Alternative Notions of Approximation and Space-Bounded Computations

by

Alina Beygelzimer

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Supervised by

Mitsunori Ogihara

Department of Computer Science
The College
Arts and Sciences

University of Rochester
Rochester, New York

April 2003

Abstract

We investigate alternative notions of approximation for problems inside P (deterministic polynomial time), and show that even a slightly nontrivial information about a problem may be as hard to obtain as the solution itself. For example, we prove that if one could eliminate even a single possibility for the value of an arithmetic circuit on a given input, then this would imply that the class P has fast (polygarithmic time) parallel solutions. In other words, this would constitute a proof that there are no inherently sequential problems in P, which is quite unlikely. The result is robust with respect to eliminating procedures that are allowed to err (by excluding the correct value) with small probability.

We also show that several fundamental linear algebra problems are hard in this sense. It turns out that it is as hard to substantially reduce the number of possible values for the determinant and rank as to compute them exactly.

Finally, we show that (in some precise sense) randomness can be nontrivially substituted for nondeterminism in space. Although it is believed that randomness does not give more than a constant factor advantage in space over determinism, it is not even known whether it is no more powerful than nondeterminism. We will show that the latter is true for a restricted version of probabilistic logspace, where the error is potentially larger than what can be achieved by amplification.

Table of Contents

Abstract **ii**

List of Figures **v**

1 Introduction **1**

2 Preliminaries **7**

2.1 Models of Computation 8

2.1.1 Circuits 10

2.1.2 Uniformity 12

2.1.3 Circuit Classes 13

2.2 Reductions 15

2.2.1 Oracle Computations 16

2.2.2 Reducibility Notions 19

2.2.3 Counting Classes 20

2.3 Approximation Notions 25

3 The Enumerability of P Collapses P to NC **34**

3.1 Introduction 34

3.2 Preliminaries 36

3.3 Finite Fields of Large Characteristic 37
3.4 Main Theorem 43
3.5 Probabilistic Enumerators 51

4 The (Non)Enumerability of the Determinant and the Rank 53
4.1 History and Motivation 53
4.2 Preliminaries 55
4.3 Enumerability of the Rank 59
4.3.1 Computing the Rank 59
4.3.2 Computing the Number of Dependent Rows 67
4.3.3 The Graph of Rank 68
4.4 Enumerability of the Determinant 69
4.5 Final Remarks 72

5 A Note on Nondeterminism versus Randomness in Space 77
5.1 Toward $\mathrm{BP_HL} \subseteq \mathrm{NL}$ 81

6 Testing Closure Properties 86
6.1 Introduction 87
6.2 Preliminaries 87
6.3 On the Structure of XOR 88
6.3.1 Connection to Testing Linearity 88
6.3.2 Spectrum of Densities 91
6.4 Testing the XOR Property 94
6.5 Testing Based on Random Examples 96

Bibliography 101

List of Figures

4.1 Block-diagonal construction 60

5.1 Known relationships. 79

5.2 Conjectured relationships. 81

1 Introduction

We shall start with a playful problem (a game, in fact). Suppose that you were given a question and asked to find the answer to it among a number of possible answers. Thankfully, you have an option of asking for some incorrect answers to be eliminated leaving you with the correct answer and some wrong ones. Would it make your task much easier? What if only a single possibility can been eliminated? We will see natural problems whose complexity drops dramatically, even if the help is so (seemingly) insignificant. For example, we will prove that if one could eliminate a single possibility for the value of an arithmetic circuit on a given input (over some finite field), then this would constitute a proof that there are no inherently sequential problems in P. Thus excluding a single possibility in this case is as hard as finding the correct value (unless all of P can be efficiently parallelized, which would be incomparably more interesting, but is probably false). Moreover, the result is robust with respect to eliminating procedures that can make mistakes by excluding the correct value, with small probability.

With some insight, the task of excluding candidate answers can be viewed as a certain notion of approximation. Indeed, the correct answer is approximated with a nontrivial subset of possibilities. Hence the above result can be viewed as showing that the circuit value problem is hard to approximate (with respect to this notion). We will also see that many linear algebraic problems are hard in

this sense: For example, it turns out that it is as hard to substantially reduce the number of possible values for matrix rank as to compute it exactly. Likewise, for the determinant and the wealth of problems reducible to it.

This suggests a natural question: What exactly is an approximation? An approximate solution is generally understood to be a solution that comes close to the correct one, in some sense. This is only natural. It is not that clear, however, what the natural definition of "coming close" is. The traditional meaning of "close" is "within some constant relative factor"; substantial research has been done on giving approximation algorithms with such a guarantee, or showing that a problem is not only hard in its exact version, but is also hard to approximate, in this sense. If a problem does not lend itself to such an approximation, does it mean that it is inherently hard? Can it still be easy to approximate in some other, equally natural sense?

For example, consider the problem of deciding whether a given graph has a clique on a fifth of its vertices. Thus, for any $n > 0$, we want to accept n-node graphs that have a clique of order $n/5$, and reject the ones that don't. What is an approximate solution in this case? Would we rather accept an n-node graph whose maximum clique order is close to $n/5$ (say no less than $n/10$), and reject otherwise; or rather accept a graph that is close to some n-node graph with a clique of order $n/5$, while rejecting if it has to be substantially modified to get any such graph? In other words, would we rather have a graph whose cost (max-clique) is close to being optimal, or a graph that is close to a graph with an optimal cost? We will just say that the first approximation task is NP-hard [Ha99], while the second can be solved in time independent of n (depending only on the parameter quantifying "significantly") [GGR98]. Notice that the above question can be seen as asking whether we want our approximation to be a function of the outputs or a function of the inputs. We will return to this discussion later once we probe the definition of approximation.

Even if a problem has a feasible approximation in the traditional sense (for example, a polynomial-time constant approximation scheme), such approximation may be of little use. The focus has shifted from approximating hard functions to approximating very basic properties of the inputs: The main difficulty is to deal with the immense size of the instances and (therefore) restricted access to them. It is often infeasible to even read the input (or any significant parts of it). Decisions must be made based on seeing a small (often random) part of the input, and such decisions must generally be approximations of some kind (and of *different* kind).

In the case when the problem is still interesting by itself (and not just by the fact that it has to be solved on huge instances), we may want to examine how difficult it is to obtain even a slightly nontrivial advantage over guessing in solving it. Recall the example in the beginning of the introduction, where we merely wanted to eliminate a single possibility for the solution. And curiously, even such seemingly weak piece of information may be as hard to obtain as the solution itself. Another question may be of comparative nature: Given several instances, how hard is it to make any meaningful statements about their relative likelihood of being the solution?

Such alternative notions of approximation are of great value. First of all, there does not seem to be the right notion in some absolute sense; one notion can deeply capture the nature of an approximate solution in one problem and be irrelevant in another. Equally importantly, a problem that is hard to approximate in one sense, may nonetheless be simple in some other, equally natural and useful sense. The notions may complement each other, yielding insightful connections between different aspects of computation.

Central to the thesis is the concept of enumerability, introduced by Cai and Hemachandra [CH89]. An *enumerator* for a function f is a procedure that reduces the number of possible values of f on a given input x by enumerating a short list

of possibilities for $f(x)$. This yields a certain notion of approximation: instead of restricting the range of a function value to an interval, an enumerator restricts it to a small set.

Summary of the thesis Chapter 2 defines basic concepts and introduces some of the notation and terminology.

In Chapter 3, we investigate the relationship between the properties of being enumerable and being inherently sequential. We show that if P is enumerable, then P collapses to NC. In particular, if one could eliminate a single possibility for the value of an arithmetic circuit on a given input (over some finite field), then P collapses to NC^3, modulo the complexity of the eliminating procedure. Thus one cannot eliminate any such possibility in NC without showing that all of P is efficiently parallelizable. We also give a deeper collapse of P all the way down to NC^1 in the case when the characteristic of the ground field is sufficiently large (alas under a stronger elimination hypothesis that depends on the characteristic).

In Chapter 4, we investigate the complexity of enumerative approximation of two fundamental problems in linear algebra, computing the rank and the determinant of a matrix. We prove that both are as hard to approximate (in the enumerative sense) as to compute exactly. In particular, we show that if there exists an enumerator that, given a matrix over some finite field, outputs a list of constantly many numbers, one of which is guaranteed to be the rank of the matrix, then it can be determined in AC^0 (with oracle access to the enumerator) which of these numbers is the rank. Thus, for example, if the enumerator is a logspace function, then the problem of computing the rank is in logspace. For the determinant function we establish the following two results: (1) If a logspace procedure could reduce the set of possibilities for the determinant of a given integer matrix to polynomially many, then the determinant can be computed exactly in logspace. (2) For any prime p, if one could eliminate a single possible value for

the determinant modulo p, then the determinant modulo p can be computed in logspace. The notion of enumerability has been studied before. This is the first time, however, it is investigated for functions inside P.

In Chapter 5, we take an alternative, probabilistic look at the logspace playground. We show that (in some precise sense) randomness can be nontrivially substituted for nondeterminism in space. Even though it is believed that BPL (bounded-error probabilistic logspace) is no more powerful than deterministic logspace, it is not even known whether BPL is contained in NL (nondeterministic logspace). We show that this inclusion holds for a restricted version of BPL, where the error is somewhat larger than what can be achieved by amplification. Amplification can make the probability of error exponentially vanishing in the input length. In our restricted version, the error *is* exponentially vanishing in the input length, but the exponent depends on the running time of the machine.

Chapter 6 showcases another approximation task. This time for decision problems. The decision task is relaxed to distinguishing between the case when a given element is in the set and the case when it is "far" from any element of the set. No behavior is required on elements that are close to some member of the set. This yields a natural notion of approximation for decision problems, namely being close to an element of the set. (Recall the discussion about the problem of deciding whether a given graph contains a large clique.) This notion was defined by Goldreich, Goldwasser and Ron [GGR98] (and implicitly by Rubinfeld and Sudan [RS96]). We consider the problem of testing, for any integer $n > 0$, whether a set of vectors in $\mathrm{GF}(2^n)$ forms a subspace. Given oracle access to an unknown set of vectors S (i.e., the ability to ask whether a vector of our choice belongs to S), we are required to decide whether S is a subspace or is "far" from any subspace (i.e., the membership of many vectors would have to be changed to make S a subspace). The goal is to minimize the number of oracle calls. We show that this number can actually be made constant, independent of the dimension of the

vectors. This problem is closely related to the problem of testing linearity, and the main result presented in Chapter 6 can be derived from known results. Our proofs are by no means superior. However, they were obtained independently, and we believe that Chapter 6 is still of interest.

All chapters are self-contained and can be read independently.

2 Preliminaries

This chapter defines basic concepts and introduces some of the notation and terminology we will need in the thesis. Further definitions will be given when needed.

Basic notation We denote the set $\{1, \ldots, n\}$ by $[n]$. A *set system* is a subset of $\mathcal{P}(n)$, the power set of $[n]$. We use the Boolean alphabet $\Sigma = \{0, 1\}$. As usual, Σ^* denotes the set of all finite-length strings over Σ, and Σ^n denotes the set of all strings of length exactly n, for some integer $n \geq 0$. Here a string is a (finite) sequence formed from the elements of Σ. A subset of Σ^* is called a *language* (or a *set*). For a language L, its *characteristic function* $\chi_L : \Sigma^* \to \{0, 1\}$ is defined as $\chi_L(x) = 1$ iff $x \in L$.

The number of elements in a set X is denoted by $|X|$. So is the length of a string. There will be no confusion in either case. We will identify a subset of X with its characteristic function on X. Then, under the identification $X = [n]$, a subset of X is a string in Σ^n (called the *characteristic sequence* of the subset). Also, a set system on X corresponds to a Boolean function on $\mathcal{P}(X)$ (or, equivalently, on Σ^n). We use the same notation both for the set and its characteristic function.

We will indulge in another notational abuse: We write $\langle x_1, \ldots, x_n \rangle$ to denote the n-tuple of $x_1, \ldots, x_n$ as well as the encoding of this n-tuple as a string over Σ,

using a fixed, well-behaved pairing function extended to tuples (computable and invertible in time and space affordable in the context).

For a language L, we let $\bar{L}$ denote its complement, the set of all strings in Σ^* that are not in L; and for a class $\mathcal{C}$ we let co-$\mathcal{C} = \{\bar{L} \mid L \in \mathcal{C}\}$.

For all algebraic concepts, the reader is referred to one of standard texts such as [Lan02, LN94]. All logarithms are base 2, unless the base says otherwise.

2.1 Models of Computation

We assume familiarity with the standard Turing machine model, encoding of problems as infinite languages over finite alphabets, standard complexity classes, and basic relationships among them. For this information the reader is referred to standard sources such as [Pap94a, HU79, HO02].

We will spend most of our time inside P (deterministic polynomial time). More formally, we define DTIME$[t(n)]$ as the class of languages that can be decided by a deterministic Turing machine running in time $O(t(n))$. The class P is $\bigcup_k$ DTIME$[n^k]$. A classical problem complete for P is the Circuit Value Problem.

Similarly, DSPACE$[s(n)]$ is the class of languages decided by a deterministic Turing machine in space $O(s(n))$. For the special case when $s(n) = O(\log n)$, the corresponding class is L. We shall also write L^k for DSPACE$[\log^k(n)]$. NTIME$[t(n)]$ denotes the class of languages that can be decided by a nondeterministic Turing machine in time $O(t(n))$. If $t(n)$ is polynomial in n, the corresponding class is NP, the class of languages with short efficiently verifiable membership proofs. A typical NP language is SAT, the set of all satisfiable Boolean formulas. In fact, SAT is one of the most expressive languages in NP, in the sense that its instances can efficiently encode any problem in NP. (See Section 2.2 for details.) Not surprisingly, NSPACE$[s(n)]$ is the class of languages decided by nondeterministic Turing machines in space $s(n)$, with NL denoting NSPACE$[\log n]$. A

problem characterizing NL is directed graph reachability: Given a pair of vertices in a directed graph, the problem is to decide whether one is reachable from the other.

We also define FP and FL, the functional versions of P and L respectively. A function is said to be in FP if it is computable by some deterministic polynomial time Turing machine. FL is defined analogously, with an aside that the write-only output tape does not participate in the space bound, so the output can be polynomially long. The functions need not be total. All functions considered in this thesis are single-valued.

All the classes we consider are defined in terms of constructible space and time bounds. A function $t(n)$ is *time-constructible* if there exists a deterministic Turing machine that halts after exactly $t(n)$ steps for every input of length n. Similarly, a function $s(n)$ is *space-constructible* if there exists a deterministic Turing machine that uses space exactly $s(n)$ on every input of length n. Nonconstructible bounds may lead to anomalies such as having an empty gap (i.e., no languages in existence) between (nonconstructible) time $t(n)$ and time doubly-exponential in $t(n)$, just because there are no machines whose time bound is in this gap. Also, if $t(n)$ is not constructible, then the class NTIME$[t(n)]$ may depend on whether all branches are required to halt within $t(n)$ steps. For a time constructible bound, the corresponding class remains intact. Most interesting bounds are constructible. The bounds that are not, are generally uninteresting with respect to understanding the complexity of natural problems.

Two relationships among space-bounded classes are worthy of note. The following result suggests that nondeterminism is much less powerful in the context of space than it is in the context of time.

Savitch's Theorem ([Sav70]) NSPACE$[s(n)] \subseteq$ DSPACE$[s^2(n)]$ for any space-constructible bound $s(n) \geq \log n$.

A further evidence is given by the following astonishing (at the time) result.

Theorem (Immerman-Szelepcsenyi[Imm88, Sze87]) NSPACE$[s(n)]$ is closed under complement, where $s(n) \geq \log n$ is a space-constructible bound. In particular, NL is closed under complement.

We give a brief account of time and space hierarchy theorems. Hartmanis and Stearns [HS65] proved that for any time-constructible functions $t(n)$ and $T(x)$ such that $t(n)^2 = o(T(n))$, DTIME$[t(n)]$ is properly contained in DTIME$[T(n)]$. Hennie and Stearns [HS66] proved that the assumption can be weakened to $t(n) \log t(n) = o(T(n))$. We also have hierarchy theorems for deterministic space. If $s(n)$ is space-constructible, then DSPACE$[s(n)]$ is a proper subset of DSPACE$[s(n) \log s(n)]$ [SHL65]. A hierarchy theorem for nondeterministic time was first proven by Cook [Coo73a], improved by Seiferas, Fischer, and Meyer [SFM75] (and simplified by Zăk[Zak83]), who showed that NTIME$[t(n)]$ is properly contained in NTIME$[T(n)]$, provided that $t(n+1) = o(T(n))$ and $t(n)$, $T(x)$ are time constructible. We also know that space bounded computations are strictly more powerful than time bounded computations. For any space constructible function $f(n)$, DTIME$[f(n)]$ is strictly contained in DSPACE$[f(n)]$, due to Hopcroft, Paul, and Valiant [HPV77].

2.1.1 Circuits

We view circuits as computing functions rather than recognizing sets. A *Boolean circuit* with n inputs and m outputs is a finite directed acyclic graph with labeled nodes, called *gates*. The n nodes with in-degree zero are labeled with input variables $x_1, \ldots, x_n$. Boolean constants 0 and 1 are treated as additional inputs, and can be fetched from any non-input gate. The m nodes with no outgoing edges are the output nodes. Every other node is labeled with a Boolean function of its

inputs (either $\vee$, $\wedge$, or $\neg$). All $\neg$-gates have in-degree 1. We will make a distinction between *bounded* fan-in circuits (in which all gates have in-degree at most two), and *unbounded* fan-in circuits (in which in-degrees of $\vee$- and $\wedge$-gates are not restricted). The circuit computes a function $f_n : \{0,1\}^n \to \{0,1\}^m$ in the natural way. A Boolean assignment to the input variables inductively (and uniquely) defines the Boolean value of each gate. The *size* of a circuit is the number of gates in it, and the *depth* is the length of the longest path from an input to an output.

As usual, we are interested in computing instances of arbitrary size, and thus will define a *family* of circuits, in which the n-th circuit has $g(n)$ inputs and $h(n)$ outputs, where g and h are nondecreasing polynomially bounded functions. The family computes the function f iff, for all $n \geq 0$, the n-th circuit in the family computes f restricted to inputs of length $g(n)$.

Some natural functions are multi-valued, i.e., their value on any given input may not be unique. All functions considered in this thesis are single-valued. For completeness, however, we follow Cook [Coo85] in treating a function as a family of binary relations (a union over all $n \geq 0$ of $R_n \subseteq \{0,1\}^{g(n)} \times \{0,1\}^{h(n)}$, where g and h are the size parameters of the function). A circuit family computes the function iff for each $n \geq 0$, the function f_n computed by the n-th circuit in the family realizes the associated relation in the sense that for each $x \in \{0,1\}^{g(n)}$, if $R_n(x,y)$ holds for some y, then $R_n(x, f_n(x))$ holds as well.

Arithmetic circuits A natural circuit model for dealing with computations over some algebraic structure $\mathbb{F}$ is the *arithmetic circuit* taking elements and constants of $\mathbb{F}$ as inputs, and having gates for the arithmetic operations over $\mathbb{F}$; all wires carry elements of $\mathbb{F}$. Any such circuit can obviously be described by a string over a finite alphabet, provided that the constants used can be described this way. When talking about arithmetic circuits, we will generally assume that the output gate is unique, and thus a circuit with n inputs computes a function

from $\mathbb{F}^n$ to $\mathbb{F}$. We shall take $\mathbb{F}$ to be the finite field $\mathrm{GF}(q)$ with $q = p^k$ elements. We often write $\mathbb{F}_q$ for $\mathrm{GF}(q)$. For finite fields, the representation of the function computed by a circuit as a polynomial is not unique (since $a^q = a$ for any $a \in \mathbb{F}_q$). Thus a circuit in fact computes a set of polynomials. If we want to compute a polynomial in the ring $\mathbb{F}[x_1, \cdots, x_n]$, we will only require that it is among the polynomials computed by the circuit. We defer the discussion of how the elements of $\mathbb{F}$ are represented until Chapter 3, where we actually use it.

For a perceptive comparison of the two circuit models (arithmetic versus Boolean), see the expositions by Allender [All97b], von zur Gathen [vzG93], and von zur Gathen and Seroussi [vzGS91].

2.1.2 Uniformity

Since we want to make circuit classes comparable to uniform[1] classes defined in terms of time and space, we need to place uniformity restrictions on circuit families. Without such restrictions, one would be allowed to invest unbounded amounts of computation in the construction of every circuit in the family, and we quickly get to such anomalies as polynomial-size circuits that compute undecidable languages. For our purposes, it will be sufficient to use logspace uniformity, meaning that there exists a logspace machine that, on input 1^n, generates a standard encoding [Ruz81] of the n-th circuit in the family. When dealing with arithmetic circuits, we also require algebraic constants to be uniform. Finer (and, with some insight, more thought out) notions of uniformity exist, based not on the efficiency of constructing the circuit but on the efficiency of answering certain question about it, but these notions are irrelevant here. For a detailed treatment of uniformity, the reader is referred to the papers by Ruzzo [Ruz81], and Barrington and Immerman [BI97]. Uniformity issues for arithmetic circuits are discussed by

[1] By uniform we mean that there exists a single machine solving the problem on *all* instances.

Eberly [Ebe89] and von zur Gathen [vzG86]. If uniformity is not stated explicitly, we mean logspace uniformity.

2.1.3 Circuit Classes

For $k \geq 0$, the class NC^k is defined as the class of all functions computable by (logspace uniform) polynomial-size, bounded fan-in Boolean circuits of depth $O(\log^k n)$. The class AC^k is defined as the class of all functions computable by (logspace uniform) polynomial-size, arbitrary fan-in Boolean circuits of depth $O(\log^k n)$. In order to define threshold circuits, we need majority gates. A majority gate outputs 1 if and only if at least half of its inputs have value 1. The class TC^k is defined as the class of all functions computable by (logspace uniform) arbitrary fan-in Boolean circuits composed entirely of majority gates; again, the circuits are of polynomial size and depth $O(\log^k n)$. In the case $k = 0$, the depth (which corresponds to parallel time) is constant.

An important note is in place. We do not make a distinction between circuits computing functions and circuits deciding languages. Thus we will allow ourselves to compare the above circuit classes to both language and function classes. There is a bold (and common) type abuse, but it will cause no confusion. Here is the interface: A function $f : \{0,1\}^* \to \{0,1\}^*$ is said to belong to one of these classes iff the set $A_f = \{\langle x, i\rangle \mid \text{the i-th bit of } f(x) \text{ is } 1\}$ belongs to the class and f is polynomially bounded. In the other direction, a language is in one of these classes iff its characteristic function is in the class. This considerably simplifies notation. Notice that NC^0 is uninteresting as a language class, since the output gate can depend only on constantly many inputs; it is interesting, however, as a function class.

The class NC is the union of NC^k for $k \geq 0$. The classes AC and TC are defined similarly. It is easy to show that, for any $k \geq 0$, $\mathrm{NC}^k \subseteq \mathrm{AC}^k \subseteq \mathrm{TC}^k \subseteq \mathrm{NC}^{k+1}$.

Thus NC = AC = TC. Only one inclusion (besides the trivial $\mathrm{NC}^0 \subset \mathrm{AC}^0$) is known to be proper, namely $\mathrm{AC}^0 \subset \mathrm{NC}^1$. (Actually, a stronger separation is known, namely AC^0 is properly contained in ACC^0, a subclass of TC^0.) Almost no lower bounds are known for classes above TC^0. For all we know, TC^0 (let alone NC^1) may be the same as NP. Again, a number of different lower bound techniques are known for AC^0. (See, for example, [FSS84, Hås87, Ajt83].) This is one of the reasons why TC^0 is interesting.

We catalogue a few facts. AC^0 can add and subtract n-bit numbers. There are (strikingly) deep connections between circuit classes and algebra, with probably the most shocking result being that NC^1 captures exactly the computations over nonsolvable algebras [Bar89]. NC^1 also captures exactly the complexity of basic logic calculations. For example, the problem of evaluating a Boolean formula on a given assignment is complete for NC^1 under AC^0 computable many-one reductions. (See Section 2.2 for definitions.) TC^0 captures exactly the complexity of integer multiplication and division. (A recent breakthrough [Hes01] (see also [ABH01]) showed that division is complete for DLOGTIME-uniform TC^0, a more restrictive version of TC^0 alluded to above. Completeness here is in respect to AC^0-computable *Turing* reductions; TC^0 is not known (or believed) to have problems complete under AC^0-computable many-one reductions.) In some sense, the question of whether NC^1 is equal to TC^0 can be viewed as asking whether basic arithmetic operations are as hard as basic operations in logic. There is another good reason to be interested in TC^0. As we will see in a later section, it has a close connection to the counting hierarchy.

2.2 Reductions

A problem A *reduces* to a problem B if A can be computed using B as an oracle. This tells us that B is at least as hard as A (or, on the brighter side,[2] that A is no harder than B), modulo the complexity of the reduction. Of course for the reduction to be interesting, it actually has to be easier to solve A with B's help than without.[3] Furthermore, the reduction itself must be computationally bounded to be meaningful: If a reduction yields a solution to A, it should not be due to significant computations hidden in the reduction itself. Thus we will at least require that our reductions are weaker than the best known computation for A.

We will consider various types of reductions depending on the form of oracle access and the computational power of the reduction. Each reducibility notion defines an equivalence relation on the set of problems: The less restrictive the reduction, the larger (potentially) the classes it yields. Most complexity classes defined in terms of abstract models of computations, have characterizations as classes of problems reducible to some natural problem under a certain notion of reducibility. For example, a perfectly good alternative way of defining NP is as the class of sets polynomial-time many-one reducible to SAT, the set of all satisfiable Boolean formulas. It is striking that these representative problems capture the nature of abstract computational notions in such a clean (and deep) way. In fact,

[2]Amusingly enough, there is also a bright side to hardness. Computationally hard problems, if we can put our hands on one, give pseudorandomness and cryptography. We will not deal with this side of randomness here. Since we do not know how to prove any explicit problem to be hard, reductions provide us with a useful tool for showing *relative* hardness, hardness relative to other problems that are more firmly believed to be hard.

[3]This is not entirely true. A result saying that A retains its complexity when armed with B, *is* interesting, as says something about the type of information that cannot increase the computational power of A. The notion of lowness, introduced into complexity theory by Schöning [Sch83], provides a formal way to discuss such questions.

a number of complexity classes were originally conceived as a closure of some natural problem under a certain type of reducibility. And, in most cases, these classes were later shown to have interesting structure (for example, unexpected alternative characterizations) that justified their definition.

For most natural problems A and B, if A reduces to B under some reducibility notion, it does so regardless of the notion being used. We will see some exceptions, however. For example, for all we know, the class of functions NC^1-reducible to the determinant may be different from the class of those reducible within AC^0. We have already seen another example: TC^0 is not believed to have complete problems under AC^0 many-one reductions, yet it has natural problems complete for it under less restrictive AC^0 Turing reductions (integer division for one). This stands in striking contrast with the fact that such classes as NEXP, or even EXPSPACE, have complete problems under AC^0 many-one reductions; again, problems that TC^0 seemingly lacks.

But first we need to formally define the computations that get help from an oracle. Once we have them at our disposal, we can define various reducibility notions.

2.2.1 Oracle Computations

Since the only computational models that we essentially use are Turing machines and circuits, we only define oracle computations in these models.

Oracle Turing machines An oracle Turing machine is an ordinary Turing machine equipped with a special one-way oracle tape and two special states: the query state and the answer state. Let B be some oracle. We will generally allow B to be any function, languages treated as Boolean functions. With some abuse of terminology, we will not only reduce sets to sets, and functions to functions,

but we will also speak of reductions from a function to a set, and conversely. We will, however, be particular about the details of all reductions we consider. As a general rule, we think in terms of functions, treating sets as Boolean functions.

The oracle machine proceeds with its computation as an ordinary Turing machine until it reaches the query state. When it does so, the contents of the oracle tape is interpreted as an instance of B, and is instantaneously replaced by the value of B on that input (erasing the query string). At the same time, the head of the oracle tape is reset to its initial, left-most position, after which the oracle machine enters the answer state. The answer state marks the point when the machine can start using the contents of the oracle tape in its further computation. The machine is charged for resources the same way it would be if it were oracle-less (with some technical provisos to be discussed). That is, the oracle tape is not subjected to the space bound. The machine, however, is charged for the time it takes to submit its queries and read answers from the oracle, as well as for the workspace used in constructing the queries.

When the machine has oracle access to B, we say that its computation is *relative* to B. All complexity classes that we defined in terms of machines can be relativized. A crucial point is how.

There is no generally accepted definition of (sublinear) space-bounded oracle machines; there does not seem to be *the* right way to define them, and there are some nontrivial delicacies involved. On one hand, if a space-bounded machine is charged for the space it uses on the oracle tape (i.e., if the oracle tape is also subject to a sublinear space bound), then we quickly get such abnormalities as sets trivially not reducible to themselves just because they do not have enough space to write down their queries. On the other hand, if the oracle tape comes for free, then it can be dishonestly used as a workspace, bypassing the space bound. For example, there are oracle worlds (under this definition), in which NL is not even in P. (See [LL76, Lyn76, Sav83] for these and other abnormalities.) Ruzzo,

Simon, and Tompa [RST84] proposed a notion of oracle access that avoids the most obvious such pitfalls, and this is the notion we adopt here. Briefly, the oracle tape is not subjected to the space bound, but the oracle machine (be it nondeterministic or probabilistic) is required to behave *deterministically* when it generates queries. That is, from the time some symbol is written on the oracle tape, the machine has to proceed deterministically until it actually makes the oracle call. This prevents the oracle machine from reaching a doubly-exponential in the space bound $s(n)$ number of configurations, since only exponentially many in $s(n)$ queries can be asked over all computation paths of the machine on a given input. (Indeed, each query depends only on the input string and on the machine's configuration when it starts writing the query.)

The question of how to define time-bounded oracle machines is much less ambiguous. Although, as discussed in [All90], there are some subtleties in this case as well. However, our original definition of oracle machines will serve us here just as well.

Oracle Circuits In order to define circuit-based reductions, we need the notion of oracle circuits. Since we view circuits as computing functions rather then recognizing sets, we will use Cook's generalization of Wilson's model [Coo85, Wil85]. In this model, circuits are allowed to have oracle gates, in addition to the usual gates. An oracle gate takes an ordered list of r inputs, naturally interpreted as a query string. The sequence of s ordered outputs corresponds to the value of the oracle on the query string. For the purpose of defining depth in bounded fan-in circuits, the oracle gate counts as depth $\lceil \log(r+s) \rceil$.

If $\mathcal{C}$ and $\mathcal{D}$ are complexity classes, and $\mathcal{C}$ is defined in terms of Turing machines or circuits, let $\mathcal{C}^{\mathcal{D}}$ denote the complexity class that results from allowing the machines (respectively, circuits) that define $\mathcal{C}$ to use an oracle from $\mathcal{D}$.

2.2.2 Reducibility Notions

We will use only deterministic reductions.

We write $A \leq B$ to denote that a problem A reduces to a problem B. Thus A is no harder than B, modulo the complexity of the reduction. The more computationally powerful the reduction, the weaker this statement. The power of the reduction is determined by the type of oracle access and by the computational complexity of the reduction.

The most restrictive type of oracle access we will consider corresponds to many-one reductions. A *many-one* reduction (denoted $\leq_{\mathrm{m}}$) is a reduction which makes only one oracle query and outputs the answer to that query as its own answer. In the case when A and B are sets, a many-one reduction is a many-one mapping f such that for any input x, $x \in A$ if and only if $f(x) \in B$. If A and B functions, the reduction typically consists of two parts: a part f mapping instances x of A to instances $f(x)$ of B, and a part g recovering from the answer y of $f(x)$ the answer $g(y)$ of x. Often g is just the identity function; if it is not, the reduction still makes one oracle query, but it is allowed some computation before it has to output its answer.

A *truth-table* reduction ($\leq_{\mathrm{tt}}$) is allowed to ask several queries but all must be asked before any of them is answered, i.e., no query can depend on the answers to the previous queries. We may assume without loss of generality that all queries are asked at the same time. The reduction is then allowed to combine the answers in an arbitrary way (within its computational power) to produce its own answer, but it is not allowed to make any further queries. A bound on the number of queries can be made explicit in the notation as a prefix to tt. For example, if the reduction makes at most $f(n)$ queries on inputs of size n, we denote it by $\leq_{f(n)\text{-tt}}$. We will distinguish *bounded* truth-table reductions (denoted $\leq_{\mathrm{btt}}$) as reductions, in which the number of queries is independent of n.

The least restrictive are *Turing* reductions ($\leq_{\mathrm{T}}$). In fact, there is no restriction on how the queries are asked. The query construction is fully adaptive. We will also write $\mathcal{C}(f)$ to denote the class of functions Turing-reducible to f via $\mathcal{C}$-computable reductions, where $\mathcal{C}$ will typically be a circuit complexity class.

The computational complexity of a reduction is indicated as a superscript to $\leq$. Access types and computational bounds can be combined in the natural way. Of course, we define reductions only for classes for which we defined relativization. We will occasionally simplify the notation and write $\mathcal{C}$-reducible to mean $\leq_{\mathrm{T}}^{\mathcal{C}}$-reducible, where $\mathcal{C}$ will typically be a circuit complexity class. All the reductions we consider are transitive, that is if $A \leq B$ and $B \leq C$ under any of these notions, then $A \leq C$. For a much better treatment of reductions, the reader is referred to [HO02].

A problem B is said to be *hard* for a complexity class $\mathcal{C}$ under reduction $\leq_r$, if every problem in $\mathcal{C}$ reduces to B under $\leq_r$. If B itself is in $\mathcal{C}$, then B is called *complete* for $\mathcal{C}$ under $\leq_r$. The *closure* of $\mathcal{C}$ under $\leq_r$ is the set

$$R_r(\mathcal{C}) = \{L \mid (\exists A \in \mathcal{C})[L \leq_r A]\}.$$

If $\mathcal{C} \subseteq R_r(\mathcal{C})$, the class $\mathcal{C}$ is said to be *closed* under $\leq_r$. It is worth emphasizing that if the class $\mathcal{C}$ is closed under $\leq_r$ and $\mathcal{C} \subseteq \mathcal{D}$ for some class $\mathcal{D}$, then for any $\mathcal{D}$-complete problem B (under $\leq_r$), $B \in \mathcal{D}$ iff $\mathcal{C} = \mathcal{D}$. Some closures are worthy of note. In particular, the classes NC^k (for any $k \geq 1$), FL, FP are all closed under NC^1 reductions.

2.2.3 Counting Classes

The class #P is the class of functions definable as the number of accepting computation paths of a nondeterministic polynomial-time Turing machine [Val79b]. Thus a function $f : \Sigma^* \to \mathbb{N}$ is in #P iff there exists an NP machine M such that $f(x) = \#\mathrm{acc}_M(x)$ for all $x \in \Sigma^*$, where $\#\mathrm{acc}_M(x)$ denotes the number of

accepting computation paths of M on input x. The class #L, first considered in [ÀJ93b], is defined similarly, but for nondeterministic logspace machines. Notice that both are classes of functions. If we need a language class, we will talk about $\mathrm{P}^{\#\mathrm{P}}$ and $\mathrm{L}^{\#\mathrm{L}}$. It came as a shock when Toda [Tod89] showed that the polynomial hierarchy PH is contained in $\mathrm{P}^{\#\mathrm{P}}$. In fact, he showed a seemingly stronger inclusion, namely $\mathrm{PP}^{\mathrm{PH}} \subseteq \mathrm{P}^{\#\mathrm{P}}$, where PP is the class of languages accepted by NP machines with the property that any string is in the language if and only if more than half of the paths are accepting. It is easy to see that $\mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#\mathrm{P}}$, i.e., knowing the highest order bit of a #P function is as powerful as knowing all the bits. The class #L, on the other hand, is much less powerful: $\#\mathrm{L} \subseteq \mathrm{TC}^1$. One of the most important results about #P is that it characterizes the complexity of computing the permanent of integer matrices [Val79b]. It took more than a decade to find out that there is an equally close connection between #L and the determinant [Tod91, Val92, Vin91, Dam91]. Since the determinant of integer matrices can take on negative values, the determinant is in fact complete for GapL, the class of functions that can be expressed as the difference of two #L functions. A function f is in GapL if and only if f is logspace many-one reducible to the determinant. As noted in [AO96], the determinant and #L are even more intimately related than the permanent and #P in the sense that the determinant is complete for GapL under a more restrictive notion of many-one reducibility, in which the part that recovers the value of a GapL function on a given input from the determinant of a matrix this input was mapped to, is the identity function. The permanent, on the other hand, is provably not complete for #P (or GapP) under such restrictive notion, unless of course $\mathrm{P} = \oplus\mathrm{P}$, since the permanent and the determinant are the same mod 2. Here $\oplus$P is another counting class, the class of languages accepted by NL machines where the definition of acceptance is defined by the parity (i.e., the least significant bit) of the number of accepting paths [GP86, PZ83].

GapL was studied as the logspace analog of the class GapP, introduced by Fenner, Fortnow, and Kurtz [FFK94], and independently by Gupta [Gup92, Gup95]. (See Ogihara and Hemaspaandra [HO02, p. 106-108] for a historical review.) One of the reasons for defining GapP was to study classes of *languages* that can be defined using *gap* functions, functions expressible as a difference of two #P functions. We will be more interested in the logspace analogs of these classes.

A language A is in PL iff there exists a GapL function f such that for any $x \in \Sigma^*$, $x \in A$ iff $f(x) > 0$. It follows immediately that the set of integer matrices whose determinant is positive is complete for PL. The class PL is better known as the class of languages computed by an unbounded error probabilistic logspace Turing machines, a logspace analog of PP defined above. Some facts: PL is in TC^1 (which follows from [BCP83]), and the machines can be restricted to run in polynomial time with no loss in power [Jun85, AO96]. In contrast, such a restriction seems to have a drastic affect on the power of *bounded* error probabilistic logspace. Without it, one gets a full power of nondeterminism for one-sided error, and a class potentially more powerful then NL, when the error is two-sided. We defer the discussion of probabilistic classes to Section 5, where we investigate the relationship between NL and probabilistic logspace classes. From the above discussion it follows that PL is closed under complement. This had already been known by the time these results appeared [Sim81b, RST84]. Although knowing the highest order bit of a #P function is as powerful as knowing all the bits, this does not seem to be true for logspace. (See [AO96].)

Another related class is $\mathrm{C}_=\mathrm{L}$ [AO96], which is the set of all languages A for which there exists a GapL function f such that for all $x \in \Sigma^*$, $x \in A$ iff $f(x) = 0$. An immediate alternative characterization is in terms of "exact counting": A is in $\mathrm{C}_=\mathrm{L}$ iff there exists an NL machine M such that for all $x \in \Sigma^*$, $x \in A$ iff the machine has the same number of accepting and rejecting paths on x. This class captures exactly the complexity of the set of singular matrices, a far more

interesting set than, say, the set of all matrices whose determinant is positive. The polynomial-time counterpart of $\mathrm{C_=L}$, denoted by $\mathrm{C_=P}$, is due to Simon [Sim75] and Wagner [Wag86a].

Allender and Ogihara [AO96] observed that, even though for most natural problems the closures under AC^0 and NC^1 (Turing) reductions coincide, this does not seem to be apparent for the determinant. This (among other things) motivated the definition of the following hierarchies:

- The exact counting logspace hierarchy $\mathrm{C_=L} \cup \mathrm{C_=L}^{\mathrm{C_=L}} \cup \mathrm{C_=L}^{\mathrm{C_=L}^{\mathrm{C_=L}}} \cdots = \mathrm{AC}^0(\mathrm{C_=L})$

  Allender, Beals, and Ogihara [ABO99] showed that AC^0 and NC^1 reducibilities coincide on $\mathrm{C_=L}$. Furthermore, the hierarchy collapses to $\mathrm{L}^{\mathrm{C_=L}}$; it collapses all the way down to $\mathrm{C_=L}$ if and only if $\mathrm{C_=L}$ is closed under complement.

- The PL hierarchy $\mathrm{PL} \cup \mathrm{PL}^{\mathrm{PL}} \cup \mathrm{PL}^{\mathrm{PL}^{\mathrm{PL}}} \cup \cdots = \mathrm{AC}^0(\mathrm{PL})$

  Ogihara [Ogi98] showed that the PL hierarchy collapses to PL under AC^0 (in fact TC^0) reductions, which was improved to NC^1 reducibility by Beigel and Fu [BF97].

- The #L hierarchy $\mathrm{L} \cup \mathrm{L}^{\#\mathrm{L}} \cup \mathrm{L}^{\#\mathrm{L}^{\#\mathrm{L}}} \cdots = \mathrm{AC}^0(\#\mathrm{L}) = \mathrm{AC}^0(\textsc{Det})$

  It is not known whether the #L hierarchy collapses to any fixed level, or whether $\mathrm{AC}^0(\#\mathrm{L}) = \mathrm{NC}^1(\#\mathrm{L})$. The latter would imply the collapse [All97a].

Analogous hierarchies defined in the context of polynomial time all coincide with the counting hierarchy $\mathrm{PP} \cup \mathrm{PP}^{\mathrm{PP}} \cup \mathrm{PP}^{\mathrm{PP}^{\mathrm{PP}}} \cup \cdots$. Indeed, PP trivially contains $\mathrm{C_=P}$, and $\mathrm{PP} \subseteq \mathrm{C_=P}^{\mathrm{C_=P}}$ due to Torán [Tor91]; and we have already seen that $\mathrm{P}^{\#\mathrm{P}} = \mathrm{P}^{\mathrm{PP}}$.

Why are these hierarchies interesting? First, they (or their base classes) precisely characterize the complexity of many natural problems [Wag86a, ABO99, Coo85]. The counting hierarchy also has strong connections with threshold circuits. The intuition is that levels of the counting hierarchy roughly correspond to circuit depth; moreover, oracle separations for levels of counting hierarchy are equivalent to *real* circuit lower bounds; the equivalence, however, is not exact. The reader is referred to an excellent survey by Allender and Wagner [AW93]. As a motivation, we just mention a concrete example: Constructing an oracle separating PSPACE from the counting hierarchy is the same problem as separating TC^0 from NC^1. In fact, one of the motivations for studying logspace counting hierarchies was the hope that they might help in understanding the power of threshold circuits [ABO99]. No interesting connection has been found; the structure of counting hierarchies in the context of logspace seems to be different from the counting hierarchy in polynomial time. In particular, there is little reason why the three logspace counting hierarchies should be the same [ABO99].

To summarize, we have the following sequence of inclusions.

$$\mathrm{AC}^0 \subset \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \begin{bmatrix} \mathrm{AC}^0(\mathrm{C}_=\mathrm{L}) \\ \mathrm{NC}^1(\mathrm{C}_=\mathrm{L}) \\ \mathrm{L}^{\mathrm{C}_=\mathrm{L}} \end{bmatrix} \subseteq \mathrm{PL} \subseteq \mathrm{AC}^0(\#\mathrm{L}) \subseteq \mathrm{NC}^1(\#\mathrm{L}) \subseteq \mathrm{TC}^1$$

Notice that all the questions we discussed are essentially squeezed between NC^1 and TC^1. No relationship to AC^1 is known (besides the easy $\mathrm{NL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{TC}^1$).

It is worth noting that #P and #L (as well as other counting classes) have characterizations in terms of uniform arithmetic circuits, obtained by arithmetizing Boolean circuits that characterize NP and NL. (Although NP and NL are typically defined in terms of Turing machines, they have characterizations in terms of uniform Boolean circuits.) This arithmetic side of #P and #L will be of no

interest to us, but we refer the reader to a survey by Allender [All97b] as a starting point.

We remark that the treatment of counting classes presented here is by no means comprehensive. It contains a selection of results that is intended to serve as a ground for upcoming discussions. Credit was not given everywhere where it was due. In particular, almost no historical account was given. The reader is referred to far better presentations cited in the section.

2.3 Approximation Notions

In this section, we probe what constitutes an approximation. Conceptually, it is the most important section of this chapter.

An *approximation* of a function f mapping from a set D to a set R, is a function that maps elements of the domain D to *subsets* of the range R. If f is a Boolean function (better thought of as a subset of D), then we define its approximation as a function mapping D to subsets of itself. All functions that we consider are single-valued.

Given a function f, we consider a *family of approximations* $\{A_{f,\epsilon}\}_{\epsilon \geq 0}$, parameterized by the error parameter ϵ. It would be natural to expect that the set output by $A_{f,\epsilon}$ on input x contains $f(x)$, and that $A_{f,0} = \{f(x)\}$. Moreover, the set $A_{f,\epsilon}(x)$ must "come close" to $f(x)$ in some sense. Notice that we require that the set as a whole does so, not necessarily every element of it. We call $A_{f,\epsilon}(x)$ the *ϵ-approximation set* of x with respect to f. We think of $A_{f,\epsilon}$ as a procedure that given x reduces the set of possible values for $f(x)$. We generally expect that if $\epsilon_1 < \epsilon_2$, then A_{f,ϵ_2} is more ambiguous than A_{f,ϵ_1} in that it excludes fewer possibilities for the value of f on any given input. We will see that some well-known notions of approximation have a further property that, for any

x, $A_{f,\epsilon_1}(x) \subseteq A_{f,\epsilon_2}(x)$ whenever $\epsilon_1 < \epsilon_2$. We will also see notions that do not have this *nesting* property.

One important remark is in order. Notice that an approximation so defined is a function of the input, not the *value* of f on that input. A rightful question is why: after all, it is the value of f that we want to approximate. Thus it seems that this value alone should determine what it means to come close to it, yet we will see that in some natural approximation notions the approximating set is indeed a function of the input: two inputs x and y with $f(x) = f(y)$ can have very different approximating sets. Moreover, we will see cases when one would explicitly want an approximating set for x not to depend on $f(x)$.

Needless to say, we expect an approximation to be *easier* (in some computational sense) than the function it approximates.

Interval Approximations

We use this term to refer to what is commonly understood by approximation, mostly in the context of approximating NP-hard optimization problems. An approximate value in this case is a value that is within some factor from the truth. This essentially restricts the set of possible values for $f(x)$ to an interval (therefore the name). The size of this interval is a function of $f(x)$, and thus if $f(x)$ is exponentially large in $|x|$, then so is the interval. Thus the smaller $f(x)$, the less ambiguous the approximation. One can imagine situations where it is not desirable to have an approximation algorithm that favors some inputs by being exponentially less ambiguous on them.

Interval approximations come in different versions. The following are the two most common. As a rule, functions for which these approximation are considered, have metric domain and range. Otherwise, restricting to an interval is not particularly meaningful. Since this notion of approximation is used primarily in

approximating optimization problems, we will assume that $D = \Sigma^*$ (the set of input instances) and R is either $\mathbb{R}$ or $\mathbb{N}$.

Relative approximation Given the error parameter $0 \leq \epsilon < 1$, the approximating set of $x \in D$ is $A_{f,\epsilon}(x) = \{y \in R \mid (1-\epsilon)f(x) \leq y \leq (1+\epsilon)f(x)\}$.

Ratio approximation Given the error parameter $0 \leq \epsilon < 1$, the approximating set of $x \in D$ is $A_{f,\epsilon}(x) = \{y \in R \mid (1-\epsilon)f(x) \leq y \leq f(x)/(1-\epsilon)\}$.

There is also a stronger notion, called *absolute* approximation. Fixing a constant $k > 0$, the interval for input $x \in D$ is given by $A_{f,k}(x) = \{y \in R \mid |y - f(x)| \leq k\}$. But how many problems lend themselves to such strong approximation guarantees? It seems that there is no interesting problem that does. Uninteresting problems are those where the exact value can be easily pinned down to a small range, and the hardest part is to find which one of the values in this small range is exact. A typical example is edge coloring: the problem is to color the edges of a given graph with the smallest possible number of colors so that no two adjacent edges have the same color. It is known that the number of colors is either Δ or $\Delta + 1$, where Δ is the maximum degree of the graph. (See, for example, [MR95].) It is NP-hard, however, to decided which one of the two is the correct number for a given graph, even if the graph is 3-regular. Thus the problem trivially has a polynomial time approximation algorithm with absolute error at most 1.

Enumerative Counting

The notion of enumerability was introduced by Cai and Hemachandra [CH89], who were motivated by investigating alternative ways to approximate #P functions. Instead of restricting the range of a function value to an interval, they considered restricting it to a small set.

Definition 1 [CH89] A $k(n)$-*enumerator* for a function $f : \Sigma^* \to \Sigma^*$ is a procedure that reduces the set of possibilities for the value of f on any given input $x \in \Sigma^*$ by enumerating a list of $k(|x|)$ values guaranteed to contain $f(x)$ (thereby eliminating the rest of the values).

Notice that the number of values on the list depends only on the size of the input; in particular, it does not depend on the *value* of the function on that input. Thus, even if the range is exponential, the enumerator still outputs an equally short list for every input of the same length. Clearly for such a procedure to be interesting, it has to do more (or rather, less) than simply compute $f(x)$ and include it in the output. Thus we stipulate that enumerators require less computational power than the functions they enumerate. If a function f has a k-enumerator, we call it k-*enumerable*. We emphasize that the set *as a whole* approximates $f(x)$.

Enumerative counting makes sense in a more general context when there is no natural ordering on the range of the function, and thus restricting the value to an interval is not particularly meaningful. Such an approximation, however, does not satisfy a natural nesting requirement mentioned above. It is neither strictly stronger nor strictly weaker than relative interval approximations.

The alert reader may have noticed a superficial connection between the enumerative and absolute approximation: They share uninteresting problems, problems whose exact value can be trivially pinned down to a small range.

Cai and Hemachandra [CH91] (see also Amir, Beigel, and Gasarch [ABG90]) showed that if a #P-complete function is poly-enumerable in polynomial time, then $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. Thus it is as hard to substantially reduce the number of possible values for the permanent as to compute it exactly. In Section 3, we show that a similar result holds for the determinant as well.

Beals et al. [BCGT99] investigate the enumerability of #GA (the function computing the number of automorphisms of an undirected graph), and show that

#GA is not poly-enumerable in polynomial time, unless GI (the set of pairs of isomorphic graphs) is in RP (one-sided error probabilistic polynomial time). Furthermore, they show that #GA is not n^ϵ-enumerable in polynomial time for $\epsilon > 1/2$, unless GI is in P. GI is easily seen to be in NP. It is also known to be incomplete for NP unless PH collapses. It was very recently shown that GI sits in SPP [AK02], the class defined in [OH93, FFK94], as the class of languages whose characteristic function is in GapP. SPP can be thought of as the gap-version of UP (unambiguous nondeterminism); it is also exactly equal to the low sets for GapP, and thus for all gap-definable classes, giving another evidence that GI is not NP-complete.

The notion of enumerability has been extended to sets by associating sets with their ranking functions.

Definition 2 [GS91] The function $r : \Sigma^* \to \mathbb{N}$ is called the *ranking function* of a language L if r maps each $x \in \Sigma^*$ to $|\{y \leq x \mid y \in L\}|$.

A language L is said to be *polynomial-time rankable* if its ranking function is computable in polynomial time. Furthermore, L is $k(n)$-*enumeratively rankable* if its ranking function is $k(n)$-enumerable. Hemachandra and Rudich [HR90] show that if every language in P is $O(1)$-enumeratively rankable (or even, for any $\epsilon > 0$, $O(n^{1/2-\epsilon})$-enumeratively rankable) then #P = FP. Their construction was built on the result of Cai and Hemachandra [CH89], and the later, stronger results of Cai and Hemachandra [CH91], in light of the Hemachandra-Rudich argument, in fact show that for each k it holds that if each language in P has a $O(n^k)$-enumerative ranking function, then #P=FP.

In a similar vein, Goldsmith et al. [GOR00] show that if the census function of every language in P is poly-enumerable in polynomial time (or equivalently, if each function in $\#\mathrm{P}_1$ is poly-enumerable in polynomial time), then $\#\mathrm{P}_1 \subseteq \mathrm{FP}$, where $\#\mathrm{P}_1$ is the tally version of #P, for which the inputs are given in unary. They show that $\#\mathrm{P}_1 \subseteq \mathrm{FP}$ would imply that P=BPP, and moreover, that $\mathrm{PH} \subseteq \mathrm{Mod}_k\mathrm{P}$ for

any $k \geq 2$. (The *census function* of L is the function mapping each $n > 0$, as a unary string 1^n, to the number $|L \cap \Sigma^n|$.)

Membership Comparability

Given two instances, can we tell which one is no more likely than the other to be in a predefined set?

Definition 3 [Sel79] A set L is called P-*selective* (or *semi-feasible*) if there is a polynomial time computable function f (called a *p-selector* for L) such that for any two strings x and y, $f(x, y) \in \{x, y\}$, and if x or y is in L then $f(x, y) \in L$.

There are four possibilities for the joint membership of x and y in L. A selector function essentially excludes one such possibility: if it outputs x then $\chi(\langle x, y\rangle) \neq 01$; if it outputs y then $\chi(\langle x, y\rangle) \neq 10$. We will only need a generalization of this notion to a larger number of strings. An excellent account of the theory of semi-feasible computation can be found in a book by Hemaspaandra and Torenvliet [HT02].

Definition 4 A set L is called $g(n)$-*membership comparable* if there is a function f (called a $g(n)$-*comparator* for L) such that for every $x_1, \ldots, x_{g(n)}$ each of length at most n, it holds that $\chi(\langle x_1, \ldots, x_{g(n)}\rangle) \neq f(x_1, \ldots, x_{g(n)})$. In words, f excludes one of the $2^{g(n)}$ possibilities for the membership of the given strings in L.

Beigel et al. [BKS95] introduced this notion for constant g, and called it *approximability*. The fully generalized notion defined above is due to Ogihara [Ogi95]. Membership comparability is a certain notion of approximation for decision problems, and an extremely weak one at that. It gives a negligible (exponentially vanishing in g) advantage over guessing.

Property Testing

Property testing, initiated by Goldreich, Goldwasser and Ron [GGR98] (and implicitly by Rubinfeld and Sudan [RS96]), is a general task of distinguishing between the case when a given object has some predetermined property and the case when it is "far" from any object with this property. Here a property is just a collection of objects (namely, the objects that have the property), and "far" is in the sense that the object has to be substantially modified before it can satisfy the property. Property testing so defined is a relaxation of the standard decision task, where one is required to distinguish between the case when an object has the property (is a YES-instance) and the case when it does not (is a NO-instance). Allowing arbitrary behavior on objects that don't have the property, yet are close to others that do, often gives tests that are significantly more efficient than the corresponding decision procedure. It also yields a natural notion of approximation for decision problems, namely *being close* to a YES-instance.

We will not even attempt to cover the results in property testing. There are excellent expositions on the subject; see [Gol97, GGR98, Ron01, Fish01]. Apart from being theoretically appealing, this notion arises naturally in the context of program testing, probabilistically checkable proofs, hardness of approximation, PAC-learning. All references can be found in the surveys cited above.

The objects we consider are sets of vectors in $\mathrm{GF}(2)^n$, or equivalently set systems on $[n]$. As usual, we associate sets with their characteristic functions, thus a set of vectors in $\mathrm{GF}(2)^n$ will be treated as a function $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$.

Given oracle access to an unknown function f (i.e. the ability to ask the value of f on an input of our choice), we are required to decide whether f satisfies some predefined property or is far from satisfying this property. In some cases we only have access to a random sample from f, i.e., a sequence of pairs of the form $\langle x, f(x)\rangle$. The distance between functions is measured as the fraction of the

domain on which the functions differ. Thus f is said to be *ϵ-far* from a class of functions defined over the same domain, if every function in the class differs from f on more than an ϵ-fraction of the domain. The goal is to minimize the number of oracle calls. Often this number can be made constant, independent of n (depending only on ϵ). This explains why the input function is given as an oracle rather than explicitly as a truth table; otherwise, it would take exponentially less time to make the assertion than to read the input.

Definition 5 A *property* $\mathcal{P} = \{\mathcal{P}_n\}_{n>0}$ is a parameterized class of functions, where the functions in $\mathcal{P}_n$ are from $\mathrm{GF}(2)^n$ to $\mathrm{GF}(2)$.

Definition 6 A *property test* for $\mathcal{P}$ is a probabilistic oracle machine M that given n, a distance parameter $\epsilon > 0$, and oracle access to an arbitrary function $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ satisfies the following requirements: it accepts f if $f \in \mathcal{P}$, and it rejects f if f is ϵ-far from $\mathcal{P}_n$. In both cases M is allowed to err with probability 1/3. The *query complexity* $q(n, \epsilon)$ of the test is the number of oracle calls M makes on inputs of length n. In the case when M queries f on inputs that are uniformly and independently sampled from $\mathrm{GF}(2)^n$, we say that M uses *random examples*.

Definition 7 If $\mathcal{P}$ has a property test with query complexity that depends only on ϵ and not on n, $\mathcal{P}$ is called *testable*.

Not surprisingly, testability is deeply related to whether the property admits simple local characterizations; i.e., whether one can show that a function satisfies certain (easily checkable) local conditions on (typically random) inputs if and only if it is close to having the property. For example, a function $f : \Sigma^n \to \Sigma$ is monotone if and only if $f(x) \leq f(y)$ holds for any $x, y \in \Sigma^n$ such that $x \prec y$, where $\prec$ denotes the natural partial order on Σ^n (i.e., $x = x_1 \ldots x_n \prec y_1 \ldots y_n = y$

if and only if $x_i \leq y_i$ for every i, and $x_i \neq y_i$ for some i). A natural test for monotonicity is to make sufficiently many random local checks; each check may consist of randomly selecting $x \in \Sigma^n$, flipping a random bit in x to obtain $y \in \Sigma^n$, and then verifying that x and y respect the local characterization. If the algorithm finds a pair (x, y) violating the property, it rejects; otherwise, it accepts. It is a feature of the area that, while it is almost immediately apparent *what* to check, the tests are infamously hard to analyse. Clearly, the test above will not reject a monotone function. The hard part is to show that any function accepted by the test with nontrivial probability is nontrivially close to some monotone function. This determines how many times the test should be repeated to guarantee that if f is far from any monotone function, this will be revealed with high probability. For Boolean functions the number of tests can be independent of n, as shown by Dodis et al. [DGL+99] (See also [GGL+00, FLN+02].)

The lack of such local characterizations certainly does not make the testing task easy. Consider, for example, the property of the input function being the parity function. No algorithm that probes the value of the function on a small fraction of inputs can distinguish with high probability between the case when it is parity and the case when it is far from it. In fact, as was shown in [GGR98], there is an abundance of function classes such that a uniformly selected function (over the same domain) is far from it and yet is indistinguishable from it (even if the distinguisher has unlimited computational power), when observing the value of the function on less than a constant fraction of the domain. In other words, there exist properties testing of which is asymptotically as hard as deciding. These observations suggest that it would be interesting to characterize properties according to the complexity of testing them. We will talk about this in more detail in Chapter 6.

3 The Enumerability of P Collapses P to NC

SUMMARY We show that if one could eliminate a single possibility for the value of an arithmetic circuit on a given input (both over some finite field), then P collapses to NC, modulo the complexity of the eliminating procedure. Thus one cannot exclude *any* such candidate in NC without showing that all of P can be efficiently parallelized. The result above is robust with respect to eliminating procedures that err (i.e., exclude the correct value) with small probability. We also give a stronger collapse of P all the way down to NC^1 for the special case when the characteristic of the ground field is sufficiently large (alas under a stronger elimination hypothesis that depends on the characteristic).

3.1 Introduction

We investigate the relationship between the properties of being enumerable and being inherently sequential. We show that if P is enumerable (in some precise sense), then P collapses to NC.

Recall that a $k(n)$-*enumerator* for a function f, defined by Cai and Hemachandra [CH89], is a procedure that reduces the set of possibilities for the value of f on a given input x of length n by enumerating a list of $k(n)$ values guaranteed

to contain $f(x)$ (thereby eliminating the rest of the values). Clearly for such a procedure to be interesting, it has to do something different than simply compute $f(x)$ and include it in the output. Thus we stipulate that enumerators require less computational power than the functions they enumerate. Recall that an enumerator yields a certain approximation of f: Instead of restricting the value of $f(x)$ to an interval (as would a classical approximator), it restricts this value to a small set. If a function f has a $k(n)$-enumerator, we shall call it $k(n)$-*enumerable*. If $k(n)$ is a polynomial, f is said to be *poly*-enumerable.

Consider the task of computing the value of an arithmetic circuit on a given input, both over some finite field GF(q). Suppose that one could eliminate a *single* possibility for this value. (Or, in the above terminology, assume that there exists a $(q-1)$-enumerator for the Arithmetic Circuit Value Problem over GF(q).) We show that then one could solve the Boolean Circuit Value Problem (CVP for short) in NC^3, modulo the complexity of the enumerator. Since CVP is complete for P under NC^1-computable many-one reductions, this would collapse P to NC^i, provided that the enumerator is computable in NC^i for some $i \geq 3$. Put another way, one cannot efficiently eliminate *any* candidate for the value of an arithmetic circuit on a given input without proving that there are no inherently sequential problems (which would be incomparably more interesting, but is probably false). Thus even such seemingly weak piece of information is as hard to obtain in this case as the solution itself.

This result is proven by adapting the techniques used in Sivakumar's proof that SAT is not $O(\log n)$-membership comparable,[1] unless UniqueSAT is in P [Siv99], which builds on the technique of Ar et al. [ALRS98] for reconstructing polynomials from noisy data (see [Sud97, GS92]). The result is also reminiscent of the sparse

[1] A $g(n)$-*membership comparing function* for a language A takes a sequence of $g(n)$ strings, each of length at most n, and outputs a string in $\Sigma^{g(n)}$ that is not their characteristic string in A. (See [Ogi95, BKS95].)

hard sets results [Ogi95, CS98, CS00, vM98].

We initiate the study of probabilistic enumerators that are allowed to err (i.e., exclude the correct value) with small probability. The result above is robust with respect to such enumerators.

Finally, we prove a stronger collapse of P all the way down to (logspace uniform) NC^1 for the special case when the characteristic of the ground field is sufficiently large (alas, under a stronger enumerability hypothesis that depends on the characteristic). In particular, let $\mathbb{F}$ be any fixed finite field with characteristic at least k^2 for some integer $k \geq 2$. We show that $\mathrm{CVP}_{\mathbb{F}}$ is not k-enumerable, unless CVP is solvable in NC^1, modulo the complexity of the enumerator. If $\mathbb{F}$ is not fixed but is given in the input (by its characteristic p, order p^q (in unary notation), and a monic irreducible polynomial $g \in \mathbb{Z}_p[x]$ of degree q), then the construction is in TC^1.

3.2 Preliminaries

Recall that an *arithmetic circuit* over some algebraic structure $\mathbb{F}$ is a circuit taking elements and constants of $\mathbb{F}$ as inputs; all gates perform arithmetic operations over $\mathbb{F}$, and all wires carry elements of $\mathbb{F}$. Any such circuit can clearly be described by a string over a finite alphabet, provided that the constants used can be described this way. A Boolean circuit is just an arithmetic circuit over GF(2).

Since we want to make circuit classes comparable to uniform classes defined in terms of time and space, we need to place uniformity restrictions on circuit families. For our purposes, it will be sufficient to use logspace uniformity, meaning that there exists a logspace machine that, on input 1^n, generates a standard encoding [Ruz81] of the n-th circuit in the family. If uniformity is not stated explicitly, we mean logspace uniformity.

We use Cook's generalization [Coo85] of Wilson's model [Wil85] to define circuit-based reductions. (We do not make a distinction between circuits computing functions and circuits deciding languages.)

An instance of the *arithmetic circuit value problem* over $\mathbb{F}$ (denoted $\mathrm{CVP}_{\mathbb{F}}$) is an encoding of an arithmetic circuit C over $\mathbb{F}$, together with an encoding of inputs $x_1, \ldots, x_n$ from $\mathbb{F}$. The circuit C has n inputs, and the problem is to compute the value of C on $x = \langle x_1, \ldots, x_n \rangle$, denoted $C(x)$.

Ladner [Lad75] introduced the (Boolean) circuit value problem, and showed that it is complete for P (under logspace computable many-one reductions). Cook observed that the problem is actually complete under NC^1 reductions [Coo85]. For infinite fields, the problem is not necessarily in FP, since gate values need not be of polynomial length. It is in FP, however, for any finite field, and remains FP-hard in any field. Correspondingly, it is P-hard to decide whether C on x outputs 1, where 1 is the multiplicative identity of the field. (See [GJH95], Appendix A.1.) The fields we consider are finite. The hardness above is in respect to NC^1-computable many-one reductions; hence it suffices to show that CVP (or $\mathrm{CVP}_{\mathbb{F}}$) is in NC^i to collapse P to NC^i for some $i \geq 1$. We will establish such a collapse under the hypothesis that, given an instance $\langle C, x \rangle$ of $\mathrm{CVP}_{\mathbb{F}}$, one can efficiently eliminate at least *one* of $|\mathbb{F}|$ possibilities for the value of C on x.

3.3 Finite Fields of Large Characteristic

We first present the proof of the special case mentioned in the introduction, and then turn to arbitrary finite fields.

Theorem 1 Let $\mathbb{F}$ be a (fixed) finite field with characteristic at least k^2 for some integer $k \geq 2$. If $\mathrm{CVP}_{\mathbb{F}}$ is k-enumerable, then $\mathrm{CVP}_{\mathbb{F}}$ is in NC^1, modulo the complexity of the enumerator. If $\mathbb{F}$ is described in the input (see below), then the same result holds with TC^1 in place of NC^1.

A Note on Representation To implement arithmetic in a finite field $\mathbb{F}$ with p^q elements, it suffices to have a monic irreducible polynomial g in $\mathbb{Z}_p[x]$ of degree q, since $\mathbb{F}$ can be viewed as $\mathbb{Z}_p[x]/(g)$, the ring of polynomials over $\mathbb{Z}_p$ modulo g. If the field $\mathbb{F}$ is fixed, then such g can clearly be precomputed in the space that we have available for constructing the circuit. If $\mathbb{F}$ is not fixed but depends on the instance length, then an irreducible polynomial defining the representation of $\mathbb{F}$ can be specified as a part of the input; or alternatively, it can be precomputed in logspace if $\mathbb{F}$ is in some specific form that allows for such precomputation. Unfortunately, such explicitly given irreducible polynomials are rare and strongly depend on the structure of p^q. See [Shp92, Chapter 2] and [LN94, Chapter 3] for excellent expositions. Theorem 1 holds if the number of elements in the field $\mathbb{F}$ is polynomial in the instance length n. Thus we allow to supply $\mathbb{F}$ in the input by its characteristic p, order p^q (in unary notation), and a monic irreducible polynomial $g \in \mathbb{Z}_p[x]$ of degree q. Then the theorem holds with TC^1 in place of NC^1. Note that the characteristic of $\mathbb{F}$ is *small*, in the sense that the parallel complexity is measured in terms of $\log p$, rather than $\log\log p$. If g is not given, constructing it may be a bottleneck, as it is in $\mathrm{NC}^{2.5}$ (again, for fields of small characteristic) [Fra91]. In what follows, we assume that the representation of $\mathbb{F}$ has been fixed.

Proof: Let $\langle C, x\rangle$ be an instance of $\mathrm{CVP}_{\mathbb{F}}$, where C is an encoding of an arithmetic circuit over $\mathbb{F}$ with n inputs, and x refers to the encoding of $\langle x_1, \ldots, x_n\rangle \in \mathbb{F}^n$. (We will refer to x as the input to C.) To avoid new notation, assume that C has n gates. Notice that, given an assignment to the gates of C, we can check it for validity in NC^1. Indeed, we just need to locally verify that the value assigned to every gate is in fact the value of the operation it computes on the values assigned to its inputs. Thus all we need is an NC^1-constructible set of gate assignments containing the correct one for C on input x. We shall call such a set a *witness set* for $C(x)$. Needless to say, we cannot simply run the

enumerator on every gate of C (viewing it as an output gate), as it would yield k^n gate assignments, which we cannot hope to verify in NC^1. Instead we will build a new circuit on top of C such that the enumerator will be forced to construct a small witness set for $C(x)$.

Consider any two gates g and h of C. We can run the enumerator to obtain k candidates for the value computed by each on input x. Let the corresponding lists of candidates be $a = \{a_1, \ldots, a_k\}$ and $b = \{b_1, \ldots, b_k\}$. (We may clearly assume that all values on each list are distinct.) We view g as a univariate degree $k-1$ polynomial $P_a(u) = \sum_{j=0}^{k-1} \beta_j u^j$ in $\mathbb{F}[u]$ such that $P_a(a_i) = d(i-1)$ for every $i \in [k]$, where d is a generator of an additive group of $\mathbb{F}$ of size at least k^2. Recall that the promise is that $\mathbb{F}$ has characteristic at least k^2. The multiplicative identity 1 of $\mathbb{F}$ is such a generator, so we can and shall take $d = 1$. Similarly define P_b for h. Thus each of the polynomials is given by its value on a set of k distinct points in $\mathbb{F}$ specified by the enumerator. Since both polynomials are of degree $k-1$, the coefficients (we will follow P_a) can be recovered by interpolation as the solution to the $k \times k$ Vandermonde system

$$\begin{pmatrix} a_1^0 & \cdots & a_1^{k-1} \\ a_2^0 & \cdots & a_2^{k-1} \\ \vdots & \ddots & \vdots \\ a_k^0 & \cdots & a_k^{k-1} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{k-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ k-1 \end{pmatrix}.$$

Since $a_1, \ldots, a_k$ are distinct, the system has full rank over $\mathbb{F}$. Clearly, if k is a constant independent of n, we can find the solution using a circuit of constant size (in fact, it would suffice to simply hardwire the polynomial as a black box with the prescribed behavior). The interesting case is when k is a function of n. (Thus we consider instances of the circuit-value problem where the field that the instance is over depends on the size of the instance.) If $|\mathbb{F}|$ is polynomial in n, we can solve such systems in logspace uniform TC^0 [CS00]. (See also [CS98, Ebe89, Pre91].)

The proofs there are for fields of characteristic 2. The extension to finite fields of characteristic other than 2 is straightforward, since the required operations can easily (by analogy) be seen in logspace uniform TC^0. The main parameter defining the depth and uniformity is n.

Vandermonde system solver

Recall that we have the value of $P_a(u) = \sum_{j=0}^{k-1} \beta_j u^j$ on k distinct points $a_1, \ldots, a_k$ in $\mathbb{F}$. Lagrange interpolation says that

$$P_a(u) = \sum_{i=1}^{k} P_a(a_i) \prod_{z \neq i} \frac{u - a_z}{a_i - a_z}.$$

Collecting the terms corresponding to each u^j gives

$$\beta_j = \sum_{i=1}^{k} (-1)^{i+1} \frac{v_i}{\prod_{z \neq i}(a_z - a_i)} \sigma_{k-j-1}(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k),$$

where σ_r denotes the r-th elementary symmetric polynomial. Let the order of $\mathbb{F}$ be p^q. Recall that $p \geq k^2$, and that p^q (and thus k) are polynomial in n. Using the representation of the elements of $\mathbb{F}$ as q-vectors over $\mathbb{F}_p$, we can add $n^{O(1)}$ elements in logspace-uniform TC^0 (using [CSV84]). We can also multiply any $n^{O(1)}$ elements in TC^0 by using the representation of the nonzero elements of $\mathbb{F}$ as powers of a primitive element ω. Recall some basic facts: The multiplicative group $\mathbb{F}^*$ of $\mathbb{F}$ is cyclic of order $p^q - 1$. Thus there exists $\omega \in \mathbb{F}$ such that $\mathbb{F} = \{0, 1, \omega, \omega^2, \ldots, \omega^{p^q-2}\}$. (In fact, $\mathbb{F}^*$ has exactly $\phi(p^q - 1)$ such generators, namely $\{\omega^i \mid 1 \leq i < p^q - 1 \text{ and } \gcd(i, p^q - 1) = 1\}$.) For all $a \in \mathbb{F}^*$, a can be uniquely written as $a = \omega^i$ for $0 \leq i < p^q$. The value i is the discrete logarithm of a in respect to ω. Now to multiply $n^{O(1)}$ elements of $\mathbb{F}^*$ we first compute their discrete logarithms in respect to ω, which can be done by table lookup in AC^0. The tables themselves can be precomputed in logspace that we have available for constructing the circuit. So can the generator; we can just hunt for it exhaustively in logspace: cycle through all $a \in \mathbb{F}^*$ and verify that there exists $i < p^q$ such that

$\omega^i = a$. Finally, the discrete logarithms are added and the sum is reduced modulo $p^q - 1$, both can be done in logspace uniform TC^0; for the reduction we need a look-up table which can be precomputed in logspace. It remains to show how to compute the elementary symmetric polynomials in logspace uniform TC^0. The j-th elementary symmetric polynomial $\sigma_j \in \mathbb{F}[y_1, \ldots, y_r]$ is given by

$$\sigma_j = \sigma_j(y_1, \ldots, y_r) = \sum_{1 \le i_1 < \cdots < i_j \le r} \prod_{i=i_1}^{i_j} y_i.$$

In words, σ_j is the sum over all j-element subsets $\{i_1, \ldots, i_j\} \subseteq [r]$ of the products $y_{i_1} \cdots y_{i_j}$. Conventionally, $\sigma_0 = 1$. The values of $\sigma_1, \ldots, \sigma_r$ can be quickly recovered by interpolation as the coefficients of the polynomial

$$G(z) = \prod_{i=1}^{r} (z + y_i) = \sigma_0 z^r + \sigma_1 z^{r-1} + \cdots + \sigma_r.$$

The plan now is to take $r = p^q - 1$ and evaluate G on all elements of $\mathbb{F}^*$ represented as powers of ω (no surprise here). Notice that G is in fact a degree $r-1$ polynomial over $\mathbb{F}^*$, since $\sigma_0 = 1$ and $a^r = 1$ for every $a \in \mathbb{F}^*$. Interpolation of G is particularly simple, given by

$$\sigma_j(y_1, \ldots, y_r) = \sum_{i=0}^{r-1} \omega^{-(j-1)i} G(\omega^i),$$

since ω generates all of $\mathbb{F}^*$, and thus the Discrete Fourier Transform matrix (a bona fide Vandermonde matrix)

$$\begin{pmatrix} 1 & \omega^0 & \cdots & \omega^{0\cdot(r-1)} \\ 1 & \omega^1 & \cdots & \omega^{1\cdot(r-1)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \omega^{r-1} & \cdots & \omega^{(r-1)\cdot(r-1)} \end{pmatrix}$$

is invertible. Note that computing any $G(\omega^i)$ is in TC^0 using the product from of G (we know how to add and multiply $n^{O(1)}$ elements of $\mathbb{F}$ in TC^0). Recovering σ_j's is thus easily seen to be in (logspace uniform) TC^0.

Clearly any $\sigma_j(y_1, \ldots, y_s)$ equals $\sigma_j(y_1, \ldots, y_s, 0, \ldots, 0)$ for any number of extra zeros. Thus we can let $(y_1, \ldots, y_{k-1})$ cycle through $(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k)$ for $1 \leq i \leq k$, and set $y_k, \ldots, y_r$ to 0. This completes the reconstruction of the original polynomial.

□

Let us return to our gates g and h with associated polynomials $P_a(u)$ and $P_b(v)$ (whose coefficients we now know). We will create a new gate f with inputs g and h. The function computed by f will be $F(u, v) = kP_a(u) + P_b(v)$. Thus $F(a_i, b_j) = k(i-1) + (j-1)$ for every $i, j \in [k]$. (Recall that we always implicitly have the multiplicative identity of $\mathbb{F}$ as a factor.) If we run the enumerator on $\langle f, x \rangle$, we will get back k candidates for $f(x)$, each corresponding to a *unique* combination of claimed values for $g(x)$ and $h(x)$, provided that $k^2 \leq p$. In particular, if the enumerator claims that $f(x) = c$, then c must be in the form $k(i-1) + (j-1)$ for some $i, j \in [k]$, yielding the claim that $g(x) = a_i$ and $h(x) = b_j$. Otherwise, $f(x) \neq c$.

Recall that we want to substantially reduce the total number of gate assignments to the original gates of C, and we do it as follows: We group the gates of C in pairs (arbitrarily) constructing a new level of gates as described above. We repeat the same procedure for the new level; continuing this way, eventually we get to a single gate (after having constructed a full binary tree on top of the original gates). We run the enumerator on this gate, obtaining k candidates for its value on input x. Each candidate value corresponds, via a downward chain of decodings, to a unique gate assignment for the original gates of C. As long as k is polynomial in n, we can verify all candidate assignments in parallel, singling out the correct one. The value of $C(x)$ is the value assigned to the output gate.

We now describe the structure of the circuit that does this. Our parallel algorithm will first run the enumerator on every gate g_i in C. Each enumerator gate takes a description of C with a marked output gate g_i, together with the

original input x, and outputs a list of k numbers, one of which is $g_i(x)$. Let us return to our pair of gates g and h with lists $(a_1, \ldots, a_k)$ and $(b_1, \ldots, b_k)$. All other pairs are treated in the same way in parallel. We feed the lists and the original input x to auxiliary circuitry $\textsc{Aux}(g, h)$ that outputs a description of a new circuit with the output gate f. (The new circuit contains the original C plus whatever circuitry has already been built on top of it. The input to the new circuit will be the original input x.) $\textsc{Aux}(g, h)$ reconstructs the polynomials P_a, P_b, and uses them to compute a description of f. It follows from the above discussion that $\textsc{Aux}(g, h)$ can be implemented in NC^0 if k is constant; and in TC^0 if p^q (and thus k) are polynomial in n. (Given P_a and P_b, the description of f can clearly be computed by NC^0 and TC^0 circuits respectively.) Since there are $O(\log n)$ levels, the entire construction is in NC^1 when $\mathbb{F}$ is fixed, and in TC^1 if $\mathbb{F}$ is specified in the input as discussed above. All circuits are logspace uniform. Note that the output of an NC^0 circuit can depend only on constantly many inputs, thus improving the complexity significantly in the case of non-constant k appears to be impossible. Once we get to a single gate, we run the enumerator on this gate, obtaining k candidates for its value on x. Each value can be uniquely passed back to the original gates of C. Passing amounts to finding i and j from the value of $k(i-1) + (j-1)$, which can obviously be done in NC^0 or TC^0, depending on whether k is constant. ∎

3.4 Main Theorem

We now prove the main result for arbitrary finite fields. Although the proof echoes the proof of Sivakumar [Siv99], we present a self-contained treatment.

Theorem 2 Let $\mathbb{F}_q$ be the finite field with $q = 2^r$ elements, and let $i \geq 3$ be an integer. If $\mathrm{CVP}_{\mathbb{F}_q}$ is $(q-1)$-enumerable in NC^i then $\mathrm{CVP} \in \mathrm{NC}^i$.

Remark. The choice of $\mathbb{F}_q$ above is made to avoid inessential complications with the presentation (in particular, to simplify the Boolean complexity of field operations). The result readily holds for any finite field (modulo these complications). We will also see that the theorem holds for fields that are not fixed but are given as a part of the input.

Proof: Let $\langle C, x\rangle$ be the instance of the (Boolean) circuit-value problem whose solution we seek, and let $g_0, \ldots, g_{n-1}$ denote the gate values of C on input x. We view the gate values of C on x as the degree $(n-1)$ univariate polynomial $\mathrm{P}_g(u) = \sum_{i=0}^{n-1} g_i u^i$ over GF(2). We will try to evaluate this polynomial on all elements of an appropriately chosen extension field GF(2^m) of GF(2). In fact, let us fix m now to be the smallest integer of the form $2 \cdot 3^l$ (for some positive integer l) such that $m = d \log n$. If the field $\mathbb{F}_q$ is fixed, we can take any $d \geq 2$. If $\mathbb{F}_q$ is not fixed but is given by q in unary, d will be bounded by a constant dependent on c (fixed later in the proof), where c is such that $r = c \log n$. In either case, m will be easily computable in logspace.

A note on the representation: If m is in the above form,[2] the polynomial $y^m + y^{m/2} + 1$ is irreducible over GF(2). (See, for example, [vL91].) Thus GF(2^m) can be constructed as the ring of polynomials over $\mathbb{Z}_2$ modulo $y^m + y^{m/2} + 1$. Each polynomial in GF(2^m) can also be represented as an m-bit string, treating bits as the corresponding coefficients (with the most significant bit encoding the constant term of the polynomial). The zero element and the multiplicative identity of the ground field are treated as the Boolean values 0 and 1, respectively. This will be an implicit assumption hereafter.

Recall that our plan is to obtain some nontrivial information about the value

[2]Note that m can be in any form. Having the power of NC^3, we can find an irreducible polynomial exhaustively by cycling through monic polynomials of degree m in $\mathbb{Z}_2[x]$ until an irreducible is found. In what follows, we assume that the representation of the extension field has been fixed.

of P_g on all elements of $\mathrm{GF}(2^m)$, and that we have a $(q-1)$-enumerator for $\mathrm{CVP}_{\mathbb{F}_q}$ at our disposal. We will use the enumerator to reduce the number of candidates for each evaluation, and then rely on the technique of Ar et al. [ALRS98] to reconstruct the coefficients of P_g, which are the gate values of C on x that we seek.

So let us fix $u \in \mathrm{GF}(2^m)$, and use the enumerator to reduce the set of possibilities for $P_g(u)$, which is an element of $\mathrm{GF}(2^m)$, and thus has an m-bit representation (and consequently 2^m possible settings of these m bits). Consider an r-element subset S of $\{0, \ldots, m-1\}$. We will try to find the r bits of $P_g(u)$ specified by S. Or rather, we will rely on the enumerator to eliminate one possible setting for these r bits by disguising this question as an instance of $\mathrm{CVP}_{\mathbb{F}_q}$ whose solution (viewed as an r-bit string) gives the desired r bits of $P_g(u)$. In the same way, we can rule out one possible setting for every r-element subset of the m bits of $P_g(u)$. We will then argue that this rules out a large portion of all possible bit settings for all the m bits.

Let us first make a small detour. We will need a bit version of $\mathrm{CVP}_{\mathbb{F}_q}$, consisting of tuples $\langle D, y, i\rangle \in \Sigma^*$ such that D is an encoding of an arithmetic circuit, y is an input to D, both D and y are over $\mathbb{F}_q$, and the i-th bit of $D(y)$ is 1, when $D(y)$ is viewed as an r-bit string. It is readily seen that the bit version of $\mathrm{CVP}_{\mathbb{F}_q}$ is complete for P. Consider the set

$$A = \{\langle C, x, u, i\rangle \mid \text{the } i\text{-th bit of } P_g(u) \text{ is } 1\},$$

where C, x, u, and $P_g(u)$ are defined as above. Clearly A is in P, since the required $\mathrm{GF}(2^m)$-arithmetic needed to compute any bit of $P_g(u)$ can be easily implemented in NC^1. Thus A many-one reduces (via some NC^1-computable function h) to the bit version of $\mathrm{CVP}_{\mathbb{F}_q}$. We will use h to map each bit of $P_g(u)$ to an instance of the bit version of $\mathrm{CVP}_{\mathbb{F}_q}$. Thus our parallel algorithm will first run h (in parallel) on all tuples $\langle C, x, u, i\rangle$ for i from 0 to $m-1$. Given $\langle C, x, u, i\rangle$, h produces an

encoding of some circuit D_i together with an input y_i, both over $\mathbb{F}_q$, and a number j_i such that the i-th bit of $P_g(u)$ is precisely the j_i-th bit of D_i on y_i. (End of detour.)

Let us fix a subset $S \subseteq \{0, \ldots, m-1\}$ of size r and turn to the task of eliminating a possible setting of these bits in $P_g(u)$. Denote the r-bit string corresponding to $P_g(u)$ restricted to the bits in S by $w = w_0 \ldots w_{r-1}$. Notice that the bits of w are essentially the bits j_i of $D_i(y_i)$, where i ranges over the bits specified by S. We will view these output bits (denoted $w_0, \ldots, w_{r-1}$ to avoid new notation) as specifying the coefficients of the polynomial $P_w(x) = \sum_{i=0}^{r-1} w_i x^i$ over GF(2). Here x is our original input in the Boolean instance of CVP that we are solving. We view $\mathbb{F}_q$ as $\mathbb{Z}_2[z]/(g)$, the ring of polynomials over $\mathbb{Z}_2$ modulo the irreducible polynomial g of degree q in $\mathbb{Z}_2[z]$. We can hunt for g exhaustively by cycling through monic polynomials of degree q until an irreducible is found. Irreducibility can be tested in parallel time $O(\log^3 q)$. We assume that the representation of $\mathbb{F}_q$ has been fixed. Thus P_w is an element of $\mathbb{F}_q$. We will construct an arithmetic circuit D_S with an input y_S (both over $\mathbb{F}_q$) such that the value of D_S on y_S, viewed as a polynomial in $\mathbb{Z}_2[z]/(g)$, is equal to P_w. Since the degree of P_w is $r-1$, $D_S(y_S)$ uniquely defines P_w. In fact, we already have such D_S and y_S at hand. Our y_S will be the collection of inputs $\{y_i \mid i \in S\}$, and D_S will contain the circuitry $\{D_i \mid i \in S\}$. Each $D_i(y_i)$ (which is an r-bit number) encodes the corresponding coefficient of P_w as its j_i-th bit. The circuit D_S will output these r bits as an r-bit representation of its output (which is an element of $\mathbb{F}_q$). The circuitry needed to do such compilation of outputs is certainly in logspace uniform NC^1. We can finally run the $(q-1)$-enumerator for $\mathrm{CVP}_{\mathbb{F}_q}$ on $\langle D_S, y_S \rangle$ to get a list of $q-1$ values in $\mathbb{F}_q$. We can similarly eliminate one possible setting of any other subset of r bits of $P_g(u)$. Not only can we do it for all sets S in parallel (u has been fixed up until now), but also for all $u \in \mathrm{GF}(2^m)$. Recall that $r = c \log n$, and $m = d \log n$ with $d > c$ to be fixed later. There are 2^m many u's, and $\binom{m}{r} < 2^m = n^{O(1)}$ many

subsets for each such u. (Needless to say that if $\mathbb{F}_q$ is fixed, the number of subsets is polynomial in n.)

How many possible bit settings for each $P_g(u)$ does it rule out? This number follows immediately from (a rather special case of) a lemma proved independently by Sauer [Sau72], Perles and Shelah [She72], and Vapnik and Chervonenkis [VC71]. (For an excellent exposition, including the lemma's proof, see Bollobás [Bol86, p.131].)

To state the lemma, we shall say that a set $I \subset [m]$ is *traced* by a set system $\mathcal{F} \subset \mathcal{P}(m)$ if the collection of intersections $\{F \cap I \mid F \in \mathcal{F}\}$ contains every element of $\mathcal{P}(I)$. The lemma states that any $\mathcal{F}$ traces at least $|\mathcal{F}|$ sets in $\mathcal{P}(m)$. In particular, this says that if $|\mathcal{F}| > \sum_{i=0}^{r-1} \binom{m}{i}$, then $\mathcal{F}$ must trace some set of size r. For us, $\mathcal{F}$ is the set of remaining possible assignments to the m bits of $P_g(u)$. We know that it does not trace any subset of r bits (since we excluded at least one possible intersection for each such subset), thus the contrapositive of the lemma says that we are left with at most $\sum_{i=0}^{r-1} \binom{m}{i} < r\binom{m}{r}$ possibilities for $P_g(u)$ (out of 2^m such possibilities). Likewise for every other $u \in \mathrm{GF}(2^m)$.

Let $a = d/c$. We use a bound on $\binom{m}{r} = \binom{m}{m/a}$ in the form: $\binom{m}{m/a} < C(a) \cdot (ea)^{n/a}$, where $C(a)$ is some constant that depends on a. A number of different estimates of this form are known; which one is used does not affect the result. We want $k \stackrel{\text{def}}{=} 2^{m/4} > C(a) \cdot (ea)^{n/a}$, so given c, we take $d = ac > 4$ to be any easily computable constant such that a satisfies the condition above.

It remains to reconstruct the coefficients of P_g using the restricted lists of possibilities for its value on a set of 2^m points. We will use the powerful technique of Ar et al. [ALRS98], which reduces the reconstruction to the factorization of an appropriately constructed bivariate polynomial $Q(u, v)$ which is zero at all points (u, v) such that $u \in \mathbb{F}_q$ and v is in the list of possibilities for $P_g(u)$. The key claim is that $(v - P_g(u))$ must appear as an irreducible factor of Q. The degree of Q will be polynomial in n in both u and v. The factorization thus gives a list of

polynomially many candidates for P_g, each corresponding to a gate assignment for C on x. All of them can be verified in parallel in NC^1, and the correct one can be singled out.

The statements regarding Q need some justification. For completeness (and since we need to bound the parallel complexity rather than sequential time), we sketch the arguments of Ar et al. [ALRS98]. (See also [Siv99].)

Reconstruction Let $\mathbb{F} = \mathrm{GF}(2^m)$ be the extension field. We will construct a bivariate non-zero polynomial $Q(u, v)$ of degree $\deg(u)$ in u and $\deg(v)$ in v (to be fixed momentarily), such that $Q(u, v) = 0$ for all $u \in \mathbb{F}$ and all v in the list of k possibilities for $P_g(u)$, and such that both u and v are present in every monomial of Q. This gives a homogeneous system of $k2^m$ linear equations in $\deg(u) \cdot \deg(v)$ unknowns. If $\deg(u) \cdot \deg(v) \geq k2^m$, we are guaranteed a solution. It remains to show that $v - P_g(u)$ will appear as an irreducible factor. Consider the univariate polynomial $Q_g(u) = Q(u, P_g(u))$, which is of degree at most $\deg(u) + n\deg(v)$ in u. We know that $Q_g(u)$ is 0 on every $u \in \mathrm{GF}(2^m)$. Thus if $\deg(u) + n\deg(v) < 2^m$, $Q_g(u)$ must be identically zero, implying that $(v - P_g(u))$ divides $Q(u, v)$. Let us choose $\deg(v)$ and $\deg(v)$ leading to this desired situation (for sufficiently large n). Recall that $k = m2^{m/4}$. We have to satisfy $\deg(u)\deg(v) \geq m2^{5m/4}$ and $\deg(u) + 2^{\frac{m}{d}}\deg(v) < 2^m$, which can be done by setting, for example, $\deg(u) = \deg(v) = 2^{\frac{2m}{3}}$.

Note crucially that $\mathbb{F}$ has polynomially in n many elements, and thus its characteristic p is small, i.e., we need the algorithms to run in depth polynomial in $\log p$ (rather than $\log\log p$). (In our particularly pleasant case $p = 2$.) In what follows, n (the length of the main input) determines the complexity and uniformity of circuits. All circuits are logspace uniform. Univariate polynomials of degree $n^{O(1)}$ over $\mathbb{F}$ (not necessarily square-free) can be factored in NC^3. A deterministic version of the univariate factoring algorithm of von zur Gathen [vzG84],

obtained by plugging in the deterministic NC^2 matrix rank algorithm of Mulmuley [Mul87], works in depth $O(\log^2 n \log(np))$ (bounded fan-in, polynomial time). Thus if p is polynomial in n, we are in NC^3 (as noted in [vzGS91]). Bivariate polynomial factoring can be reduced to univariate factoring using Hensel's lifting. Let $f(x, y) \in \mathbb{F}[x, y]$ be the polynomial that we wish to factor into irreducible factors. If $\mathbb{F}$ is a unique factorization domain (which it is, since $\mathbb{F}$ is a field), then so are $\mathbb{F}[x]$ and $\mathbb{F}[x, y]$, thus the factorization is unique (up to the order of factors). We will only seek some irreducible factor of f, hence we will then be able to compute the complete factorization in parallel for all factors.

Let the degree of f be at most $d = n^{O(1)}$ in each variable. We seek an irreducible factor $g \in \mathbb{F}[x, y]$ of f. Hensel's lifting extracts information about the factors of $f(x, y)$ by lifting up a univariate factorization of $f_0(x) = f(x, 0)$. Notice that factoring the univariate polynomial $f_0(x) = g(x, 0)h(x, 0)$ is equivalent to factoring $f(x, y) = g(x, y)h(x, y) \bmod y$. Hensel's construction will lift this factorization to the one modulo $y^2, y^{2\cdot 2}$, and so on, until we have a factorization modulo y^{2^t}, where t is sufficiently large to yield the true factorization of $f(x, y)$. We will go through the steps of the reduction only for the purpose of establishing the parallel complexity, i.e., without giving any intuitive of formal justifications. For this the reader is referred to, for example, a book by von zur Gathen and Gerhard [vzGG99].

Step 1. Factor $f_0(x) = g_0(x)h_0(x)$, where g_0 and h_0 are relatively prime and g_0 is irreducible. (This is possible if f_0 is squarefree. If f_0 is not squarefree, then its squarefree decomposition can be computed first, and then we can work with all square-free factors in parallel.) The whole factorization is in NC^3 using the univariate factorization algorithm mentioned above.

Step 2. Iteratively lift the factorization $t = 2(\lceil \log d \rceil + 1)$ times to obtain g_t, h_t such that $f(x, y) \equiv g_t(x, y)h_t(x, y) \pmod{y^{2^t}}$. Each lifting step allows us

to (uniquely) lift factorization modulo y^e to factorization modulo y^{2e}. The cost of lifting is essentially the cost of computing the quotient and remainder of polynomials of degree at most d in $\mathbb{F}[x]$, and is in NC^1 [Ebe89], putting $t = O(\log n)$ iterations in NC^2.

Step 3. Find polynomials $G(x, y)$ and $D(x, y)$ satisfying the relations

$$G(x, y) \equiv g_t(x, y)D(x, y) \pmod{y^{2^t}},$$

$\deg_x G < \deg_x f$, $\deg_x D < \deg_x f$, $\deg_y G \leq \deg_y f$, $\deg_y D \leq 2^t$, $G \not\equiv 0$. Thus Step 3 reduces to solving a system of linear equations, with unknowns being the coefficients of $G(x, y)$ and $D(x, y)$. Solving a system of linear equations is in NC^2 [BvzGH82].

Step 4. Find $\gcd(G, f)$ and if nontrivial, output G as a factor of f. If trivial, f is irreducible. The gcd computation is in NC^2 [BvzGH82]. (As warned, we will accept it as a fact that the solution to the system above exists and that the existence of G indeed yields a nontrivial factor of f.)

From the discussion above, the factorization is readily seen to be in (logspace uniform) NC^3. The computation of all factors (originating from different ways to factorize $f_0(x)$ as the product of a monic irreducible $g_0(x)$ and a relatively prime to it $h_0(x)$) can be done in parallel. ∎

Corollary 1 Let $\mathbb{F}_q$ be the finite field with q elements, and let $i \geq 3$ be an integer. If $\mathrm{CVP}_{\mathbb{F}_q}$ is $(q-1)$-enumerable in NC^i then $\mathrm{P} = \mathrm{NC}^i$. The field $\mathbb{F}_q$ can be given by q (in unary notation) as a part of the input.

3.5 Probabilistic Enumerators

Until now enumerators were not allowed to err. A natural question is what happens if they were. Before debating about whether such enumerators are conceptually fit, let us see whether they are technically interesting. First, it is not even clear how erroneous enumerators should be defined. There are two natural choices. Either the enumerator fails to include the correct answer on a small fraction of inputs at each input length; or it distributes the error by guaranteeing to include the correct answer for every input with high probability.

In this section, we consider enumerators that are allowed to err with small probability, and prove an analogue of Theorem 2 in the randomized setting. For the definition of randomized Boolean circuits and the class RNC, the reader is referred to Cook [Coo85]. It is believed highly unlikely that P $\subseteq$ RNC. (Note, however, that randomized cicuits inherently have multiple access to randomness, and RNC^1 is not known (or believed) to be contained in randomized logspace.)

Let $\epsilon : \mathbb{N} \to (0, \frac{1}{2})$. An enumerator E_f for a function f is said to have error ϵ, if for all inputs x in the domain of f, the set $E_f(x)$ includes $f(x)$ with probability at least $1 - \epsilon(|x|)$, where the probability is taken over the random bits fed to E_f.

Theorem 3 (Randomized analogue of Theorem 2) Let $\mathbb{F}_q$ be as in Theorem 2. There is no $(q-1)$-enumerator for $\mathrm{CVP}_{\mathbb{F}_q}$ with error at most inversely polynomial unless P $\subseteq \mathrm{RNC}^3$.

Proof: As in the deterministic case, we will attempt to obtain restricted lists of candidates for the value of the gate polynomial P_g on the elements of the extension field $\mathrm{GF}(2^m)$. Let $\epsilon(n) = n^{-\Omega(1)}$ be the error of the enumerator, and fix $u \in \mathrm{GF}(2^m)$. What is the probability that the list of candidates for $P_g(u)$ excludes $P_g(u)$? No more than the probability that the enumerator errs on at least one of all r-element subsets of the m bits of $P_g(u)$, and this is at most $\binom{m}{r}\epsilon(s(n))$, where

$s(n)$ is the polynomial bounding the encoding length of $\mathrm{CVP}_{\mathbb{F}_q}$ instances that the subsets were mapped to by the reduction function in the proof of Theorem 2. We obviously want $\epsilon(s(n)) \leq (1/2 - \delta) \cdot \binom{m}{r}^{-1} = n^{-\Omega(1)}$ for some $\delta > 0$ (δ can be as small as inverse polynomial in n). This can be satisfied by appropriately altering m by a constant factor (which we have control over, and which does not affect the proof in any significant way). We can then make the probability that the list excludes $P_g(u)$ vanish exponentially in the input length at the price of a polynomial increase in parallelism. Indeed, we can run polynomially many instances of the enumerator in parallel and exclude the majority vote. Chernoff bounds can be used in the straightforward way to bound the probability that $P_g(u)$ is excluded (by an inverse exponential). Hence for every $u \in \mathrm{GF}(2^m)$, we have a list of at most k values (in the notation of Theorem 2), one of which is $P_g(u)$ with probability at least $1-\gamma$ for some inverse exponential γ, implying that the expected number of bad lists is at most $\gamma 2^m$. By Markov's inequality, the probability that there are more than $3 \cdot \gamma 2^m$ such lists is at most $1/3$. Assume from now on that there are in fact no more than $3 \cdot \gamma 2^m$ bad lists. As before, we will attempt to find a polynomial $Q(u, v)$ which is zero on all $u \in \mathrm{GF}(2^m)$ and all v in the list of possibilities for $P_g(u)$. The only difference is that in order to guarantee that $v - P_g(u)$ is an irreducible factor, we need to satisfy the constraint $\deg(u) + 2^{\frac{m}{d}}\deg(v) < 2^m(1 - 3\gamma)$. This would yield that $Q(u, P_g(u))$ has more zeros than its degree, and thus must be identially zero, implying that $v - P_g(u)$ divides $Q(u, v)$. Recall that $k = m2^{m/4}$. We have to satisfy the constraints $\deg(u)\deg(v) \geq m2^{5m/4}$ and $\deg(u) + 2^{\frac{m}{d}}\deg(v) < 2^m(1 - 3\gamma)$, and our choice $\deg(u) = \deg(v) = 2^{\frac{2m}{3}}$ does so (for sufficiently large n). Thus finding Q and factoring it, gives a small list of polynomials that includes P_g. We know how to single out P_g. This completes the proof. ∎

4 The (Non)Enumerability of the Determinant and the Rank

In this chapter we show the hardness of enumerating two fundamental problems in linear algebra, computing the rank and the determinant of a matrix.

4.1 History and Motivation

Valiant [Val79b] proved that the permanent of integer matrices characterizes #P, the class of functions definable as the number of accepting computations of a nondeterministic polynomial-time Turing machine. A similar connection has been shown between the complexity of computing the determinant and #L, the logspace analog of #P [Tod91, Val92, Vin91, Dam91]. (Since the determinant of integer matrices can take on negative values, the determinant is in fact complete for GapL, the class of functions that can be expressed as the difference of two #L functions.) Toda's result [Tod89] showing the surprising power of counting in the context of nondeterministic polynomial-time, namely that the polynomial hierarchy is contained in $P^{\#P}$, naturally raises the question of whether #P functions are at least easy to approximate. In the standard sense of coming close to the correct

value (i.e., within a multiplicative factor), approximating #P functions is in Δ_3^p. Moreover, any technique for showing that it has complexity lower than Δ_2^p, would have to be non-relativizable. (See [Sto85].) In search of a better answer, Cai and Hemachandra [CH89] proposed an alternative notion of approximation, which they called *enumerative counting*. Instead of restricting the range of a function value to an interval, they consider enumerating a short list of (not necessarily consecutive) values, guaranteed to contain the correct one.

Recall that a function f is said to be $g(n)$-*enumerable* if there exists a function that, on input x, outputs a list of at most $g(|x|)$ values guaranteed to contain $f(x)$. Cai and Hemachandra [CH91] (see also Amir, Beigel, and Gasarch [ABG90]), showed that if the permanent function is poly-enumerable in polynomial time, then $\mathrm{P} = \mathrm{P}^{\#\mathrm{P}}$. This is certainly a *non*-enumerability result, as it says that a function hard for the entire polynomial hierarchy can easily bootstrap its exact value if it is left with polynomially many candidates.

We investigate the enumerability of functions complete for logspace counting classes. It is interesting whether enumerability implies a similar decrease in the complexity of the determinant; and if so, to what extent. Since #L functions have significantly less computational power (they are contained in TC^1, a subclass of NC^2), the enumerability properties of logspace counting analogs can be very different from those of #P-complete functions. This is all the more interesting, because, as far as we know, there are no results on (the hardness of) approximating matrix determinant or rank in the standard sense. Another purpose of such an investigation is to get a better understanding of the relationships among the complexity classes sandwiched between NL and uniform TC^1.

We show that if the determinant of integer matrices is poly-enumerable in logspace, then it can be computed exactly in logspace. In other words, it is as hard to poly-enumerate as to compute exactly. The result holds for matrices over finite fields of characteristic greater than the dimension of the matrix. A natural

question is whether this can be extended to arbitrary finite fields. We establish a similar result for computing the determinant modulo any prime.

We also show that constant enumerability of the rank function over finite fields implies that the rank can be computed exactly in AC^0 (modulo the complexity of the enumerator). Thus, for example, if the enumerator runs in logspace, then the problem of computing the rank is also in logspace. The result actually holds for finite commutative rings with identity.

4.2 Preliminaries

We will be concerned with the complexity of computing the following functions (with input and output in $\{0,1\}^*$):

- $\textsc{Det}_F$: Given $A \in F^{n\times n}$, compute the determinant of A.
- $\textsc{Rank}_F$: Given $A \in F^{n\times n}$, compute the rank of A.
- $\textsc{Singularity}_F$: Given $A \in F^{n\times n}$, determine whether A is singular.
- $\textsc{Independence}_F$: Given a set of vectors in F^n, determine whether they are linearly independent.

Here F is any commutative ring with identity. When $F = \mathbb{Z}$, the ring of integers, F is dropped. A usual way to represent an element of an algebraic structure F in a Boolean circuit is as an integer written in binary. Since we will be mostly interested in matrices over finite rings, or over rings whose size is polynomially related to the size of the matrix, the elements of F will be "short" (i.e., have logarithmic bit length). Thus arithmetic operations over F can be easily computed in AC^0.

Logspace Counting Classes and the Complexity of Problems in Linear Algebra

Many basic linear algebraic problems are known to be in NC^2. In order to classify and capture the exact complexity of these problems, Cook [Coo85] defined the class of problems NC^1-reducible to the determinant of integer matrices, and showed that most linear algebra problems with fast parallel algorithms are in this class. Many are in fact complete for this class; others were shown to be complete for the (potentially smaller) class of problems reducible to computing the rank [vzG93, ST98]. Santha and Tan [ST98] explored the computational difference between the functional and the verification versions of the problems under AC^0-Turing and AC^0-many-one reductions. Allender and Ogihara [AO96] observed that, even though for most natural problems the closures under AC^0 and NC^1-reductions coincide, this does not seem to be apparent for the determinant. This (among other things) motivated the definition of the following hierarchies, defined using the "Ruzzo-Simon-Tompa" oracle access model [RST84], which is standard for defining Turing reductions for space-bounded nondeterministic machines, see [AO96]:

- The exact counting logspace hierarchy $\mathrm{C_=L} \cup \mathrm{C_=L}^{\mathrm{C_=L}} \cup \mathrm{C_=L}^{\mathrm{C_=L}^{\mathrm{C_=L}}} \cdots = \mathrm{AC}^0(\mathrm{C_=L})$

  The class $\mathrm{C_=L}$ is defined as the class of languages, for which there exists a GapL function f such that for every x, x is in the language if and only if $f(x) = 0$. If follows immediately that the set of singular matrices is complete for $\mathrm{C_=L}$. Allender, Beals, and Ogihara [ABO99] showed that AC^0 and NC^1 reducibilities coincide on $\mathrm{C_=L}$; furthermore, the hierarchy collapses to $\mathrm{L}^{\mathrm{C_=L}}$. We show that $O(1)$-enumerability of Rank implies a slightly deeper collapse (to the closure of $\mathrm{C_=L}$ under appropriate reducibility).

- The PL hierarchy $\mathrm{PL} \cup \mathrm{PL}^{\mathrm{PL}} \cup \mathrm{PL}^{\mathrm{PL}^{\mathrm{PL}}} \cup \cdots = \mathrm{AC}^0(\mathrm{PL})$

  Ogihara [Ogi98] showed that the PL hierarchy collapses to PL under AC^0 (in fact TC^0) reductions, which was improved to NC^1 reducibility by Beigel and Fu [BF97]. A problem, easily seen to be complete for PL, is checking whether the determinant of integer matrices is positive.

- The #L hierarchy $\mathrm{L} \cup \mathrm{L}^{\#\mathrm{L}} \cup \mathrm{L}^{\#\mathrm{L}^{\#\mathrm{L}}} \cdots = \mathrm{AC}^0(\#\mathrm{L}) = \mathrm{AC}^0(\textsc{Det})$

  It is not known whether the #L hierarchy collapses, or whether $\mathrm{AC}^0(\#\mathrm{L}) = \mathrm{NC}^1(\#\mathrm{L})$. The latter would imply the collapse [All97a].

Allender, Beals, and Ogihara [ABO99] showed that the problems of computing the rank of integer matrices, determining whether the rank is odd, and determining the solvability of a system of linear equations, are all complete for $\mathrm{AC}^0(\mathrm{C}_=\mathrm{L})$. Clearly, the problems of computing and verifying the rank of a matrix are AC^0-equivalent (since there are just $n+1$ possibilities for the rank). However, Allender, Beals, and Ogihara classified the complexity of verifying the rank exactly, showing that it is complete for the second level of the Boolean Hierarchy over $\mathrm{C}_=\mathrm{L}$ (i.e. the class of sets expressible as an intersection of a $\mathrm{C}_=\mathrm{L}$ and a co-$\mathrm{C}_=\mathrm{L}$ set).

Our results We show that constant enumerability of the rank function over finite fields implies that the rank can be computed exactly in AC^0 (modulo the complexity of the enumerator). Thus, if there exists an enumerator that, given a matrix, outputs a list of constantly many numbers, including the rank of this matrix, it can be determined in AC^0 (with oracle gates for the enumerator) which of these numbers is the rank. For example, if the enumerator runs in logspace, then the problem of computing the rank is also in logspace. We also consider a

related problem of counting the number of vectors in a given set that are spanned by the other vectors in the set, and show it to be $\leq^{\mathrm{TC}^0}_{O(n)\text{-}tt}$-equivalent to computing the rank. Both results actually hold for commutative rings with identity with the condition that the size of the ring that the input matrix is over is polynomially related to the dimension of the matrix. More formally, we consider a sequence of rings $R_1, R_2, \ldots, R_n, \ldots$, where the size of each R_n is polynomial in n, and our $n \times n$ matrices are over R_n.

The techniques used in the proof give, from the assumption of logspace constant-enumerability of the rank, $\leq^{\log}_{O(n)\text{-}tt}$ -reductions from $\textsc{Rank}_F$ to $\textsc{Independence}_F$ and to $\textsc{Singularity}_F$. The reductions hold for arbitrary rings; in particular, for the ring of integers. Since for integer matrices, Independence and Singularity are easily seen to be complete for, respectively, co-$\mathrm{C}_=\mathrm{L}$ and $\mathrm{C}_=\mathrm{L}$, the existence of the enumerator implies that $\mathrm{AC}^0(\mathrm{C}_=\mathrm{L})$ (and thus the entire exact counting logspace hierarchy) coincides with the intersection of $\leq^{\log}_{O(n)\text{-}tt}$ -closures of $\mathrm{C}_=\mathrm{L}$ and co-$\mathrm{C}_=\mathrm{L}$, a slight improvement over a $O(\mathrm{poly}(n))$-truth-table reduction that follows from [ABO99] (unconditionally). In fact, constantly many queries suffice for the reduction to $\textsc{Independence}_F$.

For the determinant function Det (over the integers), we prove that if Det is poly-enumerable in logspace, then Det is in logspace. In other words, it is as hard to enumerate as to compute exactly. Similarly, for any prime p, if the determinant-modulo-p function is $(p-1)$-enumerable in logspace, then it can be computed exactly in logspace. Thus, if we could eliminate at least *one* of p possibilities for the determinant-modulo-p in logspace, we would be able to compute the determinant-modulo-p exactly in logspace. It is naturally interesting whether a similar result can be shown for finite fields.

4.3 Enumerability of the Rank

4.3.1 Computing the Rank

Recall that a function f is logspace $g(n)$-enumerable if there exists a logspace computable function that, on input x, outputs a list of at most $g(|x|)$ values, one of which is $f(x)$. The following lemma shows how to combine several matrices into a single matrix such that the ranks of the original matrices can be read off the rank of the combined matrix.

Lemma 1 [Block diagonal construction] There exists a logspace computable function S that given an ordered list $\mathcal{Q} = \langle A_1, \ldots, A_q \rangle$ of $n \times n$ matrices, outputs a single matrix $S(\mathcal{Q})$ of dimension $O(n^q)$ such that a logspace procedure can uniquely decode the sequence of ranks $\langle \mathrm{rank}(A_1), \ldots, \mathrm{rank}(A_q) \rangle$ from the value of $\mathrm{rank}(S(\mathcal{Q}))$. Moreover, both procedures can be implemented by uniform AC^0 circuit families.

Proof: Consider the following construction. (See Figure 4.1.) On input $\mathcal{Q} = \langle A_1, \ldots, A_q \rangle$, S outputs a block diagonal matrix (i.e. a matrix of $n \times n$ blocks sitting on the main diagonal) with the following block structure. The first block of $S(\mathcal{Q})$ corresponds to A_1, the next $(n+1)$ blocks correspond to A_2, and so on, until we get to $n^{q-1} + n^{q-2} + \cdots + 1$ blocks of A_q. The multiplicity of A_i as a block is $\sum_{j=1}^{i} n^{j-1}$, thereby the dimension of $S(\mathcal{Q})$ is $\sum_{i=1}^{q} n^i(q-i+1) = O(n^q)$. The rank of $S(\mathcal{Q})$ is the sum of all block ranks, and since the rank of each block is at most n, the original sequence of ranks $\langle \mathrm{rank}(A_1), \ldots, \mathrm{rank}(A_q) \rangle$ can be read off from the value of $\mathrm{rank}(S(\mathcal{Q}))$. It is easy to see that both the construction and the decoding can be done in uniform AC^0. ∎

The block diagonal construction above allows one to eliminate many candidate rank sequences. For example, if we were to feed each of q matrices to an r-

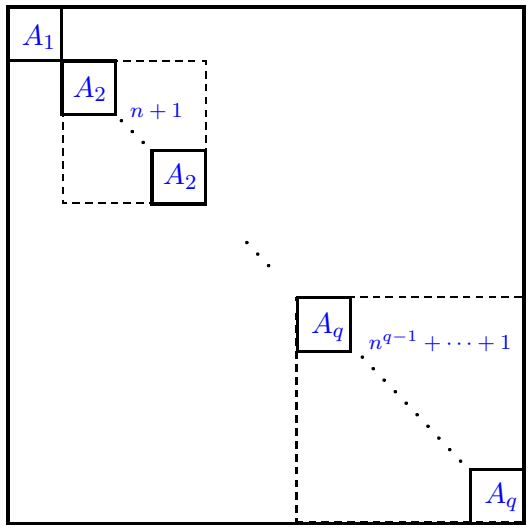

Figure 4.1: Block-diagonal construction

enumerator separately, we would get r^q purported rank sequences, whereas combining the matrices into a single query reduces the number of candidates to r. In order for the dimension of $S(\mathcal{Q})$ to be polynomial in n, the number of matrices, q, has to be constant. A simple information-theoretic argument shows that this is the best possible. Indeed, the dimension of a matrix whose rank can encode $(n+1)^q$ possible rank sequences must be at least $(n+1)^q$. Combining matrices into a single query to an r-enumerator also allows us to *link* matrices in the sense that will be explained shortly. First we need the notion of *linked sequences*.

Definition 8 Two r-element sequences $\{p_1, \ldots, p_r\}$ and $\{q_1, \ldots, q_r\}$ are said to be *linked* if $p_i = p_j$ if and only if $q_i = q_j$, for all $1 \leq i < j \leq r$.

In other words, two sequences are linked if and only if there is a one-to-one correspondence between the set of values in one and the set of values in the other. For example, the sequences $\{1, 2, 2, 3\}$ and $\{2, 3, 3, 4\}$ are linked, while $\{1, 2, 2, 3\}$ and $\{2, 3, 4, 4\}$ are not. The following claim gives an upper bound on the number of sequences guaranteed to contain at least one linked pair.

Claim 1 There exists r_0 such that for any $r > r_0$, any set of $(\frac{r}{\ln r})^r$ r-element sequences contains at least one linked pair.

Proof: The number of r-element sequences sufficient to guarantee the existence of a linked pair is precisely one more than the number of partitions of an r-element set into non-empty subsets. The latter is known as the rth Bell number, B_r. De Bruijn [dB70] gave the asymptotic formula

$$\frac{\ln B_r}{r} = \ln r - \ln\ln r - 1 + \frac{\ln\ln r}{\ln r} + \frac{1}{\ln r} + \frac{1}{2}\left(\frac{\ln\ln r}{\ln r}\right)^2 + O\left(\frac{\ln\ln r}{(\ln r)^2}\right),$$

immediately yielding the claim. Other (less explicit) asymptotic approximations for B_r are known (see, for example, [Lov93, Odl95]). ∎

Let $\kappa(r)$ be the minimum number of r-element sequences guaranteed to contain a linked pair. By Claim 1, $\kappa \stackrel{\text{def}}{=} \kappa(r) \leq (\frac{r}{\ln r})^r$. We will say that two matrices M_1 and M_2 are *linked*, relative to a fixed r-enumerator, if there exists a list of $\kappa - 2$ matrices $M_3, \ldots, M_\kappa$ satisfying the following condition: When $M_1, M_2, \ldots, M_\kappa$ are combined using the construction in Lemma 1, and the resulting matrix is fed to the enumerator, we get back a list of r values that (when decoded) correspond to κ sequences of r alleged ranks each, one sequence for each of $M_1, M_2, \ldots, M_\kappa$. The r-sequences corresponding to M_1 and M_2 must be linked in the sense of Definition 8. In other words, M_1 and M_2 are linked (as witnessed by $M_3, \ldots, M_\kappa$) if there is a direct correspondence between the values on their claimed sequences of ranks; hence knowing the true rank of one immediately gives the true rank of the other.

Given an $n \times n$ matrix A, whose rank we want to compute, let A_i (for $1 \leq i \leq n$) denote the $n \times n$ matrix with the first i rows of A and 0s elsewhere, so that $A_n = A$. An r-enumerator for the rank function defines the *equivalence graph* of A, a labeled

graph on $\{1, \ldots, n\}$ with the set of nodes corresponding to $\mathcal{A} = \{A_1, \cdots, A_n\}$, and an edge between nodes i and j if and only if there is a set of $\kappa - 2$ matrices in $\mathcal{A} - \{A_i, A_j\}$ witnessing that A_i and A_j are linked. (We can assume without loss of generality that the enumerator is deterministic, and that the combining encoding of queries in Lemma 1 is symmetric, i.e., produces the same supermatrix for any permutation of the input matrices.) Each edge is labeled by the equivalence, i.e., direct correspondence between the r claimed values for $\text{rank}(A_i)$ and the r claimed values for $\text{rank}(A_j)$, given by the lexicographically smallest κ-tuple linking A_i and A_j.

Notice that by definition every subset of κ nodes in the equivalence graph induces at least one edge. Hence the number of connected components in the equivalence graph is at most $\kappa - 1$. The following proposition shows that every pair of nodes in the same component is connected by a short path (where the length of a path is the number of edges it contains).

Proposition 1 *Any pair of nodes in the same connected component of the equivalence graph is connected by a path of length at most* $2\kappa - 3$.

Proof: Suppose that there exist nodes i and j such that the shortest path between i and j is of length at least $2\kappa - 2$. Let $v_1 = i, v_2, \ldots, v_{2\kappa-1}$ be the first $2\kappa - 1$ nodes on this path. Then either the nodes $\{v_{2t+1} \mid 0 \leq t < \kappa\}$ form an independent set of size κ contradicting the fact that every κ-tuple of nodes induces at least one edge, or there is an edge connecting a pair of nodes in this set, in which case the path from i to j can be shortened, contradicting the assumption that it is the shortest path. ∎

We will use equivalence graphs in the proof of the theorem below. Throughout the section, F denotes a finite commutative ring with identity.

Theorem 4 If, for some integer r, there exists an r-enumerator for $\textsc{Rank}_F$, then $\textsc{Rank}_F$ is computable in AC^0 with oracle gates for the enumerator.

Proof: Given an enumerator and an input matrix A, the equivalence graph on the prefix forms of A is uniquely defined. Recall that the number of equivalence classes (i.e. the number of connected components in the graph) is less then κ. Consider guessing a set of κ representative matrices, which we shall call a *kernel* of A. The fact that the actual number of components may be less than κ is not important, since we just need to find a constant number of matrices whose ranks uniquely determine the ranks of all other matrices. There are only $\binom{n}{\kappa}$ choices for the kernel, so we have no problem checking them all in parallel. We will follow a single kernel.

We can easily check whether every node in the graph is reachable from the kernel. Recall that we need only check paths of length at most $2\kappa - 3$. The existence of an edge (i, j) can be checked by an AC^0 sub-circuit with a single gate for the enumerator along any path; each path checks whether its subset of $\kappa - 2$ matrices in $\mathcal{A} - \{A_i, A_j\}$ witnesses the equivalence between A_i and A_j. If at least one of the matrices is unreachable, the kernel is incorrect, and can be weeded out. Assume without loss of generality that the kernel we follow is *good*, i.e., every node in the graph is reachable from at least one kernel node. Let $R_1, \ldots, R_\kappa \in \mathcal{A}$ be the kernel matrices, and $\tau_1, \ldots, \tau_\kappa$ be their corresponding ranks. Note that $\tau_1, \ldots, \tau_\kappa$ uniquely define the ranks of all matrices in $\mathcal{A}$. Of course, we do not know $\tau_1, \ldots, \tau_\kappa$. Instead, we will use the block diagonal construction in Lemma 1 to pack $R_1, \ldots, R_\kappa$ into a single super-matrix, which we can then feed to the enumerator to get a list of r sequences of ranks, one of which is $\langle \tau_1, \ldots, \tau_\kappa \rangle$. Denote the sequences by $\langle \tau_1^1, \ldots, \tau_\kappa^1 \rangle$, ..., $\langle \tau_1^r, \ldots, \tau_\kappa^r \rangle$.

Notice that, given a kernel, there may be a number of different paths of length at most $2\kappa - 3$ from any fixed $A_i \in \mathcal{A}$ to its closest kernel node R_j. (The closest R_j is itself not necessarily unique; the ties are broken arbitrarily.) Thus each of

$\tau_j^1, \ldots, \tau_j^r$ may correspond, via a different chain of links, to a number of different values claiming to be the rank of A_i. Furthermore, not every $\tau_j^1, \ldots, \tau_j^r$ *will* have a chain of links leading to any value claimed for A_i. Since the correspondences of values marking the links on a path from R_j to A_i were given by different calls to the enumerator, not every $\tau_j^1, \ldots, \tau_j^r$ can be traversed back to A_i, as a value in any one of the correspondences is not necessarily present in the correspondence marking the next link. The *correct* sequence $\langle \tau_1, \ldots, \tau_\kappa \rangle$, however, will always result in the correct sequence $\langle \mathrm{rank}(A_1), \ldots, \mathrm{rank}(A_n) \rangle$, i.e., regardless of the path chosen for every A_i, the path from τ_j to $\mathrm{rank}(A_i)$ will not be broken. For consistency, we will use the following convention for each A_i and each claimed value for the rank of R_j: In the case of multiple paths, we will always follow the path from R_j to A_i that yields (via a sequence of correspondences along the path) the smallest claimed value for the rank of A_i. In the case of broken links, we will map each unmatched value to n. Using these conventions, each $\langle \tau_1^i, \ldots, \tau_\kappa^i \rangle$ gives a rank sequence $\langle \upsilon_1^i, \ldots, \upsilon_n^i \rangle$ claimed to be $\langle \mathrm{rank}(A_1), \ldots, \mathrm{rank}(A_n) \rangle$. Incorrect sequences will be weeded out during verification.

Let $U_i = \{1 \leq j \leq n \mid \upsilon_j^i = \upsilon_{j-1}^i + 1\}$, where we define $\upsilon_0^i = 0$ for all $1 \leq i \leq r$; thus each U_i claims to index a maximal set of linearly independent rows of A. Of course, we need to check whether the set indexed by U_i is indeed maximal, i.e. whether every remaining row of A is a linear combination of the rows indexed by U_i. As there are only constantly many U_is, testing them in parallel causes no problem. The rank of A is given by the size of the smallest U_i that passes the maximality test. (This number can be found as the corresponding υ_n^i.)

Remark 1 Alternatively, we could arrive at sets $V_1, \ldots, V_r$ that supposedly index maximally independent sets of columns of A. The square submatrices indexed by $U_1 \times V_1$, $\cdots$, $U_r \times V_r$ all claim to be nonsingular. (We can obviously discard all candidate sequences with $|U_i| \neq |V_i|$.) Now instead of verifying the maximality claim, we can check, in parallel, which submatrices are in fact nonsingular, and

then take the maximum over all that pass the test.

Notice that the discussion above is valid for arbitrary matrices. Thus, if there exists a logspace computable $O(1)$-enumerator for $\textsc{Rank}_F$, then $\textsc{Rank}_F$ is $\leq^{\log}_{O(1)\text{-}tt}$-reducible to $\textsc{Independence}_F$ (and $\leq^{\log}_{O(n)\text{-}tt}$-reducible to $\textsc{Singularity}_F$). The reductions do not depend on the field that the matrices are over; in particular, they hold for integer matrices. We now show how to test the maximality of U_is for matrices whose entries are drawn from some finite field F; then we extend the test to finite commutative rings with identity. In fact, the results naturally generalize to the case when we have a sequence of rings $R_1, R_2, \ldots, R_n, \ldots$, and our $n \times n$ matrices are over R_n (i.e., the size of the ring that a matrix is over is polynomially related to the size of the matrix).

Testing maximality: Given row vectors $v_1, \ldots, v_q, w \in F^n$, verify that w is in the subspace spanned by $v_1, \ldots, v_q$.

Let $F = \{a_1, \cdots, a_m\}$ be the ground field. If $v_1, \ldots, v_q$ are linearly independent, w is spanned by $v_1, \ldots, v_q$, and $w \neq 0^n$ (where 0 is the null element of F), there must exist unique coefficients $c_1, \ldots, c_q \in F$ such that $c_1 v_1 + \cdots + c_q v_q + w = 0$. For each i and j, $1 \leq i \leq m$, $1 \leq j \leq q$, define the matrix

$$M_i^j = \begin{pmatrix} v_1 \\ \vdots \\ a_i v_j + w \\ \vdots \\ v_q \end{pmatrix}.$$

If the above conditions hold, then for each j, there is a unique i such that $\text{rank}(M_i^j) = q - 1$; namely, $\text{rank}(M_i^j) = q - 1$ iff $a_i = c_j$; otherwise $\text{rank}(M_i^j) = q$. We have at most nm such matrices, which is polynomial in n, provided that m is polynomial in n. We want to reduce the number of possibilities for the ranks of

these matrices to a constant, and we already know how to force the enumerator to do this for us: Recall that there is a constant $\kappa = \kappa(r)$ such that combining any κ matrices into a single query witnesses at least one equivalence relation between a pair of claimed lists of ranks. Furthermore, there are at most $\kappa - 1$ equivalence classes, and thus only polynomially (in nm) many choices of how to partition M_i^js into the equivalence classes. All choices can be verified in parallel, each one giving only a constant number of possible values for the ranks of M_i^js. For each candidate sequence of ranks we collect the coefficients c_is, assuming that this sequence is correct (i.e., that for each j there is a unique i such that $\text{rank}(M_i^j) = q - 1$, and $\text{rank}(M_l^j) = q$ for all $l \neq i$); then we verify (column-wise in parallel) that the equality $c_1 v_1 + \cdots + c_q v_q + w = 0$ holds. Notice that the maximality test can be run in parallel not only for all matrices $U_1, \cdots, U_r$, but also for *all* rows w claimed to be linear combinations of the rows of each matrix. The number of matrices that we are dealing with is less than rn^2m, so if m is polynomial in n, testing them in parallel causes no problem. It is easy to see that the entire computation can be done in Boolean AC^0 with oracle gates for the enumerator (with inputs and outputs in binary), since we are dealing with fields of polynomial size (and thus elements of logarithmic length).

To extend this method to commutative rings, we just have to test whether, for each $d \in F$, there exist coefficients $c_1, \ldots, c_q \in F$ satisfying $c_1 v_1 + \cdots + c_q v_q + dw = 0$, which can be done in parallel for each possible d. For some i there can be more than one value of j for which $\text{rank}(M_i^j) = q - 1$, but we have only to select the smallest such j. ∎

Corollary 2 If there exists an $O(1)$-enumerator for $\textsc{Rank}_F$ that runs in logspace, then $\textsc{Rank}_F$ is computable in logspace.

4.3.2 Computing the Number of Dependent Rows

Consider the following related problem: Given a set of row vectors $v_1, \ldots, v_n \in F^n$, determine how many of them are in the span of the other vectors. We will call such vectors *dependent*. Define the problem $\textsc{Dep}_F$: Given a matrix $A \in F^{n \times n}$, compute the number of dependent rows of A, written as $\mathrm{dep}(A)$. As before, F is a finite commutative ring with identity.

Proposition 2 If, for some integer r, there exists an r-enumerator for $\textsc{Dep}_F$, then $\textsc{Dep}_F$ is computable in TC^0 with oracle gates for the enumerator.

Proof: The argument is similar to the one used in the proof of Theorem 1. Namely, given a matrix $A \in F^{n \times n}$, we construct its prefix forms $A_0, \ldots, A_n$ and obtain a list of r candidates for the sequence $\langle \mathrm{dep}(A_1), \ldots, \mathrm{dep}(A_n) \rangle$, using an r-enumerator for $\textsc{Dep}_F$ the same way we used the enumerator for the rank. Denote the candidate sequences by $\langle v_1^1, \ldots, v_n^1 \rangle$, ..., $\langle v_1^r, \ldots, v_n^r \rangle$. For each j, $1 \leq j \leq n$, we have $\mathrm{dep}(A_j) = \mathrm{dep}(A_{j-1})$ if and only if row j is independent of rows $1, \ldots, j-1$. Let $R_i = \{1 \leq j \leq n \mid v_j^i = v_{j-1}^i\}$, where we define $v_0^i = 0$ for all $1 \leq i \leq r$; hence R_i indexes the lexicographically smallest basis of the row space of A, claimed by the ith sequence. Our task is to determine which v_n^i is correct. We test, as in the proof of Theorem 1, the maximality of all R_i in parallel, keeping, for each i, a Boolean vector $u_i = (u_{ij})$ of length $|R_i|$ with 1s corresponding to the basis vectors whose dependency has been revealed by the test, i.e., for each i, u_i marks those vectors in R_i that were shown to have non-zero coefficients in linear decompositions of the remaining rows of A. We can ignore the bases that do not pass the maximality test. To simplify the notation, suppose that all of them do. Now we just need to test the consistency of each claimed v_n^i with the corresponding calculated dependency vector u_i. For each $1 \leq i \leq r$, let $c_i = v_n^i - (n - |R_i|)$; that is, c_i is the number of dependent vectors in R_i claimed by the ith candidate sequence $\langle v_1^i, \ldots, v_n^i \rangle$. We check the consistency by verifying, for all i in parallel,

whether $c_i = \sum_{j=1}^{|R_i|} u_{ij}$ holds. The value of $\mathrm{dep}(A)$ is given by the v_n^i corresponding to the smallest R_i that passes the consistency test. (If such R_i is not unique, each one forms a valid basis of the row space of A; furthermore, since given a basis, every other row of A is uniquely represented as linear combination of the basis vectors, all bases that pass the consistency test must agree on their corresponding claimed value for $\mathrm{dep}(A)$.) To verify the consistency, we simply need to be able to subtract two numbers of value at most n and to count the number of 1s in an n-bit binary string; both can clearly be done in TC^0. ∎

Proposition 3 $\textsc{Dep}_F$ is $\leq^{\mathrm{TC}^0}_{O(n)\text{-}tt}$-equivalent to $\textsc{Rank}_F$, where F is a finite commutative ring with identity.

Proof: $\textsc{Rank}_F$ is clearly n-tt-reducible to $\textsc{Dep}_F$ in TC^0. Given a matrix $A \in F^{n\times n}$, we ask the $\textsc{Dep}_F$ oracle for $\mathrm{dep}(A_0), \ldots, \mathrm{dep}(A_n)$ and simply count the number of i, $1 \leq i \leq n$, such that $\mathrm{dep}(A_i) = \mathrm{dep}(A_{i-1})$, which immediately gives the rank of A. To show that $\textsc{Dep}_F$ is reducible to $\textsc{Rank}_F$, we essentially use the reduction in Proposition 2. Namely, we query the $\textsc{Rank}_F$ oracle on $A_1, \ldots, A_n$ to obtain a valid basis of A, and then find the coefficients in the linear decompositions of the remaining rows in terms of the basis vectors. We need only count the number of the basis vectors that have a non-zero coefficient in some decomposition, which can be done in TC^0. Denote this number by s. Then the value of $\mathrm{dep}(A)$ is given by $n - \mathrm{rank}(A) + s$. ∎

4.3.3 The Graph of Rank

The *graph* of a function $f : \Sigma^* \to \Sigma^*$ is the set $\mathrm{Graph}_f = \{(x, y) \mid y = f(x)\}$. It is an easy observation that f is $O(1)$-enumerable in logspace iff f reduces to its graph with respect to $\leq^{\log}_{btt}$ (equivalently, $\leq^{\log}_{bT}$) reductions, provided that f is polynomially bounded. This was explicitely stated in [BHHV02] in the context of polynomial time. Indeed, if f is $(r + 1)$-enumerable for some constant

r, then clearly $f \leq^{\log}_{r\text{-}tt} \text{Graph}_f$. Also if $f \leq^{\log}_{r\text{-}T} \text{Graph}_f$, then f is 2^r-enumerable in logspace, equally trivially. It remains to observe that $f \leq^{\log}_{r\text{-}tt} \text{Graph}_f$ implies $f \leq^{\log}_{r\text{-}T} \text{Graph}_f$, and $f \leq^{\log}_{r\text{-}T} \text{Graph}_f$ yields $f \leq^{\log}_{(2^r-1)\text{-}tt} \text{Graph}_f$. (Here r is some fixed constant.) This it not at all surprising; intuitively, it says that if reducing the number of candidates for the value of f to a constant does not make f easier to compute, then computing f is, in some precise sense, as hard as *verifying* a solution. We thus have the following theorem.

Theorem 5 $\textsc{Rank}_F$ is not $\leq^{\log}_{bT}$-reducible to its graph, unless $\textsc{Rank}_F$ is in logspace.

4.4 Enumerability of the Determinant

Theorem 6 If, for some k, $\textsc{Det}$ is n^k-enumerable in logspace, then $\textsc{Det} \in \text{FL}$.

Proof: Let G be the configuration graph of a nondeterministic logspace machine on some input x. Let $\#\text{path}_G(s,t)$ denote the number of directed paths from node s to node t in G. Define $f(G)$ as the function, whose value (written in binary) consists of a sequence of n^2 blocks of length $s = 2n\lceil \log n \rceil$, the $(n(i-1)+j)$th segment corresponding to $\#\text{path}_G(i,j)$, where $1 \leq i,j \leq n$ and n is the number of nodes in G; thus

$$f(G) = \sum_{i,j=1}^{n} 2^{(n(i-1)+j)s} \#\text{path}_G(i,j).$$

It is easy to see that f is a #L function; this can be done by exhibiting an NL machine N whose number of accepting computation paths on G is $f(G)$. The machine N, on input G, nondeterministically guesses a number $p = n(i-1)+j$, $1 \leq p \leq n^2$, after which it guesses q, $1 \leq q \leq 2^{ps}$, followed by a guess of path from node i to node j in G. If the guess is correct, N accepts; otherwise, it rejects. It is easy to see that N is a nondeterministic logspace machine that has

the required number of accepting paths. Hence $f \in$ GapL, and a function is in GapL if and only if it is logspace many-one reducible to the determinant. Thus there must exist a logspace function g such that for all G, $f(G) = \textsc{Det}(g(G))$. We will use g to transform G into a matrix $M = g(G)$, and then run the n^k-enumerator on M to obtain a list of n^k values, one of which is the determinant of M. Using the equality in the above reduction, we convert this list to a list of n^k candidates for $f(G)$ (each of which, if correct, certifies that the corresponding claimed value for $\det(M)$ is correct). Since we have only n^k candidates for $f(G)$, they can be checked in parallel. Given a purported $f(G)$, we can uniquely read off $\#\text{path}_G(i,j)$ for each pair (i,j), $1 \leq i,j \leq n$. These values can then be locally checked using the self-reducibility of $\#\text{path}_G$. ∎

The theorem above clearly holds for fields of characteristic greater than the dimension of the matrix. A natural question is whether it holds for arbitrary finite fields. We can only show a similar result for the determinant of integer matrices modulo some integer p. The problem of computing the determinant mod p is $\leq_m^{\log}$-complete for Mod_pL, defined in [BDHM92]. Mod_pL is the class of sets A for which there exists $f \in \#\text{L}$ such that for all x, $x \in A$ iff $f(x) \not\equiv 0 \bmod p$. We will also use the notion of membership comparability, due to Ogihara [Ogi95].

Definition 9 A set S is logspace $g(n)$-*membership comparable*, written as $S \in$ L-mc$(g(n))$, if there exists a function $f \in$ FL such that for any set of $g(n)$ inputs $x_1, \ldots, x_{g(n)}$, each of length at most n, f excludes one of $2^{g(n)}$ candidates for the characteristic sequence $\chi_S(x_1, \ldots, x_{g(n)})$.

We will also use the predicate version of the problem, namely $\textsc{Det}$-mod-p = $\{(A,i) \mid \det(A) \equiv i \text{ (modulo } p)\}$.

Theorem 7 Let p be any prime. If there exists a logspace computable $(p-1)$-enumerator for $\textsc{Det}$-mod-p, then $\textsc{Det}$-mod-p is in logspace.

Proof: We will show that the existence of the enumerator above implies that DET-mod-p is in L-mc$(p-1)$. The theorem will then follow from a result by Ogihara and Tantau [OT01].

Since DET-mod-$p \in \mathrm{Mod}_p\mathrm{L}$, there must exist a #L-function f certifying its membership. Requiring that p is a prime is just a matter of convenience, since in this case, according to Fermat's Little Theorem, we may assume that f is such that $x \in$ DET-mod-p iff $f(x) \equiv 1(\text{modulo } p)$, and $x \notin$ DET-mod-p iff $f(x) \equiv 0(\text{modulo } p)$. Let M_f be a nondeterministic logspace machine certifying that f is in #L.

Let $k = p-1$. Consider a k-tuple of inputs $(x_1, \ldots, x_k)$, each of length at most n. For $1 \leq i \leq k$, denote the configuration graph of M_f on input x_i by G_i; then $f(x_i)$ counts the number of paths in M_f from the initial configuration to the accepting configuration. (We may assume that there is a single accepting configuration.) Given a directed graph G, let $\#\mathrm{path}_G(s,t)$ denote the number of directed paths from node s to node t in G; then $f(x_i) = \#\mathrm{path}_{G_i}(s_i, t_i)$, where s_i and t_i are the initial and the accepting configurations of G_i, respectively. Consider a graph G consisting of all G_is plus two additional nodes s and t. Besides the edges internal to G_is, add, for each i, a new edge from s to s_i, and from t_i to t. We have $\#\mathrm{path}_G(s,t) = \sum_{i=1}^{k} f(x_i)$.

For any integer p, given a topologically sorted directed graph G, nodes s and t, the problem of counting the number of paths from s to t in G modulo p, is in $\mathrm{Mod}_p\mathrm{L}$ (in fact, it is $\leq_m^{\log}$-complete for $\mathrm{Mod}_p\mathrm{L}$). Thus there must exist a logspace computable function that takes (G, s, t) as input and produces a matrix M such that $\det(M) \equiv \#\mathrm{path}_G(s,t)(\text{modulo } p)$. We will use this reduction to transform our G into such matrix M, and then run the enumerator on M. The enumerator gives $p-1$ candidates for DET-mod-p, thereby excluding one.

Thus we have a list of $p-1$ candidates for $\#\mathrm{path}_G(s,t) \bmod p$. We will show that no subset of $p-1$ candidates can cover all 2^{p-1} possibilities for the char-

acteristic function $\chi(x_1, \ldots, x_{p-1})$ (in DET-mod-p), implying that DET-mod-$p \in$ L-mc$(p-1)$. Indeed, suppose that the enumeration list contains q. We have

$$(\sum_{i=1}^{p-1} f(x_i)) \mod p = (\sum_{i=1}^{p-1} f(x_i) \mod p) \mod p,$$

and thus $\#\text{path}_G(s,t) \equiv q(\text{modulo } p)$ iff exactly q out of $x_1, \ldots, x_{p-1}$ are in the set. (Since, for any x, $f(x)$ is congruent to either 0 or 1 modulo p, we have $\sum_{i=1}^{p-1} f(x_i) \equiv q(\text{modulo } p)$.) Therefore, each candidate q covers (i.e. is consistent with) at most $\binom{p-1}{q}$ characteristic sequences; hence any $p-1$ candidates can clearly cover at most $2^{p-1}-1$ sequences, putting DET-mod-$p \in$ L-mc$(p-1)$. To get the membership in logspace, we will use the following theorem by Ogihara [Ogi95].

(Theorem 5.5 in [Ogi95]) Let $p \geq 2$. If $\text{Mod}_p\text{P} \subseteq$ P-mc$(c \log n)$ for some $c < 1$, then $\text{Mod}_p\text{P} \subseteq$ P.

The theorem carries over to logspace, as observed by Ogihara and Tantau [OT01]. This finishes the proof, since for any constants p and c, L-mc$(p-1) \subseteq$ L-mc$(c \log n)$. ∎

4.5 Final Remarks

A natural question is whether the rank being, say, $O(\log n)$-enumerable implies that the rank is in logspace. As we mentioned, it does not seem possible to combine more than a constant number of queries into a single query to the enumerator. Another improvement would be to show that Proposition 2 holds for AC^0 in place of TC^0, the question being whether counting the number of dependent basis vectors can be avoided in this context.

It is interesting to know whether Theorem 4 can be generalized to arbitrary rings; in particular, to the ring of integers. It follows from Remark 1, that the

problem reduces to finding the largest nonsingular matrix among r matrices of dimension at most n. Let the matrices be $D_1, \ldots, D_r$ with the corresponding dimensions $\tau_1, \ldots, \tau_r$. Using a lemma by Allender, Beals, and Ogihara [ABO99] (Corollary 2.3 in their paper), we can in logspace extend every matrix to dimension n without changing its singularity, so that if it was nonsingular, it remains so; otherwise its rank becomes one less than full. For notational simplicity, let $D_1, \ldots, D_r$ denote the extended matrices. We can run the enumerator on $D_1, \ldots, D_r$ (packed using the block diagonal construction) to generate r candidate sequences for $\langle \mathrm{rank}(D_1), \ldots, \mathrm{rank}(D_r) \rangle$. View the output of the enumerator as an $r \times r$ Boolean matrix $\mathcal{D} = (d_{ij})$ with $d_{ij} = 1$ iff $\mathrm{rank}(D_j)$ claimed by the i-th sequence is n. We can safely remove all columns that do not contain ones; assume that all do. It is easy to see that our r-enumerator is of no further use. Since the number of matrices is at most the number of candidate sequences that the enumerator has to commit itself to, the enumerator can make each one of its r purported answers consistent with a different choice of a nonsingular matrix among $D_1, \ldots, D_r$. Can we find the largest nonsingular D_i without the use of the enumerator?

If the rows of $\mathcal{D}$ form a Sperner system (i.e. the rows are incomparable elements of the Boolean r-dimentional cube), then $D_1, \ldots, D_r$ can be transformed into a set of r matrices such that exactly one matrix is nonsingular; moreover, the nonsingular one has the same index as the correct row of $\mathcal{D}$. To construct the jth matrix in the set, we simply take the product of all matrices that have ones in row j. The rank of the product is at most the minimum of the individual ranks; hence the product is not full-rank iff at least one matrix in the set is singular. The (only) nonsingular product corresponds to the correct row of $\mathcal{D}$.

In the case when the row set of $\mathcal{D}$ contains a chain, the nonsingular row product may not be unique. In this case we can perform the above process iteratively for all levels of the Boolean r-dimensional lattice, starting from level r. That is, for

every level i from r to 1, we take a subset of rows of $\mathcal{D}$ with exactly i ones, and get a set of less than r matrices with the property that *at most* one matrix in the subset is nonsingular. The correct row of $\mathcal{D}$ corresponds to the first nonsingular product.

Suppose we are given a set of matrices guaranteed to contain at most one nonsingular matrix; and let $\mathcal{C}$ be a functional complexity class. If we were able to single out the only nonsingular matrix (or to establish that none is full rank) in $\mathcal{C}$, then the problem of determining the singularity of a matrix (over the same domain) would trivially be in $\mathcal{C}$. Indeed, to determine whether a matrix A is singular, we would just run the above $\mathcal{C}$ procedure on A combined with some singular matrices. If A is nonsingular, the procedure will find it; otherwise it will establish the singularity of all, certifying that A is singular. Thus it is unlikely that the scenario outlined in Remark 1 works for arbitrary rings. Since the interesting case is when the enumerator is bounded to run in logspace, one could use randomness in constructing parallel queries from $D_1, \ldots, D_p$ (forcing the enumerator to reveal the singularity of some candidates due to its limited computational ability in guessing the random bits used in constructing the queries). However, approximate rank computations over arbitrary rings are complicated by the fact that the rank is not known to be random self-reducible, and the fact that infinite rings lack nice finite field properties simplifying the analysis (such as the existence of samplable distributions invariant under addition or multiplication by a fixed element of the ring).

To conclude, let us remark on the relation between Rank and Independence. The proof of Theorem 1 shows that we can use an r-enumerator to reduce the number of candidate rank sequences for $A_1, \ldots, A_n$ to r. Given a list of r purported sequences, consider the $r \times n$ Boolean matrix $H = (h_{ij})$ with $h_{ij} = \mathrm{rank}_i(A_j) - \mathrm{rank}_i(A_{j-1})$, where for each $l \in [n]$, $\mathrm{rank}_i(A_l)$ denotes the rank of A_l claimed by the i-th candidate sequence, and $\mathrm{rank}_i(A_0) = 0$ for all $i \in [r]$. (We can immediately

reject all candidate rank sequences whose correponding rows in H contain elements other than 0 or 1.) Suppose you are given a black box for INDEPENDENCE; that is, you can ask whether any h_{ij} is indeed 1, which essentially reveals the correct value for $\text{rank}(A_j) - \text{rank}(A_{j-1})$. Consider the following question: how many (adaptive) oracle calls to INDEPENDENCE one needs to determine which purported sequence of ranks is correct? The number of queries clearly does not have to be more than r, since you can always eleminate at least one sequence with a single query. The question is whether it can be much smaller.

This problem, in a more general formulation, seems to be of fundamental interest. Namely, the rows of H can be treated as truth tables of r-variate Boolean functions. Given oracle access to some unknown function $f : \{0,1\}^r \to \{0,1\}$, we are asked to recognize it among the functions explicitly specified by H, using as few oracle calls to f as possible. In other words, the question is how many points one needs to (adaptively) examine in order to reconstruct the unknown function f, given a restricted list of candidates containing f. For example, imagine a situation when you possess some cryptic program P_f guaranteed to compute one of your favorite functions. On how many different inputs do you have to run P_f to determine *which* one of your favorite functions it computes?

The lower bound on the number of queries for any deterministic procedure is $\Omega(\log r)$, given by a simple information-theoretic argument. Notice that if the number of unique columns of H is at least $2\sum_{i=\alpha r}^{r}\binom{r}{i}$ for some constant $\alpha < 1$, then a single query reduces the number of candidate sequences (i.e. rows of H) by at least $(1-\alpha)r$. Indeed, we just have to ask any h_{ij}, where j is the index of a column containing at least $(1-\alpha)r$ ones and at least $(1-\alpha)r$ zeros. We have $\sum_{i=\alpha r}^{r}\binom{r}{i} < \alpha 2^r/r(2\alpha-1)^2$, using standard approximation arguments. Denote the matrix resulting after query i by H_i (H_i contains the rows of H that agree with the first i queries), so $H_0 = H$. Let r_i denote the number of rows in H_i. The bound above shows that if the number of unique columns of H_0 is at least

$\Omega(2^{r_0}/r_0)$, then a single query leaves us with at most αr_0 candidate rows. In this case, the number of unique columns in H_1 is at least $\frac{\alpha 2^{r+1}}{r(2\alpha-1)^2 2^{(1-\alpha)r}} = \Omega(2^{r_1}/r_1)$; hence the same argument inductively applies: a single query can eliminate at least $(1-\alpha)r_1$ rows, and so on. Therefore, if the condition on the number of unique columns of H is satisfied, $\log_{1/\alpha} r = O(\log r)$ calls to Independence suffice to solve Rank.

5 A Note on Nondeterminism versus Randomness in Space

SUMMARY: We show that a restricted version of halting BPL is in NL.

We have seen that the class PL has natural characterizations as a counting class defined in terms of GapL functions, and as a probabilistic class defined in terms of probabilistic Turing machines. In this chapter we will take an alternative, probabilistic look at the space-bounded playground: We will look at probabilistic logspace computations and the complexity classes associated with it. Interestingly, although there has been considerable success in relating probabilistic classes to its own kind and to determinstic and nondeterministic classes, almost nothing interesting is known about how they relate to counting[1] or (uniform) circuit classes. The only points of correspondence are PL, L, and NL. In this sense, it is indeed an alternative view of the interiors of P.

There are certain subtleties that one should be aware of when defining probabilistic space-bounded classes. The most prominent such subtlety is that the power of these classes seems to be drastically affected by whether the machines are required to halt or not. By halting we mean halting on all inputs and for

[1]The alert reader may argue that probabilistic classes *are* counting classes. By counting here we mean classes whose most natural characterization is as a counting class.

all sequences of random coin tosses. It is easily seen that this distinction is not important in the context of time-bounded classes.

We briefly recall the definitions of the classes we need and the known relationships among them. An excellent survey by Saks [Sak96] provides a more refined and detailed exposition on the subject.

We adopt the standard definition of a probabilistic Turing machine (PTM), as defined by Gill [Gil77]. We only consider machines that compute *languages*. The random tape is one-way: The value of a previously seen bit can only be recalled if it has been explicitly stored on the work tape. For a PTM M and input x, we let $p_M(x)$ denote the probability that M accepts x. The language L computed by M is the set of strings x such that $p_M(x) > 1/2$. The class PL contains languages computed by logspace PTMs. The class BPL is the class of languages computed by two-sided bounded-error logspace PTMs. A *two-sided bounded-error* PTM is a PTM whose probability of acceptance on any input is either at most 1/3 or at least 2/3. The error is said to be *one-sided*, if the probability is either 0 or at least 2/3, and the corresponding class is RL. By imposing a restriction that the accepting machine for the language must halt on any input and for any sequence of coin tosses, we arrive at the corresponding *halting* classes, distinguished by the subscript H.

Imposing such a halting condition for a constructible space bound $s(n) \geq \log n$ is essentially equivalent to imposing a time bound of $2^{s(n)}$ on the maximum time of the computation. (See [Sak96, Gil77, BCD+89, Nis93] for more discussion of the subtleties involved.) We have already seen in Section 2.2.3 that PL = $\mathrm{P_H L}$ ([Jun85, AO96]); thus unbounded error machines can be restricted to run in polynomial time without any loss in power. In contrast, such a restriction seems to have a drastic affect on the power of BPL and RL. Without it, one-sided error is equivalent to nondeterminism, RL=NL [Gil77], which suggests that $\mathrm{P_H L}$ is probably a better definition of one-sided error probabilistic logspace computation.

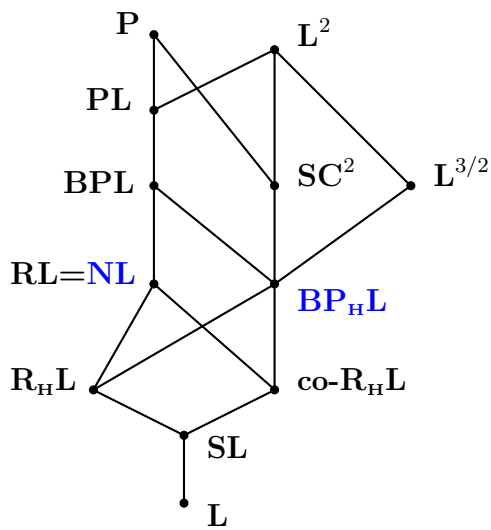

Figure 5.1: Known relationships.

Less is known when the error is two-sided. The above result certainly implies that NL $\subseteq$ BPL. Notice the outrageous difference compared to time-bounded classes: If NP $\subseteq$ BPP, then the polynomial hierarchy collapses to Σ_2^p, which follows from a theorem by Karp and Lipton [KL80] stating that if all languages in NP have polynomial-size circuits then PH $= \Sigma_2^p$; it is a simple observation that all languages in BPP have polynomial-size circuits (for example, see Theorem 11.6 in [Pap94a]). This suggests that BPL is not an appropriate definition as well (yet it is not as uninteresting as RL).

The known relationships among these classes are summarized in Figure 5.1. The only classes we have not encountered before are SC and SL. The class SC is defined as the class of languages decided by deterministic Turing machines that run simultaneously in polynomial time and polylogarithmic space [Coo85]. (As usual, the superscript indicates the power of the logarithm in the space bound.) The class SL denotes the class of languages computed by symmetric logspace

machines [LP82]. SL characterizes the complexity of *undirected* graph connectivity. Let us also remark that any randomized computation that uses only polylog many random bits has a logspace deterministic simulation [NZ96]. (Notice that simulating logarithmically many random bits is trivial.)

Even though it is believed that $\mathrm{BP_HL}$ collapses to L (i.e., that randomness does not help more than by a constant factor), it is not even known whether $\mathrm{BP_HL}$ is in NL. This is essentially the question of whether a logspace analog of BPP $\subseteq \Sigma_2^p$ ([Lau83, Sip83b]) holds. There is no obvious modification that can adapt the known nondeterministic simulations of BPP, since they seem to inherently require two-way access to nondeterministic bits. The question itself can also be thought of as asking whether randomness can be nontrivially substituted for nondeterminism in this context. We show that this inclusion holds for a restricted version of BPL, where the error is larger than what can be achieved by amplification. It is time to note that for bounded-error computation, the probability of error can be made exponentially small in the input length (by a straightforward amplification). More precisely, for every language in $L \in \mathrm{BP_HL}$ and every polynomial p, there is a $\mathrm{BP_HL}$ machine M for L with error probability at most $2^{-p(n)}$ on inputs of length n. (This can be formally shown by applying Chernoff bounds or Bernstein's law of large numbers to bound the probability that the average of sufficiently many random outcomes deviates from the expectation. We have no problem counting polynomially many outcomes in logspace.) In our restricted version, the error *is* exponentially vanishing in the input length, but the exponent depends on the running time of the machine. The argument is very simple and employs only basic structural properties of the configuration graph.

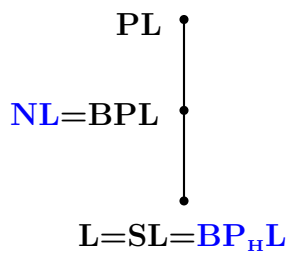

Figure 5.2: Conjectured relationships.

5.1 Toward $\mathrm{BP_HL} \subseteq \mathrm{NL}$

We will state the promised inclusion without defining the class $\mathrm{wBP_HL}$ alluded to above. Furthermore, we will start proving the inclusion as if we were proving it for $\mathrm{BP_HL}$, and see where our argument comes short of simulating arbitrary $\mathrm{BP_HL}$ machines. This will justify the definition of $\mathrm{wBP_HL} \subseteq \mathrm{BP_HL}$. We hope that the instructional benefit of doing so will outweigh the informality of presentation.

Theorem 8 $\mathrm{wBP_HL} \subseteq \mathrm{NL}$.

Proof: Take a language $L \in \mathrm{BP_HL}$, and let M be a probabilistic logspace Turing machine accepting L by clear majority. We can and shall assume that all possible runs of M on input x have length exactly $p(n)$ for some polynomial p, and the number of random choices at each step is exactly two: M reads a random bit at each step of its execution and either uses or discards it. This allows us to represent the set of all possible computations of M on input x as a leveled directed acyclic graph of depth $p(n)$. We call this graph the *configuration graph* of M on x, denoted $M(x)$. The nodes of $M(x)$ are partitioned into levels $L_0, \ldots, L_{p(n)}$, where L_i contains all possible configurations of M on x after M has seen exactly i random bits. Thus L_0 contains a single node, namely the start configuration. Transitions between two consecutive levels are defined in the obvious way. Since

M is bounded to run in polynomial time, the number of possible configurations of M on x, and thus the number of nodes in $M(x)$, is at most polynomial in n. We assume that $L_{p(n)}$ contains exactly two nodes, the accept node and the reject node. The configuration graph is in effect a full binary tree of depth $p(n)$ with subsets of nodes of the same level being embedded into a single node. We will also assume that the error of M is exponentially vanishing in the input size (see discussion above); thus for any input of length n, the error is at most $2^{-r(n)}$ for some polynomial r, fixed *before* M has been fixed.

Let x be an input of length n. We will give an NL procedure for deciding whether x is in L. Denote the set of accepting computations of M on x by $A(x)$. Alternatively, $A(x)$ is the set of all paths from the start node to the accept node of $M(x)$. To simplify notation, p and r will stand for $p(n)$ and $r(n)$, respectively. We know that if $x \in L$, then $|A(x)| \geq 2^p(1 - 2^{-r})$. On the other hand, if $x \notin L$, then $|A(x)| \leq 2^{p-r}$.

We will show that there is an NL-testable gap in the heights of the maximal full binary subtree embedded into the accepting subgraph of $M(x)$ for $x \in L$ and $x \notin L$. The accepting subgraph of $M(x)$ is the subgraph induced by $A(x)$. We denote the height of such maximal subtree by $d(x)$. Call a node of $M(x)$ an R-node, if all paths from it lead to the reject node. Clearly, in order to have an R-node at level $p - i$, there must be at least 2^i rejecting paths. Suppose that x is not in L. Then it is readily seen that $d(x) \leq p - r$, just because the number of accepting paths is at most 2^{p-r}. If, on the other hand, x is in L, the number of rejecting paths is less than 2^{p-r}, which can give rise to R-nodes only at levels $L_{p-r+1}, \ldots, L_p$. Thus $d(x) > p - (p - r) = r$. Notice that both bounds are tight. We want the gap between $p - r$ and r to be detectable in NL.

Suppose that $p = O(r)$. Then we could assume that $p < 3r/2$, and hence we would have that if $x \in L$ then $d(x) > r$; and if $x \notin L$, then $d(x) \leq r/2$. It remains to show how an NL machine could decide the membership of x in L. Easily: The

machine will just go through all nodes in level L_r (or any level numbered between $(\frac{r}{2}+1)$ and r), and check whether there is at least one R-node at that level. (Observe that all children of an R-node are necessarily R-nodes; thus if $x \notin L$, there must exist an R-node at level L_r. A node $u \in L_r$ can be checked by asking whether u is *not* reachable from the accept node. Recall that NL is closed under complement [Imm88, Sze87], and that reachability is in NL, thus the check is in NL. There are at most polynomially many nodes in L_r, which causes no problem for an NL machine.

Recall, however, that we needed the assumption that $p = O(r)$. But all we know is that for any polynomial r, there exists a desired probabilistic machine M with time bound p (for some polynomial p), whose error is vanishing as an inverse exponential in r. Here we essentially fix r after p has been fixed. Thus, although the error of M is still vanishing exponentially in the input size, this error depends on the running time of M (better thought of as the number of random bits used by M). This motivates the definition of a restricted version of $\mathrm{BP_HL}$, which we denote by $\mathrm{wBP_HL}$ (for *well-bounded error*). ∎

Formally, we define $\mathrm{wBP_HL}$ as the class of languages such that there exists a polynomial p and a PTM M accepting L using at most $p(n)$ random bits on inputs of length n (for all $n > 0$), and erring with probability at most $2^{-O(p(n))}$ on these inputs. Let $\mathrm{wBP_HL}(p)$ denote the subclass of languages in $\mathrm{wBP_HL}$, for which the defining polynomial is p.

A natural question is whether we can amplify the error to an inverse exponential without significantly increasing the number of random bits used. The short answer is "no", at least for black-box amplifiers. Let us assume that we have a probabilistic machine M that errs with some constant probability ϵ $(0 < \epsilon < 1/2)$. A straightforward way to amplify M is to run it r times and take the majority vote. This decreases the error to ϵ^r, alas the number of independent random bits used is increased by the factor of r. This does not help us much (in fact, at all). A

family of techniques has been devised for *deterministic amplification* that allows one to reduce the error probability while incurring only a small penalty in the number of additional random bits. All known amplification techniques use M as a black box. The idea is to still run M r times (using it as a black box), but to substitute independent random strings used by different runs by pseudo-random strings generated using fewer truly random bits. *Space*-bounded deterministic amplification is more involved. (Again, the rub is that the random tape is one-way, which precludes a straightforward adaptation of time-bounded amplification techniques.) Bar-Yossef, Goldreich, and Wigderson [BGW99] gave a nontrivial adaptation of the Ajtai-Komlós-Szemerédi method [AKS87] yielding the reduction of the error probability to an arbitrarily small constant with only a constant addition in the number of random bits. In particular, p random bits and error ϵ can be amplified in logspace to $p + O(r)$ random bits and error $\epsilon^{\Omega(r)}$, where r is arbitrary large, yet constant. This gives only a *constant* reduction of the error probability, even though [AKS87] actually gave an exponential reduction for BPP with only a linear increase in the number of random bits. For BPL, unfortunately, a constant reduction is the best achievable (for black-box simulators), as also shown by Bar-Yossef, Goldreich, and Wigderson [BGW99]. Thus the answer is "no", making the "well-boundedness" seemingly hard to remove.

A concluding remark. A fundamental idea exploited for many years is that computational hardness can be efficiently converted into computational randomness: Hard problems in some complexity class can be used to construct pseudo-random generators that "fool" computations in that class by generating sequences that look random to any such computation (using a much shorter truly random string). In the context of probabilistic time, there has been a sequence of results revealing an increasingly tighter tradeoff between the strength of the hardness assumption and the efficiency of the resulting simulation (ranging from a full derandomization to the weakest nontrivial (subexponential) simulation). Such a close con-

nection between hardness and randomness seems to be much less pronounced in the context of space. In fact, despite much success in understanding the structure of randomized logspace and despite the existence of unconditionally proven pseudo-random generators for logspace implying nontrivial deterministic simulations, there is no known result collapsing $\mathrm{R_H L}$ to L under any meaningful assumption. It would be extremely interesting to find an equally significant connection between the problem of derandomizing space-bounded computations and some other computational notion, or to better understand the relation between hardness and randomness in space.

Bibliography

[Ajt83] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulaion in logspace. In *Proceedings of 19th Symposium on Theory of Computing*, 132–140, 1987.

[All97a] E. Allender. A clarification concerning the #L hierarchy, October 1997. Available at ftp.cs.rutgers.edu/pub/allender/lh.pdf .

[All97b] E. Allender. Making computations count: Arithmetic circuits in the nineties. *SIGACT News Complexity Theory Column 19*, 1997.

[All90] E. Allender. Oracles vs proof techniques that do not relativize. In the *International Symposium SIGAL '90*, pages 39–52. Springer-Verlag *Lecture Notes in Computer Science* 450, 1990.

[ABH01] E. Allender, D. A. Mix Barrington, and William Hesse. Uniform circuits for division: consequences and problems. In the *Proceedings of 16th Conference on Computational Complexity*, pages 150–159. IEEE Computer Society Press, Los Alamitos, CA, 2001.

[ABO99] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8(2):99–126, 1999.

[AO96] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. *Theoretical Informatics and Applications*, 30(1):1–21, 1996.

[AW93] E. Allender and K. Wagner. Counting hierarchies: polynomial time and constant depth circuits. *Current Trends in Theoretical Computer Science*, G. Rozenberg and A.Salomaa, editors, World Scientific Series in Computer Science, Vol. 40, World Scientific Press, 469–483, 1993.

[AFKS00] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20:451–476, 2000.

[AGHP92] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Journal of Random Structures and Algorithms*, 3(3):289–304, 1992.

[AKNS99] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *Proceedings of the 40th Symposium on Foundations of Computer Science*, 645–655, 1999. Journal version: *SIAM* Journal on Computing, accepted for publication.

[ÀJ93b] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[ABG90] A. Amir, R. Beigel, and W. Gasarch. Some connections between bounded query classes and nonuniform complexity. In the *5th Structure in Complexity Theory Conference*, pages 232–243, 1990.

[ALRS92] S. Ar, R. Lipton, R. Rubinfeld, and M. Sudan. Reconstructing algebraic functions from mixed data. In *Proceedings of 33rd Symposium on Foundations of Computer Science*, pages 503–512. IEEE Computer Society Press, Los Alamitos, CA, 1992.

[ALRS98] S. Ar, R. Lipton, R. Rubinfeld, and M. Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing*, 28(2):487–510, 1998.

[AK02] V. Arvind and P. Kurur. Graph Isomorphism is in SPP. In *the 43rd Symposium on Foundations of Computer Science*, 743–750, 2002.

[BGW99] Z. Bar-Yossef, O. Goldreich, and A. Wigderson. Deterministic Amplification of Space-Bounded Probabilistic Algorithms. In *Proceedings of the 14th Conference on Computational Complexity*, 188-198, 1999.

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

[BI97] D. Barrington and N. Immerman. Time, hardware, and uniformity. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Restrospective II*, pages 1–22. Springer-Verlag, 1997.

[BCGT99] R. Beals, R. Chang, W. Gasarch, and J. Torán. On finding the number of graph automorphisms. *Chicago Journal of Theoretical Computer Science*, 1999(1), February 1999.

[BF90b] D. Beaver and J. Feigenbaum. Hiding instances in multioracle queries. In *Proceedings of 7th Symposium on Theoretical Aspects of Computer Science*, pages 37–48. Springer-Verlag *Lecture Notes in Computer Science 415*, 1990.

[BHHV02] R. Beigel, L. Hemaspaandra, H. Hempel, and J. Vogel. Optimal series-parallel trade-offs for reducing a function to its own graph. *Information and Computation*, 173: 123-131, 2002.

[BKS95] R. Beigel, M. Kummer, and F. Stephan. Approximable sets. *Information and Computation*, 120(2):304–314, 1995.

[BF97] R. Beigel and B. Fu. Circuits over PP and PL. *Journal of Computer and System Sciences*, 60(2):422–441, 2000.

[BCH+96] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan. Linearity Testing in Characteristic Two. *IEEE Transactions on Information Theory*, 42(6):1781–1795, 1996.

[BO03] A. Beygelzimer and L. Ogihara. The (non)enumerability of the determinant and the rank. *Theory of Computing Systems*, to appear.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[Bol86] B. Bollobás. *Combinatorics*. Cambridge University Press, 1986.

[BCD+89] A. Borodin, S. Cook, P. Dymond, W. Ruzzo, and M. Tompa. Two applications of inductive counting for complemetation problems. *SIAM Journal on Computing*, 18(3):559–578, 1989.

[BCP83] A. Borodin, S. Cook, and N. Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58:113–136, 1983.

[BvzGH82] A. Borodin, J. von zur Gathen, and J. Hopcroft. Fast parallel matrix and GCD computations. *Information and Control*, 52:241–256, 1982.

[BDHM92] G. Buntrock, C. Damm, U. Hertrampf, and C. Meinel. Structure and importance of Logspace-MOD class. *Mathematical Systems Theory*, 25(3):223–237, 1992.

[CH89] J. Cai and L. Hemachandra. Enumerative counting is hard. *Information and Computation*, 82(1):34–44, 1989.

[CH91] J. Cai and L. Hemachandra. A note on enumerative counting. *Information Processing Letters*, 38(4):215–219, 1991.

[CAS99] J. Cai, P. Aduri, and D. Sivakumar. On the hardness of permanent. In *Proceedings of STACS 99, 16th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 1563:90–99, Springer, 1999.

[CS98] J. Cai and D. Sivakumar. Sparse hard sets for P: resolution of a conjecture of Hartmanis. *Journal of Computer and System Sciences*, 59:280–296, 1998.

[CS00] J. Cai and D. Sivakumar. Resolution of Hartmanis' conjecture for NL-hard sparse sets. *Theoretical Computer Science*, 240:257–269, 2000.

[CSV84] A. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[Coo73a] S. Cook. A hierarchy for nondeterministic-time complexity. *Journal of Computer and System Sciences*, 7:343–353, 1973.

[Dam91] C. Damm. DET = $L^{\#L}$? Informatik-Preprint 8, Fachbereich Informatik der Humboldt-Universität zu Berlin, 1991.

[DGL+99] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikov, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. In *Proceedings of the 3rd International Workshop on Randomization and Approximation Techniques in Computer Science*, 97–108, 1999.

[dB70] N. G. de Bruijn. *Asymptotic methods in analysis*. North-Holland, Amsterdam, 1970.

[Ebe89] W. Eberly. Very fast parallel polynomial arithmetic. *SIAM Journal on Computing*, 18(5):955–976, 1989.

[FL92] U. Feige and C. Lund. On the hardness of computing permanent of random matrices. In *Proceedings of 24th Symposium on Theory of Computing*, pages 643–654, 1992.

[FFK94] S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.

[Fish01] E. Fischer. The art of uninformed decisions: A primer to property testing. In the Computational Complexity Column, *The Bulletin of the European Association for Theoretical Computer Science*, 75:97–126, 2001.

[FKR+02] E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky. Testing Juntas. In *the 43rd Symposium on Foundations of Computer Science*, 103–112, 2002.

[FLN+02] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual Symposium on Theory of Computing*, 474–483, 2002.

[FN01] E. Fischer and I. Newman, *Testing of matrix properties*, The 33rd Annual ACM Symposium on the Theory of Computing, 2001, to appear.

[Fra91] G. Frandsen. Parallel Construction of Irreducible Polynomials. *Technical Report DAIMI-PB-358*, University of Aarhus, Denmark, 1991.

[FSS84] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984.

[GS92] P. Gemmel and M. Sudan. Highly resilient correctors for polynomials. *Information Processing Letters*, 43:169–174, 1992.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[GS91] A. Goldberg and M. Sipser. Compression and ranking. *SIAM Journal on Computing*, 20(3):525–536, 1991.

[Gol97] O. Goldreich, *Combinatorial property testing*, Tech. Report TR97-056, Electronic Colloquium on Computational Complexity, 1997.

[GGL+00] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. Testing monotonicity. *Combinatorica*, 20:301–337, 2000.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[GT01] O. Goldreich and L. Trevisan, *Three theorems regarding testing graph properties*, Tech. Report TR01-010, Electronic Colloquium on Computational Complexity, 2001.

[GW02] O. Goldreich and A. Wigderson. Derandomization that is rarely wrong on advice strings that are typically good. In Proceedings of Random'02, Cambridge, MA, 2002.

[GP86] L. Goldschlager and I. Parberry. On the construction of parallel computers from various bases of boolean functions. *Theoretical Computer Science*, 43:43–58, 1986.

[GOR00] J. Goldsmith, M. Ogihara, and J. Rothe. Tally NP sets and easy census functions. *Information and Computation*, 158:29–52, 2000.

[GJH95] R. Greenlaw, H. Hoover, and W. Ruzzo. *Limits to parallel computation*. Oxford University Press, NY, 1995.

[Gup92] S. Gupta. On the closure of certain function classes under integer division by polynomially bounded functions. *Information Processing Letters*, 44(2):205–210, 1992.

[Gup95] S. Gupta. Closure properties and witness reduction. *Journal of Computer and System Sciences*, 50(3):412–432,1995.

[HS65] J. Hartmanis and R. Stearns. On the computational complexity of algorithms. *Transaction of the American Mathematics Society*, 117:285–306, 1965.

[Ha99] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[Hås87] J. Håstad. *Computational limitations for small-depth circuits*. MIT Press, Cambridge, MA, 1987.

[HO02] L. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*. Springer-Verlag, 2002.

[HR90] L. Hemachandra and S. Rudich. On the complexity of ranking. *Journal of Computer and System Sciences*, 41:251–271, 1990.

[HT02] L. Hemachandra and L. Torenvliet. *Theory of Semi-Feasible Algorithms*. Springer-Verlag, 2002.

[HS66] F. C. Hennie and R. E. Stearns. Two-tape simulation of multitape turing machines. *Journal of the Association for Computing Machinery*, 13(4):533–546, 1966.

[Hes01] W. Hesse. Division is in uniform TC^0. In the *28th International Colloquium on Automata, Languages and Programming*, 2001.

[HMU00] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 2000.

[HPV77] J. Hopcroft, W. Paul, and L. Valiant. On time versus space and related problems. *Journal of the ACM* , 24(2):332–337, 1977.

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

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

[Jun85] H. Jung. On probabilistic time and space. In the *Proceedings of 12th Conference on Automata, Languages and Programming*, pages 310–317. Springer-Verlag *Lecture Notes in Computer Science 194*, 1985.

[Kal92] E. Kaltofen. Polynomial factorization, 1987–1991. In I. Simon, editor, LATIN 92, *Lecture Notes in Computer Science*, 583:294–313, 1992.

[KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of 12th Symposium on Theory of Computing*, pages 302–309. ACM Press, 1980. Final version: Turing machines that take advice, *L'enseignement Mathématique* 28:191–209, 1982.

[Knu98] D. Knuth. *The Art of Computer Programming.* Seminumerical Algorithms, vol. 2, Addison Wesley, 1998.

[Lad75] R. Ladner. The circuit value problem is log space complete for P. *ACM SIGACT News*, 7(1):18–20, January 1975.

[LL76] R. Ladner and N. Lynch. Relativization of questions about logspace computability. *Mathematical Systems Theory*, 10(1):19–32, 1976.

[Lan02] S. Lang. *Algebra.* Springer-Verlag, New York, 2002.

[Lau83] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17:215–217, 1983.

[LP82] H. Lewis and C. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19:161–187, 1982.

[LN94] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1994.

[vL91] J. van Lint. *Introduction to coding theory.* Springer-Verlag, 1991.

[Lip91] R. Lipton. New directions in testing. In J. Feigenbaum and M. Merritt, editors, *Distributed Computing and Cryptography*, pages 191–202. DIMACS series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1991.

[Lov93] L. Lovász. *Combinatorial problems and exercises.* North-Holland, 2nd edition, 1993.

[Lyn76] N. Lynch. Log space machines with mutiple oracle tapes. *Theoretical Computer Science*, 6(1):25–39, 1976.

[MR95] R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, 1995.

[Mul87] K. Mulmuley. A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica*, 7:101–104, 1987.

[NN93] J. Naor and M. Naor. Small-bias probability spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

[New00] I. Newman, *Testing of functions that have small width branching programs*, Proceedings of the 41st Symposium on Foundations of Computer Science, 2000.

[Nis93] N. Nisan. On read-once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107:135–144, 1993.

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space, *Journal of Computer and System Sciences*, 52(1):43–52, 1996.

[Odl95] A. M. Odlyzko. Asymptotic enumeration methods. In R. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of Combinatorics*, volume I, pages 1063–1229. MIT Press, 1995.

[Ogi95] M. Ogihara. Polynomial-time membership comparable sets. *SIAM Journal on Computing*, 24(5):1168–1181, 1995.

[Ogi98] M. Ogihara. The PL hierarchy collapses. *SIAM Journal on Computing*, 27(5):1430–1437, 1998.

[OH93] M. Ogiwara and L. Hemachandra. A complexity theory for feasible closure properties. *Journal of Computer and System Sciences*, 46(3):295–325, 1993.

[OT01] M. Ogihara and T. Tantau. On the reducibility of sets inside NP to sets with low information content. Manuscript, 2002.

[Pap94a] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings of 6th GI Conference on Theoretical Computer Science*, pages 269–276. Springer-Verlag *Lecture Notes in Computer Science 145*, 1983.

[Pre91] F. Preparata. Inverting a Vandermonde matrix in minimum parallel time. *Information Processing Letters*, 38:291–294, 1991.

[Ron01] D. Ron. Property Testing (a tutorial). In P. Pardalos, S. Rajasekaran, J. Reif, J. Rolim editors, *Handbook of Randomization*, Kluwer Academic Publishers, 2001.

[RS96] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[Ruz81] W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383, 1981.

[RST84] W. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28(2):216–230, 1984.

[Sak96] M. Saks. Randomization and Derandomization in Space-Bounded Computation. In the *11th IEEE Structure in Complexity Theory Conference*, 1996.

[ST98] M. Santha and S. Tan. Verifying the determinant in parallel. *Computational Complexity*, 7(2):128–151, 1998.

[Sau72] N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory (A)*, 13:145–147, 1972.

[Sav83] W. Savitch. A note of relativized log space. *Mathematical Systems Theory*, 16(2):229–235, 1983.

[Sav70] W. Savitch. Relationships between nondeterministic and deterministic tape classes. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[Sch83] U. Schöning. A low and a high hierarchy in NP. *Journal of Computer and System Sciences*, 27:14–28, 1983.

[SFM75] J. Seiferas, M. Fisher, and A. Meyer. Separating nondeterministic time complexity classes. *Journal of the Association for Computing Machinery*, 25(1):146–167, 1975.

[Sel79] A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities on NP. *Mathematical Systems Theory*, 13:55–65, 1979.

[She72] S. Shelah. A combinatorial problem: Stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41:247–261, 1972.

[Shp92] I. Shparlinski. Computational and Algorithmic Problems in Finite Fields. *Kluwer Academic Publishers*, 1992.

[Sim75] J. Simon. *On some central problems in computational complexity.* PhD thesis, Cornell University, Ithaca, NY, 1975. Available as Cornell Department of Computer Science Technical Report TR75-224.

[Sim81b] J. Simon. Space-bounded probabilistic Turing machine complexity classes are closed under complement. In the *Proceedings of 13th Symposium on Theory of Computing*, pages 158–167. ACM Press, 1981.

[Sip83b] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of 15th Symposium on Theory of Computing*, pages 330–335. ACM Press, 1983.

[Siv99] D. Sivakumar. On membership comparable sets. *Journal of Computer and System Sciences*, 59(2):270–280, 1999.

[SHL65] R. Stearns, J. Hartmanis, and P. Lewis II. Hierarchies of memory limited computations. In *Proceedings of the Sixth Annual Symposium on Switching Circuit Theory and Logical Design*, 179–190, 1965.

[Sto85] L. Stockmeyer. On approximation algorithms for #P. *SIAM Journal on Computing*, 14(4):849–861, 1985.

[Sze87] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Bull. of the EATCS*, 33: 96-100, 1987.

[Sud97] M. Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[Tod89] S. Toda. On the computational power of PP and $\oplus$P. In the *30th IEEE Symposium on Foundations of Computer Science*, pages 514–519, 1989.

[Tod89] S. Toda. PP is as hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991. pages 514–519, 1989.

[Tod91] S. Toda. Counting problems computationally equivalent to computing the determinant. Technical Report CSIM 91-07, Department of Computer Science, University of Electro-Communications, Tokyo, Japan, 1991.

[Tor91] J. Torán. Complexity classes defined by counting quantifiers. *Journal of the Association for Computing Machinery*, 38(3):753–774, 1991.

[Val79a] L. Valiant. Completeness classes in algebra. In the *11th ACM Symposium on Theory of Computing*, pages 249–261, 1979.

[Val79b] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Val92] L. Valiant. Why is boolean complexity theory difficult. In M. Paterson, editor, *Boolean Function Complexity*, London Mathematical Society Lecture Notes Series 169, pages 84–94. Cambridge University Press, 1992.

[VV86] L. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(1):85–93, 1986.

[vM98] D. van Melkebeek. Deterministic and randomized bounded truth-table reductions of P, NL, and L to sparse sets. *Journal of Computer and System Sciences*, 58(2):213–232, 1998.

[VC71] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

[Vin91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In the *6th IEEE Structure in Complexity Theory Conference*, pages 270–284, 1991.

[vzG84] J. von zur Gathen. Parallel algorithms for algebraic problems. *SIAM Journal on Computing*, 13:802–824, 1984.

[vzG86] J. von zur Gathen. Parallel arithmetic computations. A survey. In *Proceedings, 12th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, 233: 93–112, 1986.

[vzG93] J. von zur Gathen. Parallel linear algebra. In J. Reif, editor, *Synthesis of Parallel Algorithms*, pages 574–615. Morgan Kaufmann, 1993.

[vzGG99] J. von zur Gathen and J. Gerhard. Modern Computer Algebra. *Cambridge University Press*, Cambridge, 1999.

[vzGS91] J. von zur Gathen and G. Seroussi. Boolean circuits versus arithmetic circuits. *Information and Computation*, 91:142–154, 1991.

[Wag86a] K. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[Wil85] C. Wilson. Relatizived circuit complexity. *Journal of Computer and System Sciences*, 31(2):169–181, 1985.

[Zak83] S. Zăk. A Turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983.