ be the $2^{b(x)-1}$ strings of length $(b(x)-1)$, and note that x is in A iff $\bigvee_i u_i 10^{n^k-b(x)} \leq f(x) \leq u_i 11^{n^k-b(x)}$. By corollary 10, A is in PL. (This proof is essentially identical to the proof of the analogous result for time-bounded classes, as presented in [37].) ∎

5 Exact Counting in Logspace

In this section, we introduce the class $\mathrm{C_{=}L}$ as the logspace-analog of the class $\mathrm{C_{=}P}$. We do this in order to characterize the complexity of the class of singular matrices, which is arguably a much more natural problem than the complete sets for PL that have been presented above. The set of integer matrices with determinant zero is a natural complete set for $\mathrm{C_{=}L}$.

First, we need to present some definitions.

Definition 2 *$C_{=}L$ is the class of languages A for which there is a function $g \in GapL$ such that $x \in A$ iff $g(x) = 0$.*

As is the case with $\mathrm{C_{=}P}$ [46], there are several equivalent ways of defining $\mathrm{C_{=}L}$.

Proposition 13 *The following are equivalent:*

1. *$A \in C_{=}L$.*
2. *There is a function $f \in FL$ and a function $g \in GapL$ such that $x \in A$ iff $f(x) = g(x)$.*
3. *There is a function $f \in FL$ and a function $g \in \#L$ such that $x \in A$ iff $f(x) = g(x)$.*
4. *There is a probabilistic polynomial-time logspace-bounded machine such that $x \in A$ iff the probability of accepting is exactly $\frac{1}{2}$.*

Proof. ((1) $\Longrightarrow$ (4)) Let $A \in \mathrm{C_{=}L}$, and let g be the GapL function such that $x \in A$ iff $g(x) = 0$. Let f_1 and f_2 be $\#\mathrm{L}$ functions such that $g(x) = f_1(x) - f_2(x)$. Let p be a polynomial, and let M_1 and M_2 be nondeterministic machines realizing f_1 and f_2, respectively, where each of M_1 and M_2 flip exactly $p(|x|)$ coins along every computation path. (I.e., M_1 can be constructed from an arbitrary machine M realizing f_1 by having M_1 simulate M until M halts (using coins to simulate M's nondeterministic steps), and then accepting iff all remaining coin flips are heads.)

Let M_3 be the machine that flips $p(n) + 1$ coins, simulating M_1 if the first coin flip is heads, and if the first coin flip is tails simulating M_2 and accepting iff M_2 rejects.

It is easy to verify that, out of the $2^{p(|x|)+1}$ probabilistic sequences, exactly $f_1(x)+(2^{p(|x|)}-f_2(x))$ are accepting. This number is $\frac{1}{2}$ iff $g(x) = 0$.

((4) $\Longrightarrow$ (3)) and ((3) $\Longrightarrow$ (2)) are obvious.

For ((2) $\Longrightarrow$ (1)), let f and g be the functions in FL and GapL, respectively, defining language A. Note that $g - f$ is in GapL, and $x \in A$ iff $g(x) - f(x) = 0$. ∎

Theorem 14 *The set of all singular integer matrices is complete for $C_=L$ under $\leq_m^{\log}$ reductions.*

Proof. The proof is immediate from Theorem 1. ∎

More generally, the set $\{(A, r) :$ the determinant of matrix A is $r\}$ is complete for $\mathrm{C}_=\mathrm{L}$, just as the set $\{(A, r) :$ the determinant of matrix A is $\geq r\}$ is complete for PL. (For analogous results concerning $\mathrm{C}_=\mathrm{P}$, see [40].)

Note that NL is contained in $\mathrm{C}_=\mathrm{L}$. (This is because NL is closed under complement, and membership in a coNL set is equivalent to having zero accepting computations.) Furthermore, a proof essentially identical to the proof of Theorem 5 shows:

Theorem 15 $\mathrm{C}_=\mathrm{L}^{\mathrm{NL}} = \mathrm{C}_=\mathrm{L}$.

Theorem 16 $\mathrm{C}_=\mathrm{L}$ *is closed under $\leq_m^{\mathrm{FNL}}$-reductions.*

Proposition 17 *$C_=L$ is closed under $\leq_{dtt}^{\mathrm{FNL}}$ and $\leq_{ctt}^{\mathrm{FNL}}$ reductions. (In particular, it is closed under union and intersection.)*

Proof. Let $A \in \mathrm{C}_=\mathrm{L}$, and let $B \leq_{dtt}^{\mathrm{FNL}} A$ via reduction f (so $x \in B$ iff there is an $i \leq n^k$ such that $f(x, i) \in A$. Let A' be the set of all strings (x, i) such that $f(x, i) \in A$. Then $A' \leq_m^{\mathrm{FNL}} A$; let g be the GapL function such that $(x, i) \in A'$ iff $g(x, i) = 0$. Then $x \in B$ iff $\prod_{i=1}^{n^k} g(x, i) = 0$. Theorem 9 now shows that $B \in \mathrm{C}_=\mathrm{L}$.

Similarly, if $B \leq_{ctt}^{\mathrm{FNL}} A$ via reduction f, then $x \in B$ iff $\sum_{i=1}^{n^k} g(x, i)^2 = 0$. ∎

It remains unknown if $\mathrm{C}_=\mathrm{L}$ is closed under complement.

Closure of $\mathrm{C}_=\mathrm{L}$ under $\leq_{\mathrm{dtt}}^{\log}$ reductions is also sufficient to show that $\mathrm{C}_=\mathrm{L}$ contains the class LogFew that was defined and studied in [14]. (We refer the reader to [14] for the definition of this class.)

Theorem 18 LogFew $\subseteq \mathrm{C}_=\mathrm{L}$.

Proof. The definition of LogFew makes it clear that if $A \in$ LogFew, then there is a logspace-computable function f and a #L function g such that $x \in A$ iff $\exists i \leq |x|^c \;\; g(x) = f(x, i)$. Clearly, then, A is $\leq_{\mathrm{dtt}}^{\log}$ reducible to a set in $\mathrm{C}_=\mathrm{L}$. ∎

We have one more result that makes mention of $\mathrm{C}_=\mathrm{L}$. The motivation for this result came not so much from any question about $\mathrm{C}_=\mathrm{L}$ itself, but rather from a question about the relationship between #L and GapL. As these classes are defined, it is obvious that #L is properly contained

in GapL, because GapL contains negatively-valued functions. However, if we choose any natural binary representation of GapL and #L functions (such as sign-magnitude or two's complement, etc.), it is natural to ask if GapL, viewed as a set of functions from Σ^* to Σ^*, contains any functions that are not in #L.

Proposition 19 *If every function $f : \Sigma^* \to \Sigma^*$ in GapL is also in #L, then $C_=L = NL$.*

Proof. Let $A \in \mathrm{C}_=\mathrm{L}$, and let g be the GapL function that defines A. Let $g' = -g^2$. That is, $x \in A \Longrightarrow g'(x) = 0$, and $x \notin A \Longrightarrow g'(x) < 0$.

Let $f : \Sigma^* \to \mathbf{N}$ be the function defined on the natural numbers so that $f(x)$ is the natural number whose binary representation is given by the representation of $g'(x)$. (All that we assume about the the representation chosen for g' is that zero be represented as a string of zeros.) Then it follows that $x \in A \Longrightarrow f(x) = 0$, and $x \notin A \Longrightarrow f(x) > 0$. If $f \in \#\mathrm{L}$, then $A \in \mathrm{NL}$. ∎

6 Reductions to #L and PL

Corollary 12 raises the obvious question of whether the low-order or middle bits of a #L function can be computed using the power of PL. At present, we see no reason to believe that this is possible; it seems plausible that $\oplus$L (i.e., determining the low-order bit of a #L function) is not NC^1-reducible to PL. Any attempt to investigate this question must first make precise what it should mean to try to compute something "using the power of PL."

It is a pleasing fact of complexity theory that most of the "natural" complexity classes can be defined as the class of problems "reducible" to some important problem, where the definition does *not* depend on the particular notion of reduction that is used. For example, nondeterministic logspace can be defined as the class of problems reducible to transitive closure (or $s-t$-connectivity) using any of the notions of reducibility that have been considered (e.g., $\leq_m^{\log}$, AC^0, NC^1, first-order projections, etc.).

However, the class of problems reducible to the determinant and to PL seem to be exceptions to this rule. As an extreme example, consider the class of functions that are logspace many-one reducible to the determinant; by Theorem 1 this is GapL. The zero-one functions (i.e., languages) in this class are determined by logspace-bounded nondeterministic Turing machines, where the number of accepting and rejecting computations differ by either zero or one; there seems to be no reason to believe that all of nondeterministic logspace is contained in that class. On the other hand, it is obvious that nondeterministic logspace can be reduced to the determinant if less restrictive reductions are used (since all that is required is to check if a #L function is non-zero).

When Cook [15] considered the class of problems reducible to the determinant, he framed his definition using NC^1 reducibility. Alternatively, one could consider the class of things AC^0-reducible to the determinant; we show that this class corresponds to a logspace-analog of the counting hierarchy. (We do not know if this class is equal to the class considered by Cook, but it follows easily from the results of this section that if $NC^1(DET)$ is equal to $AC^0(DET)$, then the #L Hierarchy we define below collapses at some level.)

The counting hierarchy (see [50, 46]) consists of sets in the classes PP, PP^{PP}, $PP^{PP^{PP}}$, etc. Since PP is contained in $C_=P^{C_=P}$ [46], it follows that the same class of sets results if the hierarchy is defined in terms of $C_=P$ instead of PP. One obtains a computationally-equivalent class of functions if one considers #P, $\#P^{\#P}$, etc. Let us now consider the analogous classes defined in terms of #L and PL.

Definition 3 *(The #L Hierarchy) Define* $\#LH_1$ *to be* #L*, and let* $\#LH_{i+1}$ *be the class of functions* f *such that, for some logspace-bounded nondeterministic oracle Turing machine* M*, and some function* $g \in \#LH_i$*,* $f(x)$ *is the number of accepting computations of* M *on input* x *with oracle* g*. Let* #LH *denote* $\bigcup_i \#LH_i$*.*

(The PL *Hierarchy) Define* PLH_1 *to be* PL*, and let* PLH_{i+1} *be the class of languages* A *such that, for some logspace-bounded probabilistic oracle Turing machine* M*, and some language* $B \in PLH_i$*,* A *is the language accepted by* M *with oracle* B *(where acceptance is defined using the* PL *acceptance criterion). Let* PLH *denote* $\bigcup_i PLH_i$*.*

(The $C_=L$ *Hierarchy) Define* $C_=LH_1$ *to be* $C_=L$*, and let* $C_=LH_{i+1}$ *be the class of languages* A *such that, for some logspace-bounded probabilistic oracle Turing machine* M*, and some language* $B \in C_=LH_i$*,* $x \in A$ *iff* M *with oracle* B *accepts* x *with probability exactly* $\frac{1}{2}$*. Let* $C_=LH$ *denote* $\bigcup_i C_=LH_i$*.*

In all of these definitions, the "Ruzzo-Simon-Tompa" relativization method is used (cf. Section 3).

We will show that PLH, $C_=LH$, and #LH can be defined in terms of AC^0 reductions to PL, $C_=L$, and #L, respectively. An AC^0 reduction is a uniform family of constant-depth circuits of unbounded-fan-in AND and OR gates, NOT gates, and "oracle" gates. If the oracle is a function $f : \{0,1\}^* \to \{0,1\}^*$, then an oracle gate has some number m of inputs $x_1, \ldots, x_m$ and some number of outputs, representing $f(x_1, \ldots, x_m)$. If the function f does not always take inputs of length m to outputs of the same length, then it may be necessary to assume some sort of "end-of-string" encoding; our theorems do not depend on these details. If the oracle is a language, then the oracle gate computes the characteristic function of the language. Our results do not depend very much on

the particular uniformity condition used. Logspace-uniformity is sufficient, although the theorems also hold if more restrictive uniformity conditions are used; see [12, 10] for discussions of uniformity issues involved in low-level circuit complexity.

It will turn out to be useful to consider certain refinements of AC^0 reducibility. Following [4] (see also [6]), define $\mathrm{AC}^0_i(f)$ to be the class of languages accepted by AC^0 circuits with oracle gates for f, where no path from input to output goes through more than i oracle gates. If $\mathcal{C}$ is a class of functions (languages), then $\mathrm{AC}^0_i(\mathcal{C})$ is equal to the union over all (characteristic) functions f in $\mathcal{C}$ of $\mathrm{AC}^0_i(f)$.

Theorem 20 *For all i, $\mathrm{AC}^0_i(\mathrm{PL}) = \mathrm{PLH}_i$.*

Proof. It should be stated at the outset that it has recently been shown by Ogihara that this entire hierarchy collapses to PL [35]. ($\supseteq$) Any set $A \in \mathrm{PL} = \mathrm{PLH}_1$ can be accepted with a single oracle gate. Thus assume that $\mathrm{PLH}_i \subseteq \mathrm{AC}^0_i(\mathrm{PL})$, and let A be the language accepted by M^B for some B in PLH_i and some probabilistic logspace-bounded oracle Turing machine M. Since M makes queries using the Ruzzo-Simon-Tompa restriction, there is a list $y_1, \ldots, y_{n^k}$ computable in logspace from x such that all queries of M on input x come from the set $\{y_1, \ldots, y_{n^k}\}$. The set $B' = \{(x, i) : y_i \in B\}$ is also in $\mathrm{PLH}_i \subseteq \mathrm{AC}^0_i(\mathrm{PL})$. (Without loss of generality, $B' \in \mathrm{AC}^0_i(C)$, where C is the "canonical complete set" for PL: $C = \{(Y, z) : Y$ is a transition matrix showing, for all configurations of a logspace-bounded machine, the probability of getting from one configuration to another, such that the machine accepts $z\}$.)

The $\mathrm{AC}^0_{i+1}(C)$ circuit accepting A consists of $\mathrm{AC}^0_i(C)$ circuits computing membership of (x, i) in B', followed by a layer of AND and OR gates computing the transition matrix Y of M on input x with oracle answers given by the subcircuits for B', followed by one more oracle gate, checking if (Y, x) is in C.

($\subseteq$) We prove only the induction step ($\mathrm{AC}^0_i(\mathrm{PL}) \subseteq \mathrm{PLH}_i$ implies $\mathrm{AC}^0_{i+1}(\mathrm{PL})$ ($\subseteq$) PLH_{i+1}); the proof for the basis is essentially identical.

Let A be accepted by an $\mathrm{AC}^0_{i+1}(B)$ circuit family $\{C_n\}$ for some B in PL. We describe the behavior of a probabilistic oracle machine M accepting A. On input x, M looks at circuit C_n. If the output gate of C_n is an oracle gate, then M accepts iff y is in B, where $y = y_1 y_2 \ldots y_r$ is the word input to the oracle gate. The subcircuit computing each bit y_i is an $\mathrm{AC}^0_i(B)$ circuit, and by induction hypothesis can be simulated using an oracle from PLH_i.

On the other hand, if the output gate is an AND, OR, or NOT gate, then the output of the circuit can be computed by a constant number of applications of $\leq^{\log}_{\mathrm{ctt}}$ or $\leq^{\log}_{\mathrm{dtt}}$ reductions to a set

in PLH_{i+1}. But this also defines a set in PLH_{i+1}, by the closure properties established in the preceding section.

(Alternatively, one may observe that $\mathrm{AC}^0_1(\mathrm{PL}) = \mathrm{PL}$, and that an $\mathrm{AC}^0_i(\mathrm{PL})$ circuit may be viewed as consisting of i layers of $\mathrm{AC}^0_1(\mathrm{PL})$ subcircuits.) ∎

Theorem 21 *For all i,* $\mathrm{C}_=\mathrm{LH}_i \subseteq \mathrm{AC}^0_i(\mathrm{C}_=\mathrm{L}) \subseteq \mathrm{L}^{\mathrm{C}_=\mathrm{LH}_i}$.[3]

Proof. The first inclusion has a proof essentially identical to the first inclusion proved in Theorem 20. The second inclusion is also proved by a proof essentially identical to the proof of Theorem 20, but since $\mathrm{C}_=\mathrm{L}$ is not known to be closed under complement, we cannot "absorb" any NOT gates, and thus we obtain only the inclusion $\mathrm{L}^{\mathrm{C}_=\mathrm{LH}_i}$. ∎

Theorem 22 $\#\mathrm{LH}_i \subseteq \mathrm{AC}^0_i(\#\mathrm{L}) \subseteq \mathrm{L}^{\#\mathrm{LH}_i}$.

(We remark that $\#\mathrm{LH}_i$ is defined as a class of *functions*, while $\mathrm{AC}^0_i(\#\mathrm{L})$ is defined as a class of *languages*. Note however that it is quite natural to view a circuit as computing a function; moreover, it is well-known that the set of functions defined in this way is exactly the same as the set of functions f such that the language $\{(x, i, b) : \text{bit } i \text{ of } f(x) \text{ is } b\}$ is in the associated language class. It follows from the proof below that $\#\mathrm{LH}_i$ is actually equal to the class of functions computed by $\mathrm{AC}^0_i(\#\mathrm{L})$ circuits, with the added restriction that the output gate be an oracle gate.)

Proof. The first inclusion is obvious when $i = 1$; thus assume the induction hypothesis for i and let f be in $\#\mathrm{L}^g$ for some $g \in \#\mathrm{LH}_i$. Thus there is some $f' \in \#\mathrm{L}$ such that $f(x) = f'(x, g(y_1), \ldots, g(y_{n^k}))$, where (as in the proof of the previous theorem) the list of y_i's contains all possible queries that could be asked on input x, and this list can be generated in logspace from x. Let $g'(x, i) = g(y_i)$, and note that g' is also in $\#\mathrm{LH}_i$, and by induction has $\mathrm{AC}^0_i(h)$ circuits for some $h \in \#\mathrm{L}$. Let $k(0, z) = h(z)$, and $k(1, z) = f'(z)$. Our $\mathrm{AC}^0_{i+1}(k)$ circuits for f consist of subcircuits computing $g(y_i) = k(0, x, i)$, followed by a gate computing $f(x) = k(1, x, g(y_1), \ldots, g(y_{n^k}))$ at the root.

(We remark that, with a little more effort, $\#\mathrm{LH}_i$ functions can be computed by circuits consisting of exactly i $\#\mathrm{L}$ oracle gates, where the outputs of one gate feed directly into the gate on the next layer.)

The proof of the second inclusion is quite similar to the proof of the forward inclusion in the preceding theorem. ∎

[3]Note that this corrects a mis-statement in the version of this paper that appeared in Proc. 9th IEEE Structure in Complexity Theory Conference, 1994. It is also worth noting that it has recently been shown that this entire hieararchy collapses to $\mathrm{L}^{\mathrm{C}=\mathrm{L}}$ [2].

Corollary 23

- $\#\mathrm{LH} = \mathrm{AC}^0(\#\mathrm{L}) = \mathrm{AC}^0(\mathrm{DET})$
- $\mathrm{PLH} = \mathrm{AC}^0(\mathrm{PL})$
- $\mathrm{C}_=\mathrm{LH} = \mathrm{AC}^0(\mathrm{C}_=\mathrm{L})$

(Part 2 of this corollary was previously observed by Carsten Damm and Peter Rossmanith [24, 38].)

The reader should not be alarmed by the fact that L^{PL} is contained in $\mathrm{AC}^0(\mathrm{PL})$ (and in fact the containment may even be proper), even though AC^0 is properly contained in L; we refer the reader to the discussion in [51, 52].

Some of the motivation for this study came from the question of whether or not $\mathrm{L}^{\#\mathrm{L}} = \mathrm{L}^{\mathrm{DET}} = \mathrm{L}^{\mathrm{PL}}$, by analogy to the equalities that are known to hold for the related classes #P and PP. The reader should be able to see that $\mathrm{L}^{\#\mathrm{L}}$ and $\mathrm{L}^{\mathrm{DET}}$ are indeed equal and contain L^{PL}, but we do not see how to compute the low-order or middle bits of a #L function in L^{PL} or even in $\mathrm{NC}^1(\mathrm{PL})$.

A related question is whether PLH is equal to $\mathrm{C}_=\mathrm{LH}$ (equivalently, whether PL is AC^0-reducible to $\mathrm{C}_=\mathrm{L}$).

We close this section with an observation showing that $\mathrm{L}^{\#\mathrm{L}}$ can be defined using very restricted access to the oracle. (For related observations, see [5, Proposition 2 and Section 6].)

Proposition 24 *Let $\mathcal{C}$ be any of the classes* L, NL, PL, or #L. *Then the class $\mathcal{C}^{\#\mathrm{L}}$ remains unchanged even when the oracle Turing machines defining the class are restricted to make at most one query to the oracle, and to read the answer just once from left to right.*

Proof. Let an oracle Turing machine M be given, and let the oracle be the function g. Let the run time of M be n^k, let the possible queries on an input x of length n be contained in the set $\{h(x,1),\ldots,h(x,n^l)\}$ (where h is logspace-computable), and let the number of bits in $g(y)$ be $|y|^{c'}$. Let c be greater than k and c'.

Then the function f defined by

$$\sum_{j=0}^{n^k}\sum_{i=1}^{n^l}(g(h(x,i))+2^{|y_i|^{c'}+1})2^{(jn^l+i)n^c}$$

is in #L.

Note that $f(x)$, viewed as a binary string, consists of n^k repetitions of a string containing n^l fields, where each field is of length n^c, consisting of leading zeros, followed by a one, followed by the encoding of $g(y_i)$ for some query y_i that could be asked on input x. If a machine asks the query $f(x)$ at the beginning of its computation, then it can simulate M on input x, moving one block to

the right for each step of M that is simulated, and keeping track of which cell of M's oracle tape is being scanned at any particular moment. ∎

7 Conclusion

By making use of recent theorems relating #L and the determinant, we have given simplified proofs of some facts concerning PL, and we have proved new results concerning the classes of sets reducible to PL and to the determinant. A number of open questions remain. Among them:

- What closure properties can be established for #L and GapL? In regard to this question, it is appropriate to mention that GapL can be characterized as the class of functions computed by "skew" arithmetic circuits [45]. Related results regarding arithmetic circuits and certain subclasses of TC^1 may be found in [3].
- Can any relationship be established between PL or #L and AC^1 (or logCFL)?
- Can any any relationship be established between PL (or $\mathrm{C}_=\mathrm{L}$) and bounded-error probabilistic logspace? Note in this regard that David Zuckerman and Mauricio Karchmer have recently shown that no "black-box" simulation of PL can yield a bounded-error probabilistic logspace algorithm for PL [54].
- Can anything more be said about the relationship between PLH and $\mathrm{NC}^1(\mathrm{PL})$ (or #LH and $\mathrm{NC}^1(\#\mathrm{L})$)? Note that for many classes $\mathcal{C}$ of interest (including AC^k, NC^k, NL, L, NP [4, 34]), $\mathrm{AC}^0(\mathcal{C})$ is equal to $\mathrm{NC}^1(\mathcal{C})$.

Acknowledgments

The first author thanks Birgit Jenner for several motivating and informative discussions. Dave Barrington, Michelangelo Grigni, Sanjeev Saluja, Heribert Vollmer, and Pierre McKenzie also provided helpful information. We thank Carsten Damm for constructive comments on an earlier draft.

References

[1] E. Allender, *Oracles vs Proof techniques that do not relativize*, Proc. SIGAL International Symposium on Algorithms, 1990, Lecture Notes in Computer Science 450, pp. 39–52.

[2] E. Allender, R. Beals, and M. Ogihara, *The complexity of matrix rank and feasible systems of linear equations*, manuscript.

[3] E. Allender and J. Jiao, *Depth reduction for noncommutative arithmetic circuits*, Proc. 25th STOC, 1993.

[4] C. Àlvarez, J. Balcázar, and B. Jenner, *Adaptive logspace reducibility and parallel time*, Mathematical Systems Theory 28 (1995) 117–140.

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

[6] J. Balcázar, *Adaptive logspace and depth-bounded reducibilities*, Proc. 6th IEEE Structure in Complexity Theory Conference, 1991, pp. 240–254.

[7] S. Berkowitz, *On computing the determinant in small parallel time using a small number of processors,* Information Processing Letters 18 (1984) 147–150.

[8] A. Borodin, S. Cook, and N. Pippenger, *Parallel computation for well-endowed rings and space-bounded probabilistic machines*, Information and Control 48 (1983) 113–136.

[9] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, and M. Tompa, *Two applications of inductive counting for complementation problems*, SIAM Journal on Computing 18 (1989) 559–578.

[10] S. Buss, S. Cook, A. Gupta, and V. Ramachandran, *An optimal parallel algorithm for formula evaluation*, SIAM Journal on Computing 21 (1992) 755–780.

[11] A Ben-Dor and S. Halevi, *Zero-one permanent is #P-complete, a simpler proof*, Proc. 2nd Israel Symposium on Theory of Computing and Systems (ISTCS93), IEEE press.

[12] D. Barrington, N. Immerman, and H. Straubing, *On uniformity within NC^1*, Journal of Computer and System Sciences 41 (1990) 274–306.

[13] R. Beigel, N. Reingold, and D. Spielman, *PP is closed under intersection*, Journal of Computer and System Sciences 50 (1995) 191–202.

[14] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel, *Structure and Importance of Logspace MOD-Classes*, Mathematical Systems Theory 25 (1992) 223–237.

[15] S. Cook, *A taxonomy of problems with fast parallel algorithms*, Information and Control 64 (1985) 2–22.

[16] C. Damm, $DET = L^{\#L}$?, Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[17] S. Fenner, L. Fortnow, and S. Kurtz, *Gap-definable counting classes*, Journal of Computer and System Sciences 48 (1994) 116–148.

[18] L. Fortnow and N. Reingold, *PP is closed under truth-table reductions*, Proc. 6th IEEE Structure in Complexity Theory Conference, 1991, pp. 13–15.

[19] J. Gill, *Computational complexity of probabilistic Turing machines*, SIAM Journal on Computing 6 (1977) 675–695.

[20] J. Gill, J. Hunt, and J. Simon, *Deterministic simulation of tape-bounded probabilistic Turing machine transducers*, Theoretical Computer Science 12 (1980) 333–338.

[21] F. Green, J. Köbler, K. Regan, T. Schwentick and J. Torán, *The power of the middle bit of a #P function*, to appear in Journal of Computer and System Sciences.

[22] S. Gupta, *The power of witness reduction*, to appear in SIAM Journal on Computing. A preliminary version appeared in Proc. 6th IEEE Structure in Complexity Theory Conference, 1991, pp. 43–59.

[23] D. Isaacson and R. Madsen, Markov Chains, Theory and Applications, Wiley and Sons, 1976.

[24] B. Jenner, personal communication.

[25] H. Jung, *Relationships between probabilistic and deterministic tape complexity*, Proc. 10th MFCS, Lecture Notes in Computer Science 118, 1981, pp. 339–346.

[26] H. Jung, *On probabilistic tape complexity and fast circuits for matrix inversion problems*, Proc. 11th ICALP, Lecture Notes in Computer Science 172, 1984, pp. 281–291.

[27] H. Jung, *On probabilistic time and space*, Proc. 12th ICALP, Lecture Notes in Computer Science 194, 1985, pp. 310–317.

[28] H. Jung, *Stochastische Turingmaschinen und die Kompliziertheit arithmetischer Probleme*, doctoral dissertation, Humboldt-Universität, East Berlin.

[29] R. Ladner and N. Lynch, *Relativization of questions about log space computability*, Mathematical Systems Theory 10 (1976) 19–32.

[30] R. Ladner, N. Lynch, and A. Selman, *A comparison of polynomial-time reducibilities*, Theoretical Computer Science 1 (1975) 103–123.

[31] I. Macarie, *Space-efficient deterministic simulation of probabilistic automata*, Proc. 11th STACS, Lecture Notes in Computer Science 775, 1994, pp. 109–122.

[32] I. Macarie, *Space-bounded probabilistic computation: old and new stories*, SIGACT News Complexity Theory Column 10 (edited by Lane Hemaspaandra), SIGACT News 26, number 3 (September, 1995) pp. 2–12.

[33] N. Nisan, $RL \subseteq SC$, Computational Complexity 4 (1994) 1–11.

[34] M. Ogihara, $NC^k(NP) = AC^{k-1}(NP)$, Proc. 11th STACS, Lecture Notes in Computer Science 775, 1994, pp. 313–324.

[35] M. Ogihara, *The PL Hierarchy collapses*, University of Rochester Computer Science Technical Report 587, June, 1995.

[36] M. Ogiwara and L. Hemachandra, *A complexity theory for feasible closure properties*, Journal of Computer and System Sciences 46 (1993) 295–325.

[37] K. Regan and T. Schwentick, *On the power of one bit of a #P-function*, Proc. 4th Italian Conference on Theoretical Computer Science, World Scientific Press, Singapore, 1992, pp. 317–329. See also [21].

[38] P. Rossmanith, personal communication.

[39] W. Ruzzo, J. Simon, and M. Tompa, *Space-bounded hierarchies and probabilistic computation*, Journal of Computer and System Sciences 28 (1984) 216–230.

[40] S. Saluja, *A note on the permanent problem*, Information Processing Letters 43 (1992) 1–5.

[41] J. Simon, *On tape-bounded probabilistic Turing machine acceptors*, Theoretical Computer Science 16 (1981) 158–167.

[42] J. Simon, *Space-bounded probabilistic Turing machine complexity classes are closed under complement*, Proc. 13th STOC, 1981, pp. 158–167.

[43] J. Simon, J. Gill, and J. Hunt, *On tape-bounded probabilistic Turing machine transducers*, Proc. 19th FOCS, 1978, pp. 107–112.

[44] S. Toda, *Counting problems computationally equivalent to the determinant*, Technical Report CSIM 91-07, Dept. Comp. Sci. and Inf. Math., Univ. of Electro-Communications, Tokyo, 1991.

[45] S. Toda, *Classes of arithmetic circuits capturing the complexity of computing the determinant*, IEICE Trans. Inf. and Syst., vol. E75-D (1992) 116–124.

[46] J. Torán, *Complexity classes defined by counting quantifiers*, Journal of the ACM 38 (1991) 753–774.

[47] L. Valiant, *The complexity of computing the Permanent*, Theoretical Computer Science 8 (1979) 189–201.

[48] L. Valiant, *Why is Boolean complexity theory difficult?* in *Boolean Function Complexity*, edited by M. S. Paterson, London Mathematical Society Lecture Notes Series 169, Cambridge University Press, 1992.

[49] V. Vinay, *Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits*, Proc. 6th IEEE Structure in Complexity Theory Conference (1991) 270–284.

[50] K. Wagner, *The complexity of combinatorial problems with succinct input representation*, Acta Informatica 23 (1986) 325–356.

[51] C. Wilson, *Relativized NC*, Mathematical Systems Theory 20 (1987) 13–29.

[52] C. B. Wilson, *Decomposing NC and AC*, SIAM Journal on Computing 19 (1990) 384–396.

[53] V. Zankó, *#P-completeness via many-one reductions*, International Journal of Foundations of Computer Science 2 (1991) 77–82.

[54] D. Zuckerman, personal communication.