BOUNDED DEPTH ARITHMETIC CIRCUITS

BY SAMIR DATTA

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Eric Allender

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

July, 2004

ABSTRACT OF THE DISSERTATION

Bounded Depth Arithmetic Circuits

by Samir Datta

Dissertation Director: Eric Allender

Continuing a line of investigation that has studied the function classes #P [56], #SAC^1 [60, 11], #L [12, 60, 8], and #NC^1 [24], we study the class of functions #AC^0. One way to define #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 #AC^0 follow easily from established circuit lower bounds.

One of our main results is a characterization of TC^0 in terms of #AC^0: A language A is in TC^0 if and only if there is a #AC^0 function f and a number k such that $x \in A \iff f(x) = 2^{|x|^k}$. Using the naming conventions of [31, 24], 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 [48, 23]. Using our model of arithmetic circuits, computation over finite fields yields ACC^0.

We also resolve several questions regarding the closure properties of $\#AC^0$ and $GapAC^0$ and characterize $\#AC^0$ in terms of counting paths in a specific family of bounded-width graphs.

Acknowledgements

Firstly, I would like to thank my advisor Eric Allender for his continued support and unending patience in dealing with me. Without his efforts this thesis would not have seen the light of the day. I do, however, regret not learning as much as I could have, from him.

I would also like to thank Dave Barrington and Manindra Agrawal, collaborating with whom was a learning experience. Andris Ambainis and Huong LêThanh were the other two collaborators on work reported in this thesis and working with them was a brief but gainful experience.

I am grateful to the Computer Science Department for giving me an opportunity to study here. I had the fortune to come in contact with some of the great names in Theoretical Computer Science, particularly Endre Szemeredi, Jozsef Beck and Leonid Khachiyan. Looking back, I wish I had learnt more from them.

I would also like to express my gratitude to my "buddies" in the department. Stefan Langerman and Sachin Lodha would be the two from whom I have learnt the most and without whose quick humor, the long years as a graduate student would have been insufferable. Thanks to Pradeep Sudame and Ramkrishna Chatterjee from whom I learnt a lot about life. Thanks also to Samrat Ganguly, Navin Goyal and Sambuddha Roy, my latter day friends in the department.

Lastly, I would like to thank my family - my parents, my brother Sudip and my wife Somya - whose unyielding support is the bedrock on which all my achievements are based.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Counting Classes

Certainly the best-known counting class is Valiant's class #P [56], consisting of functions that map x to the number of accepting computations of an NP-machine on input x. #P characterizes the complexity of computing the permanent of a matrix [56]. The class #P consists of very hard functions, so much so that using functions from it as oracular advice, a polynomial time machine can capture the whole of the polynomial hierarchy, which is the celebrated theorem of Toda [53].

Recently, the class #L (counting accepting computations of an NL-machine) has also received considerable attention [12, 60, 55, 42]. This class characterizes the complexity of computing the determinant [60, 55, 57, 42].

The two classes #P and #L can be thought of as counting analogs of the classes NP and NL. 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 polynomial depth," and NL sets are accepted by uniform polynomial-size "skew" circuits [59]. 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 [59] 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 [54].

The Boolean class SAC^1(also known as LOGCFL) is the class of languages accepted by a nondeterministic auxiliary pushdown automaton in logarithmic space and polynomial time. [58] gives a neat characterization of SAC^1 in terms of semi-unbounded circuits of low depth. The counting class that results by arithmetizing SAC^1(as for NP and NL above) using its circuit definition was studied in [60], where it was shown that $\#\mathrm{SAC}^1$ corresponds to counting the accepting paths of an NAuxPDA. The class has also been studied in [11].

The Boolean circuit class NC^1, which is characterized by logarithmic depth circuits of bounded indegree (see Chapter 2 for a formal definition) was arithmetized in [24] to obtain $\#\mathrm{NC}^1$. It was also shown there that $\#\mathrm{NC}^1$ is closely-related to counting paths in bounded-width branching programs.

In this thesis, we study $\#\mathrm{AC}^0$, obtained by arithmetizing the Boolean circuit class AC^0, the class of languages characterized by bounded depth circuits of unbounded degree (see Chapter 2 for a formal definition) and study its properties and that of related classes.

#P gives rise to the class GapP consisting of the difference of two #P functions. [31] highlighted the importance of this class and studies many of its properties. The classes GapL, GapSAC^1, and GapNC^1 have also been defined and studied in literature. We introduce the class $\mathrm{GapAC}^0$and study its properties. This thesis includes work originally reported in [1, 10]. Other related papers are [13, 38, 43, 26].

1.2 Circuit Lower Bounds

Why study $\#\mathrm{AC}^0$? Our motivation comes in large part from a desire to obtain more lower bounds in circuit complexity. As we shall see, $\#\mathrm{AC}^0$ straddles the boundary marking the limits of current circuit lower bound technology. Before going any further

let us make a brief foray into the history of lower bounds.

The first non linear lower bound to be proven showed that PARITY could not be computed by polynomial sized, constant depth circuits over AND, OR and NOT gates [33, 2]. This separated AC^0 from higher classes. Later, Razborov [46] proved that MAJORITY could not be computed by polynomial sized, constant depth circuits over AND, OR, NOT and PARITY. This proof was later simplified and extended by Smolensky [52] who showed that the MOD_q function cannot be computed by a polynomial sized, constant depth circuit over AND, OR, NOT and MOD_p (for primes p, q). Noticing that a threshold circuit of polynomial size and constant depth and can simulate a MOD_q function, it follows easily that the Razborov-Smolensky result separated $AC^0[p]$ from the higher class TC^0. Here the class $AC^0[p]$ arises if we augment AC^0 with MOD_p gates, while the class TC^0 arises if we augment AC^0 with threshold gates (see Chapter 2 for formal definitions).

Thus we have the following inclusion list of complexity classes:$AC^0 \subsetneq AC^0[p] \subsetneq TC^0 \subseteq NC^1 \subseteq NL \subseteq SAC^1 \subseteq NP$ and this is the best known to date in the sense that no other separations are known.

Threshold circuits of constant depth, referred to above, characterize a varied and well known class of problems like majority, multiplication and sorting. They are also the building blocks of neural nets. Thus it is important to try to understand them and study their limitations. Proving lower bounds for TC^0 is a challenging open problem.

1.3 Contributions of the Thesis

This is where $\#AC^0$ enters the picture. In Chapter 4 we define language classes $C_=AC^0$ and PAC^0 in terms of functions in $\#AC^0$ (for definitions see Chapter 2) and prove that, at least under P-uniformity these classes coincide with TC^0. For other notions of uniformity we prove somewhat weaker results.

Next, in Chapter 5, we show the close relationship between arithmetic circuits over finite fields and ACC^0, and, in particular, between $GapAC^0$ and $AC^0[2]$. 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}.)

In Chapter 8 we show that $\#AC^0{}_k$, the class of functions defined by $\#AC^0$ circuits of depth at most k (for a formal definition see Chapter 2), 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.

1.3.1 Closure Properties

Amidst the activity centered around #P and GapP, is work which studies operations under which these classes are closed (or not) [31, 44]. [31] proved the closure of #P and related classes under properties like iterated sum and product, the choose operation etc. There are other operations such as MAX, MIN, division by 2, and decrement under which it is not known whether #P is closed or not. In [44] implications and equivalences are established among these closure properties and certain other open questions in complexity theory. The closure properties have been studied for other counting classes as well.

In Chapter 6 we are able to settle most questions about these and other closure properties for the classes $\#AC^0$ and $GapAC^0$. For instance, they are not closed under MAX or division by 3, but they are closed under decrement; $\#AC^0$ is not closed under division by 2, although $GapAC^0$ is. In some cases, the answers follow easily from earlier results, but in other cases new analysis is required.

1.3.2 Counting Paths

Although arithmetic classes such as #L and $\#NC^1$ are defined in terms of arithmetic circuits, it is often nice to use equivalent definitions where we count paths in certain families of graphs. For instance, a complete problem for NL is the question of whether a directed acyclic graph has a path from vertex 1 to vertex n, and a complete problem for #L is to count the number of such paths. For any $k \geq 5$, a complete problem for NC^1 is to determine if a width-k directed acyclic graph has a path from vertex 1 to n, but it remains open whether counting the number of such paths is complete for $\#NC^1$. (See

[4] for a discussion of this problem.) Nonetheless, it was shown in [24] that a complete problem for the class GapNC^1 (the class of all functions that are the difference of two #NC^1 functions) is to compute, in a width-k graph where $k \geq 6$, the number of paths from vertex 1 to n minus the number of paths from vertex 1 to $n-1$.

The question is whether #AC^0 or GapAC^0 possess similar combinatorial characterizations. Note that certain lemmas and normal forms concerning these classes are fairly complicated to prove, whereas the analogous lemmas for larger classes such as #P, #L, and #NC^1 are much simpler because of those classes' path-based characterizations. The characterization of depth-k AC^0 presented in [18] in terms of the reachability problem for width-k grid graphs suggests the analogous conjecture that #AC^0 could be characterized by counting the number of paths connecting vertices 1 and n in bounded-width grid graphs.

In Chapter 7 we disprove this conjecture, showing that – even for width two graphs – this counting problem lies outside GapAC^0 and is complete for NC^1 (under ACC^0 reductions). In contrast, we are able to present a particular family of constant-width graphs such that counting paths in these graphs characterizes #AC^0.

Chapter 2

Definitions and Notation

2.1 Complexity Classes

First we list the definitions of some standard Boolean classes.

2.1.1 Classical Language and Counting Classes

Definition 1 AC^0 *is the class of languages recognized by a bounded depth Boolean circuit family of polynomial size over* $\wedge, \vee$, *and* $\neg$ *gates where there is no restriction on the indegree of the* $\wedge$ *and* $\vee$ *gates.*

The classes TC^0,$\mathrm{AC}^0[m]$, and ACC^0 are variants of this class. Thus we have the following definitions:

Definition 2 TC^0 *is the class of languages recognized by a bounded depth Boolean circuit family of polynomial size over* $\wedge, \vee$, *threshold, and* $\neg$ *gates where there is no restriction on the indegree of the* $\wedge$,$\vee$ *and threshold gates. Here a threshold gate on a certain number* n *of inputs returns 1 iff the number of 1's among its inputs exceeds a certain number* $k < n$, *where* k *is an arbitrary constant for the gate.*

Notice that it is not strictly necessary to include $\wedge, \vee$ gates in the above definitions as they can be obtained by setting k to $n-1$ and 0 in definition of a threshold gate.

Definition 3 $\mathrm{AC}^0[m]$, *for a positive integer* m, *is the class of languages recognized by a bounded depth Boolean circuit family of polynomial size over* $\wedge, \vee$, Mod_m *and* $\neg$ *gates where there is no restriction on the indegree of the* $\wedge, \vee$, *and* Mod_m *gates. Here a* Mod_m *gate refers to a gate that returns 0 iff the numbers of inputs which are 1, is divisible by* m *and 1 otherwise.*

$ACC^0 = \cup_m \mathrm{AC}^0[m]$. *In other words if we allow* Mod_m *gates for arbitrary integers* m.

The class NC^1 partly relaxes the depth requirement but has a bounded indegree requirement for gates:

Definition 4 NC^1 *is the class of languages recognized by a Boolean circuit family of polynomial size and logarithmically bounded depth over* $\wedge, \vee$, *and* $\neg$ *gates where the indegree of the* $\wedge$ *and* $\vee$ *gates is bounded by a constant.*

Let us also define the function classes FAC^0 and FTC^0 which are the function analogs of AC^0 and TC^0.

Definition 5 FAC^0 *is the class of functions of polynomial output length computed by a bounded depth Boolean circuit family of polynomial size over* $\wedge, \vee$, *and* $\neg$ *gates where there is no restriction on the indegree of the* $\wedge$ *and* $\vee$ *gates.* FTC^0 *is the corresponding class when we also permit threshold gates.*

For the sake of completeness let us define the complexity classes #P and #L:

Definition 6 #P *is the class of functions which map a string to the number of accepting paths on that string by a nondeterministic Turing machine running in polynomial time.* #L *is the corresponding class when the Turing machines run in logarithmic space.*

This brings us to the classes defined in this thesis.

2.1.2 Arithmetic Classes

Definition 7 *For any* $k > 0$, $\#\mathrm{AC}^0{}_k$ *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* $\#\mathrm{AC}^0 = \bigcup_{k>0} \#\mathrm{AC}^0{}_k$.[1]

[1] Tomo Yamakami [62] has defined $\#\mathrm{AC}^0$ somewhat differently, and his definition does not appear comparable to ours.

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 [31], and by analogy GapL was studied in [60, 8], and GapNC^1 was studied in [24]. 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:

$$\mathrm{P}\mathcal{C} = \{A | \exists f \in \mathrm{Gap}\mathcal{C},\ x \in A \iff f(x) > 0\},$$
$$\mathrm{C}_{=}\mathcal{C} = \{A | \exists f \in \mathrm{Gap}\mathcal{C},\ x \in A \iff f(x) = 0\}.$$

PP and PL were first studied in [34] and have been considered in many papers; C$_=$P was studied in [61] and elsewhere, and C$_=$L was studied in [6] (see also [50]). PNC^1 and C$_=$NC^1 were defined and studied in [24] (see also [41]).

A main result of this thesis 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 at first sight (although we show that both ways give rise to the *same* class PAC^0= C$_=$AC^0= TC^0).

Definition 8 DiffAC^0 *is the class of functions expressible as the difference of two* #AC^0 *functions.*

Definition 9 *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* $\mathrm{GapAC}^0 = \bigcup_{k>0} \mathrm{GapAC}^0{}_k$.

Recall that for all the classes $\mathcal{C} \in \{\mathrm{NC}^1, \mathrm{L}, \mathrm{SAC}^1, \mathrm{P}\}$, 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 $\mathcal{C}= \mathrm{AC}^0$.

The question, "Is $\mathrm{DiffAC}^0 = \mathrm{GapAC}^0$?", was answered in the affirmative by a clever proof in [13]. Since this was after the publication of some of the results reported in this thesis, we continue defining classes without assuming knowledge of the above result.

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

Definition 10 *The class* $\mathrm{C}_=\mathrm{AC}^0$ *(*$\mathrm{C}_=\mathrm{AC}^0_{circ}$*) 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 11 *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.

2.1.2.1 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 [19, 3], the mapping $n \mapsto C_n$ is computable

in polynomial time, for Logspace-uniform circuits [49], the mapping is computable in Logspace. Dlogtime-uniformity requires a somewhat more careful definition; we refer the reader to [16]. 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.

2.2 Operations on Functions

We define certain operations on functions which in a later chapter (Chapter 6) will be seen to constitute closure or nonclosure properties of $\#\mathrm{AC}^0$ and GapAC^0.

Definition 12 *The weak sum of a number of functions f_i (for $i = 1 \ldots m$) mapping $\{0,1\}^n$ to the integers, where m is a polynomial in n, is the function $\sum_{i=1}^{m} f_i$. Weak product is analogously defined as $\prod_{i=1}^{m} f_i$.*

Notice that the term "weak" refers to the assumption that m is polynomial in n.

Two simple operations MAX and MIN are defined next.

Definition 13 *If f, g are two functions mapping $\{0,1\}^n$ to the integers, then the function $\mathrm{MAX}(f,g)$ ($\mathrm{MIN}(f,g)$) maps a string $x \in \{0,1\}^n$ to the integer $max(f(x), g(x))$ ($min(f(x), g(x))$).*

Finally, we define the operation "monus", a variant of minus.

Definition 14 *If f, g are two functions mapping $\{0,1\}^n$ to the integers, then the monus of the two functions, $f \dot{-} g$ maps a string $x \in \{0,1\}^n$ to the integer $max(f(x) - g(x), 0)$.*

Chapter 3

Normal Forms

In this chapter, we obtain a number of normal forms for $\#\mathrm{AC}^0$ and GapAC^0. These help simplify some of the proofs in later chapters, and are of independent interest.

3.1 Preliminaries

Proposition 1 $\mathrm{FAC}^0 \subseteq \#\mathrm{AC}^0$.

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.*

Proof: Let C_r be the circuit $\wedge_{i=1}^{r}(1 \vee 1)$ which has $\prod_{i=1}^{r}(1+1)$ 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 $f(x) \in \#\mathrm{AC}^0$ because the number of accepting subtrees is $\sum_i b_i 2^i$. ∎

3.2 Quasi-complementation

Our first normal-form theorem is an analog of a statement that is trivially true for the classes #P and #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 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,

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

∎

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 $B(n) = \{j \mid \text{bit number } j \text{ of the binary representation of } c(n) \text{ is equal to } 1\}$. 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.

Chapter 4

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

4.1 Non-uniform and P-uniform settings

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

Theorem 7 *P-uniform* $\mathrm{TC}^0 \subseteq$ *P-uniform* $\mathrm{C}_{=}\mathrm{AC}^0$ *and Dlogtime-uniform* $\mathrm{TC}^0 \subseteq$ *Dlogtime-uniform* $\mathrm{C}_{=}\mathrm{AC}^0_{circ}$.

Proof: We will need to use the following well-known fact (see e.g. [45]),

Fact 8 *A problem is in* 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).*

We will present a polynomial time algorithm that proceeds by induction on the depth of a TC^0-circuit C (composed only of $ET^m_{m/2}$ gates) and constructs a DiffAC^0 (or Dlogtime-uniform GapAC^0) circuit f, such that, if $C(x) = 0$ then $f(x) = 0$, and if $C(x) = 1$ then $f(x)$ is equal to a constant independent of x. The following paragraph provides details for the base case of depth 1 circuits.

Let K_m be $\prod_{j=0, j\neq m/2}^{m} (m/2 - j)$. 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 $P(X) = \prod_{j\neq m/2} (X - j)$. The naïve algorithm that multiplies the terms $(X - j)$ together to

explicitly compute the coefficients of powers of X runs in polynomial time. (Note that the binary representations of the coefficients are only polynomially long). Separating the positive and negative terms we get $P(X) = Q(X) - R(X)$, where $Q(.)$ and $R(.)$ are polynomials with coefficients that can be computed by P-uniform $\#\mathrm{AC}^0$ circuits, thus we get a P-uniform DiffAC^0 function that is equal to 0 if $C(x) = 0$, and is equal to K_m if $C(x) = 1$. Also, $\prod_{j \neq m/2} ((\sum_i x_i) - j)$ is already a Dlogtime-uniform GapAC^0 function with this property.

For the inductive step, in order to prove P-uniform $\mathrm{TC}^0 \subseteq$ P-uniform $\mathrm{C}_=\mathrm{AC}^0$, our inductive hypothesis will be: for every P-uniform TC^0 family $\{C_n\}$, for every d, there is a function computable in time polynomial in n that, on input (n, C_n) outputs circuits $D_n, D_n^{'}$ of depth $O(d)$ such that if C is a gate at level d of C_n then for all x of length n, $C(x) = 0$ implies $D_n(x) - D_n^{'}(x) = 0$ and $C(x) = 1$ implies $D_n(x) - D_n^{'}(x) = (K_m)^t$ (where $t = t(m, d)$ is some function depending only on d and m). Note that we have established this claim for the case $d = 1$.

Consider a depth $d+1$ exact-threshold circuit with output gate $\mathcal{G}$, where the inputs to $\mathcal{G}$ are $\mathcal{G}_i$ $(i = 1 \ldots m)$. We show how to construct, in polynomial time, a DiffAC^0 function that takes values $(K_m)^{t(m,d+1)} = (K_m)^{mt(m,d)+1}$ and 0 whenever $\mathcal{G}$ outputs 1 and 0 respectively.

Let the 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

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

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

$$= \frac{\prod_{j\neq m/2}\left(\left(\sum_i h_i\right)-m2^q-jK_m^t\right)}{K_m^{mt+1}}$$

$$= \frac{Q'\left(\sum_i h_i\right)-R'\left(\sum_i h_i\right)}{K_m^{mt+1}}.$$

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 $\#\mathrm{AC}^0$ circuits constructed by our algorithm.

To prove Dlogtime-$\mathrm{TC}^0 \subseteq \mathrm{C}_=\mathrm{AC}^0_{circ}$ 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 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 7. ∎

Proposition 9 $\mathrm{C}_=\mathrm{AC}^0 \subseteq \mathrm{PAC}^0$ *(under all considered notions of uniformity).*

Proof: Let A be in $\mathrm{C}_=\mathrm{AC}^0$, and let f and g be $\#\mathrm{AC}^0$ functions such that $x \in A$ iff $f(x)=g(x)$. Then consider $(f(x)-g(x))^2 = (f(x))^2+(g(x)))^2-2f(x)g(x)$. Let $g_1(x) = (f(x))^2+(g(x)))^2$ and $f_1(x) = 1+2f(x)g(x)$ where clearly both f_1, g_1 are $\#\mathrm{AC}^0$ functions. Clearly $x \in A$ iff $f(x)=g(x)$ iff $f_1(x) > g_1(x)$, completing the proof.

∎

Proposition 10 *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 [48].)

Corollary 11 *In the P-Uniform and Non-Uniform Settings,*

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

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 were not able to prove[1] that $\mathrm{DiffAC}^0 = \mathrm{GapAC}^0$, we obtained 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 12 *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 7, 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 [31]. We refer the reader to [31] for further details.)

Corollary 12 is probably nearly the strongest result in this direction that one can prove. For instance, one might seek to strengthen Corollary 12 to obtain $g(n) = 1$. (This corresponds to the AC^0-analog of the class SPP studied in [31].) Note that if

[1] This was proved in [13]. Also see Section 4.3.

$g(n) = 1$, then the characteristic function of A is in GapAC^0. However, it follows from Proposition 25 that any such language A is in AC^0[2]. Thus the lower bound of [46] (showing that MAJORITY is not in AC^0[2]) shows that we cannot improve Corollary 12 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 12 so that $g(n)$ is of the form 2^{n^k} (for example). Then it would follow that TC^0 = 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 24, this property can be checked in AC^0[6]. More generally, if Corollary 12 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 section.

4.2 Logspace-uniform Setting

Theorem 13 *Logspace-uniform* PAC$^0_{circ} \subseteq$ *Logspace-uniform* TC^0.

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

Definition 15 *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 14 *(see e.g. [36]) Given primes $p_1, p_2, \ldots, p_k$ and an integer x, there are unique integers $x_1, x_2, \ldots, x_k$ (modulo $p_1, p_2, \ldots, p_k$ respectively), satisfying*

$$x \equiv x_1 \bmod p_1,$$
$$x \equiv x_2 \bmod p_2,$$
$$\vdots$$
$$x \equiv x_k \bmod p_k.$$

And if any y satisfies the congruences above then $x \equiv y \bmod P_k$, where $P_k = \prod_{i=1}^{k} p_i$. More explicitly, $x = A_k - q_k P_k$ where

$$A_k = \sum_{j=1}^{k} \left((x_j c_{k,j}) \bmod p_j \right) P_k/p_j,$$

$$c_{k,j} = (P_k/p_j)^{-1} \bmod p_j,$$

$$q_k = \lfloor A_k/P_k \rfloor .$$

And the following variant of the prime-number theorem:

Theorem 15 *(see e.g. [36]) For sufficiently large values of n, the product of all primes less than n exceeds 2^n.*

As a consequence of Theorems 14 and 15 we get the following corollary:

Corollary 16 *If $0 \leq x < 2^n$, 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 $1 \leq i \leq s$) 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 17 *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 14 and Corollary 16. 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

$$X_k = A_k - q_k P_k.$$

Thus we need to find out whether or not $X_k > P_k/2$, which is equivalent to

$$\begin{aligned} \frac{1}{2} &< \frac{X_k}{P_k} \\ &= \frac{A_k - q_k P_k}{P_k} \\ &= \frac{A_k}{P_k} - \left\lfloor \frac{A_k}{P_k} \right\rfloor \end{aligned}$$

So, essentially, we need to find out the first bit of the fractional representation of A_k/P_k. [29] 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 $1 \leq j \leq i \leq k$) and find the sums $q_i'(x) = \sum_{j=1}^{i} t_{i,j}(x)$ approximating A_i/P_i. [29] 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 $x' \geq P_k/2$, 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 [29] 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 19-7,8 below. ∎

(We remark that, instead of relying on [29], it is also possible to make use of similar results of [39, 27]. It seems to us that the construction of [29] results in a simpler circuit.)

Lemma 18 *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 15) where g is as in item 1 above.*
3. *The function $z, p \mapsto ind_{g,p}(z)$ (see Definition 15) 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 18-1) 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 18-2) Given a number z compute g^z, reducing the result g^i modulo p at each step.
- (of Lemma 18-3) 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 18-4) With numbers in this range, the standard long-division algorithm runs in logspace.
- (of Lemma 18-5) 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 18-6) Compute p_j, and then compute $c_{i,j}$ using Lemma 18-5 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 19 *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* $f : \{0,1\}^{c \log n} \rightarrow \{0,1\}^m$ *is a Logspace computable function and* $g : \{0,1\}^n \rightarrow \{0,1\}^{c \log n}$ *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 19-1) 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 19-2) Straightforward.
- (of Lemma 19-3) This is well known (e.g. see [25]).
- (of Lemma 19-4) First compute $v_i = 2^i \mod k$ for $1 \leq i \leq n$. Next compute the sum of $x_j[i]v_i$ (for all i, j) using Lemma 19-3 above. Now applying 19-1, with $f = mod_k$ and g as the function just defined, we get this result.
- (of Lemma 19-5) Let $g = ind_g \circ mod_p$. Then $prod_p = pow_g \circ sum_{p-1} \circ \vec{g}$. Result follows from Lemmas 18-4, 18-2, 18-3,19-3, 19-2, and 19-1.
- (of Lemma 19-6) Follows from 18-6 and 19-1.
- (of Lemma 19-7) Follows from 19-6 and 19-3.
- (of Lemma 19-8) 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 13) 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 $\#\text{AC}^0 \subseteq \#\text{P}$.) We use Lemmas 19-4 and Lemmas 19-5 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 17. ∎

Corollary 20 *In the Logspace-Uniform Setting,*

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

4.3 Recent Developments

Notice that in the light of recent developments there is no need to distinguish between $\mathrm{C_{=}AC^0}$ and $\mathrm{C_{=}AC^0_{\mathit{circ}}}$ (and similarly $\mathrm{PAC^0}$and $\mathrm{PAC^0_{\mathit{circ}}}$), since from [13] we know that

Theorem 21 *[13]* $\mathrm{DiffAC^0} = \mathrm{GapAC^0}$ *under all considered notions of uniformity.*

We provide a sketch of the proof for the sake of completeness.

Proof:(Sketch) The aim is to express a constant depth arithmetic circuit, with leaf inputs from $\{0, 1, -1\}$, as the difference of two constant depth arithmetic circuits with the inputs drawn from $\{0, 1\}$. The proof proceeds by induction on the depth of the tree. The base case is trivial and so is the case when the gate is a $+$-gate. The latter because the sum of a number of differences is the difference of two sums. The tricky case is when the gate is a $\times$-gate. Thus it would be sufficient to express $\prod_{i=1}^{m}(a_i - b_i)$ as the difference of two positive arithmetic circuits with inputs drawn from $\{0, 1\} \cup \bigcup_i \{a_i, b_i\}$. The ingenuity in the proof is to attempt to write the product above as a linear combination of $\prod_{i=1}^{m}(a_i + kb_i)$ as k varies. In other words try to establish the identity:

$$\prod_{i=1}^{m}(a_i - b_i) = \sum_{k=1}^{m+1} c_k \prod_{i=1}^{m}(a_i + kb_i)$$

by discovering integral c_k's depending only on m and k. Classifying the c_k's as positive and negative the product on the left can be written as the difference of two sums, each of which consists of a product of linear expressions in a's and b's.

To find the c_k's, compare the coefficients of a typical monomial consisting of $m - j$ factors which are a's and j factors which are b's, for a j lying between 0 and m. This yields, for $0 \leq j \leq m$

$$(-1)^j = \sum_{k=1}^{m+1} k^j c_k.$$

But these are $m+1$ linear equations in the $m+1$ variables c_k. Solving them by Cramer's rule, each c_k is the ratio of two Vandermonde determinants - the denominator being

the determinant of a matrix with $(a,b)^{th}$ entry being b^{a-1} $(1 \leq a, b \leq m+1)$ and the numerator being the determinant of the same matrix with the k^{th} column being replaced by powers of -1. The properties of Vandermonde determinants and some algebra yield the following expression for c_k:

$$c_k = (-1)^{k-1} k \begin{pmatrix} m+2 \\ k+1 \end{pmatrix}.$$

To see that the circuit obtained by the inductive method above is indeed uniform, it suffices to show that $\begin{pmatrix} n \\ k \end{pmatrix}$, can be computed in uniform $\#\mathrm{AC}^0$, given n and k in unary. The paper [13] goes on to show that these are indeed in Logtime-uniform $\#\mathrm{AC}^0$. The proof hinges on finding the maximum power of p, which divides the binomial coefficient above, for each prime p less than n. But it is well known that p^j divides $\begin{pmatrix} n \\ k \end{pmatrix}$ iff

$$\left\lfloor \frac{n}{p^j} \right\rfloor > \left\lfloor \frac{n-k}{p^j} \right\rfloor + \left\lfloor \frac{k}{p^j} \right\rfloor.$$

But since n, k are in unary and $p^j \leq n$, this test is in Logtime-uniform $\#\mathrm{AC}^0$. Realizing that testing whether p is a prime (for small p) is in Logtime-uniform $\#\mathrm{AC}^0$ completes the proof. ∎

Further, notice that recent advances which place the computation of the iterated product of integers in DLOGTIME-uniform TC^0, lead to a simplification of the proofs above. In fact, they yield the following theorem:

Theorem 22 *[38] All functions in DLOGTIME-uniform* $\#\mathrm{AC}^0$ *can be computed in DLOGTIME-uniform* TC^0. *Thus the equalities* $\mathrm{C}_=\mathrm{AC}^0 = \mathrm{PAC}^0 = \mathrm{PAC}^0_{circ} = \mathrm{C}_=\mathrm{AC}^0_{circ} = \mathrm{TC}^0$ *all hold in the DLOGTIME-uniform setting.*

Chapter 5

Arithmetic Circuits over Finite Fields

5.1 Connections with ACC^0

There has been earlier work characterizing TC^0 in terms of finite fields [23, 32, 48]. 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 × 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 that this is in some sense implicit in the work of Smolensky [52].)

We need the following fact from number theory.

Theorem 23 *(Dirichlet) (see e.g. [36]) 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 $\#AC^0_F$ denote the class of functions computed by $\#AC^0$ circuits, where now + and × are operations over the field F.

Theorem 24 *A language is in ACC^0 if and only if its characteristic function is in $\#AC^0_F$ for some finite field F.*

Proof: Let A be a language in ACC^0; thus A is in $AC^0[m]$ 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 23) 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 $\text{GF}(p^k)$. We will use two representations. One representation $\text{rep}_+(x)$ 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 $\text{GF}(p)$. Note that, when given n such k-tuples, their sum can easily be computed by $\text{AC}^0[p]$ circuits. (When adding up the l^{th} components, test for each $j \leq p$ if $\text{Mod}_p(rep_+(x_{1,l}) \cdots rep_+(x_{n,l})1^j)$ holds. If so, then output $1^{p-j}0^j$ as the value of the l^{th} component.)

The other representation $\text{rep}_\times(x)$ 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 $\sum_{l_i} \bmod (p^k - 1)$ (where l_i is such that $g^{l_i} = x_i$). But this is equivalent to computing, $\text{Mod}_{p^k-1}(\text{rep}_\times(x_1) \ldots \text{rep}_\times(x_n)) \bmod (p^k - 1)$, which can clearly be computed in $\text{AC}^0[p^k - 1]$. This completes the proof. ∎

5.2 $\text{AC}^0[2]$ and Circuits over GF(2)

Although in general there is no close connection between arithmetic circuits over $\text{GF}(p)$ and $\text{AC}^0[p]$ (since, for example, both PARITY and Mod_3 are computable with arithmetic circuits over 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 25 *The following are equivalent:*

1. $A \in \mathrm{AC}^0[2]$.
2. $\chi_A \in \mathrm{GapAC}^0$.
3. $\chi_A \in \mathrm{DiffAC}^0$.
4. χ_A *can be represented as the low-order bit of some* $\#\mathrm{AC}^0$ *function.*
5. $\chi_A \in \#\mathrm{AC}^0_{GF(2)}$.

Proof: Out of the implications $1 \implies 2 \implies 3 \implies 4 \implies 5 \implies 1$, the implication $2 \implies 3$ follows from the equivalence of DiffAC^0 and GapAC^0 established in [13] (see Chapter 4 for a proof sketch). For an explanation of $3 \implies 4$, see the discussion after Corollary 6. The other implications are immediate except for $1 \implies 2$, for which we give a proof below.

The 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:

$$\left(\begin{array}{c} 1 + \prod(1 - 2f(x,i)) \\ 2 \end{array} \right).$$

This is in GapAC^0 by the closure properties established in Chapter 6.

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 $\mathrm{AC}^0[p(p^k - 1)]$.

Chapter 6

Closure and Non-Closure Properties

6.1 Introduction

A study of the closure properties of #P was initiated by Ogiwara and Hemachandra in [44]. It is not known whether #P is closed under such operations as MAX, MIN, division by 2, and decrement. In [44] implications and equivalences are established among these closure properties and certain other open questions in complexity theory. In the context of $\#AC^0$ and $GapAC^0$, however, we are able to settle most questions about these and other closure properties. (For instance, they are not closed under MAX or division by 3, but they are closed under decrement; $\#AC^0$ is not closed under division by 2, although $GapAC^0$ is.) In some cases, the answers follow easily from earlier results, but in other cases new analysis is required.

We summarize the closure and non-closure properties for the classes $\#AC^0$ and $GapAC^0$.

Property	$\#AC^0$	$GapAC^0$
weak sum	$\surd$	$\surd$
weak product	$\surd$	$\surd$
choose constant	$\surd$	$\surd$
choose polylogarithmic	?	?
choose super-polylogarithmic	$\times$	$\times$
MAX, MIN	$\times$	$\times$
monus polylogarithmic	$\surd$	$\surd$
monus super-polylogarithmic	$\times$	$\times$
div with polylogarithmic numerator	$\surd$	$\surd$
div constant (non power of 2)	$\times$	$\times$
div 2^α	$\times$	$\surd$
square-root, logarithm	$\times$	$\times$

6.2 Weak Sum and Weak Product

Recall from Chapter 2 that weak sum (product) refers to a sum (product) of polynomially many functions. The following proposition is easy to see

Proposition 26 $\#AC^0$ *and* $GapAC^0$ *are closed under weak sum and weak product.*

6.3 Choose Operator

In contrast to the foregoing proposition, 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 27 $\#AC^0$*and* $GapAC^0$ *are closed under the choose operation (i.e. if* $f(x)$ *is a function in either of these classes, then so is* $\binom{f(x)}{k}$ *for any positive constant* k*).*

Proof:

We proceed by induction on the depth of the counting circuit computing the $\#\mathrm{AC}^0$ 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 $\#\mathrm{AC}^0$ functions and for some constant k and all $j \leq k$,

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

are $\#\mathrm{AC}^0$ 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, namely, 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 $\#\mathrm{AC}^0$, 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 of 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 π_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 28 $R(n,k) \leq \pi_1\pi_2\cdots\pi_k n^{k-1}$.

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,

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

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 27 for the $\#\mathrm{AC}^0$ 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 $\#\mathrm{NC}^1$ is closed under the choose operation – but (as was pointed out to us by David Mix Barrington) a much simpler proof suffices for $\#\mathrm{NC}^1$, since one can show that $\#\mathrm{NC}^1$ is closed under the "monus" and "div m" operations for any constant m. (That is, if f is in $\#\mathrm{NC}^1$, then so are the functions $\max(f(x)-1, 0)$, and $\lceil f(x)/m \rceil$.) Although one can show that $\#\mathrm{AC}^0$ is also closed under the monus operation, $\#\mathrm{AC}^0$ and GapAC^0 are not closed under div m for any $m \neq 2^\alpha$ (see Theorem 36).

This is a good time to observe that $\#\mathrm{AC}^0$ is *not* closed under some more general choose operations. For instance, let $f(x) = \sum_i x_i$; f is clearly in $\#\mathrm{AC}^0$. For any

function $g(n) \neq \log^{O(1)} n$, the function $\begin{pmatrix} f(x) \\ g(n) \end{pmatrix}$ is not in $\#\mathrm{AC}^0$, 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 [30, 28]. (This shows merely that $\begin{pmatrix} \sum_i x_i \\ g(n) \end{pmatrix}$ is not in $\#\mathrm{AC}^0$; for an improvement of this result to a lower bound for GapAC^0, see Theorem 29.) 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 [30, 28], but the currently-known proofs of this fact do not preserve the number of accepting subtrees.

Open Question 1 *Are the functions* $\begin{pmatrix} \sum_i x_i \\ \log n \end{pmatrix}$ *and* $\begin{pmatrix} \sum_i x_i \\ \log^* n \end{pmatrix}$ *in* $\#\mathrm{AC}^0$*?*

It is perhaps worth noting that the proof above actually shows that for functions g computable in AC^0, both of the classes $\#\mathrm{qAC}^0$ and GapqAC^0 are closed under $\binom{\cdot}{g}$ if g is polylogarithmic[1]. This is related to the following open question in [40].

Open Question 2 *Is the set of symmetric functions in* $\mathrm{qAC}^0[2]$ *the same as the set of symmetric functions in* $\mathrm{AC}^0[2]$*? Equivalently, does* $\begin{pmatrix} \sum_i x_i \\ k \end{pmatrix} \mod 2$ *belong to* $\mathrm{AC}^0[2]$*, for each polylogarithmic value of* k*?*

The "only if" part corresponding to the "if" statement above follows from the following theorem.

Theorem 29 *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*

$$\begin{pmatrix} \sum_{i=1}^{n} x_i \\ j \end{pmatrix}$$

[1]The complexity classes qAC^0,$\mathrm{qAC}^0[2]$, GapqAC^0, $\#\mathrm{qAC}^0$ are counterparts of AC^0, $\mathrm{AC}^0[2]$, GapAC^0, $\#\mathrm{AC}^0$ when circuits are allowed to have *quasi-polynomial*$(2^{\log^c n})$ size instead of polynomial size.

where the x_i's are Boolean variables. (In particular, for any superpolylogarithmic function $g(n)$, it is not the case that a 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 GapAC$^0{}_k$-circuit of size at most $n^{\log^{k-1} n}$ computing $\binom{\sum_i x_i}{j}$.

Using these 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 $k+1$ AC^0[2] circuits of size smaller than the $2^{\Omega(m^{1/2(k+1)})}$ lower bound proved in [52].

We need the following fact (a proof of which can be found in [21, 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 GapAC$^0{}_k$ circuit of size $n^{\log^{k-1} n}$ computing $\binom{\sum_i x_i}{j}$, and thus, by Proposition 25 and the discussion following Corollary 6 the lower-order bit of this expression can be computed by 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 [52].

∎

6.4 MAX and MIN

Following [44], let us consider a very simple closure property: MAX.

Theorem 30 *Neither* $\#\mathrm{AC}^0$ *nor* GapAC^0 *is closed under* MAX.

Proof: Let $f(x) = \sum_i x_i$, and let $g(x) = \lfloor |x|/2 \rfloor$. Let x' denote the result of changing the first 1 in x to a 0 (if such a bit exists). (It is easy to see that x' can be computed from x in Boolean AC^0, and hence this function is also in $\#\mathrm{AC}^0$.) Note that the number of 1's in x is less than or equal to $\lfloor |x|/2 \rfloor$ if and only if the low-order bits of $\mathrm{MAX}(f(x), g(x))$ and $\mathrm{MAX}(f(x'), g(x'))$ are equal. The low-order bits of any GapAC^0 function are computable in $\mathrm{AC}^0[2]$, and hence if $\mathrm{MAX}(f, g)$ were computable in GapAC^0, it would follow that the majority function could be computed in $\mathrm{AC}^0[2]$, in contradiction to [46].

∎

By an essentially identical argument we get the following:

Corollary 31 *Neither* $\#\mathrm{AC}^0$ *nor* GapAC^0 *is closed under* MIN.

6.5 Monus

Again following [44], we next consider the decrement function. Given a function f, the decrement operation applied to f is $f \dot{-} 1$ (where the *monus* operation $a \dot{-} b$ is equal to 0 if $b \geq a$, and is equal to $a - b$ otherwise). Not only is $\#\mathrm{AC}^0$ closed under decrement, but it is closed under $\dot{-}$ with any $\#\mathrm{AC}^0$ function whose growth rate is at most polylogarithmic.

Theorem 32 *If* f *and* g *are in* $\#\mathrm{AC}^0$*, and there exists a* k *such that for all* x*,* $g(x) = O(\log^k |x|)$*, then* $f \dot{-} g$ *is in* $\#\mathrm{AC}^0$*.*

Note: The polylogarithmic bound on g is necessary. To see this, let $f(x) = \sum x_i$, and let $g(x)$ be $|x|/2$ (or any other superpolylogarithmic threshold). Then, $f(x) \dot{-} g(x)$ is nonzero if and only if the number of ones in x exceeds the threshold $g(x)$. If this function were in $\#\mathrm{AC}^0$, it would imply the existence of a Boolean AC^0 circuit family computing threshold-g, contradicting [30, 37]. An argument similar to Theorem 30 shows that $f \dot{-} g$ is not even in GapAC^0.

Proof: The proof is based on Lemma 34 below.

Definition 16 *Let $\#_{t,n}$ be the following integer function on n Boolean inputs:*

$$\#_{t,n}(x_1, \ldots, x_n) = \begin{cases} |\{x_i : x_i = 1\}| & \text{if } |\{x_i : x_i = 1\}| \leq t \\ t & \text{otherwise} \end{cases}$$

The following lemma follows directly from [28, 30]:

Lemma 33 *([28, 30]) If $t = O(\log^k n)$ for some fixed k then the function $\#_{t,n}$ is in AC^0 (and hence in $\#\mathrm{AC}^0$).*

Lemma 34 *If f is a $\#\mathrm{AC}^0$ function and g is a function in $\#\mathrm{AC}^0$ taking polylogarithmically bounded values, then the predicates $[f = g]$ and $[f \leq g]$ are computable in AC^0.*

Proof: Let r be a polylogarithmic upper bound on g. We construct an AC^0-circuit by induction on the height of the $\#\mathrm{AC}^0$-circuit computing f, using Lemma 33. Let $v(C)$ denote the value of a gate C. If C is a $\sum$-gate with inputs $C_1, \ldots, C_n$ then $v(C) = g$ iff

$$\sum_{j=1}^{r} j \cdot \left(\#_{r,n}\left([v(C_1) = j], \ldots, [v(C_n) = j]\right)\right)$$

is equal to g and each $v(C_i)$ is at most r. We can inductively compute $[v(C_i) = j]$ for $j \leq r$ and $[v(C_i) \leq r]$. Similarly, if C is a $\prod$-gate then $v(C) = g$ iff

$$\prod_{j=2}^{r} \sum_{l \leq \log_j r} j^l \left[l = \#_{\log_j r,n}\left([v(C_1) = j], \ldots, [v(C_n) = j]\right)\right]$$

is equal to g and each $v(C_i)$ is at most r. Notice that if $l \leq \log_j r$, then $j^l \leq r$ and thus the transformation $(j,l) \mapsto j^l$ can be done via a lookup table. This completes the inductive proof. ∎

Continuing with the proof of Theorem 32, we build the circuit for $f \dot{-} g$ by induction on the depth of the circuit computing f. The basis, for depth-zero circuits, is trivial. For the inductive step, consider first the case where the output gate is a + gate. Note that

$$(\sum_{i=1}^{n} f_i) \dot{-} g = \sum_{i=1}^{n} \left[\sum_{j=1}^{i-1} f_j \leq g < \sum_{j=1}^{i} f_j \right] \left(f_i \dot{-} (g - \sum_{j=1}^{i-1} f_j) + \sum_{j=i+1}^{n} f_j \right).$$

It follows from Lemma 34 that $g - \sum_{j=1}^{i-1} f_j$ can be computed in AC^0 (by testing, for small values of a, whether $a + \sum_{j=1}^{i-1} f_j \leq g$). Thus the claim follows by application of Lemma 34 and by closure under sum and product.

Now consider the case where the output gate is $\times$. By Lemma 34, we first check that $\prod_{i=1}^{n} f_i \geq g$ (if not we output 0). Otherwise there are two cases. In one case, some f_i is greater than g (and the minimum such i can be identified using Lemma 34), in which case $\prod_{i=1}^{n} f_i \dot{-} g$ is $A_i = f_i \left((\prod_{j \neq i} f_j) - 1 \right) + f_i - g$. Otherwise, all f_i's are less than g, in which case we can find the minimum i such that $\prod_{j=1}^{i} f_j \geq g$, and the desired monus is

$$B_i = \left(\prod_{j=1}^{i} f_j \right) \left(\left(\prod_{j=i+1}^{n} f_j \right) - 1 \right) + \left(\prod_{j=1}^{i} f_j - g \right).$$

To see that A_i can be computed using $\#\mathrm{AC}^0$-circuits, notice that $f_i - g$ can be computed inductively and $\prod_{j \neq i} f_j - 1$ can be written as a telescoping series, $\sum_{k=1, k \neq i}^{n} ((f_k - 1) \prod_{j=k+1}^{n} f_j)$, where the $f_k - 1$'s can be computed inductively. As for B_i, notice that $\prod_{j=i+1}^{n} f_j - 1$ can be computed as a telescoping series as for A_i. Now, $f_i \leq g$ and from the minimality of i, $\prod_{j=1}^{i-1} f_j \leq g$. Thus both $\prod_{j=1}^{i} f_j$ and g are polylogarithmic, so their difference can be computed using a $\#\mathrm{AC}^0$ circuit. This completes the proof of Theorem 3. ∎

6.6 Div

A similar argument allows us to show the following:

Lemma 35 *If f is any $\#\mathrm{AC}^0$ function and g is a function in $\#\mathrm{AC}^0$ taking polylogarithmically bounded values, then the function $\lfloor g/f \rfloor$ is in $\#\mathrm{AC}^0$.*

Proof: Let r be an upper bound on g, then $\lfloor g/f \rfloor = \sum_{k=1}^{r} k\,[(k-1)g < f \leq kg]$. Since the predicate $[(k-1)g < f \leq kg]$ is computable in AC^0, the entire computation can be done in $\#\mathrm{AC}^0$. ∎

This leads to the question of whether $\lfloor g/f \rfloor$ is in $\#\mathrm{AC}^0$ when we do *not* have a polylogarithmic upper bound on g. We give a negative answer by showing the following lower bound.

Theorem 36 *For any integer m that is not a power of 2, the function $\left\lfloor \frac{\sum_i x_i}{m} \right\rfloor$ cannot be computed in* GapAC^0.

Proof: It suffices to prove the following special case: For any odd prime p, the function $\left\lfloor \frac{\sum_i x_i}{p} \right\rfloor$ cannot be computed in GapAC^0.

To see this, note that if m is not a power of 2, then there exists an odd prime p and an integer m_1 such that $m = p \cdot m_1$. The observation follows by considering

$$\left\lfloor \frac{\sum x_i}{p} \right\rfloor = \left\lfloor \frac{(\sum x_i) \cdot m_1}{p \cdot m_1} \right\rfloor.$$

Now suppose that we can compute $\left\lfloor \frac{\sum x_i}{p} \right\rfloor$ in GapAC^0. Then the low-order bit of

$$(\sum_i x_i) - p \cdot \left\lfloor \frac{\sum x_i}{p} \right\rfloor$$

is in GapAC^0 too. But this value is the low-order bit of the remainder of dividing $\sum x_i$ by p, and thus this has period p, in contradiction to [40]. ∎

The situation for powers of 2 is more complicated. GapAC^0 is closed under such divisions, but $\#\mathrm{AC}^0$ is not.

Theorem 37 *For any integer constant α and any function $F(x) \in \mathrm{GapAC}^0$ the function $\lfloor \frac{F(x)}{2^\alpha} \rfloor$ is computable in* GapAC^0.

Proof:

We first consider the case of $\alpha = 1$. Since $F \in \text{DiffAC}^0$, there exist two functions $f, h \in \#\text{AC}^0$ such that $F(x) = f(x) - h(x)$. Denote by $\textsc{Par}(f(x))$ the low-order bit of the binary representation of $f(x)$. It follows from Proposition 25 that $\textsc{Par}(f(x))$ can be computed in GapAC^0. The following formula can be easily verified:

$$\left\lfloor \frac{f(x) - h(x)}{2} \right\rfloor = \left\lfloor \frac{f(x)}{2} \right\rfloor - \left\lfloor \frac{h(x)}{2} \right\rfloor - [1 - \textsc{Par}(f(x))] \cdot \textsc{Par}(h(x)). \tag{6.1}$$

Therefore, it is enough to show that if $f(x) \in \#\text{AC}^0$, then we can build a GapAC^0 circuit computing $\left\lfloor \frac{f(x)}{2} \right\rfloor$.

Suppose that $f(x)$ is computed by a $\#\text{AC}^0$ circuit C of depth d. We will show this construction by induction on d. Let g be the output gate of C, having fanin m, where $g_1, \cdots, g_m$ are the input gates of g. Note that m is polynomial in n. Let $C_1, \ldots, C_m$ be subcircuits of C, whose output gates are $g_1, \cdots, g_m$ respectively. For each i call $g_i(x)$ the function computed at the gate g_i.

If $d = 1$ then $g_i(x) \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n, 0, 1\}$. If g is a $\times$ gate then clearly $\left\lfloor \frac{g_1(x) \cdots g_m(x)}{2} \right\rfloor = 0$, which is computable in $\#\text{AC}^0$ and hence in GapAC^0. If g is a $+$ gate, then $\left\lfloor \frac{g_1(x) + \cdots + g_m(x)}{2} \right\rfloor$ is in GapAC^0 using the identity:

$$\left\lfloor \frac{x_1 + \cdots + x_n}{2} \right\rfloor = \textsc{Par}(x_1) \cdot x_2 + \textsc{Par}(x_1, x_2) \cdot x_3 + \cdots + \textsc{Par}(x_1, x_2, \ldots, x_{n-1}) \cdot x_n$$

(where $\textsc{Par}(x_1, x_2, \ldots, x_r) = \textsc{Par}(\sum_{i=1}^{r} x_i)$) and the fact that the function $\textsc{Par}()$ is in GapAC^0 (see Proposition 25).

Now suppose that $d > 1$ and that for all subcircuits $C_1, \ldots, C_m$ of depth $\leq d - 1$ we have already constructed corresponding GapAC^0 circuits computing $\left\lfloor \frac{g_i(x)}{2} \right\rfloor$. If g is a $+$ gate, that is $g(x) = g_1(x) + \cdots + g_m(x)$, then

$$\left\lfloor \frac{g(x)}{2} \right\rfloor = \left\lfloor \frac{g_1(x)}{2} \right\rfloor + \cdots + \left\lfloor \frac{g_m(x)}{2} \right\rfloor + \left\lfloor \frac{\textsc{Par}(g_1(x)) + \cdots + \textsc{Par}(g_m(x))}{2} \right\rfloor \tag{6.2}$$

If g is a $\times$ gate, that is $g(x) = g_1(x) \cdots g_m(x)$, then

$$\begin{aligned}
\left\lfloor \frac{g(x)}{2} \right\rfloor &= \left\lfloor \frac{g_1(x)}{2} \right\rfloor \cdot g_2(x) \cdots g_m(x) + \textsc{Par}(g_1(x)) \cdot \left\lfloor \frac{g_2(x) \cdots g_m(x)}{2} \right\rfloor \\
&= \left\lfloor \frac{g_1(x)}{2} \right\rfloor \cdot g_2(x) \cdots g_m(x) + \textsc{Par}(g_1(x)) \cdot \left\lfloor \frac{g_2(x)}{2} \right\rfloor g_3(x) \cdots g_m(x) \\
&\qquad + \textsc{Par}(g_1(x)) \cdot \textsc{Par}(g_2(x)) \cdot \left\lfloor \frac{g_3(x) \cdots g_m(x)}{2} \right\rfloor \\
&\vdots \\
&\vdots \\
&= \left\lfloor \frac{g_1(x)}{2} \right\rfloor \cdot g_2(x) \cdots g_m(x) \\
&\quad + \textsc{Par}(g_1(x)) \cdot \left\lfloor \frac{g_2(x)}{2} \right\rfloor g_3(x) \cdots g_m(x) \\
&\quad + \textsc{Par}(g_1(x)) \cdot \textsc{Par}(g_2(x)) \cdot \left\lfloor \frac{g_3(x)}{2} \right\rfloor g_4(x) \cdots g_m(x) \\
&\quad + \cdots\cdots\cdots\cdots \\
&\quad + \textsc{Par}(g_1(x)) \cdot \textsc{Par}(g_2(x)) \cdots \textsc{Par}(g_{m-1}(x)) \cdot \left\lfloor \frac{g_m(x)}{2} \right\rfloor . \qquad (6.3)
\end{aligned}$$

Using the GapAC^0 circuits for $\left\lfloor \frac{g_1(x)}{2} \right\rfloor, \left\lfloor \frac{g_2(x)}{2} \right\rfloor, \ldots, \left\lfloor \frac{g_m(x)}{2} \right\rfloor$, the formulas (6.2) and (6.3) show how to build a GapAC^0 circuit for $\left\lfloor \frac{g(x)}{2} \right\rfloor$.

In order to construct a GapAC^0 circuit for $\left\lfloor \frac{F(x)}{2^\alpha} \right\rfloor$, if $\alpha > 1$, we first note that the formula (6.1) is also true for any integer function $f(x), g(x)$, hence it is true for functions in #AC^0 and in GapAC^0 too. Thus we can repeat the above process α times. Finally it is not hard to see that this construction gives a GapAC^0 circuit computing $\left\lfloor \frac{F(x)}{2^\alpha} \right\rfloor$ which has depth $O(d)$ and size polynomial in the size of C. ∎

Theorem 38 *The function* $\textsc{ExactHalf}(x) = \left\lfloor \frac{x_1 + \cdots + x_n}{2} \right\rfloor$ *cannot be computed in* #AC^0.

Note: We remark that we can actually show that exponential-size circuits are required, using a similar proof.

Proof: We need the following result:

Lemma 39 *[14] If $p(x_1, \ldots, x_n)$ is a polynomial of degree k with the property that $p(x_1, \ldots, x_n) = \oplus(x_1, \ldots, x_n)$ for all except $\epsilon 2^n$ inputs for $\epsilon < 1/2$, then*

$$k \geq n - O\left(\sqrt{n} \log\left(\frac{1}{\epsilon}\right)\right).$$

Note that

$$\oplus(x_1, \ldots, x_n) = (x_1 + \cdots + x_n) - 2 \cdot \left\lfloor \frac{x_1 + \cdots + x_n}{2} \right\rfloor.$$

Hence, if $\left\lfloor \frac{x_1+\cdots+x_n}{2} \right\rfloor$ could be computed by a polynomial of small degree, $\oplus$ could be computed by a polynomial of small degree as well. Together with Lemma 39, this means that any polynomial p such that $p(x_1, \ldots, x_n) = \left\lfloor \frac{x_1+\cdots+x_n}{2} \right\rfloor$ for all except $\epsilon 2^n$ inputs must have degree at least $n - O\left(\sqrt{n} \log\left(\frac{1}{\epsilon}\right)\right)$.

This result initially seems to have only limited application for proving results about $\#\mathrm{AC}^0$, since many functions computed by these arithmetic circuits have linear degree. One of our technical contributions is to show that the effects of large degree are not very great, when the size of the final function is small:

Lemma 40 *Let $c > 0$ be a constant. Let C_n be a depth-D, size-S_n $\#\mathrm{qAC}^0$ circuit computing the function f. Suppose that $0 \leq f(x) \leq 2^{\log^c n}$. Let $z_\varepsilon = \left(\log(1/\varepsilon) \log S_n \log^2 n\right)^D$. Then for each ε satisfying[2] $0 < \varepsilon \leq 1/S_n$ there exists a polynomial of degree $O\left(z_\varepsilon \log^{cD} n\right)$ of n variables with the property that $P(x) = f(x)$ for at least $1 - \varepsilon$ fraction of all inputs.*

Proof: For each subcircuit C_g, let C'_g be the corresponding qAC^0 circuit (i.e., the Boolean circuit obtained from C_g by replacing each + gate by an OR gate and each $\times$ gate by an AND gate). Let $g'(x)$ be the Boolean function computed by C'_g. Then, $g'(x) = 0$ if and only if there is no accepting subtree for the gate g' in the circuit C'_n, if and only if $g(x) = 0$. Thus, $g(x) = g'(x) \cdot g(x)$ for all input x.

By induction on the circuit depth we will show that for all $\varepsilon > 0$ sufficiently small, for each gate g of depth $\leq d$:

1. there exists a polynomial G of degree $O\left(z_\varepsilon \log^{cd} n\right)$ such that $\left|\left\{x : 0 < g(x) \leq 2^{\log^c n}\right.\right.$ and $\left.\left.G(x) \neq g(x)\right\}\right| \leq \varepsilon 2^{n-1}$ (in this case we say, the error is at most $\varepsilon/2$).

[2] If $\varepsilon > 1/S_n$, one can use the polynomial for $\varepsilon = 1/S_n$, getting the same result with slightly worse $z_\varepsilon = (\log^2 S_n \log^2 n)^D$.

2. there exists a polynomial H of degree $O\left(z_\varepsilon \log^{cd} n\right)$ such that $\left|\left\{x : 0 \leq g(x) \leq 2^{\log^c n}\right.\right.$ and $\left.\left.H(x) \neq g(x)\right\}\right| \leq \varepsilon 2^n$ (thus, the error is at most ε).

First we show the existence of the polynomial H by supposing that we have shown the existence of the polynomial G satisfying the conditions mentionned above. Let s and d be the size and depth of the Boolean circuit corresponding to g. Let $0 < \varepsilon_1 = \varepsilon/2$. Beigel *et al.* ([20] Lemma 6) showed that for any Boolean circuit of depth d and size s there exists a polynomial $G'(x)$ of degree $O\left(\left(\log(1/\varepsilon_1) \log s \log^2 n\right)^d\right) = O(z_\varepsilon)$ that agrees with $g'(x)$ on all except $\varepsilon_1 2^n$ inputs[3]. Define H to be $H(x) = G(x) \cdot G'(x)$. If $g(x) = 0$ then $g'(x) = 0$ and hence $G(x) \cdot G'(x) = 0$ with error $\varepsilon/2 < \varepsilon$. If $g(x) \neq 0$ then $g'(x) = 1$ and $G'(x) = 1$ with error $\varepsilon/2$. Because $G(x) = g(x)$ with error $\varepsilon/2$, this implies that $H(x) = g(x)$ with error ε.

Now we show the existence of the polynomial G by induction on the circuit depth d. For the base case of $d = 0$ just define $G(x) = g(x)$.

Consider the case of $d \geq 1$ and let $g_1, \ldots, g_m$ be the inputs of g, where each g_i is of depth $\leq d - 1$. Consider first the case of a + gate, that is $g(x) = g_1(x) + \cdots + g_m(x)$. The inputs x satisfying $0 < g(x) \leq 2^{\log^c n}$ will also satisfy $0 \leq g_i(x) \leq 2^{\log^c n}$ for all $i = 1, \ldots, m$. By the induction hypothesis, for each $\varepsilon_1 = \varepsilon/m$ and for each g_i there exists a polynomial H_i of degree $O\left(z_{\varepsilon_1} \log^{c(d-1)} n\right)$ such that if $0 \leq g_i(x) \leq 2^{\log^c n}$ then $H_i(x) = g_i(x)$ with error ε_1. Define $G = H_1 + \cdots + H_m$, than $G(x)$ will compute $g(x)$ with error $m \cdot \varepsilon_1 = \varepsilon$. The degree of G is the maximum of the degrees of H_i which is $O\left(z_{\varepsilon_1} \log^{c(d-1)} n\right)$ and which can be shown to be $O(z_\varepsilon \log^{cd} n)$ for small ε, for exemple $\varepsilon < 1/S_n$.

Consider the case where $g(x) = g_1(x) \cdots g_m(x)$. The inputs x satisfying $0 < g(x) \leq 2^{\log^c n}$ will also satisfy $0 < g_i(x) \leq 2^{\log^c n}$ for all $i = 1, \ldots, m$. By the induction hypothesis, for each $\varepsilon_1 = \varepsilon/m$ and for each g_i there exists a polynomial G_i of degree $O\left(z_{\varepsilon_1} \log^{c(d-1)} n\right)$ such that if $0 < g_i(x) \leq 2^{\log^c n}$ then $G_i(x) = g_i(x)$ with error ε_1. Note also that there are only at most $\log^c n$ values among $g_1(x), \ldots, g_m(x)$ that are

[3]In fact, Beigel *et al.* showed the existence of a probabilistic polynomial that agrees with g' with probability $1 - \varepsilon_1$. However, one can fix the probabilistic variables and obtain a polynomial in the usual sense.

strictly greater than 1. Hence $\sum_{i_1,\ldots,i_k} \prod_{j=1}^{k}(g_j(x)-1) = 0$ for all $k > \log^c n$. Therefore we have:

$$\begin{aligned} g(x) = \prod_{i=1}^{m} g_i(x) &= \prod_{i=1}^{m}[1 + (g_i(x) - 1)] \\ &= \sum_{k=0}^{m} \sum_{i_1,\ldots,i_k} \prod_{j=1}^{k}(g_{i_j}(x) - 1) \\ &= \sum_{k=0}^{\log^c n} \sum_{i_1,\ldots,i_k} \prod_{j=1}^{k}(g_{i_j}(x) - 1). \end{aligned}$$

The induction hypothesis implies that the polynomial G defined by

$$G(x) = \sum_{k=0}^{\log^c n} \sum_{i_1,\ldots,i_k} \prod_{j=1}^{k}(G_{i_j}(x) - 1)$$

will compute $g(x)$ with error $m\varepsilon_1 = \varepsilon/2$. The degree of G is at most $\log^c n$ times the maximum of the degrees of G_i which is $\log^c n \cdot O\left(z_{\varepsilon_1} \log^{c(d-1)} n\right) = O\left(z_\varepsilon \log^{cd} n\right)$ for $\varepsilon < 1/S_n$.

∎

To complete the proof of Theorem 38, Suppose that $\left\lfloor \frac{x_1+\cdots+x_n}{2} \right\rfloor$ could be computed with a $\#\mathrm{AC}^0$ circuit of depth D and of size S. We take $c = 1$. By Lemma 40, for each $\varepsilon > 0$ there exists a polynomial P of degree

$$k = O\left(\left(\log(1/\varepsilon) \log S \log^2 n\right)^D \log^D n\right) = polylog(n)$$

such that the number of inputs x where $P(x) \neq \left\lfloor \frac{x_1+\cdots+x_n}{2} \right\rfloor < 2^{\log n}$ is at most $\varepsilon 2^n$. Let $P'(x) = (x_1 + \ldots + x_n) - 2P(x)$. Then, the number of inputs x where $P'(x) \neq \oplus(x_1, \ldots, x_n)$ is also at most $\varepsilon 2^n$. That leads to a contradiction with Corollary 39. This completes the proof of Theorem 38.

∎

In fact, we show that even if we relax the requirement that we round down accurately when the number of 1's in the input is odd, it is still difficult to compute half the sum of the inputs. The following theorem makes it precise.

Theorem 41 *For* any *function g, the function*

$$\textsc{ApprHalf}(x) = \begin{cases} \frac{\sum_{i=1}^{n} x_i}{2} & \text{if } \sum_{i=1}^{n} x_i \text{ is even} \\ g(x) & \text{otherwise} \end{cases}$$

cannot be computed in $\#\mathrm{qAC}^0$.

Proof: Suppose that there is a $\#\mathrm{qAC}^0$ circuit C computing the function $\textsc{ApprHalf}(x)$. Then there exists a polynomial G of small degree such that if $\sum_{i=1}^{n} x_i$ is even then $G(x) = C(x)$ with small error (because $C(x) = \frac{\sum_{i=1}^{n} x_i}{2} < 2^{\log n}$; the existence of the polynomial follows from Lemma 40. We can choose $\epsilon = \frac{1}{n}$ there, for example). Define the polynomial $H(x) = (\sum_{i=1}^{n} x_i - 2 \cdot G(x))^2$. Then $H(x) = 0$ with small error if $\sum_{i=1}^{n} x_i$ is even and $H(x) > 1$ otherwise. The polynomial $H'(x) = 1 - 2H(x)$ clearly has small degree and satisfies $sgn(H'(x)) \neq sgn(\oplus(x))$ with small error and this contradicts [14] ∎

6.7 Miscellaneous Functions

A useful tool for showing non-membership in GapAC^0 was presented by Lu [40]. He defined the following notion of *period*: If $f : \{0,1\}^n \to \mathbb{N}$ is a symmetric function, consider f as a function from $\{0, 1, \ldots, n\}$ into $\mathbb{N}$. The *period* of f is the least integer $k > 0$ such that $f(x) = f(x + k)$ for $0 \leq x \leq n - k$. Notice that if the function f is not periodic we can take k to be equal to $n + 1$.

Theorem 42 *[40] A symmetric Boolean function f is in the class* $\mathrm{qAC}^0[2]$ *if and only if it has period* $2^{t(n)} = \log^{O(1)} n$ *(with possible exceptions at $f(i)$ and $f(n - i)$ for* $i = \log^{O(1)} n$*).*

Theorem 42 easily yields non-closure results for $\#\mathrm{AC}^0$ and GapAC^0, of which the following corollary is an example.

Corollary 43 *The functions* $\lfloor \sqrt{\sum_i x_i} \rfloor$ *and* $\lfloor \log(1 + \sum_i x_i) \rfloor$ *cannot be computed in* GapqAC^0. *Thus neither* $\#\mathrm{AC}^0$ *nor* GapAC^0 *are closed under taking of roots or logarithms.*

Chapter 7

Grid Graphs

7.1 G-graphs

The importance of grid graphs to the study of constant-depth circuits was first shown in [18]. In this paper we use an equivalent notion, that makes it formally easier to present our results.

Definition 17 *A* width-k G-graph *is a graph that has a planar embedding in which the vertices are grouped in a rectangular array of width k (the length is variable) with edges between vertices of adjacent columns only. For any G-graph, let s and t refer respectively to its lower left and upper right vertices. Also, if G_1, G_2 are width-k G-graphs then G_1G_2 denotes the G-graph formed by merging the rightmost column of G_1 and the leftmost column of G_2. This notation extends naturally to more than two width-k G-graphs.*

G-graphs are important to the study of circuit complexity, since the reachability problem for width-k G-graphs is complete for depth-k AC^0 [18]. Unfortunately, even for width-2 G-graphs, counting the number of paths from s to t cannot be done in GapAC^0. To see this, consider the small G-graph G_A illustrated in Figure 7.1 which implements the reachability matrix $A = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$. That is, there are two paths from vertex 1 (bottom row) in the first column to vertex 1 in the third column, and for all other $(i, j) \in \{1, 2\}^2$ there is exactly one path from vertex i in the first column to vertex j in the third column. Recall that all edges are directed from left to right. Note that $A^i = \begin{bmatrix} f_{2i+1} & f_{2i} \\ f_{2i} & f_{2i-1} \end{bmatrix}$ where f_j denotes the j-th Fibonacci number.

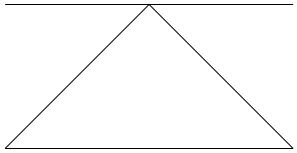

Figure 7.1: The G-graph G_A.

Now consider the homomorphism h mapping $\sigma \in \{0,1\}$ to A^σ. Thus $h(x)$ represents a string of matrices and by a slight abuse of notation can represent the product of this string. Thus, given a string x, the low-order bit of $h(x)_{1,1}$ will be 0 if and only if the number of 1's in x is equivalent to 1 (mod 3), using the easily verifiable fact that f_{2i+1} is even iff i is divisible by 3. Since the mod 3 function is not in $\mathrm{AC}^0[2]$, it follows that counting the number of paths from s to t is not in GapAC^0.

A similar argument shows that this problem is hard for NC^1 under the appropriate reductions.

Theorem 44 *Counting the number of s-t paths in width-two G-graphs is hard for* NC^1 *under* $\mathrm{AC}^0[5]$ *reductions.*

Proof: The group of two-by-two matrices with determinant 1 over the integers mod 5 is non-solvable, and hence multiplication in it is hard for NC^1 by [15]. But given any multiplication in this group, we can construct path-counting problems in a G-graph whose answers modulo 5 are the entries of the product matrix. This is because any two-by-two matrix over $\mathbb{N}$ with determinant 1 can be represented as a product of those two matrices coded for by the columns of Figure 7.1. (For a proof of this fact, see, e.g., [22, Theorem 3.1].).

Now we will show that the reduction is in $\mathrm{AC}^0[5]$. The transformation that converts the input x to a sequence of matrices from Figure 7.1 is clearly AC^0. Suppose we have an oracle gate for counting the number of s,t paths in width-2 grid graphs which gives the answer in binary. Let x_i denote the i^{th} bit of the answer and c_i the value of 2^i mod 5. Notice that since i can take a polynomially bounded value c_i can be computed in AC^0. We make c_i copies of x_i for each i and apply a Mod_5 gate to its output. As input to this oracle gate we use the above transformation mapping x to a sequence of matrices from Figure 7.1. Thus, we get an $\mathrm{AC}^0[5]$ circuit, with oracle gates for the computing

the number of s, t paths in width-2 grid graphs, which tells us whether x belongs to A.

∎

7.2 A Special Family of G-graphs

Now consider the family of G-graphs defined by composing the ones shown in Figure 7.2. The next theorem shows that counting the number of s, t paths in this family is equivalent to computing a $\#\mathrm{AC}^0$ function.

Theorem 45 *Define the σ-depth of a circuit to be the maximum number of $\sum$ gates on any path in the circuit. Arithmetic circuits of σ-depth k can be simulated by counting the number of $s-t$ paths in a G-graph of width $2k+2$, where the subgraph between any pair of columns is drawn from the family illustrated in Figure 7.2. Conversely, given such a G-graph G, the number of $s-t$ paths in G can be computed by a uniform family of $\#\mathrm{AC}^0$ circuits.*

Proof: For the forward direction, we construct a function f which associates a graph $f(C)$ with every gate C in a given $\#\mathrm{AC}^0$ circuit, such that the number of $s-t$ paths in $f(C)$ is equal to the output of the gate. We assume, without loss of generality, that the circuit is leveled so that we can construct the function f by an induction on its depth. The construction uses the graphs $G_{k,j}$ illustrated in Figure 7.2.

C is the constant c: $f(C) = G_{k,c}$.

C is the literal l: $f(C) = G_{k,l}$.

C is a $\prod$-gate at σ-depth d with inputs $C_1, \ldots, C_r$:

$$f(C) = f(C_1)G_{k,2d+2}f(C_2)G_{k,2d+2}\ldots G_{k,2d+2}f(C_r)$$

C is a $\sum$-gate at σ-depth d with inputs $C_1, \ldots, C_r$:

$$f(C) = G_{k,2d+1}f(C_1)G_{k,2d+1}f(C_2)\ldots G_{k,2d+1}f(C_r)G_{k,2d+1}$$

The G-graph for the formula $(x_1x_2 + \overline{x_3})(\overline{x_2} + x_1x_4)$ is illustrated in Figure 7.2.

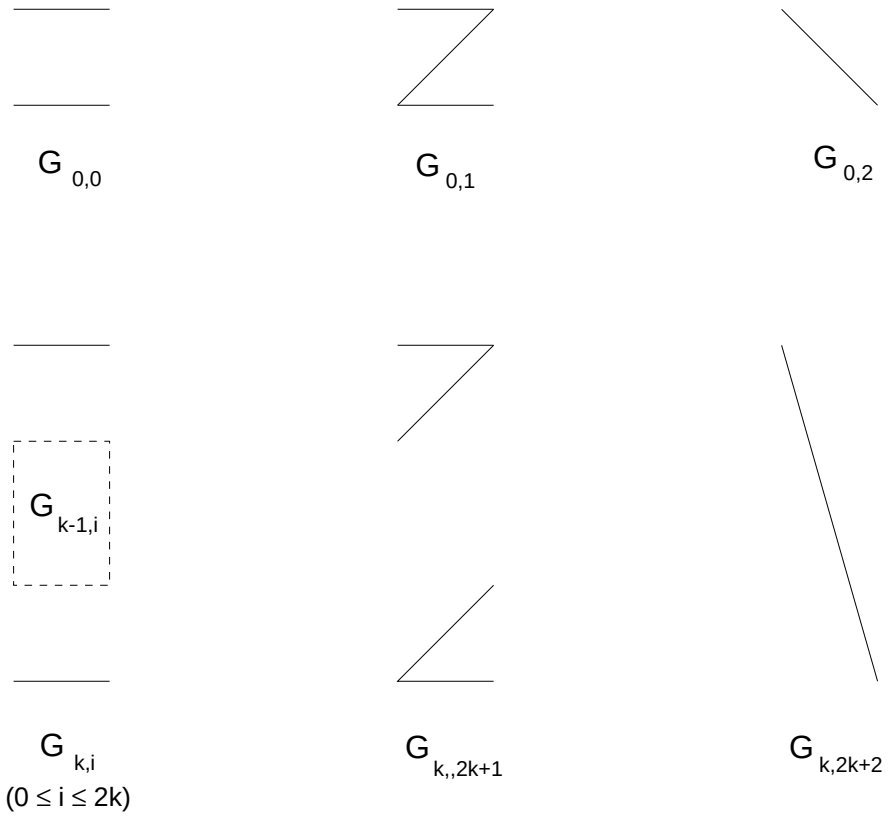

Figure 7.2: The G-graphs $G_{i,j}$.

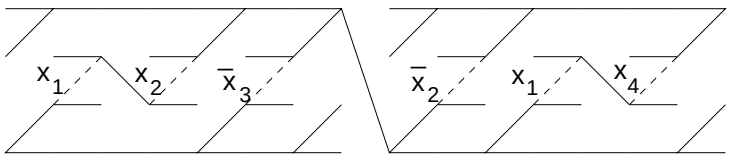

Figure 7.3: G-graph for $(x_1x_2 + \overline{x_3})\,(\overline{x_2} + x_1x_4)$

For a G-graph G of width $2k$, let $s_i(G), t_i(G)$ $(1 \leq i \leq 2k)$ denote the i-th vertex from the bottom on the left boundary and from the top on the right boundary, respectively. Thus with this convention, $s = s_1(G)$ and $t = t_1(G)$. It is straightforward to show by induction that for a gate C at σ-depth d in the circuit, the number of $s_{k-d}(f(C)) - t_{k-d}(f(C))$-paths equals the value computed by C.

Conversely, we have a width $2k$ graph $g'_1 \ldots g'_n$ (where each g'_i is one of the $G_{k-1,j}$'s illustrated in Figure 7.2) and we want to compute the number of $s-t$ paths. First, we build the width $2k+2$ graph $g_0 \ldots g_{n+1}$, where $g_0 = G_{k,2k+1} = g_{n+1}$, and for $1 \leq n$, if g'_i is $G_{k-1,j}$, then $g_i = G_{k,j}$. This does not change the number of $s-t$ paths, and makes the resulting algorithm easier to describe.

Inductively, we define a number of functions $\sigma_d[i,j]$ and $\pi_d[i,j]$, where $0 \leq d \leq k$ and $1 \leq i < j \leq n$ as follows [1] :

$$\sigma_0[i,j] = \begin{cases} \sum_{t=i+1}^{j-1} g_t & \text{if } g_i, g_{j+1} \in \{G_{k,2}, G_{k,3}\} \\ & \text{and } g_t \in \{G_{k,0}, G_{k,1}\} \text{ for } i < t < j \\ 1 & \text{otherwise} \end{cases}$$

$$\pi_0[i,j] = \begin{cases} \prod_{i \leq i' < j' \leq j} \sigma_0[i',j'] & \text{if } g_i, g_{j+1} \in \{G_{k,3}\} \\ & \text{and } g_t \in \{G_{k,0}, G_{k,1}, G_{k,2}\} \text{ for } i < t < j \\ 0 & \text{otherwise} \end{cases}$$

$$\sigma_d[i,j] = \begin{cases} \sum_{i \leq i' < j' \leq j} \pi_{d-1}[i',j'] & \text{if } g_i, g_{j+1} \in \{G_{k,2d+2}, G_{k,2d+3}\} \\ & \text{and } g_t \in \{G_{k,0}, \ldots, G_{k,2d+1}\} \text{ for } i < t < j \\ 1 & \text{otherwise} \end{cases}$$

$$\pi_d[i,j] = \begin{cases} \prod_{i \leq i' < j' \leq j} \sigma_d[i',j'] & \text{if } g_i, g_{j+1} \in \{G_{k,2d+3}\} \\ & \text{and } g_t \in \{G_{k,0}, \ldots, G_{k,2d+2}\} \text{ for } i < t < j \\ 0 & \text{otherwise} \end{cases}$$

[1] Notice that we interpret the graphs $G_{k,0}$ and $G_{k,1}$ as the numerical constants 0 and 1.

It is straightforward to show that $\pi_k[1, n]$ is the correct number of $s - t$ paths. and that the functions defined above can indeed be computed by $\#\mathrm{AC}^0$ circuits. ∎

Chapter 8

Conclusion and Future Work

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 25 and the circuit lower bounds in [46]. At the end of Chapter 6 (Corollary 43) 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 #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 46 *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 $\#\mathrm{AC}^0$, the proof for GapAC^0 is identical.

Assume that $\#\mathrm{AC}^0{}_k = \#\mathrm{AC}^0{}_{k+1}$ for some $k > 0$. It follows then that $\#\mathrm{AC}^0 = \#\mathrm{AC}^0{}_k$. Let A be a language in AC^0 but not in depth k $\mathrm{AC}^0[2]$. (See, for instance Proposition 11 in [7]. 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 $\#\mathrm{AC}^0$, and therefore, in $\#\mathrm{AC}^0{}_k$ by our assumption. But this gives a depth k $\mathrm{AC}^0[2]$ circuit for A, in contradiction to our choice of A.

Pierre McKenzie has pointed out that Sipser's functions[51], used to show that the depth hierarchy for AC^0 is strict, can also be used to prove the first part of the above theorem. For if f is the characteristic function of a language in $\mathrm{AC}^0{}_{k+1}$ but not in $\mathrm{AC}^0{}_k$, then consider the arithmetic function g obtained by arithmetizing the $\mathrm{AC}^0{}_{k+1}$ circuit for f. This is clearly in $\#\mathrm{AC}^0{}_{k+1}$. Suppose, $g \in \#\mathrm{AC}^0{}_k$ as witnessed by circuit C. Then the function f' formed by "de-arithmetizing" C (i.e. converting product gates to conjunctions and sum gates to disjunctions) is the same as f. This is because, f' is

0 iff g is 0 iff f is 0. Thus showing $f \in {\rm AC^0}_k$, which contradicts the definition of f. ∎

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 $\#{\rm AC^0}$ function.

Note that [47] argues that, if certain popular cryptographic assumptions are true, then there are no "natural proofs" of lower bounds for $\rm TC^0$ 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 $\rm TC^0$ via arithmetic circuits. At the time the work was first done, we hoped that this would be a new approach towards proving lower bounds for threshold circuits. This is still possible but the intervening years have not seen the creation of these lower bound techniques.

Since many lower bounds for $\#{\rm AC^0}$ can be proved using natural proofs, it would also be interesting to know what types of questions about $\#{\rm AC^0}$ 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. At the time the work was published we knew that ${\rm TC^0} \subseteq {\rm C_{=}AC^0} \subseteq {\rm PAC^0}$ in the Dlogtime-uniform setting, but equality was not known to hold. Thus, even though we knew that Dlogtime-uniform $\rm TC^0$ is not equal to PP, we did not know whether $\rm PAC^0$ and PP are distinct. We did, however, observe that if the inclusion ${\rm PAC^0} \subseteq {\rm TC^0}$ holds also in the Dlogtime-uniform setting, then a negative answer would follow from the lower bound of [9] showing that the permanent requires large threshold circuits. The argument is completed by noticing that the higher order bit of the permanent in in PP but from the previous sentence not in $\rm TC^0$. That the inclusion holds has been proved by [38].

It is worthwhile to note that [5] enables us to show that the determinant and the permanent are not in Non-uniform ${\rm GapAC^0}$. That paper shows that the determinant/permanent of extended lower triangular (elt) matrices is Logspace many-one complete for #L. The reduction can actually be seen to be an $\rm AC^0$-reduction. Thus it

follows that the Mod_3 function is AC^0-reducible to the determinant/permanent of elt-matrices. Thus if the determinant/permanent of arbitrary matrices is in Non-uniform GapAC^0, then so is the Mod_3 function. But we know this to be false.

Another direction worth investigating concerns branching programs. It is shown in [24] that $\#\text{NC}^1$ 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 programs over *aperiodic monoids* [17]. Is there some characterization of $\#\text{AC}^0$ or DiffAC^0 in terms of branching programs? This question has been answered to a great extent by Theorem 45. Is there a related algebraic characterization?

Recently, Hansen [35] has given a characterization of ACC^0 in terms of planar circuits of polynomial size and constant width. Perhaps, a combination of that insight with the characterization of ACC^0 by constant depth arithmetic circuits over finite fields and the equivalence of constant depth arithmetic circuits with the G-graph family of Chapter 7 may lead to interesting results.

Apart from these there are a couple of open questions in Chapter 6 related to closure of $\#\text{AC}^0$ and GapAC^0 under the polylogarithmic choose operation which are interesting in their own right.

References

[1] Manindra Agrawal, Eric Allender, and Samir Datta. On TC^0, AC^0, and Arithmetic circuits. *Journal of Computer and System Sciences*, 60(2):395–421, 2000.

[2] M. Ajtai. Σ^1_1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[3] E. Allender. P-uniform circuit complexity. *Journal of the Association for Computing Machinery*, 36:912–928, 1989.

[4] E. Allender. Making computation count: arithmetic circuits in the nineties. *SIGACT News*, 28(4):2–15, 1998.

[5] E. Allender, V. Arvind, and M. Mahajan. Arithmetic complexity, Kleene closure, and formal power series. *Theory of Computing Systems*, 36(4):303–328, 2003.

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

[7] E. Allender and U. Hertrampf. Depth reductions for circuits of unbounded fan-in. *Information and Computation*, 112:217–238, 1994.

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

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

[10] Eric Allender, Andris Ambainis, David A. Mix Barrington, Samir Datta, and Huong LêThanh. Bounded depth arithmetic circuits: Counting and closure. *Lecture Notes in Computer Science*, 1644:149–158, 1999.

[11] Eric Allender, Jia Jiao, Meena Mahajan, and V. Vinay. Non-commutative arithmetic circuits: depth reduction and size lower bounds. *Theoretical Computer Science*, 209(1–2):47–86, 1998.

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

[13] A. Ambainis, D.A.M. Barrington, and H. LeThanh. On counting AC^0 circuits with negative constants. In *Proc. of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 409–417, 1998.

[14] J. Aspnes, R. Beigel, M. Furst, and S. Rudich. The expressive power of voting polynomials. *Combinatorica*, 14(2):135–148, 1994.

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

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

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

[18] David A. Mix Barrington, Chi-Jen Lu, Peter Bro Miltersen, and Sven Skyum. Searching constant width mazes captures the AC^0 hierarchy. In *Symposium on Theoretical Aspects of Computer Science*, pages 73–83, 1998.

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

[20] R. Beigel, N. Reingold, and D. Spielman. The perceptron strikes back. In *Proceedings 6th Structure in Complexity Theory*, pages 286–291. IEEE Computer Society Press, 1991.

[21] R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4:350–367, 1994.

[22] Andreas Blass and Yuri Gurevich. Matrix transformation is complete for the average case. *SIAM Journal on Computing*, 24(1):3–29, 1995.

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

[24] Hervé Caussinus, Pierre McKenzie, Denis Thérien, and Heribert Vollmer. Nondeterministic NC^1 computation. *Journal of Computer and System Sciences*, 57:200–212, 1998.

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

[26] H. Chen. Arithmetic constant-depth circuit complexity classes. In *Proc. of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2003.

[27] George I. Davida and Bruce Litow. Fast parallel arithmetic via modular representation. *Siam Journal of Computing*, 20:756–765, 1991.

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

[29] 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, 1994.

[30] R. Fagin, M. M. Klawe, N. J. Pippenger, and L. Stockmeyer. Bounded-depth, polynomial-size circuits for symmetric functions. *Theoretical Computer Science*, 36(2-3):239–250, April 1985.

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

[32] Gudmund S. Frandsen, Mark Valence, and David A. Mix Barrington. Some results on uniform arithmetic circuit complexity. *Mathematical Systems Theory*, 27(2):105–124, 1994.

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

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

[35] K. A. Hansen. Constant width planar computation characterizes ACC^0. Technical Report TR03-025, Electronic Colloquium on Computational Complexity, 2003.

[36] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, fifth edition, 1979.

[37] J. Hastad. Almost optimal lower bounds for small depth circuits. In S. Micali, editor, *Randomness and Computation*, pages 143–170, Greenwich, Connecticut, 1989. Advances in Computing Research, vol. 5, JAI Press.

[38] W. Hesse, E. Allender, and D.A.M. Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65:695–716, 2002.

[39] Bruce Litow. On iterated integer product. *Information Processing Letters*, 42(5):269–272, 1992.

[40] Chi-Jen Lu. An exact characterization of symmetric functions in $\mathrm{qAC}^0[2]$. *Theoretical Computer Science*, 261(2):297–303, 2001.

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

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

[43] Fabrice Noilhan and Miklos Santha. Semantical counting circuits. *Lecture Notes in Computer Science*, 1767:87–??, 2000.

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

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

[46] 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.

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

[48] John H. Reif and Stephen R. Tate. On threshold circuits and polynomial computation. *SIAM J. Comput.*, 21(5):896–908, 1992.

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

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

[51] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th Symposium on Theory of Computing*, pages 61–69. ACM Press, 1983.

[52] 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.

[53] S. Toda. Counting problems computationally equivalent to computing the determinant. Technical Report CSIM 91-07, University of Electro-communications, Chofugaoka, May 1991.

[54] S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Transactions on Information and Systems*, E75-D:116–124, 1992.

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

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

[57] L. G. Valiant. Why is Boolean complexity theory difficult? In M. S. Paterson, editor, *Boolean Function Complexity*. Cambridge University Press, 1992. London Mathematical Society Lecture Notes Series 169.

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

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

[60] V. Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *Proceedings of the 6th Structure in Complexity Theory Conference*, volume 223 of *Lecture Notes in Computer Science*, pages 270–284, Berlin, 1991. Springer.

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

[62] T. Yamakami. Uniform AC^0 counting circuits. Manuscript, 1996.

Vita

Samir Datta

1991 Graduated from Colvin Taluqdars' College, Lucknow, India.

1991-95 Attended Indian Institute of Technology, Kanpur, India. Majored in Computer Science and Engineering.

1995 B.Tech., Indian Institute of Technology, Kanpur.

1995-04 Graduate work in Computer Science, Rutgers, The State University of New Jersey, New Brunswick, New Jersey.

1995-96 Excellence Fellowship, Rutgers, The State University of New Jersey.

1996-00 Graduate/Teaching Assistant, Department of Computer Science.

1997 M.S. in Computer Science, Rutgers, The State University of New Jersey.

1999 Allender, E., A. Ambainis, D.A.M. Barrington, S. Datta and H. LeThanh. Bounded Depth Arithmetic Circuits: Counting and Closure. 1(ICALP-1999, LNCS), 1644:149-158.

2000 Agrawal, M., E. Allender and S. Datta. On TC^0, AC^0, and Arithmetic Circuits. 1(JCSS), 60(2000):395-421.

2000-03 Network Architect, Tellium Inc., Oceanport, New Jersey.

2004 Ph.D. in Computer Science.