Electronic Colloquium on Computational Complexity, Report No. 33 (2001)

ECCC

Uniform Circuits for Division: Consequences and Problems*

ERIC ALLENDER †
Dept. of Computer Science
Rutgers University
allender@cs.rutgers.edu

DAVID A. MIX BARRINGTON ‡
Dept. of Computer Science
University of Massachusetts
barring@cs.umass.edu

WILLIAM HESSE §
Dept. of Computer Science
University of Massachusetts
whesse@cs.umass.edu

*To appear in Proc. IEEE Conference on Computational Complexity, 2001
†Supported in part by NSF grant CCR-9734918.
‡Supported in part by NSF grant CCR-9988260.
§Supported by NSF grant CCR-9877078.

ISSN 1433-8092

Abstract

Integer division has been known to lie in P-uniform TC^0 since the mid-1980's, and recently this was improved to L-uniform TC^0. At the time that the results in this paper were proved and submitted for conference presentation, it was unknown whether division lay in DLOGTIME-uniform TC^0 (also known as FOM). We obtain tight bounds on the uniformity required for division, by showing that division is *complete* for the complexity class FOM + POW obtained by augmenting FOM with a predicate for powering modulo small primes. We also show that, under a well-known number-theoretic conjecture (that there are many "smooth" primes), POW (and hence division) lies in FOM. Building on this work, Hesse has shown recently that division is in FOM [17].

The essential idea in the fast parallel computation of division and related problems is that of Chinese remainder representation (CRR) – storing a number in the form of its residues modulo many small primes. The fact that CRR operations can be carried out in log space has interesting implications for small space classes. We define two versions of $s(n)$ space for $s(n) = o(\log n)$: dspace$(s(n))$ as the traditional version where the worktape begins blank, and DSPACE$(s(n))$ where the space bound is established by endmarkers before the computation starts. We present a new translational lemma, and derive as a consequence that (for example), if one can improve the result of [15] that $\{0^n : n \text{ is prime}\} \notin$ dspace$(\log\log n)$ to show that $\{0^n : n \text{ is prime}\} \notin$ DSPACE$(\log\log n)$, it would follow that L $\neq$ NP.

1 Introduction

The exact complexity of division, powering, and iterated multiplication of integers has been a major open problem since Beame, Cook, and Hoover [7] showed these problems to be in P-uniform TC^0 in 1984[1]. (TC^0 is the set of problems solvable by threshold circuits of constant depth and polynomial size, "P-uniform" means that these circuits can be constructed by a poly-time Turing machine.) In a recent breakthrough, Chiu, Davida and Litow [11] showed these problems to be in L-uniform TC^0, (i.e., the circuits can be constructed in log space). They thus also solved an even older open problem by showing these problems to be solvable in log space itself.

This result naturally raises the question of whether these problems fall within the most natural version of "uniform TC^0", that of DLOGTIME-uniform circuits or problems definable by first-order formulas with MAJORITY quantifiers. Here we

[1] [7] claimed only P-uniform NC^1, but it was observed later by Reif [23] that their algorithm is implementable in TC^0.

show that the non-uniformity necessary for the construction of [11] is quite limited: In Immerman's descriptive complexity setting [19], we need only first-order formulas with MAJORITY quantifiers and a single extra numerical predicate. This predicate calculates powers modulo a prime of $O(\log n)$ bits.

We then consider various algorithms for powering modulo a small prime, and their consequences for the uniformity of division circuits. Following an argument of Chiu [10], powering modulo a small prime can be achieved by fully-uniform circuits of logarithmic depth and fan-in two ("Ruzzo-uniform NC^1") and hence division is itself in uniform NC^1, in fact in NC^1-uniform TC^0. But we also show that powering modulo a small prime lies in another natural class that *provably* does not contain all of NC^1, suggesting that the full power of NC^1-uniform TC^0 is not needed for division. Finally, we show that the special case of powering modulo a *smooth* prime is in fully-uniform TC^0. It follows that under a widely-believed but unproven hypothesis about the density of smooth primes, division and related problems actually *are* in fully-uniform TC^0.

Subsequent to this work, Hesse [17] has shown unconditionally that powering modulo *any* small prime is in fully-uniform TC^0 (in fact it is in fully-uniform AC^0). Thus integer division and the related problems considered here *are* actually all in fully-uniform TC^0. This work will not be described here but will appear together with the results of this extended abstract in a forthcoming paper [18].

Finally, we consider the implications of the new division algorithm for the study of small-space complexity classes. Most prior work on Turing machines with $O(\log \log n)$ space, for example, has assumed that the work tape starts out blank, with no marker to indicate the end of the available space. We call this class dspace$(\log \log n)$, in contrast to the class DSPACE$(\log \log n)$ where this initial marker is given.

The space-efficient CRR algorithms allow us to prove more efficient translational arguments, showing that the *unary* languages in DSPACE$(\log \log n)$ are simply the unary encodings of the languages in log space. This highlights the difference between dspace and DSPACE classes. For example, a classic result of Hartmanis and Berman [15] says that the set of unary strings of prime length is not in dspace$(o(\log n))$. The new translational lemma shows that proving an analogous result for DSPACE$(\log \log n)$ would separate the classes L and NP.

2 Circuits for Division: An Overview

We are concerned with the complexity of three basic problems in integer arithmetic (with input and output in binary representation):

- DIVISION: Given a number X of n bits and a number Y of at most n bits, find $\lfloor X/Y \rfloor$,

- POWERING: Given a number X of n bits and a number k of $O(\log n)$ bits, find X^k, and
- ITERATED MULTIPLICATION: Given n numbers $X_1, \ldots, X_n$, each of at most n bits, find the product $X_1 X_2 \ldots X_n$.

Beame, Cook, and Hoover [7] showed that each of these problems can be solved by a family of *threshold circuits* of constant depth and polynomial size. As always in circuit complexity, a key property of the circuit family is its degree of *uniformity*. The circuits of [7] are P-uniform, in that they can be constructed in time polynomial in n. Thus [7] shows that these problems are in the class *P-uniform* TC^0.

Recently Chiu, Davida, and Litow [11] have dramatically improved this result by constructing circuit families for these problems that are constructible in *log space* (hence putting the problems in *L-uniform* TC^0). A logspace machine can simulate a constant-depth threshold circuit if it can construct it, so this result also solves the longstanding open problem of putting these problems into the class L (deterministic log space).

We present a simplified division algorithm that was inspired by [11], and in our presentation we pay close attention to the use of non-uniformity in the circuits. To do this we use the formalism of *descriptive complexity* developed by Immerman [5, 4, 19], where constant-depth circuit families are presented in the form of *first-order formulas*. For example, the complexity class FO consists of those languages that can be described by first-order formulas where the variables range over the universe $\{1,\ldots n\}$, corresponding to the n positions of the input string, there are atomic formulas for equality, order, addition, and multiplication, as well as a formula $X[i]$ that evaluates to true if the ith symbol of the input is 1, and there are first-order quantifiers $\exists$ and $\forall$. This class FO is equivalent to the circuit complexity class DLOGTIME uniform AC^0 — see [5] for the exact definition and the proof. The larger class FOM ("first-order with MAJORITY") is the same as FO except that MAJORITY *quantifiers* may be used in the formulas. This class is equal in power to DLOGTIME uniform families of threshold circuits of constant depth and polynomial size ("DLOGTIME-uniform TC^0"). In FOM one can multiply two n-bit numbers, add together n n-bit numbers, and of course carry out all operations in FO.

The central idea of all the TC^0 algorithms for DIVISION and related problems is that of *Chinese remainder representation (CRR)*. An n-bit number is uniquely determined by its residues modulo polynomially many primes, each of $O(\log n)$ bits. (The Prime Number Theorem guarantees that there will be more than enough primes of that length.) Of course, our problems specify that their input and output must be in ordinary binary notation, so we are faced with the problem of converting to and from CRR.

We observe in Lemma 3.1 that converting from binary to CRR can be accomplished if we augment FOM with the *power predicate* (POW) "$a^i \equiv b \pmod{p}$" where a, b, i and p each have $O(\log n)$ bits, and p is prime. Since the multiplicative group of a prime number is cyclic, the predicate POW allows us to identify a generator of this group (the least g such that $g^{p-1} \equiv 1 \pmod{p}$ and no smaller power of g is 1) and to compute *discrete logarithms* modulo p for each number. More precisely, there is a first-order formula GEN(g, p) that has POW as a predefined predicate, that is true if and only if g is the least generator of the multiplicative group mod p. Thus finding a generator can be accomplished in the complexity class FO$+$POW. We will also call this a FO *reduction* to POW from the problem of finding generators. Similarly, it is easy to see that computing discrete logs mod p can be performed in FO + POW.

Finally, note that *if the input and output are in CRR*, the iterated multiplication problem simply reduces to the iterated addition problem (by adding the discrete logs), showing that iterated multiplication (in CRR) is in FOM once the power predicate is present (i.e., it is in the class FOM + POW).

This construction was used in [7], but additional work is required in order to compute these functions in binary, instead of CRR. In fact, most of the division algorithm presented by Beame, Cook, and Hoover in [7] is in FOM — the only exceptions are computation of discrete logarithms and conversion to and from CRR. In order to convert from binary to CRR, Beame, Cook and Hoover needed an additional predicate: the binary representation of the product of the first n^3 primes. While the power predicate is easily seen to be computable in logspace, this prime-product predicate was not known to be so computable. The central contribution of [11] is to develop better methods for working with CRR, so that the prime-product predicate is no longer needed. In the next section, we present a procedure for conversion from CRR to binary that can be computed in FOM$+$POW. Thus the power predicate, the essential ingredient in converting a binary number *into* CRR, is powerful enough (along with FOM operations) to get a number *out of* CRR into binary.

3 Converting CRR Numbers to Binary

We will refer to numbers with polynomially many bits as *long numbers*, and denote them by capital letters. Numbers of $O(\log n)$ bits will be called *short*, and denoted by small letters. We are given two long numbers X and Y and asked to find $Z = \lfloor X/Y \rfloor$. Note that it suffices to find a small number of candidates for Z, as in FOM we can compute ZY for any candidate and then verify that $X - ZY$ is non-negative and less than Y.

To fix notation, we now recapitulate the development of CRR. If we are given a

sequence of distinct primes $m_1, \ldots, m_k$, each a short number, let M be their product. Any number $X < M$ can be represented uniquely as $(x_1, \ldots, x_k)$ with $X \equiv x_i \pmod{m_i}$ for all i. For each number i, let C_i be the product of all the m_j's except m_i, let h_i be the inverse of C_i modulo m_i. It is easy to verify that X is congruent modulo M to $\sum_{i=1}^{k} x_i h_i C_i$. In fact X is *equal*, as an integer, to $(\sum_{i=1}^{k} x_i h_i C_i) - rM$ for some particular number r, called the *rank* of X with respect to M (denoted $\text{rank}_M(X)$). Note that r is a short number. It is equal to the integer part of the sum of the k rational numbers $x_i h_i C_i / M$ or $x_i h_i / m_i$, each of which is between 0 and m_i.

The computation of this rank function is central to the argument of [11] that DIVISION is in L-uniform TC^0. It is computable in logspace [13, 21], and in fact the algorithms can be adapted to put it in FOM$+$POW. (For more detail on this see [2].) Here we present a self-contained argument, without computing rank directly, that conversion from CRR to binary and DIVISION are in FOM + POW. First we note again that we can carry out the other conversion, from binary to CRR.

Lemma 3.1 *If $X, m_1, \ldots, m_k$ are each given in binary and $X < M$, we can compute $(x_1, \ldots, x_k)$ (the CRR_M form of X) in* FOM + POW.

Proof. For each modulus m_i and each $j < n$ we must calculate $2^j \pmod{m_i}$ (given by the power predicate), add the results (using iterated addition in FOM), and take the result modulo m_i (in FO). ■

It will be useful to observe that dividing by a short prime is easy.

Lemma 3.2 *Let p be a short prime. Then the binary representation of $1/p$ can be computed to $n^{O(1)}$ bits of accuracy in* FO + POW.

Proof. Let p be odd and write 2^k as $ap + b$ with $b = 2^k \bmod p$. The kth bit of the binary expansion of the rational number $1/p$ is defined to be the low-order bit of a. But since ap is congruent to b modulo 2 and p is odd, this is also the low-order bit of b. The latter can clearly be computed in FO + POW. ■

Lemma 3.3 *[13, 14] Let X and Y be numbers less than M given in CRR_M form. In* FOM + POW *we can determine whether $X < Y$.*

Proof. Clearly, $X < Y$ if and only if $X/M < Y/M$. Thus it is sufficient to show that we can compute X/M to polynomially-many bits of accuracy.

Recall that $X = (\sum_{i=1}^{k} x_i h_i C_i) - \text{rank}_M(x) M$. Thus X/M is equal to $(\sum_{i=1}^{k} x_i h_i (1/m_i)) - \text{rank}_M(x)$. The numbers x_i are given to us as the CRR_M of X. The number $C_i \bmod m_i$ can be computed in FOM + POW (by adding the discrete logs of the m_j for $j \neq i$), and h_i is simply the inverse of that number mod m_i. By Lemma 3.2, each

summand can be computed in FOM + POW to $n^{O(1)}$ bits of accuracy. Since iterated addition is in FOM, we can thus compute polynomially-many bits of the binary representation of $(\sum_{i=1}^{k} x_i h_i(1/m_i))$, which is equal to $X/M + \text{rank}_M(X)$. Since the rank is an integer, X/M is simply the fractional part of this value and this is just the value that we want. ■

One useful consequence of being able to compare integers in CRR is that it enables us easily to convert from one CRR basis to another. That is, if we are given X in CRR_M for one list of moduli $m_1, \ldots, m_k$, $M = \prod_{i=1}^{k} m_i$ and we want to convert to CRR_P for some list of distinct short primes $p_1, \ldots, p_l$, $P = \prod_{i=1}^{l} p_i$, all that is necessary is to compute $X \bmod p$ for an arbitrary short prime p.

Lemma 3.4 *Given X in CRR_M and a short prime p, we can compute $X \bmod p$ in* FOM + POW.

Proof. If p is one of the moduli in M, the answer is given explicitly in the input. Thus we assume that p does not divide M. In this case, consider the CRR base $M' = Mp$. We would like to compute X in $\text{CRR}_{M'}$, since this would give us $X \bmod p$.

Trying each of the $p = n^{O(1)}$ possible values i for $X \bmod p$, we obtain the $CRR_{M'}$ of $n^{O(1)}$ different numbers $X_0, X_1, \ldots, X_{p-1}$, one of which is X. It is easy to see that X is the only one of these numbers that is less than M.

Observe that in FOM + POW we can compute the $CRR_{M'}$ of M (by adding the discrete logs of the m_j mod p). Thus we can compute $X \bmod p$ by finding the unique X_i that is less than M, carrying out all comparisons in $\text{CRR}_{M'}$, by Lemma 3.3. ■

Our next step in the division algorithm is to show how to divide by products of distinct short primes.

Lemma 3.5 *Let $b_1, \ldots, b_\ell$ be distinct short primes, B be the product of the b_i's, and let X be given in* CRR_M *form. Then we can compute $\lfloor X/B \rfloor$, also in* CRR_M *form, in* FOM + POW.

Proof. Assume without loss of generality that B divides M. (Otherwise, extend the basis, using Lemma 3.4.) Thus let $M = BP$ where $P = \prod_{i=1}^{k} p_i$.

In FOM + POW we can compute the following quantities:

- B in CRR_M (by adding the discrete logs modulo each m_i),
- The CRR_M of $S = (\sum_{i=1}^{\ell} x_i h_i(B/b_i))$, where h_i is the multiplicative inverse of $B/b_i \bmod b_i$,
- $Y = X - S$ in CRR_M,

- $B^{-1} \bmod P$ (i.e., the unique number $T < P$ such that $BT \equiv 1 \pmod{P}$; this can be computed in CRR_P by merely inverting each nonzero component of the CRR_M of B).

Note that $S \equiv X \bmod B$, and also $S = \sum_{i=1}^{\ell} x_i h_i (B/b_i) < \sum_{i=1}^{\ell} b_i B < \ell b B$, where b is the maximum of the b_i. Note also that Y is a multiple of B, and thus Y/B is an integer. Also note that $Y/B < P$, since $Y \leq X < BP$. Thus if we compute YT in CRR_P we have the CRR_P of the integer Y/B, and from this we can compute Y/B in CRR_M.

Therefore $\lfloor X/B \rfloor$ differs from $Y/B = YT$ by at most $S/B + O(1) = n^{O(1)}$. That is, we can compute a list of $n^{O(1)}$ consecutive values, one of which is equal to $\lfloor X/B \rfloor$. We can find the correct value by determining the value j such that $(YT + j)B \leq X < (YT + j + 1)B$. ■

Theorem 3.6 *Let X be given in* CRR_M *form. Then we can compute the binary representation of X in* FOM + POW.

Proof. We remark that a simple extension of this result and Lemma 3.1 shows that it is possible in FOM + POW to convert numbers from any base to another, by first converting to CRR.

It is sufficient to show that we can compute the CRR_M of $\lfloor X/2^k \rfloor$ for any k. This is because, to get the k-th bit of a number X that is given to us in CRR, we compute $u = \lfloor X/2^k \rfloor$ and $v = \lfloor X/2^{k+1} \rfloor$, and note that the desired bit is $u - 2v$. We get this bit as a CRR number, but it is easy to recognize the CRR forms of the numbers 0 and 1.

First, we create numbers $A_1, \ldots, A_k$, each a product of polynomially many short odd primes that do not divide M, with each $A_i > M$. Let $P = M \prod_{i=1}^{k} A_i$, and compute X in CRR_P. By Lemma 3.5 (or directly) we can compute $(1 + A_i)/2$ in CRR_P. It is easy to show that $(\prod_{i=1}^{k}(A_i + 1))/\prod_{i=1}^{k} A_i < 1 + (k/M)$.

Note that in FOM + POW we can compute the CRR_P representation of $Q = \lfloor X \prod_{i=1}^{k}((1 + A_i)/2)/\prod_{i=1}^{k} A_i \rfloor$. But $X \prod_{i=1}^{k}((1 + A_i)/2)/\prod_{i=1}^{k} A_i$ is equal to $(X/2^k)(\prod_{i=1}^{k}(A_i+1))/\prod_{i=1}^{k} A_i < (X/2^k)(1+(k/M))$. Thus $Q \in \{\lfloor X/2^k \rfloor, \lfloor X/2^k \rfloor + 1\}$. We determine which of $\{Q, Q-1\}$ is the correct answer by checking if $Q2^k > X$ (using the CRR_P representation). ■

Corollary 3.7 DIVISION *is in* FOM + POW.

Proof. It is easy in FOM to divide numbers in binary if we are able to compute powers in binary [7] — we simply use a power series to approximate the quotient to within an additive error of one, and test the two possible integer quotients using

the FOM multiplication algorithm. But we have seen that ITERATED MULTIPLICATION, and hence POWERING, are in FOM + POW if the input and output are in CRR form. The FOM + POW algorithms to convert from binary to CRR and vice versa thus suffice to perform division (in binary) in FOM + POW. ■

4 The Class FOM + POW

What does this new algorithm finally tell us about the complexity of DIVISION? In one sense the circuit complexity of DIVISION has been well-understood since [7]; DIVISION can be computed by threshold circuits of constant depth and polynomial size, and since MAJORITY is reducible to DIVISION, we cannot hope to put DIVISION into a smaller circuit class.

The remaining question, of course, is *how uniform* the threshold circuits can be made to be. The main result of [11] is that the P-uniform circuits of [7] can be made L-uniform, with the important consequence that DIVISION is in L itself. Our analysis of their algorithm tells us something more, that DIVISION is in the class we have called FOM + POW. Although this class is now known to be equal to FOM itself [17], the following closer analysis of it still raises some interesting complexity issues.

Here is a list of problems that have been known since [7] to be in P-uniform TC^0 but were not known (before [17]) to be in FOM:

- DIVISION
- POW
- ITERATED MULTIPLICATION
- POWERING
- CONVERTING CRR TO BINARY[2]
- CONVERTING BINARY TO CRR
- DIVISIBILITY (i.e., given X and Y, does X divide Y?)

Note that CONVERTING BINARY TO CRR and DIVISIBILITY were known previously to be in L-uniform TC^0, by the arguments of [7]. All of these problems are in FOM + POW. For some of these problems, this is optimal, as the following observations show.

[2]To be completely formal, the statement of this problem should include a specification of the moduli used in CRR. For the purposes of this paper any reasonable definition is sufficient, and hence we leave this unspecified.

Proposition 4.1 DIVISION *is complete for* FOM + POW *under* FO *reductions.*

Proof. We must show that both MAJORITY and POW are obtainable by FO-reduction from DIVISION. Note first that Beame, Cook, and Hoover presented a FO reduction from POWERING to DIVISION in [7]. This suffices to also reduce the binary MULTIPLICATION problem to DIVISION, because the expression $XY = [(X+Y)^2 - X^2 - Y^2]/2$ reduces MULTIPLICATION to POWERING. MAJORITY is reducible to MULTIPLICATION as shown in [9].

It remains to solve the POW predicate $a^i \equiv b \pmod{m}$ with these tools. We can use POWERING to compute a^i, find $q = \lfloor a^i/m \rfloor$ using DIVISION directly, compute $a^i - qm$ using MULTIPLICATION, and compare the result to b. ■

Although the other problems are not known to be hard under FO reductions, some of them are complete under FOM reductions.

Proposition 4.2 POW, ITERATED MULTIPLICATION, *and* POWERING *are complete for* FOM + POW *under* FOM *reductions.*

Proof. (of Proposition 4.2) For POW this is a trivial observation. Since all of these problems are in FOM + POW, they are all FO-reducible to DIVISION. An argument in [7] reducing DIVISION to POWERING is easily seen to provide a FOM reduction. And, of course, POWERING is a special case of ITERATED MULTIPLICATION. ■

A natural approach to determining the complexity of all these problems, then, is to consider a variety of algorithmic attacks on POW. It is easy to see that POW is in L, and hence that FOM+POW is contained in L-uniform TC^0. A clever application of the *result* of [11], due to Chiu [10], gets us further:

Proposition 4.3 *The problem* POW *is in Ruzzo-uniform* NC^1 *[24]. Hence* FOM + POW *is contained in both uniform* NC^1 *and* NC^1*-uniform* TC^0.

Proof. (of Proposition 4.3) Since POWERING and DIVISION are each in L-uniform NC^1 by [11], for any k we can raise a k-bit number to a $\log k$-bit power modulo a k-bit number using circuits of depth $O(\log k)$ that are DSPACE$(\log k)$-uniform. Taking $k = \log n$, we can raise a small number to the power $\log n$ modulo a small number, using a circuit of depth $O(\log \log n)$ that is DSPACE$(\log \log n)$-uniform. Making a $\log n$-ary tree of such circuits, we can compute POW itself with a circuit whose depth is ($O(\log \log n)$ times $O(\log n/(\log \log n))) = O(\log n)$ that is fully uniform except for the $O(\log \log n)$-depth components that are DSPACE$(\log \log n)$ uniform.

Proving Ruzzo-uniformity of this NC^1 circuit requires answering questions about the extended connection language of the circuit, that is, questions about the node

numbers of paths of length $\log n$ through the gates [24]. We can answer such questions in FO as long as we can do so for each individual DSPACE$(\log \log n)$-uniform component. This means questions about paths of length $O(\log \log n)$ where each edge is decidable by a DSPACE$(\log \log n)$ machine. Replacing each edge with a state graph for the machine, we get a question about paths in a uniform graph with polylog many nodes. But any such questions are in FO, as desired. The full paper will contain further details about such "short path" problems. ■

A second attack on POW places it in a new complexity class defined in a recent paper of Barrington, Kadau, Lange, and McKenzie [6]. They looked at groups presented as multiplication tables and the complexity of various problems including that of computing powers. These results apply directly to the group of integers modulo m (where m is polynomial in n) because the product operation of this group is FO computable. They showed that powering in such a group, and thus POW, is in a new complexity class they called FOLL.

The class FOLL is defined to be those languages definable by first-order formulas with a quantifier block iterated $O(\log \log n)$ times, or equivalently languages recognized by uniform circuit families (of AND and OR gates) of depth $O(\log \log n)$, polynomial size, and unbounded fan-in. The key step in computing POW is to note that a^{jk}, for example, is FO computable from the complete table of j-th and k-th powers, since $a^{jk} = b$ iff $\exists c : (a^j = c) \wedge (c^k = b)$. Thus each round of FO computation *squares* the highest power computed, and (with some other clauses in the definition) after $O(\log \log n)$ rounds all powers polynomial in n can be computed.

FOLL clearly contains FO and is contained in uniform AC^1, and both containments are proper, but little else is known about FOLL (For example, is it contained in NC^1, L or NL?) We do at least know, by well-known lower bounds on circuit size and depth (e.g.,[25]), that the parity language is *not* in FOLL. Since FOLL is closed under FO reductions, it follows that *no language* in FOLL can be complete under such reductions for any class including parity, in particular for L, NC^1, or FOM.

The fact that POW is not complete for NC^1 suggests that FOM + POW does not have the full computational power of NC^1-uniform TC^0. It is plausible, given this analysis that POW itself is in FOM, and thus that the class FOM + POW (including DIVISION) collapses to FOM. In the next section we consider yet another algorithmic attack on POW, which (assuming a number-theoretic hypothesis) *does* effect this collapse. (In fact, subsequent to this work the collapse has been shown to occur unconditionally [17].)

5 Division in FOM Given Enough Smooth Primes

In this section, we show that a widely-believed and "empirically true" number-theoretic hypothesis implies that Division is in FOM. (Though this result has subsequently been shown to hold unconditionally [17], we hope the proof techniques will still be of interest.)

We define a *smooth prime* to be a prime p for which $p-1$ factors completely into small prime powers. If the smooth primes are sufficiently dense in the set of all primes, then integer division can be performed over a CRR basis containing only smooth primes. The calculations over this smooth prime basis will reduce to smaller instances of division and iterated multiplication. We will repeat this reduction a constant number of times, until we have instances of division that can be computed directly in FOM.

There are different degrees of smoothness. In number theory, a number is called Y-smooth if all of its prime factors are less than Y. Here we will require enough primes p such that all of the prime power factors of $p-1$ are less than $2^{(\log p)^{1-\epsilon}}$, which is equal to $p^{(1/\log p)^{\epsilon}}$. We need at least an inverse polynomial fraction of the primes to have this property, specifically:

Hypothesis 5.1 *There exist natural numbers M, c and a real number $\epsilon > 0$ such that for any $n > M$, there are n primes $p_1, \ldots, p_n$ such that (a) each p_i is less than n^c, and (b) any prime power q^k dividing $p-1$ satisfies $q^k < 2^{(\log n)^{1-\epsilon}}$.*

Much stronger hypotheses about smooth primes are widely believed to be true; a much larger proportion of numbers (not necessarily of the form $p-1$) are known to satisfy much stronger smoothness conditions [8]. Empirically, the fraction of numbers of the form $p-1$ that are Y-smooth is approximately the same as the fraction of all numbers that are Y-smooth, for all Y. The best that has been proven, however, is that a significant fraction of primes p less than n have $p-1$ an $n^{3/10}$-smooth number [3]. This is not sufficient for our purposes.

Under Hypothesis 5.1, there are enough smooth primes so that we may carry out the iterated product and division algorithm presented in Section 3 using *only smooth primes* as CRR moduli. Thus we need only evaluate the POW predicate for smooth primes p. (We can determine in FO whether a given prime is smooth.) This allows us to reduce POW to instances of POW with significantly smaller inputs.

We define the problem POW(x), for x a (reasonably constructible) function of n, to be the set of tuples $\langle a, i, b, m\rangle$ where $a^i \equiv b \pmod{m}$ and each of a, i, b, and m have at most x bits. Whenever $x = O(\log n)$, the arguments of Sections 3 and 4 tell us that POW(x) is FOM-equivalent to the special case of DIVISION with arguments of 2^x bits or to the special case of POWERING where the base has 2^x bits and the exponent has x bits.

Lemma 5.2 *Assume Hypothesis 5.1. Then for* $x = O(\log n)$, POW(x) *is* FOM*-reducible to* POW($x^{1-\epsilon}$)*, where* $\epsilon > 0$ *is the parameter in the hypothesis.*

Proof. (of Lemma 5.2)

Given the hypothesis, there are enough smooth primes of $O(x)$ bits to form a CRR basis that will allow us to perform DIVISION and ITERATED MULTIPLICATION on numbers of 2^x bits. Thus POW(x) FOM-reduces to the special case of POW(x) consisting only of those instances where the modulus is a smooth prime. Now we will show that we can reduce such instances of POW to those where in addition, the exponent i has only $x^{1-\epsilon}$ bits.

In general, let p be a prime number, and let d be an upper bound on the size of the prime power factors of $p-1$. That is, let $p-1$ be factorized as $q_1^{r_1} q_2^{r_2} \ldots q_k^{r_k}$ where d is the maximum of the numbers $q_i^{r_i}$. We will show that instances of POW with modulus p reduce to instances of POW with the same modulus and with the exponent less than d. The query $a^i \equiv b \pmod{p}$ requires computing the i'th power in the multiplicative group of residue classes modulo p, denoted $\mathbb{Z}_p^*$. This multiplicative group is a cyclic Abelian group, so it is a product of Abelian groups with orders equal to the prime power divisors of $p-1$.

The multiplicative group $\mathbb{Z}_p^*$ is cyclic, so it has a generator, g. Given $a \in \mathbb{Z}_p^*$, the discrete log of a, $l(a)$, is the unique natural number less than $p-1$ such that $g^{l(a)} \equiv a \pmod{p}$. Note that if s is any divisor of $p-1$, then b is an s-th power of an element $a \in \mathbb{Z}_p^*$ if and only if $l(b)$ is a multiple of s. This follows, since $a^s = b$ iff $sl(a) \equiv l(b) \pmod{p-1}$, which is equivalent to the existence of an integer t such that $l(b) = s(l(a) - (p-1/s)t)$.

Let us now observe that we can find a generator of $\mathbb{Z}_p^*$ in FO + POW, invoking POW only with exponents less than d. It can be seen that an element of $\mathbb{Z}_p^*$ is a generator if and only if its discrete log is relatively prime to $p-1$. We know that $(l(b), p-1) = 1$ if and only if q_j does not divide $l(b)$, for any q_j. As observed above, this holds if and only if b is not a q_j-th power, for any q_j. That is, b is a generator if and only if $\forall a \; a^{q_j} \neq b$. Thus by an FO-reduction to POW with exponent q_j, we can find the least such b and call it our generator g.

Now let us show that all instances of POW for a smooth prime can be solved in FO + POW, invoking POW only with exponents less than d. Since the q_j's are relatively prime, we can represent numbers less than $p-1$ in CRR using the $q_j^{r_j}$ as moduli. Given $a \in \mathbb{Z}_p^*$, where the CRR of $l(a)$ is $(a_1, \ldots, a_k)$, we can compute a_j as follows: $l(a) \equiv a_j \pmod{q_j^{r_j}}$ iff $l(ag^{-a_j}) \equiv 0 \pmod{q_j^{r_j}}$, which happens if and only if ag^{-a_j} is a $q_j^{r_j}$-th power. Thus, to check if a_j is equal to s, we compute $b = (g^{-1})^s$ and check if there exists a t such that $t^{q_j^{r_j}} = ab \in \mathbb{Z}_p^*$. Both the exponents s and $q_j^{r_j}$ are less than d, as desired. Now a general instance of the POW

predicate $a^i \equiv b \pmod{p}$ reduces to finding the CRR representation of $l(a)$ and $l(b)$ and verifying that for each j, $ia_j \equiv b_j \pmod{q_j^{r_j}}$, all of which lies in FO.

We are now left with queries $a^i \equiv b \pmod{p}$, where a, b, and p each have x bits and i is less than d. (Recall that d is $O(2^{x^{1-\epsilon}})$ and that both x and d are functions of n). We can resolve this query by multiplying a with itself i times and dividing the result by p. In particular, this is an instance of ITERATED MULTIPLICATION (where the input and output consist of $O(x2^{x^{1-\epsilon}}) = 2^{O(x^{1-\epsilon})}$ bits, followed by DIVISION where the input size is also $2^{O(x^{1-\epsilon})}$. Now the results of Section 3 show that these problems are in FOM + POW($x^{1-\epsilon}$), as desired. ■

A constant number of applications of this lemma now reduces our original problem to the following simple lemma:

Lemma 5.3 *The problem* POW($\sqrt{\log n}$) *is in* FOM.

Proof. (of Lemma 5.3) To calculate $a^i \bmod p$, we guess the numbers $a^{\lfloor i/2^j \rfloor} \bmod p$, for $0 \leq j < \sqrt{\log n}$. Specifically, we assert the existence of a bit string of $\log n$ bits that encodes these $\sqrt{\log n}$ numbers $a^{\lfloor i/2^{\sqrt{\log n}} \rfloor}$, ..., $a^{\lfloor i/2 \rfloor} \bmod p$, $a^i \bmod p$, each of which has $\sqrt{\log n}$ bits. Given this bit string, we can easily check that each number $a^{\lfloor i/2^{j-1} \rfloor} \bmod p$ is the square, mod p, of the number $a^{\lfloor i/2^j \rfloor} \bmod p$, or that square multiplied by a. The last of these numbers is equal to $a^i \bmod p$. ■

Theorem 5.4 *Assuming Hypothesis 5.1,* POW = POW($\log n$) *is in* FOM. *Thus in this case* FOM + POW = FOM *and all the problems from Section 4 are in* FOM.

6 Small space-bounded complexity classes

For many people working in computational complexity theory, space-bounded computation only "begins" with logarithmic space. To be sure, there is a large literature dealing with space bounds between $\log \log n$ and $\log n$. (For example, see [20] for a perspective on the sequence of difficult papers leading up to a separation of the bounded-alternation hierarchy for sublogarithmic-space-bounded machines.) Nonetheless, this work relies on the automata-theoretic limitations of the small-space-bounded machine. For instance, if $s(n) = o(\log n)$ is a fully-space-constructible function, then there is a constant k such that, for infinitely many n, $s(n) < k$. Thus every infinite unary language in dspace($o(\log n)$) has an infinite regular subset. This provides easy proofs of lower bounds for the space complexity of many languages, such as the proof in [15] that the set $\{0^n : n \text{ is prime}\}$ cannot be accepted in space $o(\log n)$.

However, it is still an open question whether the set of (binary encodings of) primes can be accepted in space $o(\log n)$. How can this be? Surely the binary encoding of a set cannot be easier than the unary encoding of the same set!

Let us see why this is still an open question. Usually a lower bound on the complexity of the binary encoding of a set follows from a bound on the complexity of the unary encoding, using a standard *translation lemma*, such as:

Lemma 6.1 (Traditional Translation Lemma) *If $s(\log n) = \Omega(\log \log n)$ is fully space-constructible, then the first statement below implies the second:*

- $A \in \text{dspace}(s(n))$.
- $\text{un}(A) \in \text{dspace}(\log n + s(\log n))$.

The converse also holds, if $s(\log n) = \Omega(\log n)$.

Note in particular that this translation lemma does not allow one to derive any lower bound on the space complexity of A, assuming only a logarithmic lower bound on the space complexity of $\text{un}(A)$. As an example to see that this is unavoidable, consider the regular set $A = 10^*$. Arguing as in [15] it is easy to see that $\text{un}(A) = \{0^{2^k} : k \in \mathbb{N}\}$ is not in $\text{dspace}(o(\log n))$ (since it has no infinite regular subset).

There is another reasonable way to define space complexity classes. Let $\text{DSPACE}(s(n))$ be the class of languages accepted by Turing machines that begin their computation with a worktape consisting of $s(n)$ cells (delimited by endmarkers), as opposed to the more common complexity classes $\text{dspace}(s(n))$ where the worktape is initially blank, and the machine must use its own computational power to make sure that it respects the space bound of $s(n)$. Viewed another way, $\text{DSPACE}(s(n))$ is simply $\text{dspace}(s(n))$ augmented by a small amount of "advice", allowing the machine to compute the space bound. (This model was defined under the name "DEMON-SPACE" by Hartmanis and Ranjan [16]. See also Szepietowski's book [27] on sublogarithmic space.)

$\text{DSPACE}(s(n))$ seems at first glance to share many of the properties of $\text{dspace}(s(n))$. In particular, it is still relatively straightforward to show that there are natural problems, such as the set of palindromes, that are not in $\text{DSPACE}(o(\log n))$. (This follows from a simple crossing-sequence and Kolmogorov-complexity argument [16].)

The main contribution of this section is an easy argument, showing that the efficient division algorithm of [11] provides a new translation lemma.

Lemma 6.2 New translation lemma *Let $s(n) = \Omega(\log n)$ be fully space-constructible. Then the following are equivalent:*

- $A \in \text{dspace}(s(n))$

- $\text{un}(A) \in \text{DSPACE}(\log\log n + s(\log n))$.

Proof. (of Lemma 6.2) For the forward direction, it is sufficient to present a small-space algorithm for $\text{un}(A)$.

Note that $\log\log n$ space can hold the binary representation of a short prime p. Thus on input 0^n, a $\text{DSPACE}(\log\log n)$ machine can compute the pieces of the Chinese Remainder Representation of n.

Thus, by [11], in space $\log(|n|) = \log\log n$ we can compute the bits of the binary representation of n. Thus, on input 0^n a Turing machine can simulate a $s(|n|)$-space-bounded computation (of a machine having input n) in space $s(\log n)$.

For the converse, given a Turing machine accepting $\text{un}(A)$ in space $\log\log(x)+ s(\log x)$ on input 0^x, we want to use $\log(|x|) + s(|x|) = O(s(|x|))$ space to determine if $x \in A$. We provide merely a sketch here.

The most naïve approach to carry out this simulation will not work, since we do not have enough space to record the location of the input head in a simulated computation on 0^x, and thus we cannot perform a step-by-step simulation. However, we do have enough space to carry out a simulation until either

(a) the input head returns to an endmarker without repeating a worktape configuration, or

(b) some worktape configuration is repeated.

In case (a), a step-by-step simulation is sufficient. In case (b), we can determine the period of the loop, and (doing some simple arithmetic) we can determine the state the machine will be in when it encounters the other end marker.

Thus in either case, the simulation can proceed. ■

Corollary 6.3 *Let $\mathcal{C}$ be any complexity class. In order to show $\mathcal{C}$ is not contained in* L*, it suffices to present a set $A \in \mathcal{C}$ such that* $\text{un}(A) \notin \text{DSPACE}(\log\log n)$.

In some ways, $\text{DSPACE}(\log\log n)$ is a more natural class than $\text{dspace}(\log\log n)$, in the sense that this class is related to a natural class of branching programs, whereas no similar characterization is known for $\text{dspace}(\log\log n)$. The following result makes this more precise.

For this extended abstract, we assume the reader is familiar with basic definitions regarding branching programs.

Theorem 6.4 *A is accepted by* DLOGTIME*-uniform branching programs of polynomial size and width $O(\log^{O(1)} n)$ if and only if A is FO-reducible to a language accepted by an oblivious* $\text{DSPACE}(\log\log n)$ *machine.*

Proof. (of Theorem 6.4) First, consider a language accepted by an oblivious machine M with a worktape of size $O(\log \log n)$. By definition of "oblivious", the input location scanned by M at time t can be computed in FO. Thus it is an easy matter to construct a branching program with a node for each worktape configuration on each level, with edges simulating M's transition function. The resulting branching program will be FO-uniform, and this can be transformed into an equivalent DLOGTIME-uniform branching program by standard techniques.

Conversely, let A be accepted by a DLOGTIME-uniform leveled branching program of width $\log^{O(1)} n$. It is easy to show that there is a FO reduction that, given an input string x, produces a sequence of the form

$$\#\#f_1\#f_2\#\ldots\#f_t\#\#$$

where t is the number of columns, and each f_i is a function $f_i : \{1,\ldots w\} \to \{1,\ldots w\}$, where $w = \log^{O(1)} n$ is the width of the branching program, with the property that $f_i(j) = j'$ iff the branching program, when in vertex j in column i, moves to vertex j' in column $i+1$ when querying the specified bit of x.

Note that an input x is accepted by M if and only if $f_t(f_{t-1}(\ldots(f_1(1))\ldots))$ is an accepting state of M. We encode each function f in the sequence as a list

$$(1, f(1))(2, f(2))\ldots(w, f(w)).$$

Note that there is an oblivious machine with space bound $O(\log \log n)$ that takes such a sequence of functions as input and computes the composition. ■

Essentially equivalent observations appear elsewhere. For instance, it is shown in [12] that leveled branching programs of width $O(2^{s(n)})$ correspond to non-uniform finite automata with space bound $s(n)$.

7 Acknowledgments

All three authors gratefully acknowledge the support of the NSF Computer and Computation Theory program. Much of this work was carried out during the March 2000 McGill Invitational Workshop on Complexity Theory – the authors thank the organizer Denis Thérien and all the other participants. We also thank Dieter van Melkebeek, Samir Datta, Michal Koucký, Rüdiger Reischuk, and Sambuddha Roy for helpful discussions.

Additional work on this project was carried out during the Park City Mathematics Institute's summer program in July and August 2000, supported by the Clay Mathematics Institute. The authors thank PCMI, CMI, Alexis Maciel, and the students in the PCMI undergraduate program where this material was presented.

References

[1] M. Agrawal, E. Allender, and S. Datta. On TC^0, AC^0, and Arithmetic Circuits. Journal of Computer and System Sciences **60**:395–421, 2000.

[2] E. Allender and D. A. M. Barrington. Uniform circuits for division: Consequences and problems. *Electronic Colloquium on Computational Complexity* **7:065** (2000). Preliminary version of this paper.

[3] R. C. Baker and G. Harman. Shifted primes without large prime factors. *Acta Arithmetica*, **83**.4:331–361, 1998

[4] D. A. M. Barrington and N. Immerman. Time, hardware, and uniformity. In *Complexity Theory Retrospective II*, L. A. Hemaspaandra and A. L. Selman, eds., Springer-Verlag, 1997, pp. 1–22.

[5] D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, **41**:274–306, 1990.

[6] D. A. M. Barrington, P. Kadau, K.-J. Lange, and P. McKenzie. On the complexity of some problems on groups given as multiplication tables. *Proc. 15th IEEE Conference on Computational Complexity*, 2000, pp. 62–69.

[7] P. Beame, S. Cook and J. Hoover. Log depth circuits for division and related problems. *SIAM J. Comput.*, **15**:994–1003, 1986.

[8] E. R. Canfield, Paul Erd¨os, and Carl Pomerance. On a problem of Oppenheim concerning "factorisatio numerorum". *Journal of Number Theory*, **17**:1–28, 1983.

[9] A. K. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM J. Comput.*, **13**:423–439, 1984.

[10] A. Chiu. Complexity of parallel arithmetic using the Chinese Remainder representation. Master's thesis, U. Wisconsin-Milwaukee, 1995. G. Davida, supervisor.

[11] A. Chiu, G. Davida, and B. Litow. NC^1 Division. Preliminary version. Available from the website `http://www.cs.jcu.edu.au/~bruce` as `/papers/crr00_3.ps.gz`.

[12] C. Damm and M. Holzer. Inductive Counting for Width-Restricted Branching Programs. *Information and Computation* **130**:91–99, 1996.

[13] G. I. Davida and B. Litow. Fast parallel arithmetic via modular representation. *SIAM J. Comput.*, **20**:756–765, 1991.

[14] Paul F. Dietz, Ioan I. Macarie, and Joel I. Seiferas. Bits and relative order from residues, space efficiently. *Information Processing Letters*, **50**:123–127, 1994.

[15] J. Hartmanis and L. Berman. On tape bounds for single letter alphabet language processing. *Theoretical Computer Science* **3**:213–224, 1976.

[16] J. Hartmanis and D. Ranjan. Space bounded computations: Review and new speculation. In *MFCS '89: Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 379, Springer-Verlag, 1989, pp. 49–66.

[17] W. Hesse. Division is in Uniform TC^0. In *ICALP 2001: Twenty-Eighth International Colloquium on Automata, Languages and Programming* (July 2001), to appear.

[18] W. Hesse, E. Allender, and D. A. M. Barrington. Fully Uniform Threshold Circuits for Division and Related Problems. In preparation. To be submitted to *Journal of Computer and System Sciences* (special issue for this conference).

[19] N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1999.

[20] M. Li´skiewicz and R. Reischuk. Computing with sublogarithmic space. In *Complexity Theory Retrospective II*, L. A. Hemaspaandra and A. L. Selman, eds., Springer-Verlag, 1997, pp. 197–224.

[21] I. Macarie. Space-efficient deterministic simulation of probabilistic automata. *SIAM J. Comp.* **27**:448-465, 1998.

[22] V.A. Nepomnjaščiĭ. Rudimentary predicates and Turing calculations. *Soviet Math. Dokl.* **11**:1462–1465, 1970.

[23] J. Reif and S. Tate. On threshold circuits and polynomial computation. *SIAM J. Comput.*, **21**:896–908, 1992.

[24] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 21:365–383, 1981.

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

[26] A. Szepietowski. If deterministic and nondeterministic space complexities are equal for $\log\log n$, then they are also equal for $\log n$. *Theoretical Computer Science*, **74**:115–119, 1990.

[27] A. Szepietowski. *Turing Machines with Sublogarithmic Space*. Lecture Notes in Computer Science 843, Springer-Verlag, 1994.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'