Electronic Colloquium on Computational Complexity, Report No. 20 (1998)

On Counting AC^0 Circuits with Negative Constants

Andris Ambainis[1], David Mix Barrington[2], and Hương LêThanh[3]

[1] Computer Science Division, University of California at Berkeley
ambainis@cs.berkeley.edu
[2] Computer Science Department, University of Massachusetts
barring@cs.umass.edu
[3] Laboratoire de Recherche en Informatique, Université de Paris-Sud
huong@lri.fr

Abstract. Continuing the study of the relationship between TC^0, AC^0 and arithmetic circuits, started by Agrawal et al. [1], we answer a few questions left open in this paper. Our main result is that the classes DiffAC^0 and GapAC^0 coincide, under poly-time, log-space, or log-time uniformity. From that we can derive that under logspace uniformity, the following equalities hold:

$$C_=AC^0 = PAC^0 = TC^0.$$

1 Introduction

The study of counting complexity classes was started by the pioneering work of Valiant [16] on the class $\#P$. It consists of functions which associate to a string x the number of accepting computations of an NP-machine on x. A well-known complete problem for this class is the computing of the permanent of an integer matrix. The class $\#L$ was defined later analogously with respect to NL-computation [3, 18, 14]. Each of these classes can be defined equivalently either by counting the number of accepting subtrees of the corresponding class of uniform circuits, or by computing functions via the arithmetized versions of these circuit classes [17, 18, 14].

These counting classes contain functions which take only natural numbers as values. Counting classes computing functions which might also take negative values were introduced via the so-called Gap-classes. The class Gap$P$ was defined by Fenner, Fortnow and Kurtz [8], and the class Gap$L$ was introduced by analogy in [18]. For both classes there are two equivalent definitions. They can either be defined as the set of functions computable as the difference of two functions from the corresponding counting class, or as functions which are computed by the corresponding arithmetic circuits augmented by the constant -1.

Recently, counting classes related to circuit model based language classes were also defined. The class $\#NC^1$ was introduced by Caussinus et al. in [7], and the class $\#AC^0$ by Agrawal et al. in [1]. The corresponding Gap-classes were also defined in these papers. The two definitions for GapNC^1 are again

ISSN 1433-8092

easily seen to be equivalent, the principal reason for this being the fact that the PARITY language is in NC^1. The same argument fails to work for the two definitions of GapAC^0 since PARITY can not be computed in AC^0. In fact, one of the problems left open in [1] was the exact relationship between these two classes (GapAC^0 and DiffAC^0 in the notation of the paper).

The main result of our paper is that GapAC^0 and DiffAC^0 actually coincide. We will prove this in the log-time uniform setting, thus showing that it also holds in the log-space uniform and P-uniform settings. As a consequence of this result, we can simplify the relationships among the various boolean complexity classes defined in terms of these arithmetic classes, resolving several open problems of [1]. For example, under log-space uniformity the classes TC^0, $C_=AC^0$ and PAC^0 are all equal. (This result was proven in [1] only under P-uniformity.) Under log-time uniformity, we have the new series of containments $TC^0 \subseteq C_=AC^0 \subseteq PAC^0$.

2 Preliminaries

Following [1], we will consider three notions of uniformity for circuit families. A family $\{C_n\}_{n\geq 1}$ of circuits is said to be *P uniform* if there exist a Turing machine $\mathcal{M}$ and a polynomial $T(n)$ such that $\mathcal{M}$, given n in unary, produces a description of the circuit C_n within time $T(n)$. Similarly, a family is *log-space uniform* if there exists a Turing machine that produces a description of C_n using space $O(\log n)$.

The definition of *log-time uniformity* [5] is a bit more complicated. A family $\{C_n\}$ of circuits is said to be *log-time uniform* if there is a Turing machine that can answer queries in its *direct connection language* in time $O(\log n)$. The direct connection language consists of all tuples $\langle i, j, t, y\rangle$ where i is the number of a gate of C_n, j is the number of one of its children (or the number of the referenced input x_j, if gate i is an input gate), t gives the type of gate i, and y is any string of length n. The log-time Turing machine has a read-only random-access input tape, so that it can, for example, determine the length of its input by binary search. As shown in [5] and [4], log-time uniform circuits are equivalent in power to circuits given by *first-order formulas* with variables for input positions and atomic predicates for order, equality, and binary arithmetic on these variables.

By De Morgan's law, it is sufficient to consider circuits in which negations occur only on the input level, and all the other gates are OR-gates or AND-gates. For such circuits the notion of *subtree* was introduced in [17]. Let C be a Boolean circuit and let $T(C)$ be the circuit obtained from C by duplicating all gates whose fan-out is greater than one, until the underlying graph of $T(C)$ is a tree. Let x be an input of C. A subtree H of C on input x is a subtree of $T(C)$ defined as follows:

- The output gate of the circuit $T(C)$ belongs to H.
- For each non-input gate g already belonging to H, if g is an AND-gate then all its input gates belong to H, and if g is an OR-gate then exactly one of its input gates belongs to H.

A subtree on input x is said to be an *accepting* subtree if all its leaves evaluate to 1.

We now define how to *arithmetize* a Boolean circuit. The input variables $x_1, x_2, \ldots, x_n$ take as values the natural numbers 0 or 1, and the negated input variables $\bar{x_i}$ take the values $1 - x_i$. Each OR-gate is replaced by a $+$-gate and each AND-gate by a $\times$-gate. It was shown in [17] that the number of accepting subtrees of the circuit C on input $(x_1, \bar{x_1}, \ldots, x_n, \bar{x_n})$ is equal to the output of its arithmetized circuit on the same input.

Note that the output of such an arithmetic circuit is always non-negative. If the constant -1 is allowed in the circuit, functions with negative values can also be computed.

Let $\#\mathcal{C}$ be a class of functions from $\{0,1\}^*$ to $\mathbb{N}$. By definition, $\#\mathcal{C} - \#\mathcal{C}$ is the class of functions expressible as the difference between two functions from $\#\mathcal{C}$.

Definition 1 *[1] Let U be any of our three uniformity definitions: P, log-space, or log-time. For any $k > 0$, U-uniform $\#AC^0_k$ ($GapAC^0_k$) is the class of functions computed by depth k, polynomial size, U-uniform circuits with $+,\times$-gates having unbounded fan-in, where inputs of the circuits are from $\{0, 1, x_i, 1 - x_i\}$ (from $\{0, 1, -1, x_i, 1 - x_i\}$) and $x_i \in \{0, 1\}$ for all $i = 0, \ldots, n$. Let*

$$\begin{aligned} \#AC^0 &= \bigcup_{k>0} \#AC^0_k, \\ \mathit{DiffAC}^0 &= \#AC^0 - \#AC^0, \\ \mathit{GapAC}^0 &= \bigcup_{k>0} \mathit{GapAC}^0_k. \end{aligned}$$

It is easy to see that under all three uniformity conditions, Diff$AC^0 \subseteq$ GapAC^0. A very natural question, left open by Agrawal et al. [1], is whether DiffAC^0=GapAC^0.

Let PARITY denote the usual 0-1 parity function which computes the sum of its inputs modulo 2, and let F-PARITY be its *Fourier representation*, that is F-PARITY $(x_1, \ldots, x_n) = \prod_{i=1}^n (1 - 2x_i)$. Note that the range of this function is $\{1, -1\}$. It is clear that F-PARITY is in GapAC^0. Another open question was whether this function belongs to DiffAC^0.

In the next sections we will give a positive answer to both questions. By a $\#AC^0$ circuit we mean an arithmetized AC^0 circuit in the above sense. Throughout this paper we will need the following fact.

Fact 1 *For each integer N of m bits there exists a $\#AC^0$ circuit with $O(m^2)$ gates, which on input 1^m computes N. This circuit is log-time uniform if the binary representation of N is given as input.*

Proof. We use the circuit C_r introduced in [1] to denote the $\#AC^0$ circuit of depth 2 and size $(3r + 1)$, whose number of accepting subtrees on input 1^{2r} is 2^r. Observing that

$$2^r = \underbrace{(1+1)\cdot(1+1)\cdots(1+1)}_{r \text{ times}},$$

C_r will contain at the input level $2r$ constants 1, r OR-gates of depth 1 and fan-in 2, and one AND-gate of depth 2, whose inputs are all the OR-gates. Note that the family of circuits $\{C_r\}_{r\geq 1}$ is both P-uniform and Logspace-uniform. Let $N = N_{m-1}N_{m-2}\dots N_1N_0$ be the binary representation of N. The formula $N = \sum_{i=0}^{m-1} N_i \cdot 2^i$ will give a $\#AC^0$ circuit of depth 3 and size $O(m^2)$ computing N. (Note that a log-time Turing machine can use its random access input tape to reference the single bit of N needed to answer any particular query.) □

3 DiffAC^0 = GapAC^0

Theorem 1 *DiffAC^0 = GapAC^0 for log-time uniform circuits (and hence for P uniform and log-space uniform circuits as well).*

Proof. It is enough to show that GapAC^0 $\subseteq$ DiffAC^0 under each uniformity condition. We will first describe our general construction, and then show that it can be carried out preserving log-time uniformity (and hence the other two conditions as well). Given an arithmetic circuit C for the inputs of length n, using the constant -1, we will construct two other arithmetic circuits A and B, each with only positive constants, such that for all input x of length n, we have $C(x) = A(x) - B(x)$. We will show, by induction on the depth of C, that for each gate g, we can build two $\#AC^0$ circuits A^g and B^g such that $g(x) = A^g(x) - B^g(x)$.

The construction is trivial for gates of depth 0. Consider now a gate g of depth $d \geq 1$ having as input gates $g_1, g_2, \dots, g_m$. Suppose that for each $i = 1, \dots, m$, we have already constructed two $\#AC^0$ circuits A_i^g and B_i^g satisfying $g_i(x) = A_i^g(x) - B_i^g(x)$. If g is a $+$-gate, the construction of A^g and B^g is straightforward. The interesting case is when g is a $\times$-gate. For ease of notation we set $a_i = A_i^g(x)$ and $b_i = B_i^g(x)$.

Without any negative constants, we can compute the product $\prod_{i=1}^{m}(a_i + b_i)$, which is of no immediate help in getting $\prod_{i=1}^{m}(a_i - b_i)$. The key idea is to notice that we can also compute some other products of positive linear combinations of the a_i's and b_i's as well, such as $\prod_{i=1}^{m}(a_i + 2b_i)$, and use linear algebra to solve for the combination we want.

Specifically, we will find a sequence of integers $c_1(m), c_2(m), \ldots, c_{m+1}(m)$, each of which depends only on m and has $O(m)$ bits, such that the following equalities hold:

$$\begin{aligned}
&\prod_{i=1}^{m}(a_i - b_i) && (1)\\
&= \sum_{k=1}^{m+1} c_k(m) \cdot \prod_{i=1}^{m}(a_i + k \cdot b_i) && (2)\\
&= \sum_{k: c_k(m)>0} c_k(m) \cdot \prod_{i=1}^{m}(a_i + k \cdot b_i) - \sum_{k: c_k(m)<0} (-c_k(m)) \cdot \prod_{i=1}^{m}(a_i + k \cdot b_i). && (3)
\end{aligned}$$

We must show that this sequence of integers exists, and that each $c_k(m)$ is computable by a log-time uniform $\#AC^0$ circuit. Then the log-time uniform circuits A^g and B^g can easily be constructed to calculate the two sums in expression (3), and the difference will be the value of g as desired.

Let us develop the product (1):

$$\begin{aligned}
&\sum_{k=1}^{m+1} c_k(m) \cdot \prod_{i=1}^{m}(a_i + k \cdot b_i)\\
&= \sum_{k=1}^{m+1} c_k(m) \left(\sum_{j=0}^{m} \sum_{t_1,\ldots,t_{m-j},i_1,\ldots,i_j} k^j a_{t_1} \cdots a_{t_{m-j}} \cdot b_{i_1} \cdots b_{i_j} \right)\\
&= \sum_{k=1}^{m+1} \sum_{j=0}^{m} \sum_{t_1,\ldots,t_{m-j},i_1,\ldots,i_j} k^j \cdot c_k(m) a_{t_1} \cdots a_{t_{m-j}} \cdot b_{i_1} \cdots b_{i_j}\\
&= \sum_{j=0}^{m} \sum_{t_1,\ldots,t_{m-j},i_1,\ldots,i_j} \left(\sum_{k=1}^{m+1} k^j \cdot c_k(m) \right) a_{t_1} \cdots a_{t_{m-j}} \cdot b_{i_1} \cdots b_{i_j}.
\end{aligned}$$

Thus for each j and each m-tuple $(t_1, \ldots, t_{m-j}, i_1, \ldots, i_j)$, the coefficient of the monomial $a_{t_1} \cdots a_{t_{m-j}} \cdot b_{i_1} \cdots b_{i_j}$ in the expression (2) is $\sum_{k=1}^{m+1} k^j \cdot c_k(m)$, whereas the coefficient for the same term in the expression (1) is $(-1)^j$. Therefore the values $c_k(m)$ are exactly the solutions of the following linear system of equations:

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

for all $j = 0, \ldots, m$. Call $M^{(0)}$ the corresponding matrix of this linear system. By Cramer's rule its solutions $c_k(m)$ are:

$$c_k(m) = \frac{det(M^{(k)})}{det(M^{(0)})},$$

where $M^{(k)}$ is the matrix obtained from $M^{(0)}$ by replacing the k-th column of $M^{(0)}$ by the column vector $(1, -1, (-1)^2 \dots, (-1)^m)^T$. For $0 \leq i \leq m$, the matrices $M^{(i)}$ are Vandermonde matrices, therefore the determinant of $M^{(0)}$ is clearly different from 0, and a simple calculation gives:

$$\begin{aligned}
c_k(m) &= \frac{\left(\prod_{i=1}^{k-1}(-1-i)\right)\left(\prod_{j=k+1}^{m+1}(j-(-1))\right)}{\left(\prod_{i=1}^{k-1}(k-i)\right)\left(\prod_{j=k+1}^{m+1}(j-k)\right)} \\
&= \frac{\left[(-1)^{k-1}\cdot 2\cdot 3\cdot 4\cdots k\right]\left[(k+2)(k+3)\cdots(m+2)\right]}{\left[(k-1)(k-2)\cdots 2\cdot 1\right]\left[1\cdot 2\cdot 3\cdots(m+1-k)\right]} \\
&= (-1)^{k-1}\cdot\frac{k(k+2)(k+3)\cdots(m+2)}{(m+1-k)!} \\
&= (-1)^{k-1}\cdot k\cdot\binom{m+2}{k+1}.
\end{aligned}$$

The last expression shows that the numbers $c_k(m)$ are indeed integers, and thus that our desired circuits exist at least in a non-uniform setting. It remains to show only that each number $c_k(m)$ can be computed by a log-time uniform $\#AC^0$ circuit. This follows immediately from the following:

Lemma 1 *The binomial coefficient $\binom{n}{k}$ can be computed by a log-time uniform $\#AC^0$ circuit, whose inputs are the numbers n and k in unary.*

Proof. If p is any boolean predicate in log-time uniform AC^0, its characteristic function $[p]$ (equal to the integer 1 if p is true and the integer 0 otherwise) is in log-time uniform $\#AC^0$. This is because, as observed in [1], AC^0 circuits can be systematically transformed so they have either one accepting subtree or none at all. By [5], then, any property of numbers, expressible in first-order logic over positions in the input, also has a characteristic function in $\#AC^0$. Because we can use constant-length tuples of positions to represent numbers, we can use first-order logic on numbers that are polynomial in the input size. Among the functions available to us on such numbers are $[m$ is prime$]$, $[i$ divides $m]$, and so forth. Note that these same functions are *not* necessarily in $\#AC^0$ on larger numbers, such as $\binom{n}{k}$ itself.

To calculate $\binom{n}{k}$, we will determine its prime factorization. That is, for each prime $p \leq n$, we will calculate the number $Pow(p, n, k)$, the largest power of p dividing $\binom{n}{k}$. Then we have

$$\binom{n}{k} = \prod_{p=2}^{n}\left([p \text{ is composite}] + [p \text{ is prime}]\cdot Pow(p, n, k)\right),$$

which is clearly in log-time uniform $\#AC^0$ as long as each $Pow(p, n, k)$ is.

Fixing a prime p, we can calculate the number of p's occurring in each of the two products $\prod_{i=n-k+1}^{n} i$ and $k!$ as follows. The top product has exactly $\lfloor n/p \rfloor - \lfloor (n-k)/p \rfloor$ terms divisible by p, exactly $\lfloor n/p^2 \rfloor - \lfloor (n-k)/p^2 \rfloor$ divisible by p^2, and so on, so that the total number of p's is

$$\sum_{j=1}^{\lfloor \log_p n \rfloor} \left(\lfloor n/p^j \rfloor - \lfloor (n-k)/p^j \rfloor \right).$$

Similarly, the number of p's in $k!$ is

$$\sum_{j=1}^{\lfloor \log_p k \rfloor} \lfloor k/p^j \rfloor.$$

For each j, the j'th terms of these sums differ by either one or zero. Thus, we can calculate the number $Pow(p,n,k)$ as a product of a number for each j, which is either 1 or p:

$$\prod_{j=1}^{\lfloor \log_p n \rfloor} \left\{ 1 + (p-1) \cdot \left[\lfloor \frac{n}{p^j} \rfloor \; > \; \lfloor \frac{n-k}{p^j} \rfloor + \lfloor \frac{k}{p^j} \rfloor \right] \right\}.$$

Since n, k, and p^j are each polynomial in the input size, this function is in log-time uniform $\#AC^0$. □

We have now completed the proof of Theorem 1. □

4 A Few Consequences

Corollary 1 *F-PARITY is in log-time uniform DiffAC^0.*

Corollary 2 *PARITY is in log-time uniform DiffAC^0.*

Proof. Vinay pointed out [1] that PARITY is represented by the following polynomial:

$$\sum_{i=1}^{n} \left(\prod_{j=1}^{i-1} (1-2x_j) \right) x_i.$$

The corollary thus follows immediately from Theorem 1. □

Define LDiffAC^0 to be the class of languages whose characteristic function is in DiffAC^0. Agrawal et al. proved in [1] that every language in AC^0 has its characteristic function in $\#AC^0$, hence in DiffAC^0. They left open the question whether there exists a language in AC^0 whose characteristic function is not in

LDiffAC^0. It is well known that PARITY is not in AC^0. Therefore with Corollary 2 we have separated AC^0 from LDiffAC^0.

In their paper [1], Agrawal et al. showed that the class $AC^0[2]$ is exactly the class of languages whose characteristic function is in GapAC^0. Combining this result with our result will give $AC^0[2] = \text{LDiff}AC^0$. Because of the result in [12] showing that the MAJORITY is not computable in $AC^0[2]$, we have:

$$AC^0 \subsetneq AC^0[2] = \text{LDiff}AC^0 \subsetneq TC^0.$$

Note that the modulus 2 is special. If p is any *odd* prime number, the MOD-p function is provably *not* in GapAC^0, because (as noted in [1]) the low-order bit of any GapAC^0 function is in $AC^0[2]$ and it is known [13] that MOD-p is not in $AC^0[2]$.

Also, our result DiffAC^0-GapAC^0 implies that all properties known for one of these classes are true for the other too (like normal form or the closure under the weak product defined in [1]).

For the next results we need some more definitions:

Definition 2 *[1] The class $C_=AC^0$ ($C_=AC^0_{circ}$) consists of those languages L for which there exists a function f in DiffAC^0 (GapAC^0) such that for all bit strings x the following holds:*

$$x \in L \iff f(x) = 0.$$

The class $PAC^0(PAC^0_{circ})$ is defined in a similar way where the condition $f(x) = 0$ is replaced by $f(x) > 0$.

Corollary 3 *Under the log-space or log-time uniform model the following equalities hold:*

$$\begin{aligned} C_=AC^0 &= C_=AC^0_{circ}, \\ PAC^0 &= PAC^0_{circ}. \end{aligned}$$

Proof. This follows immediately from Theorem 1. The corresponding equality for P uniformity was shown in [1]. □

Corollary 4 *For log-space uniform circuits :*

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

Proof. It was shown in [1] that in the log-space uniform setting we have:

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

The result thus follows immediately from Corollary 3. □

Corollary 5 *For log-time uniform circuits:*

$$TC^0 \subseteq C_=AC^0 \subseteq PAC^0$$

Proof. Again, this follows immediately from Corollary 3 and the corresponding result in [1]. □

These three classes may be equal under log-time uniformity as well, but the proof in [1], that $PAC^0_{circ} \subseteq TC^0$ in the log-space uniform setting, appears to make essential use of techniques that are only known to be log-space uniform.

We can also answer another open question of Agrawal et al. [1]:

Corollary 6 *Logspace-uniform PAC^0 is closed under union and intersection. Logspace-uniform $C_=AC^0$ is closed under complement.*

Proof. TC^0 is clearly closed under union, intersection, and complement, so this follows from Corollary 4. □

Do these closure properties hold under log-time uniformity as well? This remains an open question.

5 Acknowledgments

We would like to thank Michel De Rougemont and Miklos Santha for helpful discussions. In addition, we would like to thank Eric Allender, who greatly facilitated our collaboration on this work.

References

1. 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.
2. 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.
3. C. Álvarez, B. Jenner, *A very hard logspace counting class.* Theoretical Computer Science, 107:3–30, 1993.
4. D. A. M. Barrington, N. Immerman, *Time, Hardware, and Uniformity.* In L. A. Hemaspaandra and A. L. Selman, eds., *Complexity Theory Retrospective II*, Springer Verlag, pp:1–22, 1997.
5. D. A. M. Barrington, N. Immerman, and H. Straubing, *On Uniformity Within NC^1.* Journal of Computer and System Science, 41:274–306, 1990.
6. P. Beame, S. Cook, H. J. Hoover, *Log depth circuits for division and related problems.* SIAM Journal on Computing, 15:994–1003, 1986.
7. H. Caussinus, P. McKenzie, D. Thérien, H. Vollmer, *Nondeterministic NC^1.* In Proceedings of the 11th Annual IEEE Conference on Computational Complexity, pp:12–21, 1996.
8. S. A. Fenner, L. J. Fortnow, S. A. Kurtz, *Gap-definable counting classes.* Journal of Computer and System Science, 48(1):116–148, 1995.

9. J. Köbler, U. Schöning, J. Torán, *On counting and approximation.* Acta Informatica, 26:363–379, 1989.
10. B. Litow, *On iterated integer product.* Information Processing Letters, 42(5):269–272, 1992.
11. M. Mahajan, V. Vinay, *Determinant: Combinatorics, Algorithms and Complexity.* In Proceedings of SODA'97. ftp://ftp.eccc.uni-trier.de/pub/eccc/reports/1997/TR97-036/index.html
12. A. A. Razborov, *Lower bound on size of bounded depth networks over a complete basis with logical addition.* Mathematicheskie Zametli, 41:598–607, 1987. English translation in Mathematical Notes of the Academy of Sciences of the USSR, 41:333–338, 1987.
13. 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.
14. S. Toda, *Classes of arithmetic circuits capturing the complexity of computing the determinant.* IEICE Transactions, Informations and Systems, E75-D:116–124, 1992.
15. S. Toda, *Counting problems computationally equivalent to the determinant.* Manuscript.
16. L. Valiant, *The complexity of computing the permanent.* Theoretical Computer Science, 8:189–201, 1979.
17. H. Venkateswaran, *Circuit definitions of non-deterministic complexity classes.* SIAM Journal on Computing, 21:655–670, 1992.
18. V. Vinay, *Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits.* In Proceedings of the 6th IEEE Structure in Complexity Theory Conference, pp:270–284, 1991.