Division is in Uniform TC^0

William Hesse*

Department of Computer Science
University of Massachusetts
Amherst, MA 01002
FAX: (001) 413 545-1249
whesse@cs.umass.edu

Abstract. Integer division has been known since 1986 [4, 13, 12] to be in slightly non-uniform TC^0, i.e., computable by polynomial-size, constant depth threshold circuits. This has been perhaps the outstanding natural problem known to be in a standard circuit complexity class, but not known to be in its uniform version. We show that indeed division is in uniform TC^0. A key step of our proof is the discovery of a first-order formula expressing exponentiation modulo any number of polynomial size.

1 Introduction

The exact complexity of integer division has been harder to pin down than the complexities of addition, subtraction, and multiplication. In 1986, Beame, Cook, and Hoover showed that iterated multiplication, and thus division, could be performed by Boolean circuits of logarithmic depth (NC^1 circuits) [4]. In 1987, Reif showed that these circuits could be implemented as constant depth circuits containing threshold gates (TC^0 circuits) [12, 13]. Since then, the remaining issue has been the complexity of constructing these circuits. Division is the only prominent natural problem whose computation uses non-uniform circuits, circuits which require a non-trivial amount of computation for their construction.

The division problem discussed in this paper is the division of two n-bit integers, given in binary, yielding their integer quotient, also in binary. A related problem is the multiplication of n n-bit integers, computing their product as a binary integer. These problems are easily reduced to each other, so that a uniform circuit for one yields a uniform circuit for the other.

In this paper, we construct uniform constant depth circuits for division and iterated multiplication. We work within the framework of descriptive complexity, and show that there is a first-order formula using majority quantifiers that expresses division. This implies that there is an FO-uniform TC^0 circuit performing division [3]. First-order (FO) uniformity, equivalent to DLOGTIME uniformity, is the strongest uniformity requirement found to be generally applicable. A key step focuses on the one step of the TC^0 division computation not previously

* Supported by NSF grant CCR-9877078.

known to be expressible by a first order formula with majority quantifiers (an FO(M) formula). This is the problem of finding powers in the finite field $\mathbf{Z}_p$, the integers modulo a prime, where p has $O(\log n)$ bits. We show that there is a first-order formula without majority quantifiers computing powers in $\mathbf{Z}_p$. Thus this subproblem is in FO, and can be computed with uniform AC^0 circuits.

2 Definitions

We will express division as a predicate DIVISION(X, Y, i) which is true if and only if bit i of $\lfloor X/Y \rfloor$ is 1. We denote numbers with n or $n^{O(1)}$ bits by capital letters, and numbers with $O(\log n)$ bits by lowercase letters. We also refer to numbers with $O(\log n)$ bits as small, and those with $n^{O(1)}$ bits as large. We will always note the size of numbers with $(\log n)^{O(1)}$ bits explicitly. The iterated multiplication problem will be written as the predicate IMULT$(A_1, \ldots, A_n, i)$ which is true if bit i of $\prod_{j=1}^{n} A_j$ is 1; i ranges from 0 to n^2, and so has $2 \log n$ bits.

Though the size of the input to division is $2n+\log n$ and the input to iterated multiplication has size $n^2+2 \log n$, we will consider the input size, for all problems in this paper, to be n, as the circuit complexity classes and descriptive complexity classes we consider are closed under a polynomial change in the input size.

In this paper we produce simple logical formulas expressing these predicates. A problem is in the complexity class FO (first order) if the predicate corresponding to the decision problem can be expressed by a first order formula interpreted over a finite universe, the set of natural numbers $0, \ldots, n$. The inputs to the problem are encoded as relations over the universe, and are available to be used in the formula. The fixed numeric relations $<$ and BIT are also available[1]. For example, the n bits of the input X to DIVISION are represented by the values of a unary predicate $X()$ on the elements of the universe: $X(1), X(2), \ldots, X(n)$. An n^2 bit input can be represented by a binary predicate, so the inputs $A_1, \ldots, A_n$ to IMULT are represented as a binary predicate A. Short inputs to a problem, like i, the index of the result bit queried, may be represented by a constant in the range $0, \ldots, n$, which can also be regardedas a free variable. Since an FO or FO(M) formula over the universe $1, \ldots, n^k$ can be simulated by an equivalent formula over the universe $1, \ldots, n$, DIVISION and IMULT with inputs X, Y, and A_i having n^k bits, encoded by k-ary relations over $0, \ldots n$, are in the same descriptive complexity class as DIVISION and IMULT with n-bit inputs.

DIVISION and IMULT are provably not in FO, as parity is FO reducible to them, and parity is not in FO [8, 9]. They will be shown to be in the class FO(M), problems described by first-order logic plus the majority quantifier. The majority quantifier (Mx) can appear anywhere that an $(\exists x)$ or a $(\forall x)$ can appear. The formula $(Mx)\varphi(x)$ is true iff $\varphi(j)$ is true for more than half the values $0 \leq j \leq n$.

[1] Following [10], we consider FO to include ordering and BIT. The BIT predicate allows us to look at the bits of numbers. BIT(i, x) is true if bit i of the number x written in binary is 1. This is equivalent to having addition and multiplication on numbers between 0 and n.

These quantifiers let us count the number of 1 bits in a string of length n; the counting quantifiers $(\exists! i\ x)$ are definable in terms of (Mx). These quantifiers are analogous to gates with n inputs that output 1 iff at least i of their inputs are 1, called threshold gates. We see next how an FO(M) formula is equivalent to a circuit containing threshold gates.

A TC^0 circuit is a constant-depth, polynomial-size circuit with AND, OR, NOT, and threshold gates with arbitrary fanin. If the type of each gate and the connections between gates can be computed by a deterministic logtime Turing machine, or equivalently by an FO formula, then the circuit is FO-uniform. The equivalence of FO-uniform TC^0 circuits and FO(M) formulas is shown by Barrington, Immerman, and Straubing in [3]. The Boolean function computed by an FO-uniform TC^0 circuit can be computed by an FO(M) formula, a first order formula using majority quantifiers, ordering, and BIT. The converse also holds; any FO(M) formula can be turned into a uniform TC^0 circuit. Here and throughout the paper, uniform will mean FO-uniform.

TC^0 is contained in the circuit complexity class NC^1, which contains all problems decided by Boolean circuits containing NOT gates, and AND and OR gates with two inputs, with $n^{O(1)}$ gates, n inputs, and depth $O(\log n)$. TC^0 contains the class AC^0 of constant depth polynomial size circuits without threshold gates. FO-uniform AC^0 circuits are equivalent to FO formulas with only existential and universal quantifiers, no majority quantifiers [3].

3 Previous work

As stated in the introduction, Beame, Cook, and Hoover, et. al. gave NC^1 circuits deciding DIVISION and IMULT in 1986 [4]. They also gave a polynomial time algorithm for constructing the n'th circuit. Reif showed how to convert these to constant-depth threshold circuits a year later [13, 12]. Immerman and Landau then observed that the construction was logspace uniform given the product of the first n^3 primes, implying that the full construction was TC^1 uniform [11].

These circuits were based on finding the remainders of the inputs on division by a set of small primes. The value of a number modulo a set of primes uniquely determines its value modulo the product of those primes. This is referred to as the Chinese remainder representation (CRR). The circuits work by converting the inputs to CRR, computing iterated products in that representation, and converting the output to binary. In the later 1990s, Chiu, Davida, and Litow devised new ways of computing in CRR that reduced the complexity of converting from CRR into binary [5, 6]. These steps allowed them to construct logspace-uniform and NC^1-uniform TC^0 circuits for division and iterated multiplication.

Allender and Barrington reinterpreted those results in the framework of descriptive complexity, and showed that the only difficulty in expressing iterated multiplication and division in FO(M) was the difficulty of raising numbers to a power modulo a small prime [2]. The current paper completes this effort by showing that this power predicate lies in FO. As division is complete for FO(M)

via FO Turing reductions, it is unlikely that the complexity of division can be further reduced.

4 Division reduces to POW

The key problem examined by this paper is POW, the predicate expressing exponentiation modulo a prime. For a small prime p, and small arguments a, r, and b,

$$\mathrm{POW}(a, r, b, p) \iff a^r \equiv b \pmod{p} \ .$$

To be exact, we have a family of problems $\mathrm{POW}_{k \log n}(a, r, b, p)$ for $k = 1, 2, \ldots$, where the inputs to $\mathrm{POW}_{k \log n}$ have size $k \log n$. An input with $k \log n$ bits can be represented by a k-tuple of variables taking values in $0, \ldots, n$. Thus $\mathrm{POW}_{k \log n}$ is a $4k$-ary numeric relation. Though the inputs have $O(\log n)$ bits, we consider the descriptive complexity of this problem as if it had input size n. We ask whether this predicate can be represented by FO or FO(M) formulas over the universe $0, \ldots, n$.

Allender, Barrington, and the author showed that DIVISION and IMULT are in FO(M) if and only if POW is in FO(M) [2]. They did this by showing that DIVISION and IMULT are FO-Turing reducible to POW. A version of this proof, with additional simplifications, is in the full version of this paper. The predicate POW is used to convert inputs from binary to CRR, and to find discrete logarithms in the multiplicative group $\mathbf{Z}_p^*$ of integers mod p for primes p in the CRR basis.

FO-Turing reducibility in descriptive complexity classes is formally defined using generalized quantifiers in [10]. In the case of an FO-Turing reduction to POW, we shall not use this full formal definition, but a simpler characterization of FO-Turing reducibility to a relation. In the case of $\mathrm{POW}(a, r, b, p)$, which could be considered as a primitive numeric relation of arity $4k$ (if the inputs have $k \log n$ bits), we can express FO(M) Turing reducibility to POW by simply saying a predicate φ is FO(M) Turing reducible to POW if and only if there is an FO(M) formula with numeric relations $\leq$, BIT, and POW that expresses φ. This is equivalent to saying $\varphi \in \mathrm{FO}(\mathrm{M}, \mathrm{POW})$. Clearly, if POW is in FO(M), then FO(M,POW)= FO(M). We replace all uses of POW in a formula φ with the equivalent FO(M) formula. This is all we shall need to use about the reduction from DIVISION and IMULT to POW.

5 POW is FO-Turing reducible to $\mathrm{IMULT}_{O(\log n)}$ and $\mathrm{DIVISION}_{O((\log n)^2)}$

We now show that we can produce an FO formula deciding POW, provided that we allow the formula to use the results of certain smaller IMULT and DIVISION problems. These problems will have inputs constructed by FO formulas from the inputs to POW, or from the outputs of other small IMULT and DIVISION problems. This can be characterized as an FO-Turing reduction from POW to

these smaller versions of IMULT and DIVISION. Later we will show that these smaller versions of IMULT and DIVISION are in FO(M), and then show that they are in FO.

The scaled versions of IMULT and DIVISION have $(\log n)^{O(1)}$-bit inputs. We still consider them as having input size n, however, so we shall define them as

$$\text{IMULT}_{(\log n)^k}(A_1, \ldots, A_n, j) =$$

$$\text{IMULT}(A_1, \ldots, A_n, j) \ \wedge \ (\forall i) A_i < 2^{(\log n)^k} \ \wedge \ (\forall i > (\log n)^k) A_i = 1$$

$$\text{DIVISION}_{(\log n)^k}(X, Y, i) = \text{DIVISION}(X, Y, i) \wedge X < 2^{(\log n)^k} \wedge Y < 2^{(\log n)^k} \ .$$

Thus we only have to give correct answers to the problems when the inputs are small. An FO Turing reduction to these problems is more complicated than an FO(M) Turing reduction to POW because the inputs to these problems have $\omega(\log n)$ bits and so must be given as relations, not as first-order variables. We shall only point out where these problems are used in our first-order expression for POW, and state the implications if we have FO or FO(M) expressions for them.

To show that POW is in FO, we will prove a more general lemma about finding powers in groups. This is interesting in its own right, and necessary for the extension to finding powers modulo prime power moduli. We consider a group to be given in FO if group elements are labeled by elements of the universe and the product operation is given by an FO formula. Note that the identity element and inverse operation can be defined in FO from the product operation. We can also continue to use arithmetic operations on the universe, considered as the numbers $0, \ldots, n$.

Lemma 1. *Finding small powers in any group of order n is FO Turing-reducible to finding the product of $\log n$ elements.*

Proof. Suppose we want to find a^r, where a is an element of a group of order n. We will compute a set of elements $a_1, \ldots, a_k$ and exponents $u, u_1, \ldots, u_k$ such that

$$a^r = a^u a_1^{u_1} \cdots a_k^{u_k}$$

and $u_i < 2 \log n$, $u < 2(\log n)^2$.

Step 1. We choose a set of $k = o(\log n)$ primes $d_1, \ldots, d_k$, such that $d_i < 2 \log n$ and d_i is relatively prime to n, for all i. We choose them such that $n < D = d_1 d_2 \cdots d_k < n^2$. We can do this with a first order formula by choosing the first $D > n$ such that D is square-free, D and n are relatively prime, and all prime factors of D are less than $2 \log n$. We can decide, given D, whether a number is one of our d_i or not. To compute the number k from D, and to find our list d_i as a relation between i and d_i, requires, for each prime $p_0 < 2 \log n$, counting the number of primes p dividing D which are less than p_0. We can do this using the BITSUM predicate, which counts the number of one bits in a $\log n$ bit number: BITSUM(x, y) is true if the binary representation of x contains y ones. This is shown to be in FO in [3].

Step 2. We calculate $a_i = a^{\lfloor n/d_i \rfloor}$ as follows:

First we calculate $n_i = n \bmod d_i$. Compute a^{-1} using the inverse operation. We find a^{-n_i} by multiplying n_i copies of a^{-1} together. This is one place where our Turing reduction to multiplication of $\log n$ group elements is used.

We can find $a^{\lfloor n/d_i \rfloor}$ by observing that

$$(a^{\lfloor n/d_i \rfloor})^{d_i} = a^{\lfloor n/d_i \rfloor d_i} = a^{n-(n \bmod d_i)} = a^{n-n_i} = a^{-n_i} \ .$$

Observe that there is exactly one group element x such that $x^{d_i} = a^{-n_i}$: Let d_i^{-1} be the multiplicative inverse to d_i mod n, i.e. that $d_i d_i^{-1} = mn + 1$ for some m. Then

$$x = x^{mn+1} = (x^{d_i})^{d_i^{-1}} = (a^{-n_i})^{d_i^{-1}} \ .$$

Thus we can find $a_i = a^{\lfloor n/d_i \rfloor}$ as the value of x in the expression

$$(\exists x)\ x^{d_i} = a^{-n_i}$$

We compute x^{d_i} using multiplication of $\log n$ elements. We could not compute $a^{\lfloor n/d_i \rfloor}$ directly as $(a^{-n_i})^{d_i^{-1}}$ since d_i^{-1} is not necessarily $O(\log n)$.

Step 3. Now we find the exponents $u, u_1, \ldots, u_k$ such that $a^u a_1^{u_1} \cdots a_k^{u_k} = a^r$.

Since $a_i = a^{\lfloor n/d_i \rfloor}$,

$$a_1^{u_1} \cdots a_k^{u_k} = a^{\left(\sum_{i=1}^k u_i \lfloor n/d_i \rfloor\right)} \ ,$$

$$\text{and since } a^r = a^u a_1^{u_1} \cdots a_k^{u_k} = a^{\left(u + \sum_{i=1}^k u_i \lfloor n/d_i \rfloor\right)} \ ,$$

$$u \equiv r - \sum_{i=1}^k u_i \lfloor \frac{n}{d_i} \rfloor \pmod{n} \ .$$

Thus, to make the final correction term a^u computable, we must make u as small as possible, and so we want to make $\sum_{i=1}^k u_i \lfloor n/d_i \rfloor \bmod n$ as close to r as possible. To approximate r as a linear combination of $\lfloor n/d_i \rfloor$, we use the Chinese remainder theorem.

Compute $f = \lfloor rD/n \rfloor$. This step requires r to have $O(\log n)$ bits. Using the Chinese remainder theorem, if we let $D_i = D/d_i$, and let $u_i = fD_i^{-1} \bmod d_i$, then

$$\sum_{i=1}^k u_i D_i \equiv f \pmod{D} \ . \text{ Let } m \text{ be s.t. } \sum_{i=1}^k u_i D_i = f + mD \ .$$

We can calculate u_i in FO, since we can guess the possibilities for D_i^{-1} in FO. Calculating u from the u_i involves a sum of k small numbers, which, since $k < \log n$, is in FO. This, again, uses the fact that BITSUM is in FO.

We now show that $u < (\log n)^2$. We calculate the difference between r and $\sum u_i \lfloor n/d_i \rfloor$:

$$\begin{aligned}
\sum_{i=1}^{k} u_i \lfloor \frac{n}{d_i} \rfloor &= \sum_{i=1}^{k} \frac{u_i n}{d_i} - \sum_{i=1}^{k} (\frac{u_i n}{d_i} - u_i \lfloor \frac{n}{d_i} \rfloor) \\
&= \frac{n}{D} \sum_{i=1}^{k} u_i D_i - \sum_{i=1}^{k} u_i (\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor) \\
&= \frac{n}{D}(f + mD) - \sum_{i=1}^{k} u_i (\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor) \\
&= \frac{n}{D} \lfloor \frac{rD}{n} \rfloor + nm - \sum_{i=1}^{k} u_i (\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor) \\
&= r - \frac{n}{D}(\frac{rD}{n} - \lfloor \frac{rD}{n} \rfloor) + nm - \sum_{i=1}^{k} u_i (\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor) \ , \text{ so}
\end{aligned}$$

$$u = r - \sum_{i=1}^{k} u_i \lfloor \frac{n}{d_i} \rfloor \bmod n = \frac{n}{D}(\frac{rD}{n} - \lfloor \frac{rD}{n} \rfloor) + \sum_{i=1}^{k} u_i (\frac{n}{d_i} - \lfloor \frac{n}{d_i} \rfloor) \ .$$

The quantity $y - \lfloor y \rfloor$ is always between 0 and 1, and since $n/D < 1$, $u_i < 2 \log n$, and $k < \log n$, we see that $u < 2(\log n)^2 + 1$. Thus we can calculate a^u using two rounds of multiplying $\log n$ group elements.

Thus we have described group elements a_i and numbers u, u_i such that $a^u a_1^{u_1} \cdots a_k^{u_k} = a^r$ and the computation of $a^u a_1^{u_1} \cdots a_k^{u_k}$ is FO Turing reducible to the product of $\log n$ group elements. □

Because FO is closed under polynomial change in input size, and the product of $\log(n^k) = k \log n$ group elements is FO reducible to the product of $\log n$ group elements, we have

Corollary 1. *Finding powers in any group of order n^k is FO Turing-reducible to finding the product of $\log n$ elements.*

Representing a group of order n^k means representing elements as k-tuples of universe elements, and representing the product operation in FO.

We now apply this to the integers modulo p, where $p = O(n^k)$ is a prime. The multiplicative group $\mathbf{Z}_p^*$ contains the $p-1$ integers $1, \ldots, p-1$, and multiplication in this group is clearly first-order definable from multiplication and addition on $0, \ldots, n$. If a in $\mathrm{POW}(a, r, b, p)$ is zero, then we only need to check that b is zero. Otherwise, we find a^r in the multiplicative group Z_p^*. The product of $\log n$ group elements can be computed with $\mathrm{IMULT}_{k \log n}$ and $\mathrm{DIVISION}_{k \log^2 n}$, so we have the main lemma of this section:

Lemma 2. *POW is FO-Turing reducible to $IMULT_{O(\log n)}$ and $DIVISION_{O((\log n)^2)}$.*

6 DIVISION$_{(\log n)^{O(1)}}$ and IMULT$_{(\log n)^{O(1)}}$ are in FO(M)

Since our end result is that DIVISION and IMULT are in FO(M), it should be no surprise that the logarithmically smaller versions DIVISION$_{(\log n)^{O(1)}}$ and IMULT$_{(\log n)^{O(1)}}$ are in FO(M). We will prove that these smaller versions are in FO(M) by reducing them to POW$_{O(\log\log n)}$, and showing that POW$_{O(\log\log n)}$ is in FO.

Just as we have introduced scaled versions of IMULT and DIVISION, we use a scaled version of POW:

$$\text{POW}_{k\log\log n}(a,r,b,p) = \text{POW}(a,r,b,p) \ \wedge \ a,r,b,p < 2^{k\log\log n}$$

The FO(M) Turing reduction of IMULT$_{n^{O(1)}}$ to POW = POW$_{O(\log n)}$ shown by Allender et. al [2] scales to become an FO(M) Turing reduction of IMULT$_{(\log n)^{O(1)}}$ to POW$_{O(\log\log n)}$. This can be seen as follows: consider the FO(M) reduction on the problem with $(\log n)^{O(1)}$ input size, which is a Turing reduction using FO(M) formulas over the universe with $(\log n)^{O(1)}$ elements to the correspondingly scaled version of POW, POW$_{O(\log\log n)}$. But any FO(M) formula over the smaller universe can be simulated by an FO(M) formula over the larger universe, so this is an FO(M) reduction from IMULT$_{(\log n)^{O(1)}}$ to POW$_{O(\log\log n)}$.

Showing that POW$_{O(\log\log n)}$ is in FO can be done directly. Suppose the modulus p, the exponent r, and the base a all have fewer than $k\log\log n$ bits. The numbers $a_i = a^{\lfloor r/2^i \rfloor} \bmod p$, with i ranging from 0 to $k\log\log n$ can be guessed simultaneously, since there are $k\log\log n$ of them, each with $k\log\log n$ bits. An existential choice of a number x from 0 to $n-1$ can be thought of as a non-deterministic simultaneous guess of $\log n$ bits, so we can certainly simultaneously guess $(k\log\log n)^2$ bits. There is exactly one choice of the numbers $a_1,\ldots,a_{k\log\log n}$ such that the following conditions hold:

$$a_{k\log\log n} = a^{\lfloor r/2^{k\log\log n}\rfloor} = a^0 = 1 \text{ and } (\forall i)\ a_i \equiv a_{i+1}^2 a^{r_i} \pmod p \ ,$$

where r_i is bit i of r.

Extracting the numbers a_i out of our $\log n$ bit choice x and checking that they meet the above conditions can be done with an FO formula. Extracting a_0 gives us $a^r \bmod p$.

Thus we have concluded that POW$_{O(\log\log n)}$ is in FO. Since we have an FO(M) Turing reduction from IMULT$_{(\log n)^{O(1)}}$ and DIVISION$_{(\log n)^{O(1)}}$ to POW$_{O(\log\log n)}$, we can conclude

Theorem 1. *IMULT$_{(\log n)^{O(1)}}$ and DIVISION$_{(\log n)^{O(1)}}$ are in FO(M).*

7 DIVISION and IMULT are in FO(M)

Since we have an FO Turing reduction from POW to IMULT$_{O(\log n)}$ and DIVISION$_{O(\log^2 n)}$, we can conclude that we have an FO(M) formula for POW. Finally, using the FO(M) Turing reduction from IMULT and DIVISION to POW, we arrive at our main result.

Theorem 2. *Iterated multiplication of n n-bit numbers and division of 2 n-bit numbers is in FO(M).*

By the equivalence of FO(M) to FO-uniform TC^0, we have

Corollary 2. *Iterated multiplication of n n-bit numbers and division of 2 n-bit numbers is in FO-uniform* TC^0.

As both of these classes are closed under polynomial change in the input size, these results also hold for inputs with $n^{O(1)}$ bits.

8 POW is in FO

An additional result of the theorem that IMULT and DIVISION are in FO(M), is that $\mathrm{IMULT}_{(\log n)^{O(1)}}$ and $\mathrm{DIVISION}_{(\log n)^{O(1)}}$ are in FO. This is because any FO(M) formula over a universe $0, \ldots, \log n$ has an equivalent FO formula over the universe $0, \ldots, n$.

The fact that FO is closed under the introduction of counting quantifiers with polylogarithmic bounds is established in [1, 7]. Since $\mathrm{IMULT}_{(\log n)^{O(1)}}$ is equivalent to IMULT with input size $(\log n)^{O(1)}$, it is expressed by an FO(M) formula over $0, \ldots, \log n$. Therefore, $\mathrm{IMULT}_{(\log n)^{O(1)}}$ is expressed by an FO formula, and similarly $\mathrm{DIVISION}_{(\log n)^{O(1)}}$ is in FO, and we have

Theorem 3. *$\mathrm{IMULT}_{(\log n)^{O(1)}}$ and $\mathrm{DIVISION}_{(\log n)^{O(1)}}$ are in FO.*

This theorem gives us a tight bound on the size of cases of IMULT that are in FO. Since we know that $\mathrm{PARITY}_{f(n)}$ is in FO iff $f(n) = (\log n)^{O(1)}$, from Håstad [9], and PARITY is easily FO many-one reducible to multiplication of two numbers, which is FO many-one reducible to IMULT of the same size, we can conclude that $\mathrm{IMULT}_{f(n)}$ is in FO iff $f(n) = (\log n)^{O(1)}$.

Since our proof that POW was in FO(M) included an FO Turing reduction from POW to $\mathrm{DIVISION}_{O((\log n)^2)}$ and $\mathrm{IMULT}_{O(\log n)}$, and we now have FO formulas expressing $\mathrm{DIVISION}_{O((\log n)^2)}$ and $\mathrm{IMULT}_{O(\log n)}$, we can now conclude that POW is in FO. Since the restriction that the inputs to POW have $O(\log n)$ bits is equivalent to requiring that the inputs be in the range $0, \ldots, n$, we have our second main result.

Theorem 4. *The predicate POW(a, r, b, p) which is true iff $a^r \equiv b \pmod{p}$, with p prime, can be expressed by an FO formula over the universe $0, \ldots, n$, if $a, r, b, p \leq n$.*

This result can be extended to exponentiation modulo any small number n, not just modulo a prime. We can see that the equation

$$a^r \equiv b \pmod{n}$$

is true if and only if it is true modulo all the prime power factors of n:

$$a^r \equiv b \pmod{p^i} \; \forall p^i | n \ .$$

We can show that for a relatively prime to p^i, a is in the group $\mathbf{Z}^*_{p^i}$, and the above proof can be applied. If p divides a, then if $r > \log n$, $a^r \equiv 0 \pmod{p^i}$. If $r \leq \log n$, then $\text{IMULT}_{(\log n)^{O(1)}}$ can be applied. Since the prime power factors of a small number n can be found in FO, we have

Corollary 3. *The predicate $a^r \equiv b \pmod{n}$, with the inputs written in unary, is in FO.*

Finally, note that the property that any predicate expressible in FO over the universe $0, \ldots, n^k$ is expressible in FO over $0, \ldots, n$ lets us conclude that the predicate $a^r \equiv b(\bmod n)$ is in FO if the inputs have $O(\log n)$ bits, but not that it is in FO with inputs of $(\log n)^{O(1)}$ bits. This is different from the results we have for $\text{IMULT}_{(\log n)^{O(1)}}$ and $\text{DIVISION}_{(\log n)^{O(1)}}$.

9 Conclusions

Our main theorem states that division and iterated multiplication are in fully uniform TC^0. This is significant on its own and also because it eliminates the most important example of a problem known to be in a circuit complexity class, but not known to be in the corresponding uniform complexity class.

We also proved that exponentiation modulo a number is in FO when the inputs have $O(\log n)$ bits. This result was quite unexpected, since the problem was previously not even known to be in FO(M). It remains unknown if exponentiation modulo a number with $(\log n)^{O(1)}$ bits is in FO, or even in FO(M).

Finally, we have found a tight bound on the size of division and iterated multiplication problems that are in FO. We now know that these problems are in FO if and only if their inputs have $(\log n)^{O(1)}$ bits. Instances of the problems with larger inputs are known not to be in FO.

9.1 Acknowledgments

These results were found while working on [2] with Eric Allender and David Mix Barrington, who generously urged me to publish them separately.

References

1. M. Ajtai and M. Ben-Or. A theorem on probabilistic constant depth computations. In *ACM Symposium on Theory of Computing (STOC '84)*, pages 471–474, 1984. ACM Press.
2. E. Allender, D. A. Mix Barrington, and W. Hesse. Uniform circuits for division: Consequences and problems. To appear in *Proceedings of the 16th Annual IEEE Conference on Computational Complexity (CCC-2001)*, 2001. IEEE Computer Society.

3. D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.
4. P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15(4):994–1003, 1986.
5. A. Chiu, G. Davida, and B. Litow. NC^1 division. online at http://www.cs.jcu.edu.au/∼bruce/papers/crr00_3.ps.gz.
6. G. I. Davida and B. Litow. Fast Parallel Arithmetic via Modular Representation. *SIAM Journal of Computing*, 20(4):756–765, 1991.
7. R. Fagin, M. M. Klawe, N. J. Pippenger, and L. Stockmeyer. Bounded-depth, polynomial-size circuits for symmetric functions. *Theoretical Computer Science*, 36(2-3):239–250, 1985.
8. M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. In *22nd Annual Symposium on Foundations of Computer Science*, 260–270, 1981. IEEE.
9. J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, 6–20, 1986.
10. N. Immerman. *Descriptive Complexity*. Springer-Verlag, New York, 1999.
11. N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116(1):103–116, 1995.
12. J. H. Reif. On threshold circuits and polynomial computation. In *Proceedings, Structure in Complexity Theory, Second Annual Conference*, pages 118–123, IEEE Computer Society Press.
13. J. H. Reif and S. R. Tate. On threshold circuits and polynomial computation. *SIAM Journal on Computing*, 21(5):896–908, 1992.