IAS/PCMI Summer Session 2000
Clay Mathematics Undergraduate Program
Advanced Course on Computational Complexity

Lecture 8: Chinese Remainder Representation

David Mix Barrington and Alexis Maciel

July 26, 2000

1. Overview

We are now ready to begin our study of integer division and related problems by developing a method for representing long numbers by their remainders modulo a number of short primes. Eventually (in Lecture 10) we will present the new theorem of Chiu, Davida, and Litow, that DIVISION is in L, and consider the prospects for placing it in the same class FOM that contains iterated addition and multiplication. But in this lecture we need to begin developing background. Specifically:

- We define Chinese Remainder Representation (CRR) and consider the difficulty of converting numbers from binary to CRR and vice versa.
- We recapitulate the proof of the Chinese Remainder Theorem, and define the *rank* of a number in CRR, a property that will turn out to be useful in a number of ways.
- We define a *table of logarithms* for each prime modulus that will allow us to convert iterated multiplication to iterated addition in a variety of circumstances. We also consider the difficulty of constructing such a table, introducing the new complexity class FOLL.
- Finally, we show how to use the log table to carry out iterated multiplication modulo a short prime, and hence iterated multiplication with input and output in CRR, in FOM once the log table is available. The complexity class "FOM with powering modulo short primes", or FOMP, will turn out to be our best upper bound for DIVISION and related problems.

2. The Fundamentals of CRR

Let $m_1, \ldots, m_k$ be distinct prime numbers and let M be their product. By our analysis of the density of primes, we can let both k and each of the m_i's be short numbers ($O(\log n)$ bits) and still have M be a number with an arbitrarily large polynomial number of bits. We will choose such numbers to make M larger than any of the inputs and outputs to our various problems.

(In addition, we assume that the set of m_i's is easily computable — for example it could consist of all the primes in a certain range of numbers. Once we pick the range, k can be determined in FOM by counting the primes.)

If X is any number less than M, the *Chinese Remainder Representation* or CRR of X is the sequence $\langle x_1, \ldots, x_k \rangle$ where for each i, X is congruent to x_i modulo m_i. The Chinese Remainder Theorem, which we will prove below, says that every number from 0 to $M-1$ is uniquely represented by such a sequence. (Thus we will sometimes refer to "CRR with modulus M" or "CRR_M" when more than one modulus is in use.)

Many of the results we will prove about CRR also hold if the m_i's are *powers of distinct primes* rather than just distinct primes. This will occasionally simplify the statements of some results at the cost sometimes of making the proofs more complicated. We will try to take note at each point where prime powers could be used. For technical reasons we will want our primary modulus M to be the product of distinct primes, but other moduli may not be.

Our basic strategy will be to store numbers in CRR rather than in binary, and work with them in that form. This has the great advantage that we can work with each different prime in parallel, allowing us to use short-number operations to solve problems about long numbers. A potential difficulty with this method, however, is that some problems that were easy in binary notation apparently become more difficult. For example, given CRR representations of two numbers X and Y, how can we determine which is larger?

Another difficuly is that our problems demand input and output in binary rather than in CRR, so we need to be able to convert numbers from one representation to the other. What complexity resources are needed to do this? Can we do it in $\mathrm{FO} + \mathrm{BIT}$, or even in FOM?

Converting a binary number X to CRR involves dividing a long number by a short number m and getting the remainder. Since we are given X in the form $\sum_i x_i 2^i$, it suffices to find the remainder of each number $x_i 2^i$ modulo m, add the results, and reduce again mod m. (The last step, dividing one short number by another, is in $\mathrm{FO} + \mathrm{BIT}$.) We've seen that ITERATED ADDITION can be done in FOM, but what about calculating 2^i modulo m? This is an example of *powering modulo a short number*, a problem that will concern us greatly below.

It turns out that the crucial addition to FOM for our purposes will be exactly a numerical predicate for powering modulo a short number: specifically, the predicate "$a^i \equiv b$ modulo m", where a, b, i, and m are all short numbers. We define FOMP to be the class of problems definable by first-order formulas with majority quantifiers, BIT, and this new numerical predicate. We will eventually prove:

Theorem 1 *(Chiu-Davida-Litow, Allender-Barrington) DIVISION, POWERING, and ITERATED MULTIPLICATION are all in* FOMP*, hence also in* L*-uniform* TC^0, L*-uniform* NC^1*, and* L *itself.*

Thus the very computing power we need to *begin* using CRR, by converting a number into it, will turn out to be all we need to exploit it to the fullest.

What about converting a CRR number to binary? The right way to do this is not at all clear. Powering modulo short numbers does not solve the problem for us as before, but it can help in the right circumstances. Note that raising 2 to a power modulo each m_i is sufficient to get the representation of 2^k in CRR, for any short number k. If we can solve the division problem *in CRR*, then the following identity will allow us to compute the k'th bit of X in binary:

$$\mathrm{BIT}(k, X) = \lfloor X/2^k \rfloor - 2\lfloor X/2^{k+1} \rfloor$$

This gives the bit in question in CRR, but it is easy to test the result to see whether it is the CRR for 0 or for 1. We have thus shown that we can convert to and from CRR in FOMP if we can solve the possibly easier problem of DIVISION with both input and output in CRR.

3. The Chinese Remainder Theorem

It is clear that if two numbers are congruent modulo M, they are also congruent modulo each of our k primes, since each of these primes divides M. One way to prove the Chinese Remainder Theorem is to prove the converse, that two numbers with the same CRR must be congruent modulo M, by providing an algorithm to compute a number from its CRR. (Since this mapping is the inverse of the obvious mapping from numbers to CRR representations, and the set of distinct representations has size M, the mapping is one-to-one.) We'll now go through this computation to see what it can do for us.

For each modulus m_i, let C_i be the product of all the m_j with $j \neq i$. (That is, $C_i = M/m_i$.) Let h_i be the multiplicative inverse of C_i modulo m_i, and let $D_i = h_i C_i$. D_i has the property that it is congrent to 1 modulo m_i and congruent to 0 modulo each of the other m_j's. Thus the number $\sum_i x_i D_i$ has the same remainders modulo each of the primes as does X. If we subtract off a multiple rM of M from $\sum_i x_i D_i$ to get a number in the range from 0 to $M-1$, that number must be X (again, because this mapping from CRR's to numbers must be one-to-one as it has an inverse).

This number r is called the *rank* of X with respect to CRR_M. We will have to do some work to compute it, but we can at least see now that it is a short number. It is the floor of the sum $\sum_i x_i D_i/M$ or $\sum_i x_i h_i/m_i$ using the definition of D_i. Since each rational number h_i/m_i is between zero and one, we know that r is bounded above by the $\sum_i x_i$ and thus by $\sum_i m_i$, a sum of a short number of short numbers and therefore short.

Note that nothing in this section changes if the m_i's are powers of distinct primes rather than primes. Since each C_i, for example, is a product of numbers each relatively prime to m_i, the inverse h_i exists and we can compute D_i.

4. Constructing and Using Log Tables

In our new class FOMP we have available to us the answers to all questions of the form "is $a^i = b$ modulo m?" where all these numbers are short. (For convenience we will use "$=$" to denote congruence rather than the more precise "$\equiv$".) In effect we have a table of these answers available, and with ordinary first-order quantifiers we can use this table to find the *discrete logarithm*, modulo m, of any number b. If m is prime, the numbers modulo m form a finite field, and thus the nonzero numbers form a *cyclic group* under multiplication. (This is a standard fact in algebra, which you may prove in the exercises.) Thus there are elements of this field, called *generators*, such that any nonzero a is congruent to g^i modulo m for some i. We fix a particular generator for each m (such as the smallest one) and define this i to be the discrete logarithm of a modulo m. In the next section we'll use discrete logarithms to carry out iterated multiplication in FOMP.

The status of discrete logarithms modulo a prime power is similar, but somewhat more complicated. First, of course, there are numbers modulo p^e that are divisible by p and thus not relatively prime to p^e. We can deal with these by writing them as ap^d and then finding the discrete logarithm modulo p^{e-d}. Also, the integers modulo a prime power no longer form a field but only a ring, so we can no longer use our former proof that the multiplicative group is cyclic. The multiplicative group $Z^*_{p^e}$, consisting of the integers modulo p^e that are relatively prime to p^e, actually *is* cyclic with one exception (where $p = 2$ and $e > 2$). With reference to powering, we can still identify generators and compute logarithms in FOMP as before, though in the $p = 2$ case we need to redefine "logarithm" slightly. It turns out that the multiplicative group in this case has two generators (-1 and 5) and since one of them (-1) has order two we can essentially use logarithms here as well.

First, though, how difficult is it to compute powers modulo a short number, and thus get discrete logarithms? For definiteness, assume we are looking for the value of a^i modulo m. We first note that it is easy to get this in L, by simply calculating $a^0, a^1, \ldots, a^i$ in turn. We need to remember the current value of i and the current and previous values of a^i along with a, four short numbers in all. Since FOM is a subclass of L, we see that FOMP is a subclass of L as well.

Can we do better by computing a^i in another way? The technique of *repeated squaring* is promising. We note that $a^i = b$ is true if there exists a c such that either $a^{i-1} = c$ and $ca = b$ or $a^{i/2} = c$ and $cc = b$ (all modulo m). We can reduce questions about a^i by a first-order formula to questions about a^j for $j < i$, and use recursion to solve the latter problems. It is easy to see that this recursion terminates in $O(\log i) = O(\log n)$ steps. If we use this to construct a circuit, we use depth $O(1)$ to simulate each level and thus depth $O(\log n)$ in all. Size is polynomial (each gate represents a predicate of the form $a^i = b$ and there are only polynomially many choices of a, i, and b) so we have put this problem in AC^1. Unfortunately AC^1 contains L, so we are no better off than before. Using repeated squaring in sequential computation saves time, but still uses $O(\log n)$ space.

We can do better in terms of depth, however, if we use the fact that we have available not only the powers of a but the powers of all numbers modulo m. Note that if i is a perfect square j^2, for example, $a^i = b$ is true iff there exists some number c such that $a^j = c$ and $c^j = b$. We can add this method to our recursion, and note that it now closes in $O(\log \log n)$ rather than $O(\log n)$ phases.

We've now solved the powering problem (modulo a short number) with circuits of depth $O(\log \log n)$ and polynomial size. If we had the necessary terminology (you are invited to look it up for the exercises) we could describe the powering predicate in terms of families of first-order formulas where the number of quantifiers increases with n, in this case to $O(\log \log n)$. The resulting class is equivalent to *uniform* circuits of depth $O(\log \log n)$ and polynomial size, a class called FOLL (for "first-order log-log").

We'll show in Basic Lecture 10 that even *non-uniform* circuits of this size and depth cannot do the parity language, and thus cannot do all the problems in L. This suggests, but doesn't prove, that FOMP does not require the full power of L and is probably a strictly smaller class. (But for all we can prove, FOM and L might be equal, making the distinction moot.)

5. Iterated Products in the Class FOMP

Now that we have discrete logarithms, let's use them. Suppose we have numbers $y_1, \ldots, y_n$ and want the product $y_1 \cdots y_n$ modulo m where m is prime. We simply find the discrete logarithm z_i for each number y_i, find the sum s of the z_i's modulo m, and look up the answer g^s modulo m in our table of powers. Since the laws of logarithms work perfectly well modulo m, this is valid. (We have to say what to do if one of the y_i's is zero, of course, which you may do in the exercises.) We have shown that the problem of iterated multiplication modulo a short *prime* is in FOMP.

Iterated products modulo a prime power are similar though more complicated. Given numbers modulo p^e, we must find and add together the number of factors of p in the product. If this is e the product is zero, otherwise if it is $d < e$ we treat the p-free parts of each number as being modulo p^{e-d}, and multiply these using discrete logarithms modulo p^{e-d}. If $p = 2$, we may need to add together both the number of -1 factors and the discrete logs with respect to the "generator" 5 to do the multiplication. But given this, we can perform iterated multiplication modulo any prime power in FOMP.

And now, of course, we can see the virtues of CRR. If we have long numbers $X_1, \ldots, X_n$ each in CRR with respect to M, in FOMP we can take the n remainders for each individual prime modulus m and calculate their iterated product modulo m. The n answers are exactly the CRR values for the product of the X_i's. We have shown that iterated product is in FOMP *if* the input and output are both in CRR.

6. Exercises

1. Consider CRR with moduli 3, 5, 7, and 11 so that $M = 1155$. Find the CRR of 206 and the number whose CRR is $\langle 2, 1, 4, 8 \rangle$. Find the rank of each of these numbers with respect to CRR with these moduli.

2. Explain why the multiplicative group of a finite field is cyclic. (You may use the fact that a polynomial of degree d over a finite field has at most d roots, and consider the factorization of $x^e - 1$, where e is the order of the multiplicative group and 1 is the identity of the field. Look at the possible orders of other elements if there is no element of order e.)

3. Show that 2 is a generator for the nonzero numbers modulo 37.

4. Verify (perhaps with the aid of a group theory or number theory book) the claims made above about the multiplicative groups of the integers modulo prime powers.

5. Explain why the standard simulations for poly-size circuits of depth $O(\log \log n)$ and unbounded fan-in (the class FOLL) do not put this class within either NC^1 or L.

6. Show that in any graph of n vertices, we can test in FOLL whether there is a path of length $f(n)$ from s to t if the function $f(n)$ is polylog. Show that if the graph has only poly-log many vertices, its reachability problem is in FO + BIT.

7. We don't have a discrete logarithm for 0 even if the modulus is prime. How does this affect our algorithm for iterated multiplication modulo a short prime m?

8. Look up "iterated quantifier blocks" and "inductive definition" in Immerman's book and explain why FOLL can be more precisely defined as the languages described by first-order formulas with $O(\log \log n)$ quantifiers. Explain why the recursive definition of "$a^i = b$ modulo m" above puts this predicate in FOLL.

9. We have shown that the iterated product problem modulo any short prime power is in FOMP. Show that the iterated product modulo *any short number* is in FOMP, even if the inputs and outputs are given in binary.