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

Niel de Beaudrap[†]

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 problem is considered modulo an arbitrary constant $k \geqslant 2$. These problems are known to be complete for the logspace modular counting classes $\mathsf{Mod}_k\mathsf{L} = \mathsf{coMod}_k\mathsf{L}$ in special case that k is prime [7]. By considering variants of standard logspace function classes — related to #L and functions computable by UL machines, but which only characterize the number of accepting paths modulo k — we show that these problems of linear algebra are also complete for $\mathsf{coMod}_k\mathsf{L}$ for any constant $k \geqslant 2$.

Our results are obtained by defining a class of functions FUL_k which are low for $\mathsf{Mod}_k\mathsf{L}$ and $\mathsf{coMod}_k\mathsf{L}$ for $k \geqslant 2$, using ideas similar to those used in the case of k prime in Ref. [7] to show closure of $\mathsf{Mod}_k\mathsf{L}$ under NC^1 reductions (including $\mathsf{Mod}_k\mathsf{L}$ oracle reductions). In addition to the results above, we briefly consider the relationship of the class FUL_k for arbitrary moduli k to the class $\mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$ of functions whose output symbols are verifiable by $\mathsf{coMod}_k\mathsf{L}$ algorithms; and consider what consequences such a comparison may have for oracle closure results of the form $\mathsf{Mod}_k\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}} = \mathsf{Mod}_k\mathsf{L}$ for composite k.

Key words and phrases: Modular arithmetic, linear congruence, logarithmic workspace

---

1   Introduction

Solving a system of linear equations, or determining that it has no solution, 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 [10]. 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. For an arbitrary constant $k \geqslant 2$, we consider the difficulty of *deciding feasibility of linear congruences modulo k* (LCON_k) and *computing solutions to linear congruences modulo k* (LCONX_k). These are special cases of the problems LCON and LCONX defined by McKenzie and Cook [10], in which $k \in O(n)$ is taken as part of the input and represented by its prime-power factors $p_1^{e_1} p_2^{e_2} \cdots p_\ell^{e_\ell}$. Setting k to a fixed constant is a natural, if slightly restrictive, special case.

Arvind and Vijayaraghavan [3] define the class $\mathsf{ModL} \subseteq \mathsf{NC}^2$ as a logspace analogue the class ModP defined by Köbler and Toda [9]. They show that LCON is hard for ModL under P-uniform NC^1 reductions, and contained in $\mathsf{L}^{\mathsf{ModL}}/\mathrm{poly} = \mathsf{L}^{\#\mathsf{L}}/\mathrm{poly}$. 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 $j \geqslant 0$.

Buntrock *et al.* [7] show — for the special case of k prime — that determining the feasibility of systems of linear equations is complete for the complexity classes $\mathsf{Mod}_k\mathsf{L}$ which generalize $\oplus\mathsf{L}$, decidable by logspace nondeterministic Turing machines which can distinguish between having a number of accepting paths which are either zero or nonzero *mod k*. These results together with those of Ref. [3] suggest that the difficulty of solving linear equations over integer matrices is sensitive to the presence and the prime-power factorization of the modulus involved; one might suppose that LCON_k is particularly tractable for arbitrary $k \geqslant 2$. For k prime, Ref. [7] also shows that $\mathsf{Mod}_k\mathsf{L} = \mathsf{coMod}_k\mathsf{L}$: the techniques of Ref. [7] may more naturally be interpreted as proving that for k prime, LCON_k is complete for $\mathsf{coMod}_k\mathsf{L}$. Also implicit in Ref. [7] is that LCON_k is $\mathsf{coMod}_k\mathsf{L}$-hard for all $k \geqslant 2$ under NC^1 reductions. This suggests the question: for an *arbitrary* modulus k, what is the precise relationship of LCON_k to the classes $\mathsf{coMod}_k\mathsf{L}$?

We show that the proof $\mathrm{LCON} \in \mathsf{NC}^3$ by McKenzie and Cook [10] may be adapted prove $\mathrm{LCON}_k \in \mathsf{coMod}_k\mathsf{L}$, using fast parallel algorithms for matrix multiplication and computing rank modulo the prime factors of the modulus. For a constant prime modulus p, the latter problems are complete for $\mathsf{coMod}_p\mathsf{L}$: however, as is typical of counting problems, they are "evaluated" in the number of accepting branches of the computation, which is an obstacle to performing operations such as integer division required by the McKenzie–Cook algorithm. We overcome this obstacle by describing a class FUL_k of machines which evaluate functions on the work-tape, and which may be simulated in mod-logspace computations.[*] This simulation uses techniques similar to those used Buntrock *et al.* [7] for p prime, to show closure of the class $\mathsf{Mod}_p\mathsf{L}$ under NC^1 reductions. We then describe, for p^e a prime power, a FUL_{p^e} algorithm to solve the problem $\mathrm{LCONNULL}_{p^e}$ of computing a spanning set for a basis of the nullspace of a matrix modulo p^e. It follows that LCON_k is $\mathsf{coMod}_k\mathsf{L}$-complete, and both LCONX_k and $\mathrm{LCONNULL}_k$ are

---

[*]For logspace nondeterministic machines, we adopt the Russo–Simon–Tompa oracle model [12], in which nondeterministic machines are not allowed to make nondeterministic transitions while it writes on the oracle tape (i. e. oracle queries must be determined by the contents of the input and work tapes before the query has started being written).

$\mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$-complete (this class being the functional analogue of $\mathsf{coMod}_k\mathsf{L}$), for any constant $k \geqslant 2$.

Finally, for arbitrary moduli k, we consider the relationship of the class FUL_k to the function class $\mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$, and consider what insights it may suggest for oracle closure results of the form $\mathsf{Mod}_k\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}} = \mathsf{Mod}_k\mathsf{L}$ for k composite, where this problem remains open; and to consider what light it sheds on attempts to resolve it [13].

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 k which is a prime power, we typically write p^e instead, for p some prime and $e \geqslant 1$ some positive integer. We will suppose that the reader is familiar with the basic properties of the function classes $\#\mathsf{L}$ [7] and GapL [2].

We consider the complexity of the following problems, which are named in analogy to problems considered by McKenzie and Cook [10]: For an $m \times n$ integer matrix A and vector $\mathbf{y} \in \mathbb{Z}^m$ provided as input, we define the following problems:

Problem 2.1 (LCON$_k$). Determine whether $A\mathbf{x} \equiv \mathbf{y} \pmod{k}$ has solutions for $\mathbf{x} \in \mathbb{Z}^n$.

Problem 2.2 (LCONX$_k$). Output a solution to the congruence $A\mathbf{x} \equiv \mathbf{y} \pmod{k}$, if one exists.

Problem 2.3 (LCONNULL$_k$). Output vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ which span the solutions to $A\mathbf{x} \equiv \mathbf{0} \pmod{k}$.

Without loss of generality, we may suppose $m = n$ by padding the matrix A. We wish to describe how these problems relate to the classes $\mathsf{coMod}_k\mathsf{L}$ for $k \geqslant 2$, which are the complements of the classes $\mathsf{Mod}_k\mathsf{L}$ defined by Buntrock *et al.* [7]. (Because the classes $\mathsf{coMod}_k\mathsf{L}$ are our principal concern, and because of the techniques used for completeness results in Ref. [7], we will present the preliminary definitions and results which we use in terms of these classes, rather than the complementary classes $\mathsf{Mod}_k\mathsf{L}$.)

Definition 2.4. 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 $\varphi(x) \equiv 0 \bmod k$ (respectively, $\varphi(x) \not\equiv 0 \bmod k$).

Note that the classes $\mathsf{Mod}_k\mathsf{L}$ and $\mathsf{coMod}_k\mathsf{L}$ remain the same if we substitute $\#\mathsf{L}$ in the definition above with GapL, as any function $g = f_1 - f_2 \in \mathsf{GapL}$, for functions $f_1, f_2 \in \#\mathsf{L}$, is congruent modulo k to $f_1 + (k-1)f_2 \in \#\mathsf{L}$. The following results are a synopsis of (the remark which follows) Ref. [7, Theorem 10]:

Proposition 2.5. *We may characterize* $\mathsf{coMod}_k\mathsf{L}$ *as the class of decision problems which are (logspace-uniform)* NC^1*-reducible to verifying matrix determinants mod k, or verifying coefficients of integer matrix products/inverses mod k. (The corresponding* falsification *problems are complete for* $\mathsf{Mod}_k\mathsf{L}$*.)*

Proposition 2.6. *For p prime,* LCON$_p$ *is complete for* $\mathsf{coMod}_p\mathsf{L}$ *under* NC^1 *reductions.*

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 equivalent to Lemma 6, Theorem 7, and Corollary 8 of Ref. [7] via logical complementation:

Proposition 2.7 (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 2.8 (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 2.9 (limited closure under complements). *For any prime p and $e \geqslant 1$, we have $\mathsf{coMod}_{p^e}\mathsf{L} = \mathsf{coMod}_p\mathsf{L} = \mathsf{Mod}_p\mathsf{L} = \mathsf{Mod}_{p^e}\mathsf{L}$.*

By the Chinese Remainder Theorem, 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 $\mathrm{LCON}_k \in \mathsf{coMod}_k\mathsf{L}$ if and only if $\mathrm{LCON}_{p^e} \in \mathsf{coMod}_{p^e}\mathsf{L} = \mathsf{coMod}_p\mathsf{L}$ by Proposition 2.7. (In fact, this suffices to show that $\mathrm{LCON}_k \in \mathsf{coMod}_k\mathsf{L}$ for all k square-free.)

We see from Propositions 2.6 and 2.9 that the case of a prime modulus is special. For p prime, Buntrock *et al.* also implicitly characterize the complexity of LCONX_p and $\mathrm{LCONNULL}_p$. 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, Reith and Vollmer [8], for a function $f : \Sigma^* \to \Sigma^*$ on some alphabet Σ, and for some symbol $\bullet \notin \Sigma$, we may define the decision problem

$$\mathrm{bits}(f) = \left\{ (x, j, b) \;\middle|\; \begin{array}{c} \text{either } j \leqslant |f(x)| \text{ and } b = f(x)_j \\ \text{or } j > |f(x)| \text{ and } b = \bullet \end{array} \right\}. \tag{2.1}$$

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

Definition 2.10. The class $\mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$ is the set of (partial) functions f such that $|f(x)| \in \mathrm{poly}(|x|)$ for all $x \in \mathrm{dom}(f)$, and for which $\mathrm{bits}(f) \in \mathsf{coMod}_k\mathsf{L}$. (We define the class $\mathsf{FMod}_k\mathsf{L}$ similarly.)

Then Ref. [7, Theorem 10] also implicitly shows:

Proposition 2.11. *For p prime, the problems LCONX_p and $\mathrm{LCONNULL}_p$ are complete for $\mathsf{FMod}_p\mathsf{L} = \mathsf{F}\cdot\mathsf{coMod}_p\mathsf{L}$ under NC^1 reductions.*

3 Natural function classes for modular logspace

We introduce two complexity classes in logarithmic space: a modular analogue of #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 $\mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$, and to each other in the case of a prime modulus. Some of these results may be regarded as encapsulating known techniques; we present them explicitly to simplify the presentation of the main results of the article.

The first class we define is a logspace variant of the class $\#_k\mathsf{P}$ described by Valiant [14, page 193]:

Definition 3.1. The function class $\#_k\mathsf{L}$ is the set of functions $f : \Sigma^* \to \{0, 1, \ldots, k-1\}$ such that there exists $\varphi \in \#\mathsf{L}$ such that $f(x) \equiv \varphi(x) \pmod{k}$.

Note that $\#_k\mathsf{L}$ is closed under addition, multiplication, and constant powers *modulo k* by virtue of similar closure results for $\#\mathsf{L}$ over the integers; it is closed under subtraction mod k as well, as $M - N \equiv M + (k-1)N \pmod{k}$. Thus, if we were to define a similar class $\mathsf{Gap}_k\mathsf{L}$ in terms of congruence mod k to functions $g \in \mathsf{GapL}$, we would obtain $\mathsf{Gap}_k\mathsf{L} = \#_k\mathsf{L}$. Note that by its definition, the functions $f \in \#_k\mathsf{L}$ are essentially those functions whose values a $\mathsf{coMod}_k\mathsf{L}$ algorithm can verify directly:

Lemma 3.2. *For any $k \geqslant 2$, $\#_k\mathsf{L} \subseteq \mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$.*

Proof. For $f \in \#_k\mathsf{L}$ such that $f : \Sigma^* \to \{0, \ldots, k-1\}$, let $\mathbf{T}$ be a nondeterministic Turing machine which accepts on inputs x with some number of branches $\varphi(x) \equiv f(x) \pmod{k}$. Consider a nondeterministic logspace machine $\mathbf{T}'$ acting on the alphabet $\Sigma^* \cup \{0, \ldots, k-1\}$ which takes input tuples $(x, y) \in \Sigma^* \times \{0, \ldots, k-1\}$. The machine $\mathbf{T}'$ reads y and branches $(k-1)y + 1$ times. In one of these branches, $\mathbf{T}'$ simulates $\mathbf{T}$ on x, accepting if and only if $\mathbf{T}$ does; in the other branches, it accepts unconditionally. The number of accepting branches is then $\varphi(x) + (k-1)y \equiv f(x) - y \pmod{k}$, so that $\mathbf{T}'$ accepts with 0 branches mod k on input (x, y) if and only if $f(x) = y$. Thus $\mathrm{bits}(f) \in \mathsf{coMod}_k\mathsf{L}$. $\qquad\square$

The technique here is identical to that of Buntrock *et al.* [7]; one might describe Ref. [7, Theorem 10] as showing that evaluating matrix determinants modulo k, and evaluating coefficients of products/inverses of integer matrices modulo k, are contained in $\#_k\mathsf{L}$.

We are interested in logspace machines which compute $\#_k\mathsf{L}$ 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 $\#_k\mathsf{L}$.

Definition 3.3. 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 the class FUL of functions computable by nondeterministic logspace machines with a single accepting branch. This is in turn analogous to the class UPF described by Beigel, Gill, and Hertrampf [4], of functions which may be computed by a nondeterministic polynomial time Turing machine without affecting the number of accepting branches of that machine. Note that for a FUL_k machine $\mathbf{U}$, what is written on the output tape in many branches (perhaps even the vast majority of them) may not be the function $f(x)$ which $\mathbf{U}$ "computes"; but as any string $y \neq f(x)$ occurs with multiplicity a multiple of k, the branches containing such y cannot affect the number of accepting branches modulo k of any machine simulating $\mathbf{U}$ as a subroutine. In a $\mathsf{Mod}_k\mathsf{L}$ or $\mathsf{coMod}_k\mathsf{L}$ algorithm, such "incorrect results" occur in effect with measure zero.

In this sense, the closure result $\mathsf{Mod}_p\mathsf{L}^{\mathsf{Mod}_p\mathsf{L}} = \mathsf{Mod}_p\mathsf{L}$ for p prime which is implicit in Ref. [7] and explicitly shown in Ref. [8] 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. That is, a $\mathsf{Mod}_p\mathsf{L}$ oracle can be directly simulated in a $\mathsf{Mod}_p\mathsf{L}$ algorithm, 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 3.4. *For all $k \geqslant 2$, the class FUL_k is low for $\mathsf{Mod}_k\mathsf{L}$, $\mathsf{coMod}_k\mathsf{L}$, $\#_k\mathsf{L}$, and itself.*

The proof is trivial: one simply simulates the FUL_k machine computing f as part of the nondeterministic logspace machine/algorithm for the corresponding decision/function class. From simple number-theoretic considerations, the classes FUL_k have properties which are similar in appearance to those of $\mathsf{coMod}_k\mathsf{L}$ (while in fact giving rise to opposite class containments):

Theorem 3.5. *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 $\mathsf{FUL}_k = \mathsf{FUL}_{p_1 p_2 \cdots p_\ell}$.*

Proof. Throughout the following, let $\kappa = p_1 p_2 \cdots p_\ell$ 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

$$\gamma = \kappa/p_1 + \kappa/p_2 + \cdots + \kappa/p_\ell. \tag{3.1}$$

For each prime p_j, all terms in the right-hand sum except for the jth term are divisible by p_j: then γ has no prime divisors in common with κ, so that $\gcd(\gamma, \kappa) = 1$. 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 $0 \leqslant q < \kappa\beta/p_j$.

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 $m_j p_j$ for some $m_j \in \mathbb{N}$; and so $\mathbf{U}'$ has $m_j \kappa\beta$ 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 $m_j p_j + 1$ for some $m_j \in \mathbb{N}$; and so $\mathbf{U}'$ has $m_j \kappa\beta + \kappa\beta/p_j$ 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 $\beta(\kappa/p_1 + \cdots + \kappa/p_\ell) = \beta\gamma \equiv 1 \pmod{\kappa}$ 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 $f \in \mathsf{FUL}_\kappa$ be computed by an FUL_κ machine $\mathbf{U}'$ and have outputs of length bounded by $N := N(x) \in \mathrm{poly}(|x|)$.

The idea of our approach is based on the following construction for $N \in O(\log|x|)$, which would for instance apply if we wished to evaluate logarithmically many symbols of $f(x)$ in the in the work-tape of another machine. We 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 $\sigma_j(x)$ be the string computed by the jth simulation of $\mathbf{U}'$. If any of the simulations reject the input, or produces a different output from the first simulation (i.e. if $\sigma_j(x) \neq \sigma_1(x)$ for any $1 \leqslant j \leqslant k/\kappa$), then $\mathbf{U}'$ rejects. Otherwise, $\mathbf{U}'$ writes the string $\sigma_1(x)$ agreed upon by the simulations to the output tape.

The detailed analysis for $N \in \omega(\log|x|)$ proceeds by performing a similar simulation for blocks of output characters of some length $L := L(x) \in O(\log|x|)$. For each $1 \leqslant m \leqslant N/L$, define a machine $\mathbf{U}'_m$ which simulates $\mathbf{U}'$ except that it only 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 to obtain a string of length N. Let $M = N/L$ for the sake of brevity: rather than perform k/κ simulations of $\mathbf{U}'$, the machine $\mathbf{U}''$ performs k/κ simulations of each $\mathbf{U}'_m$ for $1 \leqslant m \leqslant M$, in sequence. Again, while simulating $\mathbf{U}'_m$ for any particular m, the machine $\mathbf{U}''$ stops and rejects if any of the simulations reject or produce a result inconsistent with the previous simulations; and in those branches in which $\mathbf{U}'_m$ has produced the same output $y^{(m)}$ each time, the string $y^{(m)}$ is written on the output tape (excluding any symbol $\bullet \notin \Sigma$). After finishing the simulations of $\mathbf{U}'_m$ for any $m < M$, it reuses the work-space to simulate the next machine $\mathbf{U}'_{m+1}$. Once the simulations of $\mathbf{U}'_M$ are finished, $\mathbf{U}''$ accepts unconditionally in any branch where it has not yet rejected.

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 $y^{(m)} \in (\Sigma \cup \{\bullet\})^L$ written on the tape for each $1 \leqslant m \leqslant M$, and $\Phi(x,y)$ be the number of branches in which $\mathbf{U}''$ accepts with $y \in \Sigma^*$ written on the tape. If $y \approx y^{(1)}y^{(2)}\cdots y^{(M)}$ (modulo any $\bullet$ symbols contained in any of the substrings $y^{(m)}$), then the number of branches in which $\mathbf{U}''$ accepts with a given string $y^{(m)}$ written on the mth block of L tape cells of the output tape is **(a)** independent of the other substrings $y^{(j)}$ for $j \neq m$, and **(b)** is the result of k/κ simulations of $\mathbf{U}'_m$ which each produce the substring $y^{(m)}$ as output; so that we have

$$\Phi(x,y) \;=\; \varphi_1\big(x,y^{(1)}\big)^{k/\kappa}\,\varphi_2\big(x,y^{(2)}\big)^{k/\kappa}\,\cdots\,\varphi_M\big(x,y^{(M)}\big)^{k/\kappa}. \tag{3.2}$$

Note that $\varphi_m(x,y^{(m)})$ is equal to the number of computational branches in which $\mathbf{U}'$ writes any string $\sigma \in \Sigma^*$ on the output tape, for which the mth block is similar to $y^{(m)}$ (again ignoring any $\bullet$ symbols in $y^{(m)}$). This 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 $E := \max\{e_j \mid k/p_j^{e_j} \in \mathbb{Z}\}$; 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. $\qquad\square$

The requirement that an FUL_k machine have one accepting branch mod k (or possibly zero if the machine computes a partial function) gives the following relation to the classes $\mathsf{FMod}_k\mathsf{L}$ and $\mathsf{F\cdot coMod}_k\mathsf{L}$:

Lemma 3.6. *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 $f : \Sigma^* \rightharpoonup \Sigma^*$. Consider a nondeterministic logspace machine $\mathbf{T}$ taking inputs $(x, j, b) \in \Sigma^* \times \mathbb{N} \times (\Sigma \cup \{\bullet\})$, 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 $\mathrm{bits}(f) \in \mathsf{Mod}_k\mathsf{L}$. To show $\mathrm{bits}(f) \in \mathsf{coMod}_k\mathsf{L}$, 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 $\mathsf{FUL}_k \subseteq \mathsf{FMod}_k\mathsf{L} \cap \mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$. $\qquad\square$

This identifies FUL_k as an important subclass of the existing logspace-modular function classes. For prime-power moduli, we may sharpen Lemma 3.6 to obtain a useful identity:

Lemma 3.7. *For any prime p and $e \geqslant 1$, $\mathsf{FUL}_{p^e} = \mathsf{FMod}_p\mathsf{L} = \mathsf{F}\cdot\mathsf{coMod}_p\mathsf{L}$.*

Proof. By Proposition 2.9, Theorem 3.5, and Lemma 3.6, it suffices to prove $\mathsf{FMod}_p\mathsf{L} \subseteq \mathsf{FUL}_p$ for p prime. For $f \in \mathsf{FMod}_p\mathsf{L}$, let $\mathbf{T}$ be a $\mathsf{Mod}_p\mathsf{L}$ machine which decides $\mathrm{bits}(f)$.

We construct a family of machines $\mathbf{T}_{j,b}$ (for each $j \in \mathbb{N}$ and $b \in \Sigma \cup \{\bullet\}$), where each machine $\mathbf{T}_{j,b}$ writes b on its output tape and simulates $\mathbf{T}$ to decide whether $(x, j, b) \in \mathrm{bits}(f)$ on an input $x \in \Sigma^*$. Without loss of generality, as in Ref. [8, 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. $\qquad\square$

This result is the crux of the result of Ref. [8], albeit extended beyond the characteristic functions of $L \in \mathsf{Mod}_p\mathsf{L}$: when k is a prime power, *any* function whose bits are verifiable by $\mathsf{coMod}_k\mathsf{L}$ algorithms, can also be evaluated naturally as a subroutine of a $\mathsf{coMod}_k\mathsf{L}$ algorithm. The importance of this result to us lies in the consequence for $\#_k\mathsf{L}$, as the prototypical class of functions verifiable in $\mathsf{coMod}_k\mathsf{L}$:

Corollary 3.8. $\#_{p^e}\mathsf{L} \subseteq \mathsf{FUL}_p$ *for any prime p and $e \geqslant 1$. It follows that $\#_{p^e}\mathsf{L}$ is low for $\mathsf{Mod}_p\mathsf{L} = \mathsf{coMod}_p\mathsf{L}$ and for $\#_p\mathsf{L}$ in this case.*

This follows from Proposition 3.2 and Lemma 3.4, and is the key technical ingredient of our result: it allows us to simulate logspace counting oracles modulo p^e as a part of a $\mathsf{coMod}_p\mathsf{L}$ algorithm.

4 Solving congruences and nullspaces mod k

We return to the motivating problems of this article. Let A be an $n \times n$ integer matrix and $\mathbf{y} \in \mathbb{Z}^n$ be provided as the input to LCON_k or LCONX_k; or B be an $n \times n$ matrix provided as input to $\mathsf{LCONNULL}_k$. 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 follow the analysis of Ref. [10, Section 8] which reduces solving linear congruences to computing generating sets for nullspaces modulo the primes p_j dividing k. The 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 $\#_{p^e}\mathsf{L}$ for prime powers p^e.

We consider nondeterministic logspace machines operating on an alphabet $\bar{\Sigma}_k = \Sigma_k \cup \{\bullet\}$, where $\Sigma_k = \{0, 1, \ldots, k-1\}$. For the function problems $\mathrm{LCONNULL}_k$ and 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 $\mathcal{S}_k : \Sigma_k^{n^2+n} \rightharpoonup \Sigma_k^n$ 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 $\mathbf{x} = \mathcal{S}_k(A, \mathbf{y})$ is such a solution.

For p prime, we first consider a logspace reduction to matrix inversion and iterated matrix products modulo p^e, in a machine equipped with a $\#_{p^e}\mathsf{L}$ oracle to compute certain matrix coefficients. The reduction itself is an adaptation of the analysis of McKenzie and Cook [10, Lemma 8.1], together with observations regarding the simulation of the $\#_{p^e}\mathsf{L}$ oracles.

Lemma 4.1. *For any p prime and $e \geqslant 1$, we have $\mathrm{LCONNULL}_{p^e} \in \mathsf{FUL}_p$.*

Proof. We reduce $\mathrm{LCONNULL}_{p^e}$ to $\mathrm{LCONNULL}_p$, matrix products, and access to oracles for computing coefficients of certain matrices. We proceed by showing, for each $1 \leqslant t \leqslant e$, that computing any individual coefficient from a set of vectors $\mathbf{V}_j^{(t)}$ which span $\mathrm{null}(B)$ modulo $q = p^t$ can be achieved by a $\#_{p^t}\mathsf{L}$ function. We remark on the case $t = 1$ further below, and suppose as an induction hypothesis that there exists some $1 \leqslant t < e$ for which computing the coefficients of such a spanning set can be performed by $\#_{p^t}\mathsf{L}$ functions.

We sketch the analysis of Ref. [10, Lemma 8.1] for completeness. 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, forming the columns of an $N_t \times n$ matrix $V^{(t)}$. Certainly any solution to $B\mathbf{w} \equiv 0 \pmod{p^{t+1}}$ must also be a solution to $B\mathbf{w} \equiv 0 \pmod{p^t}$; then we may decompose such $\mathbf{w}$ modulo p^e as a linear combination of the vectors $\mathbf{V}_j^{(t)}$,

$$\mathbf{w} = u_1 \mathbf{V}_1^{(t)} + \cdots + u_{N_t} \mathbf{V}_{N_t}^{(t)} + p^t \hat{\mathbf{w}} \tag{4.1a}$$

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

$$\mathbf{w} = \tilde{V}^{(t)} \mathbf{z}, \tag{4.1b}$$

for block matrices $\tilde{V}^{(t)} = \left[\mathbf{V}_1^{(t)} \ \mathbf{V}_2^{(t)} \ \cdots \ \mathbf{V}_{N_t}^{(t)} \mid p^t I \right]$ and $\mathbf{z} = \left[u_1 \ u_2 \ \cdots \ u_{N_t} \right]^{\mathsf{T}} \oplus \hat{\mathbf{w}} \in \mathbb{Z}^{N_t + n}$. 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

$$\left(\sum_{j=1}^{N_t} u_j \left[B_t \mathbf{V}_j^{(t)} + p^t \hat{B}_t \mathbf{V}_j^{(t)} \right] \right) + p^t B_t \hat{\mathbf{w}}$$

$$\equiv B \left(\sum_{j=1}^{N_t} u_j \mathbf{V}_j^{(t)} \right) + p^t \hat{\mathbf{w}} \equiv 0 \pmod{p^{t+1}}. \tag{4.2a}$$

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

$$\left(\sum_{j=1}^{N_t} u_j \left[B_t \mathbf{V}_j^{(t)} / p^t + \hat{B}_t \mathbf{V}_j^{(t)} \right] \right) + B_t \hat{\mathbf{w}} \equiv 0 \pmod{p}, \qquad (4.2\text{b})$$

or somewhat more concisely,

$$\bar{B}^{(t)} \mathbf{z} \equiv 0 \pmod{p}, \qquad (4.2\text{c})$$

where we define

$$\bar{B}^{(t)} = \left[\mathbf{b}_1^{(t)} \ \mathbf{b}_2^{(t)} \ \cdots \ \mathbf{b}_{N_t}^{(t)} \ | \ B_t \right], \qquad \text{for } \mathbf{b}_j^{(t)} = B_t \mathbf{V}_j^{(t)} / p^t + \hat{B}_t \mathbf{V}_j^{(t)}, \qquad (4.3)$$

and where $\mathbf{z}$ is as we defined it above. To find not just one vector $\mathbf{w}$ but a set of generators $\mathbf{V}_1^{(t+1)}, \ldots, \mathbf{V}_{N_{t\pm1}}^{(t+1)}$ over $\mathbb{Z}/p^e\mathbb{Z}$ for null$(B) \bmod p^{t+1}$, 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 $\mathbf{V}_h^{(t+1)} = \tilde{V}^{(t)} \mathbf{z}_h$.

Note that the dimension of the nullspace of $\bar{B}^{(t)}$ modulo p, taken as a subspace of the vector space $\mathbb{F}_p$, is bounded by $N_t + n$; we may then span null(B) mod p by 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 a collection of at most $N_t + n$ vectors $\mathbf{z}_h$ representing non-trivial vectors in null$(\bar{B}^{(t)})$ mod p which have coefficients bounded between 0 and p. We may take these vectors as the columns of a matrix $Z^{(t+1)}$: then we may compute a matrix whose columns span the nullspace of B modulo p^{t+1} as $V^{(t+1)} = \tilde{V}^{(t)} Z^{(t+1)}$. The number of columns of $V^{(t+1)}$ is $N_{t+1} \leqslant 2N_t + n$, by construction. Thus we have a reduction of LCONNULL$_{p^{t+1}}$ to evaluating $\bar{B}^{(t)}$, solving LCONNULL$_p$ on input $\bar{B}^{(t)}$ to obtain the coefficients of the matrix $Z^{(t+1)}$, and matrix multiplication modulo p^{t+1} of the matrices $\tilde{V}^{(t)}$ and $Z^{(t+1)}$.

- Coefficients of an iterated matrix product $M_1 M_2 \cdots M_{\text{poly}(n)}$ modulo k may be evaluated as $\#_k\mathsf{L}$ functions, using the approach outlined in Ref. [7, Proposition 9]. One simulates a branching program with nondeterministic choices, using the matrices in sequence as transition functions for each branching. If the coefficients of the matrices can be evaluated using an oracle for a class C which is low for $\#_k\mathsf{L}$ (such as $C = \mathsf{FUL}_k$), the coefficients of the matrix product can then be straightforwardly computed as $\#_k\mathsf{L}$ functions.

- Buntrock *et al.* [7, Theorem 10] implicitly show that individual coefficients of a spanning set for null(B) mod p for integer matrices B are $\#_p\mathsf{L}$ functions (as in the remarks following Lemma 3.2), using a sequence of NC^1-reductions — specifically those of Ref.[6, Theorem 5] and Refs. [5, 11], as well as conjunctive and disjunctive truth-table reductions which rely on Propositions 2.8 and 2.9.

For the base case of $t = 1$, the latter observation immediately shows that coefficients of $\mathcal{N}_p$ may be computed by FUL_p machines. To induct, let $q = p^{t+1}$, and suppose that for some $t \geqslant 1$ we have LCONNULL$_{p^t} \in \mathsf{FUL}_p = \mathsf{FUL}_q$. We may then evaluate the coefficients of a set of vectors $\mathbf{V}^{(t)}$ which span null(B) modulo p^t by simulating a FUL_q machine; coefficients of the matrix product with B may then be computed in $\#_q\mathsf{L}$ using the first observation above, which can be performed by a FUL_q machine using Corollary 3.8. The columns of $\bar{B}^{(t)}$ are either integer vectors of the form $B\mathbf{V}^{(t)}/p^t$, or are columns of $\hat{B}_t$: both can then be computed by FUL_q subroutines, as division by p^t (which is bounded by the constant p^e) can be performed in NC^1, and $B\mathbf{V}^{(t)}$ can be computed by simulating FUL_q machines, and $\hat{B}_t$ is also

obtained from the input matrix B by integer division by p^t. Again using $\mathrm{LCONNULL}_p \in \mathsf{FUL}_p = \mathsf{FUL}_q$, the coefficients of $Z^{(t+1)}$ are all either constant or in effect computable by FUL_q machines; the coefficients of $\tilde{V}^{(t)}$ are similarly constant or computable by FUL_q machines. We may then compute the coefficients of the matrix product $V^{(t+1)} = \tilde{V}^{(t)} Z^{(t+1)}$ as a $\#_q\mathsf{L} \subseteq \mathsf{FUL}_p$ function. By induction up to e, we may then compute coefficients of $\mathcal{N}_{p^e}$ in FUL_p.

To show that the entire function $\mathcal{N}_{p^e}$ may be computed in FUL_p, it suffices to bound the number of spanning vectors for the nullspace, to ensure that the matrices involved in the reductions are of polynomial size. By induction, the number of vectors $\mathbf{V}_j^{(e)}$ in the generating set will be $N_e \leqslant n + 2n + \cdots + 2^{e-1}n \leqslant p^e n \in O(n)$; it then follows that $\mathrm{LCONNULL}_{p^e} \in \mathsf{FUL}_p$. $\qquad \square$

A remark on oracle towers. The above reduction is recursive, but has constant depth, as $e \in O(1)$. In particular, the exponent e corresponds to the height of a tower of FUL_p oracles computing $\#_{p^t}\mathsf{L}$ functions. To simulate these oracles as part of e. g. a $\mathrm{coMod}_p\mathsf{L}$ algorithm, the space resources can be described straightforwardly using a stack model of the work tape: each nested $\#_{p^t}\mathsf{L}$ 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 $O(e\log(n)) = O(\log(n))$.

Following Ref. [10, Lemma 5.3], we may reduce LCON_k and LCONX_k for $k \geqslant 2$ to $\mathrm{LCONNULL}_k$, as follows. Suppose $A\mathbf{x} \equiv \mathbf{y} \pmod{k}$ has solutions. Consider $B = [A \,|\, \mathbf{y}]$: then there are solutions $\bar{\mathbf{x}} = \mathbf{x} \oplus x_{n+1}$ to the equation $B\bar{\mathbf{x}} \equiv 0 \pmod{k}$ 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.

Lemma 4.2. *For any prime $p \geqslant 2$ and $e \geqslant 1$, we have $\mathrm{LCON}_{p^e} \in \mathsf{Mod}_p\mathsf{L}$ and $\mathrm{LCONX}_{p^e} \in \mathsf{FMod}_p\mathsf{L}$.*

Proof. We demonstrate an algorithm for both problems on a deterministic logspace machine with a FUL_p oracle. For prime power moduli, x_{n+1} is coprime to p^e if and only if p does not divide x_{n+1}. To solve LCON_{p^e} and LCONX_{p^e}, we compute individually the final coefficients of the vectors $(\mathbf{Z}_0, \mathbf{Z}_1, \mathbf{Z}_2, \dots) = \mathcal{N}_{p^e}(B)$ 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. Finding such an index, or determining that there are none, is feasible for $\mathsf{FL}^{\mathsf{FUL}_p}$ by using the oracle to evaluate coefficients, and then deterministically testing divisibility. If there is no such index h, we indicate that no solution exists by rejecting unconditionally. Otherwise, there exists a solution to the linear congruence.

- To indicate that $(A, \mathbf{y})$ is a *yes* instance of LCON_{p^e}, we simply accept.

- To solve $\mathrm{bits}(\mathcal{S}_k)$, we store the relevant coefficient $x_{n+1}^{(h)}$ on the work tape in binary and compute $\alpha \equiv -x_{n+1}^{-1} \pmod{p^e}$ deterministically. Using the FUL_p oracle, we then query the coefficients $z_{h,j}$ of $\mathbf{Z}_h$, and compare $\alpha z_{h,j}$ to input coefficients, accepting (to indicate a *yes* instance) if and only if the coefficients match. $\qquad \square$

We may use the above results, together with the Chinese Remainder Theorem and Proposition 2.7, to show that LCON_k, LCONX_k, and LCONNULL_k are complete problems for coMod_kL and $\text{F·coMod}_k\text{L}$ respectively for arbitrary $k \geqslant 2$. This is easiest for LCON_k, but the same basic approach may be used in each case.

A remark on NC^1 reductions. Our usage of the terminology of $\text{(co)Mod}_k\text{L}$-completeness below follows that of Ref. [7] (Definition 5 and the introduction to Section 3). Note that in the case of k composite, the classes Mod_kL are not known to be closed under NC^1 reductions (this would imply, for instance, that Mod_kL is closed under complements, and in particular low for itself). By Propositions 2.8 and 2.7 together with Ref. [7, Lemma 6(v)], we may show that coMod_kL is closed under a particular kind of NC^1 reduction, in which the final output gate is an AND gate, and each subtree which produces an input to that gate is produced by an NC^1 circuit with oracles for Mod_{p_j}L for some *single* prime p_j which divides k (the primes may vary for different subtrees). In the results below one may substitute this sort of reduction for NC^1 reductions, with no confusion.

Theorem 4.3. *For all $k \geqslant 2$, LCON_k is coMod_kL-complete under NC^1 reductions.*

Proof. Let $k = q_1 q_2 \cdots q_\ell$ for powers of distinct primes $q_j = p_j^{e_j}$. As we implied in the introduction, one may reduce LCON_{p_j} to LCON_k, for any prime p_j dividing k, by considering the feasibility of the congruence

$$(kA/p_j)\mathbf{x} \equiv k\mathbf{y}/p_j \pmod{k}, \tag{4.4}$$

which is equivalent to $A\mathbf{x} \equiv \mathbf{y} \pmod{p_j}$, by dividing both sides and the modulus by $k/p_j \in \mathbb{N}$. By Propositions 2.6 through 2.8, all problems in LCON_k may be reduced to solving some instances of LCON_{p_j} for each $1 \leqslant j \leqslant \ell$: then LCON_k is coMod_kL-hard. Using the Chinese Remainder Theorem, we also have $\text{LCON}_k = \text{LCON}_{q_1} \cap \cdots \cap \text{LCON}_{q_\ell}$. As $\text{LCON}_{q_j} \in \text{Mod}_{p_j}\text{L} = \text{coMod}_{p_j}\text{L}$ for each $1 \leqslant j \leqslant \ell$, it follows by Proposition 2.7 that $\text{LCON}_k \in \text{coMod}_k\text{L}$ as well. $\qquad\square$

Theorem 4.4. *For all $k \geqslant 2$, LCONX_k and LCONNULL_k are $\text{F·coMod}_k\text{L}$-complete under NC^1 reductions.*

Proof. Again, let $k = q_1 q_2 \cdots q_\ell$ for powers of distinct primes $q_j = p_j^{e_j}$. We define $\text{congbits}(f, q_j)$ to be the decision problem of determining for inputs $(x, h, b) \in \Sigma_k^* \times \mathbb{N} \times \bar{\Sigma}_k$ whether $x \in \text{dom}(f)$, and if so, whether either $f(x)_h \equiv b \pmod{q_j}$ for $b \neq \bullet$ or $f(x)_h = \bullet = b$.

- Clearly $\text{bits}(\mathcal{S}_k)$ is the intersection of the problems $\text{congbits}(\mathcal{S}_k, q_j)$ for $1 \leqslant j \leqslant \ell$ by the Chinese Remainder Theorem. We show $\text{congbits}(\mathcal{S}_k, q_j) \in \text{coMod}_{q_j}\text{L}$ 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}. With a $\text{coMod}_{p_j}\text{L}$ algorithm, we may then decide whether $((A', \mathbf{y}'), h, b') \in \text{bits}(\mathcal{S}_{q_j})$. Thus $\text{bits}(\mathcal{S}_k) \in \text{coMod}_k\text{L}$.

- We follow the reduction of Ref. [10, Theorem 8.3], to show $\text{bits}(\mathcal{N}_k) \in \text{coMod}_k\text{L}$. 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

$$\tfrac{k}{q_1}\mathbf{X}_1^{(q_1)}, \; \ldots, \; \tfrac{k}{q_1}\mathbf{X}_{N_1}^{(q_1)}, \; \tfrac{k}{q_2}\mathbf{X}_1^{(q_2)}, \; \ldots, \tfrac{k}{q_j}\mathbf{X}_h^{(q_j)}, \; \ldots, \; \tfrac{k}{q_\ell}\mathbf{X}_{N_\ell}^{(q_\ell)} \; . \tag{4.5}$$

(We omit the vectors $k\hat{\mathbf{e}}_h$ included by Ref. [10], 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 suppose without loss of generality that $\mathcal{N}_k$ is defined, for k divisible by more than one prime, to produce this sequence of vectors as output. If we define

$$M_j = \sum_{t=1}^{j} N_t \,, \tag{4.6}$$

then each vector $\mathbf{Z}_h$ is congruent to $\mathbf{0}$ modulo q_j, for every $j \geqslant 1$ such that $h < M_{j-1}$ or $h \geqslant M_j$. We may then reduce 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 coMod$_{q_j}$L algorithm for congbits$_j(\mathcal{N}_k)$, using the logspace reduction to bits$(\mathcal{N}_{q_j})$ as above.

These suffice to show that LCONX$_k$, LCONNULL$_k \in$ F·coMod$_k$L for all k. To show that $\mathcal{S}_k$ is F·coMod$_k$L-complete under logspace many-to-one reductions, we may note (as in the proof in Theorem 4.3 for LCON$_k$) that solving bits$(\mathcal{S}_k)$ suffices to solve bits$(\mathcal{S}_{p_j})$ for each prime p_j dividing k; as bits$(\mathcal{S}_{p_j})$ is coMod$_{p_j}$L-complete for each $1 \leqslant j \leqslant \ell$ by Ref. [7, Theorem 10], we may reduce any collection of languages $L_1, L_2, \ldots, L_\ell$ such that $L_j \in$ coMod$_{p_j}$L to bits$(\mathcal{S}_k)$. Then the intersection $L = L_1 \cap L_2 \cap \cdots \cap L_\ell \in$ coMod$_k$L is also logspace reducible to bits$(\mathcal{S}_k)$. As all languages $L \in$ coMod$_k$L have such a form by Proposition 2.7, including all problems of the form bits(f) for $f \in$ F·coMod$_k$L. It then follows that bits$(\mathcal{S}_k)$ is coMod$_k$L complete; a similar result obtains for bits$(\mathcal{N}_k)$. $\qquad\square$

5   Further Remarks

The above analysis was motivated by observing that the reduction of McKenzie and Cook [10] 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 LCONNULL$_p$. By showing that logspace counting oracles modulo p^e could be simulated by a coMod$_p$L machine, using the function class FUL$_k$ as a notion of naturally simulatable oracles for the classes Mod$_k$L and coMod$_k$L, the containments of Theorems 4.3 and 4.4 became feasible.

In the recursive reduction for 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 for the coMod$_k$L algorithm to be polynomial. The FUL$_p$ machines used to implement the #$_{p^e}$L oracles, from the constructions of Theorem 3.5 and Lemma 3.7, implicitly involve many repeated simulations of coMod$_p$L algorithms ($p^e/p = p^{e-1}$ 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 coMod$_p$L machine for LCONNULL$_{p^e}$, whose running time grows slowly with e.

We may use the classes #$_k$L and FUL$_k$ to explore the consequences of other closure results. For instance, if Mod$_k$L is closed under oracle calls, we may use the following alternative an alternative formulation of Mod$_k$L to show that $\chi_k \in$ FUL$_k$:

Lemma 5.1. *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. Furthermore, without loss of generality, $x \in L \implies \varphi(x) \equiv 1 \pmod{k}$.*

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 2.7 and 2.9. 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. Following Ref. [7], we may without loss of generality suppose that $\mathbf{T}_j$ accepts $x \in L_j$ with a single branch modulo p_j. Using a similar construction to that of Theorem 3.5 for the square-free case, we may obtain a *single* nondeterministic logspace machine $\mathbf{T}$ which accepts on a single branch modulo 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 equivalent to one modulo every prime p_j, which means that it is equivalent to one mod k; otherwise, there is some prime p_j which divides the number of accepting branches, so that the number of branches is not coprime to k. □

From this characterization of $\mathsf{coMod}_k\mathsf{L}$, we may argue that any function which is computable as a subroutine of a $\mathsf{coMod}_k\mathsf{L}$ algorithm belongs to FUL_k. For instance, let $\mathbf{T}$ be an oracle machine implementing a $\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}}$ algorithm for LCON_k which simply queries an LCON_k oracle and writes the result to the work tape. Suppose there is a nondeterministic machine $\mathbf{T}'$ implementing a $\mathsf{coMod}_k\mathsf{L}$ algorithm (with the acceptance conditions of Lemma 5.1) which simulates $\mathbf{T}$. Whatever decision procedure is performed on the oracle's output, it must be possible for $\mathbf{T}'$ to accept with one branch (modulo k); thus the oracle itself may be simulated in such a way that there are a number of computational branches which coprime to k, and without loss of generality equal to 1 $\pmod{k}$. If $\mathbf{T}'$ copies the oracle result to the output tape after simulating $\mathbf{T}$ with the appropriate conditions to obtain a single accepting branch modulo k, it follows that $\mathbf{T}'$ is a FUL_k machine computing χ_k. Thus closure of $\mathsf{Mod}_k\mathsf{L}$ under oracles would have a catastrophic effect on the classes $\mathsf{Mod}_p\mathsf{L}$ for p dividing k. Let $p_1, p_2, \ldots, p_\ell$ be the distinct prime factors of k: from Theorem 3.5 and Lemma 3.6, we have

$$\mathsf{FUL}_k = \mathsf{FMod}_{p_1}\mathsf{L} \cap \mathsf{FMod}_{p_2}\mathsf{L} \cap \cdots \cap \mathsf{FMod}_{p_\ell}\mathsf{L} . \tag{5.1}$$

From $\chi_k \in \mathsf{FUL}_k$, it follows that $\mathrm{LCON}_k \in \mathsf{Mod}_{p_j}\mathsf{L}$ for each p_j, and that therefore $\mathsf{Mod}_k\mathsf{L} = \mathsf{Mod}_{p_j}\mathsf{L}$ for every prime factor p_j of k. This does not seem likely for k divisible by multiple primes, unless these classes are also identical to some other class, such as UL. We might therefore expect the classes $\mathsf{Mod}_k\mathsf{L}$ *not* to be closed under oracles, on this basis. Of course, a proof that $\mathsf{Mod}_p\mathsf{L} \neq \mathsf{Mod}_q\mathsf{L}$ for primes $p \neq q$ would imply that $\mathsf{L} \neq \mathsf{P}$; so a proof that $\mathsf{Mod}_k\mathsf{L}$ is not closed under oracles for k having more than one prime factors should perhaps be expected to be difficult.

Note that the result $\mathrm{LCON}_k \in \mathsf{coMod}_k\mathsf{L}$ is equivalent to the partial function $\varsigma_k = \mathrm{LCON}_k \times \{1\}$ being in $\mathsf{F} \cdot \mathsf{coMod}_k\mathsf{L}$. It is not difficult to show that $\chi_k \in \mathsf{F} \cdot \mathsf{coMod}_k\mathsf{L}$ is equivalent to $\mathsf{Mod}_k\mathsf{L}$ being closed under complementation, as follows. If $\chi_k \in \mathsf{F} \cdot \mathsf{coMod}_k\mathsf{L}$, we could by that fact verify instances of $\overline{\mathrm{LCON}_k}$ by $\mathsf{coMod}_k\mathsf{L}$ algorithms; conversely, having $\mathsf{coMod}_k\mathsf{L}$ algorithms for both LCON_k and its complement allows us also to verify values of χ_k. Szelepcsényi [13] attempts to show that $\mathsf{Mod}_k\mathsf{L}$ is closed under complementation if and only if it is closed under oracles; however, his approach seems to rely on there being a complete language $L \in \mathsf{coMod}_k\mathsf{L}$ whose characteristic function χ_k is in $\#_k\mathsf{L}$ (which is in fact equivalent to $\mathsf{Mod}_k\mathsf{L}$ being closed under oracles). We discuss this and Ref. [13] in the Appendix.

We conclude with two questions: **(a)** Does $\mathsf{FUL}_k = \mathsf{FMod}_k\mathsf{L} \cap \mathsf{F}\cdot\mathsf{coMod}_k\mathsf{L}$ for each $k \geqslant 2$, and if not, how can we characterize FUL_k as a subset? **(b)** Does FUL (the class of functions evaluatable by a UL machine) equal the intersection of FUL_k over all $k \geqslant 2$?

Acknowledgements

Appendix.   Remarks on a preprint of Szelepcsényi

In an apparently unpublished draft [13], Szelepcsényi demonstrated that for arbitrary $k \geqslant 2$, the hierarchy of classes

$$\mathsf{Mod}_k\mathsf{LH} := \mathsf{Mod}_k\mathsf{L} \cup \mathsf{Mod}_k\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}} \cup \mathsf{Mod}_k\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}}} \cup \cdots \tag{A.1}$$

is equal to $\mathsf{AC}^0(\mathsf{Mod}_k\mathsf{L})$, the closure of $\mathsf{Mod}_k\mathsf{L}$ under AC^0 reductions. He attempted to further demonstrate that $\mathsf{Mod}_k\mathsf{LH} = \mathsf{L}^{\mathsf{Mod}_k\mathsf{L}}$, and that this implied that closure of $\mathsf{Mod}_k\mathsf{L}$ under complementation, oracle calls, or completely general NC^1 reductions were equivalent conditions. In view of the discussion following Lemma 5.1, this is a significant claim, which we now discuss.

Szelepcsényi's main focus seems to be on the analogy between $\mathsf{coMod}_k\mathsf{L}$ (which he describes as $\mathsf{Mod}_{=k}\mathsf{L}$) and $\mathsf{C}_=\mathsf{L}$; the former being the modular equivalent of the latter essentially through the relationship described by Lemma 3.2, using the fact that $\mathsf{Gap}_k\mathsf{L} = \#_k\mathsf{L}$ by the remarks following Definition 3.1. Using this same equality, wherever Ref. [13] considers "acceptance gaps", we may simply consider the number of accepting branches; however, Szelepcényi's analysis is better motivated by the analogy to $\mathsf{C}_=\mathsf{L}$ which is suggested by the alternative definition of $\mathsf{coMod}_k\mathsf{L}$ via GapL functions. Despite the fact that $\mathsf{C}_=\mathsf{L}$ is also not known to be closed under complementation, Allender, Beals, and Ogihara [1] show that a similar hierarchy of $\mathsf{C}_=\mathsf{L}$ algorithms using nested $\mathsf{C}_=\mathsf{L}$ oracles collapses to $\mathsf{L}^{\mathsf{C}_=\mathsf{L}}$. Ref. [13] attempts to show that a similar analysis could be applied to $\mathsf{coMod}_k\mathsf{L}$.

We may sketch the main claim of Ref. [13] as follows (see the original manuscript for details). Following the proof structure of Ref. [1] and making implicit use of the characterization of $\mathsf{coMod}_k\mathsf{L}$ in Lemma 5.1 (albeit expressed in terms of acceptance gaps), Ref. [13] attempts to show that an $\mathsf{NC}^1(\mathsf{coMod}_k\mathsf{L})$ circuit — involving AND, OR, NOT, and L gates for any $L \in \mathsf{coMod}_k\mathsf{L}$ — could be simulated in $\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}}$ using the closure of $\mathsf{coMod}_k\mathsf{L}$ under intersection and conjunctive truth-table reductions. For a given circuit C in a uniform circuit family $\{C_1, C_2, \ldots\} \in \mathsf{NC}^1(\mathsf{coMod}_k\mathsf{L})$, one guesses nondeterministically at a set of gates which would produce the outcome '1', where each guess for a gate $g \in C$ is represented by a bit $x_g \in \{0, 1\}$ produced during a depth-first traversal of the circuit. The entire sequence of guesses is represented by a string $x \in \{0, 1\}^{|C|}$, and is attributed a mass m_x corresponding to the sum of $2^{d_g} x_g$ over all $g \in C$ (where d_g is the depth of $g \in C$ measured from the output). In the traversal, some of the gates are simulated by running a $\mathsf{coMod}_k\mathsf{L}$ algorithm as characterized by Lemma 5.1, giving rise to some non-trivial number of accepting paths for the entire computation.

- If a gate in C is guessed to have the value '1', it is simulated, essentially multiplying the number of accepting branches modulo k by a $\#_k\mathsf{L}$ function. If this guess is incorrect (we say a "false positive"),

the number of accepting branches is multiplied by a number which has one or more prime factors in common with k; if the guess is correct the number of accepting branches modulo k is unaffected.

- If a gate in C is guessed to have the value '0', it is skipped over in a traversal of the circuit, giving rise to no increase in the number of accepting branches. If this guess is incorrect (we say a "false negative"), the mass m_x is smaller than is might be.

One then sets up an optimization problem to try to find the string x with the largest mass, subject to having no false positives, by preventing guesses x with false positives from contributing to the number of accepting paths modulo k. This optimization problem would then have a unique optimum $\bar{x}$ with mass $\bar{m}$, which consists of the correct guesses for every gate in C, and would be verifiable on a $\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}}$ machine. This amounts to using a nondeterministic logspace machine $\mathbf{U}$ to compute functions, where "incorrect answers" may not have zero branches modulo k (as with a FUL_k machine), but instead may have a number of branches which have prime divisors in common with k; and then using this machine as an oracle in a $\mathsf{coMod}_k\mathsf{L}$ algorithm as characterized in Lemma 5.1. Given the unique optimum, one could simply search for it in logarithmic space, allowing a $\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}}$ algorithm to simulate the $\mathsf{NC}^1(\mathsf{coMod}_k\mathsf{L})$ machine. If $\mathsf{Mod}_k\mathsf{L}$ were then also closed under complements (in which case it would also be closed under arbitrary logspace truth-table reductions), the search for the optimum could be solved by a $\mathsf{Mod}_k\mathsf{L}$ algorithm, showing that $\mathsf{Mod}_k\mathsf{L}$ is closed under complements if and only if it is closed under oracles.

The $\mathsf{L}^{\mathsf{Mod}_k\mathsf{L}}$ simulation of $\mathsf{NC}^1(\mathsf{Mod}_k\mathsf{L})$ presented by Ref. [13] seems to have a flaw, in that it is not clear that that several nondeterministic guesses which each contain false positives could not contribute to simulate the existence of an optimum with mass greater than $\bar{m}$ for the given verifying algorithm. While several guesses $x^{(1)}, x^{(2)}, \ldots, x^{(r)}$ at the values of the gates may each have a number of accepting paths $\varphi^{(1)}, \varphi^{(2)}, \ldots, \varphi^{(r)}$ which have prime factors in common with k, the total number of accepting paths $\varphi^{(1)} + \varphi^{(2)} + \cdots + \varphi^{(r)}$ may be coprime to k, if no integer $\nu > 1$ divides all of the integers $\varphi^{(j)}$.

This construction could be repaired if $\chi_k \in \#_k\mathsf{L}$, where χ_k is the characteristic function of a $\mathsf{coMod}_k\mathsf{L}$-complete problem L, by simulating the nondeterministic machine $\mathbf{U}$ whose acceptance function is congruent to χ_k to simulate gates in a $\mathsf{NC}^1(\mathsf{LCON}_k)$ circuit. Indeed, such a function — or equivalently, a function $\tilde{\chi}_k$ such that $\tilde{\chi}_k(x) \equiv 0 \pmod{k}$ when $x \notin L$ and $\gcd(\tilde{\chi}_k(x), k) = 1$ otherwise — would make the analogy to the collapse result of Ref. [1] complete, as the former result depends on the fact that $\mathbb{Z}$ has no zero divisors, and such a function would allow the same simulation technique to avoid all of the zero divisors of $\mathbb{Z}/k\mathbb{Z}$ in the multiplication of branches in the simulation of $\mathsf{coMod}_k\mathsf{L}$ oracles. However, it is easy to show that $\chi_k \in \#_k\mathsf{L}$ if and only if $\chi_k \in \mathsf{FUL}_k$. If $\chi_k \in \mathsf{FUL}_k$, we may simply compute χ_k and reject if we see that the outcome of the computation is 0: thus $\chi_k \in \#_k\mathsf{L}$. Conversely, if $\chi_k \in \#_k\mathsf{L}$, this means that the partial function $\varsigma_k : \mathsf{LCON}_k \to \{1\}$ is in FUL_k; from this we may easily construct another FUL_k machine computing $\overline{\varsigma}_k : \overline{\mathsf{LCON}_k} \to \{0\}$, and obtain a FUL_k machine computing χ_k by simulating machines computing ς_k and $\overline{\varsigma}_k$ in parallel. As we saw in Section 5, $\chi_k \in \mathsf{FUL}_k$ is already equivalent to $\mathsf{Mod}_k\mathsf{L}$ being closed under oracles, which has catastrophic consequences for the mod-logspace classes.

Thus, one might expect the main result of Ref. [13] to be difficult to salvage, unless one could show that constructive interference of non-deterministic guesses which have false positives could be identified and discounted in the optimisation problem. In particular, without a correct simulation of the $\mathsf{NC}^1(\mathsf{Mod}_k\mathsf{L})$ circuit corresponding to the unique optimum of that optimisation problem, it is not clear that closure of $\mathsf{Mod}_k\mathsf{L}$ under complementation is equivalent to closure under oracles.

References

[1] E. ALLENDER, R. BEALS, AND M. OGIHARA: The complexity of matrix rank and feasible systems of linear equations. *Comp. Complexity*, 8:99–126, 1999. 15, 16

[2] E. ALLENDER AND M. OGIHARA: Relationships among PL, #L, and the determinant. In *RAIRO — Theor. Inform. Appl.,*, volume 30, pp. 1–21, 1996. 3

[3] V. ARVIND AND T. C. VIJAYARAGHAVAN: Classifying problems on linear congruences and abelian permutation groups using logspace counting classes. *Computational Complexity*, 19:57–98, 2010. 2

[4] R. BEIGEL, J. GILL, AND U. HERTRAMPF: Counting classes: Thresholds, parity, mods, and fewness. In *Proc. STACS 90, Lecture Notes in Computer Science*, volume 415, pp. 49–57, 1990. 5

[5] S. J. BERKOWITZ: On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984. 10

[6] A. BORODIN, J. VON ZUR GATHEN, AND J. HOPCROFT: Fast parallel matrix and gcd computations. In *23rd Annual Symposium on Foundations of Computer Science*, pp. 65–71, 1982. 10

[7] G. BUNTROCK, C. DAMM, U. HERTRAMPF, AND C. MEINEL: Structure and importance of logspace-mod classes. *Theory of Computing Systems*, 25:223–237, 1992. 1, 2, 3, 4, 5, 10, 12, 13, 14

[8] U. HERTRAMPF, S. REITH, AND H. VOLLMER: A note on closure properties of logspace mod classes. *Information Processing Letters*, 75:91–93, 2000. 4, 5, 8

[9] J. KÖBLER AND S. TODA: On the power of generalized mod-classes. *Mathematical Systems Theory*, 29:33–46, 1996. 2

[10] P. MCKENZIE AND S. COOK: The parallel complexity of abelian permutation group problems. *SIAM Journal of Computing*, 16:880–909, 1987. 2, 3, 9, 11, 12, 13

[11] K. MULMULEY: A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica*, 7:101–104, 1987. 10

[12] WALTER L. RUZZO, JANOS SIMON, AND MARTIN TOMPA: Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28(2):216–230, 1984. 2

[13] R. SZELEPCSÉNYI: Logspace-mod-classes with composite moduli (preliminary version). http://people.cs.uchicago.edu/~robert/papers/mod-log.ps, 1999. 3, 14, 15, 16

[14] L. G. VALIANT: The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979. 4

AUTHOR

Niel de Beaudrap
Postdoctoral Research Associate
Department of Applied Mathematics and Theoretical Physics
University of Cambridge
niel.debeaudrap@gmail.com