arXiv:1202.3949v2 [cs.CC] 5 Mar 2012

On the complexity of solving linear congruences and computing nullspaces modulo a constant

Niel de Beaudrap

DAMTP, Centre for Mathematical Sciences, University of Cambridge,
Wilberforce Road, Cambridge CB3 0WA, UK

5 March, 2012

Abstract

We consider the problems of determining the feasibility of a linear congruence, producing a solution to a linear congruence, and finding a spanning set for the nullspace of an integer matrix, where each of these problems are considered modulo an arbitrary constant $k \geqslant 2$. These problems are known to be complete for the logspace modular counting classes $\mathsf{coMod}_k\mathsf{L}$ in special case that k is prime [4]. By considering relaxed modular variants of standard logspace function classes, related to $\#\mathsf{L}$ and functions computable by UL machines but only characterizing the number of accepting paths mod k, we show that these problems of linear algebra are also complete for $\mathsf{coMod}_k\mathsf{L}$ for any constant $k \geqslant 2$.

1 Introduction

Solving a system of linear equations, or determining that it has none, is the definitive elementary problem of linear algebra over any ring. This problem is the practical motivator of the notions of matrix products, inverses, and determinants, among other concepts; and relates to other computational problems of abelian groups, such as testing membership in a subgroup [1]. Characterizing the complexity of this problem for common number systems, such as the integers, finite fields, or the integers modulo k is therefore naturally of interest.

We are interested in the difficulty of *deciding feasibility of linear congruences modulo k* (or $\mathbf{LCON}_k$) and *computing solutions to linear congruences modulo k* (or $\mathbf{LCONX}_k$) for an arbitrary constant $k \geqslant 2$. This is a special case of the problem **LCON** defined by McKenzie and Cook [1], in which k is taken as part of the input, but represented by its prime-power factors $p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$; where $e_j \in O(\log n)$ for each j (one says that each factor $p_j^{e_j}$ is *tiny*). Setting k to a constant is a natural, if slightly restrictive, special case.

Arvind and Vijayaraghavan [2] recently defined ModL (a logspace analogue the class ModP defined by Köbler and Toda [3]), which is contained in NC^2. They show that **LCON** is hard for ModL under P-uniform NC^1 reductions, and contained in $\mathsf{L}^{\mathsf{ModL}}/\mathsf{poly} = \mathsf{L}^{\#\mathsf{L}}/\mathsf{poly}$. This is of course in contrast to the problem of determining integer-feasibility of integer matrix equations, which is at least as hard as computing greatest common divisors over $\mathbb{Z}$; the latter problem is not known to be in NC^j for any $j \geqslant 0$. Furthermore, Buntrock *et al.* [4] show —

for the special case of k prime — that determining the feasibility of systems of linear equations is complete for $\mathsf{coMod}_k\mathsf{L}$; where these are the complementary classes to the better known classes $\mathsf{Mod}_k\mathsf{L}$ which generalize $\oplus\mathsf{L}$, corresponding to logspace nondeterministic Turing machines which can distinguish between having a number of accepting paths which are either zero or nonzero *modulo* k.

The above results suggest that the difficulty of solving linear equations over integer matrices is strongly governed by the presence and the prime-power factorization of the modulus involved, and indicates that $\mathbf{LCON}_k$ may be particularly tractable. Also implicit in Ref. [4] is that $\mathbf{LCON}_k$ is $\mathsf{coMod}_k\mathsf{L}$-hard for all $k \geqslant 2$. This suggests the question: for an *arbitrary* modulus k, what is the precise relationship of the problem $\mathbf{LCON}_k$ of deciding the feasibility of linear congruences modulo k, to the classes $\mathsf{coMod}_k\mathsf{L}$?

We show how the analysis of McKenzie and Cook [1] for the problem **LCON** may be adapted to exhibit a $\mathsf{coMod}_k\mathsf{L}$ algorithm for $\mathbf{LCON}_k$. Using techniques similar to those used by Hertrampf, Reif, and Vollmer [5] to show closure of the class $\mathsf{Mod}_p\mathsf{L}$ under oracle reductions for p prime, we describe a function class FUL_p which is well-suited for describing oracles which may be simulated in mod-logspace computations. We describe a recursive construction for a FUL_{p^e} algorithm (for any fixed prime power p^e) to solve the problem $\mathbf{LCONNULL}_{p^e}$ of computing a spanning set for a basis of the nullspace of a matrix modulo p^e. This allows us to demonstrate that $\mathbf{LCON}_k$ is $\mathsf{coMod}_k\mathsf{L}$-complete, and both $\mathbf{LCONX}_k$ and $\mathbf{LCONNULL}_k$ are $\mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$-complete, for any constant $k \geqslant 2$.

2 Preliminaries

Throughout the following, $k \geqslant 2$ is a constant modulus, with a factorization into powers of distinct primes $k = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$. When we consider the case of a modulus which is a prime power, we will write p^e rather than k, for p some prime and $e \geqslant 1$ some positive integer which are independent of the input.

We consider the complexity of the following problems, which are named in analogy to problems considered by McKenzie and Cook [1]:

Problems. Fix $k \geqslant 2$. For an $m \times n$ integer matrix A and vector $\mathbf{y} \in \mathbb{Z}^m$ provided as input, we define the following problems:

- $\mathbf{LCON}_k$: determine whether $A\mathbf{x} \equiv \mathbf{y} \pmod{k}$ has solutions for $\mathbf{x} \in \mathbb{Z}^n$.
- $\mathbf{LCONX}_k$: output a solution to the congruence $A\mathbf{x} \equiv \mathbf{y} \pmod{k}$, or indicate that no solutions exist.
- $\mathbf{LCONNULL}_k$: output a set $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of vectors spanning the solution space of the congruence $A\mathbf{x} \equiv \mathbf{0} \pmod{k}$.

Without loss of generality, we may suppose $m = n$ by padding the matrix A.

We wish to describe the relationship of these problems to the classes $\mathsf{coMod}_k\mathsf{L}$ for $k \geqslant 2$, which are the complements of the better known classes $\mathsf{Mod}_k\mathsf{L}$ defined by Buntrock *et al.* [4].

Definition I. The class $\mathsf{coMod}_k\mathsf{L}$ (respectively $\mathsf{Mod}_k\mathsf{L}$) is the set of languages L for which there exists $\varphi \in \#\mathsf{L}$ such that $x \in L$ if and only if $\varphi(x) \equiv 0 \pmod{k}$ (respectively, $\varphi(x) \not\equiv 0 \pmod{k}$).

The following results are a synopsis of Ref. [4, Theorem 9]:

Proposition 1. *We may characterize* $\mathsf{coMod}_k\mathsf{L}$ *as the class of decision problems which are log-space reducible to verifying matrix determinants mod* k*, or coefficients of integer matrix products or matrix inverses mod* k*.*

Proposition 2. *For* p *prime,* $\mathbf{LCON}_p$ *is* $\mathsf{coMod}_p\mathsf{L}$*-complete.*

Buntrock *et al.* also characterize the classes $\mathsf{coMod}_k\mathsf{L}$ in terms of the prime factors of k, and show closure results which will prove useful. The following are implicit in Lemma 5, Theorem 6, and Corollary 7 of Ref. [4]:

Proposition 3 (normal form)**.** *Let* $k = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$ *be the factorization of* $k \geqslant 2$ *into prime powers* $p_j^{e_j}$*. Then* $L \in \mathsf{coMod}_k\mathsf{L}$ *if and only if there are languages* $L_j \in \mathsf{coMod}_{p_j}\mathsf{L}$ *such that* $L = L_1 \cap \cdots \cap L_\ell$*. In particular,* $\mathsf{coMod}_k\mathsf{L} = \mathsf{coMod}_{p_1 p_2 \cdots p_\ell}\mathsf{L}$*.*

Proposition 4 (closure under intersections)**.** *For any* $k \geqslant 2$ *and languages* $L, L' \in \mathsf{coMod}_k\mathsf{L}$*, we have* $L \cap L' \in \mathsf{coMod}_k\mathsf{L}$*.*

Proposition 5 (limited closure under complements)**.** *For any prime* p *and* $e \geqslant 1$*, we have* $\mathsf{coMod}_{p^e}\mathsf{L} = \mathsf{coMod}_p\mathsf{L} = \mathsf{Mod}_p\mathsf{L} = \mathsf{Mod}_{p^e}\mathsf{L}$*.*

A system of linear congruences mod k has solutions if and only if it has solutions modulo each prime power divisor $p_j^{e_j}$ of k. We then have $\mathbf{LCON}_k \in \mathsf{coMod}_k\mathsf{L}$ if and only if $\mathbf{LCON}_{p^e} \in \mathsf{coMod}_{p^e}\mathsf{L} = \mathsf{coMod}_p\mathsf{L}$ by Proposition 3. (In fact, this suffices to show that $\mathbf{LCON}_k \in \mathsf{coMod}_k\mathsf{L}$ for all square-free integers $k \geqslant 2$.)

We see from Propositions 2 and 5 that the case of a prime modulus is special. For p prime, Buntrock *et al.* also implicitly characterize the complexity of $\mathbf{LCONX}_p$ and $\mathbf{LCONNULL}_p$. We may describe the complexity of these function problems as follows. For a function $f(x) : \Sigma^* \to \Sigma^*$ and $x \in \Sigma^*$, let $|f(x)|$ denote the length of the representation of $f(x)$; and let $f(x)_j$ denote the jth symbol in that representation. Following Hertrampf, Reif, and Vollmer [5], for a function $f : \Sigma^* \to \Sigma^*$ on some alphabet Σ, and for some symbol $\bullet \notin \Sigma$, we may define the decision problem

$$\mathbf{bits}(f) = \left\{ (x, j, b) \;\middle|\; \begin{array}{r} \text{either } j \leqslant |f(x)| \text{ and } b = f(x)_j \\ \text{or } j > |f(x)| \text{ and } b = \bullet \end{array} \right\}. \tag{1}$$

Abusing notation, we write $f(x)_j = \bullet$ in case $|f(x)| < j$. We extend this definition to *partial* functions f by asserting $(x, j, b) \in \mathbf{bits}(f)$ only if $x \in \mathrm{dom}(f)$.

Definition II. The class $\mathsf{F{\cdot}coMod}_k\mathsf{L}$ is the set of (partial) functions f such that $|f(x)| \in \mathrm{poly}(|x|)$ for all $x \in \Sigma^*$, and for which $\mathbf{bits}(f) \in \mathsf{coMod}_k\mathsf{L}$. (We define the class $\mathsf{FMod}_k\mathsf{L}$ similarly.)

Then Ref. [4, Theorem 9] also implicitly shows:

Proposition 6. *For* p *prime, the problems* $\mathbf{LCONX}_p$ *and* $\mathbf{LCONNULL}_p$ *are* $\mathsf{F{\cdot}coMod}_p\mathsf{L}$*-complete.*

In Section 3, we describe two additional function classes which are natural when considering modular logspace computation. Relationships between these classes in the case of prime-power modulus will allow us to easily show in Section 4 that in fact $\mathbf{LCONX}_{p^e}, \mathbf{LCONNULL}_{p^e} \in \mathsf{F{\cdot}coMod}_p\mathsf{L}$ for all prime powers p^e. These results then naturally extend to all moduli $k \geqslant 2$, so that $\mathbf{LCONX}_k, \mathbf{LCONNULL}_k \in \mathsf{F{\cdot}coMod}_k\mathsf{L}$, with $\mathbf{LCON}_k \in \mathsf{coMod}_k\mathsf{L}$ following as a corollary.

3 Natural function classes for modular logspace

We now introduce two classes for counting classes in logarithmic space: a modular analogue of $\#\mathsf{L}$, and a class of function problems which is naturally low for $\mathsf{Mod}_k\mathsf{L}$ and $\mathsf{coMod}_k\mathsf{L}$. We describe the relationships of these classes to $\mathsf{FMod}_k\mathsf{L}$ and $\mathsf{F{\cdot}coMod}_k\mathsf{L}$, and to each other in the case of a prime modulus.

Definition III. The class $\#\mathsf{L}_k$ is the set of functions $f : \Sigma^* \to \mathbb{Z}/k\mathbb{Z}$ such that $f(x) = \varphi(x) + k\mathbb{Z}$ for some function $\varphi \in \#\mathsf{L}$.

Note that $\#\mathsf{L}_k$ inherits closure under addition, multiplication, and constant powers from $\#\mathsf{L}$; it is closed under subtraction as well, as $M-N \equiv M+(k-1)N \pmod{k}$. We may then rephrase Proposition 1 as follows:

Proposition 7. *Evaluating matrix determinants modulo k, coefficients of products of integer matrices modulo k, and coefficients of inverses modulo k of integer matrices, are complete problems for $\#\mathsf{L}_k$.*

Similar containments hold for each of the problems listed in Ref. [4, Theorem 9]: any decision problem in $\mathsf{coMod}_k\mathsf{L}$ (such as the complete problems listed by Buntrock *et al.*) consists of comparing some function $f \in \#\mathsf{L}_k$ to a constant or an input value. Thus we trivially have:

Lemma 8. *For any $k \geqslant 2$, $\#\mathsf{L}_k \subseteq \mathsf{F{\cdot}coMod}_k\mathsf{L}$.*

We may adopt the common conflation between equivalence classes $a + k\mathbb{Z} \in \mathbb{Z}/k\mathbb{Z}$ and integers $0 \leqslant a < k$, in which case we may instead require $f \in \#\mathsf{L}_k$ to satisfy $0 \leqslant f(x) < k$ and $f(x) \equiv \varphi(x) \bmod k$ for some $\varphi \in \#\mathsf{L}$. This will allow us to consider logspace machines which compute $\#\mathsf{L}_k$ functions on their output tapes. We will be interested in a particular sort of nondeterministic logspace machine which is suitable for performing computations as subroutines of $\mathsf{coMod}_k\mathsf{L}$ machines: the main result of this section is to describe conditions under which it can compute functions in $\#\mathsf{L}_k$.

Definition IV. A FUL_k *machine computing a (partial) function f* is a nondeterministic logspace Turing machine which **(a)** for inputs $x \in \mathrm{dom}(f)$, computes $f(x)$ on its output tape in some number $\varphi(x, f(x)) \equiv 1 \pmod{k}$ of its accepting branches, and **(b)** for each $y \neq f(x)$ (or for any string y, in the case $x \notin \mathrm{dom}(f)$), computes y on its output tape on some number $\varphi(x, y) \equiv 0 \pmod{k}$ of its accepting branches. We say that $f \in \mathsf{FUL}_k$ if there exists a FUL_k machine which computes f.

If we replace the relation of equivalence modulo k with equality in the definition of FUL_k above, we obtain a class FUL of functions computable by nondeterministic logspace machines with a single accepting branch. This latter class is analogous to the class UPF described in Ref. [6], which is in effect a class of functions which may be computed by a nondeterministic polynomial time Turing machine as a subroutine without affecting the number of accepting branches of that machine. Modulo k and in logarithmic space, this is the significance of the class FUL_k. Note that in many branches (perhaps even the vast majority of them), what is written on the output tape of a FUL_k machine $\mathbf{U}$ may not be the function $f(x)$ which it "computes"; but any result other than $f(x)$ which $\mathbf{U}$ is meant to compute, cannot affect the number of accepting branches modulo k

of any machine which simulates $\mathbf{U}$ directly, *e.g.* as a subroutine. These "incorrect results" may therefore be neglected for the purpose of counting accepting branches modulo k, just as if all accepting branches of $\mathbf{U}$ (of which there are not a multiple of k) computed the result $f(x)$ on the output tape.

In this sense, the closure result $\mathsf{Mod}_p\mathsf{L}^{\mathsf{Mod}_p\mathsf{L}} = \mathsf{Mod}_p\mathsf{L}$ for p prime shown by Hertrampf, Reif, and Vollmer [5] may be interpreted as saying that the characteristic function of any $L \in \mathsf{Mod}_p\mathsf{L}$ may be computed by a FUL_p machine; and so a $\mathsf{Mod}_p\mathsf{L}$ oracle can be directly simulated by a $\mathsf{Mod}_p\mathsf{L}$ machine, by simulating the corresponding FUL_p machine as a subroutine. Our interest in the function class FUL_k is for essentially the same reason, *i.e.* an oracle for computing any function $f \in \mathsf{FUL}_k$ can be substituted with a simulation of the FUL_k machine itself in the same manner:

Lemma 9. $\mathsf{Mod}_k\mathsf{L}^{\mathsf{FUL}_k} = \mathsf{Mod}_k\mathsf{L}$, $\mathsf{coMod}_k\mathsf{L}^{\mathsf{FUL}_k} = \mathsf{coMod}_k\mathsf{L}$, *and* $\mathsf{FUL}_k^{\mathsf{FUL}_k} = \mathsf{FUL}_k$ *for all* $k \geqslant 2$.

The proof is essentially the same as that for the oracle closure result of Ref. [5], of which this Lemma is a natural extension. From simple number-theoretic considerations, the classes FUL_k have other properties which are similar to those of $\mathsf{coMod}_k\mathsf{L}$:

Theorem 10. *Let* $k = p_1^{e_1}p_2^{e_2}\cdots p_\ell^{e_\ell}$ *be the factorization of* $k \geqslant 2$ *into prime power factors* $p_j^{e_j}$. *Then* $\mathsf{FUL}_k = \mathsf{FUL}_{p_1} \cap \mathsf{FUL}_{p_2} \cap \cdots \cap \mathsf{FUL}_{p_\ell}$, *and in particular* $\mathsf{FUL}_k = \mathsf{FUL}_{p_1p_2\cdots p_\ell}$.

Proof. Throughout the following, let $\kappa = p_1p_2\cdots p_\ell$ be the largest square-free factor of k. We first show $\mathsf{FUL}_\kappa = \mathsf{FUL}_{p_1} \cap \cdots \cap \mathsf{FUL}_{p_\ell}$. Suppose $f \in \mathsf{FUL}_{p_j}$ for each $1 \leqslant j \leqslant \ell$, and is computed by some FUL_{p_j} machine $\mathbf{U}_j$ in each case. Let

$$\gamma \;=\; \kappa/p_1 + \kappa/p_2 + \cdots + \kappa/p_\ell \;. \tag{2}$$

For each prime p_j, all of the terms in the right-hand sum are divisible by p_j except for the jth term. Then γ is coprime to p_j for each j, and so is also coprime to κ. Let $\beta \equiv \gamma^{-1} \pmod{\kappa}$, and consider the machine $\mathbf{U}'$ which performs the following:

1. Nondeterministically write some index $1 \leqslant j \leqslant \ell$ on the work tape.
2. For each such j, nondeterministically select some integer $0 \leqslant q < \beta\kappa/p_j$.
3. In each branch, simulate $\mathbf{U}_j$ on the input x, accepting if and only if $\mathbf{U}_j$ accepts.

For any string $y \in \Sigma^*$ different from $f(x)$, the number of branches in which $\mathbf{U}_j$ accepts is m_jp_j for some $m_j \in \mathbb{N}$; and so $\mathbf{U}'$ has $m_j\beta\kappa$ branches where j is written on the work tape and y is written on the output tape. Summing over all j, we find that any $y \neq f(x)$ is written on the output tape in a number of branches which is a multiple of κ. Similarly, for the case $y = f(x)$, the number of branches in which $\mathbf{U}_j$ accepts is $m_jp_j + 1$ for some $m_j \in \mathbb{N}$; and so $\mathbf{U}'$ has $m_j\beta\kappa + \beta\kappa/p_j$ branches where j is written on the work tape and $f(x)$ is written on the output tape. Summing over all j and neglecting multiples of κ, we have $\beta\big(\kappa/p_1 + \cdots + \kappa/p_\ell\big) = \beta\gamma \equiv 1 \pmod{\kappa}$ branches in which $f(x)$ is written on the output tape; thus $\mathbf{U}'$ is an FUL_κ machine computing f. The converse containment $\mathsf{FUL}_\kappa \subseteq \mathsf{FUL}_{p_j}$ for each $1 \leqslant j \leqslant \ell$ is trivial.

It remains to show that $\mathsf{FUL}_\kappa \subseteq \mathsf{FUL}_k$, the reverse containment again being easy. Let $\mathbf{U}'$ be a FUL_κ machine computing a function $f : \Sigma^* \to \Sigma^*$ with length bounded above by $|f(x)| \leqslant N(x) \in \mathrm{poly}(|x|)$. Suppose $N(x) \in O(\log|x|)$: we may then construct a FUL_k machine $\mathbf{U}''$ which computes f by simply performing k/κ consecutive independent simulations of $\mathbf{U}'$, recording the outcome of each simulation on the work tape. For each $1 \leqslant j \leqslant k/\kappa$, in any given computational branch, let $\varphi_j(x)$ be the string computed by the jth simulation of $\mathbf{U}'$. If any of the simulations produce a different output (*i.e.* if $\varphi_h(x) \neq \varphi_j(x)$ for any $1 \leqslant h, j \leqslant k/\kappa$) or if any of the simulations rejected the input, $\bar{\mathbf{U}}$ rejects. Otherwise, $\bar{\mathbf{U}}$ writes the string $\varphi_1(x)$ agreed upon by the simulations to the output tape. More generally, if $N(x) \in \omega(\log|x|)$, then fix some $L \in O(\log|x|)$, and define for each $1 \leqslant m \leqslant N(x)/L$ a machine $\mathbf{U}'_m$ which writes the mth block of L consecutive characters from $f(x)$, padding the end of $f(x)$ with a symbol $\bullet \notin \Sigma$ if necessary. Rather than perform k/κ simulations of $\mathbf{U}'$, the machine $\mathbf{U}''$ performs k/κ simulations of each such $\mathbf{U}'_m$, again writing their outcomes (excluding any instance of the symbol $\bullet \notin \Sigma$) to the work tape if and only if each simulation accepts and agrees on their output. Once $N(x)$ symbols have been written to the output tape, $\mathbf{U}''$ accepts unconditionally.

Let $\varphi(x, y)$ be the number of computational branches in which $\mathbf{U}$ accepts with the string $y \in \Sigma^*$ written on the tape: by hypothesis, $\varphi(x, y) \equiv 0 \pmod{\kappa}$ for each $y \neq f(x)$, and $\varphi(x, f(x)) \equiv 1 \pmod{\kappa}$. Similarly, let $\varphi_m(x, y^{(m)})$ be the number of branches in which $\mathbf{U}'_m$ accepts with $y^{(m)} \in \Sigma^L$ written on the tape for each $1 \leqslant r \leqslant N(x)/L$, and $\Phi(x, y)$ be the number of branches in which $\mathbf{U}''$ accepts with $y \in \Sigma^*$ written on the tape. Let $M = N(x)/L$ for the sake of brevity. If $y = y^{(1)}y^{(2)}\cdots y^{(M)} \in \Sigma^*$, then

$$\Phi(x, y) \;=\; \varphi_1\big(x, y^{(1)}\big)^{k/\kappa}\, \varphi_2\big(x, y^{(2)}\big)^{k/\kappa} \;\cdots\; \varphi_M\big(x, y^{(M)}\big)^{k/\kappa}, \tag{3}$$

as for each $y^{(m)}$, the number of branches in which $\mathbf{U}''$ accepts with $y^{(m)}$ written on the output tape is independent of the other substrings $y^{(j)}$ for $j \neq m$, and results from k/κ simulations of $\mathbf{U}'_m$ which each produce the substring $y^{(m)}$ as output.

Note that $\varphi_m(x, \lambda_m)$ is equal to the number of computational branches in which $\mathbf{U}'$ writes a string $\sigma \in \Sigma^*$ on the output tape in which the mth block is equal to $y^{(m)}$, which is the sum of $\varphi(x, \sigma)$ over all strings σ consistent with the substring $y^{(m)}$. By hypothesis, $\varphi(x, \sigma)$ is a multiple of κ except for the single case where $\sigma = f(x)$, in which case $\varphi(x, \sigma) \equiv 1 \pmod{\kappa}$. Thus $\varphi_m(x, y^{(m)}) \equiv 1 \pmod{\kappa}$ if $y^{(m)} \in \Sigma^L$ is consistent with the mth block of $f(x)$; otherwise, $\varphi_m(x, y^{(m)}) \equiv 0 \pmod{\kappa}$. We then observe the following:

- Let $E = \max_j\{e_j\}$ be the largest power of a prime p_j dividing k; then $E \leqslant p_j^{E-1} \leqslant k/\kappa$ for any $1 \leqslant j \leqslant \ell$. As k divides $\kappa^E = p_1^E \cdots p_\ell^E \leqslant \kappa^{k/\kappa}$, we then have $\varphi_m(x, y^{(m)})^{k/\kappa} \equiv 0 \pmod{k}$ if $\varphi_m(x, y^{(m)}) \equiv 0 \pmod{\kappa}$.

- The integers which are congruent to 1 modulo κ form a subgroup of order k/κ within the integers modulo k; it then follows that $\varphi_m(x, y^{(m)})^{k/\kappa} \equiv 1 \pmod{k}$ if $\varphi(x, y^{(m)}) \equiv 1 \pmod{\kappa}$.

Taking the product over $1 \leqslant m \leqslant M$, we have $\Phi(x, y) \equiv 0 \pmod{k}$ unless each substring $y^{(m)}$ is consistent with the mth block of $f(x)$, in which case $y = f(x)$ and $\Phi(x, y) \equiv 1 \pmod{k}$. Thus $\mathbf{U}''$ is an FUL_k machine computing f. □

The requirement that an FUL_k machine have one accepting branch modulo k allows us to easily relate FUL_k to the classes $\mathsf{FMod}_k\mathsf{L}$ and $\mathsf{F{\cdot}coMod}_k\mathsf{L}$:

Lemma 11. *For all $k \geqslant 2$, we have $\mathsf{FUL}_k \subseteq \mathsf{FMod}_k\mathsf{L} \cap \mathsf{F{\cdot}coMod}_k\mathsf{L}$.*

Proof. Let $\mathbf{U}$ be a FUL_k machine computing $f : \Sigma^* \to \Sigma^*$. Consider a nondeterministic logspace machine $\mathbf{T}$ taking inputs $(x, j, b) \in \Sigma^* \times \mathbb{N} \times \big(\Sigma \cup \{\bullet\}\big)$, and which simulates $\mathbf{U}$, albeit ignoring all instructions to write to the output tape, except for the jth symbol which it writes to the work-tape. (If $j > |f(x)|$, $\mathbf{T}$ instead writes "$\bullet$" to the work-tape.) Then $\mathbf{T}$ compares the resulting symbol $f(x)_j$ against b, accepting if they are equal and rejecting otherwise. Then the number of accepting branches is equivalent to 1 modulo k if $f(x)_j = b$, and is a multiple of p otherwise, so that $\mathbf{bits}(f) \in \mathsf{Mod}_k\mathsf{L}$. To show $\mathbf{bits}(f) \in \mathsf{coMod}_k\mathsf{L}$, we may consider a machine $\mathbf{T}'$ which differs from $\mathbf{T}$ only in that it rejects if $f(x)_j = b$, and accepts otherwise. Thus $\mathsf{FUL}_k \subseteq \mathsf{FMod}_k\mathsf{L} \cap \mathsf{F{\cdot}coMod}_k\mathsf{L}$. □

This identifies FUL_k as an important subclass of the existing logspace-modular function classes. For prime-power moduli, we may sharpen Lemma 11 to obtain a useful identity:

Lemma 12. *For any prime p and $e \geqslant 1$, $\mathsf{FUL}_{p^e} = \mathsf{FMod}_p\mathsf{L} = \mathsf{F{\cdot}coMod}_p\mathsf{L}$.*

Proof. By Proposition 5 and Lemma 10, it suffices to prove $\mathsf{FUL}_p = \mathsf{FMod}_p\mathsf{L}$ for p prime. For $f \in \mathsf{FMod}_p\mathsf{L}$, we may construct from the $\mathsf{Mod}_p\mathsf{L}$ machine $\mathbf{T}$ which decides $\mathbf{bits}(f)$ a family of machines $\mathbf{T}_{j,b}$ (for each $j \in \mathbb{N}$ and $b \in \Sigma \cup \{\bullet\}$), each of which writes b on its output tape and deciding whether $(x, j, b) \in \mathbf{bits}(f)$ on an input $x \in \Sigma^*$. Without loss of generality, as in [5, Corollary 3.2] each machine $\mathbf{T}_{j,b}$ accepts on a number of branches $\varphi(x, j, b) \equiv 1 \pmod{p}$ if case $f(x)_j = b$, and $\varphi(x, j, b) \equiv 0 \pmod{p}$ otherwise.

We form a FUL_p machine $\mathbf{U}_j$ computing $f(x)_j$ by taking the "disjunction" of the machines $\mathbf{T}_{j,b}$ over all $b \in \Sigma \cup \{\bullet\}$: *i.e.* $\mathbf{U}_j$ branches nondeterministically by selecting $b \in \Sigma \cup \{\bullet\}$ to write on the work-tape and simulates $\mathbf{T}_{j,b}$, accepting with one branch mod p if and only if $b = f(x)_j$ and accepting with zero branches mod p otherwise. Given some upper bound $|f(x)| \leqslant N(x) \in \mathrm{poly}(|x|)$, we then construct a FUL_p machine $\mathbf{U}$ to compute $f(x)$ by simply simulating $\mathbf{U}_j$ for each $1 \leqslant j \leqslant N(x)$ in sequence, writing the symbols $f(x)_j$ individually on the output tape; accepting once it either computes a symbol $f(x)_j = \bullet$ (without writing $\bullet$ to the output) or the final iteration has been carried out. □

Lemma 12, together with Lemma 9, may be taken as re-iterating the closure result of Ref. [5] explicitly in terms of function classes. The importance of this result to us is in the following consequences, which follow from Proposition 8 and Lemma 9:

Corollary 13. *For any prime p and $e \geqslant 1$, $\#\mathsf{L}_{p^e} \subseteq \mathsf{FUL}_p$.*

Corollary 14. *For any prime p and $e \geqslant 1$, $\mathsf{coMod}_p\mathsf{L}^{\#\mathsf{L}_{p^e}} = \mathsf{coMod}_p\mathsf{L}$.*

The former result states that we may explicitly compute functions in $\#\mathsf{L}$ (albeit up to equivalence modulo p^e) on the work tape, as subroutines in decision algorithms for $\mathsf{coMod}_p\mathsf{L}$; this allows us to simulate logspace counting oracles modulo p^e in $\mathsf{coMod}_p\mathsf{L}$. In the following section, we use this to describe an algorithm for $\mathbf{LCONNULL}_{p^e}$ in $\mathsf{F{\cdot}coMod}_p\mathsf{L}$ by a similar analysis to McKenzie and Cook [1]. By standard techniques, we may then demonstrate containments for $\mathbf{LCON}_k$, $\mathbf{LCONNULL}_k$, and $\mathbf{LCONX}_k$ in terms of $\mathsf{coMod}_k\mathsf{L}$.

4 Solving congruences and nullspaces mod k

We return to the motivating problems of this article. We let A be an $n \times n$ integer matrix and $\mathbf{y} \in \mathbb{Z}^n$ which are provided as the input to $\mathbf{LCON}_k$ or $\mathbf{LCONX}_k$; and for $\mathbf{LCONNULL}_k$, we consider an $n \times n$ matrix B. Without loss of generality, the coefficients of A and $\mathbf{y}$, or of B, are non-negative and bounded strictly above by k (as reducing the input modulo k can be performed in NC^1). We essentially follow the analysis of Ref. [1, Section 8], which reduces solving linear congruences to computing generating sets for nullspaces modulo the primes p_j dividing k. The technical contribution of this section is to show that the latter problem can be solved for prime powers via a reduction to matrix multiplication together with modular counting oracles from $\#\mathsf{L}_{p^e}$ for prime powers p^e.

4.1 Computing nullspaces modulo prime powers

We consider an NC^1 reduction to matrix inversion and iterated matrix products modulo p^e, in a machine equipped with a $\#\mathsf{L}_{p^e}$ oracle to compute certain matrix coefficients. As we note in Proposition 7, computing individual coefficients of matrix inverses and matrix products are complete problems for $\mathsf{F{\cdot}coMod}_p\mathsf{L}$, and Corollary 14 implies that this class can simulate $\#\mathsf{L}_{p^e}$ oracles. The NC^1 reduction itself is essentially the same as that of McKenzie and Cook [1], which we may summarize as follows.

The prime modulus case. First, consider the case $e = 1$, which as we note in Section 2 is solved by Buntrock *et al.* [4, Theorem 9]. For an $n \times n$ integer matrix B, we may reduce the problem of computing a basis of $\mathrm{null}(B) \bmod p$ to rank computations and matrix inversion using the techniques of Borodin, von zur Gathen, and Hopcroft [7, Theorem 5]. This involves testing the ranks of a nested collection of sub-matrices of B, to determine a subset of columns forming a basis for $\mathrm{img}(B)$; the reduction from nullspaces is a truth-table reduction, which for the ultimate reduction to $\mathsf{coMod}_p\mathsf{L}$ means that we must use Proposition 4 (to enable conjunctive reductions) and Proposition 5 (to allow disjunctive reductions). Computing the rank of a matrix modulo a prime (*i.e.* in the field $\mathbb{F}_p$) may be reduced to computing characteristic polynomials of matrices in $\mathbb{F}_p(\tau)$ for a formal indeterminate τ using a result of Mulmuley [8]; this may be reduced to iterated matrix products over $\mathbb{F}_p(\tau)$ by a construction of Berkowitz [9], where the coefficients of the matrices are all either constants or drawn from the coefficients of the matrix M. By deriving a suitable bound on the degrees of the polynomials over τ involved in these iterated matrix products, one may substitute the polynomial coefficients by polynomial-size Toeplitz matrix blocks [10], thereby reducing the iterated matrix product over $\mathbb{F}_p(\tau)$ to one over $\mathbb{F}_p$.

Recursive reduction for higher powers of primes. The remainder of the NC^1 reduction consists essentially of Ref. [1, Lemma 8.1] which put **LCONNULL** (for a variable modulus with magnitude at most linear in n) in NC^3: in our case, we reduce $\mathbf{LCONNULL}_{p^e}$ to $\mathbf{LCONNULL}_p$ together with matrix products and access to oracles for computing coefficients of certain matrices. Inducting on $1 \leqslant t \leqslant e$, suppose that we have a generating set $\mathbf{V}_1^{(t)}, \ldots, \mathbf{V}_{N_t}^{(t)}$ over $\mathbb{Z}/p^e\mathbb{Z}$ for the nullspace of B modulo p^t. Certainly any solution to $B\mathbf{w} \equiv 0 \pmod{p^{t+1}}$

must also be a solution to $B\mathbf{w} \equiv 0 \pmod{p^t}$ as well; then we may decompose such $\mathbf{w}$ modulo p^e as a linear combination of the vectors $\mathbf{V}_j^{(t)}$,

$$\mathbf{w} = u_1\mathbf{V}_1^{(t)} + \cdots + u_{N_t}\mathbf{V}_{N_t}^{(t)} + p^t\hat{\mathbf{w}} \tag{4a}$$

for some $\hat{\mathbf{w}} \in \mathbb{Z}^n$; or more concisely,

$$\mathbf{w} = V^{(t)}\mathbf{z}\,, \tag{4b}$$

where we define the block matrices $V^{(t)} = \left[\, \mathbf{V}_1^{(t)}\ \mathbf{V}_2^{(t)}\ \cdots\ \mathbf{V}_{N_t}^{(t)} \,\middle|\, p^t I \,\right]$ and $\mathbf{z} = \left[\, u_1\ u_2\ \cdots\ u_{N_t} \,\middle|\, \hat{\mathbf{w}} \,\right]^{\mathsf{T}} \in \mathbb{Z}^{N_t+n}$. To consider the additional constraints imposed by $B\mathbf{w} \equiv 0 \pmod{p^{t+1}}$, consider a decomposition $B = B_t + p^t\hat{B}_t$, where the coefficients of B_t are bounded between 0 and p^t. We then have

$$\begin{aligned}\left(\sum_{j=1}^{N_t} u_j\left[B_t\mathbf{V}_j^{(t)} \;+\; p^t\hat{B}_t\mathbf{V}_j^{(t)}\right]\right) + p^t B_t\hat{\mathbf{w}} \\ \equiv\; B\left(\sum_{j=1}^{N_t} u_j\mathbf{V}_j^{(t)}\right) + p^t\hat{\mathbf{w}} \equiv 0 \pmod{p^{t+1}}\,.\end{aligned} \tag{5a}$$

As the vectors $B_t\mathbf{V}_j^{(t)}$ have coefficients divisible by p^t by construction, we may simplify to

$$\left(\sum_{j=1}^{N_t} u_j\left[B_t\mathbf{V}_j^{(t)}/p^t \;+\; \hat{B}_t\mathbf{V}_j^{(t)}\right]\right) + B_t\hat{\mathbf{w}} \equiv 0 \pmod{p}\,, \tag{5b}$$

or somewhat more concisely,

$$\bar{B}^{(t)}\mathbf{z} \equiv 0 \pmod{p}, \tag{5c}$$

where we define

$$\bar{B}^{(t)} = \left[\, \mathbf{b}_1^{(t)}\ \mathbf{b}_2^{(t)}\ \cdots\ \mathbf{b}_{N_t}^{(t)} \,\middle|\, B_t \,\right], \qquad \text{for } \mathbf{b}_j^{(t)} = B_t\mathbf{V}_j^{(t)}/p^t \;+\; \hat{B}_t\mathbf{V}_j^{(t)}, \tag{6}$$

and where $\mathbf{z}$ is as we defined it above. To find not just one vector $\mathbf{w}$ but a set of generators $\mathbf{V}_1^{(t+1)}, \ldots, \mathbf{V}_{N_{t+1}}^{(t+1)}$ over $\mathbb{Z}/p^e\mathbb{Z}$ for null(B) mod p^{t+1}, it suffices to find a generating set $\mathbf{z}_1, \ldots, \mathbf{z}_{N_{t+1}}$ for the nullspace of $\bar{B}^{(t)}$ mod p, and then set $\mathbf{V}_h^{(t+1)} = V^{(t)}\mathbf{z}_h$. Note that the nullspace of $\bar{B}^{(t)}$ modulo p over $\mathbb{Z}/p^e\mathbb{Z}$ will contain many vectors which are equivalent mod p, but at most N_t+n equivalence classes; we may then without loss of generality select vectors $\mathbf{z}_1 = p\hat{\mathbf{e}}_1$, $\mathbf{z}_2 = p\hat{\mathbf{e}}_2$, $\ldots$, $\mathbf{z}_{N_t} = p\hat{\mathbf{e}}_{N_t}$, and choose the remaining vectors $\mathbf{z}_h$ representing non-trivial vectors in null$(\bar{B}^{(t)})$ mod p to have coefficients bounded between 0 and p. We thus obtain $N_{t+1} \leqslant 2N_t+n$ vectors over $\mathbb{Z}/p^e\mathbb{Z}$ which span null(B) modulo p^{t+1}. Because $\mathbf{LCONNULL}_p \in \mathsf{FUL}_{p^e}$, which is low for $\mathsf{coMod}_p\mathsf{L}$, we have reduced to computing matrix products involving the matrix $\bar{B}^{(t)}$ in a $\mathsf{coMod}_p\mathsf{L}$ machine.

Matrix products in oracle models. The natural approach outlined in Buntrock *et al.* [4] for evaluating the coefficients of an iterated matrix product $M_1M_2\cdots M_{\mathrm{poly}(n)}$ modulo k — *i.e.* as a $\#\mathsf{L}_k$ function — requires access to individual coefficients at any given step of the algorithm. One simulates a branching program with nondeterministic choices, in which the matrices act as transition

functions on the row-positions of a vector $\mathbf{v}_\tau \in (\mathbb{Z}/kZ)^n$, to obtain a new vector $\mathbf{v}_{\tau+1}$. To evaluate the (h,j)-coefficient of the matrix product, we count the number of computational branches which end at a the hth row, given an initial vector $\mathbf{v}_0 = \hat{\mathbf{e}}_j$: we do this by accepting all branches which end with the row position h, and rejecting all others. This approach requires only non-deterministic selection of row-positions, logarithmic space to record the row-positions, and the ability query individual coefficients of the matrices being multiplied. When the matrices M_j are specified as part of the input, or more generally for any problem reduced *projectively* to matrix products (meaning that the matrices involved have coefficients which are either constants or taken from the input tape), the algorithm to evaluate the matrix products is straightforward; more generally, for any class C which is low for $\mathsf{coMod}_k\mathsf{L}$ (*e.g.* $C = \mathsf{FUL}_k$), we may compute any matrix product in $\mathsf{coMod}_k\mathsf{L}$ where the coefficients are obtained from the input by may be obtained by queries to C oracles.

We may use these observations to reduce $\mathbf{LCONNULL}_{p^e}$ to matrix products modulo p^e. In the recursive reduction for prime powers outlined above to $\mathbf{LCONNULL}_p$, every step is projective except for the matrix multiplications, and the problem of finding null spaces modulo p for the matrices $\bar{B}^{(t)}$ (which are not themselves part of the input). The columns of $\bar{B}^{(t)}$ are either columns of $\hat{B}_t$ (which are themselves the result of integer division of columns of B by p^t, this dividend being bounded by a constant) or are integer vectors of the form $B\mathbf{V}^{(t)}/p^t$. The coefficients of $B\mathbf{V}^{(t)}$ are computable as a matrix product, and thus may be computed in $\#\mathsf{L}_{p^e}$ from B itself and $\mathbf{V}^{(t)}$; provided a $\#\mathsf{L}_{p^e}$ oracle, we may then obtain those coefficients and divide them by p^t in NC^1. By Corollary 13, we have $\#\mathsf{L}_{p^e} \subseteq \mathsf{FUL}_p$, which is low for $\mathsf{coMod}_p\mathsf{L}$. We therefore have a $\mathsf{coMod}_p\mathsf{L}$-reduction from computing a basis for $\mathrm{null}(B)$ modulo p^{t+1} to computing the basis $\mathbf{V}_1^{(t)}, \ldots, \mathbf{V}_{N_t}^{(t)}$ modulo p^t. We may then carry out the recursive reduction to obtain a FUL_{p^e}-reduction from $\mathbf{LCONNULL}_{p^e}$ to iterated matrix products, via $\mathbf{LCONNULL}_p$; the number of vectors $\mathbf{V}_j^{(e)}$ in the generating set will, by induction, be $N_e \leqslant n + 2n + \cdots + 2^{e-1}n \leqslant p^e n \in O(n)$.

An important feature the recursive reduction described above is that the exponent e is itself a constant. The $\#\mathsf{L}_{p^e}$ oracles to compute coefficients of $\bar{B}^{(e-1)}$ require access to the coefficients of vectors $V_j^{(e)}$, which in turn will require $\#\mathsf{L}_{p^e}$ oracles to compute coefficients of $\bar{B}^{(e-2)}$, and so on. This is a sequential reduction, and the space resources can be described straightforwardly using a stack model of the work tape: each nested $\#\mathsf{L}_{p^e}$ oracle is simulated as a FUL_{p^e} subroutine which is allocated $O(\log|B|) = O(\log(n))$ space on the tape (where $|B| \in O(n^2)$ is the size of the input matrix after reduction modulo p^e), and which makes further recursive calls to FUL_{p^e} subroutines which do likewise, down depth at most e. The space resources then scale as $O(e\log(n))$; in our setting of a constant modulus, the space requirements are then $O(\log(n))$.

Consider a nondeterministic logspace machine with alphabet $\bar{\Sigma} = \Sigma \cup \{\bullet\}$ for $\Sigma = \{0, \ldots, p^e - 1\}$. Using a FUL_{p^e}-reduction to reduce $\mathbf{LCONNULL}_{p^e}$ for prime powers p^e to computing coefficients of matrix products, we may test equality of individual coefficients against some reference value $b \in \bar{\Sigma}$ provided as input. Therefore:

Lemma 15. $\mathbf{LCONNULL}_{p^e} \in \mathsf{F}\cdot\mathsf{coMod}_p\mathsf{L}$.

4.2 Completeness results for arbitrary constant moduli

The above suffices to show that $\mathbf{LCON}_k$, $\mathbf{LCONX}_k$, and $\mathbf{LCONNULL}_k$ are complete problems for $\mathsf{coMod}_k\mathsf{L}$ and $\mathsf{F{\cdot}coMod}_k\mathsf{L}$, as we now show. We consider nondeterministic logspace machines operating on an alphabet $\bar{\Sigma}_k = \Sigma_k \cup \{\bullet\}$, where Σ_k is the set of integers $0 \leqslant r < k$. For the function problems $\mathbf{LCONNULL}_k$ and $\mathbf{LCONX}_k$, we wish respectively to compute

- a function $\mathcal{N}_k : \Sigma_k^{n^2} \to \Sigma_k^{Nn}$ for $N \in O(n)$ such that $\mathcal{N}_k(B)$ is a sequence of vectors $(\mathbf{Z}_0, \mathbf{Z}_1, \ldots, \mathbf{Z}_{N-1})$ which generate $\mathrm{null}(B)$ in $\mathbb{Z}/k\mathbb{Z}$; and
- a partial function $\mathcal{S}_k : \Sigma^{n^2+n} \rightharpoonup \Sigma^n$ such that $(A, \mathbf{y}) \in \mathrm{dom}(\mathcal{S}_k)$ if and only if there exists a solution $\mathbf{x}$ to the system $A\mathbf{x} \equiv \mathbf{y} \pmod{k}$, in which case $\mathcal{S}_k(A, \mathbf{y})$ is such a solution.

Following [1, Lemma 5.3], we may reduce $\mathbf{LCON}_k$ and $\mathbf{LCONX}_k$ for $k \geqslant 2$ to $\mathbf{LCONNULL}_k$, as follows. Suppose $A\mathbf{x} \equiv \mathbf{y} \pmod{k}$ has solutions. Consider $B = [\,A \,|\, \mathbf{y}\,]$: then there are solutions to the equation $B\bar{\mathbf{x}} \equiv 0 \pmod{k}$. In particular, there will be a solutions $\bar{\mathbf{x}} = \mathbf{x} \oplus x_{n+1}$ in which $x_{n+1} = -1$, and more generally in which x_{n+1} is coprime to k. Conversely, if there is such a solution $\bar{\mathbf{x}}$ to $B\bar{\mathbf{x}} \equiv 0 \pmod{k}$, we may take $\alpha \equiv -x_{n+1}^{-1} \pmod{k}$ and obtain $A(\alpha\mathbf{x}) \equiv -\alpha x_{n+1}\mathbf{y} \equiv \mathbf{y} \pmod{k}$. To determine whether $A\mathbf{x} \equiv \mathbf{y} \pmod{k}$ has solutions, or to construct a solution, it thus suffices to compute a basis for the nullspace of B, and determine from this basis whether any of the vectors $\bar{\mathbf{x}} \in \mathrm{null}(B)$ have a final coefficient coprime to k; if so, the remainder of the coefficients of $\bar{\mathbf{x}}$ may be used to compute a solution to the original system.

In the special case $k = p^e$ of a prime power, coprimality to k simply entails that k is not divisible by p. To solve $\mathbf{LCON}_{p^e}$ and $\mathbf{LCONX}_{p^e}$, we compute individually the final coefficients of the vectors $(\mathbf{Z}_0, \mathbf{Z}_1, \mathbf{Z}_2, \ldots) = \mathcal{N}_{p^e}(B)$ for $B = [\,A \,|\, \mathbf{y}\,]$, searching for an index $1 \leqslant h \leqslant N_e$ for which the dot product $\hat{\mathbf{e}}_{n+1} \cdot \mathbf{Z}_h$ is not divisible by p. Without loss of generality, we select the minimum such h: the search problem can be formulated as a truth-table reduction on divisibility tests of these coefficients by p. Both the reduction and the divisibility test are feasible for $\mathsf{coMod}_p\mathsf{L}$; we may suppose that this reduction and test are performed by a FUL_p oracle so that the outcome is explicitly recorded on the work tape in a single branch mod p. If there is no such index h, we indicate that no solution exists by accepting unconditionally, indicating either a *no* instance of $\mathbf{bits}(\mathcal{S}_k)$ or of $\mathbf{LCON}_k$ on a $\mathsf{coMod}_p\mathsf{L}$ machine. Otherwise, there exists a solution to the linear congruence. To indicate for $\mathbf{LCON}_{p^e}$ that $(A, \mathbf{y})$ is a *yes* instance on a $\mathsf{coMod}_p\mathsf{L}$ machine, we reject on all computational branches to make the number of accepting branches zero modulo p. To solve $\mathbf{bits}(\mathcal{S}_k)$, we compute the minimum index h and the coefficient $\hat{\mathbf{e}}_{n+1}^{\mathsf{T}}\mathbf{Z}_h$, which we store on the work tape in binary. We then compute $\alpha \equiv -x_{n+1}^{-1} \pmod{p^e}$, and then obtain the coefficients of $\alpha\mathbf{Z}_h$, which we compare to input coefficients, rejecting (to indicate a *yes* instance) if the coefficients match, and accepting (to indicate a *no* instance) otherwise. Therefore:

Lemma 16. $\mathbf{LCON}_{p^e} \in \mathsf{coMod}_p\mathsf{L}$ *and* $\mathbf{LCONX}_{p^e} \in \mathsf{F{\cdot}coMod}_p\mathsf{L}$.

As we remarked in Section 2, we may solve $\mathbf{LCON}_k$ for arbitrary moduli $k = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$ by reduction to the problems $\mathbf{LCON}_{p_j^{e_j}}$ for $1 \leqslant j \leqslant \ell$; the same is true for $\mathbf{LCONX}_k$ and $\mathbf{LCONNULL}_k$. Let $q_j = p_j^{e_j}$ for the sake of

brevity. For $\mathbf{LCON}_k$, we simply have $\mathbf{LCON}_k = \mathbf{LCON}_{q_1} \cap \cdots \cap \mathbf{LCON}_{q_\ell}$. For $\mathbf{LCONNULL}_k$ and $\mathbf{LCONX}_k$, let $\mathbf{congbits}(f, q_j)$ be the decision problem of determining for inputs $(x, h, b) \in \Sigma_k^* \times \mathbb{N} \times \bar{\Sigma}_k$ whether $x \in \mathrm{dom}(f)$, and (this being granted) whether either $f(x)_h \equiv b \pmod{q_j}$ for $b \neq \bullet$ or $f(x)_h = \bullet = b$.

- Clearly $\mathbf{bits}(\mathcal{S}_k)$ is the intersection of the problems $\mathbf{congbits}(\mathcal{S}_k, q_j)$ for $1 \leqslant j \leqslant \ell$. We may show $\mathbf{congbits}(\mathcal{S}_k, q_j) \in \mathsf{coMod}_{q_j}\mathsf{L}$ for each $1 \leqslant j \leqslant \ell$, as follows. For $b \in \Sigma_k$, we may expand b in binary on the work tape and evaluate its reduction $0 \leqslant b' < q_j$ modulo a given prime power q_j; for $b = \bullet$ we simply let $b' = \bullet$ as well, so that $b' \in \bar{\Sigma}_{q_j}$. We perform a similar reduction for each coefficient in $(A, \mathbf{y})$ to obtain an input $(A', \mathbf{y}')$ with coefficients in Σ_{q_j}. Then we may simulate a $\mathsf{coMod}_{p_j}\mathsf{L}$ machine to decide whether $((A', \mathbf{y}'), h, b') \in \mathbf{bits}(\mathcal{S}_{q_j})$. Thus $\mathbf{bits}(\mathcal{S}_k) \in \mathsf{coMod}_k\mathsf{L}$.

- To show $\mathbf{bits}(\mathcal{N}_k) \in \mathsf{coMod}_k\mathsf{L}$, we follow the reduction of McKenzie and Cook in Ref. [1, Theorem 8.3]. Given vectors $\mathbf{X}_1^{(q_j)}, \ldots, \mathbf{X}_{N_j}^{(q_j)}$ spanning the nullspace of B modulo q_j for each $1 \leqslant j \leqslant \ell$, the nullspace of B modulo k is spanned over the integers modulo k by the vectors

$$\tfrac{k}{q_1}\mathbf{X}_1^{(q_1)},\ \ldots,\ \tfrac{k}{q_1}\mathbf{X}_{N_1}^{(q_1)},\ \tfrac{k}{q_2}\mathbf{X}_1^{(q_2)},\ \ldots,\ \tfrac{k}{q_j}\mathbf{X}_h^{(q_j)},\ \ldots,\ \tfrac{k}{q_\ell}\mathbf{X}_{N_\ell}^{(q_\ell)}\,. \tag{7}$$

  (We omit the vectors $k\hat{\mathbf{e}}_h$ included by Ref. [1], as these are congruent to $\mathbf{0}$ in $\mathbb{Z}/k\mathbb{Z}$.) Let $\mathbf{Z}_h$ be the list of such vectors, for $0 \leqslant h < N_1 + \cdots + N_\ell$: we define $\mathcal{N}_k$ for k divisible by more than one prime to produce this sequence of vectors as output. Notice that each $\mathbf{Z}_h$ is

  - congruent to $\mathbf{0}$ modulo q_j for every $j \neq 1$ for $0 \leqslant h < N_1$,
  - congruent to $\mathbf{0}$ modulo q_j for every $j \neq 2$ for $N_1 \leqslant h < N_1 + N_2$, and
  - generally, congruent to $\mathbf{0}$ modulo q_j for every $j \geqslant 1$, except for the index j for which $M_{j-1} \leqslant h < M_j$, where for the sake of brevity we write $M_j = \sum_{t=1}^{j} N_t$.

  We may then reduce $\mathbf{bits}(\mathcal{N}_k)$ to testing the congruence of coefficients of $\mathbf{Z}_h$ with 0 modulo q_j for all prime powers for which $h < M_{j-1}$ or $h \geqslant M_j$, and testing congruence with the coefficients of $\frac{k}{q_j}\mathbf{X}_{h-M_j+1}^{(q_j)}$ otherwise. These congruences modulo each prime power q_j can again be evaluated in $\mathsf{coMod}_{q_j}\mathsf{L}$ algorithm for $\mathbf{congbits}_j(\mathcal{N}_k)$, using the NC^1 reduction to $\mathbf{bits}(\mathcal{N}_{q_j})$ as above.

The above reductions suffice to show:

Theorem 17. *For all $k \geqslant 2$, we have $\mathbf{LCONNULL}_k$, $\mathbf{LCONX}_k \in \mathsf{F}{\cdot}\mathsf{coMod}_k\mathsf{L}$ and $\mathbf{LCON}_k \in \mathsf{coMod}_k\mathsf{L}$.*

Finally, note that one may also $\mathbf{LCON}_{p_j}$ to $\mathbf{LCON}_k$, for any prime p_j dividing k, by considering the feasibility of the congruence

$$(kA/p_j)\,\mathbf{x} \;\equiv\; k\mathbf{y}/p_j \pmod{k}, \tag{8}$$

which is equivalent to $A\mathbf{x} \equiv \mathbf{y} \pmod{p_j}$. By Propositions 2–4, all problems in $\mathbf{LCON}_k$ may be reduced to solving some instances of $\mathbf{LCON}_{p_j}$ for each $1 \leqslant j \leqslant \ell$: then $\mathbf{LCON}_k$ is $\mathsf{coMod}_k\mathsf{L}$-hard. Similar remarks apply to $\mathbf{LCONX}_k$ and $\mathbf{LCONNULL}_k$. Therefore:

Theorem 18. *For all $k \geqslant 2$, $\mathbf{LCON}_k$ is $\mathsf{coMod}_k\mathsf{L}$-complete, and $\mathbf{LCONNULL}_k$ and $\mathbf{LCONX}_k$ are $\mathsf{F}{\cdot}\mathsf{coMod}_k\mathsf{L}$-complete.*

5 Further Remarks

The above analysis was motivated by observing that the reduction of McKenzie and Cook [1] for **LCONX** and **LCONNULL** (which take the modulus k as input, as a product of prime powers $p_j^{e_j} \in O(n)$) was very nearly a projective reduction to matrix multiplication, and that it remained only to find a way to realize the division by prime powers p^t involved in the reduction to $\mathbf{LCONNULL}_p$. By showing that logspace counting oracles modulo p^e could be simulated by a $\mathsf{coMod}_p\mathsf{L}$ machine, using the function class FUL_k as a notion of naturally simulatable oracles for the classes $\mathsf{Mod}_k\mathsf{L}$ and $\mathsf{coMod}_k\mathsf{L}$, the containments of Theorem 17 became feasible.

Extending the definition of $\mathbf{bits}(f)$ to accomodate partial functions in the is crucial to our result that $\mathbf{LCONX}_k \in \mathsf{F}{\cdot}\mathsf{coMod}_k\mathsf{L}$, in the sense that there is no obvious way to extend the algorithm to decide $\mathbf{bits}(\bar{S}_k)$ for any unambiguous extension of $\mathcal{S}_k$ to infesible systems of equations, *e.g.* by accepting on some symbol "!" if and only if there is no solution to a congruence provided as input. Such an algorithm would be a signficant result, as it would follow that $\mathbf{LCON}_k \in \mathsf{Mod}_k\mathsf{L}$, thereby showing that this class is closed under complements.

In the recursive reduction for $\mathbf{LCONNULL}_{p^e}$, the fact that $e \in O(1)$ is essential not only for the logarithmic bound on the work tape, but also for the running time on a $\mathsf{coMod}_k\mathsf{L}$ machine to be polynomial. The FUL_p machines used to implement the $\#\mathsf{L}_{p^e}$ oracles, from the constructions of Theorem 10 and Lemma 12, implicitly involve many repeated simulations of $\mathsf{coMod}_p\mathsf{L}$ machines ($p^e/p = p^{e-1}$ times each) to decide equality of counting functions with residues $0 \leqslant r < p^e$: this contributes to a factor of overhead growing quickly with e. Therefore our results are mainly of theoretical interest, characterizing the complexity of these problems with respect to logspace reductions. It is reasonable to ask if there is an algorithm on a $\mathsf{coMod}_p\mathsf{L}$ machine for $\mathbf{LCONNULL}_{p^e}$, whose running time grows slowly with e.

Given the natural role of the class FUL_{p^e} in simulating of $\#\mathsf{L}_k$ oracles, one might ask *e.g.* whether the characteristic function of $\mathbf{LCON}_k$ is contained in FUL_k. It is interesting to consider the difference between such potential containments, and those proven as Theorem 17. We first note an alternative characterization of $\mathsf{coMod}_k\mathsf{L}$:

Proposition 19. *For every $k \geqslant 2$, $L \in \mathsf{coMod}_k\mathsf{L}$ if and only if there exists $\varphi \in \#\mathsf{L}$ such that $x \in L$ if and only if $\varphi(x)$ is* coprime *to k.*

Proof. For $k = p_1^{e_1}p_2^{e_2}\cdots p_\ell^{e_\ell}$ as usual, we have $L \in \mathsf{coMod}_k\mathsf{L}$ if and only if $L = L_1 \cap L_2 \cap \cdots \cap L_\ell$ for languages $L_j \in \mathsf{coMod}_{p_j}\mathsf{L} = \mathsf{Mod}_{p_j}\mathsf{L}$ by Propositions 3 and 5. Let $\mathbf{T}_1, \ldots, \mathbf{T}_\ell$ be nondeterministic logspace machines such that $\mathbf{T}_j$ accepts on input x with a number of branches not divisible by p_j if $x \in L_j$, and with zero branches modulo $p_j^{e_j}$ otherwise. Using a similar construction to that of Lemma 10 for the square-free case, we may obtain a *single* nondeterministic logspace machine $\mathbf{T}$ which accepts on a number of branches not divisible by p_j if $x \in L_j$, and on a number of branches equivalent to 0 mod p_j otherwise. If $x \in L$, then the number of branches on which $\mathbf{T}$ accepts is not divisible by any prime p_j, which means that it is coprime to k; otherwise, there exists some prime p_j which divides the number of accepting branches, so that the number of branches is not coprime to k. □

This, in turn, suggests a characterization for $\mathsf{F{\cdot}coMod}_k\mathsf{L}$ in the same vein as the definition of FUL_k:

Proposition 20. *For all $k \geqslant 2$, $\mathsf{F{\cdot}coMod}_k\mathsf{L}$ consists of those (partial) functions f computable by a nondeterministic logspace machine which* **(a)** *for inputs $x \in \mathrm{dom}(f)$, computes $f(x)$ on its output tape in some number $\varphi(x, f(x))$ of its accepting branches which is coprime to k, and* **(b)** *for each output string $y \neq f(x)$ (or for any string y, in the case $x \notin \mathrm{dom}(f)$), computes y on its output tape on some number $\varphi(x, y)$ such that $\gcd\big(\varphi(x, y), k\big) > 1$. Furthermore, we may require without loss of generality for all $y \in \Sigma^*$ that either $\varphi(x, y) \equiv 1 \pmod{k}$, or $\gcd\big(\varphi(x, y), k\big)$ is a product of some of the maximal prime powers $p_j^{e_j}$ which divide k.*

Proof. Let $k = p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$ be the factorization of k into its prime power factors. For $f \in \mathsf{F{\cdot}coMod}_k\mathsf{L}$, consider the $\mathsf{coMod}_k\mathsf{L}$ machine $\mathbf{T}$ for deciding $\mathbf{bits}(f)$ with the characteristics described in Proposition 19. Using a construction similar to that of Lemma 12, we may construct machines $\mathbf{C}_h$ which simulate $\mathbf{T}$ on inputs (x, h, b) for each $b \in \Sigma \cup \{\bullet\}$, writing the symbol b on the output tape. For $x \notin \mathrm{dom}(f)$, each possible output is written to the output tape in some number of branches which has a non-trivial common divisor with k. Otherwise, for $x \in \mathrm{dom}(f)$, this machine writes $f(x)_h$ on the tape in some number of branches coprime to k, and every $b \neq f(x)_h$ on the tape some number of branches which has prime divisors in common with k, by hypothesis. Still following Lemma 12, consider a machine $\mathbf{C}$ simulating each $\mathbf{C}_h$ in turn for $1 \leqslant h \leqslant N(x)$ up to some upper bound $|f(x)| \leqslant N(x) \in \mathrm{poly}(|x|)$ or until we encounter symbols $f(x)_h = \bullet$. A string $y \in \Sigma^*$ written to the output tape occurs in a number of accepting branches which is coprime to k if and only if each character of y occurs a number of times coprime to k, which is to say if $y = f(x)$.

The stricter characterization of the number of branches in which each $y = f(x)$ or $y \neq f(x)$ is accepted may be obtained as follows. By Proposition 3, consider functions $f_j \in \mathsf{F{\cdot}coMod}_{p_j}\mathsf{L}$ such that $\mathbf{bits}(f) = \mathbf{bits}(f_1) \cap \cdots \cap \mathbf{bits}(f_\ell)$. Using Theorem 10 and Lemma 12, consider $\mathsf{FUL}_{p_j{}^{e_j}}$ machines $\mathbf{U}_j$ which compute f_j for each $1 \leqslant j \leqslant \ell$. By a similar construction to Theorem 10, we may obtain a machine $\tilde{\mathbf{U}}$ which writes $f(x)$ on the tape in a number of branches which is equivalent to 1 modulo every prime power $p_j^{e_j}$, and which writes any $y \neq f(x)$ on the output tape a number of times which is equivalent to 0 modulo one or more powers $p_j^{e_j}$ (but equivalent to 1 for the other prime powers $p_h^{e_h}$). Then $f(x)$ is written on the output tape on one branch modulo k; the other strings $y \neq f(x)$ occur on the output tape a number of times which is divisible by some maximal prime power divisors $p_j^{e_j}$, but which is coprime to the other maximal prime power divisors $p_h^{e_h}$.

To show the converse, *i.e.* that the functions f computable by such logspace nondeterministic machines $\mathbf{C}$ are indeed in $\mathsf{F{\cdot}coMod}_k\mathsf{L}$, simply consider a machine $\tilde{\mathbf{T}}$ which takes a tuple (x, h, b) as input, and simulates a machine $\mathbf{C}$ as described above long enough to compute $f(x)_h$, accepting unconditionally. Then the number of branches on which $\tilde{\mathbf{T}}$ accepts is coprime to k if and only if $x \in \mathrm{dom}(f)$ and $f(x)_j = b$, by construction. By Proposition 19, it follows that $\mathbf{bits}(f) \in \mathsf{coMod}_k\mathsf{L}$. □

The definition of FUL_k differs from the above characterization of $\mathsf{F{\cdot}coMod}_k\mathsf{L}$ by the further requirement that, for a FUL_k machine computing some function f,

output strings $y \neq f(x)$ must occur in *zero* branches mod k and not just in a number of branches which has maximal prime power factors in common with k. Thus, we see that Definition IV does not result in a class which is entirely different in signficance from $\mathsf{F{\cdot}coMod}_k\mathsf{L}$, even for k composite.

There is no obvious way to bridge the gap between the definition of FUL_k, and the characterization of $\mathsf{F{\cdot}coMod}_k\mathsf{L}$ given by Proposition 19. Of course, $\mathbf{LCON}_k$ can be solved in FUL_k if and only if $\mathsf{FUL}_k = \mathsf{F{\cdot}coMod}_k\mathsf{L}$. As FUL_k is low for $\mathsf{coMod}_k\mathsf{L}$, this would imply $\mathsf{coMod}_k\mathsf{L}$ is closed under logspace Turing reductions, and that therefore $\mathsf{Mod}_k\mathsf{L} = \mathsf{coMod}_k\mathsf{L}$. Furthermore, by Proposition 3 and Theorem 10, $\mathsf{FUL}_k = \mathsf{F{\cdot}coMod}_k\mathsf{L}$ would imply a surprising collapse of logspace mod classes beneath $\mathsf{coMod}_k\mathsf{L}$: for any distinct prime divisors p_h, p_j of k we would have $\mathsf{FMod}_{p_h}\mathsf{L} \subseteq \mathsf{FMod}_k\mathsf{L} = \mathsf{FUL}_k \subseteq \mathsf{FUL}_{p_j} = \mathsf{FMod}_{p_j}\mathsf{L}$, and in particular $\mathsf{Mod}_{p_h}\mathsf{L} = \mathsf{Mod}_{p_j}\mathsf{L}$. The converse, that $\mathsf{Mod}_{p_h}\mathsf{L} = \mathsf{Mod}_{p_j}\mathsf{L}$ for all primes dividing k only if $\mathsf{FUL}_k = \mathsf{F{\cdot}coMod}_k\mathsf{L}$, is trivial. A similar collapse would occur even if the characteristic function of $\mathbf{LCON}_k$ could be computed in $\mathsf{F{\cdot}coMod}_k\mathsf{L}$; not only would this indicate that $\mathsf{coMod}_k\mathsf{L}$ is closed under containment, but also under oracles, as it would allow simulation of $\mathsf{F{\cdot}coMod}_k\mathsf{L}$ oracles in a way much similar to the simulation of FUL_k oracles by $\mathsf{coMod}_k\mathsf{L}$ machines (where a collection of branches having the same tape-conents are insignificant if the number of branches has prime power divisors in common with k, although not necessarily divisible by k). If we suppose that FUL_k, $\mathsf{FMod}_k\mathsf{L}$, and $\mathsf{F{\cdot}coMod}_k\mathsf{L}$ are distinct for any $k \geqslant 2$ divisible by two or more primes, it would be interesting to characterize FUL_k as a subclass of $\mathsf{FMod}_k\mathsf{L} \cap \mathsf{F{\cdot}coMod}_k\mathsf{L}$.

Acknowledgements

This work was supported by the EC project QCS. I would like to thank Bjarki Holm for feedback in the early stages of work on this problem, and for indicating helpful references in the literature on the variable modulus problem **LCON**.

Contact

Please send questions or feedback to [`niel.debeaudrap@gmail.com`].

References

[1] P. McKenzie, S. Cook. *The parallel complexity of abelian permutation group problems.* SIAM Journal of Computing **16** (pp. 880–909), 1987.

[2] V. Arvind, T. C. Vijayaraghavan. *Classifying Problems on Linear Congruences and Abelian Permutation Groups Using Logspace Counting Classes.* Computational Complexity **19** (pp. 57–98), 2010.

[3] J. Köbler, S. Toda. *On the Power of Generalized MOD-Classes.* Mathematical Systems Theory **29** (pp. 33–46), 1996.

[4] G. Buntrock, C. Damm, U. Hertrampf, C. Meinel. *Structure and importance of logspace-MOD classes.* Theory of Computing Systems **25** (pp. 223-237), 1992.

[5] U. Hertrampf, S. Reith, H. Vollmer. *A note on closure properties of logspace MOD classes.* Information Processing Letters **75** (pp. 91–93), 2000.

[6] R. Beigel, J. Gill, U. Hertrampf. *Counting classes: Thresholds, parity, mods, and fewness.* Proc. STACS 90, Lecture Notes in Computer Science **415** (pp. 49–57), 1990.

[7] A. Borodin, J. von zur Gathen, J. Hopcroft. *Fast parallel matrix and GCD computations.* 23rd Annual Symposium on Foundations of Computer Science (pp. 65–71), 1982.

[8] K. Mulmuley. *A fast parallel algorithm to compute the rank of a matrix over an arbitrary field.* Combinatorica **7** (pp. 101–104), 1987.

[9] S. J. Berkowitz. *On computing the determinant in small parallel time using a small number of processors.* Information Processing Letters **18** (pp. 147–150), 1984.

[10] J. von zur Gathen. "Parallel linear algebra". In J. H. Reif, editor, *Synthesis of parallel algorithms* (pp. 573–617). Morgan Kaufmann, 1993.