The Current State of Circuit Lower Bounds

David Mix Barrington
COINS Dept., U. of Massachusetts
Amherst MA 01003, U.S.A.
June 21, 1993

1. Abstract

We review the existing results showing languages to be outside of complexity classes defined by tight constraints on boolean circuits, using the new characterizations of these classes in terms of automata theory [BT88] and formal logic [BIS88]. We outline the methods and results in the case of three types of classes defined by circuits of constant depth, polynomial size, and unbounded fan-in, for three different types of gates. These are AND and OR gates [FSS84], AND, OR, and MOD-p gates [Ra87, Sm87], and modular gates alone with certain other restrictions [BST90]. Most of this survey was prepared for the McGill University Workshop in Theoretical Computer Science, held in February 1989.

2. Introduction

It would be nice to show that the CLIQUE problem is not solved by a family of boolean circuits where circuit size grows polynomially with input size. This would not only show $P \neq NP$, but also would demonstrate our "understanding" of polynomial-size circuits in a particular sense. We would then be provably able to separate languages computable by such circuits from languages not computable by them but near them in the complexity hierarchy.

Less ambitiously, it would be nice to show this sort of understanding of the class NC^1 (languages computable by boolean circuits of depth $O(\log n)$ and fan-in two, or by boolean formulas of polynomial length), a class of "easy" problems in our usual reckoning. So far, however, we cannot show a language to be outside of NC^1 unless it is very complex indeed — we cannot rule out the possibility $NC^1 = NP$.

One approach to this goal has been to move even lower in the complexity hierarchy, to classes which we do "understand". In this survey we will examine the techniques currently known for showing languages to be outside of circuit complexity classes. Our

point of view will be based on the characterizations of subclasses of NC^1 in terms of algebraic automata theory and formal logic as well as circuits. This approach gives a new way of organizing the existing lower bound results, and has led to some results of its own.

3. The Framework

We will look at three ways to define complexity classes within NC^1: circuits, programs over monoids, and first-order logic. We first consider circuits of constant depth, unbounded fan-in, and polynomial size. If we allow the gates to be ordinary AND and OR gates (and the inputs to be variables or negated variables) we get the class AC^0. As we shall see below, we already "understand" this class because we can take a relatively simple language (the binary strings with an odd number of ones) and prove that it is not in AC^0 [FSS84, Aj83].

By allowing new operations as well we get the AC^0 *closure* of those operations. To be specific, a language is in the AC^0 closure of a family $\mathcal{F}$ of functions if it can be computed by a constant-depth, polynomial-size family of unbounded fan-in circuits of AND gates, OR gates, and gates for functions in $\mathcal{F}$. Two popular candidates are the parity or exclusive OR operation (returning one iff an odd number of the inputs are one) and the majority operation (returning one iff at least half of the inputs are one).

It is not hard to show [FSS84] that parity is in the AC^0 closure of majority and thus that majority is itself not in AC^0. We shall see below that majority is *not* in the AC^0 closure of parity [Ra87] — we understand this class well enough to show this. In fact, if we define a "modulo p" operation appropriately for any prime p, we understand the AC^0 closure of modulo p well enough to exclude majority (and modulo q for any integer q not divisible by p) from it [Sm87]. This understanding does not extend to the closure of modulo q if q is not prime, and it is consistent with what we know so far that the closure of modulo 6 is NP. It seems unlikely that AND, OR, and constant-modulus counting gates could even do all of NC^1. Confirming this intuition would extend our understanding to the class ACC_0 (also called ACC), which is defined to be the union over all integers q of the AC^0 closure of the modulo q operation.

Since the majority operation is in NC^1 and NC^1 is its own AC^0 closure, the closure of majority is a subclass of NC^1, called TC^0 (because of an alternate characterization in terms of threshold gates [PS88]). A large number of natural problems, such as integer multiplication, are known to be AC^0 equivalent to majority (i.e., their AC^0

closure is exactly TC^0) [CSV84]. It is conjectured [HMPST87] that TC^0 is strictly contained in NC^1, but this is still open. (For the best known lower bound results, on depth-2 circuits of threshold gates, see [HMPST87].)

A surprising consequence of the simulation of NC^1 circuits by width 5 branching programs [Ba89] is that NC^1 itself can be characterized in terms of such circuits, with a new gate type. Such a gate interprets its input bits as a sequence of elements of a fixed non-solvable finite group (such as the group S_5 of permutations on five letters) and outputs a single bit determined by the product of this sequence of elements in the group. In fact the construction of [Ba89] reduces any NC^1 calculation to a single iterated multiplication (of polynomial length) in S_5.

In this way NC^1 is the AC^0 closure of an operation which can be performed by a finite state machine. This leads to the consideration of a new model of computation, *programs over finite monoids* (also called "non-uniform automata" or "NUDFA's") [BT88]. A *monoid* is a set together with a binary operation which is associative and has an identity. Let A be a finite alphabet (in applications to boolean circuits, A will usually be the set $\{0,1\}$). The action of an ordinary deterministic finite automaton on alphabet A can be viewed as a homomorphism from the monoid $A^\star$ of strings (under the operation of concatenation) to a particular finite monoid M of functions on the states of the automaton (under the operation of composition). Each letter of the input gives rise to a transformation of the states, and we multiply all these transformations together in order to get the result of the automaton's calculation. The homomorphism gives a map from $A^\star$ to M, and the regular languages *recognized* by the automaton (or, we shall say, by the homomorphism or by the monoid) are the inverse images of subsets of M under this map. Each regular language has a particular monoid, its *syntactic monoid*, which appears in any automaton which recognizes the language. Algebraic measures of the "complexity" of the syntactic monoid thus give a measure of the relative complexity of regular languages — more complicated languages require more complicated monoids.

A program family over a finite monoid M is simply a more general way of converting an input string into a multiplication problem. An *instruction* is a pair consisting of the index to an input (a number in $\{1,\ldots,n\}$) and a map from the input alphabet A to M. A *program* of *length* ℓ is a sequence of ℓ instructions. If $a_1 \ldots a_n$ is an input string, each instruction *yields* the element of M obtained by applying its map to the input it indexes. A program yields the product, taken in order, of the yields of its instructions. A *program family* is a sequence of programs $\langle p_i \rangle$ where each p_i has indices in $\{1,\ldots,i\}$. Thus a program family defines a map from $A^\star$ to M, and just as before we say that a language is *recognized* by the program family iff it is the inverse image under this map of a subset of M.

One can ask which languages can be recognized by program families over which monoids, but for most monoids this is not particularly interesting because all languages can be recognized by some program (the notable exceptions are if the monoid is a nilpotent group [BST90] or a particularly small aperiodic monoid [Th89]). However, we can define complexity classes by bounding the length of the programs in the family as a function of the input size n. When we restrict this length to be polynomial, the classes which emerge in many cases are previously studied circuit complexity classes [BT88].

To begin with, the class of languages which can be recognized by a program family over *any* finite monoid is just (non-uniform) NC^1. This is easy to see as such a program family can be translated into a family of *branching programs* of constant width and polynomial size — the result then follows from [Ba89]. By the same construction, it is easy to see that NC^1 is also the class of languages recognized by a poly-size program family over a group, over the particular group S_5, or over any fixed monoid which contains a non-solvable group (see [Ba89]).

What of poly-length program families over an aperiodic monoid (one which has no non-trivial subset forming a group under the monoid operation)? This turns out to be exactly (non-uniform) AC^0 [BT88]. Furthermore, the natural hierarchy of subclasses of AC^0, given by poly-size circuits of depth k for each constant k, can be characterized in terms of program families over aperiodic monoids of *dot-depth* k. We understand non-uniform AC^0 [FSS84], so we understand this class. In particular, we can conclude that no program family over an aperiodic monoid can recognize the parity language.

The class ACC^0 can also be characterized as those languages recognizable by poly-length program families over certain finite monoids. These are the *solvable monoids*, those which do not contain a subset which forms a non-solvable group under the monoid operation [BT88]. There is a well-developed structure theory for finite monoids extending the structure theory for finite groups. Just as all groups can be constructed from simple groups by the Jordan-Hölder theorem, all monoids can be constructed from simple groups and certain basic aperiodic components by the Krohn-Rhodes theorem [KRT68, Ei76]. The solvable groups are those made up solely from cyclic groups in this system, and the solvable monoids are similarly those made up from cyclic groups and aperiodics. A non-solvable monoid is exactly one which contains a non-solvable subgroup, so that poly-length programs over it can recognize any language in NC^1 by [Ba89].

The conjectures $ACC^0 \neq NC^1$ and $ACC^0 \neq TC^0$ could thus be proved by showing that certain languages cannot be recognized by programs over solvable monoids. Partial steps toward this goal have been made by showing this for certain subclasses of the solvable monoids. We will consider below the lower bound results of Razborov

and Smolensky [Ra87, Sm87], which can be translated into this language of programs over monoids, and those of Barrington, Straubing, and Thérien which were obtained within it.

A related view of these complexity classes arises from the logical expressibility theory of Immerman, as extended by Barrington, Immerman and Straubing [Im83, Im87, BIS88]. Consider formulas of first-order logic, where variables range over places in the input and atomic formulas are $x < y$, $x = y$, and $\pi_a(x)$ for variables x and y and input letters a. The last means "the x'th input is an a" and is the only way for the formula to access the input. Formulas are built up from atomic formulas using boolean connectives and quantifiers. For example, the formula

$$\exists x \exists y \forall z (\pi_a(x) \wedge \pi_a(y) \wedge (z < y \vee x < z))$$

is true of an input string iff it has two consecutive a's and thus defines the language $A^\star aaA^\star$.

This primitive system defines exactly the star-free regular languages (those recognizable by aperiodic monoids [Sc65]). We can add to this system in a number of ways. First, we can add new atomic formulas for other operations, such as the predicate $BIT(x, y)$, meaning "the x'th bit of the binary number y is a one". This example gives the primitive system in Immerman's work, which can express those languages recognizable by alternating Turing machines in logarithmic time and constant alternation depth [BIS88]. We can add *arbitrary* predicates on tuples of variables, giving the class non-uniform AC^0 [Im83, GL84]. Note that this definition of AC^0 does not refer explicitly to a polynomial size bound — the bound follows from the constant number of variables which can be accessed by a predicate. By adding any other predicate we get the "first-order closure" of that predicate.

Secondly, we can add new operators to the logical system. Immerman [Im83] gives logical systems characterizing P, $LOGSPACE$, and $NLOGSPACE$ in this way. By introducing an operator to sytactically iterate formulas, he characterizes a wide variety of parallel complexity classes [Im87]. In [BIS88] classes up to and including NC^1 are characterized in terms of operators which are new quantifiers in the logical system. Logical definitions can be given for AC^0 (ordinary quantifiers), ACC^0 (modular counting quantifiers), TC^0 (threshold counting quantifiers), and NC^1 (group multiplication qunatifiers). All these characterizations have finite-state uniform (with the original atomic formulas), deterministic log-time uniform (with the BIT predicate), and non-uniform (with arbitrary added predicates) versions. More detailed definitions and proofs can be found in [BIS88].

4. The Basic Result: Parity

Here we outline the proof of Furst-Saxe-Sipser [FSS84] that the parity language is not in AC^0. The independent proof of this result by Ajtai [Aj83], in a very different conceptual framework, uses very similar methods. Though [FSS84] is usually phrased as an argument by contradiction, the proof can be thought of as demonstrating a property of all AC^0 languages not posessed by parity. Consider first the special case of AC^0 where the circuits are restricted to be of depth 2. It is well-known that a predicate on n boolean variables has a unique minimal CNF (conjunctive normal form) and DNF (disjunctive normal form) formula. A language is in depth-2 AC^0 iff for each n its restriction to input size n has either CNF size or DNF size within a bound polynomial in n. A parity language (either the strings with an odd number of ones, or those with an even number of ones) is easily seen to have CNF size and DNF size $O(2^n)$, so it is not in depth-2 AC^0.

We will prove that if L is any language in AC^0, then there is a language definable from L, called a *restriction*, of L which is in depth-2 AC^0. To define a restriction, we will first adopt some terminology. Let L^n be $L \cup A^n$ (Here $A = \{0,1\}$ is the input set. Let C^n be the circuit with n inputs recognizing L^n. A *restriction* of L^n is an assignment of the values $\{0,1,\star\}$ to the n input variables. (A restriction of a language is then a set of restrictions, one for each input length.) If a restriction ϕ^n has m stars, it naturally defines a subset $\phi^n L^n$ of A^m — those inputs which, when substituted for the stars in ϕ^n, yield an element of L^n. One can easily construct a circuit $\phi^n C^n$ which recognizes $\phi^n L^n$ by substituting constant gates for the unstarred input gates of C^n. The size and depth of this circuit are no greater than those of C^n, and they may be smaller because the restriction may ***kill*** gates. That is, it may place a one as input to an OR gate or a zero as an input to an AND gate, so that the gate may itself be replaced by a constant and the rest of its inputs discarded.

If d is the depth of the circuit family, for each C^n, we will construct a series of restrictions $\phi_{d-1}, \ldots, \phi_2$ such that the circuit $\phi_2 \ldots \phi_{d-1} C^n$ is equivalent to a circuit of depth 2 (whose size is still polynomial, though it may be larger than that of C^n). Thus L^n must have a composite restriction ϕ^n so that $\phi^n L^n$ has polynomially bounded CNF or DNF size. We must keep track on the number of stars in ϕ^n — it will be at least n^ϵ for some positive ϵ depending on d. Since any restriction of a level L^n of a parity language still has CNF and DNF size exponential in its inputs (it is itself a level of a parity language with a smaller set of inputs), a parity language cannot be in AC^0.

So for each i we need a restriction ϕ_i which will take a circuit C of depth $i+1$ to one which is equivalent to a circuit of depth i. To do this we will look at the effect

of ϕ_i of the subcircuits of C of depth 2 (ANDs of ORs of literals, or ORs of ANDs of literals). Some of these will be killed, and others will be left dependent on only a small number of their variables. We will show that a *random* restriction drawn from a certain distribution, with high probability, has enough stars and leaves *all* of the depth-2 subcircuits dependent on a constant number of variables. For such a restriction, the resulting circuit is equivalent to one of depth i, because the subcircuits of depth 2 can be rewritten so that the gates on the second level are of the same type (AND or OR) as the gates on the third level, and so can be merged with them. Rewriting the subcircuits may increase the size of the circuit by a constant factor.

If a random restriction from a particular distribution has the desired property, then at least one restriction with that property must exist. This is an example of the technique of *probabilistic construction*, also used in the next section. The construction may be thought of as a disguised counting argument, where all the restrictions which do not have the desired property are counted and shown not to be all of the possible restrictions.

The restriction ϕ_i will be the composition of two restrictions ρ and σ. The variables of ρ are assigned stars with probability $n^{1/2}$, zeros and ones each with probability $(1-n^{1/2})/2$. The assignments are made independently, so that the number m of stars in ρ is a random variable, with expected value $n^{1/2}$. The second restriction σ is on m variables, and is chosen from a similar distribution where the probability of a star for each variable is $m^{1/2}$. The theorem will follow from the following technical lemma:

Lemma: Let C be of depth $i+1$ and size n^k and let ρ and σ be chosen randomly as above. Then with probability $1-o(1)$ (i.e., approaching zero as n goes to infinity), the following are all true:

- (a) The circuit ρC has at least $n^{1/2}/2$ stars.
- (b) Each depth 1 subcircuit of C is either killed by ρ or becomes a depth - subcircuit of size at most $8k$ in ρC.
- (c) The circuit $\sigma\rho C$ has at least $m^{1/2}/2$ (thus at least $n^{1/4}/4$) stars.
- (d) Each depth 2 subcircuit of $\sigma\rho C$ depends on at most 2^{40k^2} variables.

Proof Outlines: Claims (a) and (c) follow from elementary probability. For example, the standard deviation in the number of stars in ρC is $O(n^{1/4})$, and a binomial random variable is more than a few standard deviations from its mean only with very low probability. Claim (b) is proved by cases. If a depth 1 subcircuit has size greater than $8k \log n$, it is killed with probability at least $1 - n^{-2k}$, and if it has

size less than $8k \log n$, it gets fewer than $8k$ stars with probability at least $1 - n^{-2k}$. There are at most n^k such circuits, so the total probability of any of them going wrong is less than n^{-k}. Claim (d) is proved in much the same way as Claim (b), but is a bit more complicated. One proves by induction on c that if the depth 1 subcircuits are of size at most c, then the depth 2 subcircuits depend on at most 2^{5kc} variables. Details may be found in [FSS84].

Applying ϕ_i thus changes the size of the circuit from n^k to at most $n^k 2^{2^{40k^2}}$ and the number of variables from n to at least $n^{1/4}/4$. The final restriction ϕ changes the original depth d size n^k circuit to one with depth 2 and $m = O(n^{-4^d})$ variables. The size has been increased by d multiplicative constant factors depending on k, but more importantly the original size n^k is now m^{k4^d}, larger but still polynomial. This argument can be fine-tuned to get stronger bounds on constant-depth circuits for parity, but these are overshadowed by the Yao-Hastad result described below.

An argument similar to this one was used by Sipser [Si83] to separate the levels of the depth hierarchy within AC^0. That is, there are languages with depth d, poly-size circuit families and no depth $d-1$ polysize circuit families. This circuit result, together with the correspondence between circuit classes and automata [BT88], gives a new proof that star-free languages exist with arbitrary dot-depth, a fact first proved by Brzozowski and Knast [BK78].

It is natural to consider the size of the optimal constant-depth circuits for a parity language. It is easy to construct circuits of depth k and size $O(2^{n/(k-1)})$ by making a full balanced depth $(k-1)$ tree of parity gates and then implementing these parity gates with CNF and DNF circuits. Yao [Ya85] first showed that this construction was essentially optimal, by proving an $\Omega(2^{n^{(1/4k)}})$ lower bound for depth k parity circuits. His proof method was greatly improved and simplified by Hastad [Ha86], who showed a lower bound of $2^{f(k)n^{1/(k-1)}}$ on this size.

Hastad noticed that the same random restrictions applied in [FSS84] actually do much more damage to the circuits than was seen originally. When applied to an AND of ORs, where each OR is of size at most t, a random restriction from the distribution of [FSS84] (stars with probability p, non-stars zero or one with equal probability) very likely produces a function which can be written as an OR of small ANDs. The chance that ANDs of size s or more are needed is bounded above by $(5pt)^s$. This allows the [FSS84] argument to be carried out with a much smaller size blowup at each stage when the restricted circuit is reversed using DeMorgan's laws. The complete argument can be found in [Ha86,Ha87]. We will prove similar bounds in the next section as a special case of a theorem of Smolensky [Sm87].

Similarly, there are functions computable by poly-size circuits of depth k that require exponential size circuits of depth $k-1$ (that is, size $2^{\Omega(n)}$ for a function of

n variables. This important result was claimed by Yao [Ya85] and the first proof was published by Hastad [Ha87]. This has the consequence (shown in [Ha87]) that an oracle can be constructed relative to which all the levels of the polynomial-time hierarchy are distinct, finally achieving the original goal of the Furst-Saxe-Sipser work. (For more on oracles and relativized complexity, see the book by Balcazar, Diaz and Gabarro [BDG88], and its forthcoming sequel).

The method of restrictions does not appear to extend to circuits with more general kinds of gates. For example, if a restriction is applied to an exclusive OR (i.e., a MOD 2 gate), a smaller gate of the same type results. A gate is not killed as in the AND or OR cases. The same thing happens with majority gates, except that they could be killed leaving nearly half their inputs starred if all the constant values were the same. We will see in the next section that alternate methods have greater success.

The parity result is analogous to a fundamental theorem of algebraic automata theory due to Schützenberger [Sc65]. The analog of the class AC^0 within the regular languages is the class of star-free languages — those languages which can be constructed from the single-letter and empty languages using Boolean operations (which correspond to gates of bounded fan-in) and concatenation (which corresponds to an unbounded fan-in OR). Schützenberger proved that the parity language is not so constructible. The result proved here can be used, with a little more work, to characterize the regular languages in AC^0 [CFL83, BCST88]. It would be interesting to have a proof of this automata-theoretic result which did not use the combinatorial methods of the existing proofs.

5. Aperiodics and One Prime

We now move to circuits with three types of gates — AND, OR, and MOD p for a fixed prime p. A mod p gate adds up its boolean inputs and outputs a one iff the result is nonzero. One could define more general MOD p gates, which output an arbitrary boolean function of the MOD p sum of the inputs, but these can be easily simulated by the simpler gates. Circuits of constant depth and polynomial size with these gates are equivalent to programs of polynomial size over p-monoids, i.e., monoids which contain only groups whose order is a power of p. We call this complexity class $AC^0[p]$.

To show that a particular language was not in $AC^0[2]$, Razborov [Ra87] introduced an algebraic structure on the functions from $\{0,1\}^n$ to $\{0,1\}$. Such a function can be written uniquely as a polynomial in n variables over the finite field Z_2, where the variables $x_1, \ldots, x_n$ each satisfy the identity $x_i^2 = x_i$. As generalized by Smolensky

[Sm87] and Barrington [Ba86], the functions from $\{0,1\}^n$ to any finite field F can be represented as polynomials in these same n variables. In this ring of functions, addition mod p is easily represented. If $f_1, \ldots, f_r$ are the inputs to a MOD p gate, the output is (Σf_i) if $F = Z_2$ or $(\Sigma f_i)^{|F|-1}$ in general.

Polynomials of this kind are parametrized by their *degree*, the maximum number of variables occurring in any monomial. It is not true that every function in $AC^0[p]$ has small degree, but (as first shown by Razborov in the case $p = 2$) every such function can be approximated by one of small degree in the following sense. We say that a function f *approximates* a function g with *error* e if $f(x) = g(x)$ for all but e of the 2^n input settings x in $\{0,1\}^n$.

First of all, AND gates may be replaced by OR gates because negation is a special case of modular addition — the complement of f is just $1 - f$. The inputs to a circuit, which are variables and negated variables, have degree 1. If the inputs to a MOD p gate have maximum degree d, the output has degree at most $d(|F| - 1)$, because the output is the $(|F| - 1)$ power of the sum of the inputs, which has degree d. Similarly, if each input is approximated by a function of degree at most d with error at most e, the output is approximated by a function of degree at most $d(|F| - 1)$ with error at most se, where s is the number of inputs in the gate.

The key step is the approximation of an OR gate of arbitrary size by a structure which will multiply the degree by only a fixed parameter ℓ. Given r boolean inputs $\{x_1, \ldots, x_r\}$, we will choose ℓ independent random linear combinations c_i over F. That is, we choose $\ell \cdot c$ independent random elements $b_{i,j}$ of F and let $c_i = (\Sigma_{j=1}^{r} b_{i,j} x_j)^{|F|-1}$. Our approximation is to be the OR of the predicates "$c_i = 0$". This is one minus the product for all i of $1 - c_i^{|F|-1}$, which has degree $d \cdot (|F| - 1) \cdot \ell$ if the inputs have degree d. Why does this product approximate the OR of the x_j? If all the x_j are zero, then all of the c_i are zero and the approximation is correct. If any of the x_j is nonzero, each c_i is nonzero with high probability, because the sum has an equal probability of being any element of F. The approximation is wrong only if all of the c_i come out zero, which happens with probability at most $|F|^{-\ell}$ — there are at most $2^n |F|^{-\ell}$ settings on which the approximating gate and the real gate do not give the same answer.

Now suppose we approximate each OR gate in this way in an original circuit of depth d and size s. Each gate may introduce a wrong answer on an $|F|^{-\ell}$ fraction of the input settings, so the total number of wrong input settings is at most $2^n s |F|^{-\ell}$. The degree of the approximating circuit is at most $(\ell(|F| - 1))^d$. If ℓ is chosen to be $n^{1/2d}$, say, we have the following:

Lemma: [Ra87] A language in $AC^0[p]$ can be approximated by a polynomial of degree $O(\sqrt{n})$ with at most $2^{n(1-\epsilon)}$ errors, for ϵ a constant depending on the circuit

depth.

Razborov continued (in the $p = 2$ case) by showing that a certain symmetric function (a function depending only on the number of ones in the input) could not be approximated in this way. Since any symmetric function is AC^0 reducible to the majority function, this proved that majority is not in $AC^0[2]$. An easy extension of this argument [Ba86] shows that it is not in $AC^0[p]$ either. These results, however, are superseded by those of Smolensky [Sm87], which we now outline.

As an example, we will let p be odd, and we'll show that the mod 2 function is not in $AC^0[p]$. Define $y_i = (1 - 2x_i)$, and note that the degree of a function as a polynomial in the x_i and in the y_i is the same. Let P be the product of the y_i for $i = 1, n$. Suppose P (or more precisely, $(1-P)/2$, which is a boolean function) were in $AC^0[p]$. By the Lemma above it (and therefore P) would be approximable by a degree $O(\sqrt{n})$ polynomial with some set E of errors, with $|E| \leq 2^{n(1-\epsilon)}$. Now consider the set of functions approximable by polynomials of degree at most $n+O(\sqrt{n})$, with error set contained in E. This set includes all functions! It is a Z_p-vector space, and contains all monomials. Monomials of degree at most $n/2$ in the y_i are approximable with no error by themselves. Monomials of greater degree are the product of P and monomials of lesser degree, and so are approximated by the product of the approximation for P and this monomial.

But this gives a contradiction. There simply aren't enough polynomials of low degree to make all those approximations. The monomials of that degree are being expected to span a Z_p-vector space of dimension at least $2^n(1 - 2^{-n\epsilon})$, but there are only $\Sigma_{i=0}^{n+O(\sqrt{(n)})} \binom{n}{i} = 2^n(1 - \Omega(1))$ of these. So the function P can't be in $AC^0[p]$. But of course P is doing multiplication in $\{1, -1\}$, which is isomorphic to addition in Z_2 — in fact P is just $1 - 2(\bigoplus_{i=1}^{n} x_i)$. This means that parity (and majority, to which parity is AC^0 reducible) cannot be in $AC^0[p]$.

The same argument works to show that mod$q$ is not in $AC^0[p]$ if q is prime to p. We have to find a finite field of characteristic p which contains an element of multiplicative order q. This is possible because the multiplicative group $F^\star$ of F is cyclic of order $|F| - 1$, and q always divides $p^{q-1} - 1$. By keeping more careful track of the numbers, the size lower bounds on circuits of OR and mod p gates computing mod q can be made exponential and nearly optimal, as in the improved versions of the parity theorem.

It is natural to consider extending this method to circuits including modular counting gates where the modulus m is not prime. The set of functions from $\{0,1\}^\star$ to Z_m becomes a module over the ring Z_m instead of a vector space over a field. The represetnation of this module as polynomials over the variables x_i or y_i is exactly as before.

The crucial change is that it is no longer possible to convert a Z_m-valued function into a $\{0,1\}$-valued function by powering, as we did in the proof of Razborov's lemma. It is still true that majority cannot be approximated by polynomials of small degree (see, e.g., [Ba86]), but it is no longer clear that circuits and polynomials correspond in anything like the same way. In fact, a polynomial over Z_6, for example, can be thought of as two independent polynomials, one over Z_2 and one over Z_3. There seems no reason to believe that a circuit of AND, OR, and MOD 6 gates need be separable into independent components in the same way. It appears that a new technique will be needed to deal with these circuits, one which might suffice to bound the power of the whole class ACC^0.

6. Programs Over Groups

It is natural to simplify the task of bounding circuits of AND, OR, and modular gates by considering the modular gates in isolation. Just as AND and OR gates cannot simulate modular counters in constant depth and polynomial size, we would expect that modular counters cannot simulate AND and OR within these constraints. This conjecture remains unproven, however, and we now consider what is known about subcases of the problem, largely from the work of Barrington, Straubing and Thérien [BST90]. Perhaps surprisingly, lower bounds in this area have been quite difficult and have involved some interesting proof techniques. It is reasonable to hope that a solution to our conjecture might tell enough about circuit families in ACC^0 to extend our understanding to this class.

There are a number of ways to define the class CC^0 [MT89] of constant depth, poly-size "circuits with only modular gates". Here we choose a model in which constant fan-in boolean gates are allowed along with theunbounded fan-in MOD q gates. This will allow more natural desriptions of some of our special cases.

In the terminology of Barrington and Thérien [BT88], restricting to only modular gates means considering programs over finite monoids which happen to be groups. In fact, since polynomial-length programs over non-solvable groups have all the computing power of NC^1, we consider only programs over solvable groups. By taking programs of polynomial length over natural subclasses of the solvable groups, we get natural subclasses of CC^0 and thus natural subcases of our problem.

Why do programs over solvable groups give us exactly CC^0? Barrington [Ba89] gives a simulation of programs over solvable groups by circuits of AND, OR, and MOD gates, and it is easy to see that this simulation yields a circuit family in CC^0. In the other direction, the simulation is very similar to that of Barrington and Thérien

[BT88] in the case of programs over solvable monoids. Just as the solvable monoids can be characterized as all those monoids which can be constructed from both modular and threshold counting [Th81], the solvable groups are those which can be constructed from modular counting alone.

In fact the operation of combining counters by connecting them in circuits of constant depth and polynomial size corresponds to a particular algebraic operation on the monoids associated to the counters — the *wreath product*. In the special case where G and H are permutations of sets X and Y respectively, the wreath product $G \circ H$ is a group of permutations of $X \times Y$, consisting of all pairs (f, h) where $h \in H$ and f is a function from Y to G. The action of this pair on $X \times Y$ takes an element (x, y) to $(f(y)(x), h(y))$. Thus the product of (f, h) and (f', h') is given by (g, hh') where $g(y) = f(y)f'(h(y))$. The wreath product is associative on permutation groups.

The solvable groups can be described as either all wreath products of cyclic groups or all wreath products of nilpotent groups (once we also include all subgroups and quotients of these products). This gives two natural parametrizations of the solvable groups — how many cyclics, or how many nilpotents, must be wreathed together to construct a given group? These parametrizations, as we shall see, correspond roughly to the depth of the circuits in the family and to a sort of "alternation depth".

We can begin our study of subclasses with the case of programs over abelian groups. Here the instructions commute with each other, so that the only important thing is the number of instructions of each type. Furthermore, as the abelian group is the direct product of cyclic groups, we can consider each of the factor groups independently. We can represent the program, then, by a number of single modular counting gates each with a linear number of inputs. It is easy to show that for sufficiently large n, such a circuit cannot compute the AND of n variables. As circuits, these programs can be represented as constant fan-in boolean combinations of single MOD q gates.

The next case is that of p-groups (groups whose order is a power of a prime p). Polynomial length programs over p-groups correspond to constant-depth poly-size circuits of MOD p gates (and constant fan-in boolean gates), and also to Razborov-Smolensky polynomials of constant degree over Z_p. In fact, programs of superpolynomial length give no additional computing power, as each is equivalent to one of polynomial length. Since the Razborov-Smolensky polynomial of a function is unique, it is clear that the AND function (which has degree n) does not have one of constant degree.

A nilpotent group is a direct product of p-groups where p may be different for each factor. A program over a nilpotent group may be thought of, therefore, as independent programs over several p-groups. Which CC^0 circuit families correspond

to polynomial length programs over nilpotent groups? The answer turns out to be those in which the modular gates all have prime moduli, and in which whenever the output of one modular gate feeds into the input of another the moduli of the two are the same.

The independent programs over different p-groups may also be thought of as independent constant-degree polynomials, one over Z_{p_i} for each p_i. The set of inputs giving a particular output value of the program, then, may be characterized by a constant-degree polynomial over Z_m, where m is the product of all the p's. With more care and the use of characterizations of nilpotent groups in terms of subword counting [Th83], one may show that programs over nilpotent groups of class d and exponent m have exactly the same power as polynomials of degree d over Z_m^k for some k (the k-fold direct product).

The AND function has a unique representation over the Razborov-Smolensky ring $Z_m^k[x_1, \ldots, x_n]$ as the monomial $x_1 \ldots x_n$, so that it cannot be represented by a polynomial of constant degree. But a different polynomial could be used to calculate the AND function if it took one value on the input 1^n and different values on all other inputs. In the Razborov-Smolensky ring over a field, such a "weak representation" of a set can be converted into the characteristic function of the set at a cost only of multiplying the degree by a constant. But it is not immediate that this can be done over a ring such as Z_m^k which has zero-divisors. Thérien has conjectured that it can, but this remains unproven.

The proof that AND is not weakly represented by a constant-degree polynomial involves Ramsey's theorem. For sufficiently large n, there must be a subset of the variables which has a particular uniformity property with respect to its coefficients in the constant-degree polynomial, in the following sense. If 1 is substituted for the variables not in the set, then all the degree-1 monomials from this set have the same coefficient, all the degree-2 monomials have some other coefficient, and similarly through the degree-d monomials. If this set is large enough, changing the value of the variables in it has no effect on the value of the function (as long as all other variables are 1). So the polynomial cannot weakly represent the AND function, as there is another input x such that $p(x) = p(1^n)$.

Any program over a nilpotent group is equivalent to a program of polynomial size. As we shall see below, this is not true for all solvable groups. In fact, programs over any group which is not nilpotent can be constructed for any function from $\{0,1\}^n$ to $\{0,1\}$. For general solvable groups length constraints correspond to size constraints on appropriate families of constant-depth circuits of modular gates.

The smallest non-nilpotent group is S_3, the group of permutations of a three-element set. Programs over S_3 correspond to particularly simple circuits — each has

a single MOD 3 gate receiving the output of a number of MOD 2 gates of inputs. The number of MOD 2 gates is closely related to the length of the program [Ba85]. The functions from $\{0,1\}^\star$ to Z_3 computed by such circuits are easily characterized in terms of the Razborov-Smolensky ring for Z_3. There are 2^n possible MOD 2 gates, one for each subset of the n input variables. As polynomials, the functions calculated by these 2^n gates are linearly independent over Z_3 and form a basis of the Razborov-Smolensky ring. Therefore any function in the ring has a *unique* minimal representation by a circuit of this kind, and thus by a program over S_3.

It is easy to construct this unique circuit for the AND function, which has 2^n MOD 2 gates, and thus prove an $\Omega(2^n)$ lower bound on the size of an S_3 program explicitly computing the AND of the inputs. As before, even a weak representation of AND can be converted to a strong representation in the Razborov-Smolensky ring over a field, in this case essentially by squaring it. Squaring a polynomial roughly squares the size of the circuit and thus the length of the program, so we have an $\Omega(2^{n/2})$ lower bound for the length of an S_3 program weakly representing AND.

The lower bound for the group S_3 can be generalized to certain other groups which are solvable but not nilpotent, after we develop a bit of machinery. (This is taken from the paper by Barrington, Straubing, and Thérien [BST90].) Let F be a finite field of order at least 3, and let $F^\star$ be its multiplicative group of nonzero elements, well-known to be a cyclic group. Let k denote the order of $F^\star$, so that $k \leq 2$. Let us fix a generator g of the group $F^\star$, and use it to define a logarithm function, i.e., $\log h$ is defined to be the unique m such that $0 \leq m < k$ and $g^m = h$.

We will be primarily concerned with the F-vector space $\mathcal{A}^n$ of functions from $(F^\star)^n$ to F, with two particular bases and vector multiplication operations. It may be useful to keep in mind the case $F = Z_3$, where A_n can be identified with the Razborov-Smolensky ring. The first basis is the set of functions $\delta_{\mathbf{w}}$ for each $\mathbf{w} \in (F^\star)^n$, defined by $\delta_{\mathbf{w}}(\mathbf{x}) = 1$ for $\mathbf{w} = \mathbf{x}$ and zero otherwise. Pointwise multiplication of functions has a particularly simple representation with respect to this basis. We define the *support* of a function to be the number of distinct basis elements needed to represent it, or equivalently the number of inputs on which it takes a nonzero value.

Our other basis is the set of functions $P_{\mathbf{w}}$ for each $\mathbf{w} = (w_1, \ldots, w_n)$ in $(F^\star)^n$, defined by

$$P_{\mathbf{w}}(\mathbf{x}) = P_{\mathbf{w}}(x_1, \ldots, x_n) = w_1^{\log x_1} \ldots w_n^{\log x_n}.$$

These will generalize the monomials in the variables y_i from the earlier Smolensky argument. It is not too hard to see [BST90] that these vectors form a basis for $\mathcal{A}^n$. We define the *weight* of a function to be the number of distinct elements of this basis needed to represent it. Pointwise multiplication of functions is not particularly simple over this basis, but another vector product is. Define the *convolution* of two functions

f_1 and f_2 to be the function $f_1 * f_2$ in $\mathcal{A}^n$ defined by:

$$(f_1 * f_2)(\mathbf{x} = \sum_{\mathbf{w} \in (F^*)^n} f_1(\mathbf{w}) \cdot f_2(\mathbf{w}^{-1}\mathbf{x}).$$

The two bases, and these two vector multiplications, are related by an operation called the *Fourier transform*. If f is a function in $\mathcal{A}^n$, $\mathbf{T}f$ is the function in $\mathcal{A}^n$ given by:

$$\mathbf{T}f(\mathbf{w}) = \sum_{\mathbf{x} \in (F^*)^n} f(\mathbf{x}) \cdot P_{\mathbf{w}^{-1}}(\mathbf{x}).$$

It is easy to show that the weight of f is the support of $\mathbf{T}f$, and vice versa. Furthermore, $\mathbf{T}(f_1 \cdot f_2) = \mathbf{T}f_1 * \mathbf{T}f_2$ and $\mathbf{T}(f_1 * f_2) = \mathbf{T}f_1 \cdot \mathbf{T}f_2$, where the dot denotes pointwise multiplication. These facts can be used [BST90] to prove an interesting tradeoff between the weight and support of a function. Just as a function $\delta_{\mathbf{w}}$ has weight 1 and support k^n, and a function $P_{\mathbf{w}}$ has weight k^n and support 1, any function's weight and support multiply to give at least k^n.

The lower bound method for programs over certain groups proceeds as follows. We first need to represent boolean functions in $\mathcal{A}^n$. To do this we identify $\{0,1\}^n$ with $\{1,g\}^n \subseteq (F^\star)^n$, and view a function in $\mathcal{A}^n$ by restriction as being from $\{0,1\}^n$ to F. Thus for each $P_{\mathbf{w}}$ we have a function $Q_{\mathbf{w}}$ from $\{0,1\}^n$ to F with $P_{\mathbf{w}}(x_1, \ldots, x_n) = Q_{\mathbf{w}}(\log x_1, \ldots, \log x_n)$ as long as $\mathbf{x} \in \{1,g\}^n$. It can be shown [BST90] that the AND function, viewed as being from $\{0,1\}^n$ to F because $\{0,1\} \subseteq F$, is not the sum of fewer than $(\frac{k}{k-1})^n$ of the $Q_{\mathbf{w}}$. That is, any function in $\mathcal{A}^n$ having the correct values on $\{1,g\}^n$ has weight at least $(\frac{k}{k-1})^n$.

Then it remains to show that a program of length ℓ over one of the specified groups can be represented by a function of weight ℓ^K for some constant K depending on the group [BST90]. This can be done for any group which divides the wreath product of a p-group and an abelian group. The exponential lower bound on program length corresponds to an exponential lower bound on the size of circuits of a certain type computing the AND function. These would be unbounded fan-in circuits of constant depth with modular-counting gates (and binary AND and OR gates) satisfying the following restriction: if q is not a power of p, all MOD-q gates have only inputs which are inputs to the circuit.

An unproven but plausible hypothesis about $\mathcal{A}^n$ would allow us to extend this exponential lower bound to programs over more groups — those which are wreath products of p-groups with nilpotent groups. This would give an exponential lower bound on the size of circuits of unbounded fan-in, constant depth, modular gates for prime moduli, and a weaker restriction: the input to a MOD-q gate, for $q \neq p$, must be an input to the circuit or the output of another MOD-q gate.

The hypothesis is a generalization of the fact that a sum of $Q_{\mathbf{w}}$ functions computing the AND function must have exponentially many terms. A particular $Q_{\mathbf{w}}$ function can be thought of as the generator g raised to a linear function of the input variables, i.e.,

$$Q_{\mathbf{w}}(x_1, \ldots, x_n) = w_1^{x_1} \ldots w_n^{x_n} = g^{x_1(\log w_1)+\ldots+x_n(\log w_n)}.$$

Here the linear function in the exponent has its "addition" in the multiplicative group $F^\star$, which is isomorphic to Z_k. To generalize this, we allow polynomials of constant degree in the exponent instead of just linear ones. For example, there are $k^{1+n+\binom{n}{2}}$ quadratic polynomials $p(\mathbf{x})$ in $\{x_1, \ldots, x_n\}$ over Z_k, and to each one there corresponds a function $g^{p(\mathbf{x})}$ in $\mathcal{A}^n$. We define the 2-*weight* of a function f to be the minimum number of these functions which must be summed (in $\mathcal{A}^n$) to get f. Similarly, we can define the r-*weight* for any integer r (1-weight, for example, is what we have already called "weight"). The *constant-degree hypothesis* is that the AND function (reinterpreted in $\mathcal{A}^n$ as before) has exponential r-weight for any constant r. The intuition behind this conjecture is that functions of low r-weight are similar to those of low 1-weight (though their 1-weight can be very high), and should not provide much help toward forming the AND function.

Assuming this hypothesis, the proof [BST90] of the lower bound for wreath products of a p-group with an abelian group extends fairly directly [BST90] to the case of the wreath product of a p-group and a nilpotent group, and thus gives the circuit size lower bound described above. Similar methods might conceivably show that programs over solvable groups (and hence circuits of constant depth with only modular gates) require exponential length (size) to do AND. This would appear to require modifying the techniques to work with Razborov-Smolensky polynomials over general finite rings as well as just fields. Some preliminary work along these lines has been reported by Barrington [Ba90].

7. References

[Aj83] M. Ajtai, "Σ_1^1 formulae on finite structures", *Annals of Pure and Applied Logic* **24** (1983), 1-48.

[BDG88] J. L. Balcázar, J. Diaz, and J. Gabarró, *Structural Complexity I*, EATCS Monographs on Theoretical Computer Science **11** (Berlin: Springer-Verlag, 1988).

[Ba85] D. A. Barrington, "Width 3 permutation branching programs", Technical Memorandum TM-291 (Dec. 1985), M.I.T. Laboratory for Computer Science.

[Ba86] D. A. Barrington, "A note of a theorem of Razborov", COINS Technical Report 87-93 (July 1986), University of Massachusetts.

[Ba89] D. A. Barrington, "Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1", *J. Comp. Syst. Sci.* **38:1** (1989), 150-164.

[Ba90] D. A. M. Barrington, "Some problems involving Razborov-Smolensky polynomials", COINS Technical Report 90-59, University of Massachusetts.

[BCST88] D. A. M. Barrington, K. Compton, H. Straubing, and D. Thérien, "Regular languages in NC^1", *J. Comp. Syst. Sci.*, to appear. Also Technical Report BCCS-88-02, Boston College.

[BIS88] D. A. M. Barrington, N. Immerman, and H. Straubing, "On uniformity within NC^1," *J. Comp. Syst. Sci.*, in press. Preliminary version *Structure in Complexity Theory: Third Annual Conference* (Washington: IEEE Computer Society Press, 1988), 47-59.

[BST90] D. A. M. Barrington, H. Straubing, and D. Thérien, "Non-uniform automata over groups", *Information and Computation*, to appear. Also COINS Technical Report 89-56, University of Massachusetts.

[BT88] D. A. M. Barrington and D. Thérien, "Finite monoids and the fine structure of NC^1", *J. ACM* **35:4** (Oct. 1988), 941-952.

[Bu87] S. R. Buss, "The Boolean formula value problem is in ALOGTIME," *19th ACM STOC Symp.* (1987), 123-131.

[CFL83] A. K. Chandra, S. Fortune, and R. J. Lipton, "Unbounded fan-in circuits and associative functions", *Proc. 15th ACM STOC* (1983), 52-60.

[CSV84] A. K. Chandra, L. J. Stockmeyer and U. Vishkin, "Constant depth reducibility," *SIAM J. of Comp.* **13:2** (1984), 423-439.

[Co85] S. A. Cook, "A taxonomy of problems with fast parallel algorithms," *Information and Control* **64** (1985), 2-22.

[Ei76] S. Eilenberg, *Automata, Languages, and Machines*, Vol. B (New York: Academic Press, 1976).

[FSS84] M. Furst, J. B. Saxe, and M. Sipser, "Parity, circuits, and the polynomial-time hierarchy", *Math. Syst. Theory* **17** (1984), 13-27.

[Ha86] J. Håstad, "Almost optimal lower bounds for small depth circuits", *Proc. 18th ACM STOC* (1986), 6-20.

[Ha87] J. Håstad, *Computational Limitations of Small Depth Circuits*, (Cambridge, USA: MIT Press, 1988).

[GL84] Y. Gurevich and H. L. Lewis, "A logic for constant-depth circuits", *Information and Control* **61**, 65-74.

[HMPST87] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán, "Threshold circuits of bounded depth", *28th IEEE FOCS Symp.* (1987), 99-110.

[Im83] N. Immerman, "Languages which capture complexity classes," *15th ACM STOC Symp.,* (1983) 347-354. Also appeared in revised form in *SIAM J. Comput.* **16:4**, (1987).

[Im87] N. Immerman, "Expressibility as a complexity measure: Results and directions," *Second Structure in Complexity Theory Conf.* (1987), 194-202.

[KRT68] K. B. Krohn, J. Rhodes, and B. Tilson, in M. A. Arbib, ed., *The Algebraic Theory of Machines, Languages, and Semigroups* (New York: Academic Press, 1968).

[MT89] P. McKenzie and D. Thérien, "Automata theory meets circuit complexity", *Proc. 16th ICALP (Springer Lecture Notes in Computer Science* **372** (1989), 589-602.

[MP71] R. McNaughton and S. Papert, *Counter-Free Automata* (Cambridge, Mass.: MIT Press, 1971).

[PS88] I. Parberry and G. Schnitger, "Parallel computation with threshold functions", *J. Comp. Syst. Sci.* **36:3** (1988), 278-302.

[Pi86] J. E. Pin, *Varieties of Formal Languages* (New York: Plenum Press, 1986).

[Ra87] A. A. Razborov, "Lower bounds for the size of circuits of bounded depth with basis $\{\&, \oplus\}$", *Mathematicheskie Zametki* **41:4** (April 1987), 598-607 (in Russian). English translation Math. Notes Acad. Sci. USSR **41:4** (Sept. 1987), 333-338.

[Ru81] W. L. Ruzzo, "On uniform circuit complexity," *J. Comp. Sys. Sci.,* **21:2** (1981), 365-383.

[Sc65] M. P. Schützenberger, "On finite monoids having only trivial subgroups", *Information and Control* **8** (1965), 190-194.

[Sm87] R. Smolensky, "Algebraic methods in the theory of lower bounds for Boolean circuit complexity", *19th ACM STOC Symp.* (1987), 77-82.

[St79] H. Straubing, "Families of recognizable sets corresponding to certain varieties of finite monoids", *J. Pure and Applied Algebra* **18** (1979), 319-327.

[STT88] H. Straubing, D. Thérien, and W. Thomas, "Regular languages defined with generalized quantifiers", *Proc. 15th ICALP* (1988), 561-575.

[Th81] D. Thérien, "Classification of finite monoids: the language approach", *Theoretical Computer Science* **14** (1981), 195-208.

[Th83] D. Thérien, "Subword counting and nilpotent groups", in L. J. Cummings, ed., *Combinatorics on Words: Progress and Perspectives* (New York: Academic Press, 1983), 297-305.

[Th89] D. Thérien, "Programs over aperiodic monoids", *Theoretical Computer Science* **64** (1989), 271-280.

[Ya85] A. C. C. Yao, "Separating the polynomial-time hierarchy by oracles", *Proc. 26th IEEE FOCS* (1985), 1-10.