Arithmetic Circuits and Counting Complexity Classes

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

allender@cs.rutgers.edu

November 26, 2004

1 Introduction

Arithmetic circuits are the focus of renewed attention in the complexity theory community. It is easy to list a few of the reasons for the increased interest:

- Innovative work by Kabanets and Impagliazzo [KI03] shows that, in some cases, providing lower bounds on *arithmetic* circuit size can yield consequences about *Boolean* complexity classes. For instance, one of the most important problems in BPP that is not known to be in P is *Arithmetic Circuit Identity Testing* (ACIT), the problem of determining if two arithmetic circuits compute the same function. They show that the *Boolean* complexity of this problem is intimately linked to the *arithmetic* complexity of the Permanent.
- Although there has been a lack of dramatic progress in proving lower bounds for Boolean circuits, there has been some noteworthy progress in proving lower bounds for some classes of arithmetic circuits (for example, see [Shp01, GR00, Raz04]).

*Partially supported by NSF grant CCR-0104823.

- Arithmetic circuits provide useful characterizations of counting classes, and counting classes in turn characterize the *Boolean* complexity of many important computational problems.

This paper provides a detailed survey of *one small part* of the field of arithmetic circuit complexity: the relationship of counting classes to arithmetic circuits. The direction of this body of work runs entirely counter to the direction of most other work in arithmetic circuits, in the following sense. Most work on arithmetic circuits draws its interest from the fact that arithmetic circuits are *more restricted* than Boolean circuits, whereas the arithmetic circuits considered here will be shown to be *at least as powerful* as Boolean circuits. In spite of this fundamental difference, many of the techniques used in one area are equally useful in the other area as well, and for many important domains (such as circuits over finite fields) there is essentially no difference between the two settings. This is discussed in more detail in later sections.

Much of the material contained herein appeared earlier in a survey written for SIGACT News [All97]. In the intervening years, many open questions listed in [All97] have been solved, and many new developments have come about. Hence it was felt that an updated survey, with an updated list of open questions, was appropriate at this time.

The rest of this paper is organized as follows. Section 2 presents background and definitions about Boolean and arithmetic circuit complexity. Section 3 introduces the counting classes that we will concentrate on. Section 4 discusses settings where arithmetic circuits can simulate Boolean circuits. Following this, there is a sequence of sections, each focusing on a particular class of arithmetic circuits, presenting the main results and open problems related to each class. (There is no section on #P, since there are other nice surveys of this material, such as [For97].)

2 Circuit Complexity

Most of the basic definitions are the same, no matter what approach one is taking to the field of arithmetic circuit complexity. An arithmetic circuit is a directed acyclic graph with nodes (gates) computing arithmetic operations (such as $+, -, \times, \div$, etc.) over some algebraic structure R. Input gates receive elements of R, and each "wire" carries an element of R. If there are n input gates and one output gate, the circuit computes a (partial) function from R^n to R in the obvious way. (The circuit is computing only a partial function, because the result is undefined if divsion by zero occurs.) Boolean

circuits, which may be more familiar to the reader, are simply arithmetic circuits over the Boolean ring. Usually the term "arithmetic circuit" is used only when R is an algebraic structure other than the Boolean ring.)

Important measures associated with a circuit C are as follows:

Size: Usually this is taken to be the number of gates, although in some work the number of "wires" or edges is taken to be the size. Of course, these numbers are polynomially related.

Depth: The length of the longest path from an input to an output.

Degree: The degree of the formal polynomial computed by the circuit. More precisely, each node of the circuit has a particular degree. Input gates and constants are said to have degree 1. (This is a slight aberration, since the formal degree of a constant is zero; however it is more convenient to define these gates to have degree 1.) The degree of a $+$ or $-$ gate is the maximum degree of all the gates that feed into it. The degree of a $*$ gate is the sum of the degrees of all the gates that feed into it. The degree of a circuit is the degree of its output gate.

Rather than define the degree of a $\div$ gate, let us quote a fundamental result about removing (almost all) divsion gates from arithmetic circuits:

Theorem 1 *[Str73, Kal86, Kal88, BvzGH82] Let $f = g/h$ be a rational function on n variables over some field F, where g and h are relatively prime polynomials of degree d. If f can be computed by an arithmetic circuit of size s, then g and h can be computed by arithmetic circuits of size polynomial in $s + d$.*

This construction is also efficient in terms of depth (although it would be necessary to give additional details concerning fan-in, etc. for the type of circuits under consideration to give a precise statement).

Theorem 1 is most useful when the degree d of g and h is small (say, polynomial in n). In most (but not all) work on arithmetic circuits, attention is focused on circuits having degree polynomial in n. One reason for this is that a direct implementation of an arithmetic circuit is infeasible on a Boolean circuit if the degree is large. For instance, it is a simple matter to build a circuit with n multiplication gates computing the constant 2^{2^n}, and hence small circuits can produce outputs consisting of an exponential number of bits. (As observed by von zur Gathen [vzG85], over $\mathbb{Q}$ such circuits can still be evaluated efficiently with high probability, by doing the evaluation modulo a randomly-chosen prime.)

A lovely fact about small-degree arithmetic circuits is that they can simulated by circuits with small *depth*. To state this theorem, however, it is useful to introduce some notions (such as uniformity, and semi-unbounded fan-in, etc.) that are easier to motivate in the context of Boolean circuit complexity. Thus let us pause a moment in our discussion of arithmetic circuits, to introduce some fundamental concepts in Boolean circuit complexity.

2.1 Boolean Circuits

Since Boolean circuits are a special subclass of arithmetic circuits, we have already presented the most basic definitions. In almost all work dealing with (arithmetic or Boolean) circuit complexity, attention is focused on *families* of circuits $\{C_n : n \in \mathbb{N}\}$, where circuit C_n has input variables $x_1, \ldots x_n$.

2.1.1 Uniformity

Circuit families are a convenient formalism to use in defining complexity classes of languages or functions. In order to form useful connections with complexity classes defined by Turing machines, it is necessary to impose a *uniformity* requirement on circuit families. A circuit family $\{C_n\}$ is said to be *uniform* if there is a deterministic linear-time Turing machine (the "uniformity machine) that, given n and the name of a gate g, can determine all of the desired information about gate g (such as whether g is a AND gate or an OR gate, what the gates are that feed into g, etc.) Detailed definitions on uniformity can be found in [BIS90, Ruz81].[1] Since the input to the uniformity machine has length $O(\log n)$ (for a polynomial-size circuit), this notion of uniformity is usually called "DLOGTIME-uniformity". There are many other notions of uniformity that have been mentioned in the literature, such as logspace-uniformity and P-uniformity. For circuit complexity classes that are at least as powerful as logspace, it actually makes no difference if DLOGTIME-uniformity or logspace-uniformity is used [BIS90, Ruz81]. For very small complexity classes, there is general agreement that DLOGTIME-uniformity is the "right" notion to use, in the sense that it leads to more elegant statements of theorems and equivalent characterizations of complexity classes.

From one point of view, it would seem that P-uniformity would be the most natural definition of uniformity to use. For instance, if one is trying to

[1] For the case of NC^1 it seems desirable to use a slightly more restrictive notion of uniformity. See [Ruz81, Vol99] for details.

model what can be computed efficiently by circuits that are feasible to *construct*, then polynomial-time would seem to be the right notion of uniformity to consider. This point of view is explored in [All89]. However, the real reason for most occurrences of the phrase "P-uniformity" in the literature stems from the fact that for nearly two decades it was known that there are P-uniform circuits for division and related problems, having logarithmic depth, but it was not known how to improve the uniformity condition [BCH86]. The first significant step in improving the uniformity condition was reported in [CDL01] and the final solution was presented in [HAB02]. (See also [All01] for additional information.) (An unrelated problem for which P-uniformity was needed was presented in [AAI+01], but it was subsequently shown by Agrawal [Agr01] that, in this instance also, it was possible to state the theorems in terms of DLOGTIME-uniformity.) The circuits for division and iterated product presented in [HAB02] are for arithmetic over the integers, but as pointed out in [HAB02, AAI+01] the techniques are widely applicable to computation over other algebraic structures as well. As a consequence, almost every occurrence of the phrase "P-uniform" in the literature of circuit complexity (including those of the surveys [vzG93, vzGS91, vzG90]) can be replaced by "DLOGTIME-uniform".

2.1.2 Boolean Complexity Classes

Figure 1 presents some of the most important complexity classes, along with their characterization in terms of uniform families of Boolean circuits (with AND and OR gates, unless other types of gates are explicitly mentioned; inputs *and their negations* are available at the input level). In the third column of the table, you will find a list of some of the important problems that are complete for each class under the appropriate notion of reducibility. This list of complete problems helps underscore the importance of each of these classes, as a tool for helping us understand the complexity of real-world computational problems.

In the interest of a concise presentation, the table in Figure 1 contains two slight inaccuracies concerning the complete sets for TC^0 and ACC^0. For all other rows of the table, the complete sets are complete under many-one reductions computed by uniform AC^0 circuits. (AC^0 is the smallest complexity class that we will deal with in this paper. In the list of classes appearing above, it is the only one that is known to be a proper subclass of NP. As many textbooks now show [DK00, Vol99, HO02], AC^0 cannot

[2] A circuit is *skew* if each AND gate has fan-in two, and at least one input to the AND gate is an input variable x_i.

Class	Uniform Circuit Characterization	Complete Problems
NP	Size $2^{n^{O(1)}}$, Degree $n^{O(1)}$ [Ven92]	SAT, etc. [GJ79]
P	Size $n^{O(1)}$	Linear Programming, CVP, etc. [GHR95]
SAC^1	Size $n^{O(1)}$, Degree $n^{O(1)}$	Context-free Languages [Ruz80, Sud78, Ven91]
NL	Size $n^{O(1)}$, Skew^2[Ven92]	Shortest paths, Transitive Closure, etc.
NC^1	Depth $O(\log n)$, Bounded fan-in	Regular sets, Non-solvable monoids [Bar89] Boolean Formula Evaluation [Bus93]
TC^0	Depth $O(1)$, Size $n^{O(1)}$, MAJORITY gates	$\times, \div$, sorting [BCH86, RT92, HAB02]
ACC^0	Depth $O(1)$, Size $n^{O(1)}$, Unbounded fan-in, Mod_m	Solvable monoids [BT88]
AC^0	Depth $O(1)$, Size $n^{O(1)}$, Unbounded fan-in	$\{1\}$

Figure 1: Uniform circuit definitions of some complexity classes, and some sample sets complete under AC^0 reductions.

compute the parity function, and thus it is a proper subclass of ACC^0.) In contrast, it is widely believed that there are no complete sets for TC^0 and ACC^0 under this sort of reduction. Sorting, multiplication and division (over the integers) are complete for TC^0 under *Turing* reductions computable by uniform AC^0 circuits. (For definitions about circuit-based notions of reducibility, consult the text by Vollmer [Vol99].) ACC^0 is not even believed to have complete sets under this more general notion of reducibility, and thus the entry concerning "solvable monoids" deserves some explanation.

The proof in [Bar89] that there are regular sets that are complete for NC^1 relies crucially on showing that the word problem for any non-solvable group (or for any monoid containing a non-solvable group) is complete for NC^1. To build on this point, a new model of computation was presented, called a *program over a monoid*. Polynomial-size programs over any non-solvable monoid compute exactly the problems in NC^1, which leads one to ask what the computational power of solvable monoids is. It turns out to be exactly ACC^0 [BT88], and thus ACC^0 is in some sense the most natural subclass of NC^1 when viewed from this algebraic perspective. It also has a very natural circuit-based characterization as $ACC^0 = \bigcup_m AC^0[m]$ where $AC^0[m]$ is the class of problems computable by uniform families of circuits of polynomial size and constant depth, of unbounded fan-in AND, OR and

Mod_m gates (where a Mod_m gate returns 1 if and only if the number of inputs that evaluate to 1 is a multiple of m).

The reader is probably already familiar with NP and NL (nondeterministic polynomial time and nondeterministic logarithmic space, respectively). The only other information about the complexity classes listed above that we will need in what follows is that there is a nice characterization of SAC^1 as the class of "semi-unbounded" circuits of polynomial size and depth $O(\log n)$ [Ven91]. (See also Theorem 2 below.) A circuit is semi-unbounded if the AND gates have bounded fan-in, but the OR gates are allowed to have unbounded fan-in (or more generally, in the arithmetic case, the $\times$ gates have bounded fan-in, and the $+$ gates are allowed to have unbounded fan-in).

2.1.3 Non-uniform Complexity Classes

There is also a great deal of interest in studying complexity classes defined in terms of families of circuits *without* the uniformity restriction. Such circuit families are called *non-uniform*. Using the naming convention introduced in [KL82], the non-uniform analogues of P and NL are called P/poly and NL/poly. In contrast, the circuit classes are usually simply called non-uniform AC^0, non-uniform TC^0, etc. The reader should note that, using the circuit-based definition of NP given above, the "non-uniform" version of NP is not very interesting; it consists of the class of *all* Boolean functions! Thus it is quite different from the class NP/poly in the framework of [KL82].

Some of the most interesting results about Boolean circuits (and about the corresponding classes of arithmetic circuits that we will introduce in the next section) are known only to hold in the setting of non-uniform circuit complexity – although there is good reason to believe that they should hold also in the uniform setting.

2.2 Arithmetic Circuits

At this point, we can resume our discussion of arithmetic circuits where we left off, before our digression about Boolean circuits. Polynomial-size circuits of small degree can be simulated by circuits of small depth:

Theorem 2 *Let $\{C_n\}$ be a family of arithmetic circuits over a commutative semiring, having polynomial size and degree. Then there is an equivalent family of polynomial-size semi-unbounded circuits $\{D_n\}$ having depth $O(\log n)$.*

This was first proved in the non-uniform setting in [VSBR83]; related depth-reduction results for uniform circuits were proved for the Boolean ring in [Ruz81], and for $\mathbb{N}$ in [Vin91]. A general proof that works in the uniform setting over any commutative semiring appears in [AJMV98].

The time has come to distinguish the two approaches to circuit complexity that have been hinted to earlier. It is really quite simple. In the traditional approach to circuit complexity, the circuit C_n is quite limited in its ability to "look at" the values assigned to its input variables. For instance, if C_n is a circuit over the integers $\mathbb{Z}$ computing some function f, then it must compute f correctly even when given inputs whose binary representations have many more than n bits. For instance, such circuits cannot test if an input integer is odd or even, or test the value of other bits of the binary representation. Small Boolean circuits can compute (representations of) functions having very high arithmetic degree, but such functions cannot be computed by small arithmetic circuits. More detailed discussions of this sort of argument can be found in books such as [BM75, BCS96] and survey articles such as [vzG93, Bor82, vzGS91]. The real interest in proving lower bounds, therefore, consists in showing that certain *low-degree* functions cannot be computed by small arithmetic circuits. In particular, Valiant [Val79a] formulated arithmetic analogues of the classes P and NP, entirely in terms of n-variate polynomials having degree polynomial in n. This framework is surveyed nicely in [vzG87]; some more recent work in this paradigm may be found in [Bue00, Bla01]. In addition, some noteworthy recent lower bounds for classes of arithmetic circuits (not precisely in Valiant's framework) are [GR00, Shp01, Raz04].

The salient fact that needs to be emphasized is that arithmetic circuits have been studied as a *restricted* (or *structured*) model of computation – in contrast to Boolean circuits or Turing machines, which are "unrestricted" or "general" models of computation. Arithmetic circuits have a restricted set of operations that they can use; they don't have access to the individual bits of a description of an element of R that is given as an input. Proving that a function requires large arithmetic circuits does *not* imply that it requires large Boolean circuits, although the converse *does* hold (because arithmetic operations can be implemented efficiently on Boolean circuits).

In contrast to this traditional use of arithmetic circuits, for the rest of this paper we will use arithmetic circuits to represent functions on the set of strings $\{0,1\}^*$. That is, they are defined over the same domain as Boolean circuits (except that now the elements 0 and 1 are taken to be elements of the algebraic structure R). The reason for doing this is that it provides us with a convenient tool to describe Boolean complexity classes that characterize

the complexity of a variety of important computational problems.

It is worth mentioning that for any fixed finite field F, there is essentially no difference between these two approaches to studying circuits over F. As has often been observed (e.g., in [vzGS91]) each individual bit of the standard binary representation of an element of $x \in F$ can be represented by a constant-degree polynomial in x (where the constant in the degree depends on the size of F). Thus an arithmetic circuit taking inputs in F^n can compute the individual bits of each input field element, and thus traditional circuits can simulate arithmetic circuits that are given a binary representation of their inputs. Conversely, given the binary representation of an element of F, an arithmetic circuit can compute (in constant depth) the field element that is being represented. Thus all of the results in the rest of this paper that deal with finite fields carry over into the traditional setting of arithmetic circuit complexity.

3 Counting Classes

The primary motivation for the results on arithmetic circuits that we will be surveying, comes from *counting classes*. The canonical example of a counting complexity class is #P, the class of functions of the form $\#acc_M(x)$, counting the number of accepting paths of an NP machine M on input x [Val79b]. The class #L is defined similarly, but for NL machines M [AJ93].

Using the characterizations of NL and NP in terms of Boolean circuits as presented in Figure 1, we obtain a circuit-based characterization of the classes #P and #L, as follows. Start with a uniform family of Boolean circuits, and "arithmetize" these circuits in the most straightforward way, replacing each AND gate by a $\times$ gate, and each OR gate by a $+$ gate (and each negated input gate $\overline{x_i}$ is replaced by $1 - x_i$). It is shown in [Ven92] that when the arithmetic operations for the circuits are defined on $\mathbb{N}$, then the arithmetic circuits defined in this way compute precisely the functions in #P and #L.

Now that the first step has been made, it is natural to define the arithmetic versions of other complexity classes in precisely the same way, at least for classes that are defined in terms of uniform families of AND and OR gates. Referring back to Figure 1, we obtain in this way the following counting classes:

$$\#\mathrm{AC}^0 \subseteq \#\mathrm{NC}^1 \subseteq \#\mathrm{L} \subseteq \#\mathrm{SAC}^1 \subseteq \#\mathrm{P}$$

It is natural to ask what these "counting classes" are counting. #P and

#L were originally defined in terms of Turing machines, in order to count the number of accepting computation paths. What are arithmetic circuits counting? Venkateswaran and Tompa [VT89] pointed out that arithmetic circuits are counting "proofs of acceptance" – i.e. trees that "verify" that the circuit outputs 1, by selecting exactly one input to each Or gate that evaluates to 1, in the appropriate way. In this paper, we will discuss only the "arithmetization" of circuit classes defined with And and Or gates, although it is worth mentioning that this framework has also been extended to circuit families that are defined in terms of other types of gates [Che03].

At this point, we have an interpretation for arithmetic circuits over $\mathbb{N}$. Next, it is natural to ask if we can say anything about arithmetic circuits over the integers.

It turns out that this notion has been studied quite a lot. The class of functions $\{0,1\}^* \to \mathbb{Z}$ computed by uniform arithmetic circuits of exponential size and polynomial degree is exactly the class GapP studied in [FFK94] (see also [For97]); GapP was originally defined as the class of all functions that are the difference of two #P functions, but it is not hard to show that this is an equivalent definition. In exactly the same way, we obtain several other "gap" classes: GapL, GapAC^0, GapNC^1, and GapSAC^1. It requires a clever argument to show that every function in GapAC^0 can be expressed as the difference of two $\#\mathrm{AC}^0$ functions [ABL98], but for all of the other Gap$\mathcal{C}$ classes, it is straightforward to show that they can be expressed as the difference of two $\#\mathcal{C}$ functions.

GapP was originally introduced in [FFK94] as a tool for studying classes of *languages* that can be defined using GapP. In particular, the following complexity classes were singled out for study:

- $\mathrm{PP} = \{A : \exists f \in \mathrm{GapP}\ (x \in A) \Leftrightarrow f(x) > 0\}$
- $\mathrm{C_{=}P} = \{A : \exists f \in \mathrm{GapP}\ (x \in A) \Leftrightarrow f(x) = 0\}$
- $\mathrm{SPP} = \{A : \chi_A \in \mathrm{GapP}\}$ (Here, χ_A denotes the characteristic function of A.)

Let us not forget that analogous definitions in terms of #P also yield interesting and important classes:

- $\mathrm{NP} = \{A : \exists f \in \#\mathrm{P}(x \in A) \Leftrightarrow f(x) > 0\}$
- $\mathrm{coNP} = \{A : \exists f \in \#\mathrm{P}(x \in A) \Leftrightarrow f(x) = 0\}$
- $\mathrm{UP} = \{A : \chi_A \in \#\mathrm{P}\}$

Of course, in a completely analogous way, we can define classes of languages using all of the other arithmetic circuit complexity classes discussed above. For instance, the classes PL, $C_=L$, UL, PNC^1, $C_=NC^1$, PAC^0 and $C_=AC^0$ have all received study.

Lastly, we should remember that uniform arithmetic circuit families over finite fields define Boolean language classes. For instance, uniform exponential-size arithmetic circuits of polynomial degree over $\mathbb{F}_2$ provide a characterization of the well-known class $\oplus$P, and more generally, arithmetic circuits over a field F of characteristic p yield Mod$p$P.

One's initial reaction may be to recoil in horror at this overabundance of complexity classes. However, the good news is that many of these complexity classes turn out to be the same. (Sometimes this is obvious, and in other cases it is surprising and gives new insights into familiar complexity classes.) More interestingly, there are several cases where it seems safe to conjecture that two classes are the same, but we cannot yet prove that the classes coincide. Furthermore, we shall see that these classes serve to clarify the complexity of some important computational problems that have withstood precise classification.

4 Can Arithmetic Circuits Really Simulate Boolean Circuits?

It is time to back up the assertion, made in Section 1, that arithmetic circuits (in the model that we're concentrating on, where the circuits can access each bit of the input) are more powerful than Boolean circuits.

In some cases this is trivial to prove, in other cases it follows as a consequence of some rather surprising results, and in still other cases it is probably false.

First, let's see a case where it is probably false. Consider the Boolean class NP. If the zero-one valued characteristic function of all languages in NP are in #P, then NP = UP. (In fact, these are equivalent statements.) Even the weaker hypothesis that these characteristic functions are in GapP is viewed as rather unlikely. (This is equivalent to NP $\subseteq$ SPP.) Of course, in the non-uniform setting, both of these inclusions are trivially true, for the uninteresting reason that non-uniform exponential-size families of circuits can do essentially *everything*.

Next, let's mention the cases where it is trivial to prove. Any function computed by NC^1 circuits or by AC^0 circuits is easily seen to be computed by "unambiguous" circuits of the same type [CMTV98, AAD00], and thus

for these classes, arithmetic circuits over $\mathbb{N}$ are easily seen to be *at least* as powerful as Boolean circuits. (An OR gate g in a circuit C is said to accept an input x "unambiguously" if it evaluates to 1 when C is given input x, and *exactly one* of the inputs to the gate g evaluates to 1. C is said to be "unambiguous" if, for every input x and every OR gate g in C that evaluates to 1 on x, g accepts x unambiguously.) Furthermore, since AC^0 circuits are too weak even to compute the sum of their inputs (i.e., to count the number of 1's) [FSS84], $\#\mathrm{AC}^0$ is a strictly more powerful class than AC^0.

The two remaining classes, #L and $\#\mathrm{SAC}^1$, are more problematic. If every language in NL has its characteristic function in #L, then NL = UL. However, since it was shown in [RA00] that NL/poly = UL/poly, it is no longer clear if this should be considered unlikely. In fact, the results of [RA00] show that, in the non-uniform setting, every function that can be computed by Boolean NL circuits *is* in #L, and thus the arithmetic circuits *are* at least as powerful as the Boolean circuits. Analogous results hold for $\#\mathrm{SAC}^1$. Furthermore, there is reason to believe that this simulation should also be possible in the uniform setting; results of [ARZ99] and [KvM02] (using the Nisan-Wigderson generators [NW94]) show that if there is any problem in $\mathrm{DSPACE}(n)$ that requires circuits of size $2^{\epsilon n}$, then NL = UL, and every Boolean function computed by polynomial-size skew circuits is in #L, and every Boolean function computed by SAC^1 circuits is in $\#\mathrm{SAC}^1$.

Open Question 1 *Can this simulation (i.e.,* NL = UL*) be proved in the uniform setting with no unproven assumptions?*

4.1 Circuits With Zero-Tests

Another model that has received extensive study is the model of *Arithmetic-Boolean Circuits* of von zur Gathen. (For instance, see [vzG93, vzGS91, BCGR92].) These are circuits with both Boolean gates and arithmetic gates, as well as two additional types of gates (*test*: $R \to \{0,1\}$ and *select*: $R^2 \times \{0,1\} \to R$) that provide an interface between the Boolean and arithmetic parts:

$$\mathit{test}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise.} \end{cases}$$

$$\mathit{select}(x_0, x_1, y) = \begin{cases} x_0 & \text{if } y = 0 \\ x_1 & \text{otherwise.} \end{cases}$$

It is worthwhile investigating the consequences of augmenting our arithmetic circuits with gates of this sort, as a tool for studying counting classes.

For each of the classes $\#\mathcal{C}$ and Gap$\mathcal{C}$ defined above, define the classes of circuits $\mathsf{Arith.Bool}\#\mathcal{C}$ and $\mathsf{Arith.BoolGap}\mathcal{C}$, where *test* and *select* gates are also allowed.

It is easy to verify that $\mathsf{Arith.Bool}\#\mathrm{AC}^0 = \#\mathrm{AC}^0$ and $\mathsf{Arith.Bool}\#\mathrm{NC}^1 = \#\mathrm{NC}^1$. One consequence of [RA00] is that, in the non-uniform setting $\mathsf{Arith.Bool}\#\mathrm{SAC}^1 = \#\mathrm{SAC}^1$ and $\mathsf{Arith.Bool}\#\mathrm{L} = \#\mathrm{L}$. As observed above, these equalities hold also in the uniform setting, under a reasonable hypothesis. As a consequence, for arithmetic circuits over $\mathbb{N}$, it seems that zero-testing does not add much computational power.

In contrast, things look quite different when we consider arithmetic-Boolean circuits over the integers. We shall see in Section 8 that $\mathsf{Arith.BoolGapAC}^0$ coincides with the functions computable by constant-depth threshold circuits (TC^0), and thus is strictly more powerful than GapAC^0.

It is not hard to see that the condition $\mathsf{Arith.BoolGapL} = \mathrm{GapL}$ is equivalent to $\mathrm{C_=L} = \mathrm{SPL}$. (This holds also for general classes $\mathcal{C}$.) The question of whether or not $\mathrm{C_=L} = \mathrm{SPL}$ is in some sense analogous to the NL=UL question, but it is not clear how far this analogy can be pushed. In particular, there seems to be little reason to believe that the classes $\mathrm{C_=L}$ and SPL should coincide.

5 Arithmetic SAC^1 Circuits

One of the most important facts about arithmetic circuits of polynomial size and degree is that they are equivalent to semi-unbounded circuits of logarithmic depth (Theorem 2).

All of the functions in $\#\mathrm{SAC}^1$ can be computed by threshold circuits of logarithmic depth (known as TC^1 circuits). (In fact, it is observed in [AJMV98] that if gates for integer division (throwing the remainder away) are added to $\#\mathrm{SAC}^1$ circuits, then one obtains an exact characterization of TC^1.) On the other hand, nothing is known about the relative power of $\#\mathrm{SAC}^1$ and AC^1 (the class of problems accepted by logarithmic depth, unbounded-fan-in circuits of AND and OR gates). A possible first step toward answering this question is taken in [AJMV98], where it is shown that problems in AC^1 are reducible to questions about arithmetic circuits of polynomial size and degree $n^{O(\log\log n)}$, improving the trivial upper bound of $n^{O(\log n)}$.

The canonical complete problems for $\#\mathrm{SAC}^1$ have the form of counting the number of different parse trees for a given string x in some context-free grammar. No other natural complete problems are known.

#SAC^1 shares with #L and #P the property that functions in this class count accepting paths on a well-studied type of machine. More precisely, Vinay showed in [Vin91] that #SAC^1 can be characterized as the class of functions of the form $\#acc_M(x)$, where M is a logspace-bounded nondeterministic auxiliary pushdown machine. Additional connections are explored in [NR95, LR90].

Arithmetic SAC^1 circuits over $\mathbb{F}_2$ were studied in [GW96], where it was shown that, in the non-uniform setting, these circuits can simulate Boolean SAC^1 circuits. This work was built upon in [RA00], where it is shown that in the non-uniform setting, all languages in SAC^1 are accepted by unambiguous logspace-bounded auxiliary pushdown automata. Equivalently, there is a language accepted by one of these unambiguous machines that is complete for SAC^1 under non-uniform logspace reductions.

Open Question 2 *Is there an unambiguous context-free language that is complete for* SAC^1 *under (non-uniform) logspace reductions? As observed in [RA00], this would yield a logarithmic-time CREW-PRAM algorithm for recognizing context-free languages.*

6 Skew Arithmetic Circuits

#L and GapL have received a great deal of attention, because of the following important fact:

Computing the determinant of integer matrices is complete for GapL.

A paper by Mahajan and Vinay [MV97] gives a beautiful proof of this theorem, and it also provides references for the various places where this theorem was first proved independently.

Cook first focused attention on the class of problems reducible to the determinant in [Coo85]. He defined this class in terms of NC^1 reducibility, and he observed that many of the problems for which fast parallel algorithms are known lie in this class. For a great many important problems A, the class of problems reducible to A under NC^1 reductions coincides with the problems reducible to A under AC^0 reductions. Is this also the case for the determinant?

This question was first posed in [AO96], where the following hierarchies were defined:

- The Exact Counting Logspace Hierarchy $= \mathrm{C_{=}L}^{\mathrm{C_{=}L}^{\cdot^{\cdot^{\cdot^{\mathrm{C_{=}L}}}}}} = \mathrm{AC}^0(\mathrm{C_{=}L})$ = the class of problems AC^0-reducible to the set of singular integer matrices.

- The PL hierarchy $= \mathrm{PL}^{\mathrm{PL}^{\cdots^{\mathrm{PL}}}} = \mathrm{AC}^0(\mathrm{PL}) =$ the class of problems AC^0-reducible to the problem of computing the high-order bit of the determinant of integer matrices.

- The #L hierarchy $= \mathrm{L}^{\#\mathrm{L}^{\cdots^{\#\mathrm{L}}}} = \mathrm{AC}^0(\#\mathrm{L}) =$ the class of problems AC^0-reducible to computing the determinant of integer matrices.

The first two of these hierarchies collapse, and they coincide with NC^1 reducibility.

- $\mathrm{AC}^0(\mathrm{C}_=\mathrm{L}) = \mathrm{L}^{\mathrm{C}_=\mathrm{L}} = \mathrm{NC}^1(\mathrm{C}_=\mathrm{L})$ [ABO99].

- $\mathrm{AC}^0(\mathrm{PL}) = \mathrm{PL} = \mathrm{NC}^1(\mathrm{PL})$ [Ogi98, BF00].

(These hierarchies are defined using "Ruzzo-Simon-Tompa" reducibility [RST84], which is the usual notion of oracle access for space-bounded nondeterministic Turing machines.) It seems natural to conjecture that AC^0- and NC^1-reducibility coincide on #L, too. If they do, then the #L hierarchy collapses.

Proposition 3 *If* $\mathrm{AC}^0(\#\mathrm{L}) = \mathrm{NC}^1(\#\mathrm{L})$*, then the* #L *hierarchy collapses.*

This is stated without proof in [AO96], but it was pointed out by Mahajan and Vinay (personal communication) that this actually requires some proof. A proof is sketched in an appendix to this paper.

One of the first papers to explicitly study GapL in terms of arithmetic circuits was [Tod92]. One of the contributions of [Tod92] is an argument showing that some extensions of the class of "skew" arithmetic circuits also yield exactly GapL. (For instance, using Toda's generalization, it is obvious that all GapNC^1 functions are in GapL. With the original definition of "skew" circuits, this is not obvious.) This is useful for showing that GapL is closed under some forms of reducibility. However, neither these results, nor the techniques of [RA00], seem to be sufficient to prove any sort of collapse of the #L hierarchy.

One candidate that one might put forward for a problem in the second level of the #L hierarchy is the following "iterated determinant" problem: Given as input n^2 matrices $M_{i,j}$, compute the determinant of the matrix M whose (i,j)th entry is $\mathrm{DET}(M_{i,j})$. However, it was shown in [AAM03] that this function actually lies in GapL.

The main reason to be interested in $\mathrm{PL}, \mathrm{C}_=\mathrm{L}$ and related classes is this: They characterize the complexity of some important and natural problems. For instance, the set of singular matrices (matrices with determinant zero) is complete for $\mathrm{C}_=\mathrm{L}$, and a variety of other problems regarding

Class	Complete Problems
GapL	DET,Characteristic Polynomial (χ_M)
	Matrix Inversion, Matrix Powering
PL	Is DET positive?
	Inertia of matrices with no opposite nonzero eigenvalues
	Positive Stable Matrices
	Positive Semistable Matrices
$L^{C_=L}$	Similarity, Rank, Equivalence
	Is the rank odd?
	Feasible Systems of Linear Equations
	Do M_1 and M_2 have the same minimal polynomial?
	Verifying the low-order term of μ
	Diagonalizability
$C_=L$	Singular matrices
	Is the rank $> r$?
	Is degree(μ) less than d?
	Verifying the characteristic polynomial

Figure 2: Problems in linear algebra complete for logspace counting classes.

computation of the rank and determining if a system of linear equations is feasible are complete for $L^{C_=L}$ [ABO99]. (Interestingly, it was shown in [AV05] that determining feasibility of a system of linear equations *modulo a given small prime* p is hard for the seemingly larger class $L^{\#L}$, and is in non-uniform $L^{\#L}$.) Some other problems in linear algebra and problems involving Markov decision processes were shown to be complete for PL in [Jun84, AMGL00]. A series of papers by Thierauf and Hoang [HT03, HT04, HT02, HT01, HT00, Hoa03] has pinpointed the complexity of several other natural problems in linear algebra, such as the minimal polynomial (μ), the characteristic polynomial (χ), inertia, similarity, equivalence, and diagonalizability. These results are summarized in Figure 2. Note that for computation over finite fields of characteristic p, *all* of these problems are complete for Mod$p$L [BDHM91]. Additional examples of problems complete for (non-uniform) $L^{\#L}$ are presented in [AV05], including the problem of testing if two permutation groups are isomorphic, and computing the order of a permutation group.

One important problem whose complexity remains unresolved is the perfect matching problem. No uniform deterministic NC algorithm is known for matching at all, but in [ARZ99] the probabilistic NC algorithm of [MVV87]

was combined with the determinant algorithm of [MV97] to show that the matching problem lies in non-uniform SPL. Since the matching problem is hard for NL (see [ABO99, KUW86]), this "sandwiches" the complexity of this problem between NL and SPL (at least in the non-uniform setting).

Further investigation of SPL may be useful in approaching the question of whether or not the #L hierarchy collapses. By analogy with a result of [FFK94] (showing that $\mathrm{SPP} = \{A : \mathrm{GapP}^A = \mathrm{GapP}\}$), one can show that $\mathrm{SPL} = \{A : \mathrm{GapL}^A = \mathrm{GapL}\}$. Are there other interesting problems that lie in this class?

Open Question 3 *Does the* #L *hierarchy collapse (at least in the non-uniform setting)?*

Open Question 4 *Are there natural problems that are complete for higher levels of the* #L *hierarchy? Can finer comparisons be made among the various problems known to be in the* #L *hierarchy but not known to be in* GapL*, including some of the problems listed above, as well as the following:*

- *Finding roots of a univariate polynomial [Nef94].*
- *Counting Eulerian Paths [Tod92].*
- *Computing the minimal polynomial.*

Open Question 5 *For the following linear algebraic questions, the upper and lower bounds presented in [HT03, HT04, HT02, HT01, HT00] do not quite match. (Other open problems can also be found in these papers.) Can tight completeness results be proved?*

Problem	*Upper Bound*	*Lower Bound*
Congruence	PL	$\mathrm{L}^{\mathrm{C_=L}}$
Positive Definite Matrices	PL	$\mathrm{coC_=L}$

Open Question 6 *Is* $\mathrm{C_=L}$ *closed under complement?*

Open Question 7 *Can any relationship be presented between* $\mathrm{C_=L}$ *and* $\oplus$*L? (See also [Hoa03] for related open questions.)*

7 Arithmetic NC^1 Circuits

$\#\mathrm{NC}^1$ coincides with the class of functions that have arithmetic formulae of polynomial size. As such, it has been studied as a complexity class at

least since [Val79a]. Evaluating arithmetic formulae over $\mathbb{N}$ ($\mathbb{Z}$) is complete for $\#\mathrm{NC}^1$ (GapNC^1, respectively) [BCGR92]. It follows from [CDL01] that every function in $\#\mathrm{NC}^1$ is computable in logspace.

Probably the most important and fascinating open question regarding arithmetic NC^1 is the following:

Open Question 8 *Is* GapNC^1 *equal to Boolean* NC^1*?*

We have already observed that $\#\mathrm{NC}^1$ is at least as powerful as Boolean NC^1. A hint that they might be the *same* class of functions is provided by the following theorem.

Theorem 4 *[Jun85] Let* $f \in \mathrm{GapNC}^1$*. Then* f *is computed by a family of Boolean circuits having bounded fan-in, polynomial size, and depth* $O(\log n \log^* n)$*.*

Jung's proof is somewhat complicated. Here is a short and simple proof that came up in discussion with M. Agrawal and S. Datta.

Proof. It is observed in [CMTV98] that the techniques of Ben-Or and Cleve [BOC92] show that the following problem is complete for GapNC^1:
Input: A sequence of 3×3 integer matrices, $M_1, M_2, \ldots, M_n$.
Output: The (1,1) entry of $\prod_i M_i$.

It is thus not hard to show that the following function f is hard for $\#\mathrm{NC}^1$:
Input: A sequence of n 3×3 matrices, $M_1, M_2, \ldots, M_n$ of n-bit integers, and an n-bit natural number m.
Output: $\prod_i M_i \pmod m$.

Let $D(n)$ denote the depth required to compute f on instances involving n-bit integers.

Our approach to compute f will be as follows. For all primes p having $\theta(\log n)$ bits, compute each $M_i \bmod p$. Then compute $\prod_i M_i \pmod p$, and finally, using the Chinese Remainder Theorem, recover our answer $\prod_i M_i \pmod m$. Except for the problem of computing $\prod_i M_i \pmod p$, this can all be done in depth $O(\log n)$ [BCH86]. How can we compute $\prod_i M_i \pmod p$?

If we divide the sequence $M_1, M_2, \ldots M_n$ into subsequences consisting of $\log n$ matrices, then we have an instance of our original problem f of size $\log n$. By combining these subproblems in a $(\log n)$-ary tree of height $\log n / \log\log n$, we easily obtain the following recurrence relation:

$$D(n) = O(\log n) + O(\frac{\log n}{\log\log n} D(\log n)).$$

Substituting $D(\log n) = O(\log\log n) + O(\frac{\log\log n}{\log\log\log n} D(\log\log n))$ into this expression we obtain

$$\begin{aligned} D(n) &= O(\log n) + O(\frac{\log n}{\log\log n}(O(\log\log n) + O(\frac{\log\log n}{\log\log\log n} D(\log\log n)))) \\ &= O(\log n) + O(\log n) + O(\frac{\log n}{\log\log\log n}(D(\log\log n))) \end{aligned}$$

This is now easily seen to yield $D(n) = O(\log n \log^* n)$. ∎

For all "practical" purposes, $\log n \log^* n$ is $O(\log n)$. Thus there may seem to be no "practical" reason to worry about whether the factor of $\log^* n$ can be removed. However, it is certainly a tantalizing theoretical question. (Over any fixed finite field, arithmetic NC^1 circuits do coincide with Boolean NC^1.)

Even though there seems to be essentially no room "between" Boolean NC^1 and $\#\mathrm{NC}^1$, there are a surprisingly large number of natural problems lying in this area.

For example, consider the class of languages accepted by probabilistic finite automata. It follows from [CMTV98] that all of these languages are in PNC^1, and that there are some such languages that are complete for this class. (Macarie's paper [Mac98] also provides a number of references for more information about probabilistic finite automata. In spite of a large literature on these languages, their complexity has only recently become better understood.) If the $\log^* n$ factor can be removed, then their complexity will be resolved.

Another well-studied example is the two-sided Dyck language (also known as the word problem for the free group with two generators). It has been known since the work of Lipton and Zalcstein [LZ77] that this problem is in L. It was shown by Robinson [Rob93] that it is hard for NC^1. It was observed in [CMTV98] that the problem is in $\mathrm{C}_=\mathrm{NC}^1$. Thus it is sandwiched between NC^1 and $\mathrm{C}_=\mathrm{NC}^1$. If the $\log^* n$ factor can be removed, then its complexity will be resolved.

One of the most surprising and important results about Boolean NC^1 is Barrington's theorem [Bar89], characterizing NC^1 in terms of width-5 branching programs. It is natural to wonder if this characterization also gives an equivalent characterization of $\#\mathrm{NC}^1$. The authors of [CMTV98] investigated this question, by defining the class of functions corresponding to counting paths through bounded-width branching programs, #BWBP. They showed that #BWBP is contained in $\#\mathrm{NC}^1$, but it remains an open question if these classes are equal. There has even been some speculation

in the community that these two classes may really be different, since the techniques used to prove Barrington's theorem do not seem to help in this setting. On the other hand, the authors of [CMTV98] did show (using the techniques of Ben-Or and Cleve [BOC92]) that GapNC^1 is equal to the class of functions that are the difference of two #BWBP functions.

All functions computed by Boolean NC^1 circuits are in #BWBP (in width 7). (Sketch: Let $f(x) = y_r, \ldots, y_1, y_0$. We will view this string as the binary representation of the number $\sum_i 2^i y_i$. There will be $r+1$ blocks in the branching program. Width 1 will be used to provide a path into each block from the start node. Block i will use width 5 to compute the bit y_i, and then multiply this value by 2^i (re-using the width 5), and then add this value to a total accumulated in the remaining layer.) Thus if the $\log^* n$ factor in Jung's theorem can be removed, then #NC^1 and #BWBP coincide.

Multiplying together a sequence of n 3×3 integer matrices is complete for GapNC^1 under many-one reducibility. (That is, for every function $g \in$ GapNC^1, there is an AC^0 function f, with the property that for all x, $f(x)$ is a sequence of 3×3 integer matrices M_i, and $g(x)$ is equal to the (1,1) entry of the product of the M_i.) This leads naturally to the following questions:

Open Question 9

- *What about 2×2 integer matrices? (Robinson [Rob93] has shown that multiplying a sequence of 2×2 integer matrices is hard for Boolean* NC^1 *under* AC^0 *many-one reducibility, but this problem is not known to be hard for* #NC^1*.)*

- *What about $k \times k$ matrices over $\mathbb{N}$? (It is shown in [CMTV98] that for 6×6 matrices, this is hard for* #NC^1 *under* AC^0*-Turing reducibility, by computing the difference of two* #BWBP *functions. However, showing hardness under many-one reducibility would show that* #BWBP *is equal to* #NC^1*.)*

- *The result of [Rob93] alluded to above also shows that multiplying 2×2 matrices over $\mathbb{N}$ is hard for Boolean* NC^1 *(but over $\mathbb{N}$ we obtain only hardness under* AC^0*-Turing reductions; additional argument would be necessary to show hardness under many-one reductions). A special case of multiplying 2×2 matrices is the problem of counting paths in width-two grid graphs. It is known that even this restricted problem is hard for* NC^1 *under* ACC^0*-Turing reducibility [AAB$^+$99]. Can this be improved (say, to* AC^0*-Turing reducibility)?*

8 Constant-Depth Arithmetic Circuits

The main reason to be interested in $\#AC^0$ is because it provides an alternative way of viewing TC^0, the class of problems computed by constant-depth threshold circuits.

All of the other arithmetic complexity classes discussed above are at least as powerful as Boolean NC^1, and thus we do not know if there is any problem in NP that does not have small arithmetic circuits of that sort. On the other hand it is shown in [AAD00] that the zero-one-valued functions in $GapAC^0$ are exactly the languages in $AC^0[2]$ (that is, the languages accepted by constant-depth polynomial-size circuits of AND, OR, and PARITY gates). The results of [Raz87, Smo87] show that there are many simple languages (such as the Mod 3 function) that are not in $AC^0[2]$, and thus current lower bound techniques apply to these very small arithmetic circuit classes. These techniques were applied in [AAB+99] to establish various non-closure properties of $GapAC^0$.

However, the main result of [AAD00] is that TC^0 is exactly $C_=AC^0$ (which is also equal to PAC^0). (By [HAB02] this holds even in the uniform setting.) (This then establishes the assertion made back in Section 4.1, that $\mathsf{Arith.Bool}GapAC^0$ coincides with the functions computable by constant-depth threshold circuits.) Thus, if we could expand our repertoire of lower bounds for $GapAC^0$, we might obtain lower bounds for TC^0 circuits.

It is important to add very quickly that there are some reasons to expect that this attempt might not prove fruitful. Razborov and Rudich show that, if strong enough pseudorandom generators are computable in TC^0, then no "natural" proof can show that P/poly is not contained in TC^0 [RR97]. Furthermore, the results of [NR97] show that, if some popular cryptographic assumptions are correct, then strong enough pseudorandom generators *are* computable in TC^0. Thus, if popular cryptographic assumptions are correct, this attempt to prove lower bounds for TC^0 is doomed to failure, *unless* expanding our repertoire of lower bounds for $GapAC^0$ leads us outside the framework of "natural proofs" as considered in [RR97].

It is observed in [AAD00] that arithmetic AC^0 circuits over $\mathbb{F}_2$ compute exactly the functions in $AC^0[2]$, and circuits over $\mathbb{F}_3$ compute exactly the functions in $AC^0[6]$. More generally, constant-depth arithmetic circuits over finite fields give an exact characterization of ACC^0.

Constant-depth arithmetic circuits have surprising computational power. For example, a construction credited to Ben-Or in [Shp01] shows that there is a sequence of constant-depth arithmetic circuits over $\mathbb{Z}$ (in the traditional sense, where the circuits do *not* have access to the bits of the input) com-

puting the function $c_n * p_n(x_1, \ldots, x_n)$ where c_n is a sequence of constants and p_n is the degree $n/2$ elementary symmetric polynomial on n variables. As observed in [AAB+99], it follows from the fact that MAJORITY is not in $\mathrm{AC}^0[2]$ that the constants c_n cannot be removed. By the same observation, the function $\binom{\sum_{i=0}^{n} x_i}{s(n)}$ is not in GapAC^0 for any function $s(n)$ that grows more rapidly than polylogarithmic. In contrast, it is shown in [AAD00] that for *any* $f \in$ GapL and for any k, the function $\binom{f}{k}$ is in GapAC^0.

Open Question 10 *Is* $\binom{\sum_{i=0}^{n} x_i}{\log^* n}$ *in* GapAC^0*?*

9 Arithmetic Circuits Over Small Finite Fields

Up to now, whenever we have mentioned arithmetic circuits over finite fields, it was assumed that the *same* field was being used by each circuit in a family $\{C_n\}$. However, there are also instances when it is useful to consider circuit families over a varying sequence of fields. The counting classes ModP [KT96] and ModL [AV05] have this flavor; they provide language classes related to #P and #L, respectively.

Rather than burden the reader with more definitions of complexity classes at this point, let us instead consider one example of a problem in "traditional" arithmetic circuit complexity that deals with arithmetic circuits over fields of polynomial size, which can be discussed in terms of the tools that we have presented thus far.

Consider the sequence of polynomials Q_n for the n-variable Permanent function over $\mathbb{F}_{p_n}$ (where $p_1, p_2, \ldots$ is an enumeration of all of the primes). Is there a sequence of polynomial-size, constant-depth arithmetic circuits C_n, computing Q_n over $\mathbb{F}_{p_n}$? This question is still open (although a positive answer would imply that #P has non-uniform TC^0 circuits). It has only recently become possible to state unconditionally that no such family can be DLOGTIME-uniform. Otherwise, there would be DLOGTIME-uniform TC^0 circuits for the Permanent (since arithmetic circuits over $\mathbb{F}_{p_n}$ can be evaluated in uniform TC^0, and the Chinese remaindering algorithm of [HAB02] would allow us to compute the value over $\mathbb{Z}$ in uniform TC^0.) However, it was shown in [CMTV98] that #P is not in DLOGTIME-uniform TC^0. (Specific superpolynomial lower bounds are presented in [All99].)

Interestingly (and frustratingly), for any field F of characteristic $p \neq 2$, it is *not* known if the permanent over F requires superpolynomial-size uniform constant-depth arithmetic circuits (even though uniform arithmetic circuits over finite fields characterize uniform ACC^0, and the permanent is

known to require exponential size on uniform ACC^0 circuits [AG94]). In contrast, for fields F of characteristic 2 it is known that the determinant (and hence the permanent) is complete for $\oplus$L, which contains functions that are not in (non-uniform) $AC^0[2]$ [Raz87, Smo87]. Hence the permanent and determinant require superpolynomial size constant depth arithmetic circuits over any field of characteristic 2.

10 Conclusions

Arithmetic circuits provide a useful formalism with which to capture the complexity of a variety of important computational problems. They help reveal the algebraic structure underlying some important complexity classes. Many basic questions about these complexity classes are still open, and there is reason to believe that some of these questions may prove tractable.

Acknowledgments: This survey has been enhanced thanks to many conversations with colleagues, such as M. Agrawal, V. Arvind, D. Mix Barrington, S. Datta, K.-J. Lange, M. Mahajan, S. Buss, P. McKenzie, R. Niedermeier, K. Reinhardt, D. Thérien, T. Thierauf, Thanh Minh Hoang, and V. Vinay. I thank Rod Downey for arranging a visit to Victoria University in Wellington, NZ, where much of this paper was written.

References

[AAB+99] A. Ambainis, E. Allender, D. A. M. Barrington, S. Datta, and H. LêThanh. Bounded depth arithmetic circuits: Counting and closure. In *Proc. ICALP*, number 1644 in Lecture Notes in Computer Science, pages 149–158. Springer, 1999.

[AAD00] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and arithmetic circuits. *Journal of Computer and System Sciences*, 60:395–421, 2000.

[AAI+01] M. Agrawal, E. Allender, R. Impagliazzo, R. Pitassi, and S. Rudich. Reducing the complexity of reductions. *Computational Complexity*, 10:117–138, 2001.

[AAM03] E. Allender, V. Arvind, and M. Mahajan. Arithmetic complexity, Kleene closure, and formal power series. *Theory of Computing Systems*, 36:303–328, 2003.

[ABL98] A. Ambainis, D. A. M. Barrington, and H. LêThanh. On counting AC^0 circuits with negated constants. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science 1450*, pages 409–419, 1998.

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

[AG94] E. Allender and V. Gore. A uniform circuit lower bound for the permanent. *SIAM Journal on Computing*, 23:1026–49, 1994.

[Agr01] M. Agrawal. The first-order isomorphism theorem. In *Conference on Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, volume 2245 of *Lecture Notes in Computer Science*, pages 70–82, 2001.

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

[AJMV98] E. Allender, J. Jiao, M. Mahajan, and V. Vinay. Noncommutative arithmetic circuits: Depth reduction and size lower bounds. *Theoretical Computer Science*, 209:47–86, 1998.

[All89] E. Allender. P-uniform circuit complexity. *J. ACM*, 36:912–928, 1989.

[All97] E. Allender. Making computation count: Arithmetic circuits in the nineties. *ACM SIGACT News Complexity Theory Column*, 28(4):2–15, 1997.

[All99] E. Allender. The permanent requires large uniform threshold circuits. *Chicago Journal of Theoretical Computer Science*, (7), 1999.

[All01] E. Allender. The division breakthroughs. In *The Computational Complexity Column in Bulletin of the EATCS*, volume 74, pages 61–77. EATCS, 2001.

[AMGL00] E. Allender, Martin Mundhenk, Judy Goldsmith, and Christopher Lusena. The complexity of policy evaluation for finite-horizon partially-observable markov decision processes. *Journal of the ACM*, 47:681–720, 2000.

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

[ARZ99] E. Allender, K. Reinhardt, and S. Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59(2):164–181, 1999.

[AV05] V. Arvind and T. C. Vijayaraghavan. The complexity of solving linear equations over a finite ring. In *22nd Symposium on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.

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

[BCGR92] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM Journal on Computing*, 21:755–780, 1992.

[BCH86] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15:994–1003, 1986.

[BCS96] P. Bürgisser, M. Clausen, and A. Shokrollahi. *Algebraic Complexity Theory*. Grundlehren der mathematischen Wissenschaften, 315. Springer-Verlag, 1996.

[BDHM91] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. Structure and importance of logspace-MOD class. *Math. Systems Theory*, 25:223–237, 1991.

[BF00] R. Beigel and B. Fu. Circuits over PP and PL. *Journal of Computer and System Sciences*, 60:422–441, 2000.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, December 1990.

[Bla01] M. Blaeser. Complete problems for Valiant's class of qp-computable families of polynomials. In *7th Annual International Computing and Combinatorics Conference (COCOON)*, Lecture

Notes in Computer Science 2108, pages 1–10. Springer-Verlag, 2001.

[BM75] A. Borodin and I. Munro. *The Computational Complexity of Algebraic and Numeric Problems*. Elsevier, 1975.

[BOC92] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21:54–58, 1992.

[Bor82] A. Borodin. Structured versus general models in computational complexity. *L'Enseignement Mathématique*, 30:47–65, 1982. Special Issue *Logic and Algorithms*, Symposium in honour of Ernst Specker.

[BT88] D. A. Mix Barrington and D. Thérien. Finite monoids and the fine structure of NC^1. *Journal of the ACM*, 35:941–952, 1988.

[Bue00] P. Buergisser. Cook's versus Valiant's hypothesis. *Theoretical Computer Science*, 235:71–88, 2000.

[Bus93] S. R. Buss. Algorithm for Boolean formula evaluation and for tree contraction. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 96–115. Clarendon Press, Oxford, 1993.

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

[CDL01] A. Chiu, G. Davida, and B. Litow. Division in logspace-uniform NC^1. *RAIRO Theoretical Informatics and Applications*, 35:259–276, 2001.

[Che03] H. Chen. Arithmetic constant-depth circuit complexity classes. In *28th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Lecture Notes in Computer Science 2747, pages 328–337. Springer-Verlag, 2003.

[CMTV98] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

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

[DK00] D.-Z. Du and K.-I. Ko. *Theory of Computational Complexity*. Wiley-Interscience, New York, 2000.

[FFK94] Stephen A. Fenner, Lance J. Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

[For97] L. Fortnow. Counting complexity. In A. Selman and L. A. Hemaspaandra, editors, *Complexity Theory Retrospective II*, pages 81–107. Springer Verlag, New York, 1997.

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

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, New York, 1995.

[GJ79] M. Garey and D. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman and Company, 1979.

[GR00] D. Grigoriev and A. Razborov. Exponential complexity lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields. *Applicable Algebra in Engineering, Communication and Computing*, 10:465–487, 2000.

[GW96] A. Gál and A. Wigderson. Boolean vs. arithmetic complexity classes: randomized reductions. *Random Structures and Algorithms*, 9:99–111, 1996.

[HAB02] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

[HO02] Lane A. Hemaspaandra and Mitsunori Ogihara. *The complexity theory companion*. Springer-Verlag, 2002.

[Hoa03] Thanh Minh Hoang. *On the Complexity of some Problems in Linear Algebra*. PhD thesis, Univ. Ulm, 2003.

[HT00] Thanh Minh Hoang and Thomas Thierauf. The complexity of verifying the characteristic polynomial and testing similarity. In *15th IEEE Conference on Computational Complexity (CCC)*, pages 87–95. IEEE Computer Society Press, 2000.

[HT01] T. M. Hoang and T. Thierauf. The complexity of the minimal polynomial. In *26th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Lecture Notes in Computer Science 2136, pages 408–420. Springer-Verlag, 2001.

[HT02] Thanh Minh Hoang and Thomas Thierauf. The complexity of the inertia. In *22nd Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, Lecture Notes in Computer Science 2556, pages 206–217. Springer-Verlag, 2002.

[HT03] Thanh Minh Hoang and Thomas Thierauf. The complexity of the characteristic and the minimal polynomial. *Theoretical Computer Science*, 295:205–222, 2003.

[HT04] T. M. Hoang and T. Thierauf. On the minimal polynomial. *International Journal of Foundations of Computer Science*, 15:89–105, 2004.

[Jun84] H. Jung. On probabilistic tape complexity and fast circuits for matrix inversion problems. In *Proc. ICALP*, number 172 in Lecture Notes in Computer Science, pages 281–291. Springer, 1984.

[Jun85] H. Jung. Depth efficient transformations of arithmetic into Boolean circuits. In *Proc. FCT*, number 199 in Lecture Notes in Computer Science, pages 167–173. Springer, 1985.

[Kal86] E. Kaltofen. Uniform closure properties of p-computable functions. In *ACM Symposium on Theory of Computing (STOC)*, pages 330–337, 1986.

[Kal88] E. Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *Journal of the ACM*, 35:231–264, 1988.

[KI03] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *ACM Symposium on Theory of Computing (STOC)*, pages 355–364, 2003.

[KL82] R. Karp and R. Lipton. Turing machines that take advice. *L'enseignement Mathématique*, 28(3/4):191–209, 1982.

[KT96] J. Köbler and S. Toda. On the power of generalized MOD-classes. *Mathematical Systems Theory*, 29:33–46, 1996.

[KUW86] R.M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6(1):35–48, 1986.

[KvM02] Adam R. Klivans and Dieter van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31:1501–1526, 2002.

[LR90] K.-J. Lange and P. Rossmanith. Characterizing unambiguous augmented pushdown automata by circuits. In *Proc. of 15th Symposium on Mathematical Foundations of Computer Science*, number 452 in Lecture Notes in Computer Science, pages 399–406. Springer, 1990.

[LZ77] R. Lipton and Y. Zalcstein. Word problems solvable in logspace. *Journal of the ACM*, 24:522–526, 1977.

[Mac98] I. Macarie. Space-efficient deterministic simulation of probabilistic automata. *SIAM Journal on Computing*, 27:448–465, 1998.

[MV97] M. Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, (5), 1997.

[MVV87] K. Mulmuley, U. Vazirani, and V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

[Nef94] C. A. Neff. Specified precision polynomial root isolation is in NC. *Journal of Computer and System Sciences*, 48:429–463, 1994.

[NR95] R. Niedermeier and P. Rossmanith. Unambiguous auxiliary pushdown automata and semi-unbounded fan-in circuits. *Inform. and Control*, 118(2):227–245, 1995.

[NR97] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudorandom functions. In *Proceedings of the 38th IEEE Symp. on Foundations of Computer Science*, pages 458–467, 1997.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

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

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

[Raz87] A. A. Razborov. Lower bounds on the size of bounded depth networks over a complete basis with logical addition. *Mathematicheskie Zametki*, 41:598–607, 1987. English translation in Mathematical Notes of the Academy of Sciences of the USSR 41:333-338, 1987.

[Raz04] R. Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. In *ACM Symposium on Theory of Computing (STOC)*, pages 633–641, 2004.

[Rob93] D. Robinson. *Parallel algorithms for group word problems*. PhD thesis, Univ. of California, San Diego, 1993.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *Journal of Computer and System Sciences*, 55:24–35, 1997.

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

[RT92] J. Reif and S. Tate. On threshold circuits and polynomial computation. *SIAM J. Comput.*, 21:896–908, 1992.

[Ruz80] W. L. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21:218–235, 1980.

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

[Shp01] A. Shpilka. Lower bounds for small depth arithmetic and Boolean circuits. Ph.d. thesis, Hebrew University, 2001.

[Smo87] R. Smolensky. Algebraic methods in the theory of lower bounds for Boolean circuit complexity. In *Proceedings, 19th ACM Symposium on Theory of Computing*, pages 77–82, 1987.

[Str73] V. Strassen. Vermeidung von Divisionen. *J. reine u. angew. Math.*, 264:182–202, 1973.

[Sud78] I. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25:405–414, 1978.

[Tod92] S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Trans. Inf. and Syst.*, E75-D:116–124, 1992.

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

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

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43:380–404, 1991.

[Ven92] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM Journal on Computing*, 21:655–670, 1992.

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

[Vol99] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York Inc., 1999.

[VSBR83] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12:641–644, 1983.

[VT89] H. Venkateswaran and M. Tompa. A new pebble game that characterizes parallel complexity classes. *SIAM Journal on Computing*, 18:533–549, 1989.

[vzG85] J. von zur Gathen. Irreducibility of multivariate polynomials. *Journal of Computer and System Sciences*, 31:225–264, 1985.

[vzG87] J. von zur Gathen. Feasible arithmetic computations: Valiant's hypothesis. *J. Symbolic Computation*, 4:137–172, 1987.

[vzG90] J. von zur Gathen. Inversion in finite fields using logarithmic depth. *J. Symbolic Computation*, 9:175–183, 1990.

[vzG93] J. von zur Gathen. Parallel linear algebra. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 573–617. Morgan Kaufmann, 1993.

[vzGS91] Joachim von zur Gathen and Gadiel Seroussi. Boolean circuits versus arithmetic circuits. *Information and Computation*, 90(2):142–154, February 1991.

11 Appendix: Proof of Proposition 3

In this section, we provide a proof of the claim that if $\mathrm{AC}^0(\#\mathrm{L}) = \mathrm{NC}^1(\#\mathrm{L})$, then the #L hierarchy collapses.

Proof. (Sketch) The proof consists of establishing the following fact.

Fact 5 *The set of languages* $\mathrm{NC}^1(\#\mathrm{L})$ *has a complete set under logspace many-one reducibility.*

This fact is sufficient to establish the claim. To see this, assume that $\mathrm{NC}^1(\#\mathrm{L}) = \mathrm{AC}^0(\#\mathrm{L})$. By Claim 5, there is a complete set for $\mathrm{AC}^0(\#\mathrm{L})$. Since $\mathrm{AC}^0(\#\mathrm{L})$ is equal to $\mathrm{L}^{\#\mathrm{L}^{\#\mathrm{L}^{\cdot^{\cdot^{\cdot^{\#\mathrm{L}}}}}}}$, this complete set is in some fixed level of this hierarchy, and thus $\mathrm{AC}^0(\#\mathrm{L})$ collapses to this level.

It remains for us to establish Fact 5.

First, let's come up with a definition of a "canonical" way for a log-time machine to specify a circuit. Let M be a clocked[3] 3-tape[4] Turing machine

[3] A clocked Turing machine running in time $c \log n$ is a machine that, on input (n, p)

1. computes $c \log n$ (which is just c times $|n|$).
2. starts a counter that will allow it to execute only $c \log n$ steps.

(Actually, steps (1) and (2) can be begun simultaneously; there are a number of programming tricks with Turing machines that one can use. The point is, (a) there is some purely syntactic part of the Turing machine description that we will call the "clock", (b) this "clock" enforces a run-time on the Turing machine, and (c) every Turing machine is equivalent to a "clocked" Turing machine of comparable complexity.)

[4] Note that Dlogtime-uniform AC^0 and NC^1 have circuits that are Dlogtime-uniform

running for $c \log n$ time for some c.

Let us say that M is *k-good on length* n if for all p of length $\leq k \log n$, $M(n,p)$ is a string in the set

$$\{(b, \text{ORACLE}), (2, \text{AND}), (2, \text{OR}), (1, \text{NOT}), (i, \text{INPUT})\}$$

where $|b| + |p| \leq k \log n$, and $i \leq n$, and such that if $|p| = k \log n$, then $M(n,p) = (i, \text{INPUT})$ for some i, and for each prefix q of p, if $M(n,q) = (j, \text{INPUT})$, then $i = j$. (Intuitively, p is an encoding of a path from the output gate to a gate g in the NC^1 circuit, and M, on input (n,p), is producing as output the fan-in of the gate g, and the type of gate that g is. The other conditions are merely for technical convenience. The depth of the circuit is $k \log n$.)

Note that – for a suitable encoding of clocked Turing machines – the following language is in uniform AC^0, for each c and k: $\{M, n : M$ is a clocked Turing machine running in $c \log n$ time, and M is k-good on length $n\}$. (This does depend somewhat on the encoding of Turing machines. However, note that for any reasonable encoding of Turings machines M, the language $\{1^M 0^n 1^p 0^i b :$ the i-th output bit of $M(n,p)$ is $b\}$ is in Dlogtime-uniform AC^0 (since it is in Dlogtime). Checking if a circuit is 2-good can be expressed as a first-order sentence over a uniform AC^0 predicate, and thus it is in Dlogtime-uniform AC^0.)

Define the circuit $C_{M,c,k,n}$ as follows: If M is a clocked Turing machine running in time $c \log n$ and M is k-good on length n, then this is the circuit with gates having labels of the form p, where the type and fan-in of gate p is given by $M(n,p)$. If gate p has fan-in b, then for all strings q of length $|b|$ that lexicographically precede b, the gates that are input to p are the gates pq. (If M is not a clocked Turing machine that is k-good on length n, then the circuit is a circuit that trivially accepts the empty set.)

We claim that the set $C = \{M, x : M$ is a clocked Turing machine running in $5 \log n$ time, and M is 2-good on length $|x|$, and the circuit $C_{M,c,k,|x|}$ accepts x (where the oracle gates give the middle bit[5] of the function DET applied to their inputs), and $|M| \leq \log \log |x|\}$ is complete for $\text{NC}^1(\#\text{L})$.

We need to show that C is in $\text{NC}^1(\#\text{L})$, and that it is hard. Neither seems completely trivial.

First, let's show that it's in $\text{NC}^1(\#\text{L})$. Let m be fixed. We'll describe the circuit accepting C for inputs of length m. First, given input M, x, where

even with this additional restriction that the uniformity machine have three tapes [BIS90].

[5] Any bit of the DETERMINANT can be reduced to the middle bit. (If we want the lower-order bit of $f(x)$, this is the middle-bit of some function GapL function $g(x)$ defined as $f(x)2^{n^k}$ for some k.)

$|x| = n$, the circuit will evaluate the AC^0 predicate to check that M runs for $5 \log n$ time and is 2-good on length $|x|$; thus let's assume that M is good, and let's concentrate on length n. For each string p, there will be circuitry evaluating $M(n, p)$. The output of our circuit accepting C, of course, is the value of gate λ in circuit $C_{M,c,k,n}$ on input x. (Recall that λ (the empty string) is the name of the output gate of $C_{M,c,k,n}$.) Here is the circuitry that will evaluate any given gate p. With NC^0 circuitry, (with "free" calls to the AC^0 predicates that are computed only once, given $M(n, p)$) we can compute the "type" of gate p. If the gate is of type INPUT, then a subcircuit of depth $\log n$ can compute the input bit i to which gate p is connected. If the gate is of any other type, then a subcircuit of depth $\log b$ can compute the fan-in b of gate p. (That is, near the "top" of this circuit, there will be gates checking if the fan-in is 2, and attempting to compute the AND of the inputs of gate p; near the bottom of the circuit, there will be gates checking if the fan-in is n^α, and attempting to compute DET(the matrix whose encoding is given by the n^α gates with labels of the form pq), etc.)[6] The total circuit depth required to evaluate a gate of fan-in d is $O(\log d)+$ (depth of its inputs). This is all that is required in order to show that this is in $\mathrm{NC}^1(\#\mathrm{L})$.

Now, let's show hardness.

For this, we need to show that anything accepted by $\mathrm{NC}^1(\#\mathrm{L})$ circuits is accepted by circuits of the form $C_{M,c,k,n}$ for some k-good Turing machine M, for some c and k. (If we have this, then a standard "padding" reduction will show that our set C is complete.) What we need is that a log-time machine, given a path p, can compute the type and fan-in of the gate that is

[6] It is routine but tedious to see that this circuit meets the uniformity requirements of [Ruz81]. In order to fill in the missing details, the interested reader will want to observe that this circuit has a *very* regular structure:

- Use OR gates to guess the type of the gate g.
- Use ANDs to
  - check that the guess is correct, and
  - simulate the gate.

  Simulating the gate involves first computing the fan-in, by, (a) using OR gates to guess the next bit of the fan-in b of g, and (b) using AND gates to check that this bit is correct. Once we have computed the fan-in, there is a gate that actually simulates the gate g of the original circuit, and then we repeat the process for the inputs to gate g.

Looking at a path name, it is not too hard to compute what type of gate one is at, and what the fan-in needs to be. We suppress the details.

reached by following path p from the output gate. (By "following a path p", we mean that at each oracle gate of fan-in b, $\log b$ bits are used to determine the input wire from the oracle gate that is followed.)

Let A be accepted by *logspace-uniform* $\mathrm{NC}^1(\#\mathrm{L})$ circuits C_n. Note that there is a *very* uniform $\mathrm{NC}^1(\#\mathrm{L})$ family of circuits recognizing the language $\{(n, p, t, i) : g$ is the gate reached by following path p in C_n, and either i is 0 and the gate has type t, or $i \leq \log b$ and t is the i-th bit of the binary representation of the fan-in of gate $g.\}$ That is, the #L oracle can be used to determine the name of the gate reached by following a given path, and to determine the output of the uniformity machine for C_n. Now it is not hard to use oracle gates of this form to build a new circuit family recognizing A, and having the property that the circuits are specified by a k-good machine M. The details are left to the interested reader. ∎