Bounded Depth Arithmetic Circuits: Counting and Closure

Eric Allender*
Department of Computer Science
Rutgers University, Piscataway
allender@cs.rutgers.edu

Andris Ambainis
Division of Computer Science
University of California, Berkeley
ambainis@cs.berkeley.edu

David A. Mix Barrington
Department of Computer Science
University of Massachusetts, Amherst
barring@cs.umass.edu

Samir Datta*
Department of Computer Science
Rutgers University, Piscataway
sdatta@paul.rutgers.edu

Huong LêThanh
Laboratoire de Recherche en Informatique
Université de Paris-Sud
huong@lri.fr

April 19, 1999

Abstract

Constant-depth arithmetic circuits have been defined and studied in [AAD97, ABL98]; these circuits yield the function classes $\#AC^0$ and $GapAC^0$. These function classes in turn provide new characterizations of the computational power of threshold circuits, and provide a link between the circuit classes AC^0 (where many lower bounds are known) and TC^0 (where essentially no lower bounds are known). In this paper, we resolve several questions regarding the closure properties of $\#AC^0$ and $GapAC^0$.

Counting classes are usually characterized in terms of problems of counting paths in a class of graphs (simple paths in directed or undirected graphs for #P, simple paths in directed acyclic graphs for #L, or paths in bounded-width graphs for $GapNC^1$). It was shown in [BLMS98] that complete problems for depth k Boolean AC^0 can be obtained by considering the reachability problem for width k grid graphs. It would be tempting to conjecture that $\#AC^0$ could be characterized by counting paths in bounded-width grid graphs. We disprove this, but nonetheless succeed in characterizing $\#AC^0$ by counting paths in another family of bounded-width graphs.

1 Introduction

The arithmetic circuit complexity classes $\#AC^0$ and $GapAC^0$ have been the object of intense study [AAD97, ABL98, LêT98, NS98] because:

- they provide new characterizations of the complexity class TC^0 (the problems computable by constant-depth threshold circuits of polynomial size), for which essentially no nontrivial lower bounds have been proved,

*Supported in part by NSF grant CCR-9734918.

- they are closely related to the complexity classes AC^0 and $AC^0[2]$, and thus the well-developed lower-bound techniques for AC^0 and $AC^0[2]$ suffice to show that certain functions are *not* in $\#AC^0$ and $GapAC^0$, and
- they capture a mathematically interesting class of computations lying at the frontier of currently available analysis techniques. We can answer some questions about their structure at this time (possibly giving insight into the structure of related classes such as #L and #P), while progress on other questions about them is necessary to better understand TC^0 (and hence the power of neural nets).

More background motivating our interest in these classes can be found in [AAD97, ABL98, LêT98] and in the survey article [Al97]. Formal definitions for each are presented in Section 2. Here we report progress on two fronts regarding $\#AC^0$ and $GapAC^0$: closure properties and combinatorial characterizations.

1.1 Closure Properties

A study of the closure properties of #P was initiated by Ogiwara and Hemachandra in [OH93]. It is not known whether #P is closed under such operations as MAX, MIN, division by 2, and decrement. In [OH93] 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.

1.2 Combinatorial Characterizations

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 [Al97] for a discussion of this problem.) Nonetheless, it was shown in [CMTV96] 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 of whether $\#AC^0$ or $GapAC^0$ possess similar combinatorial characterizations was posed in [AAD97]. It was noted there 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 [BLMS98] 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.

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 $\#AC^0$.

2 Preliminaries

This paper studies arithmetic complexity classes. Certainly the best-known arithmetic class is Valiant's class #P [Val79], consisting of functions that map x to the number of accepting computations of an NP-machine on input x. Recently, the class #L (counting accepting computations of an NL-machine) has also received considerable attention [AJ93, Vin91, Tod92a, MV97].

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 [Ven92]. The classes #P and #L result if we "arithmetize" these Boolean circuits, replacing each OR gate by a $+$ gate, and replacing each AND gate by a $\times$ gate, where the input variables $x_1, \ldots, x_n$ now take as values the natural numbers $\{0, 1\}$ (instead of the Boolean values $\{0, 1\}$), and negated input literals $\overline{x_i}$ now take on the value $1 - x_i$. Alternatively, #P and #L arise by counting the number of "accepting subtrees" for the corresponding classes of Boolean circuits. (See [Ven92] for a formal definition of this notion; for our purposes it is sufficient to know that the number of accepting subtrees of a circuit C is (a) equal to the output of the "arithmetized" version of C, (which we denote by $\#C$) and (b) provides a natural notion of counting the number of proofs that C accepts.) The arithmetic circuits corresponding to #L were studied further by Toda [Tod92b].

The counting classes that result in this way by arithmetizing the Boolean circuit classes SAC^1 and NC^1 were studied in [Vin91, AJMV98, CMTV96]. In this paper, we study $\#\mathrm{AC}^0$.

Definition 1 *For any $k > 0$, $\#\mathrm{AC}^0_k$ is the class of functions $f : \{0,1\}^* \to \mathbb{N}$ such that, for some polynomial p, for every n there is a depth k circuit C_n of size at most $p(n)$ consisting of unbounded-fan-in $+, \times$-gates (the usual sum and product in $\mathbb{N}$), where inputs to the circuits are from $\{0, 1, x_i, 1 - x_i\}$, and for every $x = x_1 \ldots x_n \in \{0,1\}^n, f(x) = C_n(x_1, \ldots, x_n)$. Let $\#\mathrm{AC}^0 = \bigcup_{k>0} \#\mathrm{AC}^0_k$.*

Definition 2 GapAC^0 *is the class of all functions $f : \{0,1\}^* \to \mathbb{Z}$ that can be expressed as the difference of two functions in $\#\mathrm{AC}^0$; i.e.* $\mathrm{GapAC}^0 = \{f \;:\; \exists g, h \in \#\mathrm{AC}^0 \;\; f(x) = g(x) - h(x)\}$.

$\#\mathrm{AC}^0$ and GapAC^0 were first studied in [AAD97]. Some of the main open questions posed there were subsequently answered in [ABL98], where it was shown that GapAC^0 can also be characterized as those functions computed by $\#\mathrm{AC}^0$ circuits augmented with -1 as an additional constant.

3 Closure and Nonclosure Properties

Following [OH93], let us first consider a very simple closure property: MAX.

Theorem 1 *Neither* $\#\mathrm{AC}^0$ *nor* GapAC^0 *is closed under* MAX. □

Proof: Let $f(x) = \sum_i x_i$, and let $g(x) = |x|/2$. 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 $|x|/2$ 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 [Ra87]. ∎

Corollary 2 *Neither* $\#\mathrm{AC}^0$ *nor* GapAC^0 *is closed under* MIN. □

Again following [OH93], we next consider the decrement operation. More precisely, 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 #AC^0 closed under decrement, but it is closed under $\dot{-}$ with any #AC^0 function whose growth rate is at most polylogarithmic.

Theorem 3 *If f and g are in #AC^0, and there exists a k such that for all x, $g(x) = O(\log^k |x|)$, then $f \dot{-} g$ is in #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). Note that $f(x) \dot{-} g(x)$ is nonzero if and only if the number of ones in x exceeds the threshold $g(x)$. If this function were in #AC^0, it would imply the existence of a Boolean AC^0 circuit family computing threshold-g, contradicting [FKPS85, Ha86]. An argument similar to that of Theorem 1 shows that $f \dot{-} g$ is not even in GapAC^0.

Proof: The proof is built of two following lemmas. We first need the following definition.

Definition 3 *Let $\#_{t,n}$ be the following integer function on n Boolean inputs:*

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

Lemma 4 *([DGS86, FKPS85]) If $t = O(\log^k n)$ for some fixed k then the function $\#_{t,n}$ is in* AC^0 *(and hence in #AC^0).* □

Lemma 5 *If f is a #AC^0 function and g is a function in #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 #AC^0-circuit computing f and using Lemma 4. 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

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

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

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

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 3, 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 5 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 5 and by closure under sum and product.

Now consider the case where the output gate is $\times$. By Lemma 5, 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 5), 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. ∎

A similar argument allows us to show the following:

Lemma 6 *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 7 *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 [Lu98]. ∎

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

Theorem 8 *For any integer constant α and any function $F(x) \in$ 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 $F \in$ DiffAC^0, there exist two functions $f, h \in$ #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 [AAD97] that $\textsc{Par}(f(x))$ can be computed in GapAC^0. The following formula can be easily verified:

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

Therefore, it is enough to show that if $f(x) \in$ #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 #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 fan-in 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 #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 [AAD97]).

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

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

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

$$\begin{aligned}
\left\lfloor \frac{g(x)}{2} \right\rfloor &= \left\lfloor \frac{g_1(x)}{2} \right\rfloor \cdot g_2(x) \cdots g_m(x) + \textsc{Par}(g_1(x)) \cdot \left\lfloor \frac{g_2(x) \cdots g_m(x)}{2} \right\rfloor \\
&= \left\lfloor \frac{g_1(x)}{2} \right\rfloor \cdot g_2(x) \cdots g_m(x) + \textsc{Par}(g_1(x)) \cdot \left\lfloor \frac{g_2(x)}{2} \right\rfloor g_3(x) \cdots g_m(x) \\
&\qquad + \textsc{Par}(g_1(x)) \cdot \textsc{Par}(g_2(x)) \cdot \left\lfloor \frac{g_3(x) \cdots g_m(x)}{2} \right\rfloor \\
&\ \vdots
\end{aligned}$$

$$
\begin{aligned}
&\vdots \\
= &\left\lfloor \frac{g_1(x)}{2} \right\rfloor \cdot g_2(x) \cdots g_m(x) \\
&+\textsc{Par}(g_1(x)) \cdot \left\lfloor \frac{g_2(x)}{2} \right\rfloor g_3(x) \cdots g_m(x) \\
&+\textsc{Par}(g_1(x)) \cdot \textsc{Par}(g_2(x)) \cdot \left\lfloor \frac{g_3(x)}{2} \right\rfloor g_4(x) \cdots g_m(x) \\
&+ \cdots\cdots\cdots\cdots \\
&+\textsc{Par}(g_1(x)) \cdot \textsc{Par}(g_2(x)) \cdots \textsc{Par}(g_{m-1}(x)) \cdot \left\lfloor \frac{g_m(x)}{2} \right\rfloor . \qquad (3)
\end{aligned}
$$

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 (2) and (3) show how to build a GapAC^0 circuits 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 (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 9 *The function* $\textsc{ExactHalf}(x) = \lfloor \frac{x_1+\cdots+x_n}{2} \rfloor$ *cannot be computed in* #AC^0. □

Theorem 9 follows from the following stronger statement. Denote by qAC^0 the class of functions computable by a family of unbounded fan-in, constant depth circuits and *quasi-polynomial size* (that is, the size of a circuit for input length n is $2^{\log^{O(1)} n}$). Denote by #qAC^0 the corresponding counting class, and similarly let GapqAC^0 and qAC^0[2] be the quasipolynomial-size analogs of GapAC^0 and AC^0[2], respectively.

Theorem 10 *The function* $\textsc{ExactHalf}(x) = \lfloor \frac{x_1+\cdots+x_n}{2} \rfloor$ *cannot be computed in* #qAC^0. □

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

Proof: We need the following notions:

Definition 4 *[ABFR91] A polynomial P* weakly represents *a Boolean function f if P is not the constant polynomial zero, and $sgn(P(x)) = sgn(f(x))$ for all x where $P(x) \neq 0$. The* weak degree *of a Boolean function f, denoted by $d_w(f)$, is the least integer k for which there exists a degree k polynomial that weakly represents f. (Here the sign of a zero-one Boolean function f is defined as $sgn(f(x)) = 1 - 2f(x)$).*

The idea of the proof is to show that if $\lfloor \frac{x_1+\cdots+x_n}{2} \rfloor$ can be computed in #qAC^0, then there exists a polynomial of small degree that weakly represents the function $\oplus(x_1, \ldots, x_n)$, using the observation that

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

But Aspnes *et al.* showed [ABFR91] that any polynomial that weakly represents the parity function must have large degree.

Lemma 11 *[ABFR91] The weak degree of the parity function is n.* □

Lemma 12 *[ABFR91] Let P be a degree k polynomial and f any Boolean function. Let E be the set of all x for which $sgn(P(x)) \neq sgn(f(x))$. Then if $k < d_w(f)$, we have:*

$$|E| \geq \sum_{i=0}^{\left\lfloor \frac{d_w(f)-k-1}{2} \right\rfloor} \binom{n}{i}. \tag{4}$$

□

Note that

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

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 12, this means that any polynomial p such that $p(x_1, \dots, x_n) = \left\lfloor \frac{x_1+\cdots+x_n}{2} \right\rfloor$ 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 13 *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[1] $0 < \varepsilon \leq 1/S_n$ 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 if $0 < g(x) \leq 2^{\log^c n}$ then $G(x) = g(x)$ with error $\varepsilon/2$.
2. there exists a polynomial H of degree $O\left(z_\varepsilon \log^{cd} n\right)$ such that if $0 \leq g(x) \leq 2^{\log^c n}$ then $H(x) = g(x)$ with error ε.

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

First we show the existence of the polynomial H by supposing that we have shown the existence of the polynomial G satisfying the conditions mentioned 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.* [BRS91, 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[2]. 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, each g_i is of depth $\leq d-1$. Consider first the case of + 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 example $\varepsilon < 1/S_n$.

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 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 $\varepsilon < 1/S_n$.

∎

[2]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.

To complete the argument, suppose that $\lfloor \frac{x_1+\cdots+x_n}{2} \rfloor$ could be computed with a $\#\text{qAC}^0$ circuit of depth D and of size S. We take $c = 1$. By Lemma 13, 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 $P(x) \neq \lfloor \frac{x_1+\cdots+x_n}{2} \rfloor < 2^{\log n}$ 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 Lemma 12. This completes the proof of Theorem 10. ∎

In the full paper 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.

A useful tool for showing non-membership in GapAC^0 was presented by Lu [Lu98]. He gives an exact characterization of symmetric (Boolean) functions computable in $\text{qAC}^0[2]$. 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 sequence $(f(0), f(1), \ldots, f(n))$ is called the *weight spectrum* of f [FKPS85]. The *period* of f is defined as the least integer $k > 0$ such that $f(x) = f(x + k)$ for $0 \leq x \leq n - k$.

Theorem 14 *[Lu98] A symmetric Boolean function f is in $\text{qAC}^0[2]$ 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 14 easily yields non-closure results, of which the following corollary is an example.

Corollary 15 *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 $\#\text{AC}^0$ nor GapAC^0 are closed under taking of roots or logarithms.* □

As a final comment about closure properties, we note that it was shown in [AAD97] that if f is in $\#\text{AC}^0$ (or GapAC^0) and $g(x) = O(1)$, then $\binom{f(x)}{g(x)}$ is in $\#\text{AC}^0$ (GapAC^0, respectively). It remains an open question if closure holds also if g is allowed to be unbounded, although it is observed in [AAD97] that closure does not hold in general if g is superpolylogarithmic. It is perhaps worth noting that the proof in [AAD97] actually shows that for functions g computable in AC^0, both of the classes $\#\text{qAC}^0$ and GapqAC^0 are closed under $\binom{\cdot}{g}$ if and only if g is polylogarithmic. This is related to an open question in [Lu98].

4 Grid Graphs

Grid graphs were introduced into the study of constant-depth circuits in [BLMS98]. In this paper we use an equivalent notion, that makes it formally easier to present our results.

Definition 5 *A* G-graph *is a graph that has a planar embedding in which the vertices are grouped in a rectangular array of constant width (the length is a 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 G-graphs with the same width 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 G-graphs.*

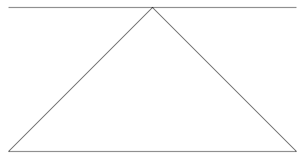

Figure 1:

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 [BLMS98]. 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 illustrated in Figure 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.

Now consider the homomorphism h mapping $\sigma \in \{0, 1\}$ to A^σ. 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). 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 complete for NC^1 ACC^0 reductions.

Theorem 16 *Counting the number of s-t paths in width-two G-graphs is complete for NC^1 under ACC^0 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 [Ba89]. 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 1. (For a proof of this fact, see, e.g., [Gu90, Theorem 3.1].) ∎

Theorem 17 *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 2. Conversely, given 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 2.

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

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

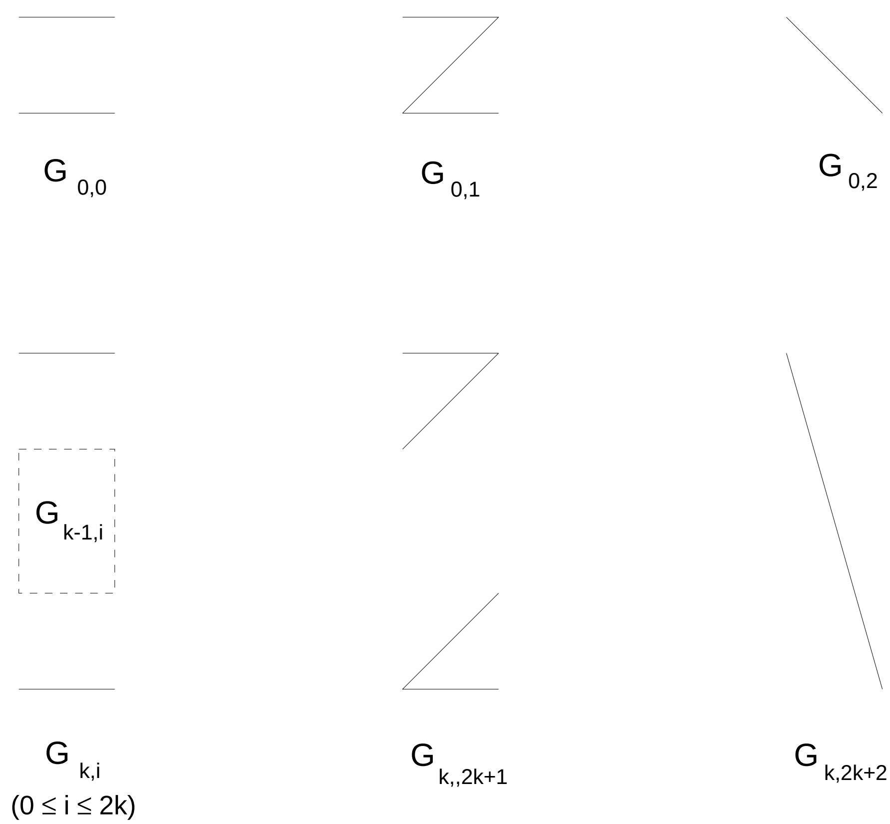

Figure 2: The G-graphs $G_{i,j}$

C is a $\prod$-gate at σ-depth d with inputs $C_1, \dots, C_r$:
$f(C) = f(C_1)G_{k,2d+2}f(C_2)G_{k,2d+2}\dots G_{k,2d+2}f(C_r)$

C is a $\sum$-gate at σ-depth d with inputs $C_1, \dots, C_r$:
$f(C) = G_{k,2d+1}f(C_1)G_{k,2d+1}f(C_2)\dots G_{k,2d+1}f(C_r)G_{k,2d+1}$

The G-graph for the formula $(x_1x_2 + \overline{x_3})(\overline{x_2} + x_1x_4)$ is illustrated in Figure 3.

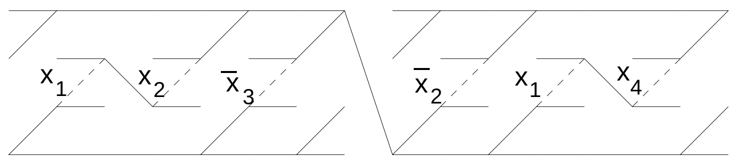

Figure 3: G-graph for $(x_1x_2 + \overline{x_3})(\overline{x_2} + x_1x_4)$

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 $t = t_1(G)$. We show 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. The proof proceeds by induction on the height of the gate.

Clearly $G_{0,0}$ and $G_{0,1}$ have respectively 0 and 1 $s-t$ paths; and since $G_{k,i}$ (for $0 \leq i \leq 2$) is the same as $G_{0,i}$ when restricted to the two middle rows (and since there is no interference from any other row), the proposition holds true for literals and constants.

If C is a $\prod$-gate of σ-depth d, with input gates $C_1, \dots, C_r$, then each C_i has σ-depth d, hence by the inductive assumption, the value computed by C_i is the number of $s_{k-d}(f(C_i)) - t_{k-d}(f(C_i))$ paths. But for $1 \leq i \leq r-1$, $t_{k-d}(f(C_i))$ and $s_{k-d}(f(C_{i+1}))$ are connected by an edge. Thus the number of $s_{k-d}(f(C)) - t_{k-d}(f(C))$ paths is $\prod_{1 \leq i \leq r}$ (*number of* $s_{k-d}(f(C_i)) - t_{k-d}(f(C_i))$ *paths*) which is the same as $\prod_{1 \leq i \leq r}$ (*value computed by* C_i).

If C is a $\sum$-gate of σ-depth d, with input gates $C_1, \dots, C_r$, then each C_i has σ-depth $d-1$, hence by the inductive assumption, the value computed by C_i is the number of $s_{k-d+1}(f(C_i)) - t_{k-d+1}(f(C_i))$ paths, which is same as the number of paths from s_{k-d} to t_{k-d} in the graph $G_{k,2d+1}f(C_i)G_{k,2d+1}$. Thus if we place the $f(C_i)$'s together with the $G_{k,2d+1}$'s between them, we get the sum of all such paths.

Conversely, we have a width $2k$ graph $g'_1 \dots g'_n$ (where each g'_i is one of the $G_{k-1,j}$'s illustrated in Figure 2) and we want to compute the number of $s-t$ paths.

First, we build the width $2k+2$ graph $g_0 \dots g_{n+1}$, 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 [3] :

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

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

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

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

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

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.

∎

Acknowledgment

We thank Jin-Yi Cai for pointing us to [Gu90].

References

[AAD97] M. Agrawal, E. Allender, S. Datta, *On TC^0, AC^0 and arithmetic circuits.* In Proceedings of the 12th Annual IEEE Conference on Computational Complexity, pp:134–148, 1997.

[ABFR91] J. Aspnes, R. Beigel, M. Furst, S. Rudich, *The expressive power of voting polynomials.* In Proceedings of the 23th ACM Symposium on Theory of Computing (STOC), pp:402–409, 1991.

[ABL98] A. Ambainis, D. M. Barrington, H. LêThanh, *On counting AC^0 circuits with negative constants.* In Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), 1998, pp:409–417.

[ABO96] E. Allender, R. Beals, M. Ogihara, *The complexity of matrix rank and feasible systems of linear equations.* In Proceedings of the 28th ACM Symposium on Theory of Computing (STOC), pp. 161–167, 1996.

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

[Al97] E. Allender, *Making computation count: Arithmetic circuits in the nineties.* In the Complexity Theory Column, edited by Lane Hemaspaandra, SIGACT NEWS 28, 4:2–15.

[AJ93] C. Álvarez, B. Jenner, *A very hard logspace counting class.* Theoretical Computer Science, 107:3–30, 1993.

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

[Ba89] D. A. Barrington, *Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in* NC^1. Journal of Computer and System Sciences 38(1):150–164, 1989.

[BLMS98] D. M. Barrington, Ch-J Lu, P. B. Miltersen, S. Skyum, *Searching constant width mazes captures the* AC^0 *Hierarchy.* In Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science, 1998.

[BRS91] R. Beigel, N. Reingold, D. Spielman, *The perceptron strikes back.* In Proceedings of the 6th Annual IEEE Structure in Complexity Theory Conference, 1991.

[CMTV96] H. Caussinus, P. McKenzie, D. Thérien, H. Vollmer, *Nondeterministic* NC^1*Computation.* In Proceedings of the 11th Annual IEEE Conference on Computational Complexity, pp:12–21, 1996.

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

[FFK94] S. A. Fenner, L. J. Fortnow, S. A. Kurtz, *Gap-definable counting classes.* Journal of Computer and System Science, 48(1):116–148, 1995.

[Gu90] Y. Gurevich. *Matrix decomposition Problem is complete for the average case.* In Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp:802–811, 1990.

[Ha86] J. Hastad, *Almost optimal lower bounds for small depth circuits.* In Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, pp:6–20, 1986.

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

[LêT98] H. LêThanh, *Circuits Arithmétiques de Profondeur Constante*, Thesis, Université Paris Sud, 1998.

[Lu98] Ch. J. Lu, *An exact characterization of symmetric functions in* $qAC^0[2]$. In Proceedings of the 4th Annual International Computing and Combinatorics Conference (COCOON), 1998.

[MV97] M. Mahajan, V.Vinay, *A combinatorial algorithm for the determinant.* Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pp:730–738, 1997.

[NS98] F. Noilhan and M. Santha, *Semantical Counting Circuits*, submitted, 1998.

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

[Ra87] A. A. Razborov, *Lower bound on 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.

[Sm87] R. Smolensky, *Algebraic methods in the theory of lower bounds for Boolean circuit complexity.* In Proceedings of the 19th ACM Symposium on the Theory of Computing (STOC), pp:77–82, 1987.

[Tod92a] S. Toda, *Counting problems computationally equivalent to the determinant*. Manuscript.

[Tod92b] S. Toda, *Classes of arithmetic circuits capturing the complexity of computing the determinant*. IEICE Transactions, Informations and Systems, E75-D:116–124, 1992.

[Val79] L. Valiant, *The complexity of computing the permanent*. Theoretical Computer Science, 8:189–201, 1979.

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

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