DIMACS Technical Report 97-48
September 1997

On TC^0, AC^0, and Arithmetic Circuits[1]

by

Manindra Agrawal[2]
Department of Computer Science
Indian Institute of Technology
Kanpur 208016, India
manindra@iitk.ernet.in

Eric Allender[3,4]
Department of Computer Science
Rutgers University
Piscataway, NJ 08855, USA
allender@cs.rutgers.edu

Samir Datta[5,6]
Department of Computer Science
Rutgers University
Piscataway, NJ 08855, USA
sdatta@paul.rutgers.edu

[1]A preliminary version of this work appeared in the Proc. IEEE Conference on Computataional Complexity, 1997, pp. 134–148.
[2]Part of this research was done while visiting the University of Ulm under an Alexander von Humboldt Fellowship.
[3]Permanent Member
[4]Supported in part by NSF grant CCR-9509603.Portions of the work were performed while this author was a visiting scholar at the Institute of Mathematical Sciences, Chennai, India
[5]Affiliated Graduate Student Member
[6]Supported in part by a Rutgers University Graduate Excellence Fellowship and by NSF grant CCR-9509603.

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs, Bellcore, and Bell Labs.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

Continuing a line of investigation that has studied the function classes #P [Val79b], $\#\mathrm{SAC}^1$ [Val79a, Vin91, AJMV], #L [AJ93b, Vin91, AO94], and $\#\mathrm{NC}^1$ [CMTV96], we study the class of functions $\#\mathrm{AC}^0$. One way to define $\#\mathrm{AC}^0$ is as the class of functions computed by constant-depth polynomial-size arithmetic circuits of unbounded fan-in addition and multiplication gates. In contrast to the preceding function classes, for which we know no nontrivial lower bounds, lower bounds for $\#\mathrm{AC}^0$ follow easily from established circuit lower bounds.

One of our main results is a characterization of TC^0 in terms of $\#\mathrm{AC}^0$: A language A is in TC^0 if and only if there is a $\#\mathrm{AC}^0$ function f and a number k such that $x \in A \iff f(x) = 2^{|x|^k}$. Using the naming conventions of [FFK94, CMTV96], this yields:

$$\mathrm{TC}^0 = \mathrm{PAC}^0 = \mathrm{C}_=\mathrm{AC}^0.$$

Another restatement of this characterization is that TC^0 can be simulated by constant-depth arithmetic circuits, with a single threshold gate. We hope that perhaps this characterization of TC^0 in terms of AC^0 circuits might provide a new avenue of attack for proving lower bounds.

Our characterization differs markedly from earlier characterizations of TC^0 in terms of arithmetic circuits over finite fields [RT92, BFS92]. Using our model of arithmetic circuits, computation over finite fields yields ACC^0.

We also prove a number of closure properties and normal forms for $\#\mathrm{AC}^0$.

1 Introduction

The circuit complexity class AC^0 is well-studied and fairly well-understood. Many lower bound techniques have been developed, showing that exponential size is required in order to compute many simple functions on AC^0 circuits. In contrast, the circuit complexity class TC^0 is only poorly understood, in spite of having been the object of many investigations. (The class TC^0 is of special interest in computer science, since it characterizes the computational complexity of such important operations as multiplication, division, and sorting, as well as being a computational model for neural nets [RT92, CSV84, PS88].) It remains an open question as to whether every function in #P has TC^0 circuits (although it is at least known that not all #P functions have *Dlogtime-uniform* TC^0 circuits [All]). The main contribution of this paper is to present a new connection between AC^0 and TC^0. We characterize TC^0 as being the class of languages that arises in several ways from counting the number of accepting subtrees of AC^0 circuits. Equivalently, we characterize TC^0 in terms of constant-depth arithmetic circuits.

In order to make these notions precise, we need to discuss counting and enumeration classes.

1.1 Counting Classes

Certainly the best-known counting class is Valiant's class #P [Val79b], consisting of functions that map x to the number of accepting computations of an NP-machine on input x. Recently, the class #L (counting accepting computations of an NL-machine) has also received considerable attention [AJ93b, Vin91, Tod, MV]. #P characterizes the complexity of computing the permanent of a matrix [Val79b], while #L characterizes the complexity of computing the determinant [Vin91, Tod, Val92, MV].

It should be noted that #P and #L can also be characterized in terms of uniform arithmetic circuits, as follows: NP and NL both have characterizations in terms of uniform Boolean circuits. (NP sets are accepted by uniform exponential-size circuits of "polynomial algebraic degree," and NL sets are accepted by uniform polynomial-size "skew" circuits [Ven92]. We will not need to define these concepts further here.) The classes #P and #L result if we "arithmetize" these Boolean circuits, replacing each OR gate by a $+$ gate, and replacing each AND gate by a $\times$ gate, where the input variables $x_1, \ldots, x_n$ now take as values the natural numbers $\{0, 1\}$ (instead of the Boolean values $\{0, 1\}$), and negated input literals $\overline{x_i}$ now take on the value $1 - x_i$. Alternatively, #P and #L arise by counting the number of "accepting subtrees" for the corresponding classes of Boolean circuits. (See [Ven92] for a formal definition of this notion; for our purposes it is sufficient to know that the number of accepting subtrees of a circuit C is (a) equal to the output of the "arithmetized" version of C, (which we denote by $\#C$) and (b) provides a natural notion of counting the number of proofs that C accepts.) The arithmetic circuits corresponding to #L were studied further by Toda [Tod92].

The counting classes that result in this way by arithmetizing the Boolean circuit classes

SAC^1 and NC^1 were studied in [Vin91] (where it was shown that $\#SAC^1$ corresponds to counting the accepting paths of a NAuxPDA) and in [CMTV96] (where it was shown that $\#NC^1$ is closely-related to counting paths in bounded-width branching programs). In this paper, we study $\#AC^0$.

Definition 1 *For any $k > 0$, ${\#AC^0}_k$ is the class of functions computed by depth k circuits with $+, *$-gates (the usual arithmetic sum and product) having unbounded fan-in where inputs to the circuits are from $\{0, 1, x_i, 1 - x_i\}$ where each $x_i \in \{0, 1\}$. Let $\#AC^0 = \bigcup_{k>0} {\#AC^0}_k$.*[1]

Why study $\#AC^0$? Our motivation comes in large part from a desire to obtain more lower bounds in circuit complexity. As we shall see, $\#AC^0$ straddles the boundary marking the limits of current circuit lower bound technology. $\#AC^0$ provides a characterization of TC^0 (for which no circuit lower bounds are known), but on the other hand $\#AC^0$ is closely related to the classes AC^0 and $AC^0[2]$, and as a consequence we can prove that many simple functions are not in $\#AC^0$. (This stands in contrast to the related classes $\#NC^1$, $\#L$, $\#SAC^1$, and $\#P$ which, for all we know, may contain all of the functions in P^{NP}.) We can also show that ${\#AC^0}_k$ is properly contained in ${\#AC^0}_{k+1}$ for every k. A better understanding of $\#AC^0$ should aid in advancing our store of lower bound techniques.

1.2 Language Classes

Counting classes such as $\#P$ and $\#L$ are closely related to associated language classes such as PP and PL. In order to develop this in a general setting, it is useful to define the "Gap" classes.

The class GapP was defined in [FFK94], and by analogy GapL was studied in [Vin91, AO94], and $GapNC^1$ was studied in [CMTV96]. In all of these cases, there are two equivalent definitions:

1. Gap$\mathcal{C}$ is the class of functions that are the difference of two $\#\mathcal{C}$ functions.

2. Gap$\mathcal{C}$ is the class of functions computed by the class of arithmetic circuits that characterize $\#\mathcal{C}$, when these circuits are augmented by having the constant -1.

(In fact, for the cases when $\mathcal{C}$ is one of NC^1, L, and P, the cited papers give many other equivalent definitions, as well.)

Now, for a given class $\mathcal{C}$, Gap$\mathcal{C}$ gives rise to two language classes:

$$\begin{aligned} \mathrm{P}\mathcal{C} &= \{A \mid \exists f \in \mathrm{Gap}\mathcal{C},\ x \in A \iff f(x) > 0\}, \\ \mathrm{C}_{=}\mathcal{C} &= \{A \mid \exists f \in \mathrm{Gap}\mathcal{C},\ x \in A \iff f(x) = 0\}. \end{aligned}$$

[1] Tomo Yamakami [Yam96] has recently defined $\#AC^0$ somewhat differently, and his definition does not appear comparable to ours.

PP and PL were first studied in [Gil77] and have been considered in many papers; $C_{=}P$ was studied in [Wag86] and elsewhere, and $C_{=}L$ was studied in [ABO96] (see also [ST]). PNC^1 and $C_{=}NC^1$ were defined and studied in [CMTV96] (see also [Mac]).

A main result of this paper is that PAC^0 and $C_{=}AC^0$ coincide with TC^0. However, there are two difficulties that must be overcome before we can even state this theorem. We must deal with (a) uniformity, and (b) the fact that the two most natural ways to define $GapAC^0$ do not seem to be equivalent (although both ways give rise to the *same* class $PAC^0 = C_{=}AC^0 = TC^0$).

Definition 2 $DiffAC^0$ *is the class of functions expressible as the difference of two* $\#AC^0$ *functions.*

Definition 3 *For any* $k > 0$, ${GapAC^0}_k$ *is the class of functions computed by depth* k *circuits with* $+, *$*-gates (the usual arithmetic sum and product) having unbounded fan-in where inputs to the circuits are from* $\{0, 1, -1, x_i, 1 - x_i\}$ *where each* $x_i \in \{0, 1\}$*. Let* $GapAC^0 = \bigcup_{k>0} {GapAC^0}_k$.

Recall that for all the classes $\mathcal{C} \in \{NC^1, L, SAC^1, P\}$, Gap$\mathcal{C}$ can be defined equivalently either as $\#\mathcal{C} - \#\mathcal{C}$ or in terms of arithmetic circuits with access to the constant -1. However, in all of those cases, the proof of equivalence relies on the fact that the PARITY language is in $\mathcal{C}$; and of course this is not true for $\mathcal{C} = AC^0$.

Open Question 1 *Is* $DiffAC^0 = GapAC^0$*?*

Open Question 2 *Is* $(-1)^{\Sigma x_i} \in DiffAC^0$*?*

(Note that $DiffAC^0$ is clearly contained in $GapAC^0$, and that $(-1)^{\Sigma x_i} = \prod(1 - 2x_i) \in GapAC^0$).

The classes $DiffAC^0$ and $GapAC^0$ each provide reasonable ways to define PAC^0 and $C_{=}AC^0$. This leads to the following two definitions:

Definition 4 *The class* $C_{=}AC^0$ *(*$C_{=}AC^0_{circ}$*) consists of those languages* L *for which there exists a function* f *in* $DiffAC^0$ *(*$GapAC^0$*) such that for all bit strings* x*,*

- *If* $x \in L$ *then* $f(x) = 0$.
- *If* $x \notin L$ *then* $f(x) \neq 0$.

Definition 5 *The class* PAC^0 *(*PAC^0_{circ}*) consists of those languages* L *for which there exists a function* f *in* $DiffAC^0$ *(*$GapAC^0$*) such that for all bit strings* x*,*

- *If* $x \in L$ *then* $f(x) > 0$
- *If* $x \notin L$ *then* $f(x) \leq 0$

At this point, the reader may fear that we are introducing too many complexity classes, with relatively little motivation. The good news is that all of these classes are different names for TC^0—at least in the P-uniform and non-uniform settings.

1.2.1 Uniformity

A (non-uniform) circuit family $\{C_n\}$ consists of a circuit C_n for each input length n. If there is an "efficient" algorithm for constructing C_n, given n, then the family is said to be *uniform*, where different notions of "efficient" give rise to different notions of uniformity. We will consider P-uniform, Logspace-uniform, and Dlogtime-uniform circuit families. For P-uniform circuits [BCH86, All89], the mapping $n \mapsto C_n$ is computable in polynomial time, for Logspace-uniform circuits [Ruz81], the mapping is computable in Logspace. Dlogtime-uniformity requires a somewhat more careful definition; we refer the reader to [BIS90]. Although Dlogtime-uniformity is widely-regarded as being the "right" notion of uniformity to use when discussing small circuit complexity classes such as TC^0 and AC^0, only a few of our theorems mention Dlogtime-uniformity.

Open Question 3 *Can the characterizations of* TC^0 *that we present in the P-uniform and Logspace-uniform setting also be shown to hold in the Dlogtime-uniform setting?*

1.3 The Characterizations

In the P-Uniform and Non-Uniform Settings

$$\mathrm{C_{=}AC}^0 = \mathrm{PAC}^0 = \mathrm{TC}^0 = \mathrm{C_{=}AC}^0_{circ} = \mathrm{PAC}^0_{circ}.$$

In the Logspace-Uniform Setting

$$\mathrm{C_{=}AC}^0 \subseteq \mathrm{PAC}^0 \subseteq \mathrm{TC}^0 = \mathrm{C_{=}AC}^0_{circ} = \mathrm{PAC}^0_{circ}.$$

In the Dlogtime-Uniform Setting

$$\mathrm{TC}^0 \subseteq \mathrm{C_{=}AC}^0_{circ} \subseteq \mathrm{PAC}^0_{circ}.$$

We are also able to prove some normal form theorems for DiffAC^0. This allows us to present technically sharper characterizations. For instance, for any set A in (non-uniform or P-uniform) TC^0, there exist a constant l, a function $g(n)$, and a $\#\mathrm{AC}^0$ function h with the following property:

- If $x \in A$, then $h(x) = 2^{|x|^l}$.
- If $x \notin A$, then $h(x) = 2^{|x|^l} + g(|x|)$.

(The function g is computable in Dlogtime-uniform $\#\mathrm{AC}^0$. The important point is that g depends only on $|x|$, and not on x itself.) Thus membership in a TC^0 set is determined in a very precise way by the number of accepting subtrees of a Boolean AC^0 circuit.

It seems reasonable to conjecture that all of these classes collapse to TC^0 in the Logspace-uniform and Dlogtime-uniform settings, just as in the P-uniform and non-uniform cases. In the mean time, however, even some very basic properties of the classes $\mathrm{C_{=}AC}^0$ and PAC^0 remain to be established. We list a few such questions below:

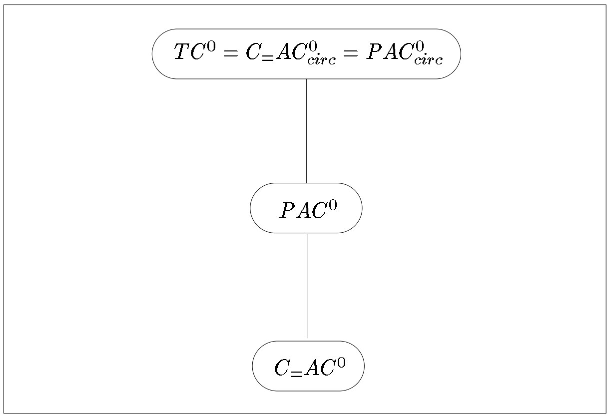

Figure 1: Logspace-uniform setting

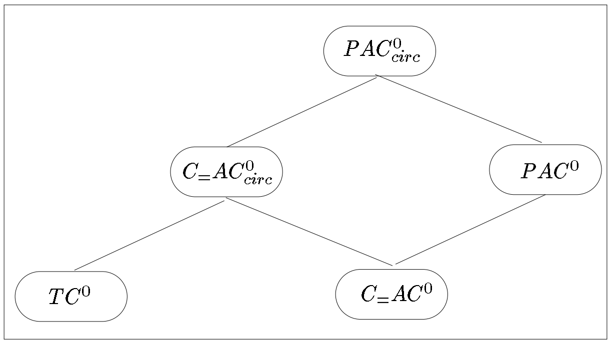

Figure 2: DLogtime-uniform setting

Open Question 4 *Is* $C_=AC^0$ *(*$C_=AC^0_{circ}$*) closed under complement in the Logspace-uniform (resp. Dlogtime-uniform) setting?*

Open Question 5 *Is* PAC^0 *closed under union or intersection in the Logspace-uniform or Dlogtime-uniform setting?*

1.4 Organization

In Section 2 we take care of some preliminary matters. In Section 3 we establish some closure properties and normal forms for the classes of functions we study. In Section 4 we prove our main results, characterizing the circuit complexity class TC^0. In Section 5 we briefly consider arithmetic circuits over finite fields. In Section 6 we present some concluding remarks.

2 Definitions and Notation

In this section we establish the notation and conventions used in the rest of the paper.

Definition 6 AC^0 *(*TC^0*) is the class of languages accepted by constant-depth circuits of unbounded fan-in AND, OR, and NOT gates (MAJORITY gates, respectively).*

It will be convenient to distinguish between *languages* (accepted by Boolean circuits with a single output gate) and *functions* computed by Boolean circuits with possibly several output gates.

Definition 7 FAC^0 *(*FTC^0*) is the class of functions computed by constant-depth circuits of unbounded fan-in AND, OR, and NOT gates (MAJORITY gates, respectively).*

Notice, that depending on the notion of uniformity used, each of the above definitions denotes several distinct classes. Thus AC^0 might be *Nonuniform-*AC^0, *P-uniform-*AC^0, *Logspace-uniform-*AC^0 or *Dlogtime-uniform-*AC^0. Context should make clear what is intended. (If uniformity is not explicitly mentioned, then the strongest interpretation is intended. That is, simulations hold in the Dlogtime-uniform setting, and lower bounds hold even in the nonuniform setting, unless we state otherwise.)

3 Closure Properties and Normal Forms

3.1 Normal Forms

In this section, we prove a number of closure properties and normal forms for $\#AC^0$ and $GapAC^0$. These help simplify some of the proofs in later sections, and are of independent interest.

Proposition 1 $FAC^0 \subseteq \#AC^0$.

Proof. We will need the following easy observation. (We will use the notation C_r also in later proofs.)

Proposition 2 *For every positive integer r, there is a depth 2 circuit C_r of size $O(r)$ having exactly 2^r accepting subtrees.*

First note that every language in AC^0 has its characteristic function in $\#\mathrm{AC}^0$. To see this, notice that one can restructure any AC^0 circuit into an equivalent one whose arithmetized version produces output in $\{0,1\}$. This is clearly true for any depth zero circuit. Now assume that this is true for all depth $k-1$ circuits and consider a depth k circuit. If the output gate is an AND then no further restructuring is necessary. If the output gate is an OR of the form $\vee_{i=1}^{m} G_i$, then replace it by the unambiguous circuit

$$\bigvee_{i=1}^{m} (G_i \wedge (\bigwedge_{j=1}^{i-1} \neg G_j)),$$

and propagate the NOT gates to the leaves.

Now consider multiple-output AC^0 circuits. Suppose the output bits $b_s \ldots b_0$ represent the binary representation of the output $f(x)$, then

$$\bigvee_{i=0}^{s} [b_i \wedge C_i]$$

is the required circuit showing that $f(x) \in \#\mathrm{AC}^0$. ∎

Our first normal-form theorem is an analog of a statement that is trivially true for the classes $\#\mathrm{P}$ and $\#\mathrm{L}$, as well as for other counting classes that can be modeled as the number of accepting paths of some sort of nondeterministic machine, where without loss of generality the machine makes one guess at each step. In the absence of such a model for $\#\mathrm{AC}^0$, a more complicated argument seems necessary.

Theorem 3 *For every AC^0-circuit M (on n inputs) and for all "sufficiently-large" polynomials $q(.)$, there is an AC^0 circuit N (on n inputs) such that,*

$$\forall x\ |x| = n \Rightarrow \#N(x) = 2^{q(n)} - \#M(x).$$

Proof. We proceed by induction on the height of M (i.e., the length of the longest path from the root to a leaf).

When the height is 0, the circuit consists simply of a literal or a constant ($=0$ or 1). Thus $\#M(x) = 0$ or 1. Thus it is sufficient to consider the following circuit,

$$N = \left[\bigvee_{i=0}^{q(n)-1} C_i\right] \vee \overline{M},$$

where $\overline{M}$ denotes the negation of the circuit M.

Now, consider a circuit M of height h. There are two subcases:

- M is a disjunction:

$$M = \bigvee_{i=1}^{k} M_i,$$

where $k = k(n) = n^{O(1)}$ is the number of gates feeding into the topmost $\vee$ gate. In this case, for each M_i, let N_i be the circuit, guaranteed by the inductive hypothesis, such that $\#N_i(x) = 2^{q_1(n)} - \#M_i(x)$. Let $q(n)$ be any polynomial such that $q(n) \geq q_1(n) + k(n)$, and let

$$N = \left[\bigvee_{i=1}^{k} N_i \right] \vee \bigvee_{i=1}^{q(n)-q_1(n)-k} C_{q_1(n)}.$$

Then,

$$\begin{aligned}
\#N(x) &= \sum_{i=1}^{k} \#N_i(x) + (q(n) - q_1(n) - k)\, 2^{q_1(n)} \\
&= \sum_{i=1}^{k} \left(2^{q_1(n)} - \#M_i(x)\right) + (q(n) - q_1(n) - k)\, 2^{q_1(n)} \\
&= (q(n) - q_1(n))2^{q_1(n)} - \sum_{i=1}^{k} \#M_i(x) \\
&= (q(n) - q_1(n))2^{q_1(n)} - \#M(x).
\end{aligned}$$

(In order to massage this into the precise form required, it suffices to appeal to Lemma 5 below.)

- M is a conjunction:

$$M = \bigwedge_{i=1}^{k} M_i,$$

where $k = k(n) = n^{O(1)}$ is the number of gates feeding into the topmost $\wedge$ gate. In this case, for each M_i, let N_i be the circuit such that, $\#N_i(x) = 2^{q_1(n)} - \#M_i(x)$ and let $M_0 = C_0$. Let

$$N = \bigvee_{i=0}^{k-1} \left[C_{iq_1(n)} \wedge N_{k-i} \wedge \left(\bigwedge_{j=0}^{k-i-1} M_j \right) \right].$$

Thus, with an appeal to Lemma 4 below, we have,

$$
\begin{aligned}
\#N(x) &= \sum_{i=0}^{k-1}\left[2^{iq_1(n)}\#N_{k-i}(x)\prod_{j=0}^{k-i-1}\#M_j(x)\right] \\
&= \sum_{i=0}^{k-1}\left[2^{iq_1(n)}\left(2^{q_1(n)}-\#M_{k-i}(x)\right)\right]\left[\prod_{j=0}^{k-i-1}\#M_j(x)\right] \\
&= 2^{kq_1(n)}-\prod_{j=1}^{k}\#M_j(x) \\
&= 2^{kq_1(n)}-\#M(x).
\end{aligned}
$$

The proof is now complete except for the following lemmas. (As David Mix Barrington [B97] has pointed out, this first lemma can be understood intuitively as computing the volume of a k-dimensional cube of side a with a piece removed.)

Lemma 4 *Let a, $a_1,\ldots,a_k$ be integers, where $a_0=1$. For all $k\geq 1$:*

$$
a^k-\prod_{i=1}^{k}a_i=\sum_{i=0}^{k-1}a^i\left(a-a_{k-i}\right)\prod_{j=0}^{k-i-1}a_j.
$$

Proof. We use induction on k.

$$
\begin{aligned}
a-a_1 &= a^0\left(a-a_1\right) \\
&= \textstyle\sum_{i=0}^{0}a^i\left(a-a_{1-i}\right)\prod_{j=0}^{1-i-1}a_j.
\end{aligned}
$$

This proves the base case ($k=1$). For the inductive step observe that,

$$
\begin{aligned}
&a^{k+1}-\prod_{i=0}^{k+1}a_i \\
&= a\left(a^k-\prod_{i=0}^{k}a_i\right)+\left(a-a_{k+1}\right)\prod_{i=0}^{k}a_i \\
&= a\sum_{i=0}^{k-1}a^i\left(a-a_{k-i}\right)\prod_{j=0}^{k-i-1}a_j+\left(a-a_{k+1}\right)\prod_{i=0}^{k}a_i \\
&= \sum_{i=0}^{k-1}a^{i+1}\left(a-a_{k-i}\right)\prod_{j=0}^{k-i-1}a_j+\left(a-a_{k+1}\right)\prod_{i=0}^{k}a_i \\
&= \sum_{i=1}^{k}a^i\left(a-a_{k-(i-1)}\right)\prod_{j=0}^{k-(i-1)-1}a_j+\left(a-a_{k+1}\right)\prod_{i=0}^{k}a_i
\end{aligned}
$$

$$\begin{aligned}
&= \sum_{i=1}^{k} a^i \left(a - a_{(k+1)-i}\right) \prod_{j=0}^{(k+1)-i-1} a_j + \left(a - a_{(k+1)-0}\right) \prod_{i=0}^{(k+1)-0-1} a_i \\
&= \sum_{i=0}^{k} a^i \left(a - a_{(k+1)-i}\right) \prod_{j=0}^{(k+1)-i-1} a_j .
\end{aligned}$$

∎

Lemma 5 *If $q(n)$ and $q_1(n)$ are polynomials, and $a(n) \in \mathrm{FAC}^0$, where $q(n) \geq q_1(n) + \log a(n)$, and if the function $a(n)2^{q_1(n)} - f(x)$ is in $\#\mathrm{AC}^0$, then $2^{q(n)} - f(x) \in \#\mathrm{AC}^0$.*

Proof. Let $c(n)$ be the value $2^{q(n)-q_1(n)} - a(n)$, and note that $c(n) \in \mathrm{FAC}^0$. Let $B(n) = \{j \mid$ bit number j of the binary representation of $c(n)$ is equal to $1\}$. The lemma now follows by considering the following $\#\mathrm{AC}^0$-computable function:

$$\begin{aligned}
& a(n)2^{q_1(n)} - f(x) + \left(\sum_{j\in B(n)} C_j\right) 2^{q_1(n)} \\
&= a(n)2^{q_1(n)} - f(x) + (2^{q(n)-q_1(n)} - a(n))2^{q_1(n)} \\
&= 2^{q(n)} - f(x).
\end{aligned}$$

∎

(End of the proof of Theorem 3.) ∎

Corollary 6 $\mathrm{DiffAC}^0 = \mathrm{FAC}^0 - \#\mathrm{AC}^0 = \#\mathrm{AC}^0 - \mathrm{FAC}^0$.

In fact, we have the stronger statement that if f and g are $\#\mathrm{AC}^0$ functions, then there exist polynomials q_1, q_2 and $\#\mathrm{AC}^0$ functions h_1, h_2 such that $f(x) - g(x) = 2^{q_1(|x|)} - h_1(x) = h_2(x) - 2^{q_2(|x|)}$. To see this, note that $f(x) - g(x) = 2^{n^k} - ((2^{n^k} - f(x)) + g(x))$. For large enough constant k, the function $((2^{n^k} - f(x)) + g(x))$ is in $\#\mathrm{AC}^0$, by Theorem 3.

3.2 Closure Properties

We begin with some simple closure properties. In [FFK94] the notions of "weak sum" and "weak product" were defined as follows:

Definition 8 *Let $\mathcal{C}$ be a class of functions.*

- *$\mathcal{C}$ is* closed under weak sum *if, for any $f \in \mathcal{C}$ and any k, the function $g(x) = \sum_{i=1}^{n^k} f(x,i)$ is in $\mathcal{C}$.*
- *$\mathcal{C}$ is* closed under weak product *if, for any $f \in \mathcal{C}$ and any k, the function $g(x) = \prod_{i=1}^{n^k} f(x,i)$ is in $\mathcal{C}$.*

Proposition 7 *$\#AC^0$ and $GapAC^0$ are closed under weak summation and weak product. $DiffAC^0$ is closed under weak sum.*

Proposition 8 *$DiffAC^0 = GapAC^0$ if and only if $DiffAC^0$ is closed under weak product.*

In contrast to the foregoing two propositions, the following closure property seems to require a much more complicated proof. Although closure under the choose operation is easy to show for classes such as #L and #P, where a machine can simply guess and execute k computation paths simultaneously, a different argument appears necessary for circuit-based counting classes.

Theorem 9 *$\#AC^0$, $DiffAC^0$ and $GapAC^0$ are closed under the choose operation (i.e. if $f(x)$ is a function in one of these classes, then so is $\binom{f(x)}{k}$ for any positive constant k).*

Proof.(of Theorem 9) Notice that once we know that $\#AC^0$ is closed under the choose operation, it follows immediately that $DiffAC^0$ is closed as well (using essentially the same proof as that of Closure Property 5 in [FFK94]).

We proceed by induction on the depth of the counting circuit computing the $\#AC^0$ function. If the circuit has depth 0, then the claim follows trivially as $f(x)$ assumes values 0 and 1 only. In order to prove the inductive step, we just need to prove that if $f_1(x)$, ..., $f_n(x)$ are $\#AC^0$ functions and for some constant k and all $j \leq k$,

$$\binom{f_1(x)}{j}, \ldots, \binom{f_n(x)}{j}$$

are $\#AC^0$ functions, then so are $\binom{\sum_{i=1}^n f_i(x)}{k}$ and $\binom{\prod_{i=1}^n f_i(x)}{k}$. Consider the identity:

$$\begin{aligned}(1+z)^{\sum_{i=1}^n f_i(x)} &= \prod_{i=1}^n (1+z)^{f_i(x)} \\ &= \prod_{i=1}^n \sum_{j=0}^{f_i(x)} \binom{f_i(x)}{j} z^j.\end{aligned}$$

The coefficient of z^k on the right hand side is $\sum \prod_{i=1}^n \binom{f_i(x)}{j_i}$, where the sum is taken over all distinct tuples $\langle j_1, \ldots, j_n \rangle$, satisfying $\sum_{i=1}^n j_i = k$. Thus comparing the coefficients of z^k on both sides of the identity we get:

$$\binom{\sum f_i(x)}{k} = \sum_{\vec{j}} \prod_{i=1}^n \binom{f_i(x)}{j_i}.$$

Here $\vec{j} = \langle j_1, \ldots, j_n \rangle$ is a partition of k into n parts. (Note that this shows that two multivariable polynomials agree on an infinite domain (the naturals); hence these polynomials agree also on the integers.)

Hence, in order to show that $\binom{\sum_{i=1}^n f_i(x)}{k}$ is in $\#AC^0$, we just need to show that the above expression involving a sum of products has polynomially many terms. But a simple inductive argument shows that it has less than n^k terms: letting $T(n,k)$ denote the number of terms in the sum and giving j_n values $0,\ldots,k$ successively, we get the following equation:

$$T(n,k) = T(1,0)T(n-1,k) + T(1,1)T(n-1,k-1) + \cdots + T(1,k)T(n-1,0).$$

Noting that $T(1,i)$ is 1 for each i and using a simple induction on k we get the result.

In order to prove that $\binom{\prod_{i=1}^n f_i(x)}{k}$ is in $\#\mathrm{AC}^0$, we first consider $\binom{ac}{k}$. Note that $\binom{ac}{k}$ is exactly the number of ways of choosing k distinct cells out of an $a \times c$ matrix. For any choice of k cells, let $b_1,\ldots,b_k$ denote the number of columns containing $1,\ldots,k$ of the chosen cells. Then an alternative way of choosing k cells out of this matrix is to first choose the integers $b_1,\ldots,b_k$, then choose the b_1 columns containing exactly one chosen cell and the chosen cell within each of these, then choose the b_2 columns containing exactly two chosen cells and the two chosen cells within them and so on.

Consider all distinct partitions of k in the form $k = 1b_1 + 2b_2 + \cdots + kb_k$. We denote by $\vec{b} = \langle b_1, b_2, \ldots, b_k\rangle$ one such partition and by S_k, the set of all such partitions. Also define $\pi(k)$ as the cardinality of the set S_k. Then,

$$\begin{aligned}
\binom{ac}{k} &= \sum_{\vec{b}\in S_k} \binom{c}{b_1}\binom{a}{1}^{b_1}\binom{c-b_1}{b_2}\binom{a}{2}^{b_2}\cdots\binom{c-b_1-b_2-\cdots-b_{k-1}}{b_k}\binom{a}{k}^{b_k} \\
&= \sum_{\vec{b}\in S_k} \frac{(b_1+b_2+\cdots+b_k)!}{b_1!b_2!\cdots b_k!}\binom{c}{b_1+b_2+\cdots+b_k}\binom{a}{1}^{b_1}\cdots\binom{a}{k}^{b_k}.
\end{aligned}$$

Thus we have shown that $\binom{ac}{k}$ can be represented by a sum of π_k terms. (As above, note that this equality holds also when a and c are integers.)
For $\vec{b} \in S_k$, let

$$F(n,k) = \binom{\prod_{i=1}^n f_i(x)}{k},$$

$$c_n = \prod_{i=1}^{n-1} f_i(x),$$

$$a_n = f_n(x),$$

$$G(a,\vec{b}) = \frac{(b_1+b_2+\cdots+b_k)!}{b_1!b_2!\cdots b_k!}\binom{a}{1}^{b_1}\cdots\binom{a}{k}^{b_k}.$$

Then using the above identity involving $\binom{ac}{k}$, we have:

$$F(n,k) = \sum_{\vec{b}} G(a_n,\vec{b})F(n-1,\sum_{i=1}^k b_i).$$

Let $R(n,k)$ denote the number of terms in this sum once the right hand side has been completely expanded as a sum of products. Then,

Claim 10 $R(n,k) \le \pi_1\pi_2\cdots\pi_k n^{k-1}$.

Proof. The claim clearly holds for $R(1,k)$. Now assume that the claim holds for all $n' < n$ and all $k' < k$, and consider the induction step.

Let S_k' be $S_k - \{\langle k, 0, , 0, \ldots, 0\rangle\}$. Then,

$$\begin{aligned}
R(n,k) &= \sum_{\vec{b}\in S_k} R\left(n-1, \sum b_i\right) \\
&= R(n-1,k) + \sum_{\vec{b}\in S_k'} R\left(n-1, \sum b_i\right) \\
&\le R(n-1,k) + \sum_{\vec{b}\in S_k'} R(n-1,k-1) \\
&\le R(n-1,k) + (\pi_k - 1) R(n-1,k-1) \\
&< R(n-1,k) + \pi_k\left(\pi_1\pi_2\cdots\pi_{k-1}(n-1)^{k-2}\right),
\end{aligned}$$

where the last inequality holds inductively. Thus,

$$\begin{aligned}
R(n,k) &< \pi_1\pi_2\cdots\pi_k \textstyle\sum_{i=0}^{n-1} i^{k-2} \\
&< \pi_1\pi_2\cdots\pi_k \textstyle\sum_{i=0}^{n-1} n^{k-2} \\
&= \pi_1\pi_2\cdots\pi_k n^{k-1}.
\end{aligned}$$

∎

This completes the proof of Theorem 9 for the #AC^0 case. Closure of GapAC^0 follows by an essentially identical proof. (The basis case needs to be augmented to deal with the constant -1.) ∎

A proof along these lines can also be used to show that #NC^1 is closed under the choose operation – but (as was pointed out to us by David Mix Barrington [B97]) a much simpler proof suffices for #NC^1, since one can show that #NC^1 is closed under the "monus" and "div m" operations for any constant m. (That is, if f is in #NC^1, then so are the functions $\max(f(x)-1, 0)$, and $\lceil f(x)/m \rceil$.) Although one can show that #AC^0 is also closed under the monus operation, #AC^0 and GapAC^0 are not closed under div m for any $m \neq 2$. Proofs of these and other recent results about #AC^0 will appear in upcoming work.

This is a good time to observe that #AC^0 is *not* closed under some more general choose operations. For instance, let $f(x) = \sum_i x_i$; f is clearly in #AC^0. For any function $g(n) \neq \log^{O(1)} n$, the function $\binom{f(x)}{g(n)}$ is not in #AC^0, since this function is 0 iff $f(x) < g(n)$, and thus the underlying Boolean AC^0 circuit would be computing the $g(n)$-threshold function, which is not in AC^0 [FKPS85, DGS86]. (This shows merely that $\binom{\sum_i x_i}{g(n)}$ is not in #AC^0; for an improvement of this result to a lower bound for GapAC^0, see Theorem 11.) This argument leaves open the question of what happens when $g(n) \neq O(1)$ but $g(n) = \log^{O(1)} n$. For g in this range, the $g(n)$ threshold is computable in AC^0 [FKPS85, DGS86], but the currently-known proofs of this fact do not preserve the number of accepting subtrees.

Open Question 6 *Are the functions* $\binom{\sum_i x_i}{\log n}$ *and* $\binom{\sum_i x_i}{\log^* n}$ *in* $\#\mathrm{AC}^0$*?*

Theorem 11 *For every integer* $k \geq 2$*, there are infinitely many integers* n *with the property that there is some* $j \leq \log^{3k^2} n$*, such that there is no* $\mathrm{GapAC^0}_k$*-circuit of size* $\leq n^{\log^{k-1} n}$ *computing the function*

$$\binom{\sum_{i=1}^n x_i}{j}$$

where the x_i*'s are Boolean variables. (In particular, for any superpolylogarithmic function* $g(n)$*, it is not the case that a* $\mathrm{GapAC^0}$ *circuit can compute* $\binom{\sum_i x_i}{j}$ *for all* $j \leq g(n)$*.)*

Proof. Assume otherwise. Thus we have a $k \geq 2$, such that for all large enough n and for each $j \leq \log^{3k^2} n$, there is a $\mathrm{GapAC^0}_k$-circuit of size at most $n^{\log^{k-1} n}$ computing $\binom{\sum_i x_i}{j}$.

Using these $\mathrm{GapAC^0}$ circuits, we will show how to compute the exact threshold predicate

$$\sum_{i=1}^m x_i = m/2$$

(for any large enough m of the form 2^r), using depth $k+1$ $\mathrm{AC^0}[2]$ circuits of size smaller than the $2^{\Omega(m^{1/2(k+1)})}$ lower bound proved in [Smo87].

We need the following fact (a proof of which can be found in [BT91, Fact 2.2]): a is divisible by 2^r iff $\binom{a}{2^j}$ is even for each $1 \leq j \leq r-1$.

Thus if $a = \sum_{i=1}^m x_i$ for some $m = 2^r$, the predicate

$$\sum_{i=1}^m x_i = m/2$$

is equivalent to

$$(\bigvee_i \bar{x}_i) \wedge (\bigvee_i x_i) \wedge \bigwedge_{t<r-1} \binom{\sum_i x_i}{2^t} \text{ is even.}$$

Let $n = \lceil 2^{m^{1/3k^2}} \rceil$. Thus $m \leq \log^{3k^2} n$, so for all $j \leq m$ there is a $\mathrm{GapAC^0}_k$ circuit of size $n^{\log^{k-1} n}$ computing $\binom{\sum_i x_i}{j}$, and thus the lower-order bit of this expression can be computed by $\mathrm{AC^0}[2]$ circuits of the same depth and size.

Thus the expression

$$(\bigvee_i \bar{x}_i) \wedge (\bigvee_i x_i) \wedge \bigwedge_{t<r-1} \binom{\sum_i x_i}{2^t} \text{ is even}$$

has depth $k+1$ $\mathrm{AC^0}[2]$ circuits of size $(n^{\log^{k-1} n})^{O(1)} = 2^{O(m^{1/3k^2} m^{(k-1)/3k^2})} = 2^{O(m^{1/3k})}$ which is asymptotically less than the lower bound of $2^{\Omega(m^{1/2(k+1)})}$ given by [Smo87].

∎

4 $\mathrm{C_{=}AC^0} = \mathrm{PAC^0} = \mathrm{TC^0}$

The most important step in proving this characterization involves showing how to simulate threshold circuits.

Theorem 12 *P-uniform* $\mathrm{TC^0} \subseteq$ *P-uniform* $\mathrm{C_{=}AC^0}$ *and Dlogtime-uniform* $\mathrm{TC^0} \subseteq$ *Dlogtime-uniform* $\mathrm{C_{=}AC^0_{\mathit{circ}}}$.

Proof. We will need to use the following well-known fact (see e.g. [PS88]),

Fact 13 *A problem is in* $\mathrm{TC^0}$ *if and only if it is accepted by a constant-depth family of "exact-threshold" gates* $ET^m_{m/2}$ *(an* ET^s_r *gate has s inputs and outputs 1 iff exactly r of them are 1).*

Let K_m be $\prod_{j=1,j\neq m/2}^{m}(m/2-j)$. First we will show that there exists a P-uniform $\mathrm{DiffAC^0}$ (Dlogtime-uniform $\mathrm{GapAC^0}$) function whose value on a string of m bits is K_m if the string contains exactly $m/2$ 1's and 0 otherwise. It is easy to see that the function

$$\Delta(x_1, x_2, \ldots, x_m) = \frac{\prod_{j\neq m/2}((\sum_i x_i) - j)}{K_m}$$

is 1 if $\sum_i x_i$ is $m/2$ and is 0 otherwise (the x_i's are Boolean variables). Thus $\Delta(x_1, x_2, \ldots, x_m) = ET^m_{m/2}(x_1, x_2, \ldots, x_m)$. Consider the function $P(X) = \prod_{j\neq m/2}(X-j)$. Expanding out the product and separating positive and negative terms, we get $P(X) = Q(X) - R(X)$, where $Q(.)$ and $R(.)$ are polynomials with coefficients whose binary representations are only polynomially long and hence can be computed by P-uniform $\#\mathrm{AC^0}$ circuits (notice that the coefficients depend only on m and not on X). Thus $Q(\sum_i x_i) - R(\sum_i x_i)$ is the required P-uniform $\mathrm{DiffAC^0}$ function. On the other hand $\prod_{j\neq m/2}((\sum_i x_i) - j)$ is already a Dlogtime-uniform $\mathrm{GapAC^0}$ function.

Inductively assume that given an exact threshold gate $\mathcal{G}$, there exists an integer t such that for each input $\mathcal{G}_i$ of $\mathcal{G}$, there is a P-uniform $\mathrm{DiffAC^0}$ (Dlogtime-uniform $\mathrm{GapAC^0}$) function taking values $(K_m)^t$ and 0 if $\mathcal{G}_i$ outputs 1 and 0 respectively. We show that there exists a P-uniform $\mathrm{DiffAC^0}$ (Dlogtime-uniform $\mathrm{GapAC^0}$) function taking values $(K_m)^{mt+1}$ and 0 whenever $\mathcal{G}$ outputs a 1 and 0 respectively. Notice that the theorem follows immediately from the base case proved above and this inductive step.

Let the $\mathrm{DiffAC^0}$ function corresponding to $\mathcal{G}_i$ be $F_i = f_i - g_i$ (f_i, g_i are $\#\mathrm{AC^0}$ functions). From Theorem 3 we know that there exists an integer q and an $\#\mathrm{AC^0}$ function h_i such that $F_i = f_i - g_i = h_i - 2^q$. Now the output of $\mathcal{G}$ is

$$\begin{aligned}
\Delta\left(\frac{F_1}{K_m^t},\ldots,\frac{F_m}{K_m^t}\right) &= \frac{\prod_{j\neq m/2}\left(\frac{\sum_i F_i}{K_m^t}-j\right)}{K_m} \\
&= \frac{\prod_{j\neq m/2}\left(\left(\sum_i F_i\right)-jK_m^t\right)}{K_m^{mt+1}} \\
&= \frac{\prod_{j\neq m/2}\left(\left(\sum_i h_i\right)-m2^q-jK_m^t\right)}{K_m^{mt+1}} \\
&= \frac{Q'\left(\sum_i h_i\right)-R'\left(\sum_i h_i\right)}{K_m^{mt+1}}.
\end{aligned}$$

Here $Q'(.)$ and $R'(.)$ are polynomials with coefficients in $\#\mathrm{AC}^0$. They are obtained by expanding the product $\prod_j (X-(m2^q+jK_m^t))$ (where $X=\sum_i h_i$) and separating the positive and negative terms. Just as in the base case their coefficients have polynomially long binary representations and hence can be computed by P-uniform $\#\mathrm{AC}^0$ circuits.

Obtaining a Dlogtime-uniform GapAC^0 is even simpler. Inductively suppose the GapAC^0 function corresponding to $\mathcal{G}_i$ is F_i. Then proceeding the same way as for DiffAC^0 functions we get

$$\Delta\left(\frac{F_1}{K_m^t},\ldots,\frac{F_m}{K_m^t}\right) = \frac{\prod_{j\neq m/2}\left(\left(\sum_i F_i\right)-jK_m^t\right)}{K_m^{mt+1}}.$$

Thus $\prod_{j\neq m/2}\left(\left(\sum_i F_i\right)-jK_m^t\right)$ is the Dlogtime-uniform GapAC^0 function corresponding to $\mathcal{G}$, whose value is either 0 or K_m^{mt+1}, according to whether gate $\mathcal{G}$ accepts.

This completes the proof of Theorem 12. ∎

Proposition 14 $\mathrm{C_=AC}^0 \subseteq \mathrm{PAC}^0$ *(under all considered notions of uniformity).*

Proposition 15 *P-uniform (non-uniform)* $\mathrm{GapAC}^0 \subseteq$ *P-uniform (non-uniform)* FTC^0*.*

(This is a simple consequence of the fact that unbounded fan-in addition and multiplication are in P-uniform TC^0 [RT92].)

Corollary 16 *In the P-Uniform and Non-Uniform Settings,*

$$\mathrm{C_=AC}^0 = \mathrm{PAC}^0 = \mathrm{TC}^0 = \mathrm{C_=AC}^0_{circ} = \mathrm{PAC}^0_{circ}.$$

Note that one interpretation of the preceding corollary is that TC^0 languages can be computed with just constant-depth arithmetic and a single threshold gate. Also note that, although we do not know if $\mathrm{DiffAC}^0 = \mathrm{GapAC}^0$, we obtain a characterization of TC^0 using *either* function class. Finally, note that our normal form theorems yield an even more restrictive characterization of TC^0.

Corollary 17 *For any set A in non-uniform or P-uniform* TC^0*, there exist a constant l, a function g in* $\#\mathrm{AC}^0$*, and a (non-uniform or P-uniform, respectively)* $\#\mathrm{AC}^0$ *function h with the following property:*

- *If $x \in A$, then $h(x) = 2^{|x|^l}$.*
- *If $x \notin A$, then $h(x) = 2^{|x|^l} + g(|x|)$.*

Proof. From the proof of Theorem 12, we know that there is a DiffAC^0 function f such that if $x \notin A$, then $f(x) = 0$, and if x is in A, then $f(x) = (K_m)^{tm+1}$ where $m = 2|x|^k$ for some k, and $K_m = \prod_{j=1, j \neq m/2}^{m} (m/2 - j)$. Let $g(|x|) = (K_m)^{tm+1}$. By Corollary 6, $f(x)$ is of the form $h(x) - 2^{|x|^l}$ for some $\#\mathrm{AC}^0$ function h and some constant l. ∎

(This corollary shows that TC^0 is the AC^0-analog of the class LWPP studied in [FFK94]. We refer the reader to [FFK94] for further details.)

Corollary 17 is probably nearly the strongest result in this direction that one can prove. For instance, one might seek to strengthen Corollary 17 to obtain $g(n) = 1$. (This corresponds to the AC^0-analog of the class SPP studied in [FFK94].) Note that if $g(n) = 1$, then the characteristic function of A is in GapAC^0. However, it follows from Proposition 28 that any such language A is in $\mathrm{AC}^0[2]$. Thus the lower bound of [Raz87] (showing that MAJORITY is not in $\mathrm{AC}^0[2]$) shows that we cannot improve Corollary 17 to obtain $g(n) = 1$.

More generally, observe that the function $g(n)$ has lots of small divisors. This is no accident. Assume for the moment that one could strengthen Corollary 17 so that $g(n)$ is of the form 2^{n^k} (for example). Then it would follow that $\mathrm{TC}^0 = \mathrm{ACC}^0$. To see this, note that the GapAC^0 function $m(x) = h(x) - 2^{n^l}$ would have the property that $m(x)$ is a multiple of 3 if and only if $x \in A$. As is clear from Theorem 27, this property can be checked in $\mathrm{AC}^0[6]$. More generally, if Corollary 17 can be strengthened so that, for some prime p, there are infinitely many n such that $g(n)$ is not a multiple of p, then there is an ACC^0 circuit family that, for infinitely many n, computes the MAJORITY function on n variables.

It is of interest to us to try to improve the uniformity condition. This leads us to the next theorem.

Theorem 18 *Logspace-uniform* $\mathrm{PAC}^0_{circ} \subseteq$ *Logspace-uniform* TC^0*.*

Before proving this theorem, we need to introduce some number theoretic machinery.

Definition 9 *Z_p is the group over $\{0, \ldots, p-1\}$ with modulo p addition (p any prime)and Z_p^* is the multiplicative group over $\{1, \ldots, p-1\}$ modulo p. Further g is a generator of Z_p^* if $Z_p^* = \{1, g, g^2, \ldots, g^{p-1}\}$ (all products are taken in Z_p^*). $ind_{g,p}, pow_{g,p} : Z_p^* \to Z_p^*$ are functions satisfying $g^{ind_{g,p}(x)} = x$ and $pow_{g,p}(x) = g^x$.*

We will need the following variant of the Chinese Remainder Theorem:

Theorem 19 *(see e.g. [HW79]) Given primes $p_1, p_2, \ldots, p_s$ and an integer x, there are unique integers $x_1, x_2, \ldots, x_s$ (modulo $p_1, p_2, \ldots, p_s$ respectively), satisfying*

$$\begin{aligned} x &\equiv x_1 \bmod p_1, \\ x &\equiv x_2 \bmod p_2, \\ &\vdots \\ x &\equiv x_s \bmod p_s. \end{aligned}$$

And if any y satisfies the congruences above then $x \equiv y \bmod P_s$, where $P_s = \prod_{i=1}^{s} p_i$. More explicitly, $x = A_s - q_s P_s$ where

$$A_s = \sum_{j=1}^{s} \left((x_j c_{s,j}) \bmod p_j \right) P_s/p_j,$$

$$c_{s,j} = (P_s/p_j)^{-1} \bmod p_j,$$
$$q_s = \lfloor A_s/P_s \rfloor .$$

And the following variant of the prime-number theorem:

Theorem 20 *(see e.g. [HW79]) For sufficiently large values of n, the product of all primes less than n exceeds 2^n.*

As a consequence of Theorems 19 and 20 we get the following corollary:

Corollary 21 *If $0 \le x < 2^n$, then x can uniquely be represented as $\vec{x} = (x_1, x_2, \ldots, x_s)$, where $x \equiv x_i \bmod p_i$ (for $1 \le i \le s$) and $p_1, p_2, \ldots, p_s$ are the primes smaller than n. $\vec{x}$ is called the Chinese Remainder Representation of x.*

Lemma 22 *There is a Logspace-uniform TC^0 circuit that decides whether a number less than P_k is actually less than $P_k/2$, given the residues modulo $p_1, p_2, \ldots, p_k$ (here $p_1, \ldots, p_k$ are the first k primes and P_k is their product).*

Proof. We consistently use the notation (viz. $p_j, P_i, c_{i,j}, q_i, \vec{x}$ etc.) introduced in Theorem 19 and Corollary 21. Let

$$X_i = A_i - q_i P_i$$

(in the remaining portion of the proof any unqualified i or j refers to an integer in the range $[1, k]$). From the Chinese Remainder Theorem we know that the number in question (i.e. the number with residues x_i modulo p_i) is

$$X_k = A_k - q_k P_k.$$

Essentially we need to find out the first bit of the fractional representation of A_k/P_k. [DMS94] shows how to do this in Logspace. We show that their method is amenable to a TC^0 circuit implementation.

The essential idea is to compute the first $3\lceil\log_2 i\rceil$ bits of each of the fractions $t_{i,j}(x) = ((x_j c_{i,j}) \bmod p_j)/p_j$ (for $1 \le j \le i \le k$) and find the sums $q_i'(x) = \sum_{j=1}^{i} t_{i,j}(x)$ approximating A_i/P_i. [DMS94] show that if the fractional part of $q_k'(x)$ contains any zeros, then the first bit of the fractional part of $q_k'(x)$ is equal to the first bit of the fractional part of A_k/P_k (which is the bit that we need to compute). If, instead, the fractional part of q_k' is all ones, then consider the number x' that results by flipping the bit x_1 (recording the residue mod 2 of x). In the Chinese Remainder Representation, the number x' is equal to $x + P_k/2$ or $x - P_k/2$; note that $x < P_k/2$ if and only if $x' \ge P_k/2$, thus if the fractional part of $q_k'(x')$ contains any zeros, we again know the value we want. If the fractional parts of $q'(x)$ and of $q'(x')$ are both all ones, then [DMS94] show that the computation can be repeated using q_{k-1}' approximating A_{k-1}/P_{k-1} (which in this case has the same bit as A_k/P_k). Thus our answer can be computed by finding the value i^*, which is the largest $i < k$ for which the first $2\lceil\log_2 i\rceil$ bits of the fractional part of $q_i'(x)$ or of $q_i'(x')$ are not all 1.

So we just need to show that each of the q_i''s and i^* can be computed using Logspace-uniform TC^0 circuits. But this follows from Lemma 24-7,8 below. ∎

(We remark that, instead of relying on [DMS94], it is also possible to make use of similar results of [Lit92, DL91]. It seems to us that the construction of [DMS94] results in a simpler circuit.)

Lemma 23 *The following are computable in $O(\log n)$ space where in the following x is n bits long and p, p_i, g, k, z are all $O(\log n)$ bits long.*

1. *A generator g of the multiplicative group Z_p, given prime p.*
2. *The function $z, p \mapsto pow_{g,p}(z)$ (see Definition 9) where g is as in item 1 above.*
3. *The function $z, p \mapsto ind_{g,p}(z)$ (see Definition 9) where g is as in item 1 above.*
4. *The function $mod_k : k, x \mapsto x \bmod k$.*
5. *The function $i, j \mapsto c_{i,j}$.*
6. *The function $t : i, j, z \mapsto$ the first $3\lceil\log_2 i\rceil$ bits of $zc_{i,j}/p_j$ (where $z < p_j$).*

Proof.

- (of Lemma 23-1) For each $h \in Z_p$ try if $h^{(p-1)/2} \equiv -1 \bmod p$, if yes then it is a generator else not. As h and p are only $O(\log n)$ bits long this can be done in logspace.
- (of Lemma 23-2) Given a number z compute g^z, reducing the result g^i modulo p at each step.
- (of Lemma 23-3) Given a number z first find its modulo p representation y then for each element $y \in Z_p$ check whether $g^y \bmod p = z$.

- (of Lemma 23-4) With numbers in this range, the standard long-division algorithm runs in logspace.

- (of Lemma 23-5) By successively testing each integer for primality compute the i^{th},j^{th} prime p_i, p_j. Now successively (re)compute p_k (for $k \leq i$) and if $k \neq j$, then find the modulo-p_j inverse of p_k (by reducing p_k modulo p_j and checking for each positive number $l < p_j$ whether $l \cdot p_k \equiv 1(\mathrm{mod} p_j)$). Keep accumulating the inverses in a product modulo-p_j. The final value of the product is the modulo-p_j inverse of P_i/p_j.

- (of Lemma 23-6) Compute p_j, and then compute $c_{i,j}$ using Lemma 23-5 above. Compute the product $xc_{i,j}$ and then produce the first $3\lceil\log_2 i\rceil$ bits of $xc_{i,j}/p_j$ using the standard long division algorithm.

∎

Lemma 24 *The following are computable using a Logspace-uniform* TC^0 *circuit (the length of the input is always* $O(n)$*):*

1. $f \circ g$ *where* $f : \{0,1\}^{c\log n} \to \{0,1\}^m$ *is a Logspace computable function and* $g : \{0,1\}^n \to \{0,1\}^{c\log n}$ *is a function in Logspace-uniform* TC^0.

2. $f \circ \vec{g}$ *(where this is defined as* $f(g(x_1), g(x_2), \ldots, g(x_n)))$*, where* f *and* g *are in Logspace-uniform* TC^0.

3. *The sum of* n *integers each having* $O(n)$ *bits (denote this function by sum).*

4. *The iterated sum modulo* k *(any* $O(\log n)$ *bit integer) of* n *integers each having* $O(n)$ *bits (denote by* sum_k*).*

5. *The iterated product modulo* p *(any* $O(\log n)$ *bit prime) of* n *integers each having* $O(n)$ *bits (denote by* $prod_p$*).*

6. *The function* $t : i, j, x \mapsto$ *the first* $3\lceil\log_2 i\rceil$ *bits of* $xc_{i,j}/p_j$ *(where* $x < p_j$*).*

7. *The function* $q' : i, \vec{x} \mapsto \sum_{j=1}^{i} t(i,j,x_j)$ *(where* $\vec{x} = \langle x_1, x_2, \ldots, x_k\rangle$*).*

8. *The function* $\vec{x} \mapsto i^*$*, where* i^* *is the maximum* i *such that the first* $2\lceil\log_2 i\rceil$ *bits of the fractional part of* $q'(i,\vec{x})$ *are not all 1's.*

Proof.

- (of Lemma 24-1) Construct a circuit whose i^{th} output bit (on input y) is given by

$$\bigvee_{j=1}^{n^c} \left(\bigwedge_{k=1}^{c\log n} (j[k] = y[k]) \bigwedge f(j)[i] \right)$$

  (here $a[i]$ denotes the i^{th} bit of integer a). This circuit is clearly Logspace uniform from the Logspace computability of f. Composing this circuit with the circuit for g yields the required circuit.

- (of Lemma 24-2) Straightforward.
- (of Lemma 24-3) This is well known (e.g. see [CSV84]).
- (of Lemma 24-4) $sum_k = mod_k \circ sum$. Result follows from Lemmas 23-4, 24-3 and 24-1.
- (of Lemma 24-5) Let $g = ind_g \circ mod_p$. Then $prod_p = pow_g \circ sum_{p-1} \circ \vec{g}$. Result follows from Lemmas 23-4, 23-2, 23-3, 24-3, 24-2, and 24-1.
- (of Lemma 24-6) Follows from 23-6 and 24-1.
- (of Lemma 24-7) Follows from 24-6 and 24-3.
- (of Lemma 24-8) Given the values of $q_i^{'}$ $(1 \leq i \leq k)$, finding the maximum i satisfying an AC^0 property is equivalent to finding the rightmost 0 in an array of length k, which is again in AC^0.

■

Proof.(of Theorem 18) Let k be the constant such that for all large n, the value of the given GapAC^0 circuit has absolute value less than 2^{n^k}. (Such a k exists, since $\#AC^0 \subseteq \#P$.) We use Lemmas 24-4 and Lemmas 24-5 to construct a TC^0 circuit that computes the answer modulo the primes less than n^{2k}. This way any positive number is mapped into $[0, P/2)$ and any negative number into $(P/2, P-1]$. Thus it is sufficient to test whether the final answer is greater than $P/2$, which can be done with the help of Lemma 22. ■

Corollary 25 *In the Logspace-Uniform Setting,*

$$\mathrm{TC}^0 = \mathrm{C}_{=}\mathrm{AC}^0_{circ} = \mathrm{PAC}^0_{circ}.$$

5 Arithmetic Circuits over Finite Fields

There has been earlier work characterizing TC^0 in terms of finite fields [BFS92, FVB94, RT92]. However, this earlier work provides no connection to AC^0, and the characterizations involve having a different finite field for each input length. Also, these earlier characterizations dealt with arithmetic circuits with additional gates (such as conjugation gates, or division gates) in addition to the $+$ and $\times$ gates that we use.

It may be interesting to point out that, when one uses our notion of arithmetic circuits over finite fields, one obtains a characterization of ACC^0. (It has been pointed out to us by David Mix Barrington [B97] that this is in some sense implicit in the work of Smolensky [Smo87].)

We need the following fact from number theory.

Theorem 26 *(Dirichlet) (see e.g. [HW79]) For any two relatively prime numbers q and r, there exist infinitely many primes in the sequence $\{qn+r\}_{n=1}^{\infty}$.*

Let F be a finite field, and let $\#\mathrm{AC}^0_F$ denote the class of functions computed by $\#\mathrm{AC}^0$ circuits, where now $+$ and $\times$ are operations over the field F.

Theorem 27 *A language is in ACC^0 if and only if its characteristic function is in $\#\mathrm{AC}^0_F$ for some finite field F.*

Proof. Let A be a language in ACC^0; thus A is in $\mathrm{AC}^0[m]$ for some m. Without loss of generality the only gates are $\wedge$ and Mod_m (since $\vee$ can be simulated by $\wedge$ and Mod_m). Our first step is to find a prime p of the form $am+1$ for some a, using Dirichlet's theorem (Theorem 26) above. Now make a copies of each gate, and replace all Mod_m gates with Mod_{p-1} gates (keeping in mind that $m \mid x \Leftrightarrow am \mid ax$). Thus at this stage the only gates are $\wedge$ and Mod_{p-1}. Now an $\wedge$ gate can be replaced by a product gate modulo p (since the value of each gate is Boolean). It remains only to simulate the Mod_{p-1} gates.

Let an arbitrary Mod_{p-1} gate have inputs $x_1, \ldots, x_r$. Consider

$$X = \prod_i (1 + (g-1)x_i) \bmod p$$

(where g is a generator of the multiplicative group modulo p); this has value 1 iff the number of x_i's that are 1 is divisible by $(p-1)$. Further, $(X+p-1)^{p-1} \bmod p$ is 0 if X is 1 mod p and is 1 otherwise. This gives us the arithmetic circuit equivalent to the Mod_{p-1} gate.

The other direction is equally simple. We will build a circuit that computes, for each gate g, a representation of the field element to which g evaluates. Let the finite field F be $\mathrm{GF}(p^k)$. We will use two representations. One representation $\mathrm{rep}_+(x)$ will be as a k-tuple of strings of the form $1^{a_i}0^{p-a_i}$, where x corresponds to element $(a_1, a_2, \ldots, a_k)$ when F is viewed as a vector space over $\mathrm{GF}(p)$. Note that, when given n such k-tuples, their sum can easily be computed by $\mathrm{AC}^0[p]$ circuits. (When adding up the l^{th} components, test for each $j \le p$ if $\mathrm{Mod}_p(rep_+(x_{1,l}) \cdots rep_+(x_{n,l})1^j)$ holds. If so, then output $1^{p-j}0^j$ as the value of the l^{th} component.)

The other representation $\mathrm{rep}_\times(x)$ will be of the form $1^i0^{p^k-i}$ where $g^i = x$, where g is a generator of the multiplicative group of F. Since F is finite, each representation is only $O(1)$ bits, and conversion between representations can be computed in AC^0. Now the product $\prod_i x_i$ can be computed by computing $\sum_i \mathrm{rep}_\times(x_i) \bmod (p^k-1)$, which can clearly be computed in $\mathrm{AC}^0[p^k-1]$. This completes the proof. ∎

Although in general there is no close connection between arithmetic circuits over $\mathrm{GF}(p)$ and $\mathrm{AC}^0[p]$ (since, for example, both PARITY and Mod_3 are computable with arithmetic circuits over $\mathrm{GF}(3)$), there is one important case where an equivalence does hold. (The proof of the following proposition is an easy modification of the foregoing.)

Proposition 28 *The following are equivalent:*

1. $A \in \mathrm{AC}^0[2]$.

2. $\chi_A \in \mathrm{GapAC}^0$.

3. χ_A *can be represented as the low-order bit of some* $\#\mathrm{AC}^0$ *function.*

4. $\chi_A \in \#\mathrm{AC}^0_{GF(2)}$.

Proof. The only one of the implications $1 \Longrightarrow 2 \Longrightarrow 3 \Longrightarrow 4 \Longrightarrow 1$ that requires much explanation is $1 \Longrightarrow 2$. This implication is proved by induction on the depth of the $\mathrm{AC}^0[2]$ circuit (composed of only PARITY and AND gates). The main observation required in the inductive step is that, in order to simulate a PARITY gate with inputs $f(x,i)$ (for $1 \leq i \leq n^k$), it suffices to use the following function:

$$\binom{1 + \prod(1 - 2f(x,i))}{2}.$$

This is in GapAC^0 by the closure properties established in Section 3.

V. Vinay (personal communication) has pointed out that the following direct construction also computes the zero-one PARITY function:

$$\sum_i \left(\prod_{j<i} (1 - 2f(x,j)) \right) f(x,i).$$

∎

A similar argument shows that, for all prime p and for all k, $\#\mathrm{AC}^0_{GF(p^k)}$ corresponds exactly to $\mathrm{AC}^0[p(p^k-1)]$. We close this section with another question concerning the relationship between $\#\mathrm{AC}^0$ and DiffAC^0.

Open Question 7 *Is there any set* $A \notin \mathrm{AC}^0$ *such that* $\chi_A \in \mathrm{DiffAC}^0$*?*

6 Lower Bounds, and Conclusions

We know many lower bounds for $\#\mathrm{AC}^0$. For instance, the Mod_3 function is not in $\#\mathrm{AC}^0$, as a consequence of Proposition 28 and the circuit lower bounds in [Raz87]. At the end of Section 3.2 we saw that some functions related to the symmetric polynomials are not in $\#\mathrm{AC}^0$. Other examples can easily be generated as easy consequences of known circuit lower bounds. (In contrast, no function in $\#\mathrm{P}$ or in P^{NP} is known not to be in $\#\mathrm{NC}^1$.) We can also show that the ${\#\mathrm{AC}^0}_k$ and ${\mathrm{GapAC}^0}_k$ hierarchies are strict using the known lower bounds.

Theorem 29 *For any* $k > 0$, ${\#\mathrm{AC}^0}_k \subset {\#\mathrm{AC}^0}_{k+1}$, *and* ${\mathrm{GapAC}^0}_k \subset {\mathrm{GapAC}^0}_{k+1}$.

Proof. We prove the theorem for $\#AC^0$, the proof for $GapAC^0$ is identical.

Assume that ${\#AC^0}_k = {\#AC^0}_{k+1}$ for some $k > 0$. It follows then that $\#AC^0 = {\#AC^0}_k$. Let A be a language in AC^0 but not in depth k $AC^0[2]$. (See, for instance Proposition 11 in [AH94]. We can choose A to be the mod 3 of the first $\log^a n$ bits, for some a.) The characteristic function of A is in $\#AC^0$, and therefore, in ${\#AC^0}_k$ by our assumption. But this gives a depth k $AC^0[2]$ circuit for A, in contradiction to our choice of A. ∎

On the other hand, we know essentially no lower bounds for threshold circuits, which amounts to studying the limits of what can be expressed as the high-order bit of a $\#AC^0$ function. A great many new questions present themselves, such as whether $DiffAC^0 = GapAC^0$. Although the resolution of this and related questions would not immediately yield lower bounds for threshold circuits, it would be informative to learn more about what can and cannot be computed in $\#AC^0$ and increase our store of lower bound techniques for dealing with this class of functions.

Note that [RR94] argues that, if certain popular cryptographic assumptions are true, then there are no "natural proofs" of lower bounds for TC^0 circuits. The model of arithmetic circuits considered here has not been studied in sufficient detail for it to be clear whether this should be considered a significant obstacle to proving lower bounds for TC^0 via arithmetic circuits. Since many lower bounds for $\#AC^0$ can be proved using natural proofs, it would also be interesting to know what types of questions about $\#AC^0$ can be addressed via natural proofs, and which cannot.

The issue of uniformity is especially interesting, and it again leads us to the frontier of current lower bound technology. It is currently an open question whether a given bit of the permanent can be computed as the high-order bit of Dlogtime-uniform $\#AC^0$ functions, although if the inclusion $PAC^0 \subseteq TC^0$ holds also in the Dlogtime-uniform setting, then a negative answer would follow from the lower bound of [All].

The main obstacle to proving a Dlogtime-uniform analog to Theorem 18 seems to be the problem of finding a generator for the multiplicative group Z_p. Note in this regard that our proof uses the fact that a logspace machine can check if the graph on $\{1, \ldots, p\}$ with edges $i \rightarrow ig(\mathrm{mod} p)$ is a cycle. (It is a cycle if and only if g is a generator.) The problem of checking if a general graph is a cycle is complete for logspace (see, e.g., [Ete95]), and thus any argument handling the Dlogtime-uniform TC^0 case will almost certainly need to make use of special properties of this graph.

Getting rid of the P-uniformity condition in Theorem 12 seems closely-related to the problem of finding Logspace-uniform (or Dlogtime-uniform) circuits for iterated integer multiplication, which in turn is equivalent to obtaining more uniform circuits for division [BCH86, IL95, RT92].

Another direction worth investigating concerns branching programs. It is shown in [CMTV96] that $\#NC^1$ is closely related to the problem of counting paths in bounded-width branching programs, and it is also known that AC^0 is the class of languages accepted by branching programs over *acyclic monoids* [BT88]. Is there some characterization of $\#AC^0$ or $DiffAC^0$ in terms of branching programs? Is there a related algebraic characterization?

Acknowledgments

We would like to thank Richard Bumby, David Mix Barrington, Pierre McKenzie, Denis Therien, Noam Nisan and Dieter van Melkebeek for discussions and suggestions on the material.

References

[AH94] E. Allender and U. Hertrampf, Depth Reduction for Circuits of Unbounded Fan-In. *Information and Computation*, 112(2):217–238, 1994.

[ABO96] E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. In *Proc. 28th ACM Symposium on Theory of Computing (STOC)*, pages 161–167, 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.

[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].

[All] E. Allender. The permanent requires large uniform threshold circuits. Submitted. A preliminary version of this paper appeared as [All96].

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

[All96] E. Allender. A note on uniform circuit lower bounds for the counting hierarchy. In *International Conference on Computing and Combinatorics Conference (COCOON)*, volume 1090 of *Lecture Notes in Computer Science*, pages 127–135. Springer-Verlag, 1996.

[AO94] E. Allender and M. Ogihara. Relationships among PL, #L, and the determinant. In *Proc. 9th IEEE Structure in Complexity Theory Conference*, pages 267–278, 1994.

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

[BFS92] J. Boyar, G. Frandsen, and G. Sturtivant. An arithmetical model of computation equivalent to threshold circuits. *Theoretical Computer Science*, 93:303–319, 1992.

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

[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.

[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.

[BT91] Richard Beigel and Jun Tarui. On ACC. *32nd Annual Symposium on Foundations of Computer Science*, pages 783–792, 1991.

[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.

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

[DGS86] L. Denenberg, Y. Gurevich, and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Control*, 70:216–240, 1986.

[DL91] George I. Davida and Bruce Litow. Fast parallel arithmetic via modular representation. *SIAM Journal on Computing*, 20(4):756–765, August 1991.

[DMS94] Paul F. Dietz, Ioan I. Macarie, and Joel I. Seiferas. Bits and relative order from residues, space efficiently. *Information Processing Letters*, 50(3):123–127, 9 May 1994.

[Ete95] K. Etessami. Counting quantifiers, successor relations, and logarithmic space. In *Proc. 10th IEEE Structure in Complexity Theory Conference*, pages 2–11, 1995.

[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.

[FKPS85] R. Fagin, M. Klawe, N. Pippenger, and L. Stockmeyer. Bounded-depth, polynomial-size circuits for symmetric functions. *Theoretical Computer Science*, 36:239–250, 1985.

[FVB94] G. Frandsen, M. Valence, and D. Mix Barrington. Some results on uniform arithmetic circuit complexity. *Mathematical Systems Theory*, 27, 1994.

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

[HW79] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford press, USA, 1979.

[IL95] N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116:103–116, 1995.

[Lit92] B. Litow. On iterated integer product. *Information Processing Letters*, 42(5):269–272, 03 July 1992.

[Mac] I. Macarie. Space-efficient deterministic simulation of probabilistic automata. To appear in *SIAM J. Comput.* Preliminary version appeared as [Mac94].

[Mac94] 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.

[MV] M. Mahajan and V. Vinay. 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.

[PS88] I. Parberry and G. Schnitger. Parallel computation with threshold functions. *Journal of Computer and System Sciences*, 36:278–302, 1988.

[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.

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

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

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

[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.

[ST] M. Santha and S. Tan. Verifying the determinant in parallel. To appear in *Computational Complexity*. A preliminary version of this paper appeared in Proceedings of the 5th ACM-SIGSAM International Symposium on Symbolic and Algebraic Computation, 1994.

[Tod] S. Toda. Counting problems computationally equivalent to the determinant. Manuscript.

[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 *Proc. 11th 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.

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

[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 *Proc. 6th IEEE Structure in Complexity Theory Conference*, pages 270–284, 1991.

[Wag86] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23:325–356, 1986.

[Yam96] T. Yamakami. Uniform AC^0 Counting Circuits. Manuscript, 1996.