Quasipolynomial Size Circuit Classes

David A. Mix Barrington
Department of Computer Science
University of Massachusetts
Amherst, MA 01003, U.S.A.

Abstract

Circuit complexity theory has tried to understand which problems can be solved by "small" circuits of constant depth. Normally "small" has meant "polynomial in the input size", but a number of recent results have dealt with circuits of size $2^{\log^{O(1)} n}$, *or* quasipolynomial *size. Here we (1) summarize the reasons for thinking about the new complexity classes so introduced, (2) survey these results and give an overview of these classes, and (3) show that the Barrington-Immerman-Straubing uniformity definition for polynomial-size classes can easily be extended to quasipolynomial size as well, with most of the key results remaining true in the uniform setting.*

1 Introduction

What determines the computational power of a network of computing elements? We need to know what the individual elements can do, the purely combinatorial properties of the network (such as its size and depth), and something about the way the network is specified (how "uniform" it is). Suppose we fix the combinatorial parameters, so that our networks are unbounded fan-in boolean circuits whose size is polynomial in the input length and whose depth is constant, and vary the other two factors. For every choice of base functions and every uniformity condition, we get a complexity class — the set of problems which can be solved by constant-depth poly-size circuits whose gates compute those functions and which meet the uniformity condition.

Over the past decade we have developed a fairly complete picture of these complexity classes for a wide variety of simple gate functions. If gates can only compute the AND and OR functions, we get the class AC^0 [FSS84]. If gates can also test the sum of their inputs for divisibility by a constant q, we get the class $ACC^0[q]$. The union over all q of $ACC^0[q]$ is called ACC^0 or ACC [Ba89, MT89]. If the gates can compute the majority function, we get the class TC^0 [PS88, HMPST87], and if they can compute membership in a fixed arbitrary regular language, we get the class NC^1. We know a number of containment relations among such classes (e.g., $AC^0 \subseteq ACC^0 \subseteq TC^0 \subseteq NC^1$), but we have not gotten very far in separating those we think are different (e.g., we think $ACC^0 \neq TC^0$ but can't prove even $ACC^0[6] \neq NP$).

One reason for looking at these particular circuit classes is their robustness. It looks like they capture something inherent about computing with these particular operations, because so many naturally defined classes in other models of computation tend to coincide with one of them. For example, AC^0 is the alternating log-time hierarchy (defined in terms of random-access Turing machines) [Si83] and also the first-order definable languages in Immerman's framework [Im87]. NC^1 can be defined as a constant-depth circuit class above, or more conventionally [Co85] in terms of log-depth binary AND/OR circuits [Ba89] (or, for that matter, constant-width branching programs [Ba89], constant-width circuits [Ba89], or non-uniform automata [BT88]).

The relationship between circuit classes with gates for certain operations and first-order logic with quantifiers for those operations is quite general, and continues to hold when uniformity is taken into account [BIS90]. Natural "uniformity conditions" (which say how easy it must be to specify the circuits in a family) correspond to natural conditions on the atomic formulae of a first-order sentence. In particular, log-time uniform circuits of constant depth and polynomial size (with a particular set of gate types) can recognize exactly those languages specified by first order sentences with atomic formulae for equality, order, and binary representation (and quantifiers for functions computable by the gates).

Another reason to study constant-depth, poly-size circuit classes (the original reason, at least for the American literature following [FSS84]), is their connection with Turing machine classes above P (see, e.g., [BDG88] for an overview of these classes). Lower bounds on the size of circuits of a certain type (e.g., constant-depth AND/OR) to compute a certain function (e.g., exclusive OR) lead to the construction of an oracle separating certain complexity classes (in this example, showing $\oplus P^A$, and hence $PSPACE^A$, is not contained in PH^A [Ya85, Ha88]). But this very general connection does not use the "poly-size" part of the definition of our circuit classes. In fact, the theorem that the exclusive OR function is not in AC^0 [FSS84, Aj83] had no such consequences in structural complexity. By examining the construction [FSS84], it is easy to see why. If, for example, $\oplus P^A \subseteq PH^A$ for some oracle A, then a poly-time constant-alternations alternating oracle Turing

machine can determine whether the number of strings of length n in A is odd or even. We can view the ATM computation as a constant-depth circuit, whose size is exponential in the (polynomial) running time of the ATM, and whose inputs are the A-membership questions for the 2^n strings of length n. The size of this circuit is thus not polynomial in its number of inputs, but *quasipolynomial*: $2^{\log^{O(1)} N}$, with $N = 2^n$. The key step in constructing the desired oracle is knowing that no such constant-depth, quasipolynomial size circuit can actually compute the parity of its inputs.

This suggests an alternate series of circuit complexity classes. For each of our gate types, consider the languages recognizable by circuit families of constant depth and quasipolynomial size. We will denote each new class by appending a small q to the corresponding poly-size class, so that qAC^0, for example, is constant-depth quasipolynomial size AND/OR circuits. The natural containments go through unchanged, so we have, for example, $qAC^0 \subseteq qACC^0 \subseteq qTC^0 \subseteq qNC^1$ (with the multiple definitions of the last class still coinciding). Furthermore, the few known separations of the poly-size classes are known to hold with exponential lower bounds. So we know that qAC^0 is strictly contained in $qACC^0[2]$ [Ya85, Ha88], that the qAC^0 hierarchy is infinite [Ha88], that $qACC^0[2]$ is strictly contained in qTC^0 [Ra87], and that the classes $qACC^0[p]$ are incomparable for prime p [Sm87]. (The few known separations of subclasses of "pure ACC", circuits with only modular counting gates, also hold with exponential size bounds [BST90].)

Recently a series of important upper bound results has lent further significance to these new classes. First Allender [Al89, AH90] noticed that the argument of Toda [To91], showing $PH \subseteq P^{PP}$, can be adapted to show $qAC^0 \subseteq qTC^0_3$ (where the subscript denotes depth-3 threshold circuits). The analogous containment of AC^0 in TC^0_3 is not known to hold. This result has been steadily extended on both sides. On the left, Yao [Ya90] showed that $qACC^0$ is also contained in qTC^0_3, giving the first nonobvious upper bound on the computing power of even ACC^0. On the right, Beigel and Tarui [BT91] and Green, Köbler and Torán [GKT92], have given sharper characterizations of the type of qTC^0_3 circuits needed to simulate $qACC^0$. An important notion in this work is that of a *polynomial* over the integers in the n boolean input variables, with poly-log degree and quasipolynomial coefficients. A variety of natural complexity classes can be defined in terms of such polynomials — we give an overview of such classes in section 2.

A natural question about the quasipolynomial size classes is whether they are as robust as their polynomial counterparts. In section 3 we show that the Barrington-Immerman-Straubing uniformity definition still holds true for the new classes. In particular, if we augment the first-order logic to allow suitably restricted second-order operators, we get logically defined complexity classes which correspond to uniform versions of the quasipolynomial size circuit classes. The appropriate notion of uniformity is that queries about the circuits must be answerable in deterministic poly-log (rather than logarithmic) time. In section 4 we show that the simulations of Beigel and Tarui and of Green, Köbler, and Torán are uniform simulations in this view.

2 An Overview of the New Classes

We may begin with some simple observations about the quasipolynomial analogs of some larger complexity classes. If we look at unrestricted binary circuits of quasipolynomial size we get qP, an analog of polynomial size or P. This is easily seen to be equal to $DTIME(2^{\log^{O(1)} n})$ (quasipolynomial time). The analogue of NC, qNC, has binary circuits of quasipolynomial size and poly-log depth, and is easily seen to be equal to $DSPACE(\log^{O(1)} n)$ (poly-log space). I argue that this class qNC also forms the analog of NC^1 and of all classes in between, such as $DSPACE(\log n)$ (logspace). To see this, we can look at either of two characterizations of NC^1, poly-size boolean formulae or poly-size constant-width branching programs, and replace "polynomial" by "quasipolynomial". This class is closed under conversion from a circuit to a formula, and contains all the constant-depth quasipolynomial size classes which we will consider.

The appropriate notion of "completeness" to consider possible collapses of these classes is constant depth, quasipolynomial size Turing reduction. As pointed out in [AH90], if NC^1 is contained in, for example, qTC^0, then all of qNC collapses to qTC^0. This is because a problem complete for NC^1 under ordinary poly-size reductions (such as iterated multiplication in a non-solvable group) is complete for qNC under these new reductions. Similarly, the majority problem (on n inputs) is complete for qTC^0, so that $qTC^0 = qACC^0$ if majority is in $qACC^0$.

Let us say that a class within qNC is "large" if it contains (ordinary) $ACC^0[6]$. Since $ACC^0[6] = NP$ is consistent with everything we know, virtually any lower bound for a large class would be a major breakthrough. Conversely, let us call a class "small" if we know that it does not contain NP. By the standard exponential lower bounds mentioned above, we know that qAC^0 and $qACC^0[p]$ for prime p are small. Turning to threshold circuits, qTC^0_2 is small [HMPST87], as is the subclass of qTC^0_3 where the bottom threshold gates are ANDs of at most $c \log n$ inputs, $c < 1/2$ being a constant [HG90].

The class qTC^0 is obviously large, but because of the sequence of upper bounds described above we know that various natural subclasses of it are also large. To define some of these classes, it is convenient to consider circuits where the bottom level consists of AND gates, each connected to at most $log^{O(1)} n$ inputs. If qC is any quasipolynomial size circuit complexity class, let qC^+ denote the languages recognized by families of circuits, each a circuit from qC augmented with such a level of AND gates. This notation is due to Green, Köbler, and Torán [GKT92], inspired by a suggestion of Beigel, Reingold, and Spiel-

man [BRS91]. The motivation for it is as follows. If qC contains only symmetric functions (e.g., $qC = SYM$, the class of all symmetric functions on quasipolynomially many inputs), then we can view a qC^+ circuit as evaluating an integer polynomial of poly-log degree in the n boolean input variables, and applying some lookup function to the result. The coefficients of the polynomial must also be quasipolynomial (poly-log bits long), as the size of the resulting qC^+ circuit is the sum of all the coefficients. The advantage of viewing the circuit in this way is that a number of useful combinatorial operations on such circuits correspond to natural algebraic operations on the polynomials. Beigel and Tarui [BT91] call qC^+ the set of functions that can be "simulated" by functions in qC. The important thing to note is that this "simulation" relation is transitive — equivalently, $(qC^+)^+ = qC^+$.

Yao [Ya90] and Beigel and Tarui [BT91] used this framework to show that SYM^+, and hence $(qTC_2^0)^+$, is large. This was improved by Green, Köbler, and Torán [GKT92], who showed that without loss of generality the arbitrary symmetric function can be replaced by the single function *Midbit*, which adds up its boolean inputs as integers and returns the middle bit of the binary representation of the sum. Thus they show that $Midbit^+$ is large (specifically, $Midbit^+$ and thus all these other classes contain $qACC^0$). This is probably a significant improvement over the naive $qACC^0 \subseteq qTC^0$, and suggests trying to show some natural function to be outside of AC^0[6], say, by showing it to be outside of $Midbit^+$. A natural candidate function is the "majority of majorities": a depth-2 circuit of MAJORITY gates of fan-in $\sqrt{n}$. As observed in [BT91], if this function (and hence $(TC_2^0)^+$) is in SYM^+ then $SYM^+ = qTC^0$. (The analogous result holds for $Midbit^+$ by exactly the same simple argument.)

All three proofs [Ya90, BT91, GKT92] proceed in the same way. The $qACC^0$ circuit is first put in a normal form where all modular gates have prime modulus and all gates on a given level compute the same function. The AND gates are then converted into modular counting gates as in [Al89, AH90]: using the method of Valiant and Vazirani [VV86], quasipolynomially many MOD_p^+ circuits are constructed, each based on one choice of the value of poly-log many "random" bits, such that most of these gates give the correct answer for the AND gate. In fact, as described in [BT91], the same random bits can be used for all the AND gates in the circuit in a way that most choices of these bits give correct answers for *all* these AND gates. The simulation of the $qACC^0$ circuit thus at this point has a MAJORITY gate at the top, and a constant number of levels of MOD_p gates and MOD_p^+ subcircuits (of course, not all for the same prime). By using a simple distributive property of poly-log ANDs over MOD_p[AH90, BT91], the poly-log AND gates can all be pushed to the bottom to give a MAJORITY of $(qCC^0)^+$ circuits. (CC^0 or "pure ACC" [Ya90] is subclass of ACC^0 with only modular counting gates.)

It remains then to show that a symmetric gate or a *Midbit* gate can "swallow" a level of MOD_p gates, i.e., that a symmetric (*Midbit*) function of quasipolynomially many MOD_p gates is in SYM^+ ($Midbit^+$). This is done using the now-famous "Toda polynomials", the first version of which was introduced by Toda [To91]. In the improved version of Beigel and Tarui [BT91], the t'th Toda polynomial $P_t(x)$ is a univariate polynomial of degree $2t-1$ which has the property that $P_t(x)$ is congruent to 0 modulo x^t and congruent to 1 modulo $(x-1)^t$. As a consequence, if N is any integer and p is a prime, the residue of $P_t(N)$ modulo p^t is equal to that of N modulo p.

Suppose we must evaluate a symmetric function of s MOD_p gates. We (1) choose t so that p^t is greater than s, (2) find for each MOD_p gate a poly-log degree integer polynomial in the inputs whose residue modulo p is the value of the gate, (3) apply P_t to each of these polynomials (getting another poly-log degree polynomial as t is poly-log), (4) add the results (still poly-log degree), and (5) take the residue of this grand sum modulo p^t. The result is exactly the number of MOD_p gates which have value one, and determines the output of the original symmetric function. In the SYM^+ case we are now essentially done, as the composition of the MOD_{p^t} function and the original symmetric function is another symmetric function. For the $Midbit^+$ case, there is some additional work to do to show that we can use this grand sum to construct another polynomial whose middle bit gives the middle bit of the sum of the original MOD_p gates. We will deal with the solution of [GKT92] in more detail below.

We can put the recent results of Barrington, Beigel, and Rudich [BBR92] in this context as well. It is natural to ask for what classes C of symmetric functions the class C^+ can be shown to be small. MAJ^+ and MOD_p^+ for prime p are small, but what about MOD_6^+? (In general, the case of MOD_6^+ is general enough to extend to MOD_m^+ for non prime-power m.) If the MOD_6 function is defined (as usual) to return zero if the sum of its inputs is divisible by six and one otherwise, then MOD_6^+ is in fact small (by [BBR92], it does not contain $\neg MOD_6$ or MOD_5). But consider more "MOD_6 functions" – the class $RMOD_6$ of functions whose value depends only on the sum of their inputs modulo six (i.e., an arbitrary lookup function is applied to the residue of the sum). Whether $RMOD_6^+$ is small is an open problem in [BBR92] – they conjecture that it does not contain the OR function, which would clearly make it small.

3 A Robust Uniformity Definition

How is one to define a "uniform" circuit of quasipolynomial size? For polynomial size and constant depth, a robust definition was provided by Barrington, Immerman, and Straubing[BIS90]. Deterministic log-time uniformity, the most restrictive reasonable condition in the family first proposed by Ruzzo [Ru81], is shown there to coincide with an

apparently stronger notion arising from Immerman's work.

Informally, let us say that a circuit family is "logically uniform" if the circuit for each input size arises from the straightforward evaluation of a single logical formula. In the logical system, variables range over places in the input, atomic formulas allow access to the input and basic operations on the place numbers, and quantifiers perform the operations of the various gate types. As an example, consider the formula "$\exists x \forall y[\pi_0(x) \wedge x \leq y]$", which says that there is a place in the input before all other places and which contains a zero (i.e., the input is in the language $0\Sigma^*$). A logically uniform circuit for this formula would have an OR gate at the top with fan-in n (for the n possible values of x). Each child of this OR gate would be an AND gate, with fan-in n for the possible values of y. The n^2 inputs to these n AND gates would be constant-size circuits evaluating the quantifier-free subformula in brackets for each pair of values for x and y (they would each contain an input gate to evaluate $\pi_0(x)$, which means "input number x is a zero"). The central result of [BIS90] is that any log-time uniform circuit, with any reasonable set of gate types, has a logically uniform "normal form". That is, it is equivalent to a first-order formula with quantifiers corresponding to the gate types used. Our goal here is to show a similar normal form for uniform constant depth, quasipolynomial size circuits.

To begin we must generalize the Ruzzo-type uniformity definition to the new setting. Deterministic log-time is no longer "reasonable" to answer queries about the circuit, as the gate numbers of the circuit are now poly-log many bits long. But if we allow deterministic poly-log time, we get a sensible notion. The following parallels the definitions of [BIS90]:

Definition: The *direct connection language* of a circuit is the set of tuples $\langle g, t, h, y\rangle$ where g and h are gate numbers, t is the gate type of g, either h is a child of g or g is an input gate for input variable x_h, and y is an arbitrary string of n bits. (The purpose of y is to make the size of the query comparable to $n - y$ must be arbitrary as the query must be answered without looking at all of it.) A circuit is *DPLT-uniform* if its direct connection language is in $DTIME(\log^{O(1)} n)$.

The notion of a quantifier corresponding to a general boolean function is introduced in [BIS90]. Here we will restrict our gates to compute boolean functions of n^k boolean inputs for some k. An important technical requirement (necessary to allow us to assume that there are that many inputs, among other things) is that it is possible to pad the domain of the function with "identity elements" without affecting its output. For example, an AND operation may be padded with ones, an OR or XOR with zeros, and a MAJORITY with zero-one pairs. If f is such a function, Q_f is a quantifier which binds k variables (each ranging from 1 to n) such that the truth value of $(Q_f x_1 \ldots x_k)\phi(x_1, \ldots, x_k)$ is the value of f applied to the n^k bits given by letting the x_i's range from 1 to n. (In case f is not commutative, we must specify that f is applied to these bits in lexicographic order. There is no need for f to be commutative or even associative — Bédard, Lemieux, and McKenzie [BLM90] apply the framework of [BIS90] without difficulty to arbitrary binary algebras with identity.)

Theorem: [BIS90] Let $\mathcal{F}$ be a set of suitable gate functions including AND and OR. Then (1) a language is computable by a family of constant depth, polynomial size circuits, whose gates compute a finite set of functions from $\mathcal{F}$, and whose direct connection language is in $DTIME(\log n)$, iff (2) the language is the set of strings satisfying some particular first-order sentence with quantifiers for $\mathcal{F}$. The first-order sentence may have boolean connectives and atomic formulas of the form $x \leq y$, $x = y$, $BIT(x, y)$, $\pi_0(x)$, and $\pi_1(x)$, where x and y are variables representing places in the input.□

The atomic formula $BIT(x, y)$ requires some explanation. It means "the x'th bit in the binary expansion of y is one", and gives the logical system the same capacity to examine a number bit-by-bit as any Turing machine has. If the BIT predicate is removed, the logical system can only describe regular languages unless the quantifiers themselves are powerful enough to do more [BIS90]. Lindell [Li92] examines why this BIT predicate is needed and what other predicates might do as well.

How, then, to augment the logical system to represent larger circuits? We do this by adding a restricted form of second-order quantifier. Such quantifiers will bind second-order variables which range over *relations* on the set $\{1, \ldots, \log n\}$. For example, if R is a 3-ary variable and x, y, and z are ordinary variables, $R(x, y, z)$ is a valid boolean term of the language as long as $1 \leq x, y, z \leq \log n$ (equivalently, we could make $R(x, y, z)$ false if any variable exceeds $\log n$). If f is a boolean function of $2^{\log^k n}$ inputs, with the padding property mentioned above, then Q_f is a second-order quantifier such that $(Q_f R)\phi(R)$ has the truth value given by applying f to the $2^{\log^k n}$ bits obtained by evaluating $\phi(R)$ for all possible R. (Here ϕ must also contain some ordinary variables, whether bound or free, because R cannot exist syntactically without some.) A second-order sentence then has a truth value which depends on the input string, just as a first-order sentence does.

Theorem: Let $\mathcal{F}$ be a set of suitable gate functions as above, containing AND and OR. Then (1) a language is computable by a family of constant depth, quasipolynomial size $DPLT$-uniform circuits with gates using a finite set of functions from $\mathcal{F}$ iff (2) the language is the set of strings satisfying a particular restricted second-order sentence, using quantifiers for functions in $\mathcal{F}$ and connectives and atomic predicates as above.

Proof: That (2) implies (1) is easy. We must show that a logically uniform circuit can be given gate numbers so that direct connection language queries are easy to compute. Just as in [BIS90], we let the numbers of the gates corresponding to quantifiers be the concatenation of strings representing the values of the currently bound variables. These strings are poly-log many bits long because each variable can be specified

with poly-log many bits. Whether one such gate is a child of another, or what type a gate is, can be read off of these strings in poly-log time. For the gates representing the quantifier-free part of the formula, the gate number is a prefix representing all the bound variables followed by a constant-length section. The answer to a query about such a node can be obtained by evaluating the quantifier-free section for the chosen values of the bound variables. These values are readily available from the gate number, and each of the atomic predicates can be evaluated in poly-log time.

To show that (1) implies (2), we first show that any predicate computable in deterministic poly-log time can be expressed by a restricted second-order sentence. This is easier than the analogous result for log-time in [BIS90], because of the more generous closure properties of the poly-log functions. If the time bound to compute the predicate is $\log^k n$, we use a single second-order existential quantifier to guess an array of $\log^{2k} n$ bits, and use first-order quantifiers, connectives, and atomic formulas to express the statement that this array describes an accepting computation of the Turing machine on the given input.

Now that we know that the parent-child relation on gate numbers and the type of gates are each expressible by restricted second-order formulae, we inductively define the value of a gate. The value of an f-gate g is given by $(Q_f R)\phi(R)$, where R ranges over all possible gate numbers and $\phi(R)$ is the value of gate number R if R is a child of g and an identity value otherwise. We can express the value of a gate with our system by defining formulae expressing statements like "g is an accepting f-gate which is at most d levels above the input". The general inductive definition allows us to express this in terms of analogous statements for gates at most $d-1$ levels above the input. We can express membership in the language by expressing the value of the output gate of the circuit.□.

This theorem shows that the logical and Ruzzo-like notions of uniformity coincide for the circuit classes with arbitrary constant depth and a fixed set of gate types. In the next section, we will want to speak of uniform versions of results where depth is fixed. For example, what is a uniform version of SYM^+ or $Midbit^+$? We will define these in terms of logical formulae obeying particular syntactic restrictions corresponding to the combinatorial restrictions on the circuits.

Definition: A SYM^+ sentence is a restricted second-order formula derived from a formula as follows. For a fixed integer k, let $r = \log^k n$ and let L be a lookup function, a boolean function with domain $\{1,\ldots,n^r\}$. Let S be the symmetric boolean function on n^r bits corresponding to L, so that $S(w) = L(|w|)$ where $|w|$ is the number of ones in the binary string w. A SYM^+ sentence is any sentence of the form

$$(Q_S f)(\forall i)[(i \le r) \rightarrow (\phi(f,i) \wedge x_{f(i)})]$$

where ϕ does not access the input (has no π_0 or π_1 predicates). (Since the first-order quantifier here is actually implementing an AND of poly-log many terms, we will later write it as such. It should be clear that there is a purely syntactic translation back to the quantifier form.) A *uniform SYM^+ sentence* is one in which $L(n)$ can be computed in poly-log time from the binary representation of n and $\phi(f,i)$ can be computed in poly-log time from those of f and i. A *uniform Midbit$^+$ sentence* is a uniform SYM^+ sentence where L is the middle-bit function.

The most common way of defining a SYM^+ function is to give an integer polynomial of poly-log degree with quasipolynomial coefficients. If we have a uniform such polynomial, e.g.,

$$R(x_1,\ldots,x_n) = \sum_{D \in \binom{n}{\le r}} c_D \bigwedge_{i \in D} x_i$$

we can easily turn a symmetric function based on it into a uniform SYM^+ sentence, as follows. The quantifier Q_S depends on the chosen lookup function L. It will bind a relation which we will interpret as a pair consisting of f (of $r \log n$ bits) and z (of s bits, where s is a power of $\log n$ such that 2^s exceeds all the c_D's). The formula $\phi(f,z,i)$ must say that (1) f is a increasing function from some prefix of $\{1,\ldots,r\}$ to $\{1,\ldots,n\}$, so that it uniquely represents a subset D, and (2) $z \le c_D$, where f represents D. The sum over all $\langle f,z\rangle$ of

$$\bigwedge_{i=1}^{r} (\phi(f,z,i) \wedge x_{f(i)})$$

is then exactly the value of $R(x_1,\ldots,x_n)$.

4 Uniformity of Upper Bounds

We now turn to previous results concerning quasipolynomial size circuits and consider their uniform versions. We begin with an important result in which uniformity is inherent, that of Allender and Gore [AG91]. They prove that the permanent function cannot be computed in (log-time) uniform ACC^0, and thus that uniform ACC^0 is properly contained in the class PP (unbounded probabilistic polynomial time). No similar bound is known for non-uniform, or even much less uniform, ACC^0. This is in constrast to all the circuit lower bounds mentioned earlier, where the methods were entirely combinatorial and the same bounds are known for both uniform and non-uniform circuits. In a later version of this work [AG92], they also prove that the permanent, and hence PP, is not contained within uniform $qACC^0$, using the same notion of poly-log time uniformity as we use here. This later version also shows, independently of this paper, that the simulation of Beigel and Tarui [BT91] is uniform according to this notion.

In [AH90], Allender and Hertrampf examine the uniformity of Allender's collapse of the qTC^0 depth hierarchy and related depth collapses (such as for $qACC^0[p]$ for prime p). For simplicity, they chose a weaker uniformity condition, that direct connection language queries be answerable in poly-log *space* or

qNC. However, they recognized the possibility of carrying through their analysis with a uniformity notion like that of [BIS90], and much of what we will do in this section draws directly on their work.

Here we will carry out a uniform version of the simulation [GKT92] of a uniform $qACC^0$ circuit family by a uniform $Midbit^+$ sentence. This uses all the techniques of the earlier simulation [BT91] by a SYM^+ sentence, but will be a bit simpler notationally. We begin by applying the result of the previous section to make the $qACC^0$ family logically uniform, with prime moduli on the gates (this involves a few syntactic tricks, such as simulating a quantifier for the MOD_{p^e} function using MOD_p quantifiers — these are straightforward and we omit them). As we outlined above, we now need uniform versions of the "Valiant-Vazirani trick" and the "Toda trick".

For the first, we put a MAJORITY quantifier at the front of the formula, binding a variable R which ranges over all choices of the random bits we need at each gate. Then a second–order $\exists$ quantifier $\exists S\phi(S)$, for example, gets converted into:

$$(Q^{MAJ}R)(\bigvee_{i=1}^{polylog})(Q^{MOD_p}S)\phi'(R,S,i)$$

, where ϕ' is closely related to ϕ. More specifically, the same majority quanitifier at the front of the whole formula binds a common random bit string R which applies to all the quantifiers simulated. By choosing R long enough (but still poly-log many bits) we can ensure that most R work for all the quanitifiers, and thus that this step of the simulation is correct. Getting rid of the poly-log-wide ORs requires an example of the distributive law addition and multiplication, just as described in [AH90]. We can do this with any two operations which act like multiplication (e.g., AND, to which the ORs may easily be converted) and addition (e.g., MOD_p for prime p). Consider a formula of the form

$$\prod_{i=1}^{r}\sum_{S} f(i,S),$$

where S ranges over k-ary relations on $\{1,\ldots,\log n\}$ and $r=\log^\ell n$. Let the relation T range over functions from $\{1,\ldots,r\}$ to k-ary relations. The formula now has an equivalent form

$$\sum_{T}\prod_{i=1}^{r} f(i,T(i)).$$

By successive steps of this form we can move all the small ANDs and ORs to the end of the formula.

We now have a formula with a MAJORITY quantifier at the front, a sequence of MOD_p quantifiers for various primes p, and a poly-log restricted $\forall$ at the end. By adding dummy variables, the MAJORITY quantifier may be made into a *Midbit* quantifier. The next step is to merge the MOD_p quantifiers one by one into the *Midbit* symmetric quantifier at the front using a Toda polynomial. To do this uniformly, we must show (1) the coefficients of a Toda polynomial can be calculated in poly-log time, (2) the composition of two uniform polynomials is another uniform polynomial, and (3) given a uniform polynomial f whose residue modulo p^t is b, we can make another uniform polynomial g such that the middle bits of the binary expansion of g are b.

For (1), we merely look at the explicit formula for $P_t(x)$ given by Beigel and Tarui [BT91]. We need to multiply two polynomials of degree t each of whose coefficients are given in terms of binomial coefficients. The obvious algorithm to evaluate these coefficients takes time polynomial in t, which is poly-log.

For (2), start with an original polynomial

$$f=\sum_{S}\bigwedge_{i=1}^{\log^k n}(\phi(S,i)\wedge x_{S(i)})$$

where S ranges over $(\log^{k+1} n)$-bit strings interpreted as functions from $\{1,\ldots,\log^k n\}$ to $\{1,\ldots,n\}$. We want to apply another polynomial (in the intended application, a Toda polynomial) to the integer which is the output of this one. Let the new polynomial g be

$$\sum_{T}\bigwedge_{j=1}^{\log^\ell n}(\psi(T,j)\wedge y_{T(j)}).$$

We pick the version of g where T ranges over functions from $\{1,\ldots,\log^\ell n\}$ to $\{1,\ldots,\log^{k+1} n\}$. Thus $T(j)$ can be interpreted as a function from $\{1,\ldots,\log^k n\}$ to $\{1,\ldots,n\}$. We get

$$y_{T(j)}=\bigwedge_{i=1}^{\log^k n}(\phi(T(j),i)\wedge x_{T(j)(i)}),$$

so that $g\circ f$ can be written in the proper form as

$$\sum_{T}\bigwedge_{j=1}^{\log^\ell n}\bigwedge_{i=1}^{\log^k n}(\psi(T,j)\wedge\phi(T(j),i)\wedge x_{T(j)(i)}).$$

For (3), the arithmetic done in [GKT92] to accomplish this is very simple. In addition to evaluating the Toda polynomials themselves, we need to raise some primes to poly-log powers and perform integer divisions and multiplications on the results. All of this is easily accomplished in poly-log time, and the resulting polynomials remain of poly-log degree with quasipolynomial coefficients.

Acknowledgements

I got the idea and considerable inspiration for this survey at the 1992 McGill Invitational Workshop on Complexity, organized by Pierre McKenzie and Denis Thérien. I would particularly like to thank the participants in that workshop, especially Richard Beigel and Eric Allender. Sections 3 and 4 draw heavily on ongoing joint research I am doing with Neil Immerman.

I would like to thank him, Vivek Gore, Fred Green, Jun Tarui, and the students in Neil's and my spring 1992 UMass complexity theory course (especially Zhi-Li Zhang).

References

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

[Al89] E. Allender, "A note on the power of threshold circuits", *Proc. 30th IEEE FOCS* (1989), 580-584.

[AG91] E. Allender and V. Gore, "On strong separations from AC^0", *Proc. 8th Int. Conf. on Fundamentals of Computation Theory (FCT '91) (Springer LNCS 529)*, 1-15.

[AG92] E. Allender and V. Gore, "A uniform circuit lower bound for the permanent," draft (1992).

[AH90] E. Allender and U. Hertrampf, "Depth reduction for circuits of unbounded fan-in", *Information and Computation*, to appear. Preliminary version available as technical report (Dec. 1990), Rutgers University.

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

[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.

[BBR92] D. A. M. Barrington, R. Beigel, and S. Rudich, "Representing Boolean functions as polynomials modulo complex numbers", *Proc. 24th ACM STOC* (1992), to appear.

[BIS90] D. A. M. Barrington, N. Immerman, and H. Straubing, "On uniformity within NC^1," *J. Comp. Syst. Sci.* **41** (1990), 274-306. *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* **89:2** (Dec. 1990), 109-132.

[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.

[BLM90] F. Bédard, F. Lemieux, and P. McKenzie, "Extensions to Barrington's M-program model", *Proc. 5th Structure in Complexity Theory* (1990), 200-210. Extended version *Theoretical Computer Science*, to appear.

[BRS91] R. Beigel, N. Reingold, and D. Spielman, "The preceptron strikes back", *Proc. 6th Structure in Complexity Theory* (1991), 286-291.

[BT91] R. Beigel and J. Tarui, "On ACC", *Proc. 32nd IEEE FOCS* (1991), 783-792.

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

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

[HG90] J. Håstad and M. Goldmann, "On the power of small-depth threshold circuits", *Proc. 30th IEEE FOCS* (1990) (Volume II), 610-618.

[GKT92] F. Green, J. Köbler, and J. Torán, "The power of the middle bit", these proceedings.

[Ha88] J. Håstad, *Computational Limitations of Small Depth Circuits*, (Cambridge, USA: MIT Press, 1988).

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

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

[Li92] S. Lindell, "A purely logical characterization of circuit uniformity", these proceedings.

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

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

[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.

[Si83] M. Sipser, "Borel sets and circuit complexity" *Proc. 15th ACM STOC* (1983), 61-69.

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

[To91] S. Toda, "PP is as hard as the polynomial-time hierarchy", *SIAM J. Computing* **20:5** (Oct. 1991), 865-877.

[VV86] L. G. Valiant and V. V. Vazirani, "NP is as easy as detecting unique solutions", *Theoretical Computer Science* **47** (1986), 85-93.

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

[Ya90] A. C. C. Yao, "On ACC and threshold circuits", *Proc. 31st IEEE FOCS* (1990), 619-627.