SIGACT News Complexity Theory Column 19

Lane A. Hemaspaandra
Dept. of Computer Science, University of Rochester
Rochester, NY 14627, USA lane@cs.rochester.edu

Introduction to Complexity Theory Column 19

This issue's expert guest column is by Eric Allender, who has just taken over the Structural Complexity Column in the *Bulletin of the EATCS*.

Regarding "Journals to Die For" (*SIGACT News* Complexity Theory Column 16), Joachim von zur Gathen, the editor-in-chief of *computational complexity*, has written me pointing out, quite correctly, that I cheated his journal of a point. His journal, in the issue checked, does include the email address of each author. I apologize for the missing point. Joachim also asked me to mention that, while the journal does not *require* the "alpha" citation style, it does strongly advise authors to use that style.

Guest Column: Making Computation Count: Arithmetic Circuits in the Nineties [1]

Eric Allender [2]

1 Introduction

There is a new direction in the study of arithmetic circuits, and it runs directly counter to the original direction. This short survey presents the main results and open questions of this new direction, and explains what motivates the new work.

First, let us explain the sense in which there has been a 180° change of direction. Traditionally, an arithmetic circuit is a directed acyclic graph with nodes (gates) computing arithmetic operations (such as $+, \Leftrightarrow, \times, \div$, etc.) over some algebraic structure R. Input gates receive elements of R, and each "wire" carries an element of R. If there are n inputs gates and one output gate, the circuit computes a (partial) function from R^n to R in the obvious way. (The more familiar Boolean circuits 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.) The field of arithmetic circuit complexity has a large literature, and it is not our attempt to survey the entire field in these few pages. The reader is referred to [vzG93, BM75, Bo82, vzG87, GS91] (among others) for this purpose. The salient fact that we require here 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

[2] Supported in part by NSF grant CCR-9509603. Department of Computer Science, Hill Center, Busch Campus, Rutgers University, Piscataway, NJ 08855 allender@cs.rutgers.edu

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. Since arithmetic circuits must "play by the rules", a number of strong lower bounds are known for various classes of arithmetic circuits (see the survey articles mentioned above, as well as [M94]). In contrast, we have painfully few lower bounds for general Boolean circuits.

Traditional arithmetic circuit complexity is still an active field. The upper and lower bounds produced there are interesting both theoretically as well as from the practical standpoint of symbolic manipulation packages. However, a tighter connection with (Boolean) computational complexity is possible if the model of arithmetic circuit is changed slightly. As we shall see, the resulting new arithmetic circuit model sheds new light on some well-known complexity classes, and helps clarify the complexity of some problems whose complexity was previously unknown.

2 The New Model

The desired change in the model is easy to describe. Instead of computing functions from R^n to R, we consider only functions from $\{0,1\}^n$ to R, where 0 and 1 are the additive and multiplicative identities of R, respectively.

Most of the upper bounds provided by traditional arithmetic circuit complexity carry over to the new model as well. On the other hand, by making this change in the model, we lose what had been one of the main points of interest in studying arithmetic circuits: This model of arithmetic circuits cannot be considered a "restriction" of Boolean circuits. Instead, as we shall see, in almost all cases we obtain a *more* powerful class of circuits. This is the sense in which this constitutes a complete change of direction. What we lose in the way of lower bounds is repaid by what we gain in the way of new insight into complexity classes. There is even some hope that this new model may help in the quest for new lower bounds for Boolean circuits.

2.1 Arithmetic complexity classes

We assume that the reader is familiar with the classes NP and NL (nondeterministic polynomial time and nondeterministic logspace, respectively). Probably #P is also familiar to the reader as the class of functions of the form $\#acc_M(x)$, counting the number of accepting paths of an NP machine M on input x [V79b]. The class #L is defined similarly, but for NL machines M [AJ93b].

There is another way of arriving at the classes #P and #L, starting with NP and NL, using circuits instead of Turing machines. Although NP and NL are usually defined in terms of nondeterministic Turing machines, these classes can also be defined in terms of uniform[3] Boolean circuits [V92]. If we now "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 $1 \Leftrightarrow x_i$), then we now obtain precisely #P and #L, when the arithmetic operations are defined on $\mathbb{N}$ [V92].

#P and #L are our first two "arithmetic complexity classes". There are three more arithmetic classes that will concern us, based on the Boolean circuit classes SAC^1, NC^1, and AC^0, respectively. Figure 2.1 summarizes the definitions of these classes, and lists some of the important complete problems motivating their study. Arithmetizing these Boolean classes provides us with the following

[3] A circuit family $\{C_n\}$ is said to be *uniform* if there is a deterministic linear-time Turing machine that, given n and the name of gates g, can determine all of the desired information about gate g (such as whether g is a $+$ gate or a $\times$ gate, what the gates are that feed into g, etc.) Detailed definitions can be found in [V92, BIS90, R81].

NP	Size $2^{n^{O(1)}}$, Degree $n^{O(1)}$ [V92]	SAT, etc.
SAC^1	Size $n^{O(1)}$, Degree $n^{O(1)}$	Context-free Languages [R80, S78, V91]
NL	Size $n^{O(1)}$, Skew[4][V92]	Shortest paths, Transitive Closure, etc.
NC^1	Depth $O(\log n)$, Bounded fan-in	Regular sets, Boolean Formulae [B93, B89]
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.

arithmetic complexity classes:

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

The remainder of this paper describes the complete problems that motivate interest in these classes, discusses some surprising recent discoveries, and points out open problems where progress can reasonably be expected.

3 Definitions

First, however, we need some more definitions.

In an arithmetic circuit with only + and × gates, each gate corresponds to a formal (multivariate) polynomial in the input variables. The *algebraic degree* (or simply, the *degree*) of a gate is the degree of this formal polynomial, if we don't take any cancellation into account. That is, input gates have degree 1, a + gate has degree equal to the maximum of the degrees of its inputs, and a × gate has degree equal to the sum of the degrees of its inputs. (Note that this is *quite* different from the indegree!)

#P is the class of functions computed by uniform arithmetic circuits over $\mathbb{N}$, having size $2^{n^{O(1)}}$ and polynomial degree. If we require that the size of the circuit be polynomial, maintaining the polynomial degree restriction, we obtain the class $\#\mathrm{SAC}^1$. Note that there are very many important circuit complexity classes that do *not* respect the polynomial degree restriction (such as AC^1, NC^2, P, ...). If we "arithmetize" these Boolean circuit classes, we obtain functions that are trivially not even in #P, since a circuit having superpolynomial degree can produce output that requires more than a polynomial number of bits even to write down. For that reason, the only "tractable" arithmetic circuit classes that we will consider are classes that are defined by circuits of AND and OR gates, defining subclasses of SAC^1.

Let $\mathcal{C}$ be a complexity class defined in terms of a class of Boolean circuits, and let R be an algebraic structure with operations + and ×, additive identity 0 and multiplicative identity 1. Define $\mathsf{Arith}(\mathcal{C}, \mathcal{R})$ to be the class of functions $f : \{0,1\}^* \to R$ such that there is a Boolean circuit family $\{C_n\}$ in $\mathcal{C}$ with the property that, for all strings x of length n, $f(x)$ is computed by the circuit C'_n that results by arithmetizing C_n.

Thus $\mathsf{Arith}(\mathrm{NP}, \mathbb{N}) = \#\mathrm{P}$ and $\mathsf{Arith}(\mathrm{NL}, \mathbb{N}) = \#\mathrm{L}$, and by definition $\mathsf{Arith}(\mathrm{AC}^0, \mathbb{N}) = \#\mathrm{AC}^0$, $\mathsf{Arith}(\mathrm{NC}^1, \mathbb{N}) = \#\mathrm{NC}^1$, and $\mathsf{Arith}(\mathrm{SAC}^1, \mathbb{N}) = \#\mathrm{SAC}^1$. Can we say anything about arithmetic circuits over the integers?

[4] 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.

It turns out that $\mathsf{Arith}(\mathrm{NP}, \mathbb{Z})$ has been studied quite a lot; it is equal to the class GapP studied in [FFK94] (see also [F97]); 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. By analogy to GapP, the other "Gap" classes have also been studied. For the purposes of this survey, we will consider these classes to be defined in terms of arithmetic circuits: $\mathsf{Arith}(\mathrm{NL}, \mathbb{Z}) = \mathrm{GapL}$, and by definition $\mathsf{Arith}(\mathrm{AC}^0, \mathbb{Z}) = \mathrm{GapAC}^0$, $\mathsf{Arith}(\mathrm{NC}^1, \mathbb{Z}) = \mathrm{GapNC}^1$, and $\mathsf{Arith}(\mathrm{SAC}^1, \mathbb{Z}) = \mathrm{GapSAC}^1$. It is actually still an open question if every function in GapAC^0 can be expressed as the difference of two $\#\mathrm{AC}^0$ functions (see [AAD97]), but all of the other "Gap" classes can be expressed as the difference of the corresponding "#" function classes.

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:

- $\mathrm{PP} = \{A : \exists f \in \mathrm{GapP}\ (x \in A) \Leftrightarrow f(x) > 0\}$
- $\mathrm{C_{=}P} = \{A : \exists f \in \mathrm{GapP}\ (x \in A) \Leftrightarrow f(x) = 0\}$
- $\mathrm{SPP} = \{A : \chi_A \in \mathrm{GapP}\}$

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, $\mathrm{C_{=}L}$, UL, PNC^1, $\mathrm{C_{=}NC}^1$, PAC^0 and $\mathrm{C_{=}AC}^0$ have all received study.

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.

3.1 Uniformity of Circuits

All circuit complexity classes come in two flavors: uniform and nonuniform. (Actually, there are infinitely many different flavors of "uniform", but that need not concern us here. The reader is referred to [BIS90, R81] for more background on uniformity.) Some of the most interesting results about arithmetic circuits are known only to hold in the setting of nonuniform circuit complexity – although some people believe that they should hold also in the uniform setting.

The reader should note that, using the definitions of NP and #P in terms of uniform circuits, the "nonuniform" versions of NP and #P are not very interesting; they consist of the sets of *all* Boolean functions and *all* functions mapping $\{0,1\}^n$ to $\{y \in \mathbb{N} : y \leq 2^{n^{O(1)}}\}$, respectively. However the nonuniform versions of all of the other complexity classes mentioned above are interesting from a purely combinatorial standpoint.

4 Can Arithmetic Circuits Really Simulate Boolean Circuits?

It is time to back up the assertion, made earlier, that arithmetic circuits (using the "new model") are more powerful than Boolean circuits.

In some cases this is trivial to prove, in other cases it follows as a consequence of some new and 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 $\mathsf{Arith}(\mathrm{NP}, \mathbb{N})$ $(= \#\mathrm{P})$, then NP = UP. (In fact, these are equivalent statements.) Even the weaker hypothesis that these characteristic functions are in $\mathsf{Arith}(\mathrm{NP}, \mathbb{Z})$ $(= \mathrm{GapP})$ is viewed as rather unlikely. (This is equivalent to NP $\subseteq$ SPP.) Of course, in the nonuniform setting, both of these inclusions are trivially true, for the uninteresting reason mentioned above.

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 [CMTV96, AAD97], and thus for these classes, arithmetic circuits over $\mathbb{N}$ are easily seen to be *at least* as powerful as Boolean circuits. 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.

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 recently shown that NL/poly = UL/poly [RA97], it is no longer clear if this should be considered unlikely. In fact, the results of [RA97] show that, in the nonuniform 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 $\#\mathrm{SAC}^1$.

4.1 Arithmetic-Boolean Circuits

Another model that has received extensive study is the model of *Arithmetic-Boolean Circuits* of von zur Gathen. (For instance, see [vzG93, vzG87, GS91, 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:

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

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

By analogy to the definition of $\mathsf{Arith}(\mathcal{C}, \mathcal{R})$, one can define classes $\mathsf{Arith.Bool}(\mathcal{C}, \mathcal{R})$. (For instance, the class we would denote by $\mathsf{Arith.Bool}(\mathrm{SAC}^1, \mathbb{N})$ is denoted by $\mathrm{SAC}^1_{\mathbb{N}}$ in [vzG93].)

It is easy to verify that

$$\mathsf{Arith.Bool}(\mathrm{AC}^0, \mathbb{N}) = \mathsf{Arith}(\mathrm{AC}^0, \mathbb{N}),$$

and

$$\mathsf{Arith.Bool}(\mathrm{NC}^1, \mathbb{N}) = \mathsf{Arith}(\mathrm{NC}^1, \mathbb{N}).$$

One consequence of [RA97] is that, in the nonuniform setting,

$$\mathsf{Arith.Bool}(\mathrm{SAC}^1, \mathbb{N}) = \mathsf{Arith}(\mathrm{SAC}^1, \mathbb{N}),$$

and

$$\mathsf{Arith.Bool}(\mathrm{NL}, \mathbb{N}) = \mathsf{Arith}(\mathrm{NL}, \mathbb{N}).$$

In contrast, we shall see in Section 8 that $\mathsf{Arith.Bool}(\mathrm{AC}^0, \mathbb{Z})$ coincides with the functions computable by constant-depth threshold circuits (TC^0), and thus is strictly more powerful than $\mathsf{Arith}(\mathrm{AC}^0, \mathbb{Z})$ ($= \mathrm{GapAC}^0$).

It is not hard to see that the condition $\mathsf{Arith.Bool}(\mathrm{NL}, \mathbb{Z}) = \mathsf{Arith}(\mathrm{NL}, \mathbb{Z})$ is equivalent to $\mathrm{C_=L} = \mathrm{SPL}$. (This holds also for general classes $\mathcal{C}$.) We have more to say about SPL in Section 6. 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.

5 #SAC^1

One of the most important facts about arithmetic circuits of polynomial size and degree is that they are equivalent to circuits of depth $O(\log n)$, where the $\times$ gates have fan-in two, and $+$ gates have unbounded fan-in. (These are called semi-unbounded fan-in circuits.) This was first proved in the nonuniform setting in [VSBR83], and related depth-reduction results for uniform circuits were proved for the Boolean ring in [R81], and for $\mathbb{N}$ in [V91]. A general proof that works in the uniform setting over any commutative semiring appears in [AJMV].

All of the functions in #SAC^1 can be computed by threshold circuits of logarithmic depth (known as TC^1 circuits). (In fact, it is observed in [AJMV] that if gates for integer division (throwing the remainder away) are added to #SAC^1 circuits, then one obtains an exact characterization of TC^1.) On the other hand, nothing is known about the relative power of #SAC^1 and AC^1 (the class of problems accepted by logarithmic depth, unbounded-fan-in circuits of AND and OR gates). It is not even known if the functions in #NC^1 have AC^1 circuits. A possible first step toward answering this question is taken in [AJMV], 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 $n^{O(\log n)}$.

There are some very nontrivial relationships between #SAC^1 and the related machine model (auxiliary pushdown machines) and classes of circuits. The reader is referred to [NR95, LR90, V91] for further information.

6 #L

#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 recent paper by Mahajan and Vinay [MV97] gives a new and beautiful proof of this theorem, and 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 [C85]. 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 are 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 =

$$\mathrm{C_{=}L}^{\mathrm{C_{=}L}^{\mathrm{C_{=}L}^{\cdots}}} = \mathrm{AC}^0(\mathrm{C_{=}L})$$

  = 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.

- $\mathrm{AC}^0(\mathrm{C_{=}L}) = \mathrm{L}^{\mathrm{C_{=}L}} = \mathrm{NC}^1(\mathrm{C_{=}L})$ [ABO96].
- $\mathrm{AC}^0(\mathrm{PL}) = \mathrm{PL} = \mathrm{NC}^1(\mathrm{PL})$ [O96, BF97].

(These hierarchies are defined using "Ruzzo-Simon-Tompa" reducibility [RST], 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.[5]

One of the first papers to explicitly study GapL in terms of arithmetic circuits was [T92]. One of the contributions of [T92] 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 [RA97], seem to be sufficient to prove any sort of collapse of the #L hierarchy.

One of the main reasons to be interested in these 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 $\mathrm{C_{=}L}$, and a variety of other problems regarding computation of the rank and determining if a system of linear equations is feasible are complete for $\mathrm{L}^{\mathrm{C_{=}L}}$ [ABO96]. Some other problems in linear algebra and problems involving Markov decision processes were shown to be complete for PL in [J84, MGA97].

One important problem whose complexity remains unresolved is the perfect matching problem. No deterministic NC algorithm is known for matching at all, but the probabilistic NC algorithm of [MVV87] can be used to show that, in the nonuniform setting, perfect matching is in $\mathrm{coC_{=}L}$ and in $\oplus\mathrm{L}$. (See [ABO96, BGW96] for details.) Recently, by combining use of the "isolating lemma" of [MVV87] with the new algorithm for the determinant of [MV97], it has been shown that the perfect matching problem is in SPL [AR97], which improves the previous upper bounds. Since the matching problem is hard for NL (see [ABO96, KUW86]), this "sandwiches" the complexity of this problem between NL and SPL (at least in the nonuniform 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 SPP = $\{A : \mathrm{GapP}^A = \mathrm{GapP}\}$), one can show that SPL = $\{A : \mathrm{GapL}^A = \mathrm{GapL}\}$. Are there other interesting problems that lie in this class?

[5] This is stated without proof in [AO96]. Recently, Mahajan (personal communication) pointed out that this actually requires some proof. More information is available in [A97].

7 $\#\mathrm{NC}^1$

$\#\mathrm{NC}^1$ 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 [V79a]. Evaluating arithmetic formulae over $\mathbb{N}$ ($\mathbb{Z}$) is complete for $\#NC^1$ (GapNC^1, respectively) [BCGR92].

Probably the most important and fascinating open question regarding $\#\mathrm{NC}^1$ is the question of whether or not it is identical to the class of functions having Boolean NC^1 circuits. 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 7.1 *[J85] Let $f \in$ 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 [CMTV96] that the techniques of Ben-Or and Cleve [BC92] 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 $D(n) = O(\log n \log^* n)$.

(Note that these circuits are P-uniform. It is not known if they can be made more uniform.) □

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. On the other hand, much of the practical value of complexity theory derives from the fact that complexity classes are useful. They are our main tool for understanding the complexity of real-world problems. The fundamental

objects of study in complexity theory are, therefore, complexity classes – and both NC^1 and $\#NC^1$ are natural and important complexity classes.

I would *love* to know if the $\log^* n$ factor can be removed!

Even though there seems to be essentially no room "between" Boolean NC^1 and $\#NC^1$, 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 [CMTV96] that all of these languages are in PNC^1, and that there are some such languages that are complete for this class. From [M97], it now follows that PNC^1 is contained in deterministic logspace. (Macarie's paper [M97] 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 [LZ76] that this problem is in L. It was shown by Robinson that it is hard for NC^1. It was observed in [CMTV96] that the problem is in $C_=NC^1$. Thus it is sandwiched between NC^1 and $C_=NC^1$. 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 [B89], characterizing NC^1 in terms of width-5 branching programs. It is natural to wonder if this characterization also gives an equivalent characterization of $\#NC^1$. The authors of [CMTV96] 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 $\#NC^1$, 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 [CMTV96] did show (using the techniques of Ben-Or and Cleve [BC92]) 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 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:

- What about 2×2 integer matrices? (This problem is hard for Boolean NC^1 by [R93], but not known to be hard for $\#NC^1$.)
- What about $k \times k$ matrices over $\mathbb{N}$? (It is shown in [CMTV96] that for 6×6 matrices, this is hard 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 $\#NC^1$.)
- It has been shown by Barrington [B97] that even multiplying 2×2 matrices over $\mathbb{N}$ is hard for NC^1 under ACC^0-Turing reducibility. Can this be improved?

8 $\#AC^0$

The main reason to be interested in $\#\mathrm{AC}^0$ is because it may offer an avenue toward lower bounds for 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 [AAD97] that the zero-one-valued functions in GapAC^0 are exactly the languages in $\mathrm{AC}^0[2]$ (that is, the languages accepted by constant-depth polynomial-size circuits of AND, OR, and PARITY gates). The results of [R87, S87] show that there are many simple languages (such as the Mod 3 function) that are not in $\mathrm{AC}^0[2]$, and thus current lower bound techniques apply to these very small arithmetic circuit classes.

However, the main result of [AAD97] is that TC^0 is exactly $\mathrm{C}_{=}\mathrm{AC}^0$ (at least in the non-uniform and logspace-uniform settings). (This then establishes the assertion made back in Section 4.1, that $\mathsf{Arith.Bool}(\mathrm{AC}^0, \mathbb{Z})$ coincides with the functions computable by constant-depth threshold circuits.) Thus, if we could expand our repertoire of lower bounds for GapL, 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 NP is not contained in TC^0 [RR94]. 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 GapL leads us outside the framework of "natural proofs" as considered in [RR94].

There are a great many more open questions involving $\#\mathrm{AC}^0$ discussed in [AAD97] – but for the sake of brevity I will refer the reader thither.

9 Conclusions

I began this survey by apologizing for not mentioning the great bulk of work on arithmetic circuits, and concentrating only on the small part of the field that seems to be heading in a completely different direction: where arithmetic circuits are *not* a "restricted" model of computation, but are in fact *more* powerful than Boolean circuits. Now, I must apologize for not even covering all of *that* field. The paper [AJMV] contains a number of results about arithmetic circuits over some *noncommutative* semirings (in particular $(\Sigma^*$,max,concat)) that characterize the optimization classes OptL and related classes [AJ92, V91]. Further references can be found there. There is also a wealth of interesting material about arithmetic circuits over finite fields that fits very nicely into the framework described here. As an example, I mention [GW96], as well as the work in [AAD97] characterizing ACC^0 in terms of arithmetic circuits over finite fields.

I'm sure that, in my rush to finish this by deadline, I have left out other important material.

The classes defined and discussed here help classify the complexity of real-world computational problems. They help reveal the algebraic structure underlying some important complexity classes. It seems possible that a better understanding of this structure could lead to insights that eventually might provide lower bounds for Boolean circuits. We shall see.

At the very least, there have been some fun surprises thus far.

Acknowledgments: This survey has been enhanced thanks to many conversations with colleagues, such as M. Agrawal, V. Arvind, D. Mix Barrington, K.-J. Lange, M. Mahajan, S. Buss, P.

McKenzie, R. Niedermeier, K. Reinhardt, D. Thérien, and V. Vinay.

References

[A97] E. Allender. A comment on the logspace counting hierarchy. `http://www.cs.rutgers.edu/ allender/publications/lh.html`

[AAD97] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and Arithmetic Circuits. In *Proc. 12th Annual IEEE Conference on Computational Complexity*, 1997, pp. 134–148.

[ABO96] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. DIMACS Technical report 97-40 (submitted for publication). A preliminary version appears in *ACM Symposium on Theory of Computing (STOC)*, 1996.

[AJ93a] E. Allender and J. Jiao. Depth reduction for noncommutative arithmetic circuits. In *Proc. 25th ACM Symposium on Theory of Computing (STOC)*, pages 515–522, 1993.

[AJ92] C. Álvarez and B. Jenner. A note on log space optimization. *Computational Complexity* 5 (1995) 155–166.

[AJ93b] C. Álvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107:3–30, 1993.

[AJMV] E. Allender, J. Jiao, M. Mahajan, and V. Vinay. Non-commutative arithmetic circuits: depth reduction and size lower bounds. To appear in *Theoretical Computer Science*. Preliminary versions appeared as [AJ93a, MV94].

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

[AR97] E. Allender and K. Reinhardt. Work in preparation, 1997.

[B89] D.A. Barrington. Bounded-width polynomial size branching programs recognize exactly those languages in NC^1, *Journal of Computer and System Sciences* 38 (1989), 150–164.

[B93] S. Buss. Algorithm for Boolean formula evaluation and for tree contraction. In *Arithmetic, Proof Theory, and Computational Complexity* (P. Clote and J. Krajíček, ed), Oxford Logic Guides, Vol. 23, 1993, pp. 96–115.

[B97] D. A. Mix Barrington, Personal Communication, 1997.

[BC92] M. Ben-Or and R. Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.* 21 (1992) 54–58.

[BCGR92] S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.* 21 (1992), 755–780.

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

[BF97] R. Beigel and B. Fu. Circuits over PP and PL. In *IEEE Conference on Computational Complexity*, pages 24–35, 1997.

[BGW96] L. Babai, A. Gál, and A. Wigderson. Superpolynomial lower bounds for monotone span programs. DIMACS Technical Report 96-37. (Submitted for publication.)

[BIS90] D. A. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

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

[Bo82] A. Borodin. Structured versus general models in computational complexity. In *Logic and Algorithms*, Symposium in honour of Ernst Specker. *L'Enseignement Mathematique* 30 (1982) 47–65.

[C85] S. Cook. A taxonomy of problems with a fast parallel algorithms. *Information and Computation* 64 (1985), 2–22.

[CMTV96] H. Caussinus, P. McKenzie, D. Thérien, and H. Vollmer. Nondeterministic NC^1 computation. In *Proceedings, 11th Annual IEEE Conference on Computational Complexity*, pages 12–21, 1996.

[F97] L. Fortnow. Counting Complexity. In *Complexity Theory Retrospective II*, (L. Hemaspaandra and A. Selman, ed), Springer-Verlag, New York, 1997.

[FFK94] Stephen A. Fenner, Lance J. Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, February 1994.

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

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

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

[GS91] J. von zur Gathen and G. Seroussi. Boolean circuits versus arithmetic circuits. *Information and Computation* 91 (1991) 142–154.

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

[J84] H. Jung. On probabilistic tape complexity and fast circuits for matrix inversion problems. In *Proc. ICALP '84, Lecture Notes in Computer Science* 172, pp. 281–291, 1984.

[J85] H. Jung. Depth efficient transformations of arithmetic into Boolean circuits. In *Proc. FCT '85, Lecture Notes in Computer Science* 199, pp. 167-173.

[LZ76] R. Lipton and Y. Zalcstein. Word problems solvable in logspace. *J. ACM* 24 (1977) 522–526.

[M94] I. Macarie. Space-efficient deterministic simulation of probabilistic automata. In *11th Symposium on Theoretical Aspects of Computing (STACS)*, volume 775 of *Lecture Notes in Computer Science*, pages 109–122. Springer-Verlag, 1994.

[M97] I. Macarie. Space-efficient deterministic simulation of probabilistic automata. To appear in *SIAM J. Comput.* Preliminary version appeared as [M94].

[MGA97] M. Mundhenk, J. Goldsmith, and E. Allender. The complexity of policy evaluation for finite-horizon partially-observable Markov decision processes, In *Proc. 25th International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science* 1295, pp. 129–138, 1997.

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

[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*, Lecture Notes in Computer Science 452, 1990, 399–406.

[M94] K. Mulmuley. Lower bounds for parallel linear programming and other problems. In *Proceedings, 26th ACM Symposium on Theory of Computing*, pages 603–614, 1994.

[MV97] M. Mahajan and V. Vinay. Determinant: Combinatorics, Algorithms, and Complexity. To appear in *Chicago Journal of Theoretical Computer Science* (`http://www.cs.uchicago.edu/publications/cjtcs/`). A preliminary version appeared as: A combinatorial algorithm for the determinant. In *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, ACM Press, 1997, pp. 730–738.

[MV94] M. Mahajan and V. Vinay. Non-commutative computation, depth reduction and skew circuits. In *Proc. 14th FST&TCS*, volume 880 of *Lecture Notes in Computer Science*, pages 48–59. Springer-Verlag, 1994.

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

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

[NR97] M. Naor and O. Reingold. Number-Theoretic constructions of efficient pseudo-random functions. To appear in: *Proc. 38th IEEE Symp. on Foundations of Computer Science*, 1997.

[O96] M. Ogihara. The PL hierarchy collapses. In *ACM Symposium on Theory of Computing (STOC)*, pages 84–88, 1996.

[R93] D. Robinson. Parallel algorithms for group word problems. Doctoral Dissertation, Mathematics Dept., University of California, San Diego, 1993.

[RA97] K. Reinhardt and E. Allender. Making nondeterminism unambiguous. To appear in *Proc. 38th IEEE Conference on Foundations of Computer Science (FOCS)*, 1997.

[R87] 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.

[RR94] A. A. Razborov and S. Rudich. Natural proofs. In *Proceedings, 26th ACM Symposium on Theory of Computing*, pages 204–213, 1994.

[R80] W. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21 (1980) 218-235.

[R81] W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 21:365–383, 1981.

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

[S78] I. H. Sudborough. On the tape complexity of deterministic context-free languages. *J. Association of Computing Machinery*, 25:405–414, 1978.

[S87] 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.

[T92] S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Trans. Inf. and Syst.*, E75-D:116–124, 1992.

[V79a] L. Valiant. Completeness classes in algebra. In *Proc. 11th ACM Symposium on Theory of Computing (STOC)*, pages 249–261, 1979.

[V79b] L. Valiant. The complexity of computing the Permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[V91] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43:380–404, 1991.

[V92] H. Venkateswaran. Circuit definitions of nondeterministic complexity classes. *SIAM Journal on Computing*, 21:655–670, 1992.

[V91] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proc. 6th IEEE Structure in Complexity Theory Conference*, pages 270–284, 1991.

[VSBR83] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM J. Comput.* 12 (1983) 641–644.