On the Complexity of Some Problems on Groups Input as Multiplication Tables

David Mix Barrington[*] Peter Kadau[†]
Klaus-Jörn Lange[‡] Pierre McKenzie[§]

January 20, 2000

Abstract

The Cayley Group Membership problem (CGM) is to input a groupoid (binary algebra) G given as a multiplication table, a subset X of G, and an element t of G, and to determine whether t can be expressed as a product of elements of X. For general groupoids CGM is P-complete, and for associative algebras (semigroups) it is NL-complete.

Here we investigate CGM for particular classes of groups. The problem for general groups is in Sym-L, but any kind of hardness result seems difficult because it would require constructing the entire multiplication table of a group.

We introduce the complexity class FOLL $=$ FO$(\log\log n)$ of problems solvable by uniform circuit families of polynomial size, unbounded fan-in, and depth O(log log n). Since parity is not in FOLL, no problem in FOLL can be complete for any class containing parity, such as NC^1, L, or SL. But FOLL is not known to be contained even in SL.

We show that CGM for cyclic groups is in FOLL $\cap$ L and that CGM for abelian groups is in FOLL. In a partial extension of the method to solvable groups, we prove that CGM for groups of constant solvability class is in FOLL and that CGM for nilpotent groups can be solved by circuits of polynomial size and depth $O((\log\log n)^2)$. (Thus the problem for nilpotent groups is provably not complete for any class containing parity.)

We also consider the problem of testing for various properties of a group input as a table: we prove that cyclicity and nilpotency can each be tested in FOLL $\cap$ L.

Finally, we examine some possible implications of our results for the complexity of iterated multiplication of integers.

[*]U. Mass. at Amherst, barring@cs.umass.edu
[†]Universität Tübingen, kadau@informatik.uni-tuebingen.de
[‡]Universität Tübingen, lange@informatik.uni-tuebingen.de
[§]Université de Montréal and (on sabbatical) in Tübingen, mckenzie@iro.umontreal.ca

1 Introduction

1.1 The Problem

One of the most natural computational problems on groups is that of *membership*: input some formal representation of a group and an element (of some supergroup), and determine whether the group contains that element. Naturally, the complexity of this problem varies dramatically with the format chosen to input the group.

If the group is given by generators and relations, the problem has long been known to be undecidable (see [22, P. 298]), and the case of matrix groups led to the introduction of Arthur-Merlin games [2, 1]. When the group is given as a set of generating permutations (and the target is given as a permutation) we get the *permutation group membership problem*, the source of deep and beautiful work culminating in an NC algorithm for the problem that relies on the classification of finite simple groups [23, 12, 7].

Here we consider a simple and very low-complexity variant of the problem, the *Cayley Group Membership* problem, or CGM:

> *Given:* group G prescribed by multiplication table, $X \subseteq G$, $t \in G$.
> *Question:* does t belong to the subgroup $\langle X \rangle$ generated by X?

CGM is a special case of the more general problem where the multiplication table need not be that of a group. With an *arbitrary* table (for a *groupoid*, which need not even be associative) the membership problem was shown to be complete for polynomial time by Jones and Laaser [17]. If the table obeys the associative law (a *semigroup*), Jones *et al.* showed the membership problem to be NL-complete [18]. (This completeness result holds even if the semigroups are group-free, making groups the natural domain to explore further.) Barrington and McKenzie first investigated the group version of the problem in 1991 [5], noting that CGM reduces to the undirected graph accessibility problem (UGAP) and is thus in the class SL (the NC^1closure of UGAP). They suggested that CGM might be complete for SL or at least hard for L, or deterministic logspace. Since then, there has been no progress on resolving the precise complexity of CGM. It is suggestive that some very simple problems involving permutation representations of groups are L-complete, such as that of determining whether a given permutation on n elements is an n-cycle [11]. (This shows that under the right circumstances even *cyclic* groups can be L-hard to analyze.) But so far all attempts to exploit these problems to get a hardness proof have failed.

In this paper we present and exploit a simple new recursive strategy for parallel computation of powers in a group given by a multiplication table. We show that for certain groups (abelian, nilpotent, and some solvable groups) the CGM problem can be solved very quickly in parallel – *more* quickly than any parallel algorithm can solve even the parity problem. Using known lower bounds for the parity problem [25] we then prove, using no unproven assumptions, that CGM for these classes of groups is *not* hard for any standard class containing parity, such as SL, L, NC^1, or even ACC^0. We conjecture that the problem for solvable groups, and for that matter probably general groups, also fails to be hard for these classes.

We also consider the complexity of *testing* a group multiplication table to see whether the group is cyclic or nilpotent. Both these problems can be solved in $O(\log \log n)$ parallel time, meaning that neither of them can be hard for any class containing parity.

Finally, if p is a prime number the group of integers modulo p is one where the binary multiplication operation is easy to calculate but calculation of powers is an interesting problem (as is its inverse, the discrete logarithm problem). We examine the implication of our new strategy for the calculation of powers and discrete logs in parallel, and the related problems of iterated multiplication and division for arbitrary integers.

1.2 The Model of Computation

We will be measuring the complexity of problems primarily as the *parallel time* required to solve them with a polynomial number of processors. This is equivalent [16, 15, 6, 4] to (a) the depth of a boolean circuit of polynomial size and unbounded fan-in *solving* the problem, or (b) the quantifier depth of a formula in first-order logic *expressing* the problem.

Consider the multiplication table of a group (or semigroup) to be input as an n by n array of numbers, each in the range from 1 to n. The number in the (i, j) position of this array represents the element obtained by multiplying i times j, where elements of the group are labeled 1 through n. A particular table represents a group if and only if it satisfies three familiar properties: it must be *associative*, have an *identity*, and have an *inverse* for each element. Each of these properties may be expressed by a first-order formula where variables range over elements in the group and there are atomic formulas to represent equality of elements and the group operation:

$$\forall x : \forall y : \forall z : ((x \cdot y) \cdot z) = (x \cdot (y \cdot z))$$

$$\exists x : \forall y : (x \cdot y) = (y \cdot x) = y$$

$$\forall x : \exists y : \forall z : ((x \cdot y) \cdot z) = z$$

As shown in [6], a property can be expressed by a fixed first-order formula of this kind if and only if it can be tested by a logtime-uniform circuit family of constant depth, unbounded fan-in and polynomial size, *i.e.*, if and only if it is in the circuit complexity class (logtime-uniform) AC^0. Thus the three group axioms, as well as other properties such as commutativity, are testable for a multiplication table in AC^0.

Immerman [16, 15, 4] has developed a formalism whereby *families* of first-order formulas may be defined to express properties outside of AC^0. Given the necessary conventions for reusing variables, we can define a formula with a *quantifier block* in front which is syntactically iterated $d(n)$ times on input of size n and then followed by some fixed formula. If the formula contains a constant number of variables each ranging from 1 to n, it can express a property if and only if it can be tested by a (logtime uniform) circuit family of polynomial size, unbounded fan-in, and depth $O(d(n))$. Such circuit families are equivalent in power to CRCW PRAMs with a polynomial number of processors and depth $O(d(n))$. For example, quantifier depth $O(\log n)$ corresponds to the circuit complexity class AC^1, which contains such other classes as L, SL, NL, and LOGCFL. In this paper we will be particularly interested in the class FOLL, which consists of the languages expressible with quantifier depth $O(\log\log n)$ — this class contains AC^0, is contained within AC^1, but is not known to be comparable to L, SL, or NL.

A convenient way to prove the existence of a formula with an iterated quantifier block is to give a *recursive definition* of a property and show that it closes within a certain number of steps (see [16, 15] for more detail on this). For example, consider the property "$a^i = b$" in a group given as a multiplication table, where i is at most n, the order of the group. The familiar recursive definition of this predicate using repeated squaring says that "$a^i = b$" is true if either:

$$(i = 0) \wedge (b = e),$$

$$(i = i) \wedge (a = b),$$

$$\exists c : (ac = b) \wedge (a^{i-1} = c),$$

or

$$\exists c : (cc = b) \wedge (a^{i/2} = c).$$

Since this definition is guaranteed to close in $O(\log n)$ iterations, it allows us to construct a circuit of depth $O(\log n)$ computing this predicate for all values of a, b, and i, where a node on level t is set to true if and only if one of the above conditions holds for the predicates on level $t-1$. (In effect, we can think of the circuit as filling in entries of a dynamic programming table for all the values of a, b, and i.)

2 Preliminaries

2.1 Notation

All groups in this paper will be considered to be input as their *Cayley representation*, consisting of an n by n table of numbers, each number in the range from 1 through n. (Thus the total size of the input is $n^2 \log n$ bits, so all our resource bounds defined in terms of n apply equally well to the true input size.) In logical terms, our formulas will have an atomic formula "$ab = c$" whose boolean value indicates whether the product of a and b in the group is c. (For convenience, we will represent the group operation by concatenation as is usual in group theory.)

2.2 FOLL

For the purposes of this extended abstract, we rely on the intuitive presentation of descriptive complexity given in Subsection 1.2 above. The formal details can be found in [16, 15, 4].

In this paper we show a number of problems to be in the complexity class $\mathrm{FO}(\log\log n)$, and take the liberty of giving this class the name FOLL. Of course, to add to the proliferation of named complexity classes requires some justification. We believe that interest in FOLL is warranted for the following reasons:

- the present work describes some non-trivial upper bounds showing natural problems to be in it,
- membership of a problem in FOLL precludes its being hard for the more conventional small classes such as TC^0 (this is justified below),
- the relationship of FOLL to the standard hierarchy of parallel complexity classes (e.g., [10, 4]) is unclear, so that the inclusion chain

  $$\mathrm{FO(constant)} = \mathrm{AC}^0 \subset \mathrm{FOLL} \subseteq \mathrm{FO}(\log n) = \mathrm{AC}^1,$$

  which follows from [24], deserves closer scrutiny.

Indeed, AC^1 seems to be the smallest conventional complexity class known to contain FOLL. In particular, none of the classes in the inclusion chain

$$\mathrm{ACC}^0 \subseteq \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq (\mathrm{LOGCFL} \cup \mathrm{DET}) \tag{1}$$

are known to contain FOLL.

On the other hand, by the Smolensky bound [25], the PARITY function is not computable by unbounded fan-in circuits of depth d and size $o(2^{(1/2)n^{1/2d}})$, even if the circuit family is non-uniform and we allow mod-3 gates along with ANDs and ORs. It follows that polynomial-size circuits require depth $\Omega(\log n/(\log\log n))$ to compute PARITY, and thus that the class $\mathrm{FO}(t(n))$ does not contain PARITY unless $t(n) = \Omega(\log n/(\log\log n))$. It follows that

Proposition 2.1 *FOLL contains no well-known class above AC^0, and in particular, no class from the inclusion chain (1). The same is true of* $\mathrm{FO}(t(n))$ *for* $t(n) = o(\log n/(\log\log n))$.

3 Results

3.1 A double-barrelled recursive strategy

Our central observation is that the inductive definition of the *power predicate* "$a = b^i$" in terms of repeated squaring can be improved so that it closes in only $O(\log\log n)$ steps. We begin by giving the definition of this predicate in terms of first-order formulas involving (1) values of the power predicate with exponents smaller than i, (2) the atomic predicate "$ab = c$" for multiplication in the group, and (3) the operations of addition and multiplication on numbers in the range from 0 to n:

- $a = b^0$ iff $a = e$
- $a = b^1$ iff $a = b$
- $a = b^i$ if $\exists j, k, c, d$ such that $i = j + k$, $c = b^j$, $d = b^k$, $a = cd$
- $a = b^i$ if $\exists j, k, c$ such that $i = jk$, $c = b^j$, $a = c^k$.

We must now argue that this first-order inductive definition closes in $O(\log\log n)$ steps if we require $i \leq n$. This is because (a) if i is not a power of two, we can choose j and k such that $i = j + k$ and j and k each have at most half as many ones in their binary expansions as does i (rounding up), and (b) if i is a power of two, we can write $i = jk$ where j and k are each powers of two and have logs at most half that of i (rounding up). Since the logs of both $\log i$ and of the number of ones in i's binary expansion are bounded by $\log\log n$, we reach the base case in at most $2\log\log n$ phases.

Using [16, 15, 4] we conclude:

Proposition 3.1 *For all a and b in the group and all $i \leq n$, the predicate "$a = b^i$" can be calculated in FOLL.*

Note that this predicate is also easily computable in L, deterministic logspace, by successively computing b, b^2, b^3, $\ldots$ up to b^i and comparing the result to a.

3.2 Testing group properties

Cayley groups are a special case of permutation groups because the regular representation of a group G, i.e. as a permutation group on itself, is explicitly given by the Cayley table of G. Although permutation groups have been studied extensively, none of the upper bounds known for permutation group problems are strong enough for our purposes. In fact, even the divide-and-conquer group decomposition strategy which was ultimately used in solving the permutation group membership and many related problems [7] in NC is not applicable in our setting because even a mere $\log n$ iterative steps is out of reach.

As in most other work on the complexity of group problems, it is sensible to investigate groups by starting with the "easiest" groups, and working our way up to the "more complicated". Standard results [22] about the structure of groups provide the basis for this approach. The most basic kind of group is *cyclic*, generated by a single element. The *abelian* groups are direct products of cyclic groups, and the *nilpotent* and *solvable* groups are successively more complicated compositions of abelian groups. In most settings solvable groups are substantially easier to work with than general, non-solvable groups. (For example, the iterated multiplication problem in a fixed solvable group is in ACC^0, while iterated multiplication in a fixed non-solvable group is complete [3] for NC^1.)

Even cyclic groups can pose non-trivial computational problems. For example, it is L-complete to determine whether a permutation of n elements is a single cycle, even if it is known to be the product of at most two disjoint cycles [11]. Testing whether a Cayley group is cyclic is clearly doable in L, and the logspace algorithm looks like an inherently sequential process. With our new parallel algorithm, however, we can show that this problem is *not* hard for logspace:

Theorem 3.2 *Testing a Cayley group for cyclicity is in FOLL $\cap$ L.*

Proof. By Proposition 3.1, in FOLL $\cap$ L we can compute the predicate $a = b^i$ for any elements a and b and any number $i \leq n$. A group is cyclic, by definition, iff $\exists g \forall a \exists i (a = g^i)$. This is a first-order formula using an atomic predicate in FOLL $\cap$ L. Since FOLL and L are each closed under FO(or AC^0) reductions, we are done. ■

Corollary 3.3 *Testing a Cayley group, or even an* arbitrary *multiplication table, for being that of a cyclic group is hard for none of the classes occurring in the inclusion chain (1).*

Abelianness of a group is a first-order property as the commutativity axiom is first-order. The next property to consider is that of being a nilpotent group. Here again, the power predicate is enough for us:

Theorem 3.4 *A group given by Cayley table can be tested for nilpotency in FOLL$\cap$L.*

Proof. For any prime number p, we define a p-element to be an element whose order is a power of p. To determine the p-elements in a group, we need to compute the orders of elements and do some simple number theory on numbers smaller than n. The former is first-order definable from the power predicate:

$$o(a) = m \leftrightarrow (a^m = e) \wedge \forall j (0 < j < m \rightarrow a^j \neq e)$$

and is thus in FOLL $\cap$ L. The latter is in FO: since $(\log n)$-bit numbers can be added and multiplied in FO, the predicates "p is a prime smaller than n" and "$m \leq n$ and no prime other than p divides m" can be expressed in FO. Our FOLL $\cap$ L nilpotency test then follows from a group-theoretic lemma: ■

Lemma 3.5 *A finite group G is nilpotent iff, for each prime p dividing $|G|$, the set of p-elements in G is a group.*

Proof. See the Appendix. ■

The power predicate does not seem to be of much help in testing solvability of a Cayley group. The best solvability test known to the authors is in LOGCFL, using a stack to look for a non-trivial iterated commutator whose existence shows the group to be non-solvable.

3.3 The CGM Problem

Recall that the *Cayley group membership* problem (CGM) is to input a Cayley group G, a set of elements X, and a target element t, and to determine whether t is in the subgroup of G generated by X. We now consider the subcases of CGM for each of our classes of groups. Note that even the simplest case of CGM(cyclic) was conjectured to be logspace-hard in [5].

Theorem 3.6 *CGM(cyclic groups) is in FOLL $\cap$ L and (since it is in FOLL) is hard for none of the classes occurring in the inclusion chain (1).*

Proof. Using the FOLL $\cap$ L power predicate, we can find a generator g for the group G and compute, for each $h \in X$, the unique integer $i < |G|$ such that $g^i = h$. The least common multiple l of these discrete logarithms can be expressed in FO, and $t \in \langle X \rangle$ iff the discrete logarithm of t divides l. ■

We now turn to the case of abelian groups, but we first formulate a useful structural property:

Definition 3.7 *A family of groups has the $f(n)$ power basis property if any set X of generators for a group G of order n from the family has the property that every element of G is a product of at most $f(n)$ powers of elements of X.*

Example. The family of all groups trivially has the n power basis property because any element a group G of order n generated by a set $X = \{g_1, \ldots, g_k\}$ is of course expressible as $\Pi_{j=1}^{m} g_{i_j}^{e_j}$ for some $m \leq n$.

Lemma 3.8 *Abelian groups have the* $\log n$ *power basis property.*

Proof. Let $B = \{b_1, \ldots, b_k\}$ be an arbitrary set of generators for G. Let G_i be the subgroup of G generated by $\{b_1, \ldots, b_i\}$. Let X be the set of elements b_i such that G_i is different from G_{i-1}. (Note that since each distinct group G_i has at least twice as many elements as its predecessor, the size of X is at most $\log n$. This argument works for an arbitrary group, showing that any generating set has a subset of size at most $\log n$ that also generates.)

Renumber the elements of X as $\{x_1, \ldots, x_j\}$ and let H_i be the subgroup spanned by $\{x_1, \ldots, x_i\}$. It is clear by induction that each group H_i is equal to the corresponding group G_ℓ generated by the elements $\{b_1, \ldots, b_\ell = x_i\}$, so that the entire set X generates G. Thus any element of G can be written as a product of elements of X, and since G is abelian this product can be rearranged into a product $x_1^{e_1} \ldots x_j^{e_j}$, a product of length at most $\log n$ as desired. ■

Theorem 3.9 *CGM(abelian) is in FOLL.*

Proof. Consider the following sets of elements defined inductively:

$$\begin{aligned} Y_0 &= \{x^i : x \in X, i \leq n\} \\ Y_{i+1} &= \{yz : y, z \in Y_i\} \end{aligned}$$

The set Y_i contains all products of at most 2^i powers of elements of X. Since G, and the subgroup of G generated by X, have the $\log n$ power basis property, $Y_{\log\log n}$ contains all products of at most $\log n$ powers of elements of the power basis and hence contains the entire subgroup generated by X.

Membership in the set Y_0 is first-order definable from the power predicate and X and is thus in FOLL. Since Y_{i+1} is first-order definable from Y_i, $Y_{\log\log n}$ is definable from Y_0, and thus from X in FOLL. We need merely check whether the target element t is in $Y_{\log\log n}$. ■

Note that this second FOLL algorithm, unlike the first one for the power predicate, cannot (as far as we know) be simulated in L. We thus do not know whether CGM(abelian) is in L.

Nilpotent groups are parametrized by *nilpotency class.* In the full paper we will define this notion and prove that nilpotent groups of class d have the $O(\log^d n)$ power basis property. Hence CGM for nilpotent groups of class d is in $\mathrm{FO}(d \log\log n)$, following the proof of Theorem 3.9. In particular, CGM for nilpotent groups of class $O(1)$ is in FOLL. These results, however, are subsumed by the following treatment of solvable groups (as a group's solvability class is no greater than its nilpotency class).

Solvability of a finite group can be defined in terms of the *derived series*, where group $G^{(0)}$ is G and group $G^{(i+1)}$ is defined to be the *commutator subgroup* of $G^{(i)}$, the group generated by all elements of $G^{(i)}$ of the form $x^{-1}y^{-1}xy$. A group is solvable iff $G^{(d)}$ is the trivial group for some d, and is defined to be *solvable of class d* if d is the smallest number making this true. We show that our approach to deciding CGM extends to some solvable groups:

Theorem 3.10 *CGM(groups of solvability class $O(1)$) is in FOLL. CGM for groups of solvability class $d(n)$ is in* $\mathrm{FO}(d(n)\log\log n)$.

Proof. See the Appendix. ∎

A general solvable group of order n may have solvability class $\Omega(\log n)$, so this result does not improve on the $\mathrm{FO}(\log n)$ existing algorithm for CGM for any group. But *nilpotent* groups cannot have such a large solvability class, according to the following result (proved by Thérien [26], possibly well-known):

Theorem 3.11 *[26] Any nilpotent group of order n has solvability class $O(\log\log n)$.*

Corollary 3.12 *CGM(nilpotent) is in* $\mathrm{FO}((\log\log n)^2)$*, and hence is not hard for any class in the inclusion chain (1).*

We conjecture that these results are not the best possible, and that in fact CGM(solvable) may be in FOLL. The question also remains whether the CGM problem for general groups is easier than the best known upper bound of SL. Might this problem be in L, or solvable in $o(\log n)$ parallel time? Any hardness results for these problems would appear to require the reduction to construct a multiplication table for a group — this has so far been an insurmountable problem even for a reduction from PARITY.

3.4 Arithmetic of small numbers

One of the outstanding open problems in complexity theory is the complexity of integer division and iterated multiplication of integers. These and related problems were shown to be in P-uniform TC^0 by Beame, Cook, and Hoover [8], but are not known to be in L-uniform TC^0 or in L itself. The non-uniformity of this algorithm depends on the need to compute discrete logarithms modulo many different primes.

A related, slightly easier problem (see [21]) is to multiply n numbers, each of $O(\log n)$ bits, modulo another number of $O(\log n)$ bits. This is known to be in L and in L-uniform TC^0, but not in NC^1 or logtime uniform TC^0.

Since the predicate "$ab \equiv c \pmod{m}$" is in FO $=$ AC^0 when a, b, c, m are $O(logn)$-bit numbers, the table for the multiplicative group of Z_m is available in FO. Hence we can identify a generator and compute discrete logs modulo m in FOLL. To do iterated multiplication we need to (i) compute the discrete logs of all the inputs, (ii) add them modulo $\phi(m)$, and (iii) raise the generator to the result. Computing $\phi(m)$ is doable in FO. Iterated addition of integers is in logtime-uniform TC^0.

We thus have a uniform parallel algorithm that places this problem in any class that contains *both* FOLLand FO+MAJ. Unfortunately uniform NC^1 is not known (or even suspected) to contain FOLL.

A related question is how close we can come to iterated multiplication modulo $O(\log n)$-bit integers in, say, logtime-uniform NC^1 or TC^0. If our integers each have $O((\log n)/(\log\log n))$ bits, then FOLL operations on these numbers can be performed in uniform NC^1. Thus we can calculate discrete logs for these moduli, add them in uniform $\mathrm{TC}^0 \subseteq \mathrm{NC}^1$, and thus perform iterated multiplication in uniform NC^1. In the full paper we will discuss these and related applications of our FOLL algorithms.

4 Conclusions and Open Problems

We have introduced a new complexity class FOLL, and shown several problems involving groups presented as multiplication tables to be in it. This class is located somewhere between AC^0 and AC^1, yet seems incomparable with the more conventional complexity classes in that complexity spectrum. We note that this "strange complexity" of Cayley group problems is not unprecedented: the Cayley group isomorphism problem [19] is one of the only problems known to be solvable in polylogarithmic space but apparently not in polynomial time.

The obvious question is whether our FOLL upper bounds extend to more complicated groups, such as all solvable groups or even general groups. Our power basis property may be of independent interest. There is also the question of whether solvability of a group can be checked in FOLL or by some other fast parallel algorithm.

5 Acknowledgements

The authors thank Denis Thérien for several very helpful discussions. Some of these, and much other fruitful work on this project, occurred at a Dagstuhl seminar in November 1999 attended by three of the authors.

References

[1] L. Babai, Trading group theory for randomness, in *Proc. 17th ACM STOC* (1985), pp. 421–429.

[2] L. Babai and E. Szemerédi, On the complexity of matrix group problems I, Proc. 25th IEEE FOCS (1984), pp. 229-240.

[3] D.A.M. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. System Sci.*, 38:150–164, 1989.

[4] D.A.M. Barrington and N. Immerman. Time, hardware, and uniformity, in *Complexity Theory Retrospective II*, ed. L.A. Hemaspaandra and A.L. Selman, (New York: Springer Verlag, 1997), pp. 1–22.

[5] D.A.M. Barrington and P. McKenzie, Oracle branching programs and Logspace versus *P*, *Information and Computation* 95 (1991), pp. 96–115.

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

[7] L. Babai, E. Luks and A. Seress, Permutation Groups in *NC*, Proc. 19th ACM STOC (1987), pp. 409–420.

[8] P. Beame, S. Cook and J. Hoover, Log depth circuits for division and related problems, *SIAM Journal on Computing*, 15(4):994–1003, 1986.

[9] A. K. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, May 1984.

[10] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1):2–22, 1985.

[11] S. A. Cook and P. McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8:385–394, 1987.

[12] M. Furst, J. Hopcroft and E. Luks, Polynomial time algorithms for permutation groups, Proc. 21st IEEE FOCS (1980), pp. 36–41.

[13] M. Furst, J. Saxe and M. Sipser, Parity, circuits, and the polynomial-time hierarchy, *Math. Systems Theory* 17 (1984), pp. 13–27.

[14] C. Hoffmann, Group-theoretic algorithms and graph isomorphism, Springer LNCS vol. 136 (1979).

[15] N. Immerman, Descriptive and computational complexity, in *Computational Complexity Theory*, ed. J. Hartmanis, *Proc. of Symposia in Applied Mathematics* **38**, (Providence, RI: American Mathematical Society, 1989), pp. 75–91.

[16] N. Immerman, *Descriptive Complexity*, (New York: Springer Verlag, 1999).

[17] N. D. Jones and W. T. Laaser, Complete problems for deterministic polynomial time, *Theoretical Computer Science* 3 (1977), pp. 105-117.

[18] N. D. Jones, E. Lien, and W. T. Laaser. New problems complete for nondeterministic log space, *Math. Systems Theory* **10** (1976), pp. 1–17.

[19] R.J. Lipton, L. Snyder, and Y. Zalcstein. *The Complexity of Word and Isomorphism Problems for Finite Groups.* Johns Hopkins University (1976).

[20] E.M. Luks and P. McKenzie, Parallel algorithms for solvable permutation groups, *J. Computer and Systems Science* 37:1 (1988), pp. 39–62.

[21] P. McKenzie and S. A. Cook, The parallel complexity of the abelian permutation group membership problem, in Proc. 24th IEEE FOCS (1983), pp. 154–161.

[22] J. Rotman, The theory of groups, 2ed, Allyn & Bacon, Boston, 1973.

[23] C.C. Sims, Computational methods in the study of permutation groups, in *Computational Problems in Abstract Algebra,* ed. J. Leech, Pergamon Press (1970), pp. 169-183.

[24] M. Sipser, Borel sets and circuit complexity, *Proc. 15th ACM STOC* (1983), pp. 61–69.

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

[26] D. Thérien, *Classification of regular languages by congruences.* Ph.D. thesis, University of Waterloo, Research Report CS-80-19 (1980).

6 Technical Appendix

6.1 Proof of Lemma 3.5

A finite group G is nilpotent iff, for each prime p dividing $|G|$, the set of p-elements in G is a group.

Let G be nilpotent. Then G is a a direct product Π of p-groups (for various primes p, see [22, Theorem 5.39]). Hence, for each prime p dividing $|G|$, the p-elements of G are precisely the elements whose direct product components outside the p-groups are trivial. Hence the set of p-elements is a group, isomorphic to the direct product of the p-groups occurring in Π.

Conversely [22, Exercise 4.12], a finite group G having a unique maximal p-subgroup for each prime divisor p of $|G|$ is the direct product of these maximal p-subgroups. Hence, if the set of p-elements of G forms a group, then the latter must be the unique maximal p-subgroup of G. If this holds for each prime p, then G is a direct product of p-groups, hence nilpotent by [22, Theorem 5.39].

6.2 Proof of Theorem 3.10

CGM for groups that are solvable of class d is in $\mathrm{FO}(d \log\log n)$*. In particular, CGM for groups that are solvable of class $O(1)$ is in FOLL.*

Our algorithm consists of d rounds, each of which performs the following FOLL operation. As in the proof of Theorem 3.9, we take the current generating set and close it first under powers and then under products of length $O(\log n)$. We must show that d such rounds, starting from any generating set for any solvable group of class d, suffice to produce the entire group.

We prove this by induction on d, with the base case of $d = 1$ being Theorem 3.9. Let G (of order n) be an arbitrary solvable group of class d and let H be G's commutator subgroup, so that H is solvable of class $d-1$. Let X be an arbitrary set generating G. Since the elements xH for each x in X generate the quotient group G/H and G/H is abelian, any element of G is a product of at most $\log n$ powers of elements of X, followed by an element of H. That is, after the first round of closure under powers and $O(\log n)$ length products, our set will contain a representative of each coset of G/H.

We claim that H has a generating set Y each of whose elements is a product of $O(\log n)$ powers of elements of X. Given this claim, we apply the inductive hypothesis to H and Y. After $d-1$ more rounds, each consisting of closure first under powers and then under $O(\log n)$ products, our set will contain every element of H. Since every element of G is a product of one of our coset representatives and an element of H, we are done.

To prove the claim, let h be an arbitrary element of H and express h as a product $x_1 \dots x_\ell$, where each x_i is an element of X. (Since X generates G, it also generates H.) Define y_0 to be e and then inductively define elements y_i such that $x_1 \dots x_i y_i$ is in H and y_i is a product of $O(\log n)$ powers of elements of X. This is possible because y_i need only be in the same coset of G/H as $(x_1 \dots x_i)^{-1}$, and G/H is abelian and has the $O(\log n)$ power basis property. Note in particular that y_ℓ can be taken to be e, as $x_1 \dots x_\ell = h$ is already in H.

Now note that $x_1 \dots x_\ell$ can be rewritten as

$$(x_1 y_1)(y_1^{-1} x_2 y_2)(y_2^{-1} x_3 y_3) \dots (y_\ell^{-1} x_\ell)$$

and that each of the parenthesized terms is in H. So H is generated by the set of terms of the form $y^{-1}xz$ where x is in X and y and z are each products of $O(\log n)$ powers of elements of X. We have proved the claim and thus the theorem.