Arithmetic Circuits: A Survey

David Mix Barrington
UMass Amherst Theory Seminar
21 February 2012

Sources: Vollmer, *Introduction to Circuit Complexity*, Chapter 5
Allender, *Arithmetic Circuits and Counting Complexity Classes*

Arithmetic Circuits: A Survey

- Boolean Circuit Complexity
- Counting Classes
- Basic Arithmetic Circuit Classes
- Ben-Or-Cleve: NC^1 Arithmetic Circuits = Matrix Programs
- Open Problems: $\#BWBP$ vs. $\#NC^1$, NC^1 vs. $GapNC^1$
- The Classes $\#AC^0$, $DiffAC^0$, and $GapAC^0$
- Characterizing TC^0 With Arithmetic Circuits

Boolean Circuit Complexity

- We can measure the complexity of problems by the **size** and **depth** of the **circuit families** needed to compute them. A family has one circuit for each input size n, and size and depth are functions of n. We also need to constrain the **uniformity** of the family, for example by having all the circuits produced by a resource-bounded machine, or defined by a single logical formula.

- The class P, up to uniformity, corresponds to poly-size boolean circuits. Within P we have the NC hierarchy of circuit classes, where NC^i and AC^i are each defined by circuits of poly size and depth $O(\log^i n)$, with fan-in two and unbounded fan-in respectively. The more familiar classes L and NL lie between NC^1 and AC^1, with SAC^1 or LOGCFL between NL and AC^1.

- The smallest class AC^0 is known not to contain the XOR function. Adding mod gates for one prime to AC^0 does not give you other primes, but adding mod 6 gates defeats all known lower bound techniques -- maybe $AC^0[6] = NP$.

The Class NC^1

- A log-depth, fan-in two boolean circuit may be arranged as a tree by duplicating gates, and still has poly size. Any poly-size tree circuit has an equivalent log-depth tree circuit by a **tree balancing argument**.

- Every regular language is in NC^1, because running a DFA can be thought of as evaluating a product in a **finite monoid** and this can be done with a binary tree of subcircuits that each have O(1) size and depth. But some regular languages, corresponding to **nonsolvable** (particularly non-commutative) monoids, are complete for NC^1 -- we can convert a log-depth boolean circuit into an iterated multiplication in such a monoid.

- We can carry out **iterated addition** (and hence multiplication) of binary integers in NC^1. This problem may well not be complete for NC^1-- the subclass TC^0 of NC^1 is what we can do with constant-depth unbounded fan-in **majority gates**. By careful use of the Chinese Remainder Theorem, we can do **iterated multiplication** of binary integers in TC^0.

Counting Classes

- Last week Marco defined the **function classes** #P and #L. A function f from {0,1}* to **N** is in #P if there is a poly-time NDTM M such that for any string x, f(x) is the number of **accepting paths** of M on x. #L is the same for a log-space NDTM. To get functions from {0,1}* to $\mathbb{Z}$, we define GapP and GapL to be the functions that are the difference of two #P or #L functions respectively.

- We can also count **accepting subtrees** of a circuit -- subtrees that include at least one child of each of their OR nodes, all children of each of their AND nodes, and only 1's at leaves. When we use circuit characterizations of NL and NP (which we'll see soon), #P and #L fit in with the counting circuit classes $\#SAC^1$, $\#NC^1$, and AC^0.

- We can count the accepting subtrees of a boolean circuit by evaluating the corresponding **arithmetic circuit**, replacing AND by × and OR by +.

Basic Arithmetic Circuit Classes

- What happens if we place size and depth constraints on families of arithmetic circuits? Poly-size circuits pose a problem in that in general × gates *square* the largest number available, so poly depth gives exponential size numbers.

- If we define **degree** of a circuit (by max for + and adding for ×), we guarantee that poly-size circuits only create numbers of poly-many bits.

- Poly size and poly degree gives $GapSAC^1$, not GapP -- the latter turns out to be *exponential* size and poly degree (following a circuit definition of NP). GapL turns out to be poly-size **skew circuits** where one input of every × gate must be an input.

- $GapNC^1$ and $GapAC^0$ have sensible definitions in terms of arithmetic circuits over integers. We also have # classes defined by circuits over **N** rather than $\mathbb{Z}$.

Ben-Or-Cleve: Circuits as Matrix Products

- As with boolean NC^1, we can turn a $GapNC^1$ circuit into an iterated product, in this case over 3×3 matrices of integers.

- We show that for any $i \neq j$, we can form a matrix with 1's on the diagonal and one entry for f, the circuit value, in the ij position. If we can make both f and -f for any function, we can implement + gates by simple product and × gates by a commutator construction as $\mathbb{Z}^{3\times 3}$ is suffficiently non-commutative. A circuit of depth d turns into a product of at most 4^d matrices.

$$\begin{vmatrix} 1 & x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & x+y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -x & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & y \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & x & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & -y \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & y \\ 0 & 1 & 0 \\ 0 & -x & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & -y \\ 0 & 1 & 0 \\ 0 & x & 1 \end{vmatrix} = \begin{vmatrix} 1 & xy & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Two Open Problems in the $\#NC^1$ Area

- But this trick crucially depends on the existence of additive inverses in $\mathbb{Z}$. If we look at $k \times k$ matrices over **N**, we can define the class #BWBP of functions defined by poly-length products (or by counting paths through a **bounded width branching program**). It is easy to see that $\#BWBP \subseteq \#NC^1$, but the opposite inclusion is *open*. (There seems to no reason it should be true, but we have no lower bound techniques yet against #BWBP.)

- By Chinese remaindering, we can implement iterated multiplication in $\mathbb{Z}^{3\times 3}$ by iterated multiplications in parallel over $\mathbb{Z}_p{}^{3\times 3}$ for poly-many primes of O(log n) bits. This costs us NC^1 (actually TC^0) for the translation in and out of CRR. We can repeat the process until we have multiplication over matrices with constant modulus, which is in NC^1. So we can compute $GapNC^1$ with boolean circuits of fan-in two and depth O(log n log*n), or majority circuits of depth O(log*n). This is *very very close* to collapsing this class to boolean NC^1.

Arithmetic Versions of AC^0

- $\#AC^0$ is defined in terms of unbounded fan-in + and × gates, with 0 and 1 inputs. $GapAC^0$ actually poses a problem in definition, as it is not obvious that the two definitions we have used coincide. From the circuit standpoint we would allow -1 constant gates, but starting from the GapP definition we would look at the difference of two $\#AC^0$ functions. Perhaps confusingly, the class with -1 constants is called "$GapAC^0$" and the other "$DiffAC^0$". The good news is that the two classes coincide. (At all the higher classes we had the parity function, the sum of the inputs mod 2, available in the # class.)

- The construction of a $DiffAC^0$ pair for an arbitrary $GapAC^0$ is recursive -- for each gate g we construct two $\#AC^0$ functions P and N such that $g = P - N$. This is easy for constants, and for a gate that is the sum of other gates g_i each with its own pair of functions P_i and N_i. The difficulty is to compute P and N such that P - N is the *product* of the functions $P_i - N_i$.

Simulating a Product Gate in DiffAC^0

- The product of $(P_i - N_i)$ has 2^n terms -- for every set $L \subseteq \{1,\ldots,n\}$ and its complement R, we have $(-1)^{|L|}\prod_L P_i \prod_R N_i$ which we'll call $(-1)^j f(L, R)$.

- The trick is to express this sum as an integer linear combination of n+1 different products, $X_k = \prod_n (P_i + kN_i)$ for each k from 0 through n. Each of these products can be computed in #AC^0 assuming that each P_i and N_i can.

- We compute each X_k as a sum of the products f(L, R), define variables c_k for k from 0 through n, and equate $\sum_k c_k X_k$ to our product above. Collecting the sets of terms for each |L|, we wind up with the set of linear equations $\sum_k k^j c_k = (-1)^j$, n+1 equations in the n+1 unknowns. The matrix of this set of equations turns out to be a Vandemonde matrix and can be solved to give us the coefficients $c_k = (-1)^{k+1}k(n+2 \text{ choose } k+1)$.

TC^0 in Terms of Arithmetic Circuits

- As Marco told us last week, there are several ways to take a function class like #P and make a language class from it. For example, NP is the set of languages L such that there is a #P function f such that $f(x) > 0$ iff $x \in L$. UP is the set of languages L such that the characteristic function χ_L is in #P.

- $C_=P$ is the set of L such that for some f in GapP, $f(x) = 0$ iff $x \in L$. PP is the corresponding set for $f(x) > 0$. It turns out that the classes $C_=AC^0$ and PAC^0 are both equal to the already-defined boolean circuit class TC^0.

- Since iterated addition and iterated multiplication are in TC^0, even under log-time uniformity, a TC^0 circuit can just evaluate a $GapAC^0$ function and make the proper comparison at the end.

- Evaluating a TC^0 circuit with a $GapAC^0$ function is a bit harder.

Simulating TC^0 With a $GapAC^0$ Function

- By a bit of hacking, we can show how to convert any TC^0 circuit into one that has only **exact threshold** gates, that have m inputs and return 1 iff the inputs are exactly evenly split between 0 and 1. We also want the circuit to be **levelled**, so that every path from a given gate to an input has the same length.

- We let μ be the product $\prod_{j \neq m/2} (m/2 - j)$. For each gate g at level t of the exact threshold circuit, we'll construct a $GapAC^0$ function that is 0 if g evaluates to 0, and μ^t if g evaluates to 1.

- Suppose that $f_1,\ldots,f_n$ are each $GapAC^0$ functions that are equal to μ^t or 0 as the gates $g_1,\ldots,g_n$ are 1 or 0, and that g is the exact threshold of the g_i's. The product $\prod_{j \neq m/2} ((\sum_i f_i) - j\mu^t)$ is clearly in $GapAC^0$, and evaluates to 0 unless exactly half the f_i's are nonzero. In this case the product is μ^{t+1}.

Lower Bounds Against GapAC^0

- Let f(n) be the familiar Fibonacci function. Given a string of boolean inputs $x = x_1,\ldots,x_n$ we can let $F(x) = f(\sum_i x_i)$. This is a natural-number function ranging from 0 to f(n).

- If h is any GapAC^0 function, look at the boolean function b given by the **low-order bit** of h. Suppose we take the circuit for h and replace each + gate with a boolean $\oplus$ gate, and each $\times$ gate with a boolean $\wedge$ gate. We now have an AC^0[2] circuit that computes b. It's fairly easy to see that a boolean function is in AC^0[2] if and only if it is the low-order bit of a GapAC^0 function.

- Smolensky's Theorem says that the mod-3 function cannot be computed by an AC^0[2] circuit. Thus our F function above cannot possibly be in GapAC^0.

- There is a hope (not yet realized) that such lower bounds could help in proving lower bounds against TC^0. They do separate GapAC^0 from GapNC^1.