Optimization Modulo Integer Linear-Exponential Programs

S Hitarth[1], Alessio Mansutti[2], and Guruprerana Shabadi[3]

[1] *Hong Kong University of Science and Technology, Hong Kong*
[2] *IMDEA Software Institute, Spain*
[3] *University of Pennsylvania, USA*

Abstract

This paper presents the first study of the complexity of the optimization problem for *integer linear-exponential programs* which extend classical integer linear programs with the exponential function $x \mapsto 2^x$ and the remainder function $(x, y) \mapsto (x \bmod 2^y)$. The problem of deciding if such a program has a solution was recently shown to be NP-complete in [Chistikov et al., ICALP'24]. The optimization problem instead asks for a solution that maximizes (or minimizes) a linear-exponential objective function, subject to the constraints of an integer linear-exponential program. We establish the following results:

- If an optimal solution exists, then one of them can be succinctly represented as an *integer linear-exponential straight-line program (ILESLP)*: an arithmetic circuit whose gates always output an integer value (by construction) and implement the operations of addition, exponentiation, and multiplication by rational numbers.
- There is an algorithm that runs in polynomial time, given access to an integer factoring oracle, which determines whether an ILESLP encodes a solution to an integer linear-exponential program. This algorithm can also be used to compare the values taken by the objective function on two given solutions.

Building on these results, we place the optimization problem for integer linear-exponential programs within an extension of the optimization class NPO that lies within $\mathrm{FNP}^{\mathrm{NP}}$. In essence, this extension forgoes determining the optimal solution via binary search.

(Page 12 includes a table of contents.)

Acknowledgements. We would like to thank Dmitry Chistikov for recommending the use of finite differences to simplify certain arguments in the proof of Proposition 3, Dario Fiore for pointing us to the work of Rivest, Shamir and Wagner on time-lock puzzles [RSW96], and Christoph Haase for pointing us to the work of Myasnikov, Ushakov and Won [MUW12]. This work began while S Hitarth and Guruprerana Shabadi were interns in IMDEA Software Institute.

1 Introduction

Integer Linear Programming (ILP), the problem of determining an optimal (maximal or minimal) value of a multivariate linear polynomial evaluated over the integer solutions to a system of linear inequalities $A \cdot \boldsymbol{x} \leq \boldsymbol{b}$, offers one of the most versatile frameworks for solving computational problems in operations research and computer science. Summarizing the preface of *"50 Years of Integer Programming 1958–2008"* [JLN+10], over decades, a rich collection of methods for solving ILP have been developed, such as cutting-plane methods, branch-and-bound algorithms, and techniques from polyhedral geometry. These developments have not only deepened our understanding of the structure of the problem and its complexity, but also have been translated into powerful solvers (e.g., `SCIP`, `CPLEX`, `Gurobi`) that can handle large-scale real-world instances very efficiently.

In this paper, we study the optimization problem of *Integer Linear-Exponential Programming (ILEP)*, which extends ILP with the *exponential function* $x \mapsto 2^x$ and the *remainder function* $(x, y) \mapsto (x \bmod 2^y)$. An instance of ILEP is a maximization (or minimization) problem

$$\begin{aligned} &\text{maximize } \tau(\boldsymbol{x}) \\ &\text{subject to } \rho_i(\boldsymbol{x}) \leq 0 \text{ for each } i \in \{1, \ldots, k\} \\ &\qquad\qquad \rho_i(\boldsymbol{x}) = 0 \text{ for each } i \in \{k+1, \ldots, m\}, \end{aligned}$$

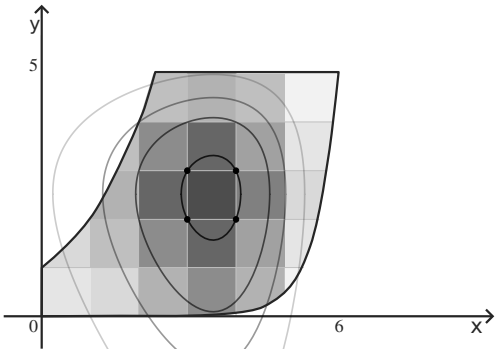
where $\boldsymbol{x}$ is a vector of variables over the non-negative integers $\mathbb{N}$, and $\tau, \rho_1, \ldots, \rho_m$ are *linear-exponential terms* of the form

$$\sum\nolimits_{i=1}^{n} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum\nolimits_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j})\right) + d, \tag{1}$$

in which all *coefficients* $a_i, b_i, c_{i,j}$ and the *constant* d are integers. The system of constraints defined by the inequalities $\rho_i(\boldsymbol{x}) \leq 0$ and equalities $\rho_i(\boldsymbol{x}) = 0$ is an *integer linear-exponential program*.

Example 1. *To get a feel for this optimization problem, let us look at the instance*

$$\begin{aligned} \text{maximize } \tau(x, y) &\coloneqq 8x + 4y - (2^x + 2^y) \\ \text{subject to } \varphi(x, y, z) &\coloneqq y \leq 5 \\ &\qquad\; y \leq 2^x \\ &\qquad\; 2^z \leq 2^{16} y \\ &\qquad\; z = 3 \cdot x. \end{aligned}$$



After projecting away the variable z, which is just a proxy for $3 \cdot x$, the plot on the right shows a heat map of the objective function τ over the feasible region defined by φ (we only consider non-negative integer solutions). Visually, we see that any point in $\{3, 4\} \times \{2, 3\}$ is optimal.

Integer linear-exponential programming lacks two of the key properties that are central to ILP:

1. In ILP, it is a classical fact that if an optimal solution exists, then there is one whose bit size is polynomially bounded by the bit size of the input [BT76, vzGS78]. This is not the case for integer linear-exponential programs, where solutions may demand a non-elementary number of bits when represented in binary: by setting $x_0 = 1$ and writing a sequence of constraints of the form $x_{i+1} = 2^{x_i}$, one can force x_i to be equal to the tower of 2s of height i.

2. In ILP, whenever (optimal) solutions exist, at least one lies near the boundary of the feasible region defined by the system of linear inequalities. (This fact is made more precise in Example 2.) In ILEP, this geometric property no longer holds. Intuitively, this can already be seen in the instance from Example 1, where all optimal solutions lie near the center of the feasible region rather than near its boundary. We will revisit this observation in Example 3. Finding optimal solutions despite this fact is a central difficulty addressed in the paper.

Although solutions to integer linear-exponential programs may require astronomically large binary representations, the complexity of the *feasibility problem* for ILEP, that is, the problem of checking if an instance has a (not necessarily optimal) solution, is comparable to that of ILP. Indeed, this problem was recently shown to be NP-complete by Chistikov, Mansutti and Starchak in [CMS24], who developed a non-deterministic polynomial-time procedure based on quantifier elimination. This implies that a short and polytime-time checkable certificate exists for at least one solution of a feasible system[1]. In contrast, it is not known whether *optimal* solutions can be represented efficiently, namely, by polynomial-size objects that can be verified as valid solutions in polynomial time. This leads to the central question we explore in this paper:

Are there efficient representations for the optimal solutions to ILEP?

As this introduction hopes to convey, answering this question yields a distinctive perspective on integer programming. The algebraic techniques we employ in this paper are, as far as we know, non-standard in the context of optimization, and they appear to be applicable to other extensions of ILP, such as quadratic [PDM17,Lok15,EGKO19] and parametric versions of integer programming [She18, BGW17]. Furthermore, the representation we consider is quite natural and can be viewed as an extension of the class of *power circuits* introduced by Myasnikov, Ushakov and Won in [MUW12], which played a crucial role in resolving several questions in algorithmic group theory, most notably in establishing that the word problem for the one-relator Baumslag group lies in P [MUW11]. To our knowledge, this is the first application of power circuits within the context of integer programming.

1.1 Succinct encoding of optimal solutions

To address the central question posed above, we introduce a new representation of solutions called *Integer Linear-Exponential Straight-Line Programs*. Let us begin by defining a *Linear-Exponential Straight-Line Program* (*LESLP*) as a sequence $\sigma := (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$ of variable assignments such that each expression ρ_i ($i \in [0..n]$) has one of the following forms: 0, $x_j + x_k$, 2^{x_j}, or *scaling expressions* $a \cdot x_j$, where the indices $j, k \in [0..i-1]$ refer to previous assignments in the program, and $a \in \mathbb{Q}$. The *bit size* of σ is defined as the number of symbols required to write it down, which includes encoding the indices $0, \ldots, n$ in unary, and the rational coefficients in scaling expressions as pairs of integers $\frac{m}{g}$ with $g \geq 1$, encoded in binary.

We define $[\![\sigma]\!] : \{x_0, \ldots, x_n\} \to \mathbb{R}$ as the map that assigns to each variable x_i the value that the expression ρ_i takes when evaluated using standard arithmetic. Note that x_0 always takes the value 0. We call σ an *Integer Linear-Exponential Straight-Line Program* (*ILESLP*) if all of its variables evaluate to integers. For example, the following LESLP σ

$$x_0 \leftarrow 0, \quad x_1 \leftarrow 2^{x_0}, \quad x_2 \leftarrow -1 \cdot x_1, \quad x_3 \leftarrow 2^{x_2}, \quad x_4 \leftarrow 2^{x_3},$$

is not an ILESLP as $[\![\sigma]\!](x_3) = \frac{1}{2}$ and $[\![\sigma]\!](x_4) = \sqrt{2}$.

[1] While these certificates are not discussed explicitly in [CMS24], they can be extracted from the accepting paths of the non-deterministic procedure.

Consider an instance (τ, φ) of ILEP, where τ is the objective function (to be maximized or minimized) and φ is an integer linear-exponential program. An ILESLP σ is a solution to (τ, φ) whenever *(i)* the set $\{x_0, \ldots, x_n\}$ contains (at least) all the variables of τ and φ, and *(ii)* the variable assignment $[\![\sigma]\!]$ satisfies all constraints in φ. We now state our main theorem:

Theorem 1. *If an instance of integer linear-exponential programming has an optimal solution, then it has one representable with a polynomial-size* ILESLP*.*

We defer giving an overview of the proof of Theorem 1 to Section 1.4. Let us stress that, for the sake of a simpler exposition, we solely focus on integer linear-exponential programs with *variables ranging over* $\mathbb{N}$. Our results can however be easily adapted to variables ranging over $\mathbb{Z}$ by similar arguments as the ones given in [CMS24, Sec. 8] for the feasibility problem. That being said, the variables in ILESLPs must still range over $\mathbb{Z}$, and auxiliary variables not occurring in the instance of ILEP are necessary to succinctly encode a solution. Consider for example the linear-exponential program $\varphi(x, y, z) \coloneqq x = k \wedge y = 2^x \wedge z = 2^y - 1$, where k is a positive integer encoded in binary. A (short) ILESLP σ representing the only solution to φ is

$$x_0 \leftarrow 0, \quad x_1 \leftarrow 2^{x_0}, \quad x \leftarrow k \cdot x_1, \quad y \leftarrow 2^x, \quad x_2 \leftarrow 2^y, \quad x_3 \leftarrow -1 \cdot x_1, \quad z \leftarrow x_2 + x_3,$$

where $x_0, \ldots, x_3$ are auxiliary variables, and $[\![\sigma]\!](x_3)$ is negative. Intuitively, it is not possible to have a short ILESLP in which all variables evaluate to non-negative integers, because the binary expansion of $2^{2^k} - 1$ has doubly exponentially many 1s with respect to the bit size of φ.

Power circuits. In [MUW12], Myasnikov, Ushakov and Won consider a class of straight-line programs, which they refer to as *(constant) power circuits*, that feature the operations $x + y$, $x - y$ and $x \cdot 2^y$, and the constant 1. In the paper, the authors develop several polynomial-time algorithms for manipulating such circuits. The main one is a normalization procedure that, among other things, reduces the operation $x \cdot 2^y$ to the simpler exponential function 2^y. Given that power circuits are semantically restricted to integer-valued variables, ILESLPs thus represent a natural generalization that introduces scaling by rational constants via the expressions $a \cdot x$, with $a \in \mathbb{Q}$. The need for rational coefficients is, in fact, already discussed in [MUW12, Section 9.1], as we explain next.

Consider an integer linear-exponential program φ whose constraints imply $3 \cdot x = 2^{2y} - 1$ and require y to be a positive integer of exponential magnitude in the size of φ. To see why any polynomial-size ILESLP σ encoding a solution to φ must have a scaling expression with a non-integer coefficient, observe that for every $k \geq 1$, the number $\frac{2^{2k}-1}{3}$ is a positive integer, and moreover its binary representation is $1(01)^{k-1}$. Since $[\![\sigma]\!](y)$ is large, the binary expansion of $[\![\sigma]\!](x)$ must then alternate between 0s and 1s exponentially many times relative to the size of φ. However, one can show that an ILESLP with only integer coefficients (alternatively, a power circuit) can only encode numbers whose binary expansion alternates between 0s and 1s at most polynomially many times in the bit size of the ILESLP. Therefore, σ must either feature some non-integer coefficient, or be exponentially larger than φ.

1.2 Recognizing ILESLPs and when they encode solutions

Theorem 1 indicates that the optimization problems of ILP and ILEP are close: while integers must be encoded more succinctly in the case of ILEP, both problems admit short representations for optimal solutions. The first difference arises when we consider the problems of recognizing the set of ILESLPs, and of checking whether an ILESLP is a solution to an instance of ILEP.

Consider a LESLP $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$. The snippet of code below decides whether σ is an ILESLP by testing whether $[\![\sigma]\!](x_i) \in \mathbb{Z}$ iteratively on i from 1 to n. Such a test comes for free for additions: if $[\![\sigma]\!](x_j)$ and $[\![\sigma]\!](x_k)$ are integers, and σ features $x_i \leftarrow x_j + x_k$, then $[\![\sigma]\!](x_i) \in \mathbb{Z}$.

1: **for** $i = 1$ to n **do** ▷ *during the ith iteration, we already know that* $x_0, \ldots, x_{i-1}$ *are integers*
▷ *in the next two lines, "****assert*** φ*" stands for "****if*** $\neg\varphi$ ***then return*** *false"*
2: **if** ρ_i is of the form 2^x **then assert** $[\![\sigma]\!](x) \geq 0$ ▷ *recall:* $[\![\sigma]\!](x) \in \mathbb{Z}$
3: **if** ρ_i is of the form $\frac{m}{g} \cdot x$ **then assert** $\frac{g}{\gcd(m,g)}$ divides $[\![\sigma]\!](x)$
4: **return** true

Given an LESLP $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$, let us write $[\![\sigma]\!]_\bullet$ as a shorthand for $[\![\sigma]\!](x_n)$. Lines 2 and 3 of the above code only verify properties of variables whose values were already established to be integers in earlier iterations of the **for** loop. Therefore, the problem of checking if an LESLP is an ILESLP reduces to deciding the following two properties of an input ILESLP σ:

$\textsc{Nat}_{\textsc{ileslp}}$: Is $[\![\sigma]\!]_\bullet \geq 0$?

$\textsc{Div}_{\textsc{ileslp}}$: Is $[\![\sigma]\!]_\bullet$ divisible by g, for $g \in \mathbb{N}_{\geq 1}$ given in binary?

The problem $\textsc{Nat}_{\textsc{ileslp}}$ is the "linear-exponential analogue" of the well-known PosSLP problem, which involves straight-line programs featuring assignments $x_i \leftarrow x_j \cdot x_k$ in place of exponentiation, and whose complexity is still wide open [BJ24, BDJ24]. In contrast, the corresponding decision problem for power circuits is known to be decidable in polynomial time [MUW12, Sec. 7.5]. We show that this result carries over to the more general setting of ILESLPs:

Lemma 1. $\textsc{Nat}_{\textsc{ileslp}}$ *can be decided in polynomial time.*

In [MUW12], one notable feature of the previously-mentioned normalization procedure for power circuits is that it makes checking the sign trivial: once a circuit is in normal form, the sign of the encoded number is immediately evident from the structure of the circuit. (Another key property is that power circuits representing the same number have the same normal form.) While we believe that a similar normal form exists for ILESLPs, in this paper we instead provide a direct procedure for solving $\textsc{Nat}_{\textsc{ileslp}}$. Setting aside complexity considerations for now, the procedure originates from a simple idea. Given an ILESLP (or power circuit) σ where $[\![\sigma]\!](z) = a \cdot 2^{[\![\sigma]\!](x)} - b \cdot 2^{[\![\sigma]\!](y)}$ for three variables x, y, z and positive integers a, b, look at the distance $k \coloneqq |[\![\sigma]\!](x) - [\![\sigma]\!](y)|$. One possibility is for k to be at least $c \coloneqq \lceil \log_2(\max(a, b)) \rceil$: the sign of $[\![\sigma]\!](z)$ is then the sign of the coefficient a or b corresponding to the larger variable among x and y. We can check $k \geq c$ by opportunely modifying σ so as to be able to test $[\![\sigma]\!](x) - [\![\sigma]\!](y) - c \geq 0$ and $[\![\sigma]\!](y) - [\![\sigma]\!](x) - c \geq 0$ with two recursive calls to the algorithm for $\textsc{Nat}_{\textsc{ileslp}}$. If $k < c$ instead, k is logarithmic in the bit size of σ. We can then compute k: a naïve solution is to perform binary search on a suitable interval, repeatedly invoking the algorithm for $\textsc{Nat}_{\textsc{ileslp}}$ on a modified ILESLP. Then, $[\![\sigma]\!](z)$ has the same sign as either $a \cdot 2^k - b$ or $a - b \cdot 2^k$, depending on which of the two variables, x or y, is larger. While our final polynomial-time procedure differs from this outline, the distinction between "large distance" and "short distance" remains central.

Turning to the problem $\textsc{Div}_{\textsc{ileslp}}$, we show that it can be decided in polynomial time when having access to an integer factorization oracle. This is arguably the best we can hope for, as solving $\textsc{Div}_{\textsc{ileslp}}$ in P (in fact, even in BPP) would refute the *Sequential Squaring Assumption*, a well-known cryptographic assumption put forward by Rivest, Shamir and Wagner in [RSW96].[2]

[2] A proof of this hardness result is provided for completeness in Appendix A; see also [CJSS21] for a further reference. It is worth noting that rational constants do not have any role in this proof: the problem is unlikely to be in P even in the more restricted setting of power circuits.

Lemma 2. $\textsc{Div}_{\textsc{ileslp}}$ *is in* $\textsc{P}^{\textsc{factoring}}$.

The main step in establishing Lemma 2 is showing that, even though $[\![\sigma]\!](x)$ can be astronomically large, we can still compute $[\![\sigma]\!](x) \bmod \phi(g)$ in polynomial time using the factoring oracle, where ϕ stands for Euler's totient function. This allows us to then efficiently compute $2^{[\![\sigma]\!](x)} \bmod g$ using the exponentiation-by-squaring method [BW08, Ch. 1.4].

As per all $\textsc{P}^{\textsc{factoring}}$ algorithms, given an input ILESLP σ and $g \in \mathbb{N}_{\geq 1}$, there is a polynomial-sized set of small primes that, when provided as an advice, enables running the algorithm deciding $\textsc{Div}_{\textsc{ileslp}}$ in polynomial time, avoiding all calls to the factorization oracle. We will explicitly construct this set in Section 8. Looking back at line 3 of the above snippet of code, note that the algorithm deciding $\textsc{Div}_{\textsc{ileslp}}$ has to be invoked only on divisors of the denominators g appearing in the rational coefficients of the LESLP σ. This allows us to define a common set $\mathbb{P}(\sigma)$ of polynomially-many small primes that suffices to decide in polynomial time all instances of $\textsc{Div}_{\textsc{ileslp}}$ that are relevant when determining if a LESLP is an ILESLP. From Lemmas 1 and 2, along with the fact that primality testing is in P [AKS04], we then establish the following result:

Proposition 1. *Given an* LESLP σ *and* $\mathbb{P}(\sigma)$*, one can decide in polynomial time if* σ *is an* ILESLP*. In order words, the set* $U \coloneqq \{(\sigma, \mathbb{P}(\sigma)) : \sigma \text{ is an ILESLP}\}$ *is recognizable in polynomial time.*

The set U in Proposition 1 represents the *universe* of all certificates for ILEP. Since $\mathbb{P}(\sigma)$ can be encoded using polynomially many bits relative to the size of σ, Theorem 1 implies that any instance of ILEP with an optimal solution has one representable by a polynomial-size element of U. Proposition 1 highlights a nuanced distinction between ILP and ILEP: certificates for the latter problem require some external objects (the sets $\mathbb{P}(\sigma)$) which are introduced to achieve polynomial-time recognizability of the certificates, but are not inherently required to encode solutions.

Let us now consider the problem of checking whether a given $(\sigma, \mathbb{P}(\sigma)) \in U$ is a solution to an instance of ILEP. To verify if σ satisfies an inequality of the form $\sum_{i=1}^{n} \left(a_i \cdot x_i + b_i \cdot 2^{x_i}\right) + d \leq 0$, we first check that $[\![\sigma]\!](x_i) \geq 0$ for all $i \in [1..n]$; as solutions are over $\mathbb{N}$. We then append new assignments to σ, to obtain an ILESLP σ' such that $[\![\sigma']\!]_\bullet = \sum_{i=1}^{n} \left(a_i \cdot [\![\sigma]\!](x_i) + b_i \cdot 2^{[\![\sigma]\!](x_i)}\right) + d - 1$. The ILESLP σ satisfies the inequality if and only if the algorithm for $\textsc{Nat}_{\textsc{ileslp}}$ returns false when applied to σ'. For the more general case of the linear-exponential terms from Equation (1), we must also account for the expressions $(x_j \bmod 2^{x_k})$ involving the remainder function. We show that these expressions are unproblematic (Section 9): starting from $(\sigma, \mathbb{P}(\sigma))$, we can compute in polynomial time an ILESLP σ'' such that $[\![\sigma'']\!]_\bullet = [\![\sigma]\!](x_j) \bmod 2^{[\![\sigma]\!](x_k)}$. Consequently, after appropriately updating the ILESLP, the verification proceeds similarly to the case without remainder functions. We stress the fact that the computation of σ'' requires access to $\mathbb{P}(\sigma)$; an integer factoring oracle can alternatively be used. (No oracle is needed in the case of power circuits, see [MUW12, Sec. 7.4].)

Proposition 2. *Checking whether* $(\sigma, \mathbb{P}(\sigma)) \in U$ *encodes a solution to an instance* (τ, φ) *of ILEP can be done in polynomial time in the bit sizes of* σ *and* φ*.*

1.3 Comparing values of the objective function without computing them

Continuing our comparison between ILP and ILEP, we need to address one last problem: the evaluation of the objective function. In ILP, the objective function $\tau(\boldsymbol{x})$, being a linear term, is trivial to evaluate: it suffices to perform a few additions and multiplications, and return the resulting integer, which is guaranteed to be of polynomial bit size with respect to the bit size of the solution and of τ. The property of τ being polynomial-time computable is a common feature of all optimization problems belonging to the complexity class NPO from [AMC+99]. This property implies that maximizing (or minimizing) τ subject to an integer linear program $\varphi(\boldsymbol{x})$ can be achieved

through *binary search* over a suitable interval $[a..b] \subseteq \mathbb{Z}$ containing the optimal value of τ; repeatedly solving an instance of the *feasibility* problem of ILP at each step of the search. For example, the first query checks whether $\varphi(\boldsymbol{x}) \wedge \tau(\boldsymbol{x}) \geq \frac{b-a}{2}$ is satisfiable, and updates the interval to $[a..\lfloor\frac{b-a}{2}\rfloor]$ or $[\lceil\frac{b-a}{2}\rceil..b]$ accordingly to the answer. When a and b are encoded in binary, polynomially many feasibility queries suffice to locate an optimal solution; that is, NPO $\subseteq \mathrm{FP}^{\mathrm{NP}}$.

In ILEP there seems to be no easy way to perform binary search over the set of numbers encoded by polynomial-size ILESLPs (Open problem 2 in Section 1.5 formalizes this issue). However, given an instance (τ, φ) of ILEP, we can still *compare* the values of τ at two solutions $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, each encoded as an ILESLP, in polynomial time relative to the sizes of τ, $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$. This is a direct consequence of the fact that $\textsc{Nat}_{\textsc{ileslp}}$ is in P (Lemma 1): to perform the comparison $\tau(\boldsymbol{s}_1) \leq \tau(\boldsymbol{s}_2)$, we construct an ILESLP σ such that $[\![\sigma]\!]_\bullet = \tau(\boldsymbol{s}_2) - \tau(\boldsymbol{s}_1)$, and then use the algorithm for $\textsc{Nat}_{\textsc{ileslp}}$ to determine the sign of this difference.

As a way of summarizing our comparison between ILP and ILEP, we introduce an adequate complexity class, which we denote by NPO-cmp. In this class, the requirement "the objective function is computable in polynomial-time" of NPO is weakened to "comparisons between values taken by the objective function can be performed in polynomial time"; see Section 10 for the formal definition of NPO-cmp. This relaxation forgoes the ability to search for the optimum via binary search; and so instead of an inclusion with $\mathrm{FP}^{\mathrm{NP}}$, we have NPO-cmp $\subseteq \mathrm{FNP}^{\mathrm{NP}}$. From the above discussion, and Theorem 1 and Propositions 1 and 2, we obtain:

Corollary 1. *The optimization problem for integer linear-exponential programs is in* NPO-cmp.

Of course, whether NPO-cmp should be considered a "natural" complexity class is open for debate and lies beyond the scope of this paper. Echoing Goldreich [Gol08, Chapter 2.1.1.1], understanding the true content of this class is challenging because, like NPO, it is defined solely in terms of the "external behavior" (algorithmic properties) instead of the "internal structure" of its problems. Nonetheless, at an intuitive level, NPO-cmp seems "natural" in the context of optimization problems whose solutions must be encoded succinctly, and where it is therefore unreasonable to require the objective function to produce, in polynomial time, an integer encoded in binary.

1.4 Overview of the proof of Theorem 1

To establish Theorem 1, the starting point is given by the non-deterministic polynomial-time algorithm designed in [CMS24] for solving the feasibility problem for ILEP (we give an overview of this procedure in Section 2). In a nutshell, this algorithm solves the linear integer-exponential program by progressively obtaining linearly occurring variables, which are eliminated with a procedure that combines Bareiss's algorithm for Gaussian elimination [Bar68] with a quantifier elimination procedure for Presburger arithmetic [Pre29] (that is, the first-order theory of the structure $\langle \mathbb{N}; 0, 1, +, \leq \rangle$). This "variable elimination step" only preserves the equisatisfiability of the formula; consequently, in the setting of optimization, the algorithm may miss all optimal solutions. We look closely at this issue, and show that the variable elimination step can be strengthened to ensure that at least one optimal solution is preserved (provided one exists). Furthermore, each non-deterministic branch of execution can be associated with an ILESLP whose size is polynomial in the sizes of the intermediate formulae produced during the run. When the execution terminates successfully, this ILESLP encodes the computed solution. Then, the final component of the proof involves analyzing the running time of the algorithm.

Variable elimination. Without going into full-details, one can abstract the "variable elimination step" we seek to define into the template given in Algorithm 1 (ElimVars). It describes a procedure

Algorithm 1 ElimVars: A template for variable elimination.

Input: $\boldsymbol{x}$: variables; f: an objective function; φ: a system of constraints.

1: **while** some variable from $\boldsymbol{x}$ appears in f or φ **do**
2: $\quad(a \cdot x = \tau) \leftarrow$ **guess** an element in $TP(\boldsymbol{x}, f, \varphi)$ ▷ *guesses an equality with* $a \neq 0$
3: $\quad(f, \varphi) \leftarrow \mathit{Elim}(f, \varphi, a \cdot x = \tau)$ ▷ *subproblem in which* $x = \frac{\tau}{a}$
4: **return** (f, φ)

that, given in input a vector of variables $\boldsymbol{x}$ to be eliminated, an objective function f, and some system of constraints φ, iteratively performs the following operations:

1. Guess an equality $a \cdot x = \tau$ from a finite set $TP(\boldsymbol{x}, f, \varphi)$ (line 2), where $a \in \mathbb{Z} \setminus \{0\}$, x is a variable in $\boldsymbol{x}$ occurring in φ or f, and τ is an expression over variables in f or φ other than x.

2. Apply an *elimination discipline Elim* (line 3). This operator updates f and φ to a new objective function and constraint system, representing the subproblem obtained by narrowing the search space to only those solutions where x is set to $\frac{\tau}{a}$.

Slightly overloading terminology from computer algebra, we refer to elements $a \cdot x = \tau$ of $TP(\boldsymbol{x}, f, \varphi)$ as *test points*, emphasizing that ElimVars *tests* the case where x is set to $\frac{\tau}{a}$. The algorithm only explores solutions corresponding to such tests. Hence, if too few test points are used, the algorithm may fail to find any solution to some satisfiable formula, i.e., it might be *incomplete*. Even when it is complete, it may still miss all optimal solutions, if none of them corresponds to some test point. Given a specific class of objective functions and constraint systems, one can therefore ask: *how should the test points be chosen to ensure that the algorithm runs in non-deterministic polynomial time and explores at least one optimal solution?*

Example 2 (ILP with divisibility constraints)**.** *Consider the optimization problem:*

$$\textit{maximize } f(\boldsymbol{y}) \textit{ subject to } \varphi(\boldsymbol{y}) \coloneqq \left(A \cdot \boldsymbol{y} \leq \boldsymbol{b} \wedge \textstyle\bigwedge_{i=1}^{k} m_i \mid \tau_i(\boldsymbol{y})\right), \tag{2}$$

where f is a linear polynomial, A is an integer matrix, $\boldsymbol{b}$ is an integer vector, and each $m_i \mid \tau_i$ is a divisibility constraint *featuring a non-zero divisor $m_i \in \mathbb{Z}$ and a linear polynomial τ_i. Given $a, b \in \mathbb{Z}$, $a \mid b$ is true whenever a is a divisor of b. From quantifier elimination procedures for Presburger arithmetic (see, e.g., [Wei90]), we know that defining the (finite) set of test points as*

$$TP(\boldsymbol{y}, f, \varphi) \coloneqq \left\{ a \cdot x = \tau - s : \begin{array}{r} \textit{the variable } x \textit{ appears in } \varphi \textit{ or } f, \\ (a \cdot x - \tau) \textit{ is either } -x \textit{ or a row of } A \cdot \boldsymbol{y} - \boldsymbol{b}, \\ a \neq 0 \textit{ and } s \in [0..\,|a| \cdot \operatorname{lcm}(m_1, \ldots, m_k) - 1] \end{array} \right\}$$

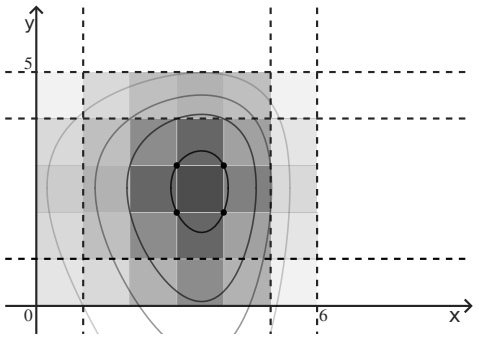
ensures that a solution over $\mathbb{Z}$ is explored. In essence, this set shows that a solution can always be found by shifting the hyperplanes describing the feasible region defined by φ or, when x appears only in f, by shifting the constraint $-x = 0$. In fact, in Lemma 8 (Section 3) we will see that this set also guarantees exploration of an optimal solution, due to the monotonicity of the linear objective f.

Given f, φ and an equality $a \cdot x = \rho$ from $TP(\boldsymbol{y}, f, \varphi)$, we can define Elim as the operator that replaces x with $\frac{\tau}{a}$ in both f and φ (performing basic manipulations to preserve the integrality of the coefficients in φ), and appends the divisibility constraint $a \mid \tau$ to φ. These updates mirror those performed by quantifier elimination procedures for Presburger arithmetic. Complexity-wise, this elimination discipline is suboptimal, as it causes the bit sizes of the integers in φ to grow exponentially in the number of eliminated variables. The results in [CMS24] show how to fix this issue by relying on Bareiss algorithm. We will rely on similar arguments in Section 5.

In the case of non-monotone objective functions, the instantiation of ElimVars given in the above example fails to explore optimal solutions.

Example 3. *Consider the optimization problem*

$$\begin{aligned} \textit{maximize } f(x,y) &\coloneqq 8x + 4y - (2^x + 2^y) \\ \textit{subject to } \varphi(x,y,z) &\coloneqq 0 \leq x \leq 6 \\ &\quad\; 0 \leq y \leq 5 \end{aligned}$$



In the figure, vertical and horizontal lines represent the test points in the set $T \coloneqq TP(\{x,y\}, f, \varphi)$ from Example 2. None intersect an optimal solution, and T is therefore insufficient to solve the problem of maximizing a linear-exponential term subject to an integer linear program.

In order to instantiate ElimVars to the context of ILEP, we must consider a class of objective functions represented as *Linear-Exponential Arithmetic Circuits* (*LEACs*). Informally, a LEAC C is an ILESLP that includes some *free variables* $\boldsymbol{y}$, that is, variables that appear in arithmetic expressions but are not themselves assigned any expression within the straight-line program. For a given *output variable* z in C, the function represented by C takes values for the free variables $\boldsymbol{y}$ as input, evaluates all expressions in the circuit, and returns the integer corresponding to the expression assigned to z. (LEACs are formally defined in Section 4.1; see Definition 2.) The function f from Example 3 can be represented with a LEAC.

Exploring optimal solutions. Returning to Example 3, we can ensure an optimal solution is explored by adding the equalities $x = 3$ and $x = 4$ to the set T. One way of interpreting this addition is by looking at two subproblems: one where x ranges over $[0..3]$, and another where it ranges over $[4..6]$. Within each of these intervals, the function f is monotone in x as both $x = 3$ and $x = 4$ are near a zero of the partial derivative $\frac{\partial f}{\partial x} = 8 - \ln(2) \cdot 2^x$ of f in x. Because of monotonicity, each subproblem can be tackled using the test points from Example 2, and the union of the test points of the two subproblems is exactly the set $T \cup \{x = 3, x = 4\}$.

In essence, our instantiation of ElimVars for ILEP adapts the above observation to the setting of LEACs. We show how to decompose the search space in such a way that the objective function encoded by the LEAC exhibits a form of monotonicity within each region of the decomposition, to then rely on the idea from Example 2 that, for monotone functions, an optimal solution must occur near the boundary of the feasible region. We refer to these decompositions as *monotone decompositions*. Since variables range over $\mathbb{N}$ instead of $\mathbb{R}$, we use *finite differences* instead of derivatives: for a function $f(x, \boldsymbol{y})$ in $1 + d$ variables (in our case, a LEAC) and $p \in \mathbb{N}$, the *p-spaced partial finite difference of f with respect to x*, denoted $\Delta_x^p[f]$, is the function $f(x + p, \boldsymbol{y}) - f(x, \boldsymbol{y})$. The function f is said to be *(x,p)-monotone locally to a set $S \subseteq \mathbb{N}^{1+d}$* if there is a sign $\sim \in \{<, =, >\}$ such that, for every $(u, \boldsymbol{v}) \in S$ with $(u+p, \boldsymbol{v}) \in S$, we have $\Delta_x^p[f](u, \boldsymbol{v}) \sim 0$. (Similarly to Example 2, our instantiation of ElimVars adds divisibility constraints. The integer p in the finite difference corresponds to the least common multiple of the divisors in these constraints.)

Example 4. *Let f and φ be as in Example 3. The 1-spaced partial finite difference in x of f is $\Delta_x^1[f] = 8 - 2^x$. This function is positive for $x \leq 2$, zero at $x = 3$, and negative for $x \geq 4$. Accordingly, the monotone decomposition of the search space features three regions, given by the sets*

of solutions to $\varphi \wedge (x \leq 2)$, $\varphi \wedge (x = 3)$, *and* $\varphi \wedge (x \geq 4)$. *The function* f *is* $(x, 1)$*-monotone locally to each region, and we define the set* $TP(\{x, y\}, f, \varphi)$ *to include* $x = 2$, $x = 3$ *and* $x = 4$.

Complexity. After defining the set of test points by relying on monotone decompositions, most of the technical effort required to prove Theorem 1 is devoted to ensuring that no exponential blow-up occurs during the procedure. (In fact, this effort starts when defining the monotone decompositions, as doing so uncarefully would already cause such a blow-up; see the discussion on page 26.) As already mentioned in Example 2, an important step in avoiding exponential blow-ups is the design of an efficient elimination discipline, which we base on a variation of Bareiss algorithm. Once the elimination discipline is in place, a careful complexity analysis, tracking several parameters of both the integer linear-exponential programs and the LEACs, is required to show that the entire procedure runs (non-deterministically) in polynomial time.

Remark 1. *As noted at the beginning of page 3, the techniques in this paper also appear applicable to quadratic and parametric versions of integer programming. In a nutshell, this is because it is relatively simple to define monotone decompositions in those contexts.*

1.5 Open problems and future directions

The results presented in this paper provide a positive answer to the question of whether optimal solutions to ILEP admit efficient representations, and offer what we believe to be a first satisfactory perspective on the computational differences between ILP and ILEP. Yet this perspective gives rise to several open problems, some of the most interesting of which we outline below.

Among the problems related to the complexity of ILEP, a fundamental question is whether our $\mathrm{FNP}^{\mathrm{NP}}$ upper bound can be improved to $\mathrm{FP}^{\mathrm{NP}}$.

Open problem 1. *Is the optimization problem for integer linear-exponential programs in* $\mathrm{FP}^{\mathrm{NP}}$*?*

Based on our discussion in Section 1.3, this problem can be settled with an algorithm for performing binary search on a large set of ILESLPs. We formalize this objective in the following open problem (here, $\#S$ stands for the cardinality of a set S):

Open problem 2. *Let* S *be the set of all ILESLPs of size at most* k*. Is there an algorithm with runtime polynomial in* k *that, given as input* $\sigma_1, \sigma_3 \in S$*, computes* $\sigma_2 \in S$ *such that the size of each of the sets* $S_1 \coloneqq \{\sigma : [\![\sigma_1]\!]_\bullet \leq [\![\sigma]\!]_\bullet \leq [\![\sigma_2]\!]_\bullet\}$ *and* $S_2 \coloneqq \{\sigma : [\![\sigma_2]\!]_\bullet \leq [\![\sigma]\!]_\bullet \leq [\![\sigma_3]\!]_\bullet\}$ *is in* $\Omega(\#S_1 + \#S_2)$*?*

Although missing a formal connection, the fact that $\textsc{Div}_{\textsc{ileslp}}$ is unlikely to lie in P (Appendix A) suggests that the above open problem may need to be relaxed to also allow for algorithms that run in polynomial time with access to an integer factoring oracle. For example, this would apply to algorithms that first construct an LESLP σ_2 of size at most k, to then check that σ_2 is an ILESLP. Efforts to address Open problem 2 might begin by focusing on non-trivial subsets of S. For instance, one could consider the problem of performing binary search on power circuits of size at most k, hence avoiding rational constants. A closely related open problem is the *successor problem*: given $\sigma_1 \in S$, find (if it exists) $\sigma_2 \in S$ satisfying $[\![\sigma_2]\!]_\bullet = \min\{[\![\sigma]\!]_\bullet : [\![\sigma_1]\!]_\bullet < [\![\sigma]\!]_\bullet\}$.

The connection between $\textsc{Div}_{\textsc{ileslp}}$ and the Sequential Squaring Assumption (Appendix A) suggests that it is unlikely that integer linear-exponential programs with at least three variables can be solved in polynomial time. In contrast, ILP can be solved in polynomial time for any fixed number of variables [Len83]. Can we say more about the complexity of ILEP in fixed dimension?

Open problem 3. *When the number of variables is fixed, can integer linear-exponential programming be solved in polynomial time with access to an integer factoring oracle?*

1.6 ILEP in context

For the interested reader, we conclude this overview by providing a broader perspective on ILEP. A notable trend in computer science sees integer linear programming being used not only in its classical applications (such as scheduling, logistics, and finance) but also in automated reasoning and program analysis. This is due in large part to the advances in *Satisfiability Modulo Theory (SMT)* solvers [BT18]. These solvers bootstrap general (semi-)decision procedures for full first-order logical theories starting from tools that solve the so-called "conjunctive fragment" of these theories. For instance, ILP is the conjunctive fragment of Presburger arithmetic, and SMT solvers rely on tools for ILP to decide the feasibility problem of Presburger arithmetic [KBT14].

One challenge in applying Presburger arithmetic (and thus ILP) to areas such as program analysis stems from limitations in its expressive power. The simplest example of this comes from bit-vector analysis. Let us see a bit-vector b of length n as the non-negative integer $\sum_{i=0}^{n} b[i] \cdot 2^i$, where $b[i]$ denotes the ith entry of b. Presburger arithmetic lacks the ability to express even the simple two-variables formula $\mathit{bit}(b, y)$ asserting that $b[y] = 1$, i.e., that the bit in position y is set. Owing to these limitations, recent research focuses on extending Presburger arithmetic and ILP with additional predicates and functions while retaining decidability [KLN$^+$25, BH24, DHMP24]. A prominent extension is given by *Semenov arithmetic* [Sem84], which adds to Presburger arithmetic the *exponential function* $x \mapsto 2^x$. Although it still cannot express the formula $\mathit{bit}(b, y)$, Semenov arithmetic can reason about *bit sizes*: the formula $2^y \leq x \wedge x < 2 \cdot 2^y$ binds y to be the bit size of x. Because of this, Semenov arithmetic has recently found applications for reasoning about program (non)termination [FG24].

Further extending Semenov arithmetic with the remainder function $(x, y) \mapsto (x \bmod 2^y)$ yields a first-order theory called *Büchi-Semenov arithmetic*. From a logic viewpoint, ILEP is the conjunctive fragment of Büchi-Semenov arithmetic. This theory is more expressive than Semenov arithmetic: back to our toy example, $\mathit{bit}(b, y)$ is definable simply as $\exists z : z = y + 1 \wedge (b \bmod 2^z) - (b \bmod 2^y) \geq 1$. Recent research shows that Büchi-Semenov arithmetic has practical applications in solving string constraints [WCW$^+$23, DHM24]. This also prompted the study of extensions of ILEP featuring *regular predicates* (constraints $x \in R$ where R is a regular expression), though the complexity of the feasibility problem for these extensions cease to be in NP and becomes PSpace-hard [DHM24, Sta25]. It is worth noting that, at the time of writing this paper, all existing tools for Semenov and Büchi-Semenov arithmetic, such as those described in [WCW$^+$23, FG24], are limited to providing yes/no answers or binary-encoded solutions. In this setting, the ILESLPs studied in this paper offer what is arguably the most natural certificate format these tools could use.

Table of contents

Part I
Polynomial-size ILESLPs for optimal solutions

This first part of the paper is fully devoted to proving Theorem 1. After introducing some preliminary definitions and notation (see below), we begin (in Section 2) with a high-level overview of the algorithm from [CMS24] for solving the feasibility problem for ILEP. In particular, we expand on the description given in Section 1.4, identifying the specific step —which we refer to as the "variable elimination step"— where the non-deterministic executions of this algorithm may fail to cover optimal solutions. We also explain how each execution is ultimately constructing an ILESLP.

In Section 3, we present a framework for deriving a variable elimination step tailored for optimization. The framework relies on splitting the search space into regions within which the objective function is (in some sense) monotone. An optimal solution can then be found by examining points that are close to the boundary of these regions. Section 4 instantiates this framework to ILEP. This instantiation reveals a set of additional constraints, beyond the ones required to solve the feasibility problem, that are required to characterize the regions of the decomposition.

The results in Section 4 carry over to Section 5, where we implement the optimum-preserving variable elimination step. Ensuring that the overall procedure runs (non-deterministically) in polynomial time requires great care. To this end, we revisit the arguments from [CMS24] concerning the integration of Bareiss algorithm into the quantifier elimination procedure of Presburger arithmetic, and show that the constraints added by the monotone decomposition retain enough structure to allow a suitable variation of Bareiss algorithm to be successfully implemented.

Finally, Section 6 presents the complete optimization procedure for integer linear-exponential programming. From the correctness and complexity analysis of this procedure, we conclude that its output is a polynomial-size ILESLP, thereby proving Theorem 1.

We now present the preliminaries for this part of the paper. Some of the concepts introduced here reiterate those from Section 1, albeit given in a slightly more formal manner.

Basic notation. For $a \in \mathbb{R}$, we write $|a|$, $\lceil a \rceil$, and $\log a$ for the *absolute value*, *ceiling*, and (if $a > 0$) the *binary logarithm* of a. All numbers encountered by our algorithm are encoded in binary; assuming that $n \in \mathbb{Z}$ is represented using $\lceil \log(|n| + 1) \rceil + 1$ bits. For $a, b \in \mathbb{R}$, we write $[a..b]$ to denote the set $\{n \in \mathbb{Z} : a \leq n \leq b\}$. Vectors are denoted using boldface letters, as in $\boldsymbol{x}$ or $\boldsymbol{y}$. We write $\#\boldsymbol{x}$ for the number of entries in $\boldsymbol{x}$; similarly, $\#S$ stands for the cardinality of a finite set S.

Integer Linear-Exponential Terms. A *linear-exponential term* τ is an expression

$$\sum\nolimits_{i=1}^{n} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum\nolimits_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j})\right) + d,$$

where $a_i, b_i, c_{i,j} \in \mathbb{Z}$ are the *coefficients* of the term and $d \in \mathbb{Z}$ is its *constant*. If all b_i and $c_{i,j}$ are zero then the term is said to be *linear*. If $a_i \neq 0$, we call $a_i \cdot x_i$ a *linear occurrence* of x_i. If $b_i = 0$, we say that x_i *does not occur in exponentials*; this is weaker than saying that x_i *only occurs linearly*, as in this case we also have $c_{i,j} = 0$ for all $j \in [1..n]$. We assume all variables used in linear-exponential terms to belong to a totally-ordered countable set $\mathbb{X}$, and write $\tau(\boldsymbol{x})$ if all variables in the term τ are from the vector (or set) $\boldsymbol{x}$. The 1-norm of τ is defined as $\|\tau\|_1 \coloneqq \sum_{i=1}^{n}(|a_i| + |b_i| + \sum_{j=1}^{n} |c_{i,j}|) + |d|$. The size of τ is defined as the number of symbols needed to write down the term, assuming that integers are encoded in binary, and that the kth variable in the ordering of $\mathbb{X}$ requires k bits. Given a map $\nu\colon X \to \mathbb{N}$, where X is a subset of $\mathbb{X}$ including the variables in $\boldsymbol{x}$, we write $\nu(\tau)$ for the

integer obtained by *evaluating* τ *on* ν, that is, replacing every variable x occurring in τ with $\nu(x)$, and evaluating all operations in the resulting term.

For a variable $y \in \mathbb{X}$ and $b \in \mathbb{N}$, we write $[y \mapsto b]$ for the map ν with domain $\{y\}$ and such that $\nu(y) = b$. Let $\nu_1 \colon X_1 \to \mathbb{N}$ and $\nu_2 \colon X_2 \to \mathbb{N}$ be two maps. The expression $\nu_1 + \nu_2$ defines the map $(\nu_1 + \nu_2) \colon X_1 \cup X_2 \to \mathbb{N}$ assigning $\nu_1(x) + \nu_2(x)$ to every $x \in X_1 \cup X_2$, where we assume $\nu_i(x) = 0$ whenever $x \notin X_i$. Therefore, $\nu + [y \mapsto b]$ stands for the map obtained from ν by adding b to the value given to y (again, assuming $\nu(y) = 0$ if $y \notin X$).

Integer Linear-Exponential Programs. A *(integer) linear-exponential program* φ is a conjunction of constraints $\tau = 0$ and $\tau \leq 0$, where τ is a linear-exponential term. If all terms are linear, then φ is an *(integer) linear program*. We sometimes diverge from this syntax, but the intended meaning of the constraints should always be clear from the context. For instance, we sometimes write $\tau_1 \leq \tau_2$ as a shorthand for $\tau_1 - \tau_2 \leq 0$, and $\tau_1 < \tau_2$ as a shorthand for $\tau_1 - \tau_2 + 1 \leq 0$. We write $\varphi(\boldsymbol{x})$ when the free variables of φ are from the vector $\boldsymbol{x}$.

While linear-exponential programs only feature equalities and inequalities, symbolic procedures for ILEP, such as the one developed in [CMS24], require the introduction of additional *divisibility constraints* $d \mid \tau$, where τ is a linear-exponential term, $d \in \mathbb{N}$ is non-zero, and $\mid$ is the *divisibility predicate*, $\{(d, n) \in \mathbb{Z} \times \mathbb{Z} : n = k \cdot d \text{ for some } k \in \mathbb{Z}\}$. Without loss of generality, we assume all integers in the term τ to belong to $[0..d-1]$; our procedures will tacitly enforce this assumption by reducing all integers modulo d. We say that the linear-exponential program is *with divisions* if we allow divisibility constraints to occur in it. For simplicity of the presentation, we also sometimes consider arbitrary *formulae* from Büchi-Semenov arithmetic. In this theory, linear-exponential programs with divisions are extended to include the standard features of first-order logic, such as conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), implication ($\Longrightarrow$) and first-order quantification ($\forall$ and $\exists$). For example, in the forthcoming sections we will often write equalities $u = 2^{x-y}$, which should be seen as shortcuts for formulae $\exists z\, (u = 2^z \wedge z = x - y)$, where z is a fresh variable. Note that, since we are only interested in non-negative *integer* solutions (see below), $u = 2^{x-y}$ implies $x \geq y$.

Let φ be a linear-exponential program with divisions. We write:

- $\#\varphi$ for the number of constraints (inequalities, equalities and divisibility constraints) in φ;
- $\mathrm{vars}(\varphi)$ for the set of all variables occurring in φ;
- $\mathrm{terms}(\varphi)$ for the set of all terms τ occurring in inequalities $\tau \leq 0$ or equalities $\tau = 0$ of φ;
- $\|\varphi\|_1 \coloneqq \max\{\|\tau\|_1 : \tau \in \mathrm{terms}(\varphi)\}$;
- given a vector $\boldsymbol{x}$ of variables, $\mathit{mod}(\boldsymbol{x}, \varphi)$ for the least common multiple of the divisors d of the divisibility constraints $d \mid \tau$ of φ in which at least one variable from $\boldsymbol{x}$ occur (with a non-zero coefficient). We omit $\boldsymbol{x}$, and simply write $\mathit{mod}(\varphi)$, when considering all variables in φ.

The size of φ is defined as the number of symbols required to write it down (following the same assumptions used for defining the size of a term).

A map $\nu \colon X \to \mathbb{N}$, where X is a finite subset of $\mathbb{X}$, is a *solution* to a linear-exponential program with divisions φ whenever (i) X includes all variables occurring in φ, and (ii) replacing each variable x in φ with $\nu(x)$ lead to all constraints (inequalities, equalities and divisibilities) being satisfied. For convenience, we sometimes see the set of solutions to φ not as a set of maps but as a subset $S \subseteq \mathbb{N}^d$, where d is the number of variables in φ. The ith entry of each vector in S corresponds to the ith variable occurring in φ, with respect to the total order of the set $\mathbb{X}$.

Integer Linear-Exponential Programming (ILEP). By *Integer Linear-Exponential Programming* we mean solving the maximization problem (or the analogous minimization problem)

$$\text{maximize } \tau(\boldsymbol{x}) \text{ subject to } \varphi(\boldsymbol{x}),$$

where τ is a linear-exponential term (the *objective function*) and φ is a linear-exponential program (without divisions). Unless otherwise stated, we stress that all the variables in an instance of integer linear-exponential programming range over the natural numbers.

A map $\nu\colon X \to \mathbb{N}$, is a solution to an instance of integer linear-exponential programming whenever X includes all variables occurring in τ and φ, and ν is a solution to φ. The *value* of the objective function τ for the solution ν is the integer $\nu(\tau)$.

Substitutions. For technical reasons, we need an ad-hoc form of term substitution. We denote such a substitution with $[\frac{\tau}{a} \,/\, b \cdot x]$, where τ is a linear-exponential term, x is a variable, and a and b are two non-zero integers. (This substitution can be interpreted as enforcing the equality $a \cdot b \cdot x = \tau$.) When applied to a linear-exponential term ρ, the resulting term $\rho[\frac{\tau}{a} \,/\, b \cdot x]$ is constructed as follows:

1. Multiply every integer in ρ by $|a|$.
2. Consider the linear occurrence of x in ρ (if there is one). Try to factorize its coefficient as $a \cdot b \cdot c$, for some non-zero $c \in \mathbb{Z}$. If successful, replace $a \cdot b \cdot c \cdot x$ with $c \cdot \tau$.

Observe that, to eliminate x using this substitution, we need to ensure that it only occurs linearly in ρ, and that its coefficient is divisible by b. We omit a and/or b from $[\frac{\tau}{a} \,/\, b \cdot x]$ when they are equal to one, writing for instance $[\tau \,/\, x]$ instead of $[\frac{\tau}{1} \,/\, 1 \cdot x]$.

We will also need to simultaneously apply multiple substitutions to terms. Consider distinct variables $x_1, \ldots, x_n$, terms $\tau_1, \ldots, \tau_n$ not featuring these variables, and two non-zero integers a and b. By *simultaneously applying the substitutions* $[\frac{\tau_1}{a} \,/\, b \cdot x_1], \ldots, [\frac{\tau_n}{a} \,/\, b \cdot x_n]$ to the term ρ we mean the process of first multiplying every integer in ρ by $|a|$, to then apply to the resulting term the substitutions $[\tau_1 \,/\, a \cdot b \cdot x_1], \ldots, [\tau_n \,/\, a \cdot b \cdot x_n]$ (in any order). So, differently from sequentially applying $[\frac{\tau_1}{a} \,/\, b \cdot x_1], \ldots, [\frac{\tau_n}{a} \,/\, b \cdot x_n]$, simultaneous substitutions multiply by $|a|$ only once.

When applying a substitution $[\frac{\tau}{a} \,/\, b \cdot x]$ to a linear-exponential program with divisions φ, the resulting program $\varphi[\frac{\tau}{a} \,/\, b \cdot x]$ is constructed as follows:

- For every equality $\rho = 0$ or inequality $\rho \leq 0$ occurring in φ, replace ρ with $\rho[\frac{\tau}{a} \,/\, b \cdot x]$.
- Replace every divisibility constraint $d \mid \rho$ occurring in φ with $(|a \cdot b| \cdot d) \mid \rho[\frac{\tau}{a \cdot b} \,/\, x]$.

2 The algorithm for deciding ILEP feasibility, briefly

We present a high-level overview of the procedure from [CMS24] for deciding the feasibility problem of ILEP, highlighting its properties in the context of optimization. As we will see, the main loop of the procedure can be divided in four steps (Steps I–IV). Steps I and III preserve optimal solutions; we can thus use them as black-boxes when designing our optimization procedure. Appendix B gives more information on these two steps, as well as their pseudocode. In contrast, Step II and IV may discard all optimal solutions. Step II is the main "variable elimination step", which we will focus on in the upcoming sections of this part of the paper. Step IV is a simplified variant of Step II, and will be handled directly when presenting the full optimization procedure in Section 6.

Let φ be an input ILEP. As a preliminary step, the procedure in [CMS24] non-deterministically guesses an ordering θ of the form $2^{x_n} \geq \cdots \geq 2^{x_1} \geq 2^{x_0} = 1$. Here, $x_1, \ldots, x_n$ is a permutation

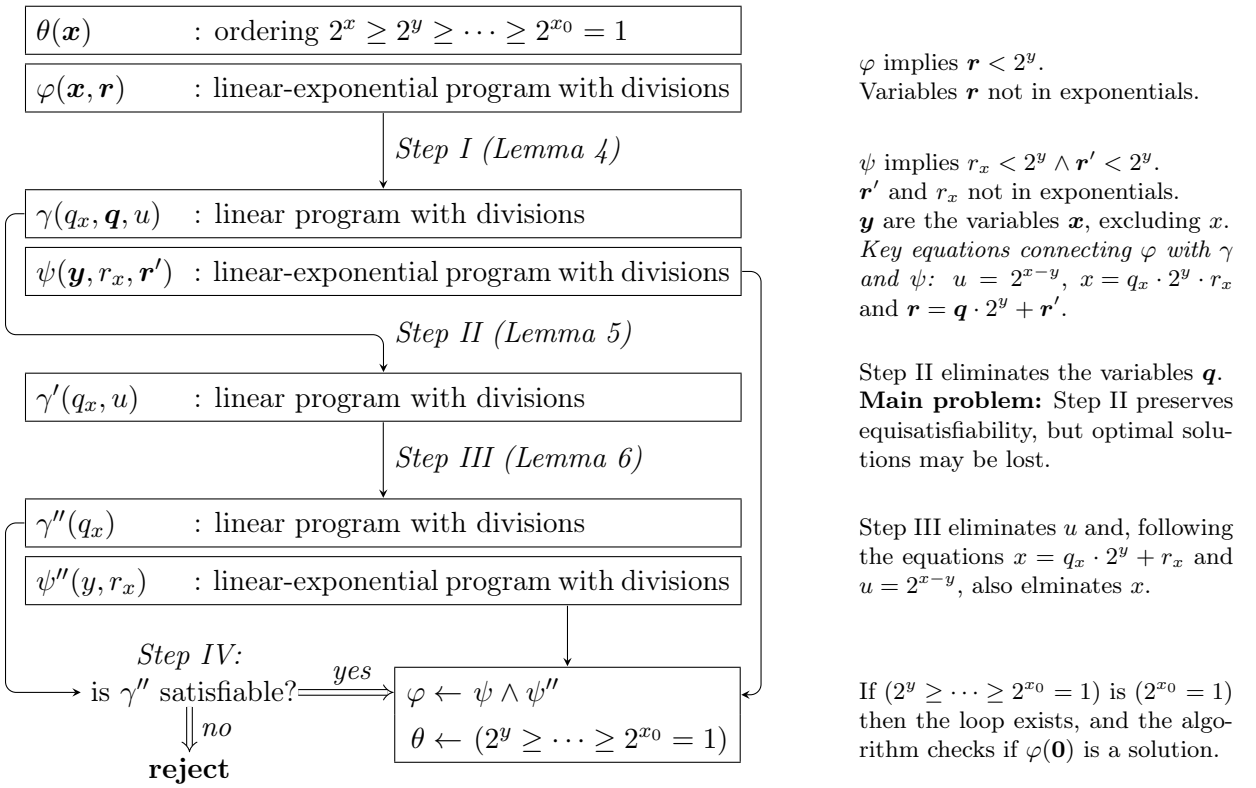


Figure 1: Flowchart of the main loop of [CMS24].

of the variables in φ, whereas x_0 is a fresh variable introduced to handle the termination of the algorithm. Note that $\varphi \wedge (2^{x_0} = 1)$ is equivalent to the disjunction $\bigvee_{\theta \in \Theta}(\varphi \wedge \theta)$ ranging over the set of all orderings Θ. In the context of optimization, no optimal solution is lost in this step of the procedure: it suffices to optimize locally to each disjunct $\varphi \wedge \theta$, and then take the maximum (or minimum) of the resulting optimal solutions.

After guessing the ordering θ, the algorithm enters its main loop, where it iteratively eliminates from φ and θ all variables $x_1, \ldots, x_n$, starting from the largest one in θ. These eliminations introduce new *remainder variables* $\boldsymbol{r}$, variables that never occur in exponentials, and are always smaller than the largest term in θ. After eliminating $x_n, \ldots, x_i$, the ordering θ is updated to $2^{x_{i-1}} \geq \cdots \geq 2^{x_0} = 1$, and all remainder variables are constrained to be smaller than $2^{x_{i-1}}$. After n iterations of the main loop, θ reduces to just $2^{x_0} = 1$, and φ becomes a formula $\varphi'(x_0, \boldsymbol{r})$ that implies $\boldsymbol{r} < 2^{x_0}$. The main loop terminates. The only possible solution for $\varphi' \wedge (2^{x_0} = 1)$ is $(x_0, \boldsymbol{r}) = \mathbf{0}$; and if this is a solution, then the original formula φ is satisfiable.

We now describe an iteration of the main loop, dividing it in the aforementioned Steps I–IV.[3] To aid in following the interactions between these steps, Figure 1 is provided alongside the description.

Step I (division by 2^y). Let 2^x and 2^y be the largest and second-largest terms in θ (so, $x \neq x_0$). The first step to eliminate x is to symbolically divide all linear occurrences of this variable, as well as all remainder variables $\boldsymbol{r}$, by 2^y. That is, the algorithm rewrites the linear occurrences of x as

[3] Our division of the procedure into steps differs from that used by the authors of [CMS24] to describe the algorithm. Specifically, we have included lines 4–14 of Algorithm 2 of [CMS24] as part of the Step I, instead of considering them separately. This adjustment is made solely for the sake of presentation clarity.

$q_x \cdot 2^y + r_x$, and $\boldsymbol{r}$ as $\boldsymbol{q} \cdot 2^y + \boldsymbol{r'}$, where the fresh variables $(q_x, \boldsymbol{q})$ and $(r_x, \boldsymbol{r})$ represent the *quotient* and *remainder* of the division by 2^y, respectively. The variables r_x and $\boldsymbol{r'}$ are remainder variables in the next iteration of the main loop; and indeed the algorithm adds the constraints $r_x < 2^y$ and $\boldsymbol{r'} < 2^y$ to the system. The variables q_x and $\boldsymbol{q}$ are called *quotient variables*. The procedure introduces a further expression $u = 2^{x-y}$, with u fresh, and through several manipulations decouples the quotient variables from all other variables except u (this is similar to a *monadic decomposition* [Lib03]). The key equivalences enabling this decoupling are given in the next lemma. In the context of the algorithm, the integer t in this lemma corresponds to a linear term featuring the variable u and the quotient variables $\boldsymbol{q}$, whereas the integer s corresponds to a linear-exponential term involving the remainder variables r_x and $\boldsymbol{r'}$, linear occurrences of y, and all the variables in θ that are distinct from x and y. The key point is that, in the right-hand side of the equivalences in the lemma, t and s are decoupled (that is, they never appear together within a single (in)equality).

Lemma 3 [CMS24]**.** *Let $C, D \in \mathbb{Z}$, with $C \leq D$. For $y \in \mathbb{N}$, $t \in \mathbb{Z}$, and $s \in [C \cdot 2^y .. D \cdot 2^y]$, the following equivalences hold:*

1. $t \cdot 2^y + s = 0 \iff \bigvee_{r=C}^{D} \big(t + r = 0 \wedge s = r \cdot 2^y\big)$,
2. $t \cdot 2^y + s \leq 0 \iff \bigvee_{r=C}^{D} \big(t + r \leq 0 \wedge (r-1) \cdot 2^y < s \leq r \cdot 2^y\big)$,
3. $t \cdot 2^y + s < 0 \iff \bigvee_{r=C}^{D} \big(t + r + 1 \leq 0 \wedge s = r \cdot 2^y\big) \vee \big(t + r \leq 0 \wedge (r-1) \cdot 2^y < s < r \cdot 2^y\big)$.

To see Lemma 3 in action, consider the equality $2^x - 2^y + y - z = 0$. Assuming $\theta = (2^x \geq 2^y \geq 2^z)$ and $u = 2^{x-y}$, we can rewrite this equality as $(u-1) \cdot 2^y + y - z = 0$. Moreover, we see that θ implies that $y - z$ belongs to $[0 \cdot 2^y .. 1 \cdot 2^y]$. Then, Lemma 3.1 tells us that the equality can be rewritten as $\bigvee_{r=0}^{1}((u-1) + r = 0 \wedge (y - z) = r \cdot 2^y)$. Here, the equation $(u-1) + r = 0$ is in a sense the "quotient" of the division by 2^y, whereas $y - z = r \cdot 2^y$ indicates properties of the "remainder" of the division (in this case, that $y - z$ has remainder zero when divided by 2^y).

The effects of Step I of the main loop are formalized in the next lemma, where the output formula γ_β contains the "quotients" of the divisions by 2^y, and ψ_β contains constraints on the "remainders".

Lemma 4 [CMS24]**.** *There is a non-deterministic procedure with the following specification:*

Input: $\theta(\boldsymbol{x})$ *: ordering of exponentiated variables;*
[Below, let 2^x and 2^y be the largest and second-largest terms in this ordering, and let $\boldsymbol{y}$ be the vector obtained by removing x from $\boldsymbol{x}$.]
$\varphi(\boldsymbol{x}, \boldsymbol{r})$ *: linear-exponential program with divisions, implying $\boldsymbol{r} < 2^x$. Variables $\boldsymbol{r}$ do not occur in exponentials.*

Output of each branch (β):

$\gamma_\beta(q_x, \boldsymbol{q}, u)$ *: linear program with divisions;*
$\psi_\beta(\boldsymbol{y}, r_x, \boldsymbol{r'})$ *: linear-exponential program with divisions, implying $r_x < 2^y \wedge \boldsymbol{r'} < 2^y$. Variables r_x and $\boldsymbol{r'}$ do not occur in exponentials.*

The variables q_x, $\boldsymbol{q}$, u, $\boldsymbol{y}$, r_x and $\boldsymbol{r'}$ are common to all outputs, across all non-deterministic branches. The procedure ensures that the system

$$\begin{bmatrix} x \\ \boldsymbol{r} \end{bmatrix} = \begin{bmatrix} q_x \\ \boldsymbol{q} \end{bmatrix} \cdot 2^y + \begin{bmatrix} r_x \\ \boldsymbol{r'} \end{bmatrix}, \tag{3}$$

yields a one-to-one correspondence between the solutions of $\varphi \wedge \theta$ and the solutions of the formula $\bigvee_\beta \big(\gamma_\beta \wedge \psi_\beta \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x) \wedge \theta\big)$. This correspondence is the identity for the variables these two formulae share (that is, the variables in $\boldsymbol{x}$).

The one-to-one correspondence described in Lemma 4 implies that no solution is lost in this step of the procedure. We can therefore use Step I also in the context of optimization.

Step II (variable elimination: the problematic step). The procedure now considers the *linear* program with divisions $\gamma(q_x, \boldsymbol{q}, u)$ in output of Step I, and applies a quantifier elimination procedure to remove all the quotient variables in $\boldsymbol{q}$. (Not q_x, this quotient variable cannot be eliminated yet, because the equalities $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$ shown in Lemma 4 make the variable u depend exponentially on q_x.) This elimination step mixes ingredients from the quantifier elimination procedure for Presburger arithmetic [Wei90] with Bareiss' version of Gaussian elimination [Bar68]. As in the case of the former of these two procedures, this step introduces new divisibility constraints. Here is the specification of Step II:

Lemma 5 [CMS24]**.** *There is a non-deterministic procedure with the following specification:*

Input: $\gamma(q_x, \boldsymbol{q}, u)$ *: linear program with divisions.*

Output of each branch (β)***:*** $\gamma'_\beta(q_x, u)$ *: linear program with divisions.*

The procedure ensures that the formulae $\exists \boldsymbol{q}\, \gamma$ *and* $\bigvee_\beta \gamma'_\beta$ *are equivalent. Let* $\boldsymbol{q} = (q_1, \ldots, q_k)$*. Furthermore, for every branch* β*, there is a system of equalities*

$$a_1 \cdot q_1 = \tau_1(u, q_x)\,,\ \ldots\ ,\ a_k \cdot q_k = \tau_k(u, q_x)\,, \tag{4}$$

where each $a_i \in \mathbb{Z}$ *is non-zero and each* τ_i *is a linear term, with the following property. The formula* γ'_β *is obtained from* γ *by performing the sequence of substitutions* $[\frac{\tau_1}{a_1} / q_1], \ldots, [\frac{\tau_k}{a_k} / q_k]$ *and conjoining the system of divisibilities* $(a_1 \mid \tau_1(u, q_x)) \wedge \cdots \wedge (a_k \mid \tau_k(u, q_x))$*.*

Concerning optimization, the guarantees achieved by this crucial step of the procedure are too weak. Rather than establishing a one-to-one correspondence between the solutions of the input and those of the outputs, it only achieves an equivalence with respect to the variables q_x and u. Notably, if some variables from $\boldsymbol{q}$ appear in the objective function, then optimal solutions may be lost.

Step III (elimination of x and u). The third step of the main loop is somewhat similar to the first one. We start with the formula $\gamma'(q_x, u)$ obtained from Step II, add the constraints $x = q_x \cdot 2^y + r_x$ and $u = 2^{x-y}$, and decouple q_x from all other variables. By using machinery developed by Semenov in [Sem84], this decoupling makes it possible to eliminate the variables x and u.

Here is the specification of Step III:

Lemma 6 [CMS24]**.** *There is a non-deterministic procedure with the following specification:*

Input: $\gamma'(q_x, u)$ *: linear program with divisions.*

Output of each branch (β)***:*** $\gamma''_\beta(q_x)$ *: linear program with divisions;*
$\psi''_\beta(y, r_x)$ *: linear-exponential program with divisions.*

The procedure ensures that the equation

$$x = q_x \cdot 2^y + r_x \tag{5}$$

yields a one-to-one correspondence between the solutions of $\gamma' \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x)$ *and the solutions of* $\bigvee_\beta \left(\gamma''_\beta \wedge \psi''_\beta\right)$*. This correspondence is the identity for the variables these two formulae share (that is,* y*,* q_x *and* r_x*).*

As in the case of Step I, the one-to-one correspondence described in Lemma 6 ensures that optimal solutions are preserved during this step of the main loop.

Step IV (elimination of q_x). After Step III completes, we are left with its output formulae $\gamma''(q_x)$ and $\psi''(y, r_x)$, and the formula $\psi(\boldsymbol{y}, r_x, \boldsymbol{r'})$ computed in Step I. The main loop of now performs one last operation: it checks whether γ'' (a univariate linear program with divisions) is satisfiable. If it is, γ'' can be replaced with $\top$, effectively eliminating the variable q_x. (Otherwise, the non-deterministic branch of the program rejects.) An alternative way of implementing Step IV is to apply the variable elimination procedure underlying Lemma 4, but again this may cause the algorithm to lose all optimal solutions. In this particular case, since γ'' only features the variable q_x, the formula constructed by the variable elimination procedure simply replaces q_x with an integer c; i.e., the system of equalities analogous to the one in Equation (4) simplifies in this case to just $q_x = c$.

This concludes the current iteration of the main loop of the procedure. If y is not the variable x_0, the loop performs another iteration. In that iteration, the input to Step I becomes the ordering θ' obtained from θ by removing the term 2^x (2^y is now the largest term), together with the linear-exponential program with divisions $\psi(y, r_x) \wedge \psi''(\boldsymbol{y}, r_x, \boldsymbol{r'})$.

2.1 Where are the ILESLPs?

As emphasized in Lemmas 4 to 6, the procedure in [CMS24] is in a sense guided by Equations (3) to (5). Upon closer inspection, we see that these equations are constructing an ILESLP. Let us reason bottom-up and suppose that we have constructed an ILESLP σ that is a solution to the formula $\psi(y, r_x) \wedge \psi''(\boldsymbol{y}, r_x, \boldsymbol{r'}) \wedge \theta'$ described above. We construct an ILESLP that is a solution for $\varphi \wedge \theta$ by appending further assignments to σ. The first three assignments are

$$z_1 \leftarrow 2^y, \quad z_2 \leftarrow c \cdot z_1, \quad x \leftarrow z_2 + r_x,$$

where z_1 and z_2 are auxiliary fresh variables, and c is the integer in the equation $q_x = c$. We are essentially performing the assignment $x \leftarrow c \cdot 2^y + r_x$, accordingly to Equation (5). Observe that σ already assigns expressions to y and r_x. Next, we add assignments to represent each variable in $\boldsymbol{r}$. For each variable v belonging to $\boldsymbol{r}$ we have

$$\begin{aligned}
v &= q_v \cdot 2^y + v' && \text{(Equation (3), where } v' \text{ is some variable in } \boldsymbol{r'}\text{)} \\
&= \frac{\tau(u, q_x)}{a} \cdot 2^y + v' && \text{(Equation (4))} \\
&= \frac{b \cdot 2^{x-y} + d}{a} \cdot 2^y + v' && \text{(using } u = 2^{x-y} \text{ and } q_x = c\text{)} \\
&= \frac{b \cdot 2^x + d \cdot 2^y}{a} + v',
\end{aligned}$$

for some integers a, b, d, with $a \neq 0$. We can easily add assignments to σ to obtain $v \leftarrow \frac{b \cdot 2^x + d \cdot 2^y}{a} + v'$. The resulting ILESLP is guaranteed to be a solution to $\varphi \wedge \theta$.

2.2 OptILEP: from feasibility to optimization

Following the above description of the procedure from [CMS24], it should be now clear that a way to obtain a procedure for the optimization problem of ILEP is to focus on the "variable elimination step" (Step II), strengthening it into a procedure that is guaranteed to explore an optimal solution. In the remainder of the paper, we refer to the resulting procedure as OptILEP. We will present the pseudocode of this procedure in Section 6 (see Algorithm 6). For now, the specific details of the procedure are unimportant; the only features to keep in mind are the following:

- The procedure begins by guessing an ordering θ of the form $2^{x_n} \geq \cdots \geq 2^{x_1} \geq 2^{x_0} = 1$, where $x_1, \ldots, x_n$ are the variables appearing input instance of ILEP.

- It then iteratively eliminates the variables $x_n, \dots, x_1$ (in this order), updating both the linear-exponential program and the objective function. Every iteration of this "main loop" appeals to Step I and Step III of [CMS24], interposed with an optimum-preserving "variable elimination step". This elimination step instantiates the template given by Algorithm 1 (ElimVars), introduced in Section 1.4. The main loop concludes with a step analogous to Step IV, modified along the same lines as Step II to ensure preservation of optimal solutions.

3 Exploring optimal solutions through monotone decompositions

As explained in the overview given in Section 1.4, the template for the "variable elimination step" given by Algorithm 1 (ElimVars) eliminates some variables $\boldsymbol{x}$ from a system of constraints φ (in which the variables $\boldsymbol{x}$ occur linearly) and an objective function f, by iteratively

1. Guessing an equation $a \cdot x = \tau$ from a finite set of *test points* $TP(\boldsymbol{x}, f, \varphi)$, where a is a non-zero integer, and x is a variable from $\boldsymbol{x}$ that still occurs in f or φ.
2. Appealing to an *elimination discipline Elim*, which updates f and φ by "replacing x with $\frac{\tau}{a}$". (Intuitively, this means that we will only be searching for solutions lying inside the hyperplane described by the equation $a \cdot x = \tau$.)

In this section, we describe a method, based on *monotone decompositions* of the search space, for constructing sets of test points that are guaranteed to preserve at least one optimal solution. We develop the approach in a general setting, where the objective function is treated as a black box.

Some notation. Given $d \in \mathbb{N}$ and $i \in [1..d]$, we write $\boldsymbol{e}_i^d$ for the i-th vector of the canonical basis of $\mathbb{R}^d$ (i.e., $\boldsymbol{e}_i^d$ has a 1 at the i-th component and 0 elsewhere); omitting the superscript d when clear from the context. A set $S \subseteq \mathbb{N}^d$ is said to be (i,p)-*periodic* if for any $\boldsymbol{v} \in S$ and $\boldsymbol{v} + m \cdot \boldsymbol{e}_i \in S$ with $m \geq p$, we also have $\boldsymbol{v} + p \cdot \boldsymbol{e}_i \in S$. We write $\Delta_i^p[f]$ for the *i-th p-spaced partial finite difference* of a function $f \colon \mathbb{R}^d \to \mathbb{R}$. It is defined as $\Delta_i^p[f](\boldsymbol{x}) \coloneqq f(\boldsymbol{x} + p \cdot \boldsymbol{e}_i) - f(\boldsymbol{x})$, for every $\boldsymbol{x} \in \mathbb{R}^d$.

A function $f \colon \mathbb{R}^d \to \mathbb{R}$ is said to be (i,p)-*monotone locally to a set* S if there is a sign $\sim \in \{<, =, >\}$ such that for every $\boldsymbol{v} \in S$ with $\boldsymbol{v} + p \cdot \boldsymbol{e}_i \in S$, we have $\Delta_i^p[f](\boldsymbol{v}) \sim 0$. When considering logical formulae, we will abuse this notation and, in the above definitions, replace the indices of the entries of vectors by variable names. For instances, we will say that the set of solution of a formula $\varphi(\boldsymbol{x})$ is (x,p)-periodic, with x being a variable from $\boldsymbol{x}$, and we will write $\Delta_x^p[f]$ for the p-spaced partial finite difference of f with respect to (the coordinate corresponding to) x.

Locating optimal solutions for monotone functions. We start by adapting a folklore result from Presburger arithmetic to ILEP: with respect to a linearly occurring variable, the set of solutions of an integer linear-exponential program is periodic. (See [Smo91, Theorem 4.10] for a similar result.)

Lemma 7. *Let φ be a linear-exponential program with divisions, and let x be a variable occurring linearly in φ. The set of solutions of φ is $(x, mod(x,\varphi))$-periodic.* *Proof in page 92*

The periodic behavior described in the above lemma implies that, for monotone functions, optimal solutions are located near the boundary of the feasible region described by the integer program. This boundary is determined by equalities of $\tau = 0$ that are derived from (in)equalities $\tau \sim 0$ appearing in the program (where $\sim \in \{=, \leq\}$). To ensure we find an optimal solution, it suffices to examine shifted versions of these boundary equalities, that is equalities of the form $\tau + s = 0$, where s ranges over a small set of integer offsets.

Lemma 8. *Let $\varphi(\boldsymbol{x})$ be a linear-exponential program with divisions, and let x be a variable occurring linearly in φ. Let $p \coloneqq mod(x, \varphi)$, and let $f(\boldsymbol{x})$ be a (x, p)-monotone function locally to the set of solutions to φ. If the instance (f, φ) has a maximum (analogously, a minimum), then it has one satisfying an equation $a{\cdot}x{+}\tau{+}r = 0$, where $(a{\cdot}x{+}\tau) \in \text{terms}(\varphi \wedge x \geq 0)$, $a \neq 0$, and $r \in [0..\, |a|{\cdot}p{-}1]$.* *Proof in page 92*

By considering $f(x) = x$ and considering the minimization problem, Lemma 8 simplifies to the following corollary. This corollary, in fact, captures the core argument found in nearly all proofs of quantifier elimination in Presburger arithmetic (see, e.g., [Wei90, Lemma 2.6]).

Corollary 2. *Let φ be a linear-exponential program with divisions, and let x be a variable occurring linearly in φ. If φ has a solution, then it has one satisfying an equation $a \cdot x + \tau + r = 0$, where $(a \cdot x + \tau) \in \text{terms}(\varphi \wedge x \geq 0)$, $a \neq 0$, and $r \in [0..\, |a| \cdot mod(x, \varphi) - 1]$.* *Proof in page 93*

Locating optimal solutions for non-monotone functions. Lemma 8 suggests a natural strategy for tackling optimization problems involving non-monotone functions: partition the search space into multiple regions where the function becomes monotone. This idea is formalized with the subsequent definition of *monotone decomposition* and Lemma 9. It is important to note that, depending on the objective function, constructing such a decomposition with only regions that can be characterized with integer linear-exponential programs (or other desired classes of constraints) can be highly non-trivial or even impossible. In the next section, we show how to achieve this only in the specific setting needed to solve the ILEP optimization problem.

Definition 1 (Monotone decomposition)**.** *A (i, p)-monotone decomposition of $S \subseteq \mathbb{N}^d$ for a function $f \colon \mathbb{R}^d \to \mathbb{R}$ is a finite family $R_1, \ldots, R_t \subseteq \mathbb{N}^d$ such that (i) $S = \bigcup_{j=1}^{t} R_j$ and (ii) for every $j \in [1..t]$, $S \cap R_j$ is (i, p)-periodic and f is (i, p)-monotone locally to $S \cap R_j$.*

Lemma 9. *Let φ be a linear-exponential program with divisions, and x be a variable occurring linearly in φ. Suppose that the set of solutions to φ has a $(x, mod(x, \varphi))$-monotone decomposition $R_1, \ldots, R_t$ for a function f, where each R_i is the set of solutions of an integer linear-exponential program with divisions ψ_i in which x occurs linearly. If the instance (f, φ) has a maximum (analogously, a minimum), then it has one satisfying an equation $a \cdot x + \tau + r = 0$ such that $a \neq 0$, $(a \cdot x + \tau) \in \text{terms}(\psi_i \wedge x \geq 0)$ and $r \in [0..\, |a| \cdot mod(x, \psi_i) - 1]$, for some $i \in [1..t]$.* *Proof in page 93*

Remark 2. *Consider φ, x, and f as in Lemma 9. By defining $TP(x, f, \varphi)$ as the set of all equalities $a{\cdot}x{+}\tau{+}r = 0$ specified in that lemma, we are guaranteed to explore at least one optimal solution (if optimal solutions exists). Notably, when designing a non-deterministic polynomial-time procedure, neither the number of regions nor the number of constraints required to describe each region is inherently restrictive. The key requirement is instead the ability to guess* a single equation *involving the variable x targeted for elimination. This means that, while the total number of distinct constraints containing x can be at most exponential across regions, the number of constraints independent of x can be arbitrarily large (and these need not even be expressible as linear-exponential programs).*

Remark 3. *All lemmas in this section were formulated for linear-exponential programs with divisions. However, upon inspecting their proofs, it should be evident that the only crucial assumption is the* linear occurrence *of x. Indeed, these lemmas could have been stated for any expansion of integer linear programming augmented with arbitrary functions, provided that "occurring linearly" is interpreted as "occurring only within the scope of addition".*

4 Monotone decompositions for ILEP

We now specialize the setting from the previous section, constructing monotone decompositions for the instances that ElimVars must be able to handle in order for OptILEP to explore optimal solutions. In doing so, great attention must be paid to ensure that the formulae and ILESLPs computed throughout the procedure remain of polynomial size. Proving that this is the case will occupy us through the end of Section 6.

To simplify the exposition, we work under the following assumptions:

1. The linear-exponential program φ in input to OptILEP features the variables $x_1, \ldots, x_n$.
2. The goal is to maximize a single variable x_m, with $m \in [1..n]$. (Section 6 will relax this assumption to handle both maximization and minimization of general linear-exponential terms.)
3. The ordering θ guessed at the very beginning of OptILEP is $\theta \coloneqq (2^{x_n} \geq \cdots \geq 2^{x_1} \geq 2^{x_0} = 1)$.

As stated in Section 2.2, OptILEP iteratively eliminates $x_n, \ldots, x_1$. Throughout the section, we assume that the variables $x_n, \ldots, x_{n-k+1}$ have already been successfully eliminated while preserving optimal solutions, for some $k \in [0..n-1]$. We focus on the elimination of x_{n-k}, which occurs during the $(k+1)$th iteration of the main loop of OptILEP. More precisely, we consider the appeal to ElimVars during this $(k+1)$th iteration. According to Section 2.2, this appeal is preceded by an application of Step I of the procedure from [CMS24] (Lemma 4), and is followed by Step III of the same procedure (Lemma 6). With respect to this $(k+1)$th iteration, we define:

4. The vector $\boldsymbol{x}_k \coloneqq (x_{n-k}, \ldots, x_n)$ containing all variables that have been eliminated, plus x_{n-k}.
5. The ordering $\theta_k \coloneqq (2^{x_{n-k}} \geq \cdots \geq 2^{x_0} = 1)$ obtained from θ by removing the variables that have been eliminated. We also define $\boldsymbol{y}_k \coloneqq (x_0, \ldots, x_{n-k})$ for the variables in this ordering. Note that $\boldsymbol{y}_k$ and $\boldsymbol{x}_k$ only share the variable x_{n-k}.
6. The vectors $\boldsymbol{q}_k \coloneqq (q_{n-k}, \ldots, q_n)$ and $\boldsymbol{r}_k \coloneqq (r_{n-k}, \ldots, r_n)$ of the *quotient variables* and *remainder variables* introduced during the $(k+1)$th iteration the main loop of OptILEP (accordingly to Lemma 4). We assume the variables $q_1, \ldots, q_n, r_1, \ldots, r_n$ to be reused at each iteration; e.g., $\boldsymbol{q}_{k-1} = (q_{n-(k-1)}, \ldots, q_n)$ is the vector of quotient variables used at the kth iteration. Given $\ell \in [0..k]$, we also write $\boldsymbol{q}_{[\ell,k]}$ for the vector $(q_{n-k}, \ldots, q_{n-\ell})$.

4.1 Setup

We now introduce a class of circuits that model the evolution of the objective function during the execution of OptILEP, and characterize the class of objective functions and systems of constraints for which we design our monotone decomposition.

Linear-Exponential arithmetic circuits. In Section 2.1 we gave a bottom-up argument of how the procedure from [CMS24] constructs ILESLPs. An analogous top-down perspective shows the construction of ILESLPs by progressively building *Linear-Exponential Arithmetic Circuits* (*LEACs*):

Definition 2 (LEAC)**.** *Let $k \in [0..n-1]$ and $\ell \in [0..k]$. An (k,ℓ)-linear-exponential arithmetic circuit C —a (k,ℓ)-LEAC, in short— is a sequence of assignments of the form*

$$q_{n-i} \leftarrow \frac{\tau_{n-i}(u, \boldsymbol{q}_{[\ell,k]})}{\eta} \qquad \text{for } i \text{ from } \ell - 1 \text{ to } 0,$$

$$x_{n-i} \leftarrow \frac{\sum_{j=i+1}^{k} a_{i,j} \cdot 2^{x_{n-j}}}{\mu} + q_{n-i} \cdot 2^{x_{n-k-1}} + r_{n-i} \qquad \text{for } i \text{ from } k \text{ to } 0,$$

where each τ_{n-i} is a linear term (with integer coefficients), every $a_{i,j}$ is in $\mathbb{Z}$, and the denominators η and μ are positive integers. We refer to these denominators as μ_C and η_C, respectively. When $k = 0$, the sum $\sum_{j=i+1}^{k} a_{i,j} \cdot 2^{x_{n-j}}$ equals 0, and so we are free to update μ_C to any positive integer; we postulate $\mu_C \coloneqq 1$ in this case. If $\ell = 0$, then η_C is undefined; for technical reasons we postulate $\eta_C \coloneqq \mu_C$ in this case. Moreover, we define $\xi_C \coloneqq \sum\{|a_{i,j}| : i \in [0..k],\, j \in [i+1..k]\}$, and write vars($C$) *for the set of* free variables *of C, i.e., u, x_{n-k-1}, and those in the vectors $\boldsymbol{q}_{[\ell,k]}$ and $\boldsymbol{r}_k$.*

Objective functions. Consider a (k,ℓ)-LEAC C, and a variable x_m with $m \in [1..n]$. We denote by $C[x_m]$ the (objective) function defined as follows: If $n - k > m$, then $C[x_m]$ takes as input maps $\nu\colon X \to \mathbb{N}$ where X is a set of variables featuring x_m, and returns $\nu(x_m)$. Else, $C[x_m]$ takes as input maps $\nu\colon X \to \mathbb{N}$ such that vars(C) $\subseteq X$, and outputs the number $C[x_m](\nu)$ computed as follows:

1: update C: replace each variable $z \in$ vars(C) with $\nu(z)$
2: evaluate C ▷ *each assignment becomes $y \leftarrow a$ where a is a number*
3: **return** the number assigned to x_m in C

The instances. For our purposes, it suffices to define the monotone decomposition for elements of a set $\bigcup_{k=0}^{n-1}\bigcup_{\ell=0}^{k-1}\mathcal{I}_k^\ell$, where $\mathcal{I}_k^\ell$ (defined also for $\ell = k$) is the set of all pairs $(C, \langle\gamma\,;\psi\rangle)$ such that:

(i) $C = (y_1 \leftarrow \rho_1, \ldots, y_t \leftarrow \rho_t)$ is a (k,ℓ)-LEAC such that μ_C divides η_C, as well as all coefficients of the variables $\boldsymbol{q}_{[\ell,k]}$ occurring in the term τ_{n-i} featured in assignments $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_C}$ of C, with $i \in [0..\ell-1]$. (Recall that $\eta_C \coloneqq \mu_C$ for $\ell = 0$.)

(ii) The formula $\langle\gamma\,;\psi\rangle$ is a *conjunction* of a linear program with divisions $\gamma(u, \boldsymbol{q}_{[\ell,k]})$ and a linear-exponential program with divisions ψ. Inequalities and equalities in γ are such that all the coefficients of the variables $\boldsymbol{q}_{[\ell,k]}$ are divisible by μ_C. Moreover, for every q in $\boldsymbol{q}_{[\ell,k]}$, γ contains an inequality $a \cdot q \geq 0$, for some $a \geq 1$ (divisible by μ_C). The system ψ is of the form $\chi(\boldsymbol{y}_{k-1}, \boldsymbol{r}_k) \wedge \theta_k \wedge (x_{n-k} = q_{n-k} \cdot 2^{x_{n-k-1}} + r_{n-k}) \wedge (u = 2^{x_{n-k} - x_{n-k-1}})$.

(We prefer writing $\langle\gamma\,;\psi\rangle$ instead of $\gamma \wedge \psi$ as it emphasize more the distinction between γ and ψ. ElimVars only updates γ, treating ψ as an invariant used to ensure correctness.)

(iii) The formula $\langle\gamma\,;\psi\rangle$ implies the formula $\Psi(C)$ defined as

$$\boldsymbol{0} \leq \boldsymbol{r}_k < 2^{x_{n-k-1}} \wedge \exists \boldsymbol{q}_{[0,\ell-1]}\Big(\boldsymbol{0} \leq \boldsymbol{q}_k \cdot 2^{x_{n-k-1}} + \boldsymbol{r}_k < 2^{x_{n-k}} \wedge \exists \boldsymbol{x}_{k-1}\big(\theta \wedge \textstyle\bigwedge_{i=1}^{t}(y_i = \rho_i)\big)\Big).$$

(For $\ell = 0$, simply conjoin $\boldsymbol{0} \leq \boldsymbol{r}_k < 2^{x_{n-k-1}}$ with the formula in the scope of $\exists \boldsymbol{q}_{[0,\ell-1]}$. Analogously, for $k = 0$, the subformula $\exists \boldsymbol{x}_{k-1}\big(\theta \wedge \bigwedge_{i=1}^{t}(y_i = \rho_i)\big)$ becomes $\theta \wedge \bigwedge_{i=1}^{t}(y_i = \rho_i)$.)

We remark that the variables that are quantified in $\Psi(C)$ are those assigned to some expression in C, excluding x_{n-k}. In essence, elements of $\mathcal{I}_k^\ell$ satisfy certain basic properties that are sufficient to obtain a monotone decomposition. (While our proof relies on all of these properties, it remains unclear whether they are truly necessary for achieving a monotone decomposition.) For example, the subformula $\exists \boldsymbol{x}_{k-1}\big(\theta \wedge \bigwedge_{i=1}^{t}(y_i = \rho_i)\big)$ appearing in $\Psi(C)$ ensures that from any solution of $\langle\gamma;\psi\rangle$, we can assign values to the eliminated variables $x_n, \ldots, x_{n-k+1}$ that preserve the ordering θ, simply by following the assignments defined in the LEAC C. This is consistent with the goal of OptILEP of finding a solution to the input integer linear-exponential program respecting θ.

Algorithm 2 Additional terms for the monotone decomposition.

▷ *see Proposition 3 for the definitions of C, γ and p*

1: $(a, d) \leftarrow$ **guess** an element from $[-L..L]^2$, where $L \coloneqq 3 \cdot \mu_C \cdot (4 \cdot \lceil \log_2(2 \cdot \xi_C + \mu_C) \rceil + 8)$
2: $\lambda \leftarrow \frac{\eta_C}{\mu_C}$
3: $q', q'' \leftarrow$ **guess** two elements in $\boldsymbol{q}_{k-1}$ (they can be equal)
4: $\tau' \leftarrow$ **if** C assigns an expression $\frac{\tau}{\eta_C}$ to q' **then** τ **else** $\eta_C \cdot q'$
5: $\tau'' \leftarrow$ **if** C assigns an expression $\frac{\tau}{\eta_C}$ to q'' **then** τ **else** $\eta_C \cdot q''$
6: **if** $*$ **then** $\tau' \leftarrow \tau'[q_{n-\ell} + p \,/\, q_{n-\ell}]$ ▷ *$*$ stands for non-deterministic choice*
7: **if** $*$ **then** $\tau'' \leftarrow \tau''[q_{n-\ell} + p \,/\, q_{n-\ell}]$
8: $(b \cdot q_{n-\ell} - \rho) \leftarrow$ term obtained from $(a \cdot u + \mu_C \cdot (q' - q'') + d)$ by simultaneously applying the substitutions $[\frac{\tau'}{\lambda} \,/\, \mu_C \cdot q']$ and $[\frac{\tau''}{\lambda} \,/\, \mu_C \cdot q'']$
9: **assert**$(b \neq 0)$ ▷ *else, reject this non-deterministic branch*
10: **return** $(b \cdot q_{n-\ell} - \rho)$

4.2 The monotone decomposition

We are now ready to formalize our monotone decomposition:

Proposition 3. *Consider $(C, \langle \gamma \,;\, \psi \rangle) \in \mathcal{I}_k^\ell$, with $\ell < k$, and $p \coloneqq mod(q_{n-\ell}, \gamma)$. The set of solutions to $\langle \gamma \,;\, \psi \rangle$ has a $(q_{n-\ell}, p)$-monotone decomposition $R_1, \ldots, R_t$ for the function $C[x_m]$. Each R_i is the set of solutions of a linear-exponential program with divisions φ_i satisfying $mod(q_{n-\ell}, \varphi_i) = p$, and in which all constraints featuring $q_{n-\ell}$ are either from $\gamma \wedge \gamma[q_{n-\ell} + p \,/\, q_{n-\ell}]$, or they are inequalities $\tau \leq 0$, where τ is a term (non-deterministically) returned by Algorithm 2.*

The remainder of this section is dedicated to proving Proposition 3. At this stage, we are unable to motivate why the formulae φ_i given in Proposition 3 suffice for obtaining a monotone decomposition; rather, these the formulae that naturally emerge when trying to build such a decomposition.

Preliminary results. Before proceeding with the proof of Proposition 3, we need a few lemmas. The first is a small technical result giving sufficient conditions under which an expression of the form $2^C - 2^{C/2} - d \cdot C$ is non-negative (where $d, C \in \mathbb{R}$). Ultimately, this lemma plays a role in the definition of the quantity L defined in line 1 of Algorithm 2.

Lemma 10. *Let $d, C \in \mathbb{R}$ with $d \geq 1$ and $C \geq 4 \cdot \log_2(d) + 8$. Then, $2^C - 2^{C/2} - d \cdot C \geq 0$.* *Proof in page 94*

The next lemma echoes some ideas firstly used by Semenov for proving the decidability of Presburger arithmetic enriched with the exponential function [Sem84]. In a nutshell, it establishes that, under appropriate hypotheses, any inequality of the form $\sum_{i=1}^{\ell} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d \leq 0$ can be reduced to true (in the lemma, $0 \leq 0$), false ($1 \leq 0$), or a simplified inequality where the sum $\sum_{i=1}^{\ell} a_i \cdot 2^{x_i}$ is replaced with a single exponential term $a \cdot 2^{x_1}$. In our case, this lemma will play a central role in characterizing the regions of our monotone decomposition.

Lemma 11. *Let E be an expression $\sum_{i=1}^{\ell} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d$, where each a_i is in $\mathbb{Z}$, and $\mu, d \in \mathbb{N}$. Let $M, k \in \mathbb{N}$ such that $M \geq \max(1 + 2 \cdot \log_2(\sum_{i=1}^{\ell} |a_i| + k \cdot \mu),\ 4 \cdot \log_2(\mu) + 8,\ d)$. Also consider a formula $\psi(x_1, \ldots, x_\ell, y)$, with y ranging over $\mathbb{Z}$ and $x_1, \ldots, x_\ell$ ranging over $\mathbb{N}$, of the form*

$$\psi(x_1, \ldots, x_\ell, y) \ \coloneqq\ -k \cdot 2^{x_1} \leq y \leq k \cdot 2^{x_1} \wedge \bigwedge\nolimits_{i=1}^{\ell-1} (x_{i+1} \sim_i x_i + d_i),$$

where each pair $(\sim_i, d_i)$ *is either* $(\geq, M)$ *or is of the form* $(=, g)$*, with* $g \in [0..M-1]$*. Let* $\sim \in \{\leq, =\}$*. There is an expression* E' *from the set* $\{0,1\} \cup \{a \cdot 2^{x_1} + \mu \cdot y + \mu \cdot d : a \in [-4^{\ell \cdot M}..4^{\ell \cdot M}]\}$ *such that the formula* ψ *implies* $(E \sim 0 \iff E' \sim 0)$*.*

Proof. First, observe that if $\ell = 0$, then we can take $E' \coloneqq E$. Hence, below let us assume $\ell \geq 1$. Note that ψ implies the ordering $x_\ell \geq x_{\ell-1} \geq \cdots \geq x_1$. By induction on j from 1 to ℓ, we show that for every expression E_j of the form $h_j \cdot 2^{x_j} + \sum_{i=1}^{j-1} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d$, with $|h_j| \leq 4^{(\ell-j+1)M}$, there is an expression E'_j such that ψ implies $(E_j \sim 0 \iff E'_j \sim 0)$, and E'_j belongs to the set $\{0,1\} \cup \{a \cdot 2^{x_1} + \mu \cdot y + \mu \cdot d : a \in [-|h_j| \cdot 4^{(j-1)M}..|h_j| \cdot 4^{(j-1)M}]\}$. The lemma then follows from the fact that E is an expression of the form of E_m, setting $a_m = h_m$; as indeed $|a_m| \leq 4^M$.

base case: $j = 1$**.** For every expression E_1 of the form $h_1 \cdot 2^{x_1} + \mu \cdot y + \mu \cdot d$, we can take $E'_1 \coloneqq E_1$.

induction hypothesis. Given $j > 1$, from every $E_{j-1} = h_{j-1} \cdot 2^{x_{j-1}} + \sum_{i=1}^{j-2} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d$, with $|h_{j-1}| \leq 4^{(\ell-j+2)M}$, there is an expression E'_{j-1} from the set $\{0,1\} \cup \{a \cdot 2^{x_1} + \mu \cdot y + \mu \cdot d : a \in [-|h_{j-1}| \cdot 4^{(j-1)M}..|h_{j-1}| \cdot 4^{(j-1)M}]\}$, such that ψ implies $(E_{j-1} \sim 0 \iff E'_{j-1} \sim 0)$.

induction step: $j > 1$**.** Consider an expression E_j of the form $h_j \cdot 2^{x_j} + \sum_{i=1}^{j-1} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d$, with $|h_j| \leq 4^{(\ell-j+1)M}$. If $h_j = 0$, then we directly obtain E'_j by applying the induction hypothesis on the expression $\sum_{i=1}^{j-1} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d$, since from the assumption on M in the statement of the lemma, we have $|a_{j-1}| \leq 4^M$. Below, let us assume then that $h_j \neq 0$. We distinguish two cases, depending on whether the constraint $x_j \sim_{j-1} x_{j-1} + d_i$ occurring in ψ is of the form $x_j \geq x_{j-1} + M$ or $x_j = x_{j-1} + g$ for some $g \in [0..M-1]$.

case: $x_j \geq x_{j-1} + M$ **occurs in** ψ**.** Intuitively, in this case ψ is constraining x_j to be so large comparatively to x_{j-1} that, in any solution to ψ, $E_j \neq 0$ and the sign of E_j is solely dictated by the sign of h_j. When $\sim$ from $E \sim 0$ is the equality symbol, we can pick $E'_j = 1$, making $E'_j \sim 0$ unsatisfiable. When $\sim$ is instead $\leq$, we set $E'_j = 0$ if h_j is negative, and $E'_j = 1$ otherwise. To show that h_j dictates the sign of E_j, it suffices to establish that ψ implies $2^{x_j} > \left|\sum_{i=1}^{j-1} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d\right|$. First, note that ψ implies $2^{x_j} \geq 2^M \cdot 2^{x_{j-1}}$. As $M \geq d$ and ψ implies both $x_{j-1} \geq \cdots \geq x_1$ and $|y| \leq k \cdot 2^{x_1}$, we have

$$\begin{aligned}\left|\sum_{i=1}^{j-1} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d\right| &\leq \sum_{i=1}^{j-1} |a_i|\, 2^{x_i} + |\mu \cdot y| + \mu \cdot d \\ &\leq \left(\sum_{i=1}^{j-1} |a_i| + k \cdot \mu + M \cdot \mu\right) \cdot 2^{x_{j-1}}.\end{aligned}$$

Therefore, it suffices to show that $2^M > \sum_{i=1}^{j-1} |a_i| + k \cdot \mu + M \cdot \mu$; or equivalently that $2^M - M \cdot \mu > \sum_{i=1}^{j-1} |a_i| + k \cdot \mu$. Since $M \geq 4\log_2(\mu) + 8$, by Lemma 10 we have $2^M - M \cdot \mu \geq 2^{M/2}$. Then, by $M > 2 \cdot \log_2(\sum_{i=1}^{\ell} |a_i| + k \cdot \mu)$,

$$2^M - \mu \cdot M \geq 2^{M/2} > 2^{\lceil \log_2(\sum |a_i| + k \cdot \mu) \rceil} \geq \sum_{i=1}^{j-1} |a_i| + k \cdot \mu.$$

case: $x_j = x_{j-1} + g$ **occurs in** ψ**.** Let E_{j-1} be the expression obtained from E_j by replacing 2^{x_j} by $2^g \cdot 2^{x_{j-1}}$; that is, $E_{j-1} = (h_j \cdot 2^g + a_{j-1}) \cdot 2^{x_{j-1}} + \sum_{i=1}^{j-2} a_i \cdot 2^{x_i} + \mu \cdot y + \mu \cdot d$. We have ψ implies $(E_j \sim 0 \iff E_{j-1} \sim 0)$. To conclude the proof, it suffices to prove that the induction hypothesis can be applied to E_{j-1}. This is the case as soon as

$|h_j \cdot 2^g + a_{j-1}| \leq 4^{(\ell-j+2)M}$ holds, which we show below:

$$\begin{aligned}
|h_j \cdot 2^g + a_{j-1}| &\leq |h_j| \cdot 2^g + |a_{j-1}| \\
&\leq 4^{(\ell-j+1)M} \cdot 2^M + 2^M && \text{\{bounds on } |h_j|, g \text{ and } M\text{\}} \\
&\leq 2^{2(\ell-j+1)M+2M} && \text{\{recall: } M \geq 8\text{\}} \\
&\leq 4^{(\ell-j+2)M}.
\end{aligned}$$

□

The set of possible expressions E' appearing in the conclusion of Lemma 11 can be significantly reduced by noticing that if the absolute value of the integer a exceeds $\mu \cdot (k+d)$, then the truth of $E' \sim 0$ becomes independent of the value given to x_1. This observation allows us to eliminate the polynomial dependence of a on the magnitude of the coefficients $a_1, \ldots, a_\ell$ in the original expression E. Although not obvious, it turns out that this plays a critical point in the proof of Theorem 1. Retaining values of a with polynomial dependence on $a_1, \ldots, a_\ell$ would cause the integers in the LEAC constructed by OptILEP to grow polynomially within each variable elimination step. As a result, their bit sizes would become exponential by the end of the procedure. (In the proof of Proposition 3, we will apply Lemma 11 using a value for the integer d that depends only logarithmically on $a_1, \ldots, a_\ell$; hence, the resulting values of a do in fact retain a logarithmic dependence on these coefficients.) The next lemma gives the refined set of expressions E'.

Lemma 12. *Let the expression E, the non-negative integers μ, d, M and k, the formula ψ, and the symbol $\sim$ be defined as in Lemma 11. Let $b \coloneqq \mu \cdot (k+d)$. There is an expression E' from the set $\{0,1\} \cup \{a \cdot 2^{x_1} + \mu \cdot y + \mu \cdot d : a \in [-b..b]\}$ such that the formula ψ implies $(E \sim 0 \iff E' \sim 0)$.*

Proof. Following Lemma 11, it suffices to take E' to be an expression of the form $a \cdot 2^{x_1} + \mu \cdot y + \mu \cdot d$, where $a \in [-4^{\ell \cdot M}..4^{\ell \cdot M}]$, and show that if a lies outside $[-b..b]$, then E' can be rewritten to 0 or 1.

From its definition, ψ implies $|\mu \cdot y + \mu \cdot d| \leq \mu \cdot (k+d) \cdot 2^{x_1}$. Hence, as soon as $|a| > b$, the truth of $E' \sim 0$ is determined by the sign of a. In particular, whenever $\sim$ is the symbol $=$, or when $a > 0$, the expression E' can be replaced with 1; that is, in this case ψ implies $\neg(E' \sim 0)$. Otherwise, E' can be replaced with 0; and in this case ψ implies $E' \sim 0$. □

Proof of Proposition 3. We are now ready to prove Proposition 3. While long, the proof is divided in several steps and claims. An intuition of the construction is given after defining various objects required for the proof; see the paragraph titled "Construction of $\psi_1, \ldots, \psi_s$: some intuition".

Proof. Throughout the proof, we let $\varphi \coloneqq \langle \gamma \,;\, \psi \rangle$. Let X be the set of all variables appearing in $\boldsymbol{q}_{[\ell,k]}$, $\boldsymbol{r}_k$, $x_0, \ldots, x_{n-k}$ and u. Remark that all variables occurring in φ are among the set X. The value $C[x_m](\nu)$ of objective function $C[x_m]$ is defined for every map $\nu \colon X \to \mathbb{N}$, and corresponds to the value taken by x_m when evaluating C on ν. During the proof, we refer to the variable $q_{n-\ell}$ simply as q. By the definition of $\mathcal{I}_k^\ell$, the variable q appears linearly in γ and does not appear in ψ. Thus, by Lemma 7 the set of solutions to φ is (q,p)-periodic, where $p = \mathit{mod}(q,\gamma) = \mathit{mod}(q,\varphi)$.

Let us begin by considering the (corner) case where x_m satisfies $n-k \geq m$. When $n-k > m$, the variable x_m does not occur in C. Consequently, the function $C[x_m]$ is constant in the variable q (that is, for any solution $\nu \colon X \to \mathbb{N}$ to φ, the function $C[x_m]$ is constant on maps of the form $\nu + [q \mapsto j]$ for $j \in \mathbb{Z}$). Similarly, when $m = n-k$, the circuit C assigns to the variable x_m the expression $q_{n-k} \cdot 2^{x_{n-k-1}} + r_{n-k}$. As all involved variables (q_{n-k}, x_{n-k-1} and r_{n-k}) belong to X and are distinct from q, once again we obtain that the function $C[x_m]$ is constant in the variable q. We conclude that if $m \leq n-k$, then $C[x_m]$ is (q,p)-monotone locally to φ. By Lemma 8, for a monotone decomposition it thus suffices to take a single set R_1 given by the set of solutions to φ.

In the remaining of the proof, we assume that x_m satisfies $m > n-k$. This means that C contains an assignment to x_m, and that x_m is not x_{n-k}. We will construct a sequence of linear-exponential programs $\psi_1, \dots, \psi_s$ satisfying the following conditions:

I. The formula $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\psi_1 \vee \cdots \vee \psi_s$.

II. $C[x_m]$ is (q,p)-monotone locally to (the solutions of) $\varphi \wedge \varphi[q+p\,/\,q] \wedge \psi_i$, for every $i \in [1..s]$.

III. Each ψ_i is of the form $\bigwedge_{j=1}^{n_i} \psi_{i,j}$, where every $\psi_{i,j}$ is an inequality with the following property. Let $\psi_{i,j}$ be $\tau \leq 0$ (and note that then $-\tau-1 \leq 0$ is equivalent to $\neg\psi_{i,j}$). If q occurs in $\psi_{i,j}$, both τ and $-\tau-1$ are terms returned by a non-deterministic branch of the execution of Algorithm 2 with respect to C, γ and p. (This is as required in the statement of the proposition; remark also that in $\varphi \wedge \varphi[q+p\,/\,q]$, all constraints featuring q are from $\gamma \wedge \gamma[q+p\,/\,q]$.)

Then, the desired (q,p)-monotone decomposition $R_1, \dots, R_t$ is given by the formulae in the set

$$\Big\{\varphi \wedge \varphi[q+p\,/\,q] \wedge \psi_i : i \in [1..s]\Big\} \cup \Big\{\varphi \wedge \bigwedge\nolimits_{i=1}^{s} \neg\psi_{i,f(i)} : f \in \mathcal{G}\Big\},$$

where $\mathcal{G}$ is the set of all function $f\colon [1..s] \to \mathbb{N}$ such that $f(i) \in [1..n_i]$ for every $i \in [1..s]$. Indeed, the function $C[x_m]$ is (q,p)-monotone locally to the formula $\varphi \wedge \neg\varphi[q+p\,/\,q]$, and so also locally to any formula $\varphi \wedge \bigwedge_{i=1}^{s} \neg\psi_{i,f(i)}$; as the latter implies the former by Item I. Additionally, the sets of solutions of all the formulae $\varphi \wedge \varphi[q+p\,/\,q] \wedge \psi_i$ and $\varphi \wedge \bigwedge_{i=1}^{s} \neg\psi_{i,f(i)}$ are (q,p)-periodic, since q occurs linearly in these formulae, and none of the ψ_i contains any divisibility constraints. (A side remark: the number of formulae $\psi_1, \dots, \psi_s$ will be exponential in the size of φ, making the number of constraints in each formula $\bigwedge_{i=1}^{s} \neg\psi_{i,f(i)}$ also exponential. As explained in Remark 2, this is unproblematic: to eliminate q, it suffices to guess a *single* constraint from any of these formulae.)

Construction of $\psi_1, \dots, \psi_s$: a preliminary step. We begin by introducing an ℓ-LEAC C^{+p} that will be associated to the formula $\varphi[q+p\,/\,q]$. We define C^{+p} as the ℓ-LEAC obtained from C by replacing q with $q+p$, and renaming to $\overline{v}$ every variable v among $q_{n-\ell+1}, \dots, q_n, x_{n-k+1}, \dots, x_n$. These are the variables to which C assigns an expression, and whose value may depend on the value given to q. To clarify, if C is defined as

$$q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta} \qquad \text{for } i \text{ from } \ell-1 \text{ to } 0,$$

$$x_{n-i} \leftarrow \frac{\sum_{j=i+1}^{k} a_{i,j} \cdot 2^{x_{n-j}}}{\mu} + q_{n-i} \cdot 2^{x_{n-k-1}} + r_{n-i} \qquad \text{for } i \text{ from } k \text{ to } 0,$$

then C^{+p} is defined as

$$\overline{q}_{n-i} \leftarrow \frac{\tau_{n-i}[q+p\,/\,q]}{\eta} \qquad \text{for } i \text{ from } \ell-1 \text{ to } 0,$$

$$x_{n-k} \leftarrow q_{n-k} \cdot 2^{x_{n-k-1}} + r_{n-k},$$

$$\overline{x}_{n-i} \leftarrow \frac{a_{i,k} \cdot 2^{x_{n-k}} + \sum_{j=i+1}^{k-1} a_{i,j} \cdot 2^{\overline{x}_{n-j}}}{\mu} + q_{n-i} \cdot 2^{x_{n-k-1}} + r_{n-i} \qquad \text{for } i \text{ from } k-1 \text{ to } \ell+1$$

$$\overline{x}_{n-\ell} \leftarrow \frac{a_{\ell,k} \cdot 2^{x_{n-k}} + \sum_{j=i+1}^{k-1} a_{\ell,j} \cdot 2^{\overline{x}_{n-j}}}{\mu} + (q_{n-\ell}+p) \cdot 2^{x_{n-k-1}} + r_{n-i}$$

$$\overline{x}_{n-i} \leftarrow \frac{a_{i,k} \cdot 2^{x_{n-k}} + \sum_{j=i+1}^{k} a_{i,j} \cdot 2^{\overline{x}_{n-j}}}{\mu} + \overline{q}_{n-i} \cdot 2^{x_{n-k-1}} + r_{n-i} \qquad \text{for } i \text{ from } \ell-1 \text{ to } 0.$$

To simplify the presentation, we introduce the symbolic aliases:

$$(z_1,\ldots,z_{2k+1}) \coloneqq (x_n, x_{n-1},\ldots,x_{n-k+1},\overline{x}_n,\overline{x}_{n-1},\ldots,\overline{x}_{n-k+1},x_{n-k}),$$
$$(y_1,\ldots,y_{2k+1}) \coloneqq (q_n, q_{n-1},\ldots,q_{n-k+1},\overline{q}_n,\overline{q}_{n-1},\ldots,\overline{q}_{n-\ell+1},(q_{n-\ell}+p),q_{n-\ell-1}\ldots,q_{n-k}),$$
$$(s_1,\ldots,s_{2k+1}) \coloneqq (r_n, r_{n-1},\ldots,r_{n-k+1},r_n,r_{n-1},\ldots,r_{n-k+1},r_{n-k}).$$

For every $j \in [1..k]$ (resp. $j \in [k+1..2k]$), the variables y_j and s_j represent the quotient and remainder variables occurring in the expression assigned to z_j in C (resp. C^{+p}). Note that both C and C^{+p} include the assignment $x_{n-k} \leftarrow q_{n-k}\cdot 2^{x_{n-k-1}} + r_{n-k}$, which, using of our aliases, is expressed as $z_{2k+1} \leftarrow y_{2k+1}\cdot 2^{x_{n-k-1}} + s_{2k+1}$. Additionally, we introduce symbolic aliases for all variables to which C or C^{+p} assign an expression:

$$(w_1,\ldots,w_{2(k+\ell)+1}) \coloneqq (\underbrace{z_1,\ldots,z_k,y_1,\ldots,y_\ell}_{\text{assigned in } C},\underbrace{z_{k+1},\ldots,z_{2k},y_{k+1},\ldots,y_{k+\ell}}_{\text{assigned in } C^{+p}},\underbrace{z_{2k+1}}_{\text{a.k.a. } x_{n-k}}).$$

For $j \in [1..2(k+\ell)+1]$, let ρ_j denote the expression assigned to the variable w_j in either C or C^{+p}. Since x_{n-k} is the only variable shared between the two circuits, the definition of ρ_j is unambiguous. In particular, $\rho_{2(k+\ell)+1}$ corresponds to the expression $q_{n-k}\cdot 2^{x_{n-k-1}} + r_{n-k}$.

Recall that φ implies the formula $\Psi(C)$ defined as

$$\Psi(C) \coloneqq \mathbf{0} \le \boldsymbol{r}_k < 2^{x_{n-k-1}} \wedge \exists \boldsymbol{q}_{[0,\ell-1]} : \Big(\mathbf{0} \le \boldsymbol{q}_k\cdot 2^{x_{n-k-1}} + \boldsymbol{r}_k < 2^{x_{n-k}} \wedge \exists x_{n-k+1}\ldots\exists x_n\ \big(\theta \wedge \textstyle\bigwedge_{i=1}^{k+\ell}(w_i=\rho_i)\wedge(w_{2(k+\ell)+1}=\rho_{2(k+\ell)+1})\big)\Big).$$

Similarly, it is simple to see that $\varphi[q+p\,/\,q]$ implies the formula $\Psi(C^{+p})$ defined as

$$\Psi(C^{+p}) \coloneqq \mathbf{0} \le \boldsymbol{r}_k < 2^{x_{n-k-1}} \wedge \exists \overline{\boldsymbol{q}}_{[0,\ell-1]} : \Big(\textstyle\bigwedge_{j=0}^{\ell-1}\big(0 \le \overline{q}_{n-j}\cdot 2^{x_{n-k-1}} + r_{n-j} < 2^{x_{n-k}}\big) \wedge \big(0 \le q\cdot 2^{x_{n-k-1}} + p\cdot 2^{x_{n-k-1}} + r_{n-\ell} < 2^{n-k}\big) \wedge \textstyle\bigwedge_{j=\ell+1}^{k}\big(0 \le q_{n-j}\cdot 2^{x_{n-k-1}} + r_{n-j} < 2^{x_{n-k}}\big) \wedge \exists\,\overline{x}_{n-k+1}\ldots\exists\,\overline{x}_n\ \big(\overline{\theta}\wedge\textstyle\bigwedge_{i=k+\ell+1}^{2(k+\ell)+1}(w_i=\rho_i)\big)\Big),$$

where $\overline{\theta} \coloneqq 2^{\overline{x}_n} \ge \cdots \ge 2^{\overline{x}_{n-k+1}} \ge 2^{x_{n-k}} \ge \cdots \ge 2^{x_0} = 1$. To show that $\varphi[q+p\,/\,q]$ implies $\Psi(C^{+p})$, observe that $(\varphi[q+p\,/\,q] \implies \Psi(C^{+p}))$ is syntactically equal to $(\varphi \implies \Psi(C))[q+p\,/\,q]$, except for the names used for the existentially quantified variable. Specifically, every such variable v in $\Psi(C)$ is replaced with $\overline{v}$ in $\Psi(C^{+p})$. Since $(\varphi \implies \Psi(C))$ is a valid formula, and validity is preserved under substitution and variable renaming, it follows that $(\varphi[q+p\,/\,q] \implies \Psi(C^{+p}))$ is also valid.

The following claim establishing further properties of $C^{+p}[\overline{x}_m]$ follows directly from the fact that C^{+p} is obtained from C by replacing with $q+p$ all occurrences of the variable q.

Claim 1. *Let ν and $\nu + [q \mapsto p]$ be two solutions to φ. Then, $C[x_m](\nu + [q\mapsto p]) = C^{+p}[\overline{x}_m](\nu)$.*

Construction of $\psi_1,\ldots,\psi_s$: mixing xs and $\overline{x}$s into a single ordering. Before giving some more intuition on the construction of $\psi_1,\ldots,\psi_s$, we need some additional formulae manipulations. Let $\boldsymbol{w}$ denote the vector of all variables that appear quantified in the formulae $\Psi(C)$ or $\Psi(C^{+p})$. Specifically, these are the variables $w_1,\ldots,w_{2(k+\ell)}$ (note that $w_{2(k+\ell)+1} = x_{n-k}$ is a free variable

in both formulae). Observe that, by the definition of $\Psi(C)$ and $\Psi(C^{+p})$, the linear-exponential program $\varphi \wedge \varphi[q+p\,/\,q]$ implies

$$\exists \boldsymbol{w} : (x_n \geq x_{n-1} \geq \cdots \geq x_{n-k}) \wedge (\overline{x}_n \geq \overline{x}_{n-1} \geq \cdots \geq \overline{x}_{n-k+1} \geq x_{n-k}) \wedge \bigwedge\nolimits_{i=1}^{2(k+\ell)+1} (w_i = \rho_i). \quad (6)$$

Let us denote with $\mathcal{P}$ the set of all permutations $\sigma\colon [1..2k+1] \to [1..2k+1]$ (on the indices of the variables $z_1, \ldots, z_{2k+1}$) that satisfy:

- $\sigma^{-1}(1) \leq \sigma^{-1}(2) \leq \cdots \leq \sigma^{-1}(k)$, i.e., σ respects the ordering $x_n \geq \cdots \geq x_{n-k+1}$.
- $\sigma^{-1}(k+1) \leq \sigma^{-1}(k+2) \leq \cdots \leq \sigma^{-1}(2k)$, i.e., σ respects the ordering $\overline{x}_n \geq \cdots \geq \overline{x}_{n-k+1}$.
- $\sigma(2k+1) = 2k+1$, i.e., the variable x_{n-k} is smaller or equal than any other variable.

The formula in Equation (6) is equivalent to $\bigvee_{\sigma \in \mathcal{P}} \chi_\sigma$, where χ_σ is defined as

$$\chi_\sigma \coloneqq \exists \boldsymbol{w} : (z_{\sigma(1)} \geq z_{\sigma(2)} \geq \cdots \geq z_{\sigma(2k)} \geq z_{\sigma(2k+1)}) \wedge \bigwedge\nolimits_{i=1}^{2(k+\ell)+1} (w_i = \rho_i). \quad (7)$$

Construction of $\psi_1, \ldots, \psi_s$: some intuition. We are now ready to provide the promised intuition behind the construction of $\psi_1, \ldots, \psi_s$. Since χ_σ includes the equations $\bigwedge_{i=1}^{2(k+\ell)+1}(w_i = \rho_i)$, which describe the assignments in the circuits C and C^{+p}, for every solution to $\varphi \wedge \varphi[q+p\,/\,p]$ there is one and only one assignment to the variables in $\boldsymbol{w}$ that yields a solution to the quantifier-free part of the formula in Equation (7). This assignment is determined by the values computed by the circuits C and C^{+p}. The order that σ induces on the variables $z_1, \ldots, z_{2k+1}$ implies either $x_m \geq \overline{x}_m$ or $\overline{x}_m \geq x_m$. Therefore, by relying on Claim 1, one concludes that the function $C[x_m]$ is (q,p)-monotone locally to $\varphi \wedge \varphi[q+p\,/\,q] \wedge \chi_\sigma$:

Claim 2. *For every $\sigma \in \mathcal{P}$, the function $C[x_m]$ is (q,p)-monotone locally to $\varphi \wedge \varphi[q+p\,/\,q] \wedge \chi_\sigma$.* *Proof in page 94*

Clearly, we cannot use the formulae χ_σ as the desired formulae $\psi_1, \ldots, \psi_s$, as these formulae are quantified over variables not occurring in φ (we will thus go against our goal of achieving variable elimination). Instead, we will further refine and manipulate the ordering in Equation (7) and, by appealing to Lemma 11, restate it solely in terms of variables that occur (free) in φ. This will allow us to push the ordering outside the scope of the quantifiers $\exists \boldsymbol{w}$, leading to formulae of the form $\psi \wedge \exists \boldsymbol{w} \bigwedge_{i=1}^{2(k+\ell)+1}(w_i = \rho_i)$. Next, we will show that the function $C[x_m]$ is (q,p)-monotone locally to $\varphi \wedge \varphi[q+p\,/\,q] \wedge \psi \wedge \exists \boldsymbol{w} \bigwedge_{i=1}^{2(k+\ell)+1}(w_i = \rho_i)$. Since $\exists \boldsymbol{w} \bigwedge_{i=1}^{2(k+\ell)+1}(w_i = \rho_i)$ is implied by $\varphi \wedge \varphi[q+p\,/\,q]$, this ensures that $C[x_m]$ is (q,p)-monotone locally to $\varphi \wedge \varphi[q+p\,/\,q] \wedge \psi$, as required by Item II. Moreover, because of how these formulae are constructed, the disjunction over all such formulae ψ will still be implied by $\varphi \wedge \varphi[q+p\,/\,q]$, fulfilling Item I. Finally, we will manipulate ψ to meet the structural requirements of Item III.

Construction of $\psi_1, \ldots, \psi_s$. We start by further refining the orderings induced by the permutations in $\mathcal{P}$ by quantifying the gaps between variables. Let $\mathcal{D}$ be the set of all functions $d\colon [1..2k] \to [0..M]$, where $M \coloneqq 4 \cdot \lceil \log_2(2 \cdot \xi_C + \mu_C) \rceil + 8$. For $g \in [0..M-1]$, we write $\sim_g$ as an alias for $=$, and $\sim_M$ as an alias for $\geq$. For $\sigma \in \mathcal{P}$, the formula χ_σ in Equation (7) is equivalent to the formula $\bigvee_{d \in \mathcal{D}} \chi_{\sigma,d}$ where

$$\chi_{\sigma,d} \coloneqq \exists \boldsymbol{w} : \bigwedge\nolimits_{j=1}^{2k} \left(z_{\sigma(j)} \sim_{d(j)} z_{\sigma(j+1)} + d(j) \right) \wedge \bigwedge\nolimits_{i=1}^{2(k+\ell)+1} (w_i = \rho_i). \quad (8)$$

Essentially, every constraint $z_{\sigma(j)} \geq z_{\sigma(j+1)}$ from χ_σ is refined in $\chi_{\sigma,d}$ to either $z_{\sigma(j)} \geq z_{\sigma(j+1)}+M$ or $z_{\sigma(j)} = z_{\sigma(j+1)}+g$, for some $g \in [0..M-1]$. Note that each $\chi_{\sigma,d}$ implies χ_σ. Therefore, by Claim 2, the function $C[x_m]$ is (q,p)-monotone locally to $\varphi \wedge \varphi[q+p\,/\,q] \wedge \chi_{\sigma,d}$. Furthermore, the formula $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\bigvee_{\sigma\in\mathcal{P}} \bigvee_{d\in\mathcal{D}} \chi_{\sigma,d}$.

Let $\sigma \in \mathcal{P}$ and $d \in \mathcal{D}$. The next step of the proof involves manipulating the constraints $z_{\sigma(j)} \sim_{d(j)} z_{\sigma(j+1)}+d(j)$ from the formula $\chi_{\sigma,d}$. This manipulation produces a formula $\chi'_{\sigma,d}$ in which these constraints only feature variables from φ. Moreover, $\varphi \wedge \varphi[q+p\,/\,q]$ implies $(\chi_{\sigma,d} \iff \chi'_{\sigma,d})$. The value of M introduced when defining $\chi_{\sigma,d}$ was chosen to make this manipulation possible by appealing to Lemma 11. The core step in this manipulation is given in the next claim.

Claim 3. *Let $\sigma \in \mathcal{P}$ and $d \in \mathcal{D}$. The formula $\varphi \wedge \varphi[q+p\,/\,q]$ implies*

$$\forall \boldsymbol{w}\Big(\bigwedge\nolimits_{i=1}^{2(k+\ell)+1}(w_i=\rho_i) \implies \Big(\bigwedge\nolimits_{j=1}^{2k}\big(z_{\sigma(j)} \sim_{d(j)} z_{\sigma(j+1)}+d(j)\big) \iff \bigwedge\nolimits_{j=1}^{2k}(0 \sim_{d(j)} E_j)\Big)\Big),$$

where each E_j (with $j \in [1..2k]$) is 0, 1, or an expression of the form

$$a \cdot 2^{x_{n-k}} + \mu_C \cdot \big((y_{\sigma(j+1)} - y_{\sigma(j)}) \cdot 2^{x_{n-k-1}} + (s_{\sigma(j+1)} - s_{\sigma(j)})\big) + \mu_C \cdot d(j), \tag{9}$$

for some $a \in [-b..b]$, where $b \coloneqq \mu_C \cdot (1+M)$.

Proof of Claim 3. The proof is by induction on t from $2k+1$ to 1, with induction hypothesis stating that $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\forall \boldsymbol{w}\big(\bigwedge_{i=1}^{2(k+\ell)+1}(w_i=\rho_i) \implies \Gamma^{(t)}_{\sigma,d}\big)$, where $\Gamma^{(t)}_{\sigma,d}$ is defined as

$$\bigwedge\nolimits_{j=t}^{2k}\big(z_{\sigma(j)} \sim_{d(j)} z_{\sigma(j+1)}+d(j)\big) \iff \bigwedge\nolimits_{j=t}^{2k}(0 \sim_{d(j)} E_j), \tag{10}$$

for some suitable expressions E_j having the form described in the statement of the claim.

base case: $t = 2k+1$**.** This case is trivial, as $\Gamma^{(2k+1)}_{\sigma,d}$ is defined as the tautology ($\top \iff \top$).

induction hypothesis. For $t \in [1..2k]$, $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\forall \boldsymbol{w}(\bigwedge_{i=1}^{2(k+\ell)+1}(w_i=\rho_i) \implies \Gamma^{(t+1)}_{\sigma,d})$.

induction step: $t < 2k+1$**.** From the induction hypothesis, $\varphi \wedge \varphi[q+p\,/\,q]$ implies

$$\forall \boldsymbol{w}\Big(\bigwedge\nolimits_{i=1}^{2(k+\ell)+1}(w_i=\rho_i) \implies \Big(\bigwedge\nolimits_{j=t}^{2k}\big(z_{\sigma(j)} \sim_{d(j)} z_{\sigma(j+1)}+d(j)\big) \iff \bigwedge\nolimits_{j=t+1}^{2k}\big(0 \sim_{d(j)} E_j\big) \wedge \boxed{z_{\sigma(t)} \sim_{d(t)} z_{\sigma(t+1)}+d(t)}\Big)\Big), \tag{11}$$

for some suitable expressions E_j adhering to the form described in the statement of the claim.

We construct E_t by modifying the formula in Equation (11), specifically by only updating the *boxed* occurrence of $(z_{\sigma(t)} \sim_{d(t)} z_{\sigma(t+1)}+d(t))$ that appears on the right-hand side of the double implication. After all manipulations, the resulting formula is still implied by $\varphi \wedge \varphi[q+p\,/\,q]$.

We rewrite the *boxed* constraint $(z_{\sigma(t)} \sim_{d(t)} z_{\sigma(t+1)}+d(t))$ by replacing the variables $z_{\sigma(t)}$ and $z_{\sigma(t+1)}$ with the corresponding expressions featured in the antecedent $\bigwedge_{i=1}^{2(k+\ell)+1}(w_i=\rho_i)$ of the implication in Equation (11). Specifically, if w_{i_1} is an alias of $z_{\sigma(t)}$ and w_{i_2} is an alias of $z_{\sigma(t+1)}$, then we replace $z_{\sigma(t)}$ by ρ_{i_1} and $z_{\sigma(t+1)}$ by ρ_{i_2}. By the definition of C and C^{+p}, the result is a constraint $(0 \sim_{d(t)} E)$ where E is of the form

$$\sum_{i=t+1}^{2k+1} a_i \cdot 2^{z_{\sigma(i)}} + \mu_C \cdot \big((y_{\sigma(t+1)} - y_{\sigma(t)}) \cdot 2^{x_{n-k-1}} + (s_{\sigma(t+1)} - s_{\sigma(t)})\big) + \mu_C \cdot d(t), \tag{12}$$

where every $\sum_{i=t+1}^{2k+1} |a_i|$ is bounded, in absolute value, by $2 \cdot \xi_C$.

After the updates, the formula in Equation (11) is still implied by $\varphi \wedge \varphi[q + p \,/\, q]$. Because of the induction hypothesis, to show this it suffices to see that

- $w_{i_1} = \rho_{i_1} \wedge w_{i_2} = \rho_{i_2} \wedge (z_{\sigma(t)} \sim_{d(t)} z_{\sigma(t+1)} + d(t))$ implies $(0 \sim_{d(t)} E)$, and
- $w_{i_1} = \rho_{i_1} \wedge w_{i_2} = \rho_{i_2} \wedge \neg(z_{\sigma(t)} \sim_{d(t)} z_{\sigma(t+1)} + d(t))$ implies $\neg(0 \sim_{d(t)} E)$.

Both these implications follows trivially from how E is constructed.

We further manipulate the expression E from Equation (12). The following three facts hold:

- The right-hand side of the double implication of the updated formula from Equation (11), now featuring $(0 \sim_{d(t)} E)$, implies (because of the double implication) the constraints $\big(z_{\sigma(i)} \sim_{d(i)} z_{\sigma(i+1)} + d(i)\big)$ for every $i \in [t+1..2k]$.
- Since $\varphi \wedge \varphi[q + p \,/\, q]$ implies the formula in Equation (11), we can bound the term $\big((y_{\sigma(t+1)} - y_{\sigma(t)}) \cdot 2^{x_{n-k-1}} + (s_{\sigma(t+1)} - s_{\sigma(t)})\big)$ occurring in E as follows:

$$-2^{x_{n-k}} < \big((y_{\sigma(t+1)} - y_{\sigma(t)}) \cdot 2^{x_{n-k-1}} + (s_{\sigma(t+1)} - s_{\sigma(t)})\big) < 2^{x_{n-k}}. \tag{13}$$

  Indeed, from the definition of $\Psi(C)$ and $\Psi(C^{+p})$, for both $j \in \{\sigma(t), \sigma(t+1)\}$, we have:
  - If C and C^{+p} do not assign an expression to y, then the formula $\varphi \wedge \varphi[q + p \,/\, q]$ implies $0 \leq y_j \cdot 2^{x_{n-k-1}} + s_j < 2^{x_{n-k}}$.
  - If C or C^{+p} assign an expression ρ to y_j, then the formula $\varphi \wedge \varphi[q + p \,/\, q]$ implies $\exists y_j \, (0 \leq y_j \cdot 2^{x_{n-k-1}} + s_j < 2^{x_{n-k}} \wedge y_j = \rho)$. In this case, observe that since y_j is (an alias of) a quotient variable among $q_n, \ldots, q_{n-\ell+1}, \overline{q}_n, \ldots, \overline{q}_{n-\ell+1}$, the expression ρ only contains variables occurring in φ (and it does not contain y_j). Consequently, the value of y_j is uniquely determined given a solution to $\varphi \wedge \varphi[q + p \,/\, q]$. This means that $\varphi \wedge \varphi[q + p \,/\, q]$ also implies $\forall y_j \, (y_j = \rho \implies 0 \leq y_j \cdot 2^{x_{n-k-1}} + s_j < 2^{x_{n-k}})$.

  We conclude that $\varphi \wedge \varphi[q + p \,/\, q]$ implies

$$\forall \boldsymbol{w} \Big(\bigwedge\nolimits_{i=1}^{2(k+\ell)+1} (w_i = \rho_i) \implies \bigwedge\nolimits_{j \in \{\sigma(t), \sigma(t+1)\}} 0 \leq y_j \cdot 2^{x_{n-k-1}} + s_j < 2^{x_{n-k}} \Big),$$

  which allows us to bound $(y_{\sigma(t+1)} - y_{\sigma(t)}) \cdot 2^{x_{n-k-1}} + (s_{\sigma(t+1)} - s_{\sigma(t)})$ as in Equation (13).
- The integer M used to define the map d satisfies

$$M \geq \max\big(1 + 2 \cdot \log_2\big(\sum\nolimits_{i=t+1}^{2k+1} |a_i| + \mu_C\big),\ 4 \cdot \log_2(\mu_C) + 8,\ d(t)\big).$$

Due to the three items above, we can invoke Lemma 11 and Lemma 12 to rewrite E as an expression that is either 0 or 1, or has the form

$$a \cdot 2^{x_{n-k}} + \mu_C \cdot \big((y_{\sigma(t+1)} - y_{\sigma(t)}) \cdot 2^{x_{n-k-1}} + (s_{\sigma(t+1)} - s_{\sigma(t)})\big) + \mu_C \cdot d(t),$$

for some $a \in [-b..b]$, where $b \coloneqq \mu_C \cdot (1 + M)$. This completes the proof of the claim. □

Resuming the proof of Proposition 3, from the chain of equivalences involving Equations (6) to (8) and by applying Claim 3, we see that $\varphi \wedge \varphi[q + p \,/\, q]$ implies $\bigvee_{\sigma \in \mathcal{P}} \bigvee_{d \in \mathcal{D}} \chi'_{\sigma,d}$ where

$$\chi'_{\sigma,d} \coloneqq \exists \boldsymbol{w} : \bigwedge\nolimits_{j=1}^{2k} (0 \sim_{d(j)} E_{\sigma,d,j}) \wedge \bigwedge\nolimits_{i=1}^{2(k+\ell)+1} (w_i = \rho_i), \tag{14}$$

and each $E_{\sigma,d,j}$ is an expression having one of the forms given in Claim 3. (Below, we simply write E_j instead of $E_{\sigma,d,j}$, as σ and d will be clear from the context.) Moreover, as a consequence of Claim 2, the function $C[x_m]$ is (q,p)-monotone locally to $\varphi \wedge \varphi[q+p\,/\,q] \wedge \chi'_{\sigma,d}$.

Given $\sigma \in \mathcal{P}$ and $d \in \mathcal{D}$, we further manipulate the formula $\chi'_{\sigma,d}$. We consider every expression E_j of the form $a \cdot 2^{x_{n-k}} + \mu_C \cdot \big((y_{\sigma(j+1)} - y_{\sigma(j)}) \cdot 2^{x_{n-k-1}} + (s_{\sigma(j+1)} - s_{\sigma(j)})\big) + \mu_C \cdot d(j)$. Let us write $L_{\sigma,d,j}$ and $R_{\sigma,d,j}$ (again, we omit σ and d for simplicity) for the two expressions:

$$L_j \coloneqq a \cdot u + \mu_C \cdot (y_{\sigma(j+1)} - y_{\sigma(j)}), \qquad R_j \coloneqq \mu_C \cdot (s_{\sigma(j+1)} - s_{\sigma(j)}) + \mu_C \cdot d(j).$$

Recall that φ occurs in $\mathcal{I}_k$, and therefore it features the equality $u = 2^{x_{n-k}-x_{n-k-1}}$; so $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\forall \boldsymbol{w} : E_j = L_j \cdot 2^{x_{n-k-1}} + R_j$. Moreover, from the definition of $\Psi(C)$, we see that φ implies $(0 \leq s_{\sigma(j)} < 2^{x_{n-k-1}})$ and $(0 \leq s_{\sigma(j+1)} < 2^{x_{n-k-1}})$. We then conclude that $\varphi \wedge \varphi[q+p\,/\,q]$ implies $-\mu_C \cdot 2^{x_{n-k-1}} < R_j < (\mu_C + 1) \cdot M \cdot 2^{x_{n-k-1}}$. Given $r \in [-\mu_C..(\mu_C+1)\cdot M]$, we write $\gamma_{\sigma,d,j,r}$ (often omitting σ and d, as they will be clear from the context) for the formula given by:

- if $\sim_{d(j)}$ stands for $=$, then $\gamma_{j,r} \coloneqq \big(L_j + r = 0 \wedge R_j = r \cdot 2^{x_{n-k-1}}\big)$, and,
- if $\sim_{d(j)}$ stands for $\geq$, then $\gamma_{j,r} \coloneqq \big(L_j + r \leq 0 \wedge (r-1) \cdot 2^y < R_j \leq r \cdot 2^y\big)$.

From Lemma 3, $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\forall \boldsymbol{w}\big((0 \sim_{d(j)} E_j) \iff \bigvee_{r=-\mu_C}^{(\mu_C+1)\cdot M} \gamma_{j,r}\big)$.

We further update the expressions $L_j + r$ above by updating $y_{\sigma(j)}$ and $y_{\sigma(j+1)}$ using the corresponding expressions featured in the formula $\bigwedge_{i=1}^{2(k+\ell)+1}(w_i = \rho_i)$, if such expressions exist. That is, if a variable w_i in this formula is an alias for $y_{\sigma(j)}$, then we consider the substitution $[\frac{\rho_i}{\lambda}\,/\,\mu_C \cdot y_{\sigma(j)}]$, where $\lambda \coloneqq \frac{\eta_C}{\mu_C}$. Since $(C,\varphi) \in \mathcal{I}_k^\ell$, we know that μ_C divides η_C, hence λ is a positive integer. Observe moreover that in L_j, the variables $y_{\sigma(j)}$ and $y_{\sigma(j+1)}$ have a coefficient of $\pm\mu_C$; this means that applying the substitution $[\frac{\rho_i}{\lambda}\,/\,\mu_C \cdot y_{\sigma(j)}]$ eliminates $y_{\sigma(j)}$. If no variable w_i is an alias for $y_{\sigma(j)}$, we consider instead the substitution $[\frac{\eta_C \cdot y_{\sigma(j)}}{\lambda}\,/\,\mu_C \cdot y_{\sigma(j)}]$ (applying this substitution to $L_j + r$ simply scales all coefficients by λ). Observe that these substitutions are among those constructed in lines 4 and 6 of Algorithm 4. We handle $y_{\sigma(j+1)}$ analogously, and simultaneously apply the resulting substitutions to $L_j + r$. Let $L'_{j,r}$ be the resulting expression. It is of the form

$$\lambda \cdot a \cdot u + \tau(u, \boldsymbol{q}_{[\ell,k]}) + \lambda \cdot r \tag{15}$$

where $\tau(u, \boldsymbol{q}_{[\ell,k]})$ is a difference $\tau' - \tau''$ of two terms τ' and τ'' having forms among the following:

- a variable $\eta_C \cdot q_{n-i}$ with $i \in [\ell..k]$; note that these variables occurs free in $\Psi(C)$ and $\Psi(C^{+p})$,
- the expression $\eta_C \cdot (q+p)$; note that $(q+p)$ is aliased by one of the variables $y_1, \ldots, y_{2k+1}$, and that no expression in C and C^{+p} is assigned to q.
- the term τ_{n-i}, for some $i \in [0..\ell-1]$, which occurs in C in the expression assigned to q_{n-i},
- the term $\tau_{n-i}[q+p\,/\,q]$, for some $i \in [0..\ell-1]$, which occurs in C^{+p} in the expression for $\overline{q}_{n-i}$.

Lastly, let $\gamma'_{j,r}$ be the formula obtained from $\gamma_{j,r}$ by replacing $L_j + r$ with $L'_{j,r}$, and rewriting $L'_{j,r} = 0$ as $L'_{j,r} \leq 0 \wedge -L'_{j,r} \leq 0$. Note that no variable from $\gamma'_{j,r}$ occurs in the vector $\boldsymbol{w}$. We are now in the position of establishing the following two results, which almost complete the proof of Proposition 3.

Claim 4. *The formula $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\bigvee_{\sigma\in\mathcal{P}} \bigvee_{d\in\mathcal{D}} \bigwedge_{j=1}^{2k} \bigvee_{r=-\mu_C}^{(\mu_C+1)\cdot M} \gamma'_{\sigma,d,j,r}$.*

Claim 5. *For every $\sigma \in \mathcal{P}$ and every $d \in \mathcal{D}$, the function $C[x_m]$ is (q,p)-monotone locally to the set of solutions of the formula $\varphi \wedge \varphi[q+p\,/\,q] \wedge \bigwedge_{j=1}^{2k} \bigvee_{r=-\mu_C}^{(\mu_C+1)\cdot M} \gamma'_{\sigma,d,j,r}$.*

Proof of Claims 4–5. We have already established that:

1. The formula $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\bigvee_{\sigma\in\mathcal{P}}\bigvee_{d\in\mathcal{D}}\chi'_{\sigma,d}$, where $\chi'_{\sigma,d}$ is as in Equation (14).
2. For every $\sigma \in \mathcal{P}$ and $d \in \mathcal{D}$, $\varphi \wedge \varphi[q+p\,/\,q]$ implies $\forall \boldsymbol{w}\big((0 \sim_{d(j)} E_j) \iff \bigvee_{r=-\mu_C}^{(\mu_C+1)\cdot M} \gamma_{j,r}\big)$.
3. The function $C[x_m]$ is (q,p)-monotone locally to $\varphi \wedge \varphi[q+p\,/\,q] \wedge \chi'_{\sigma,d}$.

Every formula $\gamma'_{j,r}$ is obtained from $\gamma_{j,r}$ by "applying" the equalities from $\bigwedge_{i=1}^{2(k+\ell)+1}(w_i = \rho_i)$ as substitutions. Together with Item 2 above, we thus conclude that $\chi'_{\sigma,d}$ is equivalent to

$$\exists \boldsymbol{w} : \Big(\Big(\bigwedge\nolimits_{j=1}^{2k}\bigvee\nolimits_{r=-\mu_C}^{(\mu_C+1)\cdot M}\gamma'_{j,r}\Big) \wedge \bigwedge\nolimits_{i=1}^{2(k+\ell)+1}(w_i = \rho_i)\Big).$$

The variables $\boldsymbol{w}$ do not occur in any formula $\gamma'_{j,r}$, and so the above formula is equivalent to

$$\Big(\bigwedge\nolimits_{j=1}^{2k}\bigvee\nolimits_{r=-\mu_C}^{(\mu_C+1)\cdot M}\gamma'_{j,r}\Big) \wedge \exists \boldsymbol{w} : \bigwedge\nolimits_{i=1}^{2(k+\ell)+1}(w_i = \rho_i).$$

The two claims then follows from Items 1 and 3, together with the fact that the formula $\varphi\wedge\varphi[q+p\,/\,q]$ implies $\Psi(C) \wedge \Psi(C^{+p})$, which in turn implies $\exists \boldsymbol{w} \bigwedge_{i=1}^{2(k+\ell)+1}(w_i = \rho_i)$ by definition. □

At last, let us define the formulae $\psi_1, \ldots, \psi_s$. They correspond to all the linear-exponential systems occurring as disjuncts of the disjunctive normal form of the formulae $\bigwedge_{j=1}^{2k}\bigvee_{r=-\mu_C}^{(\mu_C+1)\cdot M}\gamma'_{\sigma,d,j,r}$, for every $\sigma \in \mathcal{P}$ and $d \in \mathcal{D}$. Directly from Claim 4 and Claim 5, we conclude that these formulae satisfy the desired conditions in Items I and II. To complete the proof, it suffices to show that the condition in Item III is also satisfied.

Every constraint occurring in the formulae $\psi_1, \ldots, \psi_s$ that features the variable q is of the form $\pm L'_{\sigma,d,j,r} \leq 0$, where $L'_{\sigma,d,j,r}$ is an expression as in Equation (15). Therefore, the term $\pm L'_{\sigma,d,j,r}$ is of the form $\lambda \cdot a' \cdot u + \tau'(u, \boldsymbol{q}_{[\ell,k]}) + \lambda \cdot r'$ where

- $a' \in \mathbb{Z}$ belongs to the set $[-\mu_C\cdot(1+M)..\mu_C\cdot(1+M)]$. (Recall: $M \coloneqq 4\cdot\lceil\log_2(2\cdot\xi_C+\mu_C)\rceil+8$.)
- $\tau'(u, \boldsymbol{q}_{[\ell,k]})$ is the term obtained from $(y_{\sigma(j+1)} - y_{\sigma(j)})$ or $(y_{\sigma(j)} - y_{\sigma(j+1)})$ by applying suitable substitutions. These are among the substitutions considered in lines 4–7 of Algorithm 2.
- $r' \in \mathbb{Z}$ belongs to $[-(\mu_C+1)\cdot M..(\mu_C+1)\cdot M]$.

Note that $\neg(\pm L'_{\sigma,d,j,r} \leq 0)$ is equivalent to $\mp L'_{\sigma,d,j,r}+1 \leq 0$. Then, in order to cover $\neg(\pm L'_{\sigma,d,j,r} \leq 0)$ with terms $\lambda \cdot a' \cdot u + \tau'(u, \boldsymbol{q}_{[\ell,k]}) + \lambda \cdot r'$ as above, it suffices to increase the interval for the integers r' to $[-(\mu_C+1)\cdot M - 1..(\mu_C+1)\cdot M + 1]$. It is then easy to see that Item III holds. In fact, for simplicity of the presentation, Algorithm 2 uses slightly larger ranges for a' and r' (these integers are called a and d in the pseudocode, respectively, see line 1). Indeed, since $\mu_C \geq 1$ and $M \geq 8$, both $\mu_C \cdot (1+M)$ and $(\mu_C+1)\cdot M + 1$ are bounded by $3 \cdot \mu_C \cdot M$. □

5 An efficient variable elimination that preserves optimal solutions

Building on our monotone decomposition, this section instantiates Algorithm 1 (ElimVars) into the optima-preserving variable elimination procedure that was promised in Section 2.2. We also provide the proofs of correctness and complexity of this algorithm.

The pseudocode of the instantiation of ElimVars is given on page 35. The instantiation is obtained by *(i)* defining the inputs of ElimVars, *(ii)* providing an algorithm for computing the test points and *(iii)* implementing the elimination discipline. Following the arguments in Section 4, achieving the first two points is simple. In particular, appealing to the notation from Section 4:

- The *inputs* of ElimVars are triples $(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$, for every $k \in [0..n-1]$, where $(C, \langle \gamma \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^0$. (The procedure will thus eliminate the variables $q_{n-k+1}, \ldots, q_n$, but not the variable q_{n-k}. This is consistent with the sketch of the procedure from [CMS24] given in Section 2, where the latter variable is eliminated only after Step III.)

- The pseudocode of the procedure for computing the test points is given in Algorithm 4. Briefly, the procedure first (non-deterministically) computes a term $(a \cdot q - \tau)$ stemming from the monotone decomposition of Proposition 3, and then returns an equality $(a \cdot q = \tau - s)$, where s is a non-negative shift that suffices to explore optimal solutions, by Lemma 9.

The implementation of an efficient elimination discipline is a more complex task. As noted in the introduction (Example 2), the naïve approach of rewriting a formula γ as $\gamma[\frac{\tau - s}{a} \,/\, q] \wedge (|a| \mid \tau - s)$, where $(a \cdot q = \tau - s)$ is a test point, would lead to exponential growth in the bit length of integer coefficients, during the execution of ElimVars. In [CMS24], this problem is avoided by extending Bareiss' algorithm for Gaussian elimination [Bar68] to integer linear programs. While our elimination discipline is also based on Bareiss' algorithm, it is different from the one in [CMS24]. In particular, we do not introduce "slack variables", and add some technical machinery to handle the additional test points required by the monotone decomposition (those computed by Algorithm 2).

5.1 Efficient elimination discipline: the high-level idea

The pseudocode of our elimination discipline is given in Algorithm 5. We now discuss the overall idea behind this procedure. Consider one of its inputs: a pair $(C[x_m], \langle \gamma ; \psi \rangle)$, where $(C, \langle \gamma ; \psi \rangle) \in \mathcal{I}_k^\ell$ with $\ell < k$, and an equality $a \cdot q_{n-\ell} = \tau$ returned by Algorithm 4 on input $(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$.

Let us briefly explain why the naïve approach of eliminating the variable $q_{n-\ell}$ from γ by performing the substitution $[\frac{\tau}{a} \,/\, q_{n-\ell}]$ leads to an exponential growth. Assuming $a > 0$, this substitution rewrites an inequality $b \cdot q_{n-\ell} \leq \tau'$ as $b \cdot \tau \leq a \cdot \tau'$. Let c and d denote the coefficients of another variable, say y, in τ and τ', respectively. Then, in the term $b \cdot \tau - a \cdot \tau'$, the coefficient of y is $b \cdot c - a \cdot d$. In essence, this shows that the coefficients of the variables in γ may grow quadratically within each elimination of one of the variables in $\boldsymbol{q}_{k-1}$. As a result, by the end of ElimVars, the naïve approach may produce a linear program with coefficients of exponential bit size.

The above explosion can be avoided by observing that the variable coefficient $b \cdot c - a \cdot d$ is exactly the one we would obtain when performing a naïve version of the Gaussian elimination procedure for putting a matrix with entries over $\mathbb{Z}$ in echelon form. Building on Bareiss's observation [Bar68], we see that these growing variable coefficients accumulate common factors as we iteratively eliminate variables. Divisions by these common factors after each variable elimination keep variable coefficients of polynomial bit size. (These divisions are performed in lines 5–6 of Algorithm 5.) While variable coefficients evolve as in Gaussian elimination, the *constants* of the terms do not. In particular, the shift performed in line 3 of Algorithm 4 disrupt any structure in the constants. This however does not pose a problem. Inequalities of the form $\lambda \cdot \rho + c \leq 0$ (where $\lambda \geq 1$ is the common factor, ρ a term, and c is the integer constant) can be rewritten as $\rho + \lceil \frac{c}{\lambda} \rceil \leq 0$. For equalities $\lambda \cdot \rho + c = 0$ instead, observe that they are unsatisfiable when c is not a multiple of λ, and otherwise they can be rewritten as $\rho + \frac{c}{\lambda} = 0$. Algorithm 5 implements this reasoning in lines 4–6, where the positive integer λ defined in line 2 represents the common factor. (We will clarify why this common factor is exactly the ratio $\frac{\eta_C}{\mu_C}$ of the two denominators of the (k, ℓ)-LEAC C in Section 5.4.) A similar argument can be made for updates performed to the circuit C, by seeing each assignment $q \leftarrow \frac{\tau}{\eta_C}$ as the equality $\eta_C \cdot q = \tau$; see line 7 of Algorithm 5.

Further technical details must be added to the above picture to keep the complexity of OptILEP in check. The main problem arises when Algorithm 4 produces a test point by appealing to Algo-

Algorithm 3 Instantiation of ElimVars (Algorithm 1).

Input: $(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$: a triple such that $(C, \langle \gamma \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^0$, for some $k \in [0..n-1]$.

1: **while** some variable from $\boldsymbol{q}_{k-1}$ appears in γ or in vars(C) **do**
2: $\quad (a \cdot q = \tau) \leftarrow$ **guess** an element in $TP(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$ ▷ *Algorithm 4*
3: $\quad (C[x_m], \langle \gamma \,;\, \psi \rangle) \leftarrow Elim(C[x_m], \langle \gamma \,;\, \psi \rangle, a \cdot q = \tau)$ ▷ *Algorithm 5*
4: **return** (f, φ)

Algorithm 4 *TP*: Non-deterministic generation of the test points for ILEP.

Input: $(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$: a triple such that $(C, \langle \gamma \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^\ell$, with $\ell < k$.

1: $p \leftarrow mod(q_{n-\ell}, \gamma)$
2: $(a \cdot q_{n-\ell} - \tau) \leftarrow$ **guess** a term with $a \neq 0$ that is either from terms$(\gamma \wedge \gamma[q_{n-\ell} + p \,/\, q_{n-\ell}])$, or computed using Algorithm 2 with respect to (C, γ, p).
3: $s \leftarrow$ **guess** an element in $[0..\,|a| \cdot p - 1]$
4: **return** $(a \cdot q_{n-\ell} = \tau - s)$

Algorithm 5 *Elim*: An efficient elimination discipline for ILEP.

Input: $(C[x_m], \langle \gamma \,;\, \psi \rangle)$: a pair such that $(C, \langle \gamma \,;\, \psi \rangle) \in \mathcal{I}_k^\ell$, with $\ell < k$;
$a \cdot q_{n-\ell} = \tau$: an equality returned by Algorithm 4 on input $(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$.

1: **if** $a < 0$ **then** $(a, \tau) \leftarrow (-a, -\tau)$ ▷ *consider* $-a \cdot q_{n-\ell} = -\tau$ *instead*
2: $\lambda \leftarrow \frac{\eta_C}{\mu_C}$; $\quad \alpha \leftarrow \frac{a}{\mu_C}$
3: $\gamma \leftarrow \gamma[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}] \wedge (a \mid \tau)$
4: **assert**(in every equality $\tau = 0$ of γ, the constant of the term τ is divisible by λ)
▷ *here and below,* ***assert****(false) causes the non-deterministic branch to reject*
5: update each equality $\tau = 0$ in γ :
• divide all integers appearing in the term τ by λ
6: update each inequality $\tau \leq 0$ in γ :
• divide all variable coefficients in the term τ by λ
• replace the constant c of the term τ with $\lceil \frac{c}{\lambda} \rceil$
7: update C :
• for every $i \in [0..\ell - 1]$, consider the assignment $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_C}$ in C
– **assert**(the constant of the term $\tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ is divisible by λ)
– replace $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_C}$ with $q_{n-i} \leftarrow \frac{\tau'_{n-i}}{a}$, where the term τ'_{n-i} is obtained from the term $\tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ by dividing all integers by λ
• prepend the assignment $q_{n-\ell} \leftarrow \frac{\tau}{a}$
8: **return** $(C[x_m], \langle \gamma \,;\, \psi \rangle)$ ▷ $(C, \langle \gamma \,;\, \psi \rangle)$ *belongs to* $\mathcal{I}_k^{\ell+1}$

rithm 2. Such test points depend on the assignments featured in the circuit C. This dependence makes it challenging to maintain the delicate "Gaussian-elimination-style evolution" that variable coefficients must have throughout ElimVars. The solution Algorithm 5 implements starts from the observation that all non-zero coefficients of the quotient variables $\boldsymbol{q}_k$ in the term in line 8 of Algorithm 2 are (before substitutions) only $\pm\mu_C$. Together with the constraints imposed by $\mathcal{I}_k^\ell$, which ensure that all coefficients of the variables $\boldsymbol{q}_{[\ell,k]}$ are divisible by μ_C, this observation will enable us to give a variation of Bareiss algorithm, from which we can prove that ElimVars, and later OptILEP, run in non-deterministic polynomial time.

Before moving to a more technical analysis of ElimVars, let us note that the overwhelming presence of μ_C in both Algorithm 2 and elements of $\mathcal{I}_k^\ell$ implies the following property of Algorithm 4:

Lemma 13. *Given in input a triple $(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$ with $(C, \langle \gamma \,;\, \psi \rangle) \in \mathcal{I}_k^\ell$ Algorithm 4 guesses in line 2 a linear term $a \cdot q_{n-\ell} - \tau(u, \boldsymbol{q}_{[\ell+1,k]})$ in which all coefficients of $\boldsymbol{q}_{[\ell,k]}$ are divisible by μ_C.* *Proof in page 98*

5.2 Correctness of ElimVars

The integration of the machinery from Bareiss algorithm to keep the growth of the coefficients in check has a "presentational drawback": the arguments for establishing the complexity of the algorithm now mix with those needed to prove its correctness, as we must ensure that this machinery is implemented correctly.

To ease the presentation, we structure this and the next three (Sections 5.3 to 5.5) as follows. In the current section, we isolate the key property necessary for the correct implementation of Bareiss's machinery. This property is formalized in Claim 6. Assuming this claim to hold, we then establish the correctness of ElimVars. Sections 5.3 and 5.4 develop the arguments needed to prove Claim 6, while also setting up the properties required for the complexity analysis. More precisely, Section 5.3 presents a variation of Bareiss algorithm in which the evolution of variable coefficients precisely mirrors that of ElimVars. We state a series of results characterizing this evolution; their proofs are deferred to Appendix C.3 —these proofs involve a detour to linear algebra and Bareiss algorithm, and we prefer to keep the focus of this section on ElimVars. In Section 5.4 we formalize the connection between this variation of Bareiss algorithm and ElimVars, and use it to prove Claim 6. Finally, in Section 5.5 we leverage this connection to analyze the complexity of ElimVars.

Here is the aforementioned key property related to Bareiss algorithm:

Claim 6. *The following property is true across all the executions of Algorithm 5 performed in all non-deterministic branches of* ElimVars*, on any of its inputs. In all equalities and inequalities of the formula $\gamma[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ computed in line 3, and in terms $\tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ computed in line 7, all coefficients of the variables $\boldsymbol{q}_{[\ell+1,k]}$ are divisible by η_C, and all coefficients of u are divisible by $\frac{\eta_C}{\mu_C}$.*

The following lemma establishes the correctness of ElimVars.

Lemma 14. *There is a non-deterministic procedure with the following specification:*

Input: $\boldsymbol{q}_{k-1}$ *: the vector of quotient variables $q_{n-(k-1)}, \ldots, q_n$;* *(for any k)*
$C[x_m]$ *: objective function, where C is a $(k,0)$-LEAC;*
$\langle \gamma \,;\, \psi \rangle$ *: linear exponential program with divisions, such that the pair $(C, \langle \gamma \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^0$.*

Output of each branch (β): $C'_\beta[x_m]$ *: objective function, where C'_β is a (k,k)-LEAC;*
$\langle \gamma'_\beta \,;\, \psi \rangle$ *: linear exponential program with divisions, such that $(C'_\beta, \langle \gamma'_\beta \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^k$.*

The procedure ensures the satisfaction of the following two properties:

- ***Equivalence:*** *The formulae* $\exists \boldsymbol{q}_{k-1}\, \gamma$ *and* $\bigvee_\beta \gamma'_\beta$ *are equivalent. Consider a branch* β*, and let* $q_{n-(k-1)} \leftarrow \frac{\tau_{n-(k-1)}}{\eta}, \ldots, q_n \leftarrow \frac{\tau_n}{\eta}$ *be the assignments to the variables* $\boldsymbol{q}_{k-1}$ *occurring in* C'_β*. Given a solution* $\nu\colon \{u, q_{n-k}\} \to \mathbb{N}$ *to* γ'_β*, the map* $\nu + \sum_{i=0}^{k-1} [q_{n-i} \mapsto \frac{\nu(\tau_{n-i})}{\eta}]$ *is a solution to* γ*.*
- ***Preservation of maximum:*** *if* $\max\{C[x_m](\nu) : \nu$ *is a solution to* $\langle \gamma \,;\, \psi \rangle\}$ *exists, then it is equal to* $\max\{C'_\beta[x_m](\nu) : \beta$ *is a branch,* ν *is a solution to* $\langle \gamma'_\beta \,;\, \psi \rangle\}$*.*

Proof (assuming Claim 6). By induction on $\ell = k, \ldots, 0$, with induction hypothesis:

induction hypothesis. The specification given in Lemma 14 holds when executing the **while** loop of Algorithm 1 (ElimVars) on a triple $(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$ where $(C, \langle \gamma \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^\ell$. That is to say,

- **Output:** the output of each branch β is $(C'_\beta[x_m], \langle \gamma'_\beta \,;\, \psi \rangle)$, with $(C'_\beta, \langle \gamma'_\beta \,;\, \psi \rangle) \in \mathcal{I}_k^k$.
- **Equivalence:** The formulae $\exists \boldsymbol{q}_{[\ell,k-1]} \gamma$ and $\bigvee_\beta \gamma'_\beta$ are equivalent. Consider a branch β, and let $q_{n-(k-1)} \leftarrow \frac{\tau_{n-(k-1)}}{\eta}, \ldots, q_{n-\ell} \leftarrow \frac{\tau_{n-\ell}}{\eta}$ be the assignments to the variables $\boldsymbol{q}_{[\ell,k-1]}$ in C'_β. Given a solution $\nu\colon \{u, q_{n-k}\} \to \mathbb{N}$ to γ'_β, the map $\nu + \sum_{i=\ell}^{k-1} [q_{n-i} \mapsto \frac{\nu(\tau_{n-i})}{\eta}]$ is a solution to γ.
- **Preservation of maximum:** if $\max\{C[x_m](\nu) : \nu$ is a solution to $\langle \gamma \,;\, \psi \rangle\}$ exists, then it is equal to $\max\{C'_\beta[x_m](\nu) : \beta$ is a branch, ν is a solution to $\langle \gamma'_\beta \,;\, \psi \rangle\}$.

Observe that setting $\ell = 0$ in the above induction hypothesis yields the statement of Lemma 14.

base case: $\ell = k$**.** In this case, we have $(C, \langle \gamma \,;\, \psi \rangle) \in \mathcal{I}_k^k$. By definition of $\mathcal{I}_k^k$ (page 23) this means that C is a (k,k)-LEAC, and γ only features the variables q_{n-k} and u. Therefore, no variable in $\boldsymbol{q}_{k-1}$ occurs in these objects, and the condition of the **while** loop of ElimVars fails. The algorithm then returns $(C[x_m], \langle \gamma ; \psi \rangle)$, and all requirements stated in the induction hypothesis are trivially satisfied.

induction step: $\ell \in [0..k-1]$**.** Let $\varphi \coloneqq \langle \gamma \,;\, \psi \rangle$. By definition of $\mathcal{I}_k^\ell$, the linear program γ features an inequality $b \cdot q_{n-\ell} \geq 0$, for some $b \geq 1$. Let $p \coloneqq \mathit{mod}(q_{n-\ell}, \gamma)$. Since $q_{n-\ell}$ belongs to $\boldsymbol{q}_{k-1}$, the body of the **while** loop of ElimVars executes. The call to Algorithm 4 (nondeterministically) returns an equality $a \cdot q_{n-\ell} = \tau - s$ such that *(i)* $a \neq 0$, *(ii)* $(a \cdot q_{n-\ell} - \tau)$ is either from terms$(\gamma \wedge \gamma[q_{n-\ell} + p \,/\, q_{n-\ell}])$ or it is computed using Algorithm 2 with respect to (C, γ, p), and *(iii)* $s \in [0..\,|a| \cdot p - 1]$. Again by definition of $\mathcal{I}_k^\ell$, the variables occurring in τ are from the vector $\boldsymbol{q}_{[\ell+1,k]} = (q_{n-k}, \ldots, q_{n-(\ell+1)})$.

From Corollary 2 (which concerns satisfiability) and Proposition 3 and Lemma 9 (which concern optimization) we conclude that if φ has a solution (analogously, if $(C[x_m], \varphi)$ has a maximum), then it has one satisfying an equality $a \cdot q_{n-\ell} = \tau - s$ returned by Algorithm 4. The lemma is therefore implied by the induction hypothesis (applied to elements in $\mathcal{I}_k^{\ell+1}$) together with the following claim (below, for brevity we write ρ instead of $\tau - s$):

Claim 7. *Given in input* $(C[x_m], \varphi)$ *and an equality* $a \cdot q_{n-\ell} = \rho$ *computed by Algorithm 4, Algorithm 5 behaves as follows:*

1. *If the statements in the* ***assert*** *commands in lines 4 and 7 are not satisfied, then the algorithm rejects. In this case the formula* $\exists q_{n-\ell} : \varphi \wedge (a \cdot q_{n-\ell} = \rho)$ *is unsatisfiable.*

2. *Else, the algorithm returns* $(C'[x_m], \langle \gamma' ; \psi \rangle)$ *such that* $(C', \langle \gamma' ; \psi \rangle) \in \mathcal{I}_k^{\ell+1}$. *In this case:*
   (a) *The formula* $\langle \gamma' ; \psi \rangle$ *is equivalent to* $\exists q_{n-\ell} : \varphi \wedge (a \cdot q_{n-\ell} = \rho)$.
   (b) *Consider a solution* $\nu \colon X \setminus \{q_{n-\ell}\} \to \mathbb{N}$ *to* $\langle \gamma' ; \psi \rangle$, *and let* $\nu' \coloneqq \nu + [q_{n-\ell} \mapsto \frac{\nu(\rho)}{a}]$. *For each* $i \in [0..\ell-1]$, *given the assignments* $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_C}$ *and* $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_{C'}}$ *from* C *and* C', *respectively, we have* $\frac{\nu'(\tau_{n-i})}{\eta_C} = \frac{\nu(\tau'_{n-i})}{\eta_{C'}}$. *Moreover,* $C'[x_m](\nu) = C[x_m](\nu')$.

Let us prove Claim 7. Observe that line 1 rewrites the equality $a \cdot q_{n-\ell} = \rho$ and $-a \cdot q_{n-\ell} = -\rho$ whenever $a < 0$, hence forcing the coefficient of $q_{n-\ell}$ to be positive. Hence, in what follows we assume without loss of generality that $a > 0$ (hence $\alpha = \frac{a}{\mu_C}$ in line 2 is positive).

Let us consider Item 1 of the claim. Assume that one of the **assert** commands in lines 4 and 7 is not satisfied. In the case of line 4, this means that an equation $\tau' = 0$ from $\gamma[\frac{\rho}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ is such that the constant c of τ' is not divisible by λ. Since $\lambda = \frac{\eta_C}{\mu_C}$, assuming Claim 6, $\tau' = 0$ can be written as $\lambda \cdot \tau'' + c = 0$. But then, whenever the variables in τ'' are evaluated to some integers, this equation is asserting that a multiple of λ is equal to $-c$; contradicting the fact that c is not divisible by λ. This implies that $\gamma[\frac{\rho}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ is unsatisfiable, and thus so is $\exists q_{n-\ell} : \varphi \wedge (a \cdot q_{n-\ell} = \rho)$. The argument is similar when the **assert** command of line 7 is not satisfied. Indeed, consider $i \in [0..\ell-1]$ such that $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_C}$ occurs in C, and the constant of the term $\tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ is not divisible by λ. From the definition of $\mathcal{I}_k^\ell$ (more precisely, the definition of $\Psi(C)$), φ implies $\exists q_{n-i} : \eta_C \cdot q_{n-i} = \tau_{n-i}$, which in turn means that $\exists q_{n-\ell} : \varphi \wedge (a \cdot q_{n-\ell} = \rho)$ implies $\exists q_{n-i} : \alpha \cdot \eta_C \cdot q_{n-i} = \tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$. By definition, λ is a divisor of η_C. Hence, assuming Claim 6, in the equality $\alpha \cdot \eta_C \cdot q_{n-i} = \tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ all variable coefficients are divisible by $\lambda = \frac{\eta_C}{\mu_C}$, but the constant term is not. As in the previous case, this means that $\alpha \cdot \eta_C \cdot q_{n-i} = \tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ is unsatisfiable, and thus so is $\exists q_{n-\ell} : \varphi \wedge (a \cdot q_{n-\ell} = \rho)$. This completes the proof of the first of the two items in Claim 7.

We move to Item 2. Assume that the **assert** commands in lines 4 and 7 are satisfied, and therefore that the algorithm returns some pair $(C'[x_m], \langle \gamma' ; \psi \rangle)$. We first show that $\langle \gamma' ; \psi \rangle$ is equivalent to $\exists q_{n-\ell} : \varphi \wedge (a \cdot q_{n-\ell} = \rho)$. From $(C, \langle \gamma ; \psi \rangle) \in \mathcal{I}_k^\ell$, in all (in)equality from γ the coefficients of $q_{n-\ell}$ are divisible by μ_C. This means that the substitution $[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ performed in line 3 eliminates $q_{n-\ell}$. Therefore, the formula $\gamma[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}] \wedge (a \mid \tau)$ is a linear program with divisions over the variables $\boldsymbol{q}_{[\ell+1,k]}$ and u. Clearly, this formula is equivalent to $\exists q_{n-\ell} : \gamma \wedge (a \cdot q_{n-\ell} = \rho)$, proving Item 2a. Since ψ does not contain the variable $q_{n-\ell}$, it thus suffices to show that transformation in lines 4–6, which produces γ' from $\gamma[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}] \wedge (a \mid \tau)$, preserves formula equivalence. These lines rely on the following two equivalences, that hold for any linear term τ' (since such terms evaluate to integers):

$$\lambda \cdot \tau' = 0 \iff \tau' = 0 \qquad \text{(since } \lambda \neq 0\text{)}$$

$$\lambda \cdot \tau' + c \leq 0 \iff \tau' + \left\lceil \frac{c}{\lambda} \right\rceil \leq 0 \qquad \text{(since } \lambda \geq 1\text{)}$$

To ensure that lines 4–6 correctly implement these equivalences, we need to verify that all divisions performed in these lines are without remainder. Assuming Claim 6, the coefficients of the variables $\boldsymbol{q}_{[\ell+1,k]}$ in (in)equalities of $\gamma[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ are divisible by η_C, while the coefficients of u are divisible by $\lambda = \frac{\eta_C}{\mu_C}$. Since the **assert** command of line 4 is satisfied, the constants occurring in equalities are also divisible by λ. It follows that the divisions performed in lines 5 and 6 are without remainder. (Note that this also shows that γ' is a linear program with divisions in variables $\boldsymbol{q}_{[\ell+1,k]}$ and u.)

Next, we show that the returned pair $(C'[x_m], \langle \gamma' ; \psi \rangle)$ is such that $(C', \langle \gamma' ; \psi \rangle) \in \mathcal{I}_k^{\ell+1}$. Recall that, by Lemma 13, the equality $a \cdot q_{n-\ell} = \rho$ computed by Algorithm 4 is such that $a \cdot q_{n-\ell} - \rho$ is a linear term featuring variables u and $\boldsymbol{q}_{[\ell,k]}$, in which the coefficients of the variables $\boldsymbol{q}_{[\ell,k]}$ divisible by μ_C. Below, Items (i) to (iii) refers to the items characerizing $\mathcal{I}_k^{\ell}$ (page 23).

- *Item (i):* we must prove that C' is a $(k, \ell+1)$-LEAC such that $\mu_{C'}$ divides $\eta_{C'}$, as well as all coefficients of the variables $\boldsymbol{q}_{[\ell+1,k]}$ occurring in the term τ'_{n-i} featured in assignments $q_{n-i} \leftarrow \frac{\tau'_{n-i}}{\eta_{C'}}$ of C', with $i \in [0..\ell]$.

  The circuit C' is constructed in line 7. This line updates all assignments $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_C}$ in C, where $i \in [0..\ell-1]$, and prepends the assignment $q_{n-\ell} \leftarrow \frac{\tau}{a}$. The denominator of all these assignments is $\eta_{C'} = a$, which is divisible by $\mu_{C'} = \mu_C$. Recall that, from $(C, \langle \gamma ; \psi \rangle) \in \mathcal{I}_k^{\ell}$, the term τ_{n-i} is a linear term in variables $\boldsymbol{q}_{[\ell,k]}$ and u, in which the coefficient of the variable $q_{n-\ell}$ is divisible by μ_C. From Lemma 13, we conclude that $\tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ is a linear term in variables $\boldsymbol{q}_{[\ell+1,k]}$ and u. Therefore, C' is a $(k, \ell+1)$-LEAC. Lastly, assuming Claim 6, the coefficients of the variables $\boldsymbol{q}_{[\ell+1,k]}$ in the term $\tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ are divisible by η_C, and the coefficient of u is divisible by $\lambda = \frac{\eta_C}{\mu_C}$. Since the **assert** command of line 7 is satisfied, the constant of this term is also divisible by λ. We conclude that the divisions by λ performed to compute τ'_{n-i} are without remainder, and that in this term the coefficients of the variables in $\boldsymbol{q}_{[\ell+1,k]}$ are divisible by $\mu_{C'}$.
- *Item (ii):* We have already shown that γ' is a linear program with divisions in variables $\boldsymbol{q}_{[\ell+1,k]}$ and u. It thus suffices to show that γ' satisfies the following properties: *(i)* every coefficient of the variables in $\boldsymbol{q}_{[\ell+1,k]}$ is divisible by $\mu_{C'}$, and *(ii)* for each q in $\boldsymbol{q}_{[\ell+1,k]}$, γ' contains an inequality of the form $a \cdot q \geq 0$ for some $a \geq 1$.

  For the first property, let us go back to the fact that the coefficients of the variables $\boldsymbol{q}_{[\ell+1,k]}$ in (in)equalities of $\gamma[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ are divisible by η_C (Claim 6). Following the (remainder-less) divisions by $\lambda = \frac{\eta_C}{\mu_C}$ performed in lines 5 and 6, we conclude that in all (in)equalities of γ' the coefficients of the variables in $\boldsymbol{q}_{[\ell+1,k]}$ are divisible by $\mu_{C'} = \mu_C$.

  For the second property, recall that for every q in $\boldsymbol{q}_{[\ell+1,k]}$, γ features an inequality of the form $b \cdot q \geq 0$ for some $b \geq 1$. In γ' this inequality is transformed into $\frac{\alpha \cdot b}{\lambda} \cdot q \geq 0$, where $\frac{\alpha \cdot b}{\lambda}$ is positive and, assuming Claim 6, an integer.
- *Item (iii):* For brevity, let $C = (y_1 \leftarrow \rho_1, \ldots, y_t \leftarrow \rho_t)$ and $C' = (y_0 \leftarrow \rho'_0, \ldots, y_t \leftarrow \rho'_t)$, where $y_0 \leftarrow \rho'_0$ is an alias for the assignment $q_{n-\ell} \leftarrow \frac{\rho}{a}$ that the algorithm prepend to C in line 7, in order to define C'. We must show that $\langle \gamma' ; \psi \rangle$ implies $\Psi(C')$, that is,

$$\mathbf{0} \leq \boldsymbol{r}_k < 2^{x_{n-k-1}} \wedge \exists \boldsymbol{q}_{[0,\ell]} \Big(\mathbf{0} \leq \boldsymbol{q}_k \cdot 2^{x_{n-k-1}} + \boldsymbol{r}_k < 2^{x_{n-k}} \wedge \exists \boldsymbol{x}_{k-1} \big(\theta \wedge \textstyle\bigwedge_{i=0}^{t} (y'_i = \rho'_i) \big) \Big).$$

  We have already established that $\langle \gamma' ; \psi \rangle$ is equivalent to $\exists q_{n-\ell} : \varphi \wedge (a \cdot q_{n-\ell} = \rho)$. Since φ implies $\Psi(C)$, we then conclude that $\langle \gamma' ; \psi \rangle$ implies

$$\mathbf{0} \leq \boldsymbol{r}_k < 2^{x_{n-k-1}} \wedge \exists \boldsymbol{q}_{[0,\ell]} \Big(\mathbf{0} \leq \boldsymbol{q}_k \cdot 2^{x_{n-k-1}} + \boldsymbol{r}_k < 2^{x_{n-k}} \\ \wedge \exists \boldsymbol{x}_{k-1} \big(\theta \wedge \textstyle\bigwedge_{i=1}^{t} (y_i = \rho_i) \wedge (a \cdot q_{n-\ell} = \rho) \big) \Big).$$

  Hence, as line 7 does not modify the assignments in C that feature variables in $\boldsymbol{x}_{k-1}$, to conclude that $\langle \gamma' ; \psi \rangle$ implies $\Psi(C')$ it suffices to show that, for every $i \in [0..\ell-1]$,

$$a \cdot q_{n-\ell} = \rho \implies \big(\eta_C \cdot q_{n-i} = \tau_{n-i} \iff a \cdot q_{n-i} = \tau'_{n-i} \big), \tag{16}$$

where τ_{n-i} is the term such that $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_C}$ occurs in C, and τ'_{n-i} is the term such that $q_{n-i} \leftarrow \frac{\tau'_{n-i}}{a}$ occurs in C'. Recall that $\alpha = \frac{a}{\mu_C} \geq 1$, $\lambda = \frac{\eta_C}{\mu_C} \geq 1$, and that all divisions performed in line 7 are without remainder. Then, Equation (16) follows from the equivalences below:

$$\begin{aligned}
&(\eta_C \cdot q_{n-i} - \tau_{n-i})[\tfrac{\rho}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}] \\
&= \alpha \cdot \eta_C \cdot q_{n-i} - (\tau_{n-i}[\tfrac{\rho}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]) && \text{⟅by def. of substitution⟆} \\
&= \frac{\alpha \cdot \eta_C}{\lambda} \cdot q_{n-i} - \tau'_{n-i} && \text{⟅division by } \lambda\text{⟆} \\
&= a \cdot q_{n-i} - \tau'_{n-i} && \text{⟅from } \alpha = \tfrac{a}{\mu_C},\ \lambda = \tfrac{\eta_C}{\mu_C}\text{⟆}
\end{aligned}$$

This concludes the proof that $(C', \langle \gamma' \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^{\ell+1}$. To conclude the proof Claim 7, it remains to show Item 2b. Consider a solution $\nu \colon X \setminus \{q_{n-\ell}\} \to \mathbb{N}$ to $\langle \gamma' \,;\, \psi \rangle$, and let $\nu' \coloneqq \nu + [q_{n-\ell} \mapsto \frac{\nu(\rho)}{a}]$. Since $\langle \gamma' \,;\, \psi \rangle$ is equivalent to $\exists q_{n-\ell} : \varphi \wedge (a \cdot q_{n-\ell} = \rho)$, it follows that ν' is a valid assignment of variables into $\mathbb{N}$. By definition, for each $i \in [0..\ell{-}1]$, evaluating C' on ν assigns to the variable q_{n-i} the (non-negative) integer $\frac{\nu(\tau'_{n-i})}{a}$. Directly from Equation (16), we have $\frac{\nu(\tau'_{n-i})}{a} = \frac{\nu'(\tau_{n-i})}{\eta_C}$. Moreover, since C and C' agree on all the assignments to all variables in $\boldsymbol{x}_{k-1}$, we conclude that $C[x_m](\nu) = C[x_m](\nu')$. □

5.3 A variation of Bareiss algorithm

We now introduce our variation of Bareiss algorithm. As already stated, Bareiss algorithm is commonly used to calculate the echelon form of a matrix with integer entries. For a formal definition of echelon form, we refer the reader to standard linear algebra textbooks (e.g., [HK71]). However, below we do not rely on this definition, as we characterize every single entry that the manipulated matrix has throughout the procedure, in terms of the original ones.

Some notation. Consider a $m \times d$ integer matrix B_0:

$$B_0 \coloneqq \begin{pmatrix} b_{1,1} & \dots & b_{1,d} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \dots & b_{m,d} \end{pmatrix}.$$

Let $\ell \in [0..\min(m,d)]$. We write $b^{(\ell)}_{i,j}$ (with $(i,j) \in [1..m]\times[1..d]$) and $b^{(\ell)}_{r\leftarrow j}$ (with $(r,j) \in [1..\ell]\times[1..d]$) to denote the following sub-determinants of B_0:

$$b^{(\ell)}_{i,j} \coloneqq \det \begin{pmatrix} b_{1,1} & \dots & b_{1,\ell} & b_{1,j} \\ \vdots & \ddots & \vdots & \vdots \\ b_{\ell,1} & \dots & b_{\ell,\ell} & b_{\ell,j} \\ b_{i,1} & \dots & b_{i,\ell} & b_{i,j} \end{pmatrix}, \qquad b^{(\ell)}_{r\leftarrow j} \coloneqq \det \begin{pmatrix} b_{1,1} & \dots & b_{1,r-1} & b_{1,j} & b_{1,r+1} & \dots & b_{1,\ell} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{\ell,1} & \dots & b_{\ell,r-1} & b_{\ell,j} & b_{\ell,r+1} & \dots & b_{\ell,\ell} \end{pmatrix}.$$

Let us fix $k \in [0..\min(m,d)]$ (this quantity corresponds to the number of iterations the algorithm will perform). For every $\ell \in [1..k]$, we define $\lambda_0 \coloneqq 1$ and $\lambda_\ell \coloneqq b^{(\ell-1)}_{\ell,\ell}$. (This notation is chosen intentionally: we will later show that $|\lambda_\ell|$ corresponds to the integer λ computed by Algorithms 4 and 5 when applied to inputs featuring pairs from $\mathcal{I}_k^\ell$.) Throughout this section, we assume that every λ_ℓ is non-zero.

Let us moreover fix a positive integer μ (we will later set μ to the integer μ_C of the LEAC C in input of Algorithms 4 and 5), fix in $g \in [k..d]$ (an index of a column in the B_0), and consider the diagonal matrix $U_g \coloneqq \mathrm{diag}(\mu, \ldots, \mu, 1, \ldots, 1)$ having μ in all positions (i,i) with $i \in [1..g]$, and 1 in positions (i,i) with $i \in [g+1..d]$.

The algorithm. The input matrix is of the form

$$B'_0 \;\coloneqq\; B_0 \cdot U_g \;=\; \begin{pmatrix} \mu \cdot b_{1,1} & \ldots & \mu \cdot b_{1,g} & b_{1,g+1} & \ldots & b_{1,d} \\ \vdots & \ddots & \vdots & & & \\ \mu \cdot b_{m,1} & \ldots & \mu \cdot b_{m,g} & b_{m,g+1} & \ldots & b_{m,d} \end{pmatrix}. \tag{17}$$

The algorithm iteratively constructs a sequence of matrices $B'_1, \ldots, B'_k$ as follows. Consider $\ell \in [0..k-1]$, and let B'_ℓ be the matrix

$$B'_\ell \coloneqq \begin{pmatrix} h_{1,1} & \ldots & h_{1,d} \\ \vdots & \ddots & \vdots \\ h_{m,1} & \ldots & h_{m,d} \end{pmatrix}.$$

The matrix $B'_{\ell+1}$ is constructed from B'_ℓ by applying the following transformation

01: **let** $\pm$ be the sign of $h_{\ell+1,\ell+1}$, and $\alpha \coloneqq \frac{\pm h_{\ell+1,\ell+1}}{\mu}$ ▷ *this division is without remainder*
02: multiply the row $\ell+1$ of B_ℓ by ± 1
03: **for** every row i except row $\ell+1$ **do**
04: **let** $\beta \coloneqq \frac{h_{i,\ell+1}}{\mu}$ ▷ *this division is without remainder*
05: multiply the ith row of B'_ℓ by α ▷ *$B'_\ell(i, \ell+1)$ is now $\alpha \cdot g_{i,\ell+1}$*
06: subtract $\pm\beta \cdot (h_{\ell+1,1}, \ldots, h_{\ell+1,d})$ from the ith row of B'_ℓ
07: divide each entry of the ith row of B'_ℓ by $|\lambda_\ell|$ ▷ *these divisions are without remainder*

Characterization of the entries of the matrices. The following three lemmas fully characterize all entries in the matrices $B'_1, \ldots, B'_k$ in terms of the entries of B'_0. Their proofs are given in Appendix C.4, after introducing the necessary background on the (classical) Bareiss algorithm.

Lemma 15. *For all $\ell \in [0..k-1]$, the $(\ell+1)$th row of $B'_{\ell+1}$ is obtained by multiplying the $(\ell+1)$th row of B'_ℓ by the sign of its $(\ell+1)$th entry.* *Proof in page 96*

Lemma 16. *Consider $\ell \in [0..k]$ and $i \in [\ell+1..m]$, and let $\pm$ be the sign of λ_ℓ. Then:* *Proof in page 96*

1. *For every $j \in [1..g]$, the entry in position (i,j) of the matrix B'_ℓ is $\pm\mu \cdot b^{(\ell)}_{i,j}$. (In particular, this entry is zero whenever $j \le \ell$.)*
2. *For every $j \in [g+1..d]$, the entry in position (i,j) of the matrix B'_ℓ is $\pm b^{(\ell)}_{i,j}$.*

Lemma 17. *Consider $\ell \in [1..k]$ and $i \in [1..\ell]$, and let $\pm$ be the sign of λ_ℓ. Then:* *Proof in page 97*

1. *For every $j \in [1..g]$, the entry in position (i,j) of B'_ℓ is $\pm\mu \cdot b^{(\ell)}_{i \leftarrow j}$. (In particular, this entry is zero if $j \le \ell$ and $i \ne j$, and it is instead $\pm\mu \cdot b^{(\ell-1)}_{\ell,\ell}$ when $i = j$.)*
2. *For every $j \in [g+1..d]$, the entry in position (i,j) of B'_ℓ is $\pm b^{(\ell)}_{i \leftarrow j}$.*

The next lemma provides an alternative algorithm to reconstruct the matrix B'_ℓ starting from its first ℓ rows and from B'_0, without constructing $B'_1, \dots, B'_{\ell-1}$. In the next section, this lemma will turn out useful when analyzing the terms computed by Algorithm 2.

Lemma 18. *Let $\ell \in [0..k]$ and $i \in [\ell+1..m]$. Consider the following transformation applied to B'_0:* *Proof in page 97*

1: *multiply the ith row of B'_0 by $|\lambda_\ell|$*

2: ***for*** *r in $[1..\ell]$* ***do*** *subtract $b_{i,r} \cdot \boldsymbol{u}_r$ to the ith row of B'_0, where $\boldsymbol{u}_r$ is the rth row of B'_ℓ*

After the transformation, the ith rows of B'_0 and B'_ℓ are equal.

5.4 How coefficients evolve as ElimVars executes, and proof of Claim 6

We now prove that the evolution of the variable coefficients during ElimVars mirrors that of the matrix entries in our variation of Bareiss algorithm. (For brevity, in this section by Bareiss algorithm we always mean this variation.) This is done by setting up a sequence of matrices $M_0, M_1, \dots$, where M_ℓ snapshots the coefficients of the variables $\boldsymbol{q}_k$ and u in the (in)equalities of γ or in the terms of the circuit C, after ℓ iterations the **while** loop in ElimVars. We then show that M_ℓ can alternatively be obtained from M_0 by performing ℓ iterations of Bareiss algorithm. In doing so, we also establish Claim 6.

Throughout the section, let $(\boldsymbol{q}_{k-1}, C[x_m], \langle \gamma \,;\, \psi \rangle)$ be the triple in input to ElimVars, with $(C, \langle \gamma \,;\, \psi \rangle) \in \mathcal{I}^0_k$, for some $k \in [0..n-1]$. Also, let $\mu \coloneqq \mu_C$. Since Algorithm 5 does not modify the formula ψ, each non-deterministic branch of ElimVars can be identified with a sequence of pairs

$$(C, \gamma) = (C_0, \gamma_0) \xrightarrow{e_0} (C_1, \gamma_1) \xrightarrow{e_1} (C_2, \gamma_2) \dots \xrightarrow{e_{j-1}} (C_j, \gamma_j) \xrightarrow{e_j} \dots \tag{18}$$

where e_ℓ is the equality computed by Algorithm 4 during the $(\ell+1)$th iteration of ElimVars, and C_ℓ and γ_ℓ are the circuit C and formula γ at the completion of the ℓth iteration of ElimVars, respectively. At this point, the length of this sequence is unknown. It might be short (e.g., if one of the **assert** commands of Algorithm 5 fails and the algorithm rejects), or even infinite —we are not assuming Claim 6, and thus cannot guarantee that variables are eliminated correctly. Nonetheless, we will prove that in non-rejecting runs, each iteration of the **while** loop in ElimVars eliminates one of the variables $\boldsymbol{q}_{k-1}$. Because of this, we truncate the above sequence to some $j \in [0..k]$, and focus our analysis on this finite prefix.

Towards defining the matrices. Let us fix an enumeration

$$(\rho_1 \sim_1 0), (\rho_2 \sim_2 0), \dots, (\rho_t \sim_t 0) \tag{19}$$

of all the equalities and inequalities of γ_0 (that is, each $\sim_i$ belongs to $\{=, \leq\}$). Observe that Algorithm 5 constructs $\gamma_{\ell+1}$ from γ_ℓ by simply applying a substitution (line 3), adding a divisibility constraint (again line 3), and performing some integer divisions. Since substitutions are applied locally to each constraints, this implies not only that the number of (in)equalities in $\gamma_0, \gamma_1, \dots, \gamma_j$ does not change, but that in fact there is a one-to-one mapping between (in)equalities of γ_0 and those in γ_ℓ. That is, there is an enumeration of the (in)equalities of γ_ℓ such that the ith element of the enumeration is the inequality obtained from $\rho_i \sim_i 0$ by applying all the substitutions and divisions performed by Algorithm 5 during the first ℓ iterations of ElimVars. We will denote such a one-to-one mapping, from (in)equalities of γ_0 to those in γ_ℓ, as Λ_ℓ (Λ_0 is the identity).

Let us now look at the equality e_ℓ (with $\ell \in [0..j-1]$). Following Algorithm 4, this equality is of the form $a \cdot q = \tau - s$, where s is the shift introduced in line 3 of Algorithm 4, and $a \cdot q - \tau$ is a term of one of the following two types:

$$
\begin{array}{c}
\begin{array}{cccccc} q_n & q_{n-1} & \dots & q_{n-k+1} & q_{n-k} & u \end{array} \\
\left(\begin{array}{cccccc}
\mu \cdot b_{1,1} & \mu \cdot b_{1,2} & \dots & \mu \cdot b_{1,k} & \mu \cdot b_{1,k+1} & b_{1,k+2} \\
\mu \cdot b_{2,1} & \mu \cdot b_{2,2} & \dots & \mu \cdot b_{2,k} & \mu \cdot b_{2,k+1} & b_{2,k+2} \\
\vdots & & & & & \\
\mu \cdot b_{j,1} & \mu \cdot b_{j,2} & \dots & \mu \cdot b_{j,k} & \mu \cdot b_{j,k+1} & b_{j,k+2} \\
\mu \cdot b_{j+1,1} & \mu \cdot b_{j+1,2} & \dots & \mu \cdot b_{j+1,k} & \mu \cdot b_{j+1,k+1} & b_{j+1,k+2} \\
\vdots & & & & & \\
\mu \cdot b_{j+t,1} & \mu \cdot b_{j,2} & \dots & \mu \cdot b_{j+t,k} & \mu \cdot b_{j+t,k+1} & b_{j+t,k+2}
\end{array}\right)
\begin{array}{l}
\text{coefficients of } g_0 \\
\text{coefficients in } g_1 \\
\\
\text{coefficients in } g_{j-1} \\
\text{coefficients in } \rho_1 \\
\\
\text{coefficients in } \rho_t
\end{array}
\end{array}
$$

Figure 2: Structure of the matrix M_0. Observe that all coefficients of the variables in $\boldsymbol{q}_k$ have μ as a common factor. This is because $(C, \langle \gamma \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^0$.

I. *A term from* terms$(\gamma_\ell \wedge \gamma_\ell[q + p \,/\, q])$. Note that the substitution $[q + p \,/\, q]$ affects only the constants of (in)equalities. Consequently, there is an (in)equality $\rho' \sim 0$ in γ_ℓ where the term ρ has the same variable coefficients as $a \cdot q - \tau$. In this case, we define the *generator* g_ℓ *of* e_ℓ to be the (in)equality $\rho \sim 0$ of γ_0 such that $\Lambda_\ell(\rho \sim 0) = (\rho' \sim 0)$.

II. *A term computed using Algorithm 2 with respect to* (C_ℓ, γ_ℓ, p). This is a term obtained by simultaneously applying two substitutions to a term ρ of the form $a \cdot u + \mu_{C_\ell} \cdot (q' - q'') + d$, with $a, d \in \mathbb{Z}$ and q', q'' from $\boldsymbol{q}_{k-1}$. We define the *generator* g_ℓ *of* e_ℓ to be the equality $\rho = 0$.

We say that e_ℓ is of *Type I* or *Type II*, depending on which of the two cases above it falls under.

The matrix associated to γ_0. Each of the matrices $M_0, \dots, M_j$ we define have $j+t$ rows (where t is the number of equalities and inequalities in γ_0) and $k+2$ columns. For every $i \in [0..k]$, the $(i+1)$th column contains coefficients of the variable q_{n-i}. The $(k+2)$th column contains coefficients of u.

In the matrix M_0, for $i \in [1..j]$, the ith row stores the coefficients of the variables $\boldsymbol{q}_k$ and u occurring in the generator g_{i-1} of e_{i-1}. The remaining t rows store the coefficients of the variables $\boldsymbol{q}_k$ and u occurring in (in)equalities of γ_0: following the enumeration in Equation (19), the $(j+r)$th row stores the variable coefficients of ρ_r. The structure of the matrix M_0 is illustrated in Figure 2.

The matrices $M_1, \dots, M_j$. Let $\ell \in [1..j]$. At the ℓth iteration of the **while** loop of ElimVars, Algorithm 5 *prepends* a single assignment $q \leftarrow \frac{\tau}{a}$ to the circuit $C_{\ell-1}$ (where $\pm a \cdot q = \pm\tau$ is $e_{\ell-1}$, with $\pm$ being the sign of a), and modifies all other assignments to variables from $\boldsymbol{q}_{k-1}$ so that the denominator of the assigned expression becomes a. This means that $\eta_{C_\ell} = a$, which we abbreviate as η_ℓ. Note that then C_ℓ features ℓ assignments to variables in $\boldsymbol{q}_{k-1}$, and the remaining assignments are the original ones from C_0, featuring the variables $\boldsymbol{x}_k$. In particular, $\mu_{C_\ell} = \mu$. We define the matrix M_ℓ as follows:

i. For $i \in [1..\ell]$, let $q \leftarrow \frac{\tau}{\eta_\ell}$ be the $(\ell - (i-1))$th assignment in C_ℓ. The ith row of M_ℓ contains the coefficients of the variables $\boldsymbol{q}_k$ and u from the term $\eta_\ell \cdot q - \tau$.

ii. For every $i \in [\ell + 1..j]$, if e_{i-1} is of Type I, then the ith row of M_ℓ contains the coefficients of the variables $\boldsymbol{q}_k$ and u occurring in the (in)equality $\Lambda_\ell(g_{i-1})$.

iii. For $i \in [\ell+1..j]$, if e_{i-1} is of Type II, then let $\rho = 0$ be g_{i-1}. The ith row of M_ℓ contains the coefficients of the variables $\boldsymbol{q}_k$ and u from the term obtained from ρ as follows:

1: multiply every integer in ρ by the quotient of the division of η_ℓ by μ
2: **for** r in $[1..\ell]$ **do** $\rho \leftarrow \rho[\tau \,/\, \eta_\ell \cdot q]$, where $q \leftarrow \frac{\tau}{\eta_\ell}$ is the rth assignment in C_ℓ

(We will later see that this is in fact a simultaneous substitution.)

iv. For every $i \in [1..t]$, the $(j+i)$th row of M_ℓ contains the coefficients of the variables $\boldsymbol{q}_k$ and u in the term of the (in)equality $\Lambda_\ell(\rho_i \sim_i 0)$.

The following lemma is immediate:

Lemma 19. *Let $\ell \in [0..j]$. For every assignment $q \leftarrow \frac{\tau}{\eta_\ell}$ in C_ℓ, with q in $\boldsymbol{q}_k$, there is a row in M_ℓ whose entries encode the coefficients that $\boldsymbol{q}_k$ and u have in $\eta_\ell \cdot q - \tau$. Similarly, for every (in)equality $\rho \sim 0$ in γ_ℓ, there is a row of M_ℓ whose entries encode the coefficients that $\boldsymbol{q}_k$ and u have in ρ.*

Proof. For $\ell = 0$, C_0 has no assignments to variables in $\boldsymbol{q}_k$, and the ith entry in the enumeration of Equation (19) is in row $j+i$. For $\ell \geq 1$, the lemma follows from Items (i) and (iv) above. □

Correspondence between ElimVars and Bareiss algorithm. The matrix M_0 has the form required to run (our) Bareiss algorithm. Let us denote by B_0 the $(j+t) \times (k+2)$ integer matrix such that $M_0 = B_0 \cdot \mathrm{diag}(\mu, \ldots, \mu, 1)$, and by b_{ij} the entry of B_0 in position (i,j); as in Figure 2. We also use the notation $b^{(\ell)}_{i,j}$ and $b^{(\ell)}_{r \leftarrow j}$ to denote the sub-determinants of B_0 analogous to those in Section 5.3, and define $\lambda_0 \coloneqq 1$ and $\lambda_\ell \coloneqq b^{(\ell-1)}_{\ell,\ell}$. We write $B'_1, \ldots, B'_j$ for the sequence of matrices iteratively constructed by Bareiss algorithm, starting from the matrix $B'_0 \coloneqq M_0$. The next lemma establishes the key correspondence between these matrices and $M_1, \ldots, M_j$.

Lemma 20. *Consider $\ell \in [0..j]$. Then, $M_\ell = B'_\ell$, $\frac{\eta_\ell}{\mu} = |\lambda_\ell| \neq 0$, and if $\ell \geq 1$, then Claim 6 holds when restricted to Algorithm 5 having as input $(C_{\ell-1}[x_m], \langle \gamma_{\ell-1} \,;\, \psi \rangle)$ and the equality $e_{\ell-1}$.*

Proof. The proof is by induction on $\ell \in [0..j]$.

base case: $\ell = 0$. By definition, $M_0 = B'_0$. Since C_0 is a $(k,0)$-LEAC, $\eta_0 = \mu$, and so $\frac{\eta_0}{\mu} = 1 = \lambda_0$.

induction hypothesis. For $\ell \geq 1$, $M_{\ell-1} = B'_{\ell-1}$ and $\frac{\eta_{\ell-1}}{\mu} = |\lambda_{\ell-1}| \neq 0$. Moreover, if $\ell \geq 2$, then Claim 6 holds when restricted to Algorithm 5 having as input $(C_{\ell-2}[x_m], \langle \gamma_{\ell-2} \,;\, \psi \rangle)$ and the equality $e_{\ell-2}$.

induction step: $\ell \geq 1$. By induction hypothesis, Claim 6 holds whenever Algorithm 5 is called on the inputs $(C_r[x_m], \langle \gamma_r \,;\, \psi \rangle)$ and the equality e_r, for every $r \in [0..\ell-2]$. In particular, following the correctness arguments given in the proof of Lemma 14, we conclude that ElimVars is correct for the first $\ell - 1$ iterations, and $(C_{\ell-1}[x_m], \langle \gamma_r \,;\, \psi \rangle)$ thus belong to $\mathcal{I}_k^{\ell-1}$. Indeed, to obtain correctness for the first $\ell - 1$ iterations, it suffices to restrict Claim 6 to the first $\ell - 1$ calls of Algorithm 5, featuring the equalities $e_0, \ldots, e_{\ell-2}$.

Our first goal is to prove that the variable coefficients of the equality $e_{\ell-1}$ are stored in the ℓth row of $M_{\ell-1}$. Note that since $(C_{\ell-1}[x_m], \langle \gamma_r \,;\, \psi \rangle) \in \mathcal{I}_k^{\ell-1}$, Algorithm 4 outputs an equality of the form $a \cdot q_{n-(\ell-1)} = \tau$ with $a \neq 0$, and that, by definition, this equality is $e_{\ell-1}$.

Claim 8. *The ℓth row of $M_{\ell-1}$ contains the variable coefficients of the term $a \cdot q_{n-(\ell-1)} - \tau$.*

Proof. If $e_{\ell-1}$ is of Type I, then by definition the ℓth row of $M_{\ell-1}$ contains the coefficients of the variables $\boldsymbol{q}_k$ and u occurring in the (in)equality $\rho' \sim 0$ given by $\Lambda_{\ell-1}(g_{\ell-1})$. (In particular, for $\ell = 1$, we have $\rho' \sim 0$ equal to g_0.) By definition of $\Lambda_{\ell-1}$, the terms ρ' and $a \cdot q_{n-(\ell-1)} - \tau$ share the same variable coefficients.

If $e_{\ell-1}$ is of Type II, then $g_{\ell-1}$ is an equality $\rho = 0$ where ρ is of the form $b \cdot u + \mu \cdot (q' - q'') + d$, with $b, d \in \mathbb{Z}$ and q', q'' from $\boldsymbol{q}_k$. Moreover, inspecting Algorithm 2, we see that the variable coefficients of $a \cdot q_{n-(\ell-1)} - \tau$ corresponds to the ones obtained from $b \cdot u + \mu \cdot (q' - q'') + d$ by simultaneously applying two substitutions ν_1 and ν_2. The substitution ν_1 has one of the following forms (recall that $\frac{\eta_{\ell-1}}{\mu} = |\lambda_{\ell-1}|$, by induction hypothesis):

1. $[\frac{\eta_{\ell-1} \cdot q'}{|\lambda_{\ell-1}|} \,/\, \mu \cdot q']$; this is the case when $C_{\ell-1}$ does not assign any expression to q', and the **guess** in line 6 returns false.
2. $[\frac{\eta_{\ell-1} \cdot q' + \eta_{\ell-1} \cdot p}{|\lambda_{\ell-1}|} \,/\, \mu \cdot q']$, with $p \coloneqq \mathit{mod}(q_{n-(\ell-1)}, \gamma_{\ell-1})$; this is the case when $q' = q_{n-(\ell-1)}$, $C_{\ell-1}$ does not assign any expression to q', and the **guess** in line 6 returns true.
3. $[\frac{\tau'}{|\lambda_{\ell-1}|} \,/\, \mu \cdot q']$; in this case $C_{\ell-1}$ features $q' \leftarrow \frac{\tau'}{\eta_{\ell-1}}$ and the **guess** in line 6 returns false.
4. $[\frac{\tau'\sigma}{|\lambda_{\ell-1}|} \,/\, \mu \cdot q']$ where σ is the substitution $[q_{n-(\ell-1)} + p \,/\, q_{n-(\ell-1)}]$; in this case $C_{\ell-1}$ features $q' \leftarrow \frac{\tau'}{\eta_{\ell-1}}$ and the **guess** in line 6 returns true.

Observe that the coefficients of $\boldsymbol{q}_k$ and u are the same in τ' and $\tau'\sigma$, and that moreover these terms do not contain variables to which $C_{\ell-1}$ assigns some expressions (because $C_{\ell-1}$ is a $(k, \ell-1)$-LEAC). We also note that the only effect that the substitutions in Items 1 and 2 have on the variable coefficients of ρ is to multiply them by $|\lambda_{\ell-1}|$, because after this multiplication, $|\lambda_{\ell-1}| \cdot \mu \cdot q'$ is replaced by $\eta_{\ell-1} \cdot q'$ or $\eta_{\ell-1} \cdot q' + \eta_{\ell-1} \cdot p$, but $\eta_{\ell-1} = |\lambda_{\ell-1}| \cdot \mu$.

An analysis similar to the one above can be performed for ν_2 (simply change q' for q'', and line 6 for line 7, in the items above). From the definition of simultaneous substitution, we then conclude that the variables coefficients of $a \cdot q_{n-(\ell-1)} - \tau$ are exactly those in the term obtained from ρ by simultaneously applying all substitutions of the form $[\frac{\rho'}{|\lambda_{\ell-1}|} \,/\, \mu \cdot q]$, where $q \in \{q_n, \ldots, q_{n-(\ell-2)}\}$ and $q \leftarrow \frac{\rho'}{\eta_{\ell-1}}$ is an assignment in $C_{\ell-1}$. For $\ell = 1$, this list of substitutions is empty, and indeed by definition of M_0, the ℓth row contains the variable coefficients of ρ. For $\ell > 1$, these substitutions correspond to the transformation applied to ρ in Item (iii) of the definition of $M_{\ell-1}$, in order to define its ℓth column. The claim then holds. □

We now analyze M_ℓ and B'_ℓ row by row, showing that the two matrices are equal. In the process, we will also prove the other statements in the lemma. Below, we write $\pm$ for the sign of the coefficient $a \neq 0$ in the term $a \cdot q_{n-(\ell-1)} - \tau$, and define $\alpha \coloneqq \frac{\pm a}{\mu}$ (by Lemma 13 this division is without remainder). Let us also write

$$q_{n-(\ell-2)} \leftarrow \frac{\tau_{n-(\ell-2)}(u, \boldsymbol{q}_{[\ell-1,k]})}{\eta_{\ell-1}}, \quad \ldots, \quad q_n \leftarrow \frac{\tau_n(u, \boldsymbol{q}_{[\ell-1,k]})}{\eta_{\ell-1}}$$

for the sequence of all the assignments to variables in $\boldsymbol{q}_k$ featured in $C_{\ell-1}$ (this is a prefix of all the assignments in $C_{\ell-1}$, since this circuit is a $(k, \ell-1)$-LEAC).

The following sequence of Claims summarizes our analysis. As their proofs are rather similar (each crucially relying on Lemmas 15 to 18) below we only provide a detailed proof of Claim 10, deferring the proofs of the remaining claims to Appendix C.6.

Claim 9. *The ℓth rows of M_ℓ and B'_ℓ are equal. Moreover, $\eta_\ell = \pm a$ and $\alpha = \frac{\eta_\ell}{\mu} = |\lambda_\ell| \neq 0$.* *Proof in page 98*

Claim 10. *Let $i \in [1..\ell-1]$. The ith rows of M_ℓ and B'_ℓ are equal. Moreover, in all terms $\tau_{n-r}[\frac{\pm\tau}{\alpha} \,/\, \mu \cdot q_{n-\ell-1}]$, with $r \in [0..\ell-2]$, all coefficients of the variables $\boldsymbol{q}_{[\ell,k]}$ are divisible by $\eta_{\ell-1}$, and all coefficients of u are divisible by $\frac{\eta_{\ell-1}}{\mu}$.*

Claim 11. *Let $i \in [j{+}1..j{+}t]$. The ith rows of M_ℓ and B'_ℓ are equal. Moreover, in all equalities and inequalities $\gamma_{\ell-1}[\frac{\pm\tau}{\alpha} \,/\, \mu \cdot q_{n-\ell-1}]$, all coefficients of the variables $\boldsymbol{q}_{[\ell,k]}$ are divisible by $\eta_{\ell-1}$, and all coefficients of u are divisible by $\frac{\eta_{\ell-1}}{\mu}$.* *Proof in page 99*

Claim 12. *Let $i \in [\ell+1..j]$. The ith rows of M_ℓ and B'_ℓ are equal.* *Proof in page 99*

Proof of Claim 10. By definition (Item (i)), the matrix M_ℓ includes, in these rows, the variable coefficients of the assignments in C_ℓ, ranging from the $(\ell-1)$th assignment to the 2nd one (in this reverse order). The corresponding rows in $M_{\ell-1}$ contain the variable coefficients of the terms $(\eta_{\ell-1} \cdot q_n - \tau_n)$ to $(\eta_{\ell-1} \cdot q_{n-(\ell-2)} - \tau_{n-(\ell-2)})$, which arise from the assignments in $C_{\ell-1}$. Let $i \in [1..\ell-1]$. Since line 7 of Algorithm 5 prepends one assignment to $C_{\ell-1}$ and only updates the rest, the ith row of M_ℓ corresponds precisely to the term obtained by running line 7 on the assignment $q_{n-(i-1)} \leftarrow \frac{\tau_{n-(i-1)}}{\eta_{\ell-1}}$. We will analyze this update below. At the same time, since the variable coefficients of this assignment are stored in the ith row of $B'_{\ell-1}$, we can alternatively track how Bareiss algorithm updates this row when computing B'_ℓ from $B'_{\ell-1}$. We will then deduce that the ith rows of M_ℓ and B'_ℓ are equal.

Let us examine how line 7 updates the assignment $q_{n-(i-1)} \leftarrow \frac{\tau_{n-(i-1)}}{\eta_{\ell-1}}$. Let us write $\tau_{n-(i-1)}$ as $\beta \cdot \mu \cdot q_{n-(\ell-1)} + \tau'$. Line 7 first constructs the term $\tau_{n-(i-1)}[\frac{\pm\tau}{\alpha} \,/\, \mu \cdot q_{n-(\ell-1)}]$. The substitution multiplies $\tau_{n-(i-1)}$ by $\alpha \geq 1$, to then replace $\alpha \cdot \mu \cdot q_{n-(\ell-1)}$ by $\pm\tau$. The resulting term is $\pm\beta \cdot \tau + \alpha \cdot \tau'$ Note that multiplying the denominator by α yields $\alpha \cdot \eta_{\ell-1} = \frac{\pm a}{\mu} \cdot \eta_{\ell-1} = \pm a \cdot \frac{\eta_{\ell-1}}{\mu}$. So, at this intermediate stage of line 7, the assignment can be viewed as

$$q_{n-(i-1)} \leftarrow \frac{\pm\beta \cdot \tau + \alpha \cdot \tau'}{\pm a \cdot \frac{\eta_{\ell-1}}{\mu}}\,. \tag{20}$$

In the upcoming analysis of the update performed by Bareiss algorithm, we will show that every variable coefficient in the term $\pm\beta \cdot \tau + \alpha \cdot \tau'$ is divisible by $\frac{\eta_{\ell-1}}{\mu}$. This implies that the constant of the term must also be divisible by $\frac{\eta_{\ell-1}}{\mu}$; otherwise the expression in Equation (20) would fail to evaluate to an integer under any variable assignment. This explains the **assert** command of line 7. Line 7 concludes by dividing every integer in the term $\pm\beta \cdot \tau + \alpha \cdot \tau'$ by $\frac{\eta_{\ell-1}}{\mu}$, and setting the denominator to $\pm a$. Let us write $(\pm\beta \cdot \tau + \alpha \cdot \tau')/\frac{\eta_{\ell-1}}{\mu}$ for the term resulting from these divisions. The circuit C_ℓ thus features the assignment

$$q_{n-(i-1)} \leftarrow \frac{(\pm\beta \cdot \tau + \alpha \cdot \tau')/\frac{\eta_{\ell-1}}{\mu}}{\pm a}\,,$$

and the ith row of M_ℓ stores the variable coefficients of $\pm a \cdot q_{n-(i-1)} - \big((\pm\beta \cdot \tau + \alpha \cdot \tau')/\frac{\eta_{\ell-1}}{\mu}\big)$.

We now turn to Bareiss algorithm. By induction hypothesis, the ith row of $B'_{\ell-1}$ contains the variable coefficients of the term $\eta_{\ell-1} \cdot q_{n-(i-1)} - (\beta \cdot \mu \cdot q_{n-(\ell-1)} + \tau')$. The algorithm first multiplies this row by α, resulting in the variable coefficients of $\alpha \cdot \eta_{\ell-1} \cdot q_{n-(i-1)} - \alpha(\beta \cdot \mu \cdot q_{n-(\ell-1)} + \tau')$. Next, the algorithm subtracts to this row the quantity $\pm(-\beta) \cdot \boldsymbol{r}_\ell$, where $\boldsymbol{r}_\ell$ is

the ℓth row of $B'_{\ell-1}$. By Claim 8, $\boldsymbol{r}_\ell$ holds the variable coefficients of $a \cdot q_{n-(\ell-1)} - \tau$. Hence, after this subtraction, the ith row contains the variable coefficients of the term

$$\begin{aligned} &\alpha \cdot \eta_{\ell-1} \cdot q_{n-(i-1)} - \alpha(\beta \cdot \mu \cdot q_{n-(\ell-1)} + \tau') - \pm(-\beta) \cdot (a \cdot q_{n-(\ell-1)} - \tau) \\ = \; &\alpha \cdot \eta_{\ell-1} \cdot q_{n-(i-1)} - (\pm\beta \cdot \tau + \alpha \cdot \tau'). \end{aligned}$$

Lastly, each entry of the ith row is divided by $|\lambda_{\ell-1}| = \frac{\eta_{\ell-1}}{\mu}$. Thanks to Lemma 17, we know that these divisions are exact, since the results correspond to sub-determinants of the matrix B'_0. Moreover, because $q_{n-(i-1)}$ does not appear in neither τ nor τ', we conclude that every variable coefficient in $\pm\beta \cdot \tau + \alpha \cdot \tau'$ is divisible by $\frac{\eta_{\ell-1}}{\mu}$. Therefore, the divisions performed in line 7 of Algorithm 5 are also without remainder. Since $\alpha \cdot \eta_{\ell-1} = \pm a \cdot \frac{\eta_{\ell-1}}{\mu}$, we conclude that the ith row of B'_ℓ holds the variables coefficients of $\pm a \cdot q_{n-(i-1)} - \big((\pm\beta \cdot \tau + \alpha \cdot \tau')/\frac{\eta_{\ell-1}}{\mu}\big)$. That is, the ith rows of M_ℓ and B'_ℓ coincide.

To complete the proof, let us address the second statement of the claim. Consider once more the term $\tau_{n-(i-1)}[\frac{\pm\tau}{\alpha} \,/\, \mu \cdot q_{n-(\ell-1)}]$, that is, $\pm\beta \cdot \tau + \alpha \cdot \tau'$. We have already established that all variable coefficients of this term are divisible by $\frac{\eta_{\ell-1}}{\mu}$; in particular, this shows the second statement of the claim for the variable u. As for the remaining variables, Lemma 17 guarantee that, once divided by $\frac{\eta_{\ell-1}}{\mu}$, their coefficients are still divisible by μ. Therefore, in $\pm\beta \cdot \tau + \alpha \cdot \tau'$, all coefficients of variables other than u are divisible by $\eta_{\ell-1}$. □

Lemma 20 follows: Claims 9 to 12 imply that $M_\ell = B'_\ell$. Claim 9 establishes $\frac{\eta_\ell}{\mu} = |\lambda_\ell| \neq 0$. Claims 10 and 11 imply that Claim 6 holds when restricted to Algorithm 5 having as input the pair $(C_{\ell-1}[x_m], \langle \gamma_{\ell-1} \,;\, \psi \rangle)$ and the equality $e_{\ell-1}$. □

Claim 6 follows as a corollary of Lemma 20; thus completing the proof of correctness of ElimVars. Proof in page 101

5.5 Complexity of ElimVars

In addition to being crucial for establishing the correctness of ElimVars, Lemma 20 allows us to obtain a refined complexity analysis of the procedure. The next lemma summarizes this analysis.

Lemma 21. *The algorithm from Lemma 14 runs in non-deterministic polynomial time. Consider its execution on an input $(\boldsymbol{q}, C[x_m], \langle \gamma \,;\, \psi \rangle)$, where $(C, \langle \gamma \,;\, \psi \rangle)$ belongs to $\mathcal{I}^0_k$, and define:* Proof in page 101

$$\begin{aligned} L &\coloneqq 3 \cdot \mu_C \cdot (4 \cdot \lceil \log_2(2 \cdot \xi_C + \mu_C) \rceil + 8), \\ Q &\coloneqq \max\{|b| : b \in \mathbb{Z} \text{ is a coefficient of } q_{n-k} \text{ or of a variable in } \boldsymbol{q}, \text{ in a term from } \mathrm{terms}(\gamma)\}, \\ U &\coloneqq \max\{|a| : a = L \text{ or } a \in \mathbb{Z} \text{ is a coefficient of } u \text{ in a term from } \mathrm{terms}(\gamma)\}, \\ R &\coloneqq \max\{|d| : d = L \text{ or } d \in \mathbb{Z} \text{ is a constant of a term from } \mathrm{terms}(\gamma)\}. \end{aligned}$$

In each non-deterministic branch β, the algorithm returns a pair $(C'[x_m], \langle \gamma' \,;\, \psi \rangle)$ such that:

1. *γ' features k constraints more than γ, they are all divisibility constraints.*
2. *The circuits C and C' assign the same expressions to $x_{n-k}, \ldots, x_n$ (in particular, $\mu_C = \mu_{C'}$).*
3. *In terms τ either from $\mathrm{terms}(\gamma')$ or in assignments $q_{n-i} \leftarrow \frac{\tau}{\eta_{C'}}$ of C' (where $i \in [0..k-1]$),*
   - *the coefficient of the variable q_{n-k} is $\mu_C \cdot c$, for some $c \in \mathbb{Z}$ with $|c| \leq (k+1)^{k+1}\big(\frac{Q}{\mu_C}\big)^{k+1}$;*
   - *the absolute value of the coefficient of the variable u is bounded by $(k+1)^{k+1}\big(\frac{Q}{\mu_C}\big)^{k} U$;*

- *the absolute value of the constant is bounded by* $\frac{((k+1)\cdot Q)^{2(k+2)^2}}{(\mu_C)^{2k^2}} \cdot mod(\gamma) \cdot R$.

4. *The positive integer* $mod(\gamma')$ *divides* $c \cdot mod(\gamma)$*, for some positive integer* $c \leq \frac{(k\cdot Q)^{k^2}}{(\mu_C)^{k(k-1)}}$.
5. *We have* $\eta_{C'} = \mu_C \cdot g$*, for some positive integer* $g \leq k^k \big(\frac{Q}{\mu_C}\big)^k$.

Proof idea. The bounds follow by applying Lemma 20 in conjunction with Lemmas 16 and 17, and recalling that the Leibniz formula for determinants yields $|\det(A)| \leq d^d \cdot \prod_{i=1}^{d} \alpha_i$ for any $d \times d$ integer matrix A in which the entries of the ith column are bounded, in absolute value, by $\alpha_i \in \mathbb{N}$. □

6 Proof of Theorem 1

In this section, we complete the proof of Theorem 1: we define the procedure OptILEP for solving the integer linear-exponential programming optimization problem, to then show that the procedure runs in non-deterministic polynomial time and returns an ILESLP encoding an (optimal) solution, if one exists. The section is divided into four parts. We begin with an overview of the procedure, expanding on the brief summary provided in Section 2.2. As part of this overview, we introduce a slight variant of LEACs, which we refer to as PreLEACs. In Section 6.2 we present the full correctness proof of OptILEP, followed by its complexity analysis in Section 6.3. The pseudocode of OptILEP considers the setting of maximizing a single variable x subject to an integer linear-exponential program. In Section 6.4 we show (using rather standard arguments), how to extend the procedure to the optimization (maximization or minimization) of arbitrary linear-exponential terms, completing the proof of Theorem 1.

6.1 Overview of OptILEP

The pseudocode of OptILEP is given in Algorithm 6. Echoing Section 2.2, the procedure starts by guessing an ordering θ of the form $2^{x_n} \geq \cdots \geq 2^{x_1} \geq 2^{x_0} = 1$, where $x_1, \ldots, x_n$ are the variables appearing input (φ, w) of OptILEP, see lines 1–4. These lines also initialize the *remainder variables* $\boldsymbol{r}$ (as described in Section 2), and the circuit C, which will ultimately become the ILESLP encoding the computed solution. After this initialization step, the procedure enters its main loop.

Let 2^x and 2^y be the leading and second-leading exponential terms of θ, respectively (as in lines 6 and 7). As mentioned in Section 2.2, the main loop of the procedure eliminates x by mirroring the four steps of the procedure from [CMS24], with the key difference that Step II is replaced by our optimum-preserving procedure ElimVars. We refer the reader back to Section 2 for a refresher, particularly on the specifications of Steps I and III, which we treat here as black boxes.

As discussed in Section 2, Step I "divides" all constraints in φ by 2^y, non-deterministically computing from φ and θ a pair of formulae of the form $(\gamma(q_x, \boldsymbol{q}, u), \psi(\boldsymbol{y}, r_x, \boldsymbol{r'}))$, where, in particular, γ is a linear program with divisions. As described in the specification of Step I given by Lemma 4, φ and (γ, ψ) are "coupled" by the system featuring the equalities $x = q_x \cdot 2^y + r_x$ and $\boldsymbol{r} = \boldsymbol{q} \cdot 2^y + \boldsymbol{r'}$ (Equation (3)), with q_x and $\boldsymbol{q}$ *quotient variables*, and r_x and $\boldsymbol{r'}$ fresh remainder variables. The change of variables given by this system must be applied also to the circuit C; this is done in line 9.

The goal of Step II is to eliminate the variables $\boldsymbol{q}$ from γ. We preserve optimal solutions while eliminating these variables by appealing to our instantiation of ElimVars. However, according to Lemma 14, a correct invocation to this algorithm requires that its input belong to $\mathcal{I}_k^0$ (for some k). Accordingly, lines 10–12 perform the necessary manipulations on γ and ψ to ensure this condition is met.

Algorithm 6 OptILEP: Exploration of optimal solutions for ILEP.

Input: $\varphi(\boldsymbol{x})$: integer linear-exponential program;
w : a variable from $\boldsymbol{x}$ (to be maximized)
Output of each branch (β): An ILESLP σ_β.

$C \leftarrow \emptyset$ ▷ *the empty 0-PreLEAC. The objective function is $C[w]$*
let x_0 be a fresh variable
$\theta \leftarrow$ **guess** ordering $2^{x_n} \geq \ldots \geq 2^{x_1} \geq 2^{x_0} = 1$, where $x_1, \ldots, x_n$ is a permutation of $\boldsymbol{x}$
▷ *below, we write x_m for the variable among $x_1, \ldots, x_n$ corresponding to w*
$\boldsymbol{r} \leftarrow$ empty vector of (remainder) variables
while θ is not the ordering $2^{x_0} = 1$ **do**
  Step I:
  $2^x \leftarrow$ leading exponential term of θ
  $2^y \leftarrow$ second-leading exponential term of θ
  $(\gamma(q_x, \boldsymbol{q}, u), \psi(\boldsymbol{y}, r_x, \boldsymbol{r'})) \leftarrow$ apply the algorithm from Lemma 4 on (φ, θ)
  ▷ *$(q_x, \boldsymbol{q})$ quotient variables, $(r_x, \boldsymbol{r'})$ new remainder variables, u proxy for 2^{x-y}*
  update C: add the assignment $x \leftarrow q_x \cdot 2^y + r_x$, and replace each variable in $\boldsymbol{r}$ following the system $\boldsymbol{r} = \boldsymbol{q} \cdot 2^y + \boldsymbol{r'}$ stemming from the above call of the algorithm form Lemma 4
  Step II:
  $\gamma \leftarrow \gamma \wedge \boldsymbol{q} \geq 0 \wedge q_x \geq 0$ ▷ *prepare formulae for the call to* ElimVars
  update γ: replace each (in)equality $\tau \sim 0$ with $\mu_C \cdot \tau \sim 0$
  $\psi' \leftarrow \psi \wedge \theta \wedge (x = q_x \cdot 2^y + r_x) \wedge (u = 2^{x-y})$
  $(C[x_m], \langle \gamma' ; \psi' \rangle) \leftarrow$ ElimVars($\boldsymbol{q}$, $C[x_m]$, $\langle \gamma ; \psi' \rangle$) ▷ *Lemma 14*
  $(\gamma''(q_x), \psi''(y, r_x)) \leftarrow$ apply the algorithm from Lemma 6 on γ' ▷ Step III
  Step IV:
  $\ell \leftarrow$ greatest non-negative lower bound of q_x in γ'' ▷ *default:* 0
  $h \leftarrow$ least upper bound of q_x in γ'' ▷ *default:* ∞
  if $h = \infty$ **then** $h \leftarrow \ell + mod(\gamma'')$
  $v \leftarrow$ **guess** a value in $[\ell..h]$ such that $\gamma''(v)$ is true
  update C: translate into a PreLEAC following Remark 4, replacing u for 2^{x-y} and q_{n-k} for v
  $\boldsymbol{r} \leftarrow (r_x, \boldsymbol{r'})$
  $\varphi \leftarrow \psi \wedge \psi''$
  remove 2^x from θ
assert($\varphi(\mathbf{0})$ is true)
update C: replace x_0 and every variable in $\boldsymbol{r}$ with 0
return C ▷ *C is an ILESLP encoding a solution to φ*

After eliminating the variables $\boldsymbol{q}$ and appropriately updating the circuit C via ElimVars, line 14 applies Step III from [CMS24]. This eliminates the variables x and u (where u is the proxy for 2^{x-y}). According to the specification in Lemma 6, this step transforms the linear program with divisions $\gamma'(q_x, u)$ produced as output of ElimVars into a pair consisting of a linear program with divisions $\gamma''(q_x)$ and linear-exponential program with divisions $\psi''(y, r_x)$.

Step IV (lines 15–19) eliminates q_x. In [CMS24], this is achieved by checking whether the linear program with divisions γ'' is satisfiable, and replacing it with $\top$ if so. In contrast, OptILEP instead works by trying "all" the solutions to γ''. More precisely, since γ'' is univariate, all of its inequalities can be rewritten in the form $\ell^* \leq q_x$ or $q_x \leq h^*$, with $\ell^*, h^* \in \mathbb{Z}$. Then, every solution to γ'' must lie in the interval $[\ell..h]$, where ℓ is either 0 or the largest such integer ℓ^*, and h is either ∞ or the smallest such integer h^*. If $h \neq \infty$, we can (non-deterministically) test all values in this interval (in the complexity proof we will show that both ℓ and h have polynomial bit size). If instead $h = \infty$, in the correctness proof we will show that either no optimal solution exists, or the objective function is independent of q_x. To cover the latter case, the algorithm updates h from ∞ to $\ell + \mathit{mod}(\gamma'')$ (line 17), ensuring that at least one solution of γ'' is explored. After eliminating q_x, the body of the loop terminates with a small "Step V" (lines 20–22), which prepares φ for the next iteration, and updates θ by removing 2^x (making 2^y the new leading exponential term).

In the above overview, we have not elaborated on the structure of the circuit C during the procedure. According to Lemma 14, C must be a $(k, 0)$-LEAC when ElimVars is called, and it evolves into a (k, k)-LEAC by the time this algorithm terminates. From this information, we know that C must become a $(k, 0)$-LEAC precisely when line 9 executes. Prior to this line, however, C contains no quotient variable, and instead has the structure given in the following definition:

Definition 3 (PreLEAC)**.** *Let $k \in [0..n]$. A k-PreLEAC C is a sequence of assignments*

$$x_{n-i} \leftarrow \frac{\sum_{j=i+1}^{k} a_{i,j} \cdot 2^{x_{n-j}}}{\mu} + r_{n-i} \qquad \text{for } i \text{ from } k-1 \text{ to } 0,$$

where every $a_{i,j}$ is in $\mathbb{Z}$, and the denominator μ is a positive integer.

We transfer the notation used for LEACs also to PreLEACs. In particular, we refer to the denominator μ as μ_C, postulating $\mu_C \coloneqq 1$ when $k = 0$ (note that C is the empty sequence in this case). We also define $\xi_C \coloneqq \sum\{|a_{i,j}| : i \in [0..k-1],\, j \in [i+1..k]\}$, and write $\mathrm{vars}(C)$ for the set of *free variables* of C, that is, x_{n-k} and the variables r_{n-i}. Lastly, for a variable x_m with $m \in [0..n]$, we write $C[x_m]$ for the function analogous to the one defined for LEACs on page 22.

It is easy to verify that, starting from C being a k-PreLEAC, line 9 of OptILEP produces a $(k, 0)$-LEAC. More interesting is the transformation that occurs in line 19, where the variables u and q_{n-k} are removed from the (k, k)-LEAC returned by OptILEP. The following remark describes this transformation, which yields a $(k+1)$-PreLEAC.

Remark 4 (From (k, k)-LEACs to $(k+1)$-PreLEACs)**.** *Let $k \in [0..n-1]$. Let C be a (k, k)-LEAC*

$$q_{n-i} \leftarrow \frac{b_{n-i} \cdot u + c_{n-i} \cdot q_{n-k} + d_{n-i}}{\eta} \qquad \text{for } i \text{ from } k-1 \text{ to } 0,$$

$$x_{n-i} \leftarrow \frac{\sum_{j=i+1}^{k} a_{i,j} \cdot 2^{x_{n-j}}}{\mu} + q_{n-i} \cdot 2^{x_{n-k-1}} + r_{n-i} \qquad \text{for } i \text{ from } k \text{ to } 0,$$

such that μ divides η. Let $\lambda = \frac{\eta}{\mu}$. By replacing u for $2^{x_{n-k} - x_{n-k-1}}$, assigning an integer v to q_{n-k}, and substituting the expressions for $q_{n-(k-1)}, \ldots, q_n$ into the expressions for $x_{n-k}, \ldots, x_n$, one trans-

forms C into the following $(k+1)$-PreLEAC C':

$$x_{n-k} \leftarrow \frac{\eta \cdot v \cdot 2^{x_{n-k-1}}}{\eta} + r_{n-k}$$
$$x_{n-i} \leftarrow \frac{(c_{n-i} \cdot v + d_{n-i}) \cdot 2^{x_{n-k-1}} + (\lambda \cdot a_{i,k} + b_{n-i}) \cdot 2^{x_{n-k}} + \sum_{j=i+1}^{k-1} \lambda \cdot a_{i,j} \cdot 2^{x_{n-j}}}{\eta} + r_{n-i}$$

for i from $k-1$ to 0.

Note that $\mathrm{vars}(C') = \mathrm{vars}(C) \setminus \{u, q_{n-k}\}$*, and when evaluating C and C' on a map $\nu\colon \mathrm{vars}(C) \to \mathbb{N}$ satisfying $\nu(q_{n-k}) = v$ and $\nu(u) = 2^{\nu(x_{n-k}) - \nu(x_{n-k-1})}$, the values taken by $x_{n-k}, \ldots, x_n$ coincide, that is, $C[x_{n-i}](\nu) = C'[x_{n-i}](\nu)$ for every $i \in [0..k]$.*

6.2 Correctness of OptILEP

The next proposition states that OptILEP correctly solves ILEP.

Proposition 4. *There is a non-deterministic procedure with the following specification:*

Input: *$\varphi(\boldsymbol{x})$: an integer linear-exponential program;*
w : a variable occurring in $\boldsymbol{x}$ (to be maximized).

Output of each branch (β): *σ_β : an ILESLP.*

The algorithm ensures that, if φ is satisfiable (resp., the problem of maximizing x subject to φ has a solution), then there a branch β such that $[\![\sigma_\beta]\!]$ is a solution (resp., an optimal solution) to φ.

Proof. Let $\varphi_0(\boldsymbol{x})$ be the linear-exponential program in input of OptILEP, and let n denote the number of variables in φ_0. In line 3, the algorithm guesses an ordering $2^{x_n} \geq \ldots \geq 2^{x_1} \geq 2^{x_0} = 1$, where $x_1, \ldots, x_n$ is a permutation of $\boldsymbol{x}$, and x_0 is a fresh variable. Let Θ be the set of all such ordering. Clearly, φ_0 is equivalent to $\bigvee_{\theta \in \Theta}(\varphi_0 \wedge \theta)$. To prove the proposition, it suffices to show, for a given $\theta \in \Theta$, that if $\varphi_0 \wedge \theta$ is satisfiable (resp., the problem of maximizing x subject to $\varphi_0 \wedge \theta$ has a solution), then, in a non-deterministic branch β, the algorithm returns an ILESLP σ_β such that $[\![\sigma_\beta]\!]$ is a solution (resp., an optimal solution) to $\varphi_0 \wedge \theta$. Therefore, throughout the proof we fix the ordering guessed in line 3 to be some $\theta_0 \coloneqq (2^{x_n} \geq \ldots \geq 2^{x_1} \geq 2^{x_0} = 1)$ from Θ. Moreover, let x_m (for some $m \in [1..n]$) denote the variable to be maximized, with respect to the order θ_0.

Throughout the proof, we write S_k for the set of all triples (C, θ, φ) that represent the state of OptILEP in any of its non-deterministic branches at the point when the execution reaches line 5 (the condition of the only **while** loop of the procedure) for the $(k+1)$th time. Since we fix the ordering θ_0, in particular $S_0 = \{(\emptyset, \theta_0, \varphi_0)\}$, where $\emptyset$ is the empty sequence assigned to C in line 1.

We show that the **while** loop of OptILEP enjoys the following loop invariant:

loop invariant. Let $k \in \mathbb{N}$. For every $(C, \theta, \varphi) \in S_k$:

- I_1. θ is the ordering $\theta_k \coloneqq (2^{x_{n-k}} \geq \cdots \geq 2^{x_1} \geq 2^{x_0} = 1)$.
- I_2. φ is a linear-exponential program with divisions featuring variables $\boldsymbol{y}_k \coloneqq (x_0, \ldots, x_{n-k})$ and *remainder variables* $\boldsymbol{r}_{k-1} \coloneqq (r_{n-(k-1)}, \ldots, r_n)$. The remainder variables do not occur in exponentials, and for every $i \in [0..k-1]$, φ implies $r_{n-i} < 2^{x_{n-k}}$.
- I_3. C is a k-PreLEAC of the form $(x_{n-(k-1)} \leftarrow \frac{\tau_{n-(k-1)}}{\mu_C} + r_{n-(k-1)}\,,\ \ldots\ ,\ x_n \leftarrow \frac{\tau_n}{\mu_C} + r_n)$.
- I_4. Given a solution $\nu\colon \mathrm{vars}(\varphi \wedge \theta) \to \mathbb{N}$ to $\varphi \wedge \theta$, the map $\nu + \sum_{i=0}^{k-1}[x_{n-i} \mapsto g_{n-i}]$, where g_{n-i} is the value taken by x_{n-i} when evaluating $C[x_m]$ on ν, is a solution to $\varphi_0 \wedge \theta_0$.

Moreover:

I_5. $\exists \boldsymbol{x}_{k-1}(\varphi_0 \wedge \theta_0)$ is equivalent to $\exists \boldsymbol{r}_{k-1} \bigvee_{(C,\theta,\varphi)\in S_k}(\varphi \wedge \theta)$, where $\boldsymbol{x}_{k-1} \coloneqq (x_{n-(k-1)}, \ldots, x_n)$.

I_6. If $\max\{\nu(x_m) : \nu \text{ is a solution to } \varphi_0 \wedge \theta_0\}$ exists, then it is equal to

$$\max\{C[x_m](\nu) : (C,\theta,\varphi) \in S_k \text{ and } \nu \text{ is a solution to } \varphi \wedge \theta\}.$$

(For $k \in [0..n]$, the loop invariant assumes a fixed set of variables $r_0, \ldots, r_n$ that are reused across different non-deterministic branches and across iterations of the **while** loop. This assumption is without loss of generality.)

Let $(C, \theta, \varphi) \in S_n$. The loop invariant implies that θ is $2^{x_0} = 1$. This causes the condition in the **while** loop to fail, terminating the loop. (In particular, we have $S_k = \emptyset$ for all $k > n$.) Moreover, from Item I_2, we conclude that the only solution to $\varphi \wedge \theta$ is the map assigning 0 to every variable. Accordingly, the algorithm checks whether this is indeed a solution (line 23), and if so, replaces all free variables in C with 0. Since C is a n-PreLEAC, this results in a sequence of the form

$$x_{n-i} \leftarrow \frac{a_{i,n} + \sum_{j=i+1}^{n-1} a_{i,j} \cdot 2^{x_{n-j}}}{\mu_C} \qquad \text{for } i \text{ from } n-1 \text{ to } 0,$$

which can easily be represented as an ILESLP. The proposition then follows from Items I_5 and I_6. Therefore, to complete the proof, it suffices to verify that the loop invariant holds.

The invariant is trivially true for S_0. (In particular, $\boldsymbol{r}_{-1}$ and $\boldsymbol{x}_{-1}$ are empty in this case, and formulae like $\exists \boldsymbol{x}_{-1}(\varphi_0 \wedge \theta_0)$ simplify to just $\varphi_0 \wedge \theta_0$.) Hence, let us assume that the loop invariant is true when the execution reaches line 5 for the $(k+1)$th time, with $k \in [0..n-1]$, and show that the invariant still holds when the algorithm comes back to this line for the $(k+2)$th time.

Consider $(C, \theta, \varphi) \in S_k$, and let T_{k+1} be the set of those triples from S_{k+1} that are constructed by (non-deterministially) running the body of the **while** loop starting from (C, θ, φ). More precisely, we will show that each triple in T_{k+1} satisfies Items I_1–I_4, and that moreover

I'_5. $\exists x_{n-k} \exists \boldsymbol{r}_{k-1}(\varphi \wedge \theta)$ is equivalent to $\exists \boldsymbol{r}_k \bigvee_{(C',\theta',\varphi')\in T_{k+1}}(\varphi' \wedge \theta')$.

I'_6. If $\max\{C[x_m](\nu) : \nu \text{ is a solution to } \varphi \wedge \theta\}$ exists, then it is equal to

$$\max\{C'[x_m](\nu) : (C',\theta',\varphi') \in T_{k+1} \text{ and } \nu \text{ is a solution to } \varphi' \wedge \theta'\}.$$

We divide the proof following the five steps identified in Section 6.1: Step I (lines 6–9), Step II (lines 10–13), Step III (line 14), Step IV (lines 15–19) and Step V (lines 20 –22).

Step I (lines 6–9). By Item I_1, θ is θ_k, and in it the leading and second-leading exponential terms are $2^{x_{n-k}}$ and $2^{x_{n-k-1}}$, respectively. Following the pseudocode of OptILEP, throughout the proof we write x for x_{n-k}, and y for x_{n-k-1}. According to Item I_2, within $\varphi(\boldsymbol{y}_k, \boldsymbol{r}_{k-1})$ the variables from $\boldsymbol{r}_{k-1}$ do not occur in exponentials, and moreover φ implies $\boldsymbol{r}_{k-1} < 2^x$. In line 8, OptILEP invokes the algorithm from Lemma 4 on the pair (φ, θ). Let E_1 be the set of pairs (γ, ψ) returned by this algorithm across its non-deterministic branches. By Lemma 4, each γ is a linear program with division in variables $\boldsymbol{q}_k \coloneqq (q_{n-k}, \ldots, q_n)$ (called *quotient variables*) and u, whereas each ψ is a linear-exponential program with divisions in variables $\boldsymbol{y}_{k+1}$ and $\boldsymbol{r'} \coloneqq (r'_{n-k}, \ldots, r'_n)$, the latter being fresh *remainder variables* not occurring in exponentials, and such that ψ implies $\boldsymbol{r'} < 2^y$.

Again as in the pseudocode, we often write q_x for q_{n-k}, and r_x for r'_{n-k}. The system

$$\begin{bmatrix} x \\ r_{n-(k-1)} \\ \vdots \\ r_n \end{bmatrix} = \begin{bmatrix} q_x \\ q_{n-(k-1)} \\ \vdots \\ q_n \end{bmatrix} \cdot 2^y + \begin{bmatrix} r_x \\ r'_{n-(k-1)} \\ \vdots \\ r'_n \end{bmatrix}, \tag{21}$$

yields a one-to-one correspondence between the solutions of $\varphi \wedge \theta$ and those of $\bigvee_{(\gamma,\psi)\in E_1} \Phi(\gamma,\psi)$, where $\Phi(\gamma,\psi) \coloneqq \big(\gamma \wedge \psi \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x) \wedge \theta\big)$. This correspondence is the identity for the variables these two formulae share (i.e., the variables in $\boldsymbol{y}_k$).

In line 9, C is updated following the system described in Equation (21). Let C' be the resulting sequence. By Item I_3, C is a k-PreLEAC, and therefore C' takes the form:

$$x_{n-i} \leftarrow \frac{\tau_{n-i}}{\mu} + q_{n-i} \cdot 2^y + r'_{n-i} \qquad \text{for } i \text{ from } k \text{ to } 0, \tag{22}$$

where each τ_{n-i} is a term of the form $\sum_{j=i+1}^{k} a_{i,j} \cdot 2^{x_{n-j}}$, with each $a_{i,j}$ in $\mathbb{Z}$, and μ is a positive integer. In other words, C' is a $(k,0)$-LEAC. The claim below follows directly from the one-to-one correspondence given by Equation (21).

Claim 13. *The following properties hold:*

1. *The formulae* $\exists \boldsymbol{r}_{k-1}(\varphi \wedge \theta)$ *and* $\exists \boldsymbol{q}_k \exists u \exists \boldsymbol{r}' \bigvee_{(\gamma,\psi)\in E_1} \Phi(\gamma,\psi)$ *are equivalent.*
2. *Consider* $(\gamma,\psi) \in E_1$*, and let* $\nu\colon \mathrm{vars}(\Phi(\gamma,\psi)) \to \mathbb{N}$ *be a solution to* $\Phi(\gamma,\psi)$*. Then, the map* $\nu + \sum_{i=0}^{k-1}[r_{n-i} \mapsto \nu(q_{n-i}) \cdot 2^{\nu(y)} + \nu(r'_{n-i})]$ *is a solution to* $\varphi \wedge \theta$*.*
3. *If* $\max\{C[x_m](\nu) : \nu$ *is a solution to* $\varphi \wedge \theta\}$ *exists, it is equal to*

$$\max\{C'[x_m](\nu) : \nu \text{ is a solution to } \Phi(\gamma,\psi), \text{ for some } (\gamma,\psi) \in E_1\}.$$

Step II (lines 10–13). Fix $(\gamma,\psi) \in E_1$. In a nutshell, Step II removes the quotient variables $\boldsymbol{q}_{k-1} = (q_{n-(k-1)}, \ldots, q_n)$ from γ. Let $\widetilde{\gamma}$ be the formula obtained from γ by performing the updates in lines 10 and 11; $\widetilde{\gamma}$ and γ are equivalent, as all variables range over $\mathbb{N}$ and $\mu_{C'} \geq 1$. Let ψ' be the formula in line 12, that is, $\psi' \coloneqq (\psi \wedge \theta \wedge (x = q_x \cdot 2^y + r_x) \wedge (u = 2^{x-y}))$. By definition, the three formulae $\langle \widetilde{\gamma}\,;\psi' \rangle$, $\langle \gamma\,;\psi' \rangle$ and $\Phi(\gamma,\psi)$ are equivalent. We show that $(C', \langle \widetilde{\gamma}\,;\psi' \rangle)$ is in $\mathcal{I}_k^0$; and hence that the call to ElimVars performed in line 13 adheres to the specification of this algorithm (from Lemma 14). Below, we refer to the Items (i)–(iii) characterizing $\mathcal{I}_k^0$ (page 23):

- *Item (i):* We have already seen that C' is a $(k,0)$-LEAC.
- *Item (ii):* By definition, $\widetilde{\gamma}$ is a linear exponential program with divisions, in variables u and $\boldsymbol{q}_k$, and in which all inequalities and equalities are such that the coefficients of the variables $\boldsymbol{q}_k$ are divisible by $\mu_{C'}$. Moreover, $\widetilde{\gamma}$ contains an inequality $\mu_{C'} \cdot q \geq 0$ for every q in $\boldsymbol{q}_k$. Lastly, the formula ψ' trivially satisfies the conditions specified in Item (ii).
- *Item (iii):* We need to show that $\langle \widetilde{\gamma}\,;\psi' \rangle$ implies the formula $\Psi(C')$ given by

$$\mathbf{0} \leq \boldsymbol{r}' < 2^y \wedge \mathbf{0} \leq \boldsymbol{q}_k \cdot 2^y + \boldsymbol{r}' < 2^x \wedge \exists \boldsymbol{x}_{k-1}\big(\theta_0 \wedge \textstyle\bigwedge_{i=0}^{k}(x_{n-i} = \rho_{n-i})\big),$$

  where ρ_{n-i} is the expression assigned to x_{n-i} in C' (following Equation (22)).

Consider a solution $\nu\colon \mathrm{vars}(\langle\widetilde{\gamma}\,;\psi'\rangle) \to \mathbb{N}$ to $\langle\widetilde{\gamma}\,;\psi'\rangle$. This is also a solution to $\Phi(\gamma,\psi)$. By definition, ψ implies $\mathbf{0} \leq \boldsymbol{r}' < 2^y$. From the one-to-one correspondence given by Equation (21), the map $\nu' \coloneqq \nu + \sum_{i=0}^{k-1}[r_{n-i} \mapsto \nu(q_{n-i})\cdot 2^{\nu(y)} + \nu(r'_{n-i})]$ is a solution to $\varphi \wedge \theta$. By Item I_2, φ implies $\boldsymbol{r}_{k-1} < 2^x$. Therefore, $\langle\widetilde{\gamma}\,;\psi'\rangle$ implies $\mathbf{0} \leq \boldsymbol{q}_k \cdot 2^y + \boldsymbol{r}' < 2^x$.

Lastly, we see that $\langle\widetilde{\gamma};\psi'\rangle$ also implies $\exists \boldsymbol{x}_{k-1}\big(\theta_0 \wedge \bigwedge_{i=0}^{k}(x_{n-i} = \rho_{n-i})\big)$. Indeed, by Item I_4, the map $\nu'' \coloneqq \nu' + \sum_{i=0}^{k-1}[x_{n-i} \mapsto \frac{\nu'(\tau_{n-i})}{\mu_C} + \nu'(r_{n-i})]$ is a solution to $\varphi_0 \wedge \theta_0$. Together with the fact that ψ' implies $x = q_x \cdot 2^y + r_x$, this means that $\langle\widetilde{\gamma}\,;\psi'\rangle$ implies $\exists \boldsymbol{x}_{k-1}\big(\theta_0 \wedge \bigwedge_{i=0}^{k}(x_{n-i} = \frac{\tau_{n-i}}{\mu_C} + q_{n-i}\cdot 2^y + r'_{n-i})\big)$. By definition, $\rho_{n-i} = (\frac{\tau_{n-i}}{\mu_C} + q_{n-i}\cdot 2^y + r'_{n-i})$.

Therefore, $(C', \langle\widetilde{\gamma}\,;\psi'\rangle) \in \mathcal{I}_k^0$. In line 13, OptILEP calls ElimVars, eliminating the quotient variables $\boldsymbol{q}_{k-1}$. Let us denote by $E_2(\gamma,\psi)$ the set of all triples (C'',γ',ψ') such that $(C''[x_m], \langle\gamma'\,;\psi'\rangle)$ is a pair returned by a non-deterministic execution of ElimVars with as input the formulae computed from γ and ψ in lines 10–13. Then, by direct application of Lemma 14, we obtain:

Claim 14. *Let $(\gamma,\psi) \in E_1$. The following properties hold:*

1. *The formulae $\exists \boldsymbol{q}_{k-1}\Phi(\gamma,\psi)$ and $\bigvee_{(C'',\gamma',\psi')\in E_2(\gamma,\psi)}\langle\gamma'\,;\psi'\rangle$ are equivalent.*
2. *Consider $(C'',\gamma',\psi') \in E_2(\gamma,\psi)$. Let $q_{n-(k-1)} \leftarrow \frac{\tau''_{n-(k-1)}}{\eta}, \ldots, q_n \leftarrow \frac{\tau''_n}{\eta}$ be the assignments to the variables $\boldsymbol{q}_{k-1}$ occurring in C''. Let $\nu\colon \mathrm{vars}(\langle\gamma'\,;\psi'\rangle) \to \mathbb{N}$ be a solution to $\langle\gamma'\,;\psi'\rangle$. Then, the map $\nu + \sum_{i=0}^{k-1}[q_{n-i} \mapsto \frac{\nu(\tau''_{n-i})}{\eta}]$ is a solution to $\Phi(\gamma,\psi)$.*
3. *If $\max\{C'[x_m](\nu) : \nu$ is a solution to $\Phi(\gamma,\psi)\}$ exists, then it is equal to*

$$\max\{C''[x_m](\nu) : \nu \text{ is a solution to } \langle\gamma'\,;\psi'\rangle, \text{ for some } (C'',\gamma',\psi') \in E_2(\gamma,\psi)\}.$$

4. *For every $(C'',\gamma',\psi') \in E_2(\gamma,\psi)$, the pair $(C'',\langle\gamma'\,;\psi'\rangle)$ belongs to $\mathcal{I}_k^k$.*

Step III (line 14). Let $(C'',\gamma',\psi') \in E_2(\gamma,\psi)$, for some $(\gamma,\psi) \in E_1$. Step III eliminates x and u from γ' by applying the algorithm from Lemma 6, which non-deterministically returns a pair (γ'',ψ''). We write $E_3(\gamma')$ for the set of all such resulting pairs. By Claim 14.4, γ' is a linear program with divisions in variables q_x and u. So, by Lemma 6, γ'' is a linear program with divisions in the single variable q_x, and ψ'' is a linear-exponential program with divisions in the variables y and r_x. Moreover, the equation $x = q_x\cdot 2^y + r_x$ yields a one-to-one correspondence between the solutions of $\gamma' \wedge (u = 2^{x-y}) \wedge (x = q_x\cdot 2^y + r_x)$ and those of $\bigvee_{(\gamma'',\psi'')\in E_3(\gamma')}(\gamma'' \wedge \psi'')$. This correspondence is the identity for the variables these two formulae share (that is, y, q_x and r_x). Roughly speaking, this one-to-one correspondence allows us to remove x and u without changing the set of solutions.

Recall that $\psi' \coloneqq (\psi \wedge \theta \wedge (x = q_x\cdot 2^y + r_x) \wedge (u = 2^{x-y}))$, and observe that $\theta \wedge (u = 2^{x-y})$ is equal to $\theta_{k+1} \wedge (u = 2^{x-y})$, because $u = 2^{x-y}$ implies $2^x \geq 2^y$ (as u ranges over $\mathbb{N}$). Then, the claim below follows immediately from the one-to-one correspondence given by the equation $x = q_x\cdot 2^y + r_x$.

Claim 15. *Let $(\gamma,\psi) \in E_1$ and $(C'',\gamma',\psi') \in E_2(\gamma,\psi)$. The following properties hold:*

1. *The formulae $\exists x \exists u \langle\gamma'\,;\psi'\rangle$ and $\bigvee_{(\gamma'',\psi'')\in E_3(\gamma')}(\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1})$ are equivalent.*
2. *Let $(\gamma'',\psi'') \in E_3(\gamma')$, and $\nu\colon \mathrm{vars}(\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1}) \to \mathbb{N}$ be a solution to $\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1}$. Define $g \coloneqq \nu(q_x)\cdot 2^{\nu(y)} + \nu(r_x)$. The map $\nu + [x \mapsto g] + [u \mapsto 2^{g-\nu(y)}]$ is a solution to $\langle\gamma'\,;\psi'\rangle$.*

3. *If* $\max\{C''[x_m](\nu) : \nu$ *is a solution to* $\langle \gamma' ; \psi' \rangle\}$ *exists, it is equal to*

$$\max\{C''[x_m](\nu) : \text{for some } (\gamma'', \psi'') \in E_3(\gamma'), \text{ the map } \nu \text{ is a solution to the formula } \gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1} \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x)\}.$$

(In Item 3, the constraints $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$ are added to handle the fact that u still appears as a free variable of C''. This discrepancy is resolved in line 19.)

Step IV (lines 15–19). Let $(C'', \gamma', \psi') \in E_2(\gamma, \psi)$ and $(\gamma'', \psi'') \in E_3(\gamma')$, for some $(\gamma, \psi) \in E_1$. Step IV removes the quotient variable q_x from the linear program with divisions γ'', and translates the (k, k)-LEAC into a $(k+1)$-PreLEAC. Let us treat each equality $\tau = 0$ in γ'' as a conjunction of two inequalities: $\tau \leq 0 \wedge -\tau \leq 0$. Since γ'' contains only the variable q_x, every inequality in it is either of the form $a \leq b \cdot q_x$ or $b \cdot q_x \leq a$, with b non-negative. Let us update γ'' by rewriting these as $\lceil \frac{a}{b} \rceil \leq q_x$ and $q_x \leq \lfloor \frac{a}{b} \rfloor$, respectively. Line 15 computes the greatest non-negative integer ℓ such that $\ell \leq q_x$ occurs in γ'', while Line 16 computes the smallest integer h such that $q_x \leq h$ occurs in γ''. By default, ℓ and h are initialized as 0 and ∞, respectively, so in particular we always have $\ell \geq 0$. If $h = \infty$, the algorithm updates it to $\ell + mod(\gamma'')$ in line 17, ensuring that at least one solution to γ'' is explored (if one exists). Let us write $B(\gamma'')$ for the set $\{v \in [\ell..h] : \gamma''(v) \text{ holds}\}$. Step IV concludes by guessing $v \in B(\gamma'')$ (line 18), to then translating C'' into a $(k+1)$-PreLEAC following Remark 4. If $B(\gamma'')$ is empty then γ'' is unsatisfiable; in this case the **guess** instruction fails, and the non-deterministic branch of the algorithm rejects.

Let us write $E_4(C'', \gamma'')$ for the set of all circuits obtained from C'' when running line 19, with respect to some $v \in B(\gamma'')$. We show the following claim (whose proof clarifies why it is sufficient to restrict q_x to values in $[\ell..h]$ in order to explore optimal solutions).

Claim 16. *Let* $(\gamma, \psi) \in E_1$, $(C'', \gamma', \psi') \in E_2(\gamma, \psi)$ *and* $(\gamma'', \psi'') \in E_3(\gamma')$. *We have:*

1. *If* $B(\gamma'')$ *is non-empty, then* $\exists q_x(\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1})$ *is equivalent to* $\psi'' \wedge \psi \wedge \theta_{k+1}$.
2. *Let* $v \in B(\gamma'')$, *and let* $\nu \colon \mathrm{vars}(\psi'' \wedge \psi \wedge \theta_{k+1}) \to \mathbb{N}$ *be a solution to* $\psi'' \wedge \psi \wedge \theta_{k+1}$. *Then, the map* $\nu + [q_x \mapsto v]$ *is a solution to* $\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1}$.
3. *If* $M \coloneqq \max\{C''[x_m](\nu) :$ *for some* $(\gamma'', \psi'') \in E_3(\gamma')$, *the map* ν *is a solution to the formula* $\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1} \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x)\}$ *exists, it is equal to*

$$\max\{C^*[x_m](\nu) : \nu \text{ is a solution to } \psi'' \wedge \psi \wedge \theta_{k+1}, \text{ for some } C^* \in E_4(C'', \gamma'')\}.$$

Proof. In the formula $\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1}$, the variable q_x only occurs in γ''. Then, the left-to-right direction of Item 1 follows from Corollary 2, whereas the right-to-left direction holds from Item 2, which in turn follows directly from the definition of $B(\gamma'')$.

We prove Item 3. Let $\Psi \coloneqq (\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1} \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x))$. We divide the proof depending on the variable x_m we are maximizing:

case: $m < n - k$**.** Suppose M exists. The circuit C'' does not feature an assignment to the variable x_m, and the same is true for every $C^* \in E_4(C'', \gamma'')$. For both C'' and C^*, given a map ν from their free variables plus x_m to $\mathbb{N}$, we have $C''[x_m](\nu) = \nu(x_m) = C^*[x_m](\nu)$. Let ν be a solution to Ψ. Since ν is also a solution to $\psi'' \wedge \psi \wedge \theta_{k+1}$, we have $\max\{C^*[x_m](\nu) : \nu$ is a solution to $\psi'' \wedge \psi \wedge \theta_{k+1}$, for some $C^* \in E_4(C'', \gamma'')\} \geq M$.

Ad absurdum, assume $C^*[x_m](\nu^*) > M$, for some solution $\nu^*\colon \mathrm{vars}(\psi'' \wedge \psi \wedge \theta_{k+1}) \to \mathbb{N}$ to $\psi'' \wedge \psi \wedge \theta_{k+1}$, and $C^* \in E_4(C'', \gamma'')$. By definition, C^* is constructed from C'' by replacing q_x with some $v \in B(\gamma'')$, and u with 2^{x-y}. By Item 2, $\nu' \coloneqq \nu^* + [q_x \mapsto v]$ is a solution to $\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1}$. Define $g \coloneqq \nu'(q_x) \cdot 2^{\nu'(y)} + \nu'(r_x)$. By Claim 15.2, the map $\nu'' \coloneqq \nu' + [x \mapsto g] + [u \mapsto 2^{g-\nu'(y)}]$ is a solution to $\langle \gamma' \,;\, \psi' \rangle$. Since ψ' implies $u = 2^{x-y} \wedge x = q_x \cdot 2^y + r_x$, we conclude that ν'' is a solution to Ψ. We have $C''[x_m](\nu'') = \nu''(x_m) = \nu^*(x_m) = C^*[x_m](\nu^*) > M$, contradicting the fact that M is maximal. Therefore, Item 3 holds.

case: $m \geq n - k$**.** First, let us consider the case where $h \neq \infty$ when defined in line 16. Then, all the solutions to γ'' lie in the interval $[\ell..h]$. The set $E_4(C'', \gamma'')$ is constructed to consider all these solutions, and therefore Item 3 follows.

Suppose instead that $h = \infty$ in line 16. In this case, we establish Item 3 by showing that M does not exist. Consider an arbitrary solution ν to Ψ. Let $p \coloneqq \mathit{mod}(\gamma'')$. Since $h = \infty$, we note that increasing $\nu(q_x)$ by any positive multiple of p, and increasing $\nu(x)$ and $\nu(u)$ accordingly, still yields a solution to Ψ. More precisely, if we want to increase the $\nu(q_x)$ by $i \cdot p$, with $i \in \mathbb{N}$, the resulting map is $\nu + [q_x \mapsto i \cdot p] + [x \mapsto i \cdot p \cdot 2^{\nu(y)}] + [u \mapsto (2^{i \cdot p \cdot 2^{\nu(y)}} - 1) \cdot 2^{\nu(x) - \nu(y)}]$. By definition, C'' contains the assignment $x \leftarrow q_x \cdot 2^y + r_x$. Therefore, arbitrarily increasing $\nu(q_x)$ results in arbitrarily large values that x evaluates to in $C''[x_m]$.

Let ν be a solution to Ψ. To complete the proof, it suffices to show that the value taken by x when evaluating $C''[x_m]$ on ν is at most $C''[x_m](\nu)$, in other words, that $C''[x_{n-k}](\nu) \leq C''[x_m](\nu)$. From Claim 15.2, ν is also a solution to $\langle \gamma' \,;\, \psi' \rangle$. By Claim 14.4, $(C'', \langle \gamma' \,;\, \psi' \rangle)$ belongs to $\mathcal{I}_k^k$. By definition of $\mathcal{I}_k^k$, $\langle \gamma' \,;\, \psi' \rangle$ implies $\exists \boldsymbol{q}_{k-1} \exists \boldsymbol{x}_{k-1} \big(\theta_0 \wedge \bigwedge_{i=1}^{t} (y_i = \rho_i)\big)$, where $(y_1 \leftarrow \rho_1, \ldots, y_t \leftarrow \rho_t) = C''$. The variables in $\boldsymbol{q}_{k-1}$ and $\boldsymbol{x}_{k-1}$ are all among $y_1, \ldots, y_t$, meaning that there is exactly one evaluation for these variables for which $\theta_0 \wedge \bigwedge_{i=1}^{t} (y_i = \rho_i)$ is satisfied: the values that these variables take when $C''[x_m]$ is evaluated on ν. Since $m \geq n - k$, the ordering θ_0 implies $x_{n-k} \leq x_m$. Hence, when evaluating $C''[x_m]$ on ν, the value taken by x is at most $C''[x_m](\nu)$, as required. □

Step V (lines 20–22). These lines simply prepare the linear-exponential system for the next loop iteration. Let $(\gamma, \psi) \in E_1$, $(C'', \gamma', \psi') \in E_2(\gamma, \psi)$, $(\gamma'', \psi'') \in E_3(\gamma')$ and $C^* \in E_4(C'', \gamma'')$. Line 20 sets $\boldsymbol{r}'$ as the remainder variables for the next iterations of the loop (in this proof, $\boldsymbol{r}_k$). Line 21 sets $\varphi^* \coloneqq \psi \wedge \psi''$ as the formula for the next iteration, whereas line 22 updates θ to θ_{k+1}. This concludes the body of the **while** loop, and T_{k+1} is the set of all possible triples $(C^*, \theta_{k+1}, \varphi^*)$.

Let us now complete the proof by showing that all items in the loop invariant are satisfied.

- *Item I_1:* θ_{k+1} is indeed the ordering required by this item.
- *Item I_2:* In $\varphi^* = \psi \wedge \psi''$, both ψ and ψ'' are linear-exponential programs with divisions in variables $\boldsymbol{y}_{k+1}$ and $\boldsymbol{r}'$. Moreover, ψ (which was defined in Step I), implies $\boldsymbol{r}' < 2^y$, as required.
- *Item I_3:* This follows directly from the manipulation performed in line 19 to construct C^*, recalling that C'' is a (k, k)-LEAC. Below, let $C^* = (x_{n-k} \leftarrow \frac{\tau^*_{n-k}}{\mu^*} + r'_{n-k}\,, \ldots\,, x_n \leftarrow \frac{\tau^*_n}{\mu^*} + r'_n)$.
- *Item I_4:* We must show that, given a solution $\nu\colon \mathrm{vars}(\varphi^* \wedge \theta_{k+1}) \to \mathbb{N}$ to $\varphi^* \wedge \theta_{k+1}$, the map $\nu + \sum_{i=0}^{k} [x_{n-i} \mapsto g_{n-i}]$, where g_{n-i} is the value taken by x_{n-i} when evaluating $C^*[x_m]$ on ν, is a solution to $\varphi_0 \wedge \theta_0$. In a nutshell, this is shown by appealing to the second Items in

Claims 13–16, to then apply the induction hypothesis. Indeed, observe that starting from ν, these Items construct a solution ν^* for $\varphi \wedge \theta$. Recall that the initial circuit C is a k-PreLEAC

$$x_{n-i} \leftarrow \frac{\tau_{n-i}}{\mu} + r_{n-i} \qquad \text{for } i \text{ from } k-1 \text{ to } 0.$$

This circuit is then manipulated into the circuit C'

$$\begin{aligned} x &\leftarrow q_x \cdot 2^y + r_x, \\ x_{n-i} &\leftarrow \frac{\tau_{n-i}}{\mu} + q_{n-i} \cdot 2^y + r'_{n-i} \qquad \text{for } i \text{ from } k-1 \text{ to } 0, \end{aligned}$$

and C'' is constructed from C' by adding assignments $q_{n-i} \leftarrow \frac{\tau''_{n-i}(u,q_x)}{\eta}$ for every $i \in [0..k-1]$. Lastly, C^* is essentially obtained from C'' by replacing q_x the integer v from line 18 , and u with 2^{x-y}. By induction hypothesis, the map obtained from ν^* by adding $\sum_{i=0}^{k-1}[x_{n-i} \mapsto t_{n-i}]$, where t_{n-i} is the value taken by x_{n-i} when evaluating $C[x_m]$ on ν^* is a solution to $\varphi_0 \wedge \theta_0$. Then, because of the updates required to obtain C^* from C, it suffices to show that

$$\begin{aligned} \nu^*(x) &= v \cdot 2^{\nu(y)} + \nu(r_x), \\ \nu^*(r_{n-i}) &= \frac{\tau''_{n-i}(2^{\nu^*(x_n)-\nu(y)}, v)}{\eta} \cdot 2^{\nu(y)} + \nu(r'_{n-i}) \qquad \text{for } i \text{ from } k-1 \text{ to } 0. \end{aligned}$$

This follows directly from the identities below:

$$\begin{aligned} &\nu' = \nu + [q_x \mapsto v] && \text{⟅Claim 16.2⟆} \\ &\nu'' = \nu' + [x \mapsto g] + [u \mapsto 2^{g-\nu(y)}], \quad \text{where } g \coloneqq v \cdot 2^{\nu(y)} + \nu(r_x) && \text{⟅Claim 15.2⟆} \\ &\nu''' = \nu'' + \textstyle\sum_{i=0}^{k-1}[q_{n-i} \mapsto \frac{\nu''(\tau''_{n-i})}{\eta}] && \text{⟅Claim 14.2⟆} \\ &\nu^* = \nu''' + \textstyle\sum_{i=0}^{k-1}[r_{n-i} \mapsto \nu'''(q_{n-i}) \cdot 2^{\nu(y)} + \nu(r'_{n-i})] && \text{⟅Claim 13.2⟆} \end{aligned}$$

Let us now show Item I'_5; then Item I_5 follows directly from the induction hypothesis:

$$\begin{aligned} & \exists x \exists \boldsymbol{r}_{k-1}(\varphi \wedge \theta) \\ \iff & \exists x \exists \boldsymbol{q}_k \exists u \exists \boldsymbol{r'} \textstyle\bigvee_{(\gamma,\psi)\in E_1} \Phi(\gamma,\psi) && \text{⟅Claim 13.1⟆} \\ \iff & \exists x \exists q_x \exists u \exists \boldsymbol{r'} \textstyle\bigvee_{(\gamma,\psi)\in E_1,\,(C'',\gamma',\psi')\in E_2(\gamma,\psi)} \langle \gamma' \,;\, \psi' \rangle && \text{⟅Claim 14.1⟆} \\ \iff & \exists q_x \exists \boldsymbol{r'} \textstyle\bigvee_{(\gamma,\psi)\in E_1,\,(C'',\gamma',\psi')\in E_2(\gamma,\psi),\,(\gamma'',\psi'')\in E_3(\gamma')} (\gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1}) && \text{⟅Claim 15.1⟆} \\ \iff & \exists \boldsymbol{r'} \textstyle\bigvee_{(\gamma,\psi)\in E_1,\,(C'',\gamma',\psi')\in E_2(\gamma,\psi),\,(\gamma'',\psi'')\in E_3(\gamma')\,\text{s.t.}\,B(\gamma'')\neq\emptyset} (\psi'' \wedge \psi \wedge \theta_{k+1}) && \text{⟅Claim 16.1⟆} \\ \iff & \exists \boldsymbol{r'} \textstyle\bigvee_{(C^*,\theta_{k+1},\varphi^*)\in T_{k+1}} (\varphi^* \wedge \theta_{k+1}) && \text{⟅def. of } T_{k+1}\text{⟆} \end{aligned}$$

Lastly, we show Item I'_6, which implies Item I_6 by induction hypothesis. Suppose that $M \coloneqq \max\{C[x_m](\nu) : \nu \text{ is a solution to } \varphi \wedge \theta\}$ exists. Then,

$$\begin{aligned} M &= \max\{C'[x_m](\nu) : \nu \text{ is a solution to } \Phi(\gamma,\psi), \text{ for some } (\gamma,\psi) \in E_1\} && \text{⟅Claim 13.3⟆} \\ &= \max\{C''[x_m](\nu) : \nu \text{ is a solution to } \langle \gamma' \,;\, \psi' \rangle, \text{ for some } (\gamma,\psi) \in E_1, \\ &\qquad\qquad \text{and } (C'',\gamma',\psi') \in E_2(\gamma,\psi)\} && \text{⟅Claim 14.3⟆} \\ &= \max\{C''[x_m](\nu) : \nu \text{ is a solution to } \gamma'' \wedge \psi'' \wedge \psi \wedge \theta_{k+1} \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x) \\ &\qquad\qquad \text{for some } (\gamma,\psi) \in E_1,\ (C'',\gamma',\psi') \in E_2(\gamma,\psi), \text{ and } (\gamma'',\psi'') \in E_3(\gamma')\} && \text{⟅Claim 15.3⟆} \end{aligned}$$

$$= \max\{C^*[x_m](\nu) : \nu \text{ is a solution to } \psi'' \wedge \psi \wedge \theta_{k+1}, \text{ for some } (\gamma,\psi) \in E_1,$$
$$(C'',\gamma',\psi') \in E_2(\gamma,\psi),\ (\gamma'',\psi'') \in E_3(\gamma'), \text{ and } C^* \in E_4(C'',\gamma'')\} \quad \text{⟅Claim 16.3⟆}$$
$$= \max\{C^*[x_m](\nu) : \nu \text{ is a solution to } \varphi^* \wedge \theta_{k+1}, \text{ for some } (C^*,\theta_{k+1},\varphi^*) \in T_{k+1}\} \quad \text{⟅def. of } T_{k+1}\text{⟆}$$

Therefore, the loop invariant holds, completing the proof of the proposition. □

6.3 Complexity of OptILEP

We now provide the complexity analysis of OptILEP, which requires tracking several parameters of linear-exponential systems. For a linear-exponential program with divisibilities φ, we track:

- The parameters $\#\varphi$, $\|\varphi\|_1$ and $mod(\varphi)$, defined in the preliminaries (page 14).
- The *linear norm* $\|\varphi\|_{\mathfrak{L}} \coloneqq \max\{\|\tau\|_{\mathfrak{L}} : \tau \text{ is a term appearing in an equality or inequality of } \varphi\}$. Given a linear-exponential term $\tau = \sum_{i=1}^{n} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j})\right) + d$, we define its linear norm as $\|\tau\|_{\mathfrak{L}} \coloneqq \max\{|a_i|, |c_{i,j}| : i,j \in [1..n]\}$.
- Consider an ordering of exponentiated variables $\theta \coloneqq (\theta(\boldsymbol{x}) \coloneqq 2^{x_n} \geq 2^{x_{n-1}} \geq \cdots \geq 2^{x_0} = 1)$. Let $\boldsymbol{r}$ be the variables from φ that are not in $\boldsymbol{x}$. We track the set of the *least significant terms*

$$lst(\varphi,\theta) \coloneqq \big\{ \pm\rho : \rho \text{ is the least significant part of a term } \tau \text{ appearing in an equality or inequality } \tau \sim 0 \text{ of } \varphi, \text{ with respect to } \theta \big\}.$$

  The *least significant part* of a term $a \cdot 2^{x_n} + b \cdot x_n + \tau'(x_0,\ldots,x_{n-1},\boldsymbol{r})$ with respect to θ is defined as the term $b \cdot x_n + \tau'$.

For a k-PreLEAC C, we track the growth of the parameters μ_C and ξ_C.

Remark 5. *From the above parameters one can bound the sizes of φ and C: the size of $\varphi(\boldsymbol{x})$ is in poly$(\#\varphi, \#\boldsymbol{x}, \log\|\varphi\|_1, \log mod(\varphi))$, and the size of a k-PreLEAC C is in poly$(k, \log\mu_C, \log\xi_C)$.*

The complexity of Steps I and III from [CMS24]. For the complexity analysis of lines 8 and 14 of OptILEP, which correspond to Steps I and III of [CMS24], we refer directly to the analysis carried out in [CMS24]. (We remind the reader that Appendix B gives more information on these two steps). This analysis is reported in the following two lemmas.

Lemma 22 [CMS24]**.** *The algorithm from Lemma 4 (Step I) runs in non-deterministic polynomial time. Consider its execution on an input (θ,φ) where $\theta(\boldsymbol{x})$ is an ordering of exponentiated variables and $\varphi(\boldsymbol{x},\boldsymbol{r})$ is a linear exponential program with divisions. In each non-deterministic branch β, the algorithm returns a pair (γ,ψ), where $\gamma(q_x,\boldsymbol{q},u)$ is a linear program with divisions and $\psi(\boldsymbol{y},r_x,\boldsymbol{r'})$ a linear-exponential program with divisions, such that (for every $\ell,s,a,c,d \geq 1$):*

$$\textit{if } \begin{cases} \#lst(\varphi,\theta) & \leq \ell \\ \#\varphi & \leq s \\ \|\varphi\|_{\mathfrak{L}} & \leq a \\ \|\varphi\|_1 & \leq c \\ mod(\varphi) & \mid d \end{cases} \textit{ then } \begin{cases} \#lst(\psi,\theta') & \leq \ell + 2\cdot k \\ \#\psi & \leq s + 6\cdot k + 2\cdot\ell \\ \|\psi\|_{\mathfrak{L}} & \leq 3\cdot a \\ \|\psi\|_1 & \leq 4\cdot c + 5 \\ mod(\psi) & \mid d \end{cases} \textit{ and } \begin{cases} \#\gamma & \leq s + 2\cdot k \\ \|\gamma[2^u \,/\, u]\|_{\mathfrak{L}} & \leq 3\cdot a \\ \|\gamma\|_1 & \leq 2\cdot c + 3 \\ mod(\gamma) & \mid d \end{cases}$$

where θ' is the ordering obtained from θ by removing its largest term 2^x, and $k \coloneqq 1 + \#\boldsymbol{r}$.

Below, ϕ denotes Euler's totient function. Recall that given a positive integer a,

$$\phi(a) \coloneqq \prod_{i=1}^{k} \left((p_i - 1) \cdot p_i^{e_i - 1}\right), \text{ where } p_1^{e_1} \cdots p_k^{e_k} \text{ is the prime factorization of } a. \tag{23}$$

Lemma 23 [CMS24]**.** *The algorithm from Lemma 6 (Step III) runs in non-deterministic polynomial time. Consider its execution on an input linear program with divisions γ'. In each non-deterministic branch β, the algorithm returns a pair (γ'', ψ''), where γ'' is a linear program with divisions and ψ'' is a linear-exponential program with divisions, such that (for every $s, a, c, d \geq 1$):*

$$\textit{if} \begin{cases} \#\gamma' & \leq s \\ \|\gamma'[2^u \,/\, u]\|_{\mathfrak{L}} & \leq a \\ \|\gamma'\|_1 & \leq c \\ \textit{mod}(\gamma') & \mid d \end{cases} \textit{then} \begin{cases} \#\gamma'' & \leq s + 2 \\ \|\gamma''\|_{\mathfrak{L}} & \leq a \\ \|\gamma''\|_1 & \leq \max(2^5 c^3, c \cdot d) \\ \textit{mod}(\gamma'') & \mid \operatorname{lcm}(d, \phi(d)) \end{cases} \textit{and} \begin{cases} \#\psi'' & \leq 3 \\ \|\psi''\|_{\mathfrak{L}} & \leq 1 \\ \|\psi''\|_1 & \leq 12 + 4 \cdot \log(\max(c, d)) \\ \textit{mod}(\psi'') & \mid \phi(d) \end{cases}$$

The complexity of performing one iteration of the main loop is given in the next lemma:

Lemma 24. *Consider the execution of Algorithm 6 on an linear-exponential program $\varphi(x_1, \ldots, x_n)$, with $n \geq 1$. Let (φ, θ, C) be the system, circuit, and ordering obtained after the kth iteration of the **while** loop of line 5. The $(k+1)$th iteration of the **while** loop runs in non-deterministic polynomial time in the bit sizes of φ and C. Each non-deterministic execution of the loop updates the triple (φ, θ, C) into a triple (φ', θ', C') such that (for every $\ell, s, a, c, d \geq 1$):*

$$\textit{if} \begin{cases} \#\textit{lst}(\varphi, \theta) & \leq \ell \\ \#\varphi & \leq s \\ \|\varphi\|_{\mathfrak{L}} & \leq a \\ \|\varphi\|_1 & \leq c \\ \textit{mod}(\varphi) & \mid d \end{cases} \quad \textit{then} \quad \begin{cases} \#\textit{lst}(\varphi', \theta') & \leq \ell + 2 \cdot k + 3 \\ \#\varphi' & \leq s + 6 \cdot k + 2 \cdot \ell + 3 \\ \|\varphi'\|_{\mathfrak{L}} & \leq 3 \cdot a \\ \|\varphi'\|_1 & \leq 12 + 4 \cdot \max(c, \log \beta) \\ \textit{mod}(\varphi') & \mid \operatorname{lcm}(d, \phi(\alpha \cdot d)) \\ \xi_{C'} & \leq \xi_C \cdot (3 \cdot k \cdot a)^k + 2^6 (k+1) \cdot \beta^4 \\ \mu_{C'} & \leq \mu_C \cdot (3 \cdot k \cdot a)^k, \end{cases}$$

with $\alpha \in [1..(3 \cdot k \cdot \mu_C \cdot a)^{k^2}]$, and $\beta \coloneqq d \cdot \left(2^7 (k+1) \cdot \mu_C \cdot \max(c, \log(\xi_C + \mu_C))\right)^{3(k+2)^2}$.

Proof. Throughout the proof, μ is short for μ_C. We analyze how the parameters evolve over the five steps of the while loop of OptILEP.

Step I (lines 6–9). Let γ and ψ be the systems computed by the algorithm in line 8. Directly from Lemma 22, we derive the following bounds on their parameters:

$$\begin{cases} \#\textit{lst}(\psi, \theta') & \leq \ell + 2 \cdot k \\ \#\psi & \leq s + 6 \cdot k + 2 \cdot \ell \\ \|\psi\|_{\mathfrak{L}} & \leq 3 \cdot a \\ \|\psi\|_1 & \leq 4 \cdot c + 5 \\ \textit{mod}(\psi) & \mid d \end{cases} \text{ and } \begin{cases} \#\gamma & \leq s + 2 \cdot k \\ \|\gamma[2^u \,/\, u]\|_{\mathfrak{L}} & \leq 3 \cdot a \\ \|\gamma\|_1 & \leq 2 \cdot c + 3 \\ \textit{mod}(\gamma) & \mid d \end{cases}$$

Note that the updates performed to C in line 9 do not change the values of ξ_C and μ.

Step II (lines 10–13). From Lemma 4, line 10 adds $k+1$ inequalities of the form $q \geq 0$ to γ (one for each quotient variable). The following line 11 multiplies all (in)equalities in γ by μ. Therefore, when the program reaches line 13, the formula γ satisfies the following:

$$\begin{cases} \#\gamma & \leq s + 3k + 1 \\ \|\gamma[2^u \,/\, u]\|_{\mathfrak{L}} & \leq 3 \cdot \mu \cdot a \\ \|\gamma\|_1 & \leq \mu \cdot (2 \cdot c + 3) \\ \mathit{mod}(\gamma) & \mid d \end{cases}$$

Observe that the formula ψ' constructed in line 12 has bit size polynomial in ψ'. The procedure ElimVars does not update this formula (Lemma 14), moreover ψ' is not used again within the loop. Therefore, no further analysis on ψ' is necessary. Together with ψ', the call to ElimVars in line 13 returns a linear program with divisions γ' and a (k,k)-LEAC C^*. (More precisely, we have $(C^*, \langle \gamma' \,;\, \psi' \rangle) \in \mathcal{I}_k^k$.) We bound the parameters of these two objects using Lemma 21. In order to simplify the analysis, let us define $M \coloneqq 2^6 \cdot \mu \cdot \max(c, \log(\xi_C + \mu))$. The values L, Q, U and R defined in Lemma 21 are all bounded by M. Furthermore, from the bound on $\|\gamma[2^u \,/\, u]\|_{\mathfrak{L}}$, we have $Q \leq 3 \cdot \mu \cdot a$, and therefore $\frac{Q}{\mu} \leq 3 \cdot a$.

number of constraints in γ': Directly from Lemma 21.1, $\#\gamma' \leq s + 4 \cdot k + 1$.

linear norm of γ': Since $(C^*, \langle \gamma' \,;\, \psi' \rangle) \in \mathcal{I}_k^k$, the system γ' only features the variable u and the quotient variable q_{n-k}. By Lemma 21.3, the linear norm of γ' is thus:

$$\begin{aligned} \|\gamma'\|_{\mathfrak{L}} &\leq \max\left(\mu \cdot (k+1)^{k+1} \left(\tfrac{Q}{\mu}\right)^{k+1}, (k+1)^{k+1} \left(\tfrac{Q}{\mu}\right)^{k} U\right) \\ &\leq \max\left(\mu \cdot (3 \cdot a \cdot (k+1))^{k+1}, (k+1)^{k+1} (3 \cdot a)^k M\right) \\ &\leq (3 \cdot (k+1) \cdot a)^{k+1} M. && \text{(as } \mu \leq M\text{)} \end{aligned}$$

1-norm of γ': First, by Lemma 21.3 the bound on the constants of the terms from terms(γ') is

$$\begin{aligned} &\frac{((k+1) \cdot Q)^{2(k+2)^2}}{\mu^{2k^2}} \cdot \mathit{mod}(\gamma) \cdot R \\ &\leq ((k+1) \cdot M)^{3(k+2)^2} d. && \text{(as } R, Q \leq M \text{ and } \mathit{mod}(\gamma) \leq d\text{)} \end{aligned}$$

Let us define $N \coloneqq 3 \cdot ((k+1) \cdot M)^{3(k+2)^2}$. Terms in terms($\gamma'$) are of the form $b_1 \cdot q + b_2 \cdot 2^u + b_3$, where $|b_1|$ and $|b_2|$ are bounded by $\|\gamma'\|_{\mathfrak{L}} \leq (3 \cdot (k+1) \cdot a)^{k+1} M$. Therefore, $\|\gamma'\|_1 \leq N \cdot d$.

modulus of γ': By Lemma 21.4, $\mathit{mod}(\gamma') \mid \alpha \cdot \mathit{mod}(\gamma)$, for some positive integer $\alpha \leq (3 \cdot k \cdot \mu \cdot a)^{k^2}$. (Observe that then $\mathrm{mod}(\gamma') \leq (k \cdot M)^{k^2} d$; we will silently use this fact when computing the bounds in Equation (24) below.)

denominator η_{C^*}: From Lemma 21.5, $\eta_{C^*} = \mu \cdot g$, for some $g \leq k^k (\frac{Q}{\mu})^k \leq (3 \cdot k \cdot a)^k$.

denominator μ_{C^*} and parameter ξ_{C^*}: by Lemma 21.2, $\mu_{C^*} = \mu$ and $\xi_{C^*} = \xi_C$.

numerators in the new assignments of C^*: Let us also observe that the terms τ occurring in assignments $q \leftarrow \frac{\tau}{\eta}$ of C^*, with q quotient variable, are linear terms in the variables u and q_{n-k}. From Lemma 21.3, $\|\tau\|_{\mathfrak{L}} \leq (3 \cdot (k+1) \cdot a)^{k+1} \cdot M$, whereas the constant of τ is bounded by $((k+1) \cdot M)^{3(k+2)^2} d$ (same computations as for $\|\gamma'\|_{\mathfrak{L}}$ and $\|\gamma'\|_1$).

Step III (line 14). Starting from γ', line 14 produces two formulae γ'' and ψ''. From the bounds we have just obtained from γ'', and by appealing to Lemma 23, we get:

$$\begin{cases} \#\gamma'' & \leq s + 4k + 3 \\ \|\gamma''\|_1 & \leq 2^5 N^3 d^3 \\ mod(\gamma'') & \mid \operatorname{lcm}(\alpha \cdot d,\, \phi(\alpha \cdot d)) \end{cases} \quad \text{and} \quad \begin{cases} \#\psi'' & \leq 3 \\ \|\psi''\|_{\mathfrak{L}} & \leq 1 \\ \|\psi''\|_1 & \leq 12 + 4 \cdot \log(N \cdot d) \\ mod(\psi'') & \mid \phi(\alpha \cdot d), \end{cases} \tag{24}$$

for some positive integer $\alpha \leq (3 \cdot k \cdot \mu \cdot a)^{k^2}$.

Step IV (lines 15–19). The integers ℓ and h in these lines are bounded by $\|\gamma''\|_1 + mod(\gamma'')$. Therefore, the value v chosen in line 18 satisfies $0 \leq v \leq \|\gamma''\|_1 + mod(\gamma'')$. Observe that $mod(\gamma'') \leq (\alpha \cdot d)^2 \leq (k \cdot M)^{2k^2} d^2 \leq N \cdot d^2$. Hence, $|v| \leq 2^6 N^3 d^3$. In line 19, the algorithm constructs the $(k+1)$-PreLEAC C' whose bounds we are interested in. Following Remark 4, we obtain the bounds on $\mu_{C'}$ and $\xi_{C'}$ reported in the statement of the lemma:

denominator $\mu_{C'}$: Remark 4 tells us that $\mu_{C'} = \eta_{C^*} \leq \mu \cdot (3 \cdot k \cdot a)^k$.

parameter $\xi_{C'}$: In our case, λ from Remark 4 is equal to $\frac{\eta_{C^*}}{\mu} = g$. We have:

$$\begin{aligned} \xi_{C'} &\coloneqq \underbrace{|\eta_{C^*} \cdot v|}_{\text{from assignment to } x_{n-k}} + \sum\nolimits_{i=0}^{k-1} \underbrace{\Big(|c_{n-i} \cdot v + d_{n-i}| + |\lambda \cdot (a_{i,k} + b_{n-i})| + \sum\nolimits_{j=i+1}^{k-1} |\lambda \cdot a_{i,j}| \Big)}_{\text{from assignment to } x_{n-i}} \\ &\leq v \cdot \Big(\eta_{C^*} + \sum\nolimits_{i=0}^{k-1} (|c_{n-i}| + |d_{n-i}| + g \cdot |b_{n-i}|) \Big) + g \cdot \xi_C. \end{aligned} \tag{25}$$

Observe now that $|c_{n-i}|$, $|d_{n-i}|$ and $|b_{n-i}|$ are integers occurring in assignments $q \leftarrow \frac{\tau}{\eta}$ of C^*, with q quotient variable. From the bounds already deduced for these integers, we have

$$g \cdot |b_{n-i}| \leq (3 \cdot k \cdot a)^k (3 \cdot (k+1) \cdot a)^{k+1} M \leq \tfrac{N}{3},$$

and, similarly, $|c_{n-i}| \leq \frac{N}{3}$ and $|d_{n-i}| \leq \frac{N}{3} \cdot d$. Resuming the computation in Equation (25):

$$\begin{aligned} \xi_{C'} &\leq v \cdot (\eta_{C^*} + k \cdot N \cdot d) + g \cdot \xi_C \;\leq\; v \cdot (\mu \cdot (3 \cdot k \cdot a)^k + k \cdot N \cdot d) + g \cdot \xi_C \\ &\leq 2^6 (k+1) \cdot N^4 d^4 + (3 \cdot k \cdot a)^k \xi_C. \end{aligned}$$

The bound on $\xi_{C'}$ in the statement of the lemma then follows from

$$N \cdot d \leq 3 \cdot d \cdot ((k+1) \cdot M)^{3(k+2)^2} \leq 3 \cdot d \cdot ((k+1) \cdot 2^6 \mu \cdot \max(c, \log(\mu + \xi_C)))^{3(k+2)^2} \leq \beta.$$

Step V (lines 20–22). We have $\varphi' \coloneqq \psi \wedge \psi''$. Then, the bounds on the parameters of φ' given in the statement of the lemma are obtained by simply combining those computed for ψ and ψ''.

Moving to the running time of performing one iteration of the body of the **while** loop, the bounds established above show that all operations performed (excluding calls to subprocedures) involves objects of polynomial size with respect to the sizes of φ and C. It is simple to see that all these operations (e.g., those in lines 9 or 16) can be performed in polynomial time. Additionally, the guess in line 18 ranges over an interval of integers with polynomial bit length. Then, the non-deterministic polynomial-time complexity of one iteration of the loop follows directly from Lemmas 21 to 23. □

The complexity of performing k iterations of the main loop. To complete the complexity analysis of OptILEP it now suffices to iterate the bounds computed in Lemma 24 across multiple iterations of the main loop of the algorithm.

Lemma 25. *Algorithm 6 runs in non-deterministic polynomial time. Consider its execution on an integer linear-exponential program $\varphi(x_1, \ldots, x_n)$ with $n \geq 1$. Let $(\varphi_k, \theta_k, C_k)$ the system, circuit, and ordering obtained at the end of kth iteration of the **while** loop of line 5, in any non-deterministic branch of the algorithm. Then, the following bounds hold (for every $\ell, s, a, c \geq 1$):*

Proof in page 104

$$\textit{if} \begin{cases} \#\mathit{lst}(\varphi, \theta) & \leq \ell \\ \#\varphi & \leq s \\ \|\varphi\|_{\mathfrak{L}} & \leq a \\ \|\varphi\|_1 & \leq c \\ \mathit{mod}(\varphi) & \mid 1 \end{cases} \quad \textit{then} \quad \begin{cases} \#\mathit{lst}(\varphi_k, \theta_k) & \leq \ell + 3 \cdot k^2 \\ \#\varphi_k & \leq s + 3 \cdot k^3 + 2 \cdot \ell \cdot k \\ \|\varphi_k\|_{\mathfrak{L}} & \leq 3^k a \\ \|\varphi_k\|_1 & \leq 3^{8(k+1)} c \\ \mathit{mod}(\varphi_k) & \leq 3^{2 \cdot k^8} a^{2 \cdot k^7} \\ \xi_{C_k} & \leq 3^{8(k+2)^8} c^{8(k+2)^7} \\ \mu_{C_k} & \leq 3^{k^3} a^{k^2}. \end{cases}$$

Proof sketch. The proof is by induction on k, assuming as the induction hypothesis that the bounds stated hold at the kth iteration of the loop. This hypothesis, combined with Lemma 24, is sufficient to establish all bounds except for the one given to $\mathit{mod}(\varphi_k)$, which we discuss next.

Let $(\varphi_0, C_0), \ldots, (\varphi_k, C_k)$ denote the formulae and PreLEACs constructed by the algorithm during the first k iterations of the **while** loop. We are looking to bound $\mathit{mod}(\varphi_{k+1})$. In particular, φ_0 is the linear-exponential program given as input to OptILEP, and C_0 is the empty 0-PreLEAC initialized in line 1. By Lemma 24, for every $i \in [0..k]$, there is $\alpha_{i+1} \in [1..(3 \cdot i \cdot \mu_{C_i} \cdot \|\varphi_i\|_{\mathfrak{L}})^{i^2}]$ such that $\mathit{mod}(\varphi_{i+1})$ is a divisor of $\operatorname{lcm}(\mathit{mod}(\varphi_i), \phi(\alpha_{i+1} \cdot \mathit{mod}(\varphi_i)))$. Let us define $\alpha^* \coloneqq \operatorname{lcm}(\alpha_1, \alpha_2, \ldots, \alpha_{k+1})$, and consider the integers $c_0, \ldots, c_{k+1}$ such that $c_0 \coloneqq 1$ and $c_{i+1} \coloneqq \operatorname{lcm}(c_i, \phi(\alpha^* \cdot c_i))$ for $i \in [0..k]$.

Claim 17. *For every $j \in [0..k+1]$, $\mathit{mod}(\varphi_j)$ divides c_j.*

Proof. The proof is by induction on j.

base case: $j = 0$**.** We have $\mathit{mod}(\varphi_0) = 1 = c_0$.

induction step: Assume that the claim holds for $j \in [0..k]$. Then,

$$\begin{aligned} \mathit{mod}(\varphi_{j+1}) &\coloneqq \operatorname{lcm}(\mathit{mod}(\varphi_j), \phi(\alpha_{j+1} \cdot \mathit{mod}(\varphi_j))) \\ &\mid \operatorname{lcm}(c_j, \phi(\alpha_{j+1} \cdot \mathit{mod}(\varphi_j))) && \text{⟅}\mathit{mod}(\varphi_i) \mid c_i \text{ by induction hypothesis⟆} \\ &\mid \operatorname{lcm}(c_j, \phi(\alpha^* \cdot c_j)) && \text{⟅}q \mid r \text{ implies } \phi(q) \mid \phi(r)\text{⟆} \\ &= c_{j+1}. \end{aligned}$$

□

Given Claim 17, in order to bound $\mathit{mod}(\varphi_{k+1})$ it suffices to bound c_{k+1}. The next lemma from [CMS24] will help us analyze this integer.

Lemma 26 [CMS24, Lemma 7]**.** *Let $\alpha \geq 1$ be in $\mathbb{N}$. Let $b_0, b_1, \ldots$ be the integer sequence given by the recurrence $b_0 \coloneqq 1$ and $b_{i+1} \coloneqq \operatorname{lcm}(b_i, \phi(\alpha \cdot b_i))$. For every $i \in \mathbb{N}$, $b_i \leq \alpha^{2 \cdot i^2}$.*

By Lemma 26, $c_{k+1} \leq (\alpha^*)^{2(k+1)^2}$. Therefore, $c_{k+1} \leq 3^{2(k+1)^8} a^{2(k+1)^7}$ follows from

$$\begin{aligned} \alpha^* &\leq \textstyle\prod_{i=0}^{k} (3 \cdot i \cdot \mu_i \cdot \|\varphi_i\|_{\mathfrak{L}})^{i^2} \\ &\leq (3 \cdot k \cdot (3^{k^3} a^{k^2}) \cdot (3^k a))^{k^2(k+1)} && \text{⟅by induction hypothesis⟆} \\ &\leq 3^{(k+1)^6} a^{(k+1)^5}. \end{aligned}$$

□

6.4 Maximization and minimization of arbitrary linear-exponential terms

Together, Sections 6.2 and 6.3 establish Theorem 1 for the case of maximizing a variable x under a linear-exponential program. We now complete the proof of Theorem 1 by extending the argument to general linear-exponential objective functions, and to include the case of minimization.

Let us consider first the maximization problem

$$\text{maximize } \tau(\boldsymbol{x}) \text{ subject to } \varphi(\boldsymbol{x}), \tag{26}$$

where τ is a linear-exponential term, and φ an integer linear-exponential program. Let z be a variable not in $\boldsymbol{x}$. Since variables in ILEP range over $\mathbb{N}$, a common approach to solving this problem is to distinguish two cases based on the sign of $\tau(\boldsymbol{x})$ in the optimal solution The variable z is used to represent the value of $\tau(\boldsymbol{x})$, adjusting its sign accordingly when assuming τ to be negative. Here is the corresponding pseudocode:

1: **if** (maximize z subject to $\varphi(\boldsymbol{x}) \wedge z = \tau(\boldsymbol{x})$) has an optimal solution σ_1 **then return** σ_1
2: **if** $\varphi(\boldsymbol{x}) \wedge \tau(\boldsymbol{x}) \geq 0$ is satisfiable **then return** "no optimal solution exists"
3: **if** (minimize z subject to $\varphi(\boldsymbol{x}) \wedge -z = \tau(\boldsymbol{x})$) has an optimal solution σ_2 **then return** σ_2
4: **return** "φ is unsatisfiable"

From Sections 6.2 and 6.3, we know that the maximization problem in line 1 admits an optimal solution, then it has one representable with a polynomial-size ILESLP. Regarding the minimization problem in line 3, since z ranges over $\mathbb{N}$, a minimal solution is guaranteed to exist as soon as $\varphi(\boldsymbol{x}) \wedge -z = \tau(\boldsymbol{x})$ is satisfiable. Again from Sections 6.2 and 6.3 when $\varphi(\boldsymbol{x}) \wedge -z = \tau(\boldsymbol{x})$ is satisfiable, then there is *a* solution representable with a polynomial-size ILESLP σ. Then, to solve the minimization problem in line 3, we can consider the equivalent maximization problem "maximize $[\![\sigma]\!](z) - z$ subject to $\varphi(\boldsymbol{x}) \wedge -z = \tau(\boldsymbol{x})$", since $[\![\sigma]\!](z) - z$ attains its maximum precisely when z is minimal. Given that $[\![\sigma]\!](z) - z$ is non-negative, this problem can be reformulated as "maximize w subject to $\varphi(\boldsymbol{x}) \wedge (-z = \tau(\boldsymbol{x})) \wedge (w = [\![\sigma]\!](z) - z)$", where w is a fresh variable. Of course, $[\![\sigma]\!](z)$ may not be representable in binary using polynomially many bits. Instead, we incorporate directly the ILESLP σ directly into the constraints of the linear-exponential program. To do so, we first rename every variable y occurring in σ as y' to avoid conflicts with the variables $\boldsymbol{x}$, z and w. Let σ be now of the form $(y'_0 \leftarrow \rho_1, \ldots, y_t \leftarrow \rho_t)$. We then solve the following maximization problem

$$\text{maximize } w \text{ subject to } \varphi(\boldsymbol{x}) \wedge (-z = \tau(\boldsymbol{x})) \wedge (w = z' - z) \wedge \textstyle\bigwedge_{i=1}^{t}(y'_i = \rho_i).$$

From Sections 6.2 and 6.3, if this problem has an optimal solution, then it has one representable with a polynomial-size ILESLP. We conclude that the same holds for the problem in Equation (26).

We can treat the minimization problem

$$\text{minimize } \tau(\boldsymbol{x}) \text{ subject to } \varphi(\boldsymbol{x}),$$

in a similar way. Again following the sign of τ, this problem is solved as follows:

1: **if** (maximize z subject to $\varphi(\boldsymbol{x}) \wedge -z = \tau(\boldsymbol{x})$) has an optimal solution σ_1 **then return** σ_1
2: **if** $\varphi(\boldsymbol{x}) \wedge \tau(\boldsymbol{x}) < 0$ is satisfiable **then return** "no optimal solution exists"
3: **if** (minimize z subject to $\varphi(\boldsymbol{x}) \wedge z = \tau(\boldsymbol{x})$) has an optimal solution σ_2 **then return** σ_2
4: **return** "φ is unsatisfiable"

We already know that if one of the optimization problems in the code above has an optimal solution, then it has one representable by a polynomial-size ILESLP. This concludes the proof of Theorem 1.

Part II
Deciding properties of ILESLPs

In this second part of the paper, we discuss algorithms for deciding $\text{NAT}_{\text{ILESLP}}$ and $\text{DIV}_{\text{ILESLP}}$ (Sections 7 and 8, respectively), and for computing ILESLPs representing terms of the form $(x \bmod 2^y)$ (Section 9). This part is almost completely independent of Part I, the sole exception being an appeal to Lemma 10 when proving that $\text{NAT}_{\text{ILESLP}}$ is in P. We refer the reader to section Section 1.1 for the definition of ILESLPs.

Some notation and an auxiliary lemma. Let $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$ be an ILESLP. We write $e(\sigma)$ (respectively, $d(\sigma)$) for the absolute value of the product of all numerators $m \neq 0$ (respectively, denominators g) occurring in rational constants $\frac{m}{g}$ of the scaling expressions $\frac{m}{g} \cdot x_j$ in σ. By convention, this product is defined to be 1 when taken over an empty set. Given an expression $E \coloneqq \sum_{j \in J} a_j \cdot 2^{x_j}$, where $J \subseteq [0..n]$ and each a_j is an integer, we write $[\![\sigma]\!](E)$ for the number obtained by *evaluating* E on σ, that is, $[\![\sigma]\!](E) \coloneqq \sum_{j \in J} a_j \cdot 2^{[\![\sigma]\!](x_j)}$. The next auxiliary lemma recasts σ into a form that is more amenable to our subsequent algorithms.

Lemma 27. *Consider an ILESLP $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$ and let $i \in [0..n]$. One can compute, in time polynomial in the size of σ, an expression E_i of the form $\sum_{j=0}^{i-1} a_{i,j} \cdot 2^{x_j}$ such that $[\![\sigma]\!](E_i) = d(\sigma) \cdot [\![\sigma]\!](x_i)$. For every $j \in [0..i-1]$, the coefficient $a_{i,j}$ is* (i) *an integer whose absolute value is bounded by $2^i \cdot e(\sigma) \cdot d(\sigma)$, and* (ii) *non-zero only if $[\![\sigma]\!](x_j) \geq 0$.*

Proof in page 107

Proof sketch. Given $i \in [0..n]$, let σ_i denote the ILESLP $(x_0 \leftarrow \rho_0, \ldots, x_i \leftarrow \rho_i)$ obtained by truncating σ after $i+1$ assignments. We remark that $d(\sigma_i)$ divides $d(\sigma_j)$ for every $i \leq j$.

Inductively on i, one shows that it is possible to compute a vector of rational numbers $\boldsymbol{b}_i = (b_{i,0}, \ldots, b_{i,i-1})$ satisfying $[\![\sigma]\!](x_i) = \sum_{j=0}^{i-1} b_{i,j} \cdot 2^{[\![\sigma]\!](x_j)}$, where each $b_{i,j}$ is of the form $\frac{m}{d(\sigma_i)}$ for some $m \in \mathbb{Z}$ satisfying $|m| \leq 2^i \cdot e(\sigma_i) \cdot d(\sigma_i)$, and $m \neq 0$ only if $[\![\sigma]\!](x_j) \geq 0$. With this result at hand, the expression E_i in the statement of the lemma is computed by multiply all these rational numbers by $d(\sigma)$, as to make them all integers. In particular, if $b_{i,j} = \frac{m}{d(\sigma_i)}$, then in E_i the coefficient of 2^{x_j} is $a_{i,j} \coloneqq m \cdot \frac{d(\sigma)}{d(\sigma_i)}$. Hence, $|a_{i,j}| \leq 2^i \cdot e(\sigma_i) \cdot d(\sigma_i) \cdot \frac{d(\sigma)}{d(\sigma_i)} \leq 2^i \cdot e(\sigma) \cdot d(\sigma)$. Note that the bit size of each $a_{i,j}$ is thus polynomial in the size of σ. With this in mind, the fact that the whole computation can be performed in polynomial time follows immediately from the inductive proof. □

7 Deciding $\text{NAT}_{\text{ILESLP}}$ in polynomial time

$\text{NAT}_{\text{ILESLP}}$	
Input:	An ILESLP σ.
Question:	Is $[\![\sigma]\!]_\bullet \geq 0$?

The pseudocode of our procedure for deciding $\text{NAT}_{\text{ILESLP}}$ is given in Algorithm 7. In a nutshell, given an ILESLP $\sigma = (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$, the algorithm constructs a map M with the following property: for every $i, j \in [0..n]$, the entry $M(i,j)$ stores the value of the difference $[\![\sigma]\!](x_i) - [\![\sigma]\!](x_j)$ up to a certain threshold C defined in line 1. If the absolute value of this difference exceeds the threshold, then $M(i,j)$ is instead equal to $+\infty$ or $-\infty$, depending on the sign of the difference. After constructing M, the algorithm checks whether $M(n,0) \geq 0$ to decide if $[\![\sigma]\!](x_n) \geq 0$.

Algorithm 7 A polynomial-time algorithm for NAT$_{\text{ILESLP}}$.

Input: ILESLP $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$.

1: $C \leftarrow 8 \cdot (\text{bit size of } \sigma) + 8$
2: **let**, for $i \in [0..n]$, E_i be an expression $\sum_{j=0}^{i-1} a_j \cdot 2^{x_j}$, with all $a_j \in \mathbb{Z}$, and $[\![\sigma]\!](x_i) = \frac{\sum_{j=0}^{i-1} a_j \cdot 2^{[\![\sigma]\!](x_j)}}{d(\sigma)}$
3: $M \leftarrow$ empty map from $[0..n]^2$ to $[-C..C] \cup \{-\infty, \infty\}$
4: **for** i from 0 to n **do**
5: $\quad M(i,i) \leftarrow 0$
6: $\quad$ **let** $\{\ell_0 = 0, \ldots, \ell_m\}$ maximal subset of $[0..i-1]$ such that $M(\ell_k, \ell_{k-1}) \geq 0$ for every $k \in [1..m]$
7: $\quad$ **for** j from 0 to $i-1$ **do**
8: $\quad\quad E \leftarrow E_i - E_j$ ▷ *E is of the form $\sum_{k=0}^{m} c_k 2^{x_{\ell_k}}$*
9: $\quad\quad$ **for** k from m to 1 **do**
10: $\quad\quad\quad$ **if** $M(\ell_k, \ell_{k-1}) \leq C$ **then** replace $2^{x_{\ell_k}}$ with $2^{M(\ell_k,\ell_{k-1})} 2^{x_{\ell_{k-1}}}$ in E
11: $\quad\quad\quad$ **else**
12: $\quad\quad\quad\quad a \leftarrow$ coefficient of $2^{x_{\ell_k}}$ in E
13: $\quad\quad\quad\quad$ **if** $a > 0$ **then** $M(i,j) \leftarrow +\infty$
14: $\quad\quad\quad\quad$ **if** $a < 0$ **then** $M(i,j) \leftarrow -\infty$
15: $\quad\quad\quad\quad$ **if** $a \neq 0$ **then break**
16: $\quad\quad$ **if** E is of the form $h \cdot 2^{x_0}$ for some $h \in \mathbb{Z}$ **then**
17: $\quad\quad\quad$ **if** $\frac{h}{d(\sigma)} \in [-C..C]$ **then** $M(i,j) \leftarrow \frac{h}{d(\sigma)}$
18: $\quad\quad\quad$ **else** $M(i,j) \leftarrow$ **if** $\frac{h}{d(\sigma)} > 0$ **then** $+\infty$ **else** $-\infty$
19: $\quad\quad M(j,i) \leftarrow -M(i,j)$
20: **return** true **if** $M(n,0) \geq 0$ **else** false

Following Lemma 27, for every $i \in [0..n]$, the algorithm starts by "flattening" the expression ρ_i of x_i into the form $\frac{E_i}{d(\sigma)}$, where $E_i = \sum_{k=0}^{i-1} a_k \cdot 2^{x_k}$ with $a_0, \ldots, a_{i-1}$ integers (line 2). The computation of $M(i,j)$ with $j < i$ occurs at the $(i+1)$th iteration of the loop of line 4 and $(j+1)$th iteration of the loop of line 7. To compute $M(i,j)$, the expression $E \coloneqq E_i - E_j$ is considered (line 8). This is again of the form $\sum_{k=0}^{i-1} b_k \cdot 2^{x_k}$, where $b_k \neq 0$ only if $[\![\sigma]\!](x_k) \geq 0$ (again by Lemma 27). Since all entries of M involving variables $x_0, \ldots, x_{i-1}$ are already computed in the earlier iterations, we can reduce E to an expression $\sum_{k=0}^{m} c_k \cdot 2^{x_{\ell_k}}$, where $\ell_0, \ldots, \ell_m \in [0..i-1]$ are the indices of the variables x_ℓ with $[\![\sigma]\!](x_\ell) \geq 0$, in ascending order (see line 6).

Intuitively, if $[\![\sigma]\!](x_{\ell_m})$ is large enough compared to $[\![\sigma]\!](x_{\ell_{m-1}})$, then the sign of $\sum_{k=0}^{m} c_k \cdot 2^{[\![\sigma]\!](x_{\ell_k})}$ is solely determined by the sign of the integer c_m (assuming $c_m \neq 0$). The threshold C has been chosen to capture this idea of x_{ℓ_m} being "large enough". In particular, one can show that for every $k \in [1..m]$, if $[\![\sigma]\!](x_{\ell_k}) - [\![\sigma]\!](x_{\ell_{k-1}}) > C$ and $c_k \neq 0$, then $\left|c_k 2^{[\![\sigma]\!](x_{\ell_k})}\right| > \left|\sum_{j=0}^{k-1} c_j 2^{[\![\sigma]\!](x_{\ell_j})}\right| + d(\sigma) \cdot C$. So, if $M(\ell_m, \ell_{m-1}) > C$ and $c_m \neq 0$, we have $M(i,j) = \pm\infty$ (lines 12–15). Otherwise, if $[\![\sigma]\!](x_{\ell_m})$ is small compared to $[\![\sigma]\!](x_{\ell_{m-1}})$, that is, $M(\ell_m, \ell_{m-1}) \leq C$, we replace $2^{x_{\ell_m}}$ with $2^{M(\ell_m,\ell_{m-1})} \cdot 2^{x_{\ell_{m-1}}}$ in E, and iterate the same reasoning; now on variables $x_{\ell_{m-1}}$ and $x_{\ell_{m-2}}$. At the end of the loop of line 9, either $M(i,j)$ has been set to $\pm\infty$, or we have reduced E into an expression of the form $h \cdot 2^{x_0}$. In the latter case, we have $[\![\sigma]\!](x_i) - [\![\sigma]\!](x_j) = \frac{h \cdot 2^{[\![\sigma]\!](x_0)}}{d(\sigma)} = \frac{h}{d(\sigma)}$. If $\frac{h}{d(\sigma)}$ belongs to $[-C..C]$, the algorithm sets $M(i,j) = \frac{h}{d(\sigma)}$ (line 17). Else, $M(i,j)$ is set to $\pm\infty$, according to the sign of $\frac{h}{d(\sigma)}$.

To prove that Algorithm 7 decides NAT$_{\text{ILESLP}}$ in polynomial time, the key observation is that C is linear in the bit size of σ, and thus so is the bit size of the integers $2^{M(\ell_k,\ell_{k-1})}$ computed in line 10.

We now formalize the above explanation, proving correctness and polynomial running time

of Algorithm 7. The correctness proof centers on the semantics of the map M.

Lemma 28. *Given an input ILESLP $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$, Algorithm 7 constructs a map $M : [0..n]^2 \to [-C..C] \cup \{-\infty, \infty\}$, where $C \coloneqq 8 \cdot (\text{bit size of } \sigma) + 8$. For all $i, j \in [0..n]$, this map satisfies $M(i,j) = T_C(\llbracket\sigma\rrbracket(x_i) - \llbracket\sigma\rrbracket(x_j))$, where T_C is the truncation function*

$$T_C(g) \coloneqq \begin{cases} g & \text{if } g \in [-C..C] \\ -\infty & \text{if } g < -C \\ +\infty & \text{if } g > C \end{cases} \qquad \text{for every } g \in \mathbb{Z}.$$

Proof. The map M is initialized as empty in line 3. Let $E_0, \ldots, E_n$ be the expressions computed in line 2, following Lemma 27. They satisfy $\llbracket\sigma\rrbracket(E_i) = d(\sigma) \cdot \llbracket\sigma\rrbracket(x_i)$, for every $i \in [0..n]$. We prove by induction on $i \in \mathbb{N}$ that after the $(i+1)$th iteration of the outer **for** loop of line 4, the map M satisfies $M(j,k) = T_C(\llbracket\sigma\rrbracket(x_j) - \llbracket\sigma\rrbracket(x_k))$ for every $j, k \in [0..i]$.

base case: $i = 0$**.** In the first iteration, line 5 sets $M(0,0) = 0$ as required. The inner loop of line 7 does not execute for $i = 0$ as the range of j is empty.

induction hypothesis. For every $j, k \in [0..i-1]$, $M(j,k) = T_C(\llbracket\sigma\rrbracket(x_j) - \llbracket\sigma\rrbracket(x_k))$.

induction step: $i \geq 1$**.** In the $(i+1)$th iteration, the algorithm sets $M(i,j)$ and $M(j,i)$ for all $j \in [0..i]$. Line 5 correctly sets $M(i,i) = 0$. For each $j \in [0..i-1]$, the inner loop of line 7 computes $M(i,j)$ and $M(j,i)$. As $M(j,i) = -M(j,i)$ by line 19, it suffices to show that the computed $M(i,j) = T_C(\llbracket\sigma\rrbracket(x_i) - \llbracket\sigma\rrbracket(x_j))$ for all $j \in [0..i-1]$.

In line 6, the algorithm computes the maximal subset of indices $\{\ell_0 = 0, \ldots, \ell_m\} \subseteq [0..i-1]$ such that $M(\ell_k, \ell_{k-1}) \geq 0$ for every $k \in [1..m]$. By the induction hypothesis, $M(\ell_k, \ell_{k-1}) = T_C(\llbracket\sigma\rrbracket(x_{\ell_k}) - \llbracket\sigma\rrbracket(x_{\ell_{k-1}}))$, which implies $0 = \llbracket\sigma\rrbracket(x_{\ell_0}) \leq \llbracket\sigma\rrbracket(x_{\ell_1}) \leq \ldots \leq \llbracket\sigma\rrbracket(x_{\ell_m})$. Moreover, by the maximality of this subset, the variables $x_{\ell_0}, \ldots, x_{\ell_m}$ are exactly those among $x_0, \ldots, x_{i-1}$ for which $\llbracket\sigma\rrbracket$ is non-negative.

Computation of $M(i,j)$: This computation occurs at the $(j+1)$th iteration of the inner **for** loop of line 7, with $j \in [0..i-1]$. Let E be the expression $E_i - E_j$ as in line 8. By Lemma 27, E is of the form $\sum_{k=0}^{i-1} a_k \cdot 2^{x_k}$ where

1. each a_k is an integer whose absolute value is bounded by $2^{n+1} \cdot e(\sigma) \cdot d(\sigma)$,
2. if $a_k \neq 0$, then $\llbracket\sigma\rrbracket(x_k) \geq 0$.

The second property above implies that E can be written as $\sum_{k=0}^{m} a_{\ell_k} \cdot 2^{x_{\ell_k}}$. From the definition of E_i and E_j, we also have $\llbracket\sigma\rrbracket(E) = d(\sigma) \cdot (\llbracket\sigma\rrbracket(x_i) - \llbracket\sigma\rrbracket(x_j))$.

The algorithm now enters the inner **for** loop of line 9, iterating k from m down to 1. This loop progressively rewrites the expression E. Let $E^{(t)}$ denote for the value of E after the tth iteration of the loop. During the tth iteration, the value of the variable k is $m - t + 1$. The following two claims (proved later) define the behaviour of this loop:

Claim 18. *If the tth iteration of the loop of line 9 completes without executing the **break** statement of line 15, then $E^{(t)}$ is of the form $c_{\ell_{m-t}} \cdot 2^{x_{\ell_{m-t}}} + \sum_{k=0}^{m-t-1} a_{\ell_k} \cdot 2^{x_{\ell_k}}$, where $c_{\ell_{m-t}}$ is an integer satisfying $\left|c_{\ell_{m-t}}\right| < 2^{tC+n+2} \cdot e(\sigma) \cdot d(\sigma)$. Moreover, $\llbracket\sigma\rrbracket(E^{(t)}) = \llbracket\sigma\rrbracket(E)$.*

Claim 19. *If the tth iteration of the loop of line 9 executes the **break** statement of line 15, then $|\llbracket\sigma\rrbracket(E)| > d(\sigma) \cdot C$, and $\llbracket\sigma\rrbracket(E)$ and the coefficient of $2^{x_{\ell_{m-t+1}}}$ in $E^{(t-1)}$ have the same sign.*

Using these two claims, we can now verify that $M(i,j)$ is computed correctly. If the **break** statement of line 15 is executed during some iteration t of the loop of line 9, then by Claim 19, we have $\left|\frac{[\![\sigma]\!](E)}{d(\sigma)}\right| > C$, and $\frac{[\![\sigma]\!](E)}{d(\sigma)}$ has the same sign as that of the coefficient of $2^{x_{\ell_{m-t+1}}}$ in $E^{(t-1)}$. This coefficient is the integer a referenced in line 12, and since the **break** statement was executed, $a \neq 0$. Accordingly, lines 13 and 14 set the value of $M(i,j)$ to $\pm\infty$, following the sign of a. Hence, $M(i,j) = T_C(\frac{[\![\sigma]\!](E)}{d(\sigma)}) = T_C([\![\sigma]\!](x_i) - [\![\sigma]\!](x_j))$, as required.

Suppose now that the **break** statement of line 15 is never executed: the loop of line 9 terminates after m iterations, and the expression $E^{(m)}$ is defined. By Claim 18, this expression is of the form $c_{\ell_0} \cdot 2^{x_{\ell_0}}$, for some integer c_{ℓ_0} (the bound on c_{ℓ_0} given in Claim 18 will be later used in the runtime analysis in Lemma 1). Since $\ell_0 = 0$ and $[\![\sigma]\!](x_0) = 0$, we have $[\![\sigma]\!](E^{(m)}) = c_{\ell_0}$. Recall that $[\![\sigma]\!](E_m) = [\![\sigma]\!](E) = d(\sigma) \cdot ([\![\sigma]\!](x_i) - [\![\sigma]\!](x_j))$, so $M(i,j)$ must be set to $T_C(\frac{c_{\ell_0}}{d(\sigma)})$. This is exactly what the algorithm does in lines 16–18.

To complete the induction step, it is now sufficient to prove Claims 18 and 19.

Proof of Claim 18. For simplicity, let $B \coloneqq 2^{n+1} \cdot e(\sigma) \cdot d(\sigma)$. Note that $B \in [2..2^{3\cdot(\text{bit size of } \sigma)+1}]$; in particular, $B < 2^C$. The proof is by induction on t.

base case: $t = 0$**.** Before the first iteration of the loop, we have $E^{(0)} = E$. Recall that E is an expression of the form $\sum_{k=0}^{i-1} a_k \cdot 2^{x_k}$ where each a_k is an integer whose absolute value is bounded by B. Thus, $|a_0| < 2 \cdot B$, as required.

induction hypothesis. If the $(t-1)$th iteration of the loop of line 9 completes without executing the **break** statement of line 15, then the expression $E^{(t-1)}$ is of the form $c_{\ell_{m-t+1}} \cdot 2^{x_{\ell_{m-t+1}}} + \sum_{k=0}^{m-t} a_{\ell_k} \cdot 2^{x_{\ell_k}}$, where $c_{\ell_{m-t+1}} \in \mathbb{Z}$ satisfies $\left|c_{\ell_{m-t+1}}\right| < 2^{(t-1)C+1} \cdot B$. Moreover, $[\![\sigma]\!](E^{(t-1)}) = [\![\sigma]\!](E)$.

induction step: $t \geq 1$**.** Suppose that the tth iteration of the loop of line 9 completes without executing the **break** statement of line 15. Then, the same must hold for the $(t-1)$th iteration, and so the induction hypothesis applies. Since the **break** statement is not executed, there are two options:

1. The condition of the **if** statement in line 10 is true, i.e., $M(\ell_{m-t+1}, \ell_{m-t}) \leq C$, or
2. $M(\ell_{m-t+1}, \ell_{m-t}) = +\infty$ and the coefficient $c_{\ell_{m-t+1}}$ of $2^{x_{\ell_{m-t+1}}}$ in $E^{(t-1)}$ is zero.

In the second case, no update is needed: $E^{(t-1)}$ is already of the form $\sum_{k=0}^{m-t} a_{\ell_k} \cdot 2^{x_{\ell_k}}$, and we have $E^{(t)} = E^{(t-1)}$. In the first case, $E^{(t)}$ is constructed from $E^{(t-1)}$ by replacing $2^{x_{\ell_{m-t+1}}}$ by $2^{M(\ell_{m-t+1},\ell_{m-t})} \cdot 2^{x_{\ell_{m-t}}}$; see line 10. This removes $2^{x_{\ell_{m-t+1}}}$, and modifies the coefficient of $2^{x_{\ell_{m-t}}}$ from $a_{\ell_{m-t}}$ to $c_{\ell_{m-t}} \coloneqq (a_{\ell_{m-t}} + c_{\ell_{m-t+1}} \cdot 2^{M(\ell_{m-t+1},\ell_{m-t})})$. The coefficients of the terms $2^{x_{\ell_k}}$ with $k \in [0..m-t-1]$ are unchanged. As $M(\ell_{m-t+1}, \ell_{m-t}) \geq 0$, we have $c_{\ell_{m-t}} \in \mathbb{Z}$. Finally, we bound the absolute value of $c_{\ell_{m-t}}$ as follows:

$$
\begin{aligned}
\left|c_{\ell_{m-t}}\right| &= \left|a_{\ell_{m-t}}\right| + \left|c_{\ell_{m-t+1}}\right| \cdot 2^{M(\ell_{m-t+1},\ell_{m-t})} \\
&\leq B + \left|c_{\ell_{m-t+1}}\right| \cdot 2^{C} && \text{⟅bounds on } a_{\ell_{m-t}} \text{ and } M(\ell_{m-t+1},\ell_{m-t})\text{⟆} \\
&\leq B + (2^{(t-1)C+1} \cdot B - 1) \cdot 2^{C} && \text{⟅by induction hypothesis⟆} \\
&< 2^{C} + (2^{(t-1)C+1} \cdot B - 1) \cdot 2^{C} && \text{⟅from } B < 2^C\text{⟆} \\
&< 2^{tC+1} \cdot B.
\end{aligned}
$$

□

Proof of Claim 19. If the tth iteration of the loop of line 9 executes the **break** statement of line 15, then the first $t-1$ iterations completed without executing the **break** statement.

By Claim 18, $E^{(t-1)}$ is of the form $c_{\ell_{m-t+1}} \cdot 2^{x_{\ell_{m-t+1}}} + \sum_{k=0}^{m-t} a_{\ell_k} \cdot 2^{x_{\ell_k}}$, and $[\![\sigma]\!](E^{(t-1)}) = [\![\sigma]\!](E)$. Since the tth iteration executes the **break** statement, we have:

1. $M(\ell_{m-t+1}, \ell_{m-t}) = +\infty$ (from the condition of the **if** statements of line 10),
2. $c_{\ell_{m-t+1}} \neq 0$ (from the condition of the **if** statement of line 15).

We show that

$$2^{[\![\sigma]\!](x_{\ell_{m-t+1}})} > \left|\sum_{k=0}^{m-t} a_{\ell_k} \cdot 2^{[\![\sigma]\!](x_{\ell_k})}\right| + d(\sigma) \cdot C. \tag{27}$$

From the definition of $E^{(t-1)}$, and the fact that $[\![\sigma]\!](E^{(t-1)}) = [\![\sigma]\!](E)$ and $c_{\ell_{m-t+1}} \neq 0$, this inequality implies Claim 19.

To prove Equation (27), we first note that $M(\ell_{m-t+1}, \ell_{m-t}) = +\infty$ implies, from the induction hypothesis in the main proof, that $[\![\sigma]\!](x_{\ell_{m-t+1}}) > [\![\sigma]\!](x_{\ell_{m-t}}) + C$. Also, by definition of the indices $\ell_0, \ldots, \ell_m$, $[\![\sigma]\!](x_{\ell_{m-t}}) \geq [\![\sigma]\!](x_{\ell_k}) \geq 0$ for every $k \in [0..m-t-1]$. Hence, $2^{[\![\sigma]\!](x_{\ell_{m-t+1}})} > 2^C \cdot 2^{[\![\sigma]\!](x_{\ell_{m-t}})}$ and $\left(\sum_{k=0}^{m-t} |a_{\ell_k}|\right) \cdot 2^{[\![\sigma]\!](x_{\ell_{m-t}})} \geq \left|\sum_{k=0}^{m-t} a_{\ell_k} \cdot 2^{[\![\sigma]\!](x_{\ell_k})}\right|$, which in turn implies that Equation (27) holds as soon as we prove $2^C \geq \left(\sum_{k=0}^{m-t} |a_{\ell_k}|\right) + d(\sigma) \cdot C$. To show this inequality, we establish that $2^{C/2} \geq \sum_{k=0}^{m-t} |a_{\ell_k}|$ and $C \geq 4 \cdot \log_2(d(\sigma)) + 8$; the inequality then follows from Lemma 10. We have

$$\begin{aligned}
&2 \cdot \left\lceil \log_2(\max(1, \textstyle\sum_{k=0}^{m-t} |a_{\ell_k}|)) \right\rceil + 4 \left\lceil \log_2(d(\sigma)) \right\rceil + 8 \\
&\leq 2 \cdot \left\lceil \log_2(n \cdot 2^{n+1} \cdot e(\sigma) \cdot d(\sigma)) \right\rceil + 4 \left\lceil \log_2(d(\sigma)) \right\rceil + 8 && \text{\{bound on each } a_{\ell_k}\text{\}} \\
&\leq 2 \cdot (\underline{n+1} + \underline{\left\lceil \log_2(n) \right\rceil} + \left\lceil \log_2(e(\sigma)) \right\rceil + \underline{\left\lceil \log_2(d(\sigma)) \right\rceil}) + 4 \cdot \underline{\left\lceil \log_2(d(\sigma)) \right\rceil} + 8 \\
&\leq 8 \cdot (\text{bit size of } \sigma) + 8 = C. && \text{\{each underlined quantity is} \leq \text{(bit size of } \sigma)\text{\}}
\end{aligned}$$

Therefore, both $2^{C/2} \geq \sum_{k=0}^{m-t} |a_{\ell_k}|$ and $C \geq 4 \cdot \log_2(d(\sigma)) + 8$ hold. □

This completes the proof of Lemma 28. □

Lemma 1. $\text{NAT}_{\text{ILESLP}}$ *can be decided in polynomial time.*

Proof. Let $\sigma = (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$ be the input ILESLP of Algorithm 7.

Correctness: By line 20, the algorithm returns true if and only if $M(n, 0) \geq 0$. Recall that $[\![\sigma]\!](x_0) = 0$. It then follows from Lemma 28 that $M(n, 0) = T_C([\![\sigma]\!](x_n))$. Since $C > 0$, $T_C([\![\sigma]\!](x_n)) \geq 0$ if and only if $[\![\sigma]\!](x_n) \geq 0$. Therefore, $M(n, 0) \geq 0$ if and only if $[\![\sigma]\!](x_n) \geq 0$; showing the correctness of the algorithm.

Complexity: To prove that Algorithm 7 runs in polynomial time, observe that:

- The bit length of C is bounded logarithmically in the bit size of σ.
- The expressions $E_0, \ldots, E_n$ are computed in polynomial time following Lemma 27.
- The map M requires only $O(n^2 \log_2 C)$ space.
- All **for** loops (lines 4, 7 and 9) iterate on intervals of size linear in the bit size of σ.

Following these observations, the only remaining crucial point is showing that repeated executions of line 10, locally to one iteration of the **for** loop of line 7, do not cause the integers in the expression E to grow superpolynomially. This property is already established in Claim 18 of the proof of Lemma 28. In particular, the absolute value of each integer in E is bounded by $2^{nC+n+2} \cdot e(\sigma) \cdot d(\sigma)$; the bit length of this number is polynomial in the bit size of σ. With this key observation, it follows that all remaining operations performed by the algorithm run in polynomial time. □

Algorithm 8 A $\textsc{FP}^{\textsc{factoring}}$ algorithm for computing $[\![\sigma]\!]_\bullet \bmod g$.

Input: ILESLP $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$, and $g \in \mathbb{N}_{\geq 1}$ encoded in binary.

1: $M \leftarrow$ empty map from $[0..n]^2$ to $\mathbb{N}$
2: **for** k form 0 to n **do** $M(0,k) \leftarrow 0$
3: **for** i from 1 to n **do**
4: **let** $a_0, \ldots, a_{i-1} \in \mathbb{Z}$ such that $[\![\sigma]\!](x_i) = \frac{\sum_{j=0}^{i-1} a_j \cdot 2^{[\![\sigma]\!](x_j)}}{d(\sigma)}$
5: **for** k from 0 to $(n-i)$ **do**
6: $h \leftarrow \nu_\sigma^k(g) \cdot d(\sigma)$ ▷ *requires factorization oracle*
7: **let** $m, q \in \mathbb{N}$ such that $q = odd(h)$ and $h = 2^m \cdot q$
8: **for** j from 0 to $i-1$ **do**
9: $b \leftarrow$ **if** $[\![\sigma]\!](x_j) \geq m$ **then** 0 **else** $2^{[\![\sigma]\!](x_j)}$ ▷ *uses the algorithm for* $\textsc{Nat}_{\textsc{ileslp}}$
10: $c \leftarrow 2^{M(j,k+1)} \bmod q$
11: **let** r_j be the (only) value in $[0..h-1]$ such that 2^m divides $r_j - b$, and q divides $r_j - c$
12: $M(i,k) \leftarrow (\frac{1}{d(\sigma)} \sum_{j=0}^{i-1} a_j \cdot r_j) \bmod \nu_\sigma^k(g)$
13: **return** $M(n,0)$

8 Deciding $\textsc{Div}_{\textsc{ileslp}}$ in $\textsc{P}^{\textsc{factoring}}$

$\textsc{Div}_{\textsc{ileslp}}$	
Input:	An ILESLP σ, and $g \in \mathbb{N}_{\geq 1}$ encoded in binary.
Question:	Is $[\![\sigma]\!]_\bullet$ divisible by g ?

We describe a procedure that, given an ILESLP $\sigma = (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$ and $g \in \mathbb{N}_{\geq 1}$ encoded in binary, outputs (the binary encoding of) the remainder of $[\![\sigma]\!]_\bullet$ modulo g. The decision problem $\textsc{Div}_{\textsc{ileslp}}$ is solved by checking if the remainder in output is 0. The pseudocode of this procedure is shown in Algorithm 8.

We denote by $\nu_\sigma \colon \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ the function $\nu_\sigma(x) \coloneqq \phi(odd(x \cdot d(\sigma)))$, where $odd(a)$ denotes the largest odd factor of $a \in \mathbb{N}_{\geq 1}$, and ϕ denotes Euler's totient function (see Equation (23) on page 59 for the formal definition). We denote by ν_σ^k the kth iterate of ν_σ, that is, $\nu_\sigma^0(x) \coloneqq x$ and $\nu_\sigma^{k+1}(x) \coloneqq \nu_\sigma(\nu_\sigma^k(x))$ for every $k \in \mathbb{N}$.

Algorithm 8 constructs a map M with the following property: for every $i, k \in [0..n]$ with $i + k \leq n$, the entry $M(i,k)$ stores the value of $[\![\sigma]\!](x_i) \bmod \nu_\sigma^k(g)$. Once the map is opportunely populated, it returns the value $M(n,0)$ corresponding to $[\![\sigma]\!]_\bullet \bmod g$ (line 13).

The core of the algorithm is the **for** loop of line 3. During its ith iteration, this loop populates the entries $M(i,0), \ldots, M(i,n-i)$. (The base case of $i = 0$ is handled in line 2, as $[\![\sigma]\!](x_0) = 0$.) Similarly to Algorithm 7 and following Lemma 27, in line 4 the algorithm "flattens" the expression associated to x_i into one of the form $\frac{1}{d(\sigma)} \sum_{j=0}^{i-1} a_j \cdot 2^{x_j}$, with $a_1, \ldots, a_{i-1}$ integers. Let $h \coloneqq \nu_\sigma^k(g) \cdot d(\sigma)$.

To compute $M(i,k)$ (during the $(k+1)$th iteration of the loop of line 5) we reason modulo $\nu_\sigma^k(g)$:

$$\begin{aligned} M(i,k) &= \tfrac{1}{d(\sigma)} \cdot \textstyle\sum_{j=0}^{i-1} a_j \cdot 2^{[\![\sigma]\!](x_j)} \\ &= \tfrac{1}{d(\sigma)} \cdot \textstyle\sum_{j=0}^{i-1} a_j \cdot (2^{[\![\sigma]\!](x_j)} \bmod h), \end{aligned}$$

where the last number is an integer, because the sum $\sum_{j=0}^{i-1} a_j \cdot 2^{[\![\sigma]\!](x_j)}$ is divisible by $d(\sigma)$ (as σ is an ILESLP). To compute $2^{[\![\sigma]\!](x_j)} \bmod h$, we appeal to the Chinese Remainder Theorem (CRT).

$\nu_\sigma^0(g) = g$	$\nu_\sigma(g)$	$\nu_\sigma^2(g)$	$\nu_\sigma^3(g)$	$\nu_\sigma^4(g)$	
0	0	0	0	0	x_0
$[\![\sigma]\!]\, x_1 \bmod g$	$[\![\sigma]\!]\, x_1 \bmod \nu_\sigma(g)$	$[\![\sigma]\!]\, x_1 \bmod \nu_\sigma^2(g)$	$[\![\sigma]\!]\, x_1 \bmod \nu_\sigma^3(g)$	$\times$	x_1
$[\![\sigma]\!]\, x_2 \bmod g$	$[\![\sigma]\!]\, x_2 \bmod \nu_\sigma(g)$	$[\![\sigma]\!]\, x_2 \bmod \nu_\sigma^2(g)$	$\times$	$\times$	x_2
$-$	$-$	$\times$	$\times$	$\times$	x_3
$-$	$\times$	$\times$	$\times$	$\times$	x_4

Figure 3: Illustration of how the matrix M looks like after 2 iterations of the loop of line 3 (in the case of $n = 4$). To fill the next row, we only need the values in the rows above it. Entries with $\times$ are left empty in the algorithm as they are not needed to perform the computation of $[\![\sigma]\!]\,(x_n) \bmod g$.

Write $h = 2^m \cdot q$ with $m, q \in \mathbb{N}$ and q odd. Compute $b \coloneqq 2^{[\![\sigma]\!](x_j)} \bmod 2^m$ and $c \coloneqq 2^{[\![\sigma]\!](x_j)} \bmod q$, and then use CRT to find the only $r \in [0..h-1]$ such that $r \equiv b \bmod 2^m$, and $r \equiv c \bmod q$ (line 11).

The value b is computed via Algorithm 7 (line 9). To compute c, we see that $2^{\phi(q)} \bmod q = 1$, by Euler's theorem. Therefore, $2^{[\![\sigma]\!](x_j)}$ and $2^{[\![\sigma]\!](x_j) \bmod \phi(q)}$ have the same reminder modulo q. We have $\phi(q) = \nu_\sigma^{k+1}(g)$ by definition, and so $M(j, k+1) = [\![\sigma]\!]\,(x_j) \bmod \phi(q)$. Note that $j < i$, and so $M(j, k+1)$ has been populated in a previous iteration of the loop of line 3; the algorithm thus constructs c by computing $2^{M(j,k+1)} \bmod q$ (line 10).

Regarding the complexity of Algorithm 8, most of its operations can be implemented in polynomial time. The map M requires polynomial space, since $\nu_\sigma^k(g)$ is bounded by $d(\sigma)^k \cdot g$. Moreover, while $2^{M(j,k+1)}$ can in principle be of exponential bit size, computing it modulo q can be done in polynomial time in the bit size of $M(j, k+1)$ and q by relying on the exponentiation-by-squaring method [BW08, Ch. 1.4]. The only difficulty stems from the computation of $\nu_\sigma^k(g)$. For this we use the integer factorization oracle in order to compute Euler's totient function, following Equation (23).

We now formalize the above arguments into a full proof of correctness and runtime analysis.

Lemma 2. $\text{DIV}_{\text{ILESLP}}$ *is in* $\text{P}^{\text{FACTORING}}$.

Proof. Consider as input an ILESLP $\sigma = (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$ and $g \in \mathbb{N}_{\geq 1}$. We show that Algorithm 8 computes $[\![\sigma]\!]_\bullet \bmod g$, and runs in polynomial time with a factoring oracle. For simplicity of the exposition, let us see the map M as an $(n+1) \times (n+1)$ matrix over $\mathbb{N}$, with indices for rows and columns in $[0..n]$. The matrix is initially empty, and the algorithm only populates it in a "triangular way", only filling the entries (i, k) such that $i + k \leq n$. Upon completion of the algorithm, we will have $M(i, k) = ([\![\sigma]\!]\, x_i \bmod \nu_\sigma^k(g))$, for every such entry (i, k). Figure 3 depicts this matrix.

Since $\phi(a) \leq a$ for all $a \in \mathbb{N}_{\geq 1}$, we have $\nu_\sigma^k(g) \leq d(\sigma)^k g$. As a result, the number of bits required to store M is in $O(n^2 \log_2(d(\sigma)^n g))$. We prove by induction on $i \in \mathbb{N}$ that, during the i-th iteration of the **for** loop of line 3, the algorithm correctly fills the matrix representing M up to the i-th row, and does so in polynomial time, assuming access to a factoring oracle.

base case: $i = 0$**.** (i.e., before the loop of line 3 starts), the 0-th row of the matrix is already populated with all 0s (line 2).

induction hypothesis. The first $(i-1)$th rows of the matrix M are correctly populated. That is, for each $j \in [0..i-1]$ and $k \in [0..n-j]$, we have $M(j, k) = [\![\sigma]\!]\, x_k \bmod \nu_\sigma^j(g)$.

induction step: $i \geq 1$**.** The ith iteration of the loop handles the variable x_i. Line 4 computes integers $a_0, \ldots, a_{i-1}$ such that $[\![\sigma]\!]\,(x_i) = \frac{1}{d(\sigma)} \sum_{j=0}^{i-1} a_j 2^{[\![\sigma]\!](x_j)}$. Following Lemma 27, this computation can be performed in polynomial time. Next, for each $k \in [0..n-i]$, the inner

for loop of line 5 computes $[\![\sigma]\!](x_i) \bmod \nu_\sigma^k(g)$, storing the in $M(i,k)$ (see line 12). In order to compute $[\![\sigma]\!](x_i) \bmod \nu_\sigma^k(g)$, the algorithm uses the identity

$$[\![\sigma]\!](x_i) \bmod \nu_\sigma^k(g) \;=\; \left(\frac{1}{d(\sigma)} \cdot \sum\nolimits_{j=0}^{i-1} a_j \cdot \left(2^{[\![\sigma]\!](x_j)} \bmod \nu_\sigma^k(g) \cdot d(\sigma)\right)\right) \bmod \nu_\sigma^k(g). \tag{28}$$

We discuss the $(k+1)$th iteration of the inner **for** loop of line 5, analyzing it line by line.

line 6. The algorithm computes $\nu_\sigma^k(g)$. The integer factorization oracle is used to factorize the arguments of Euler's totient function, to then compute the result of this function via Equation (23). Since $\nu_\sigma^k(g)$ is bounded by $d(\sigma)^k g$, the computation is polynomial time except the oracle calls. The value $h = \nu_\sigma^k(g) \cdot d(\sigma)$ is then computed in polynomial time.

line 7. The algorithm factorizes h as $h = 2^m \cdot q$, where $q \in \mathbb{N}$ is the largest odd divisor of h. This is done by repeatedly dividing h by 2 until it becomes odd to compute q; which takes polynomial time. Note that 2^m and q are coprime.

The values m and q will be used to compute the value $r_j \coloneqq 2^{[\![\sigma]\!](x_j)} \bmod h$, following Equation (28), in the $(j+1)$th iteration of the inner **for** loop of line 8. To do this, we first compute $b \coloneqq 2^{[\![\sigma]\!](x_j)} \bmod 2^m$ and $c \coloneqq 2^{[\![\sigma]\!](x_j)} \bmod q$. Then, by the CRT, we compute the unique $r_j \in [0..h-1]$ such that $r_j \equiv b \bmod 2^m$ and $r_j \equiv c \bmod q$.

We analyse lines 9–11 locally to the $(j+1)$th iteration of the **for** loop of line 9 ($j \in [0..i-1]$).

line 9. This line computes $b = 2^{[\![\sigma]\!](x_j)} \bmod 2^m$. If $[\![\sigma]\!](x_j) \geq m$ then clearly $b = 0$, and otherwise $b = 2^{[\![\sigma]\!](x_j)}$. The comparison $[\![\sigma]\!](x_j) \geq m$ can be performed in polynomial time using Algorithm 7. Moreover, when we set $b = 2^{[\![\sigma]\!](x_j)}$, its value is less than 2^m, so its bit size remains polynomial in the input size.

line 10. This line computes $c = 2^{[\![\sigma]\!](x_j)} \bmod q$. First, recall that Euler's theorem states that if two numbers a and q are coprime, then $a^{\phi(q)}$ is congruent to 1 modulo q. Since q is odd, we thus have $c = 2^{[\![\sigma]\!](x_j) \bmod \phi(q)} \bmod q$. From the induction hypothesis, we have already set $M(j,k+1)$ to $[\![\sigma]\!](x_j) \bmod \phi(q)$ in previous iterations of the loop of line 3. Therefore, to compute c it only remains to compute $2^{M(j,k+1)} \bmod q$ in polynomial time; which can be done by relying on the exponentiation-by-squaring method [BW08, Ch. 1.4].

line 11. Given b and c, the algorithm replies on (a constructive version of) the CRT to compute r_j in polynomial time. More precisely, for this computation one can use the extended Euclidean algorithm to compute two integers ℓ_1 and ℓ_2 such that $\ell_1 \cdot 2^m + \ell_2 \cdot q = 1$ (ℓ_1 and ℓ_2 exist by Bézout's identity). Then, $r_j \coloneqq (b \cdot \ell_2 \cdot q + c \cdot \ell_1 \cdot 2^m) \bmod h$.

When the **for** loop of line 8 ends, the algorithm has computed r_j for every $j \in [0..i-1]$. Line 12 executes, populating $M(i,k)$ in polynomial time following the formula in Equation (28).

From the analysis above, we conclude that the algorithm is correct. Regarding the running time, note that each loop in the procedure iterates only over natural numbers bounded by n (which is bounded by the bit size of σ). Furthermore, all other operations performed by the algorithm run in polynomial time, except for line 6, which relies on the integer factorization oracle to compute $\nu_\sigma^k(g)$. So, Algorithm 8 runs in polynomial time with a factoring oracle, and $\textsc{Div}_{\textsc{ileslp}}$ is in $\mathrm{P}^{\textsc{factoring}}$. □

We can avoid appealing to the factorization oracle by providing Algorithm 8 with the set

$$\mathbb{P}(\sigma, g) \coloneqq \{p \text{ prime} : p \text{ divides either } d(\sigma) \text{ or } \nu_\sigma^k(g), \text{ for some } k \in [0..n-2]\}. \tag{29}$$

Algorithm 9 An FP$^{\text{FACTORING}}$ algorithm for computing x mod 2^y.

Input: ILESLP $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$, and two variables x and y in σ.

1: **if** $[\![\sigma]\!](y) \leq 0$ **then return** the ILESLP $(x_0 \leftarrow 0)$
2: **let** $a_0, \ldots, a_{n-1} \in \mathbb{Z}$ such that $[\![\sigma]\!](x) = \frac{\sum_{j=0}^{n-1} a_j \cdot 2^{[\![\sigma]\!](x_j)}}{d(\sigma)}$
3: $I \leftarrow \{j \in [0..n-1] : [\![\sigma]\!](x_j) < [\![\sigma]\!](y)\}$ ▷ *each comparison resolved with Algorithm 7*
4: **let** $S = \sum_{j \in I} a_j \cdot 2^{x_j}$ and $L = \sum_{j \in [0..n-1] \setminus I} a_j \cdot 2^{x_j}$
5: $A \leftarrow \sum_{j \in I} |a_j|$
6: Perform binary search to find $q \in [-A..A]$ satisfying $0 \leq [\![\sigma]\!](S) - q \cdot 2^{[\![\sigma]\!](y)} < 2^{[\![\sigma]\!](y)}$ ▷ *each iteration of binary search uses Algorithm 7*
7: **let** r be the residue of $[\![\sigma]\!](L) \cdot 2^{-[\![\sigma]\!](y)} + q$ modulo $d(\sigma)$ ▷ *uses Algorithm 8*
8: **return** an ILESLP ξ such that $[\![\xi]\!]_\bullet = \frac{1}{d(\sigma)} \cdot \left([\![\sigma]\!](S) + (r-q) \cdot 2^{[\![\sigma]\!](y)} \right)$

Observe that this set only contains polynomially many primes of polynomial bit size with respect to the sizes of σ and g, since $\nu_\sigma^k(g) \leq d(\sigma)^k \cdot g$.

Lemma 29. *Algorithm 8 runs in polynomial time, when provided with the set $\mathbb{P}(\sigma, g)$ (or any superset of this set) as an additional input.*

Proof. Following the proof of Lemma 2, the only line that requires the factoring oracle is line 6. In particular, this line asks to compute $\nu_\sigma^k(g)$, for a value of k that ranges in $[0..n-1]$. By induction on $k \in [0..n-1]$, we show that knowing $\mathbb{P}(\sigma, g)$ suffices to compute this number in polynomial time.

base case: $k = 0$**.** Since $\nu_\sigma^0(g) = g$, and g is part of the input, this case is trivial.

induction hypothesis. For $k \geq 1$, the positive integer $\nu_\sigma^{k-1}(g)$ can be computed in polynomial time in the sizes of σ and g, by relying on $\mathbb{P}(\sigma, g)$.

induction step: $k \geq 1$**.** By induction hypothesis, we can compute $\nu_\sigma^{k-1}(g)$ in polynomial time. Observe that $k-1 \leq n-2$, and therefore all prime divisors of $\nu_\sigma^{k-1}(g)$ occur in $\mathbb{P}(\sigma, g)$. Recall that this set has size polynomial in the sizes of σ and g. We iterate through $\mathbb{P}(\sigma, g)$ in order to find all prime divisors of $\nu_\sigma^{k-1}(g)$, as well as those of $d(\sigma)$. With this set of primes at hand, we can efficiently compute the prime factorization of $odd(\nu_\sigma^{k-1}(g) \cdot d(\sigma))$. Finally, we compute $\nu_\sigma^k(g) = \phi(odd(\nu_\sigma^{k-1}(g) \cdot d(\sigma)))$ using Equation (23). □

9 Computing an ILESLP representing x mod 2^y

Computation of x mod 2^y	
Input:	An ILESLP σ and two of its variables x and y.
Output:	An ILESLP ξ such that $[\![\xi]\!]_\bullet = [\![\sigma]\!](x) \bmod 2^{[\![\sigma]\!](y)}$.

Algorithm 9 describes a procedure for solving the above problem. It builds on Algorithms 7 and 8, inheriting a polynomial running time given either a factoring oracle or access to the set $\mathbb{P}(\sigma, \nu_\sigma(1))$.

Lemma 30. *Given an ILESLP σ and two of its variables x and y, Algorithm 9 returns an ILESLP ξ such that $[\![\xi]\!]_\bullet = [\![\sigma]\!](x) \bmod 2^{[\![\sigma]\!](y)}$. The algorithm runs in polynomial time with a factoring oracle.*

Proof. We analyze the runtime and correctness of the algorithm line by line, providing the underlying intuition throughout. Below, let $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$. Recall that $d(\sigma)$ is a positive integer.

line 1. Over the reals, given $a, m \in \mathbb{R}$ with $m \neq 0$, $(a \bmod m)$ is defined as $a - m \cdot \lfloor \frac{a}{m} \rfloor$. In particular, $(a \bmod 2^\ell) = 0$ whenever $\ell \leq 0$. Accordingly, line 1 checks whether $[\![\sigma]\!](y) \leq 0$, and in that case the algorithm returns an ILESLP that encodes the number 0. This line can be implemented in polynomial time by appealing to Algorithm 7. Below, assume $[\![\sigma]\!](y) \geq 1$.

line 2. As done in Algorithms 7 and 8, this line computes in polynomial time (Lemma 27) an expression $E \coloneqq \sum_{j=0}^{n-1} a_j \cdot 2^{x_j}$, where $a_0, \ldots, a_{n-1} \in \mathbb{Z}$, and $[\![\sigma]\!](x) = \frac{E}{d(\sigma)}$.

lines 3 and 4. The algorithm sorts the monomial $a \cdot 2^z$ in the expression E depending on the comparison $[\![\sigma]\!](z) < [\![\sigma]\!](y)$. This is done by computing the set $I \subseteq [0..n-1]$ of indices j of variables x_j such that $[\![\sigma]\!](x_j) < [\![\sigma]\!](y)$. Then, E can be rearranged as $L + S$, where $S \coloneqq \sum_{j \in I} a_j \cdot 2^{x_j}$ contains the exponentials 2^z that are "small" comparatively to $2^{[\![\sigma]\!](y)}$, and $L \coloneqq \sum_{j \in [0..n-1] \setminus I} a_j \cdot 2^{x_j}$ contains those that are "large". In particular, $2^{[\![\sigma]\!](y)}$ divides $[\![\sigma]\!](L)$. The set I can be constructed in polynomial time, by appealing n times to Algorithm 7.

line 5. This line computes (in polynomial time) $A \coloneqq \sum_{j \in I} |a_j|$. Observe that:

$$|[\![\sigma]\!](S)| = \sum\nolimits_{j \in I} |a_j| \cdot 2^{[\![\sigma]\!](x_j)} \leq A \cdot 2^{[\![\sigma]\!](y)}. \tag{30}$$

line 6. This line computes the quotient q of the division of $[\![\sigma]\!](S)$ by $2^{[\![\sigma]\!](y)}$; formally, the only integer satisfying $0 \leq [\![\sigma]\!](S) - q \cdot 2^{[\![\sigma]\!](y)} < 2^{[\![\sigma]\!](y)}$. By Equation (30), we know that $q \in [-A..A]$. Since A has bit size polynomial in the size of σ, we can compute q in polynomial time by performing binary search on the interval $[-A..A]$, appealing to Algorithm 7. For the sake of completeness, let us briefly explain how the search is implemented. Suppose knowing that the required q belongs to $[\ell..u]$, where $l, u \in \mathbb{Z}$. Initially, $[\ell..u] = [-A..A]$. Let $v \coloneqq \lceil \frac{\ell+u}{2} \rceil$. Then,

- If $[\![\sigma]\!](S) - v \cdot 2^{[\![\sigma]\!](y)} < 0$, then we can restrict the search to $[\ell..v]$.
- If $[\![\sigma]\!](S) - v \cdot 2^{[\![\sigma]\!](y)} > 2^{[\![\sigma]\!](y)}$, then we can restrict the search to$[v..u]$.
- If none of the previous two cases hold, then v is the required q.

The conditions in the first two cases above are checked in polynomial time using Algorithm 7. Specifically, by following the operations in the expression $S - v \cdot 2^y$, it is simple to extend the ILESLP σ into a new ILESLP σ' such that $[\![\sigma']\!]_\bullet = [\![\sigma]\!](S) - v \cdot 2^{[\![\sigma]\!](y)}$. One can then apply Algorithm 7 to σ' to check the first of the two cases (the second case is handled similarly).

From the definition of the expression E, we have:

$$[\![\sigma]\!](x) = \frac{[\![\sigma]\!](E)}{d(\sigma)} = \frac{[\![\sigma]\!](L) + [\![\sigma]\!](S)}{d(\sigma)} = \frac{([\![\sigma]\!](L) + q \cdot 2^{[\![\sigma]\!](y)}) + ([\![\sigma]\!](S) - q \cdot 2^{[\![\sigma]\!](y)})}{d(\sigma)}. \tag{31}$$

line 7. This line computes the residue r of $[\![\sigma]\!](L) \cdot 2^{-[\![\sigma]\!](y)} + q$ modulo $d(\sigma)$. This is done by constructing, in polynomial time, an ILESLP σ' encoding $[\![\sigma]\!](L) \cdot 2^{-[\![\sigma]\!](y)} + q$, and then calling Algorithm 8 on σ' and $d(\sigma)$. Thus, this line can be implemented in polynomial time with access to the factoring oracle —this is the only line of the algorithm requiring the oracle. To construct σ', recall that the expression $L = \sum_{j \in [0..n-1] \setminus I} a_j \cdot 2^{x_j}$ is such that $[\![\sigma]\!](x_j) \geq [\![\sigma]\!](y)$ for every $j \in [0..n-1] \setminus I$. By following the operations in the expression $L' \coloneqq \sum_{j \in [0..n-1] \setminus I} a_j \cdot 2^{x_j - y}$,

we can extend σ into an ILESLP σ'' such that $[\![\sigma'']\!]_\bullet = [\![\sigma]\!](L') = [\![\sigma]\!](L) \cdot 2^{-[\![\sigma]\!](y)}$. Finally, σ'' can be further extended to produce the desired σ'.

Following Equation (31), we see that:

$$\begin{aligned}
[\![\sigma]\!](x) &= \frac{([\![\sigma]\!](L) + q \cdot 2^{[\![\sigma]\!](y)}) + ([\![\sigma]\!](S) - q \cdot 2^{[\![\sigma]\!](y)})}{d(\sigma)} \\
&= \frac{([\![\sigma]\!](L) + q \cdot 2^{[\![\sigma]\!](y)} - r \cdot 2^{[\![\sigma]\!](y)}) + ([\![\sigma]\!](S) - q \cdot 2^{[\![\sigma]\!](y)} + r \cdot 2^{[\![\sigma]\!](y)})}{d(\sigma)} \\
&= \frac{[\![\sigma]\!](L) + q \cdot 2^{[\![\sigma]\!](y)} - r \cdot 2^{[\![\sigma]\!](y)}}{d(\sigma)} + \frac{[\![\sigma]\!](S) - q \cdot 2^{[\![\sigma]\!](y)} + r \cdot 2^{[\![\sigma]\!](y)}}{d(\sigma)} \\
&= \frac{[\![\sigma]\!](L') + q - r}{d(\sigma)} \cdot 2^{[\![\sigma]\!](y)} + \frac{[\![\sigma]\!](S) + (r-q) \cdot 2^{[\![\sigma]\!](y)}}{d(\sigma)}. \qquad (32)
\end{aligned}$$

By definition of r, $\frac{[\![\sigma]\!](L')+q-r}{d(\sigma)}$ is an integer. Then, since $[\![\sigma]\!](x)$ and $[\![\sigma]\!](y)$ are both integers, and the latter is positive, Equation (32) shows that $\ell \coloneqq \frac{[\![\sigma]\!](S)+(r-q)\cdot 2^{[\![\sigma]\!](y)}}{d(\sigma)}$ is an integer.

line 8. From Equation (32), we conclude that $[\![\sigma]\!](x) \bmod 2^{[\![\sigma]\!](y)} = \ell \bmod 2^{[\![\sigma]\!](y)}$. We will now show that $\ell \in [0..2^{[\![\sigma]\!](y)} - 1]$, which implies that ℓ is in fact $[\![\sigma]\!](x) \bmod 2^{[\![\sigma]\!](y)}$. Accordingly, line 8 of the algorithm constructs (and returns) an ILESLP ξ encoding ℓ. Clearly, ξ can be constructed in polynomial time by extending σ, following the operations in the expression $\frac{1}{d(\sigma)} \cdot \big(S + (r-q) \cdot 2^y\big)$. Recall that $([\![\sigma]\!](S) - q \cdot 2^{[\![\sigma]\!](y)}) \in [0..2^{[\![\sigma]\!](y)} - 1]$ and $r \in [0..d(\sigma) - 1]$, by definition of q and r, respectively. Then, $0 \leq [\![\sigma]\!](S) - q \cdot 2^{[\![\sigma]\!](y)} \leq [\![\sigma]\!](S) + (r-q) \cdot 2^{[\![\sigma]\!](y)}$, and $[\![\sigma]\!](S) + (r-q) \cdot 2^{[\![\sigma]\!](y)} < 2^{[\![\sigma]\!](y)} + r \cdot 2^{[\![\sigma]\!](y)} \leq d(\sigma) \cdot 2^{[\![\sigma]\!](y)}$. Therefore, by definition of ℓ, we conclude that $\ell \in [0..2^{[\![\sigma]\!](y)} - 1]$. □

Lastly, we show that $[\![\sigma]\!](x) \bmod 2^{[\![\sigma]\!](y)}$ is computable in polynomial time given $\mathbb{P}(\sigma, \nu_\sigma(1))$.

Lemma 31. *Algorithm 9 runs in polynomial time when provided $\mathbb{P}(\sigma, \nu_\sigma(1))$ as an additional input.*

Proof. As explained during the proof of Lemma 30, only line 7 requires the factoring oracle. This line requires computing the residue of $[\![\sigma]\!](L) \cdot 2^{-[\![\sigma]\!](y)} + q$ modulo $d(\sigma)$, where $L \coloneqq \sum_{j \in [0..n-1] \setminus I} a_j \cdot 2^{x_j}$ defined in line 4 is such that $[\![\sigma]\!](x_j) \geq [\![\sigma(y)]\!]$ for every $j \in [0..n-1] \setminus I$. Therefore,

$$[\![\sigma]\!](L) \cdot 2^{-[\![\sigma]\!](y)} + q \;=\; \sum\nolimits_{j \in [0..n-1] \setminus I} a_j \cdot 2^{[\![\sigma]\!](x_j) - [\![\sigma]\!](y)} + q.$$

Since all a_j and q have a bit size polynomial in the size of σ, it suffices to show how to compute $2^{[\![\sigma]\!](x_j) - [\![\sigma]\!](y)} \bmod d(\sigma)$ in polynomial time. The arguments are similar as those in Section 8.

Let $m \in \mathbb{N}$ be such that $d(\sigma) = 2^m \cdot \mathit{odd}(d(\sigma))$. By the CRT, $2^{[\![\sigma]\!](x_j) - [\![\sigma]\!](y)} \bmod d(\sigma)$ can be computed in polynomial time given the following two values:

$$b \coloneqq 2^{[\![\sigma]\!](x_j) - [\![\sigma]\!](y)} \bmod 2^m \quad \text{and} \quad c \coloneqq 2^{[\![\sigma]\!](x_j) - [\![\sigma]\!](y)} \bmod \mathit{odd}(d(\sigma)),$$

The value b can be computed in polynomial time by appealing to Algorithm 7. This is done as for the identically named value "b" in line 9 of Algorithm 8; see the proof of Lemma 2.

By definition, the set $\mathbb{P}(\sigma, \nu_\sigma(1))$ contains all prime factors of $d(\sigma)$. Therefore, we can compute $t \coloneqq \nu_\sigma(1) = \phi(\mathit{odd}(d(\sigma)))$ in polynomial time using Equation (23). For obtaining the value c, we then first derive the residue $r \coloneqq ([\![\sigma]\!](x_j) \bmod t)$ and the residue $s \coloneqq ([\![\sigma]\!](y) \bmod t)$ in polynomial time using Algorithm 8. Afterwards, $c = \big(2^{(r-s) \bmod t} \bmod \mathit{odd}(d(\sigma))\big)$ is computed in polynomial time using the exponentiation-by-squaring method. □

Part III
On the complexity of ILEP

We combine the results of the two previous parts of the paper to show Corollary 1, i.e., that the optimization problem for integer linear-exponential programs is in NPO-CMP. The first section of this part of the paper introduces the class NPO-CMP. The second section proves the corollary.

10 The complexity class NPO-CMP

We briefly recall the notion of an optimization problem. An *optimization problem* $\mathcal{P}$ is characterized by a quintuple $(I, U, \mathrm{sol}, m, \mathrm{goal})$ where:

- I is the set of instances of $\mathcal{P}$,
- U is a set (or, *universe*) containing all possible solutions,
- $\mathrm{sol}\colon I \to 2^U$ assigns each input instance $x \in I$ to the set of its solutions $\mathrm{sol}(x)$,
- $m\colon I \times U \rightharpoonup \mathbb{Z}$ is the *measure function*, a partial function defined for every $x \in I$ and $y \in \mathrm{sol}(x)$,
- $\mathrm{goal} \in \{\min, \max\}$ specifies a minimization or a maximization objective.

For $x \in I$, the set of *optimal solutions* of x is defined as

$$\mathrm{opt}(x) \coloneqq \{y \in \mathrm{sol}(x) : m(x, y) = \mathrm{goal}\{m(x, z) : z \in \mathrm{sol}(x)\}\}.$$

The computational task associated to $\mathcal{P}$ is the following:

Input: An instance $x \in I$.
Output: An element $y \in \mathrm{opt}(x)$ if $\mathrm{opt}(x) \neq \emptyset$, otherwise `reject`.

Below, we assume the elements of the sets I and U to be endowed with a notion of size $|\cdot|$. We define NPO-CMP as the class of all optimization problems $\mathcal{P} = (I, U, \mathrm{sol}, m, \mathrm{goal})$ such that:

1. The sets I and U are recognizable in polynomial time.
2. Given in input $x \in I$ and $y \in U$, checking $y \in \mathrm{sol}(x)$ is in P.
3. m is computable, and checking $m(x, y_1) \leq m(x, y_2)$, given $x \in I$ and $y_1, y_2 \in \mathrm{sol}(x)$, is in P.
4. There is a polynomial $q\colon \mathbb{N} \to \mathbb{N}$ such that, for all $x \in I$, $\mathrm{short}(x) \coloneqq \{y \in \mathrm{sol}(x) : |y| \leq q(|x|)\}$ satisfies: *(a)* if $\mathrm{sol}(x) \neq \emptyset$ then $\mathrm{short}(x) \neq \emptyset$, and *(b)* if $\mathrm{opt}(x) \neq \emptyset$ then $\mathrm{opt}(x) \cap \mathrm{short}(x) \neq \emptyset$.
5. Given an instance $x \in I$, deciding $\mathrm{sol}(x) \neq \emptyset \wedge \mathrm{opt}(x) = \emptyset$ is in NP.

It is worth noting that some authors prefer replacing Properties (4) and (5) above with the simpler

4'. There is a polynomial $q\colon \mathbb{N} \to \mathbb{N}$ such that $|y| \leq q(|x|)$ for every $x \in I$ and $y \in \mathrm{sol}(x)$,

which in particular implies the finiteness of $\mathrm{sol}(x)$ (see, e.g., the definition of NPO in [AMC+99]). The only difference between Properties (4) and (5) and Property (4') lies in whether only small solutions are considered: if an optimization problem $(I, U, \mathrm{sol}, m, \mathrm{goal})$ is in NPO-CMP, then the problem $(I, U, \mathrm{short}, m, \mathrm{goal})$, where short is the function required by Property (4), is also in NPO-CMP and

satisfies Property (4'). In other words, Property (4') reflects the idea that only polynomial size solutions are reasonable solutions. Our rationale for preferring the more wordy Properties (4) and (5) is that they provide a nice blueprint for organizing the results in the previous parts of the paper. This modest goal is indeed the main purpose behind the class NPO-cmp; as stated in the introduction, we make no presumption on the naturality of this class in a broader context.

Aside from the differences between Properties (4) and (5) and Property (4'), starting from the definition of NPO-cmp, one obtains the class NPO by replacing Property (3) with the stronger

3'. m is computable in polynomial time, assuming a binary encoding for the integer in output.

Therefore, every problem in NPO belongs to NPO-cmp.

Expanding on the discussion in Section 1.3, we see that, when $\mathcal{P}$ belongs to NPO, Properties (3') and (4') ensure that, for any input $x \in I$, one can compute in polynomial time two integers a and b such that for every (short) solution $y \in \mathrm{sol}(x)$ we have $m(x, y) \in [a..b]$. (Implicitly, this step assumes knowing the polynomial q in Property (4'), as well as a polynomial bounding the runtime of m.) One can then search for the optimal solution by performing binary search: at each iteration, the interval $[a..b]$ shrinks in half following the answer to the query $\exists y \in U : y \in \mathrm{sol}(x) \wedge m(x, y) \geq \frac{b-a}{2}$. By Properties (2), (3') and (4'), this query is solvable in NP. Using this approach, it follows that NPO problems can be solved by polynomial-time Turing machines with access to an NP oracle, that is, $\mathrm{NPO} \subseteq \mathrm{FP}^{\mathrm{NP}}$. (In fact, $\mathrm{NPO} = \mathrm{FP}^{\mathrm{NP}}$ for a suitable model of computation characterizing NPO, see [Kre88, CP89].)

In the case of NPO-cmp, Property (4b) ensures that $\{m(x, y) : y \in \mathrm{short}(x)\}$ is a set of exponentially many integers containing the optimal value for m (if one exists). However, NPO-cmp does not fix any representation on the integers returned by m (we only know that one such representation exists, since m is computable). Therefore, the size of these integers is unknown, and there is no guarantee that binary search can be performed on this set. Instead of an inclusion within $\mathrm{FP}^{\mathrm{NP}}$, we have $\text{NPO-cmp} \subseteq \mathrm{FNP}^{\mathrm{NP}}$. Indeed: a polynomial-time non-deterministic Turing machine with access to an NP oracle can solve an NPO-cmp problem in the following simple way:

1: Check that the input x belongs to I; if not, `reject` ▷ *In* P *by Property* (1).
2: Query the NP oracle to determine if $\mathrm{sol}(x) \neq \emptyset \wedge \mathrm{opt}(x) = \emptyset$ holds; if the answer is *yes*, `reject`
▷ *This query can be solved in* NP *by Property* (5).
3: Guess a string y of length $q(|x|)$, where q is the polynomial in Property (4)
4: Check $y \in U$ and $y \in \mathrm{sol}(x)$; if not, `reject` ▷ *In* P *by Properties* (1) *and* (2).
5: Query the NP oracle to determine if there exists $z \in \mathrm{short}(x)$ such that $m(x, z) > m(x, y)$ (assuming goal = max); if the answer is *yes*, `reject`
▷ *This query can be solved in* NP *because* $|z| \leq q(|x|)$*, and checking whether* $z \in \mathrm{sol}(x)$ *and* $m(x, z) > m(x, y)$ *can be done in polynomial time by Properties* (2) *and* (3).
6: **return** y

11 ILEP is in NPO-cmp

We now prove that the optimization problem for integer linear-exponential programs is in NPO-cmp (Corollary 1). Let us first define the objects I, U, sol and m, noting that goal is simply min or max:

- I is the set of all pairs (τ, φ) where τ is a linear-exponential term (the objective function) and φ is an integer linear-exponential program. The size $|(\tau, \varphi)|$ of $(\tau, \varphi) \in I$ is the sum of the sizes of τ and φ.
- $U \coloneqq \{(\sigma, \mathbb{P}(\sigma)) : \sigma \text{ is a ILESLP}\}$, where $\mathbb{P}(\sigma) \coloneqq \mathbb{P}(\sigma, d(\sigma) \cdot \nu_\sigma(1))$ and

– $d(\sigma)$ is the product of all denominators occurring in rational constants of scaling expressions in σ (as defined at the beginning of Part II);
– ν_σ is the function $\nu_\sigma(x) \coloneqq \phi(odd(x \cdot d(\sigma)))$, as defined in Section 8;
– $\mathbb{P}(\sigma, g)$ is the set of primes defined in Equation (29) on page 71.

The size $|(\sigma, \mathbb{P}(\sigma))|$ of $(\sigma, \mathbb{P}(\sigma)) \in U$ is the sum of the bit sizes of σ and $\mathbb{P}(\sigma)$.

- Given $(\tau, \varphi) \in I$, we define $\mathrm{sol}(\tau, \varphi)$ as the set of all $(\sigma, \mathbb{P}(\sigma)) \in U$ with the following property. Let $\sigma = (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$. Then,
  (a) the set $\{x_0, \ldots, x_n\}$ contains (at least) all variables in τ and in φ;
  (b) each variable x occurring in φ or τ is such that $[\![\sigma]\!](x) \geq 0$;
  (c) the map assigning to each x in φ the value $[\![\sigma]\!](x_i)$ is a solution of φ.
- Given $(\tau, \varphi) \in I$ and $(\sigma, \mathbb{P}(\sigma)) \in \mathrm{sol}(\tau, \varphi)$, we define $m((\tau, \varphi), \sigma)$ as the integer $\tau(\sigma)$ obtained by evaluating τ, replacing each variable x occurring in it with $[\![\sigma]\!](x)$.

Let us prove that these objects satisfy the five properties of NPO-CMP.

Property (1)**.** The set I is clearly recognizable in polynomial time. We show that the same is true for the set U —this is the content of Proposition 1 (Section 1.2):

Proposition 1. *Given an* LESLP σ *and* $\mathbb{P}(\sigma)$*, one can decide in polynomial time if* σ *is an* ILESLP*. In order words, the set* $U \coloneqq \{(\sigma, \mathbb{P}(\sigma)) : \sigma$ *is an* ILESLP$\}$ *is recognizable in polynomial time.*

Proof. Consider a pair (σ, S), where S is a set of positive integers, and $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$ is a LESLP (both objects are clearly recognizable in polynomial time). We first check that $S = \mathbb{P}(\sigma)$. Recall that, given $g \in \mathbb{N}_{\geq 1}$, $\mathbb{P}(\sigma, g) \coloneqq \{p \text{ prime} : p \text{ divides } d(\sigma) \text{ or } \nu_\sigma^k(g), \text{ for some } k \in [0..n-2]\}$; and $\mathbb{P}(\sigma) = \mathbb{P}(\sigma, d(\sigma) \cdot \nu_\sigma(1))$. Here, ν_σ^k stands for the kth iterate of the function ν_σ. To check $S = \mathbb{P}(\sigma)$, we first we use the polynomial time algorithm for primality testing [AKS04] to verify that all elements of S are primes. Afterwards, we check that these primes are exactly those appearing in the prime factorization of $d(\sigma)$ or of numbers of the form $\nu_\sigma^k(d(\sigma) \cdot \nu_\sigma(1))$, with $k \in [0..n-2]$. For this second step, the arguments are similar to those in the proof of Lemma 29. Below, we give the pseudocode of a polynomial time procedure preforming this step:

```
1: assert S contains all prime divisors of d(σ)
2: compute ν_σ(1) = φ(odd(d(σ))) by relying on the prime divisors of d(σ)   ▷ see Equation (23)
3: m_0 ← d(σ) · ν_σ(1)                                                      ▷ m_i = ν_σ^i(d(σ) · ν_σ(1))
4: for k from 0 to n − 2 do
5:     assert S contains all prime divisors of m_k
6:     if k ≠ n − 2 then
7:         compute ν_σ(m_k) = φ(odd(m_k · d(σ))) by relying on the prime divisors of d(σ) and m_k
8:         m_{k+1} ← ν_σ(m_k)
9: assert every prime in S divide ∏_{i=0}^{k−2} m_i
10: return true
```

After establishing $S = \mathbb{P}(\sigma)$, we determine whether the LESLP σ is actually an ILESLP. We recall the snippet of code from Section 1.2 that solves this problem:

```
1: for i = 1 to n do
2:     if ρ_i is of the form 2^x then assert [[σ]](x) ≥ 0
```

3: **if** ρ_i is of the form $\frac{m}{g} \cdot x$ **then assert** $\frac{g}{\gcd(m,g)}$ divides $[\![\sigma]\!](x)$
4: **return** true

By Lemma 1, the condition in the **assert** command of line 2 can be checked in polynomial time. To show that the same is true for the **assert** command of line 3, first observe that $\frac{g}{\gcd(m,g)}$ is a divisor of $d(\sigma)$. Then, the statement follows from Lemma 29, as soon as we show the following claim:

Claim 20. *for every $a, b \in \mathbb{N}_{\geq 1}$ such that a is a divisor of b, $\mathbb{P}(\sigma, a) \subseteq \mathbb{P}(\sigma, b)$.*

Proof of Claim 20. It suffices to show that $\nu_\sigma^k(a)$ divides $\nu_\sigma^k(b)$. The proof is by induction on k.

base case: $k = 0$**.** Since $\nu_\sigma^0(x) = x$, the statements follows trivially.

induction hypothesis. Given $k \geq 1$, $\nu_\sigma^{k-1}(a)$ divides $\nu_\sigma^{k-1}(b)$.

induction step: $k \geq 1$**.** By definition of *odd*, $odd(a \cdot d(\sigma))$ divides $odd(b \cdot d(\sigma))$. Moreover, one of the basic properties of Euler's totient function ϕ is that $\phi(g)$ divides $\phi(c)$ whenever g divides c (this follows directly from the definition of φ). Hence, $\nu_\sigma(a)$ divides $\nu_\sigma(b)$. By induction hypothesis, $\nu_\sigma^{k-1}(\nu_\sigma(a))$ divides $\nu_\sigma^{k-1}(\nu_\sigma(b))$; in other words, $\nu_\sigma^k(a)$ divides $\nu_\sigma^k(b)$. □

This concludes the proof of Proposition 1. □

Property (2)**.** We start with an auxiliary lemma which we will also use to show Property (3).

Lemma 32. *There is a polynomial time procedure with the following specification:*

Input: *$(\sigma, \mathbb{P}(\sigma)) \in U$ and a linear-exponential term τ featuring variables X from σ.*
Output: *An ILESLP ξ.*

Under the assumption that $[\![\sigma]\!](x) \geq 0$ for all $x \in X$, the algorithm ensures that $[\![\xi]\!]_\bullet$ is the integer obtained by evaluating τ by replacing all $x \in X$ with $[\![\sigma]\!](x)$.

Proof. Let $X = \{y_1, \ldots, y_\ell\}$, and τ be the term $\sum_{i=1}^{\ell} \left(a_i \cdot y_i + b_i \cdot 2^{y_i} + \sum_{j=1}^{\ell} c_{i,j} \cdot (y_i \bmod 2^{y_j})\right) + d$. Here is the pseudocode of the algorithm:

for $(i, j) \in [1..\ell] \times [1..\ell]$ **do**
  let $\xi_{i,j}$ be the ILESLP computed by Algorithm 9 on input $(\sigma, y_i, y_j, \mathbb{P}(\sigma))$
  Rename variables in $\xi_{i,j}$ to be distinct from those in σ;
  let $z_{i,j}$ be the variable in the last assignment of $\xi_{i,j}$ ▷ *the one encoding $[\![\xi_{i,j}]\!]_\bullet$*
  Extend σ by appending the assignments in $\xi_{i,j}$
return an ILESLP for the expression $\sum_{i=1}^{\ell} \left(a_i \cdot y_i + b_i \cdot 2^{y_i} + \sum_{j=1}^{\ell} c_{i,j} \cdot z_{i,j}\right) + d$

Each call to Algorithm 9 runs in polynomial time (Lemma 31 and Claim 20), and by Lemma 30 it produces an ILESLP $\xi_{i,j}$ with $[\![\xi_{i,j}]\!]_\bullet = ([\![\sigma]\!](y_i) \bmod 2^{[\![\sigma]\!](y_j)})$. Upon reaching line 6, the (augmented) ILESLP σ is of polynomial size, and contains not only the initial assignments, but also all those added by the **for** loop of line 1 —specifically, assignments to variables $z_{i,j}$ satisfying $[\![\sigma]\!](z_{i,j}) = ([\![\sigma]\!](y_i) \bmod 2^{[\![\sigma]\!](y_j)})$. The expression in line 6 involves $O(\ell^2)$ additions, exponentiations, and multiplications by integer constants. So, line 6 can be implemented in polynomial time by appending, to σ, a suitable sequence of assignments corresponding to these operations. □

The following proposition (first stated in Section 1.2) implies Property (2).

Proposition 2. *Checking whether $(\sigma, \mathbb{P}(\sigma)) \in U$ encodes a solution to an instance (τ, φ) of ILEP can be done in polynomial time in the bit sizes of σ and φ.*

Proof. Checking condition Item (a) in the definition of sol can clearly be done in polynomial time; let us write X for the variables occurring in φ or τ. For Item (b), given a variable $x \in X$, we can decide $[\![\sigma]\!](x) \geq 0$ in polynomial time by appealing to Algorithm 7 (see Lemma 1). For Item (c), we need to check whether the map ν assigning to each $x \in X$ the value $[\![\sigma]\!](x)$ satisfies φ. Let $\tau \leq 0$ be an inequality in φ (equalities $\tau = 0$ are treated analogously by viewing them as conjunctions $\tau \leq 0 \wedge -\tau \leq 0$). By Lemma 32, we can construct, in polynomial time, an ILESLP ξ such that $[\![\xi]\!]_\bullet$ is the integer $\tau(\sigma)$ obtained by evaluating τ on ν. Next, we append an assignment $y \leftarrow -1 \cdot x$ to ξ, where x is the variable in the last assignment of ξ (the one encoding $[\![\xi]\!]_\bullet$), and y is a fresh variable. Then, $\tau(\sigma) \leq 0$ if and only if $[\![\xi]\!]_\bullet \geq 0$, and we can check whether $[\![\xi]\!]_\bullet \geq 0$ in polynomial time via Algorithm 7. □

Property (3)**.** The map m is clearly computable. Consider an instance $(\tau, \varphi) \in I$ and two of its solutions $(\sigma_1, \mathbb{P}(\sigma_1)), (\sigma_2, \mathbb{P}(\sigma_2)) \in \mathrm{sol}(\tau, \varphi)$. An algorithm to decide $\tau(\sigma_1) \leq \tau(\sigma_2)$ is the following:

1: Construct a polynomial-size ILESLP ξ such that $[\![\xi]\!]_\bullet = \tau(\sigma_2) - \tau(\sigma_1)$
2: **return** true **if** $[\![\xi]\!]_\bullet \geq 0$ **else** false

The ILESLP ξ in line 1 is computed in polynomial time by relying on the algorithm in Lemma 32. The check $[\![\xi]\!]_\bullet \geq 0$ is performed in polynomial time by appealing to Algorithm 7.

Property (4)**.** By Theorem 1 (proven in Part I), if an instance $(\tau, \varphi) \in I$ has an (optimal) solution, then it has one representable with a polynomial-size ILESLP σ. We remark that our proof of this theorem is constructive, meaning that it allows one to explicitly derive a suitable monotonic polynomial h_1, such that, for every $(\tau, \varphi) \in I$, the corresponding ILESLP σ from Theorem 1 has (when it exists) size bounded by $h_1(|(\tau, \varphi)|)$. Moreover, the set of primes $\mathbb{P}(\sigma)$ has size polynomial in σ (see page 71, and note that the definition of $\mathbb{P}(\sigma)$ is constructive). Let h_2 be a monotonic polynomial bounding the size of $\mathbb{P}(\sigma)$ given the size of any ILESLP σ. Setting $q(x) \coloneqq h_1(x) + h_2(h_1(x))$ results in the polynomial required by Property (4): given an instance $(\tau, \varphi) \in I$, if an (optimal) solution exists, then there is $(\sigma, \mathbb{P}(\sigma)) \in \mathrm{sol}(\tau, \varphi)$ such that σ has size s bounded by $h_1(|(\tau, \varphi)|)$, and $\mathbb{P}(\sigma)$ has size bounded by $h_2(s) \leq h_1(h_1(|(\tau, \varphi)|))$. Then, $|(\sigma, \mathbb{P}(\sigma))| \leq q(|(\tau, \varphi)|)$.

Property (5)**.** Informally, this property asks for an NP procedure to check whether the input instance is *unbounded*, that is, it has an infinite sequence of solutions in which the value of the objective function strictly increases (assuming goal = max; the argument we give is analogous for goal = min). We reason similarly to how this property is proven in ILP. Let q and short be the polynomial and function from Property (4). From the above discussion, we have an explicit definition for q, and this polynomial is monotonic. Let $(\tau, \varphi) \in I$. We show that $\mathrm{sol}(\tau, \varphi) \neq \emptyset \wedge \mathrm{opt}(\tau, \varphi) = \emptyset$ holds if and only if the following integer linear-exponential program is feasible:

$$\varphi \wedge \tau \geq \|\tau\|_1 \cdot 2^{z_s} \wedge z_1 = 2^s \wedge \bigwedge\nolimits_{i=2}^{s} z_i = 2^{z_i}\,, \tag{33}$$

where $s \coloneqq q(|(\tau, \varphi)|) + 3$ and $z_1, \ldots, z_s$ are variables not occurring in φ or τ. The size of this linear-exponential program is polynomial in $|(\tau, \varphi)|$, and its feasibility can be decided in NP by Theorem 1 (or, alternatively, the original algorithm from [CMS24]); Property (5) follows. Below, let ξ be the ILESLP $\xi \coloneqq (z_{-1} \leftarrow 0,\ z_0 \leftarrow 2^{z_{-1}},\ z_1 \leftarrow 2^s \cdot z_0,\ z_2 \leftarrow 2^{z_1},\ \ldots,\ z_s \leftarrow 2^{z_s},\ z_{s+1} \leftarrow \|\tau\|_1 \cdot z_s)$, and observe that in any solution to Equation (33), the value taken by the term $\|\tau\|_1 \cdot 2^{z_s}$ is exactly $[\![\xi]\!]_\bullet$.

For the left-to-right direction of the double implication, supposes $\mathrm{sol}(\tau, \varphi) \neq \emptyset \wedge \mathrm{opt}(\tau, \varphi) = \emptyset$. As stated above, this means that there is an infinite sequence of solutions of φ in which the value of τ strictly increases. Therefore, there is a solution for which the value of τ exceeds $[\![\xi]\!]_\bullet$, and this implies the feasibility of Equation (33).

For the right-to-left direction, we consider the contrapositive. Assume that either $\mathrm{sol}(\tau,\varphi)=\emptyset$ or $\mathrm{opt}(\tau,\varphi)\neq\emptyset$. If $\mathrm{sol}(\tau,\varphi)=\emptyset$ then φ is infeasible and therefore so is Equation (33). If instead $\mathrm{opt}(\tau,\varphi)\neq\emptyset$, then by Property (4b) we have $\mathrm{opt}(\tau,\varphi)\cap\mathrm{short}(\tau,\varphi)\neq\emptyset$. To show that Equation (33) is infeasible, it suffices to show that for every $(\sigma,\mathbb{P}(\sigma))\in\mathrm{short}(\tau,\varphi)$, the integer $\tau(\sigma)$ obtained by evaluating τ on σ is strictly smaller than $[\![\xi]\!]_\bullet$. Let $\boldsymbol{x}\coloneqq(x_1,\ldots,x_n)$ be the variables occurring in τ. By definition, $\tau(\boldsymbol{x})\leq\|\tau\|_1\cdot 2^{\max(x_1,\ldots,x_n)}$ for all values given to $\boldsymbol{x}$ among the natural numbers. Consider then $(\sigma,\mathbb{P}(\sigma))\in\mathrm{short}(\tau,\varphi)$, with $\sigma=(y_0\leftarrow\rho_0,\ldots,y_m\leftarrow\rho_m)$. By definition of short, we have $m\leq q(|(\tau,\varphi)|)$. We show that, for every $i\in[1..m]$, $|[\![\sigma]\!](y_i)|\leq[\![\xi]\!](z_i)$. Together with the fact that $[\![\xi]\!](z_{j-1})<[\![\xi]\!](z_j)$ for every $j\in[1..s]$, this implies $\tau(\sigma)\leq\|\tau\|_1\cdot 2^{[\![\xi]\!]_\bullet}$, concluding the proof.

base case: $i=1$**.** The expression ρ_1 has one of the following forms: 0, $a\cdot y_0$ (for some $a\in\mathbb{Q}$), y_0+y_0, or 2^{y_0}. Since $[\![\sigma]\!](y_0)=0$, we have $[\![\sigma]\!](y_1)\in\{0,1\}$. On the other hand, $[\![\xi]\!](z_1)=2^s>1$.

induction hypothesis. Given $i\geq 2$, we have $|[\![\sigma]\!](y_j)|\leq[\![\xi]\!](z_j)$ for every $j\in[1..i-1]$.

induction step: $i\geq 2$**.** The expression ρ_i has one of the following forms: 0, $a\cdot y_j$, y_j+y_k or 2^{y_j}, where $j,k\in[1..i-1]$. Let $\ell\coloneqq\max\{|[\![\sigma]\!](y_j)|:j\in[1..i-1]\}$, and $k\in[1..i-1]$ be such that $|[\![\sigma(y_k)]\!]|=\ell$. Then, $[\![\sigma]\!](y_i)\leq\max(|a|\cdot\ell,2^\ell)$. We show that $\max(|a|\cdot\ell,2^\ell)\leq[\![\xi]\!](z_i)$:

- $2^\ell\leq[\![\xi]\!](z_i)$**:** By induction hypothesis, $\ell=|[\![\xi]\!](y_k)|\leq[\![\xi]\!](z_k)$. By definition of ξ, $[\![\xi]\!](z_{j-1})<[\![\xi]\!](z_j)$ for every $j\in[1..s]$, and therefore $[\![\xi]\!](z_k)\leq[\![\xi]\!](z_{i-1})$. Moreover, by definition $[\![\xi]\!](z_i)=2^{[\![\xi]\!](z_{i-1})}$. Therefore, $2^\ell\leq 2^{[\![\xi]\!](z_k)}\leq 2^{[\![\xi]\!](z_{i-1})}=[\![\xi]\!](z_i)$.
- $|a|\cdot\ell\leq[\![\xi]\!](z_i)$**:** Since $0\leq\ell\leq[\![\xi]\!](z_{i-1})$ (from the previous point in the proof), it suffices to show $|a|\cdot[\![\xi]\!](z_{i-1})\leq 2^{[\![\xi]\!](z_{i-1})}$. This inequality is trivial for $a=0$. Else, by Lemma 10, we see that the inequality is true as soon as $[\![\xi]\!](z_{i-1})\geq 4\cdot\log_2(|a|)+8$. Observe that the bit size of a is bounded by the bit size of σ, and therefore $\log_2(|a|)\leq q(|(\tau,\varphi)|)$. By definition of ξ, we also have $[\![\xi]\!](z_{i-1})\geq[\![\xi]\!](z_1)=2^{q(|(\tau,\varphi)|)+3}$. Then,

$$4\cdot\log_2(|a|)+8\leq 4\cdot q(|(\tau,\varphi)|)+8\leq 2^{q(|(\tau,\varphi)|)+3}\leq[\![\xi]\!](z_{i-1}).$$

This completes the proof of Corollary 1.

Part IV
Appendices

A The Sequential Squaring Assumption and ILESLPs

This appendix contains a detour on the *time-lock puzzle* introduced in [RSW96], which we use to establish a lower bound for the problem $\textsc{Div}_{\textsc{ileslp}}$ from Section 1.2 (or, equivalently, the problem of deciding if an LESLP is an ILESLP) in terms of a well-established cryptographic assumption.

Basic number theory concepts for cryptography. A prime p is said to be *safe* whenever $\frac{p-1}{2}$ is also prime. A number b is a *quadratic residue* modulo N whenever it is congruent to r^2 modulo N, for some $r \in [0..N-1]$; if b is also in $[0..N-1]$, then it is a quadratic residue *of* N. Given two distinct safe primes p and q, the set of all quadratic residues modulo $N \coloneqq p \cdot q$ forms a multiplicative cyclic subgroup of order $\frac{(p-1)\cdot(q-1)}{4}$; the key point being that it is then possible to generate all quadratic residues of N starting from any of them, but it is impossible to do so in time polynomial in the bit sizes of p and q. A function $f\colon \mathbb{N} \to (0,1)$ is said to be *negligible* if for every $c \in \mathbb{N}$ there is an integer M_c such that $f(n) < \frac{1}{n^c}$ for every $n > M_c$.

The time-lock puzzle. We give a brief description of the time-lock puzzle from [RSW96], referring the reader to that paper for a full account on the problem and its applications. The objective is to encrypt a message M in a way that gives not only strong guarantees on the minimum amount of time any adversary must spend to decrypt it, but also some (mild) guarantees on the maximum time a strong adversary would take. As usual in the computational model of cryptography, adversaries are modelled as probabilistic polynomial-time Turing machines, and we moreover assume to know a reasonably tight upper bound S on the number of squaring per second that these adversaries can perform, modulo any number. At our disposal, we also have a pair of symmetric-key cryptographic algorithms (Encrypt,Decrypt); these algorithms are known to the adversary. Besides minor changes that we will discuss later, [RSW96] proposes the following protocol for encrypting M:

1: $p, q \leftarrow$ two distinct safe primes such that $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are both congruent to 3 modulo 8
2: $T \leftarrow S \cdot t$ ▷ *t: number of seconds the puzzle must last. Given in input with M and S*
3: generate a secret key $K \in [0..N-1]$ for the pair of algorithms (Encrypt,Decrypt)
4: $C \leftarrow \textsc{Encrypt}(K, M)$
5: $E \leftarrow 2^T \bmod \phi(N)$ ▷ *use exponentiation-by-squaring method [BW08, Ch. 1.4]...*
6: $D \leftarrow (K + 2^E) \mod N$ ▷ *...twice*
7: **return** (N, D, C, T)

To solve the time-lock puzzle, an adversary must retrieve the message M. Except for trying to compute K from C —which is infeasible, since secure symmetric-key cryptographic algorithms exists (the simplest of all being one-time pad)— the only way for the adversary to retrieve M is to extract K from D, and then run $\textsc{Decrypt}(K, C)$. That can be done by computing $y \coloneqq 2^{2^T} \bmod N$ via repeated squaring, to then subtract it from D (modulo N). The key cryptographic assumption implying the security of the time-lock puzzle thus focuses on the computation of y:

Conjecture 1 (Sequential Squaring Assumption)**.** *There is a polynomial $P\colon \mathbb{N} \to \mathbb{N}$ such that for every probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function $negl\colon \mathbb{N} \to (0,1)$*

such that for all $\lambda \in \mathbb{N}$ *(in unary):*

$$\left| Pr\left[b = b' : \begin{array}{rl} (p,q,N) & \leftarrow \textsc{GenMod}(1^\lambda) \\ T & \xleftarrow{\$} P(2^\lambda) \\ b & \xleftarrow{\$} \{0,1\} \\ \textit{if } b = 0 & \textit{then } y \coloneqq 2^{2^T} \bmod N \\ \textit{if } b = 1 & \textit{then } y \xleftarrow{\$} \mathbb{QR}_N \\ b' & \leftarrow \mathcal{A}(1^\lambda, N, T, y) \end{array} \right] - \frac{1}{2} \right| \leq \textit{negl}(\lambda).$$

In Conjecture 1, λ is a security parameter that governs the bit sizes of the primes p and q, the "time limit" T of the time-puzzle, and the runtime of the adversary $\mathcal{A}$. The function GenMod is a probabilistic polynomial-time algorithm that returns a triple (p, q, N) where p and q are distinct safe primes such that $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are congruent to 3 modulo 8 (i.e., those computed in line 1 of the protocol), and $N \coloneqq p \cdot q$. The arrow $\xleftarrow{\$}$ stands for uniform sampling. In a nutshell, Conjecture 1 states that adversaries can only distinguish between a y computed as $(2^{2^T} \bmod N)$ and one randomly sampled among the quadratic residues of N (denoted $\mathbb{QR}_N$ above) with negligible probability.

Proposition 5. *Conjecture 1 implies that* $\textsc{Div}_{\textsc{ileslp}}$ *is not in* BPP.

Proof. Suppose that $\textsc{Div}_{\textsc{ileslp}}$ is in BPP. A simple adversary $\mathcal{A}$ breaking the binary sequential squaring assumption is defined as follows. Given the input $(1^\lambda, N, T, y)$, $\mathcal{A}$ constructs the ILESLP σ:

$$x_0 \leftarrow 0, \quad x_1 \leftarrow 2^{x_0}, \quad x_2 \leftarrow T \cdot x_1, \quad x_3 \leftarrow 2^{x_2}, \quad x_4 \leftarrow 2^{x_3}, \quad x_5 \leftarrow -y \cdot x_1, \quad x_6 \leftarrow x_4 + x_5,$$

which evaluates to $[\![\sigma]\!]_\bullet = 2^{2^T} - y$. The adversary then invokes the BPP algorithm for $\textsc{Div}_{\textsc{ileslp}}$ with inputs σ and N. If the algorithm returns true, $\mathcal{A}$ outputs 0; otherwise, it outputs 1. □

On the security of the time-lock puzzle. As mentioned above, the protocol (and thus the cryptographic assumption) considered in [RSW96] differ very slightly from the one reported here. In particular, the protocol in [RSW96] allows in line 5 to use exponentiation x^E instead of 2^E, where $x \in [2..N-1]$ is randomly chosen. Correspondingly, the cryptographic assumption in Conjecture 1 would sample uniformly at random x in $[2..N-1]$ to then define $y \coloneqq x^{2^T} \bmod N$ in the case of $b = 0$. The fact that the protocol is believed to be secure then stems from the fact that FACTORING is not believed to be in BPP and that, with high probability, the period of the sequence $x_0, x_1, x_2, \ldots$, where $x_i \coloneqq x^{2^i} \bmod N$, is large comparatively to N. (Note that we can assume $\textsc{GenMod}(1^\lambda)$ to return an N in $\Omega(2^\lambda)$, hence the period of the sequence is also large comparatively to T.) As remarked in [RSW96], we can fix $x = 2$ as long as we guarantee this period to still be large. This is ensured by the constraints on the primes p and q imposed in line 1 of the protocol (which are absent in [RSW96]), as we explain below.

Denote by $\lambda \colon \mathbb{N}_{\geq 1} \to \mathbb{N}_{\geq 1}$ the Carmichael function. Given a positive integer n, this function returns the smallest positive integer m such that $a^m \equiv 1 \pmod{n}$ holds for every a coprime with n.

Theorem 2 [BBS86]**.** *Let* $N \coloneqq p \cdot q$*, with* p *and* q *distinct safe primes such that* $\frac{p-1}{2}$ *and* $\frac{q-1}{2}$ *are congruent to* 3 *modulo* 8*. The period of the sequence* $x_0, x_1, \ldots$*, where* $x_i = 2^{2^i} \bmod N$*, is* $\lambda(\lambda(N))$*.*

Proof. Denote by π the period of the sequence $x_0, x_1, \ldots$, and by $\mathrm{ord}_M(x)$ the (multiplicative) order of x modulo M (assuming x and M coprime). From the definition of the Charmichael function one can show that $\frac{\lambda(N)}{2} = \frac{p-1}{2} \cdot \frac{q-1}{2}$. In particular, since both $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are odd, $\mathrm{ord}_{\lambda(N)/2}(2)$ is well-defined. In [BBS86] the following results are established:

1. Under the sole assumptions that p and q are distinct primes that are congruent to 3 modulo 4, and that 2 is a quadratic residue modulo N, we have $\pi \mid \lambda(\lambda(N))$. (See [BBS86, Theorem 6].)

2. Under the same assumptions as Item 1, and further assuming $\mathrm{ord}_{\lambda(N)/2}(2) = \lambda(\lambda(N))$ and $\mathrm{ord}_N(2) = \frac{\lambda(N)}{2}$, it holds that $\lambda(\lambda(N)) \mid \pi$. (See [BBS86, Theorem 7].)

3. Under the sole assumption that p and q are distinct safe primes that are congruent to 3 modulo 4, and that 2 is a quadratic residue with respect to *at most* one among $\frac{p-1}{2}$ and $\frac{q-1}{2}$, then $\mathrm{ord}_{\lambda(N)/2}(2) = \lambda(\lambda(N))$. (See [BBS86, Theorem 8].)

The theorem then follows as soon as we establish that all the above assumptions are covered by "p and q are distinct safe primes such that $\frac{p-1}{2}$ and $\frac{q-1}{2}$ are congruent to 3 modulo 8":

Assumption: "p and q are distinct are congruent to 3 modulo 4"**.** This assumption is equivalent to asking $\frac{p-1}{2}$ and $\frac{q-1}{2}$ to be odd, which they are, since they are congruent to 3 modulo 8.

Assumption: "2 is a quadratic residue modulo N"**.** First, remark that 2 is a quadratic residue modulo $p \cdot q$ if and only if 2 is a quadratic residue modulo p and modulo q. The left-to-right direction of this double implication is trivial. For the right-to-left direction, suppose $r^2 \equiv 2 \pmod{p}$ and $s^2 \equiv 2 \pmod{q}$. By the Chinese remainder theorem, there is $z \in [0..p \cdot q - 1]$ such that $z \equiv r \pmod{p}$ and $z \equiv s \pmod{q}$. Then, $z^2 - 2$ is divisible by both p and q, and so, by coprimality of p and q, it is also divisible by $p \cdot q$; i.e., 2 is a quadratic residue of $p \cdot q$.

Let us then show that 2 is a quadratic residue modulo p (same arguments for q). The second supplement to the law of quadratic reciprocity (see the entry for *"quadratic reciprocity, law of"* in [Nel08]) states that 2 is a quadratic residue modulo p if and only if $p \equiv \pm 1 \pmod{8}$. Note that if $p \equiv 1 \pmod{8}$, then $\frac{p-1}{2} \equiv 0 \pmod{4}$ and therefore p cannot be a safe prime. The congruence $p \equiv -1 \pmod{8}$ is instead equivalent to $\frac{p-1}{2} \equiv 3 \pmod{4}$ (and there are safe primes satisfying these pairs of constraints, take e.g. $p = 7$). Since we are assuming $\frac{p-1}{2} \equiv 3 \pmod{8}$, we also have $\frac{p-1}{2} \equiv 3 \pmod{4}$; as required.

Observe that we have now shown that the assumptions of Item 1 apply.

Assumption: $\mathrm{ord}_N(2) = \frac{\lambda(N)}{2}$**.** Since p and q are safe primes, the set of quadratic residues modulo N forms a multiplicative cyclic group of order $\frac{(p-1)\cdot(q-1)}{4} = \frac{\lambda(N)}{2}$; meaning in particular that all quadratic residues have the same order (i.e., $\frac{\lambda(N)}{2}$). From the previous point, 2 is a quadratic residue modulo N.

Assumption: "2 is a quadratic residue with respect to at most one among $\frac{p-1}{2}$ and $\frac{q-1}{2}$"**.** Again from the second supplement to the law of quadratic reciprocity, 2 is a quadratic residue modulo $\frac{p-1}{2}$ if and only if $\frac{p-1}{2} \equiv \pm 1 \pmod{8}$. We are however imposing $\frac{p-1}{2} \equiv 3 \pmod{8}$, so 2 is not a quadratic residue modulo $\frac{p-1}{2}$ (nor modulo $\frac{q-1}{2}$).

This is the last assumption we needed to show: it completes the assumptions in Item 3 and, following the conclusion of that item, also the assumptions in Item 2. □

One last point: Conjecture 1 requires GenMod to run in probabilistic polynomial-time, and so we also need to check that the set of numbers p that are safe primes satisfying $\frac{p-1}{2} \equiv 3 \pmod{8}$ is not only infinite, but also not too sparse. However, already whether there are infinitely many safe primes is not known. The usual (well-corroborated) assumption in cryptography is that among the first n integers, $\Omega\left(\frac{n}{\log(n)^2}\right)$ are safe primes. Following Dirichlet's theorem, it is natural to expect such a bound to hold also for the safe primes p satisfying $\frac{p-1}{2} \equiv 3 \pmod{8}$:

Conjecture 2. *Among the first $n \in \mathbb{N}_{\geq 1}$ positive integers, $\Omega\big(\frac{n}{\log(n)^2}\big)$ are safe primes numbers p that satisfy $\frac{p-1}{2} \equiv 3 \pmod 8$.*

The bound in Conjecture 2 is also implied by the well-known Bateman-Horn conjecture. Taken together, Theorem 2, Conjecture 2, and the assumption that FACTORING is not in BPP provide a rationale for believing that Conjecture 1 holds.

Proof that the Bateman-Horn conjecture implies Conjecture 2. Define the maps $f_1(x) \coloneqq 8 \cdot x + 3$ and $f_2(x) \coloneqq 2 \cdot f_1(x) + 1$. Note that the subset of $[1..n]$ we are interested in is

$$S(n) \coloneqq \{f_2(i) : i \in \mathbb{N} \text{ and both } f_1(i) \text{ and } f_2(i) \text{ are prime}\} \cap [1..n].$$

We first assume Dickson's conjecture and show that it implies that $S(n)$ is infinite. We will later appeal to Bateman–Horn conjecture (which implies Dickson's conjecture) to obtain an estimation on $\#S(n)$. Recall that Dickson's conjecture states that, for any given finite family $f_1, \ldots, f_k$ of univariate functions $f_i(x) \coloneqq a_i \cdot x + b_i$, where $a_i \in \mathbb{N}_{\geq 1}$ and $b_i \in \mathbb{Z}$:

1. there are infinitely many $n \in \mathbb{N}$ such that $f_1(n), \ldots, f_k(n)$ are all primes, or
2. there is a single integer $\alpha \geq 2$ dividing, for every $m \in \mathbb{N}$, the product $\prod_{i=1}^{k} f_i(m)$.

We simply have to exclude the second of the two cases above, showing that there are two integers $m_1, m_2 \in \mathbb{N}$ such that $f_1(m_1) \cdot f_2(m_1)$ and $f_1(m_2) \cdot f_2(m_2)$ are coprime. This holds already for $m_1 = 1$ and $m_2 = 2$: $f_1(1) \cdot f_2(1) = 11 \cdot 23$ and $f_1(2) \cdot f_2(2) = 3 \cdot 13 \cdot 19$.

Moving to the density estimation, since the first of the two cases in Dickson's conjecture applies, the Bateman–Horn conjecture implies that there is a real number $C \in \mathbb{R}$ dependent on f_1 and f_2 (and independent on n) such that $\#S(n) \geq C \cdot \int_2^n \frac{dt}{(\log t)^2}$; i.e., $\#S(n)$ is in $\Omega(\frac{n}{(\log n)^2})$ (in fact, the Bateman–Horn conjecture gives $\Theta(\frac{n}{(\log n)^2})$, but we only need a lower bound). □

B The algorithm for deciding ILEP: Further information on Steps I and III

In this appendix, we provide a further information on the Steps I and III of the algorithm in [CMS24]. In particular, we import the pseudocode of these steps, as well as their complexity analysis. When appealing to formal statements from [CMS24], we refer to the full version of the paper (as indicated in the corresponding bibliography entry).

Some additional notation. Troughout this appendix, we sometimes write a divisibility constraint $d \mid \tau$ as $\tau \equiv_d 0$. We need a few definitions from [CMS24]. Below, let θ be the ordering of exponentiated variables $\theta(\boldsymbol{x}) \coloneqq 2^{x_n} \geq 2^{x_{n-1}} \geq \cdots \geq 2^{x_0} = 1$, for some $n \geq 1$.

Definition 4 (Quotient System)**.** *A* quotient system induced by θ *is a system $\varphi(\boldsymbol{x}, \boldsymbol{q}, \boldsymbol{r})$ of equalities, inequalities, and divisibility constraints $\tau \sim 0$, where $\sim \in \{<, \leq, =, \equiv_d \colon d \geq 1\}$ and τ is an* quotient term (induced by θ)*, that is, a term of the form*

$$a \cdot 2^{x_n} + f(\boldsymbol{q}) \cdot 2^{x_{n-1}} + b \cdot x_{n-1} + \tau'(x_0, \ldots, x_{n-2}, \boldsymbol{r}),$$

where $a, b \in \mathbb{Z}$, $f(\boldsymbol{q})$ is a linear term on quotient variables $\boldsymbol{q}$*, and τ' is a linear-exponential term in which the* remainder *variables $\boldsymbol{r}$ do not occur exponentiated. Furthermore, for every remainder variable r, the quotient system φ features the inequalities $0 \leq r < 2^{x_{n-1}}$.*

A disclaimer: Observe that quotient systems can be syntactically equal to linear-exponential programs. In particular, this happens when every linear term $f(\boldsymbol{q})$ appearing in quotient terms is an integer. However, [CMS24] keeps the two types of objects somewhat separated, as if they are distinct "types" (in the sense of programming languages). That is, quotient systems are not linear-exponential programs. In the algorithm from [CMS24], quotient systems only appear at the beginning of Step I of the algorithm. To be more precise, let us look at the pseudocode of Step I (Algorithm 10). In input, this step takes an ordering θ, and a linear-exponential program with divisions φ. The **foreach** loop of line 2 translates φ into a quotient system (adding new quotient and remainder variables). When the procedure reaches line 9, the translation is complete. All other systems constructed by the algorithm (in Step I, γ and ψ) are linear-exponential programs.

The above distinction between linear-exponential programs and quotient systems is important for the definition of *least significant part* of a term (introduced in Section 6 for linear-exponential programs, but restated below to avoid confusion). In this definition, quotient terms and linear-exponential terms are treated differently, and the definition becomes ill-formed if quotient systems are mistakenly regarded as linear-exponential programs.

Definition 5 (Least Significant Part)**.** *The* least significant part *of a term τ, with respect to the ordering θ, is defined as follows:*

1. *If τ is a* linear-exponential term $a \cdot 2^{x_n} + b \cdot x_n + \tau'$*, with τ' linear-exponential term only featuring x_n in remainders ($x \bmod 2^y$), its least significant part is the term $b \cdot x_n + \tau'$.*
2. *If τ is a* quotient term $a \cdot 2^{x_n} + f(\boldsymbol{q}) \cdot 2^{x_{n-1}} + b \cdot x_{n-1} + \tau'$*, with τ' not featuring x_n nor x_{n-1}, its least significant part is the term $b \cdot x_{n-1} + \tau'$.*

Moreover, let φ be either a linear-exponential program with divisions or a quotient system. We denote by $lst(\varphi, \theta)$ the following set of least significant terms:

$$lst(\varphi,\theta) = \left\{ \pm\rho : \begin{array}{l} \rho \text{ is the least significant part of a term } \tau \text{ appearing in an (in)equality } \tau \sim 0 \\ \text{of } \varphi, \text{ with respect to } \theta \end{array} \right\}.$$

An analogous distinction arises in the definition of *linear norm*:

Definition 6 (Linear norm)**.** *The* linear norm $\|\tau\|_{\mathfrak{L}}$ *of a term τ is defined as follows:*

1. *If τ is a* linear-exponential term $\sum_{i=1}^{n} \left(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j})\right) + d$*, then $\|\tau\|_{\mathfrak{L}} \coloneqq \max\{|a_i|, |c_{i,j}| : i, j \in [1..n]\}$, i.e., it reflects the maximum absolute value among the coefficients of the linear terms x_i and the remainder terms $(x_i \bmod 2^{x_j})$.*
2. *If τ is a quotient term induced by θ, of the form $\tau = a \cdot 2^{x_n} + f(\boldsymbol{q}) \cdot 2^{x_{n-1}} + b \cdot x_{n-1} + \tau'$, then $\|\tau\|_{\mathfrak{L}} \coloneqq \max(|b|, \|\tau'\|_{\mathfrak{L}}, \|f\|_{\mathfrak{L}})$. Note that $\|\tau\|_{\mathfrak{L}}$ accounts for the coefficients of the variables $\boldsymbol{q}$.*

Moreover, let φ be either a linear-exponential program with divisions or a quotient system. The linear norm of φ is defined as $\|\varphi\|_{\mathfrak{L}} \coloneqq \max\{\|\tau\|_{\mathfrak{L}} : \tau \text{ is a term appearing in an (in)equality of } \varphi\}$.

The parameters $\#\varphi$, $\|\varphi\|_1$ and $mod(\varphi)$ extend instead trivially to quotient systems φ:

- $\#\varphi$ for the number of constraints (inequalities, equalities and divisibility constraints) in φ;
- $\text{terms}(\varphi)$ for the set of all terms τ occurring in inequalities $\tau \leq 0$ or equalities $\tau = 0$ of φ;
- $\|\varphi\|_1 \coloneqq \max\{\|\tau\|_1 : \tau \in \text{terms}(\varphi)\}$.
  For a quotient term $\tau = a \cdot 2^{x_n} + f(\boldsymbol{q}) \cdot 2^{x_{n-1}} + b \cdot x_{n-1} + \tau'$, we have $\|\tau\|_1 \coloneqq |a| + |b| + \|f\|_1 + \|\tau'\|_1$.
- $mod(\varphi)$ is the least common multiple of the divisors d of the divisibility constraints $d \mid \tau$ of φ.
  We recall that, in these constraints, all integers appearing in τ belong to $[0..d-1]$.

B.1 Step I

Algorithm 10 presents the pseudocode of Step I. This pseudocode is obtained by merging (exclusively to simplify the presentation) lines 4–14 of Algorithm 2 with lines 1–21 of Algorithm 3 from [CMS24]. The full specification of Algorithm 10 is recalled below.

Lemma 4 [CMS24]**.** *There is a non-deterministic procedure with the following specification:*

Input: $\theta(\boldsymbol{x})$ *: ordering of exponentiated variables;*
[Below, let 2^x *and* 2^y *be the largest and second-largest terms in this ordering, and let* $\boldsymbol{y}$ *be the vector obtained by removing* x *from* $\boldsymbol{x}$*.]*
$\varphi(\boldsymbol{x}, \boldsymbol{r})$ *: linear-exponential program with divisions, implying* $\boldsymbol{r} < 2^x$*.*
Variables $\boldsymbol{r}$ *do not occur in exponentials.*

Output of each branch (β)***:***

$\gamma_\beta(q_x, \boldsymbol{q}, u)$ *: linear program with divisions;*
$\psi_\beta(\boldsymbol{y}, r_x, \boldsymbol{r'})$ *: linear-exponential program with divisions, implying* $r_x < 2^y \wedge \boldsymbol{r'} < 2^y$*.*
Variables r_x *and* $\boldsymbol{r'}$ *do not occur in exponentials.*

The variables q_x*,* $\boldsymbol{q}$*,* u*,* $\boldsymbol{y}$*,* r_x *and* $\boldsymbol{r'}$ *are common to all outputs, across all non-deterministic branches. The procedure ensures that the system*

$$\begin{bmatrix} x \\ \boldsymbol{r} \end{bmatrix} = \begin{bmatrix} q_x \\ \boldsymbol{q} \end{bmatrix} \cdot 2^y + \begin{bmatrix} r_x \\ \boldsymbol{r'} \end{bmatrix}, \tag{3}$$

yields a one-to-one correspondence between the solutions of $\varphi \wedge \theta$ *and the solutions of the formula* $\bigvee_\beta \big(\gamma_\beta \wedge \psi_\beta \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x) \wedge \theta\big)$*. This correspondence is the identity for the variables these two formulae share (that is, the variables in* $\boldsymbol{x}$*).*

Proof. The correctness of Algorithm 10 follows from [CMS24, Proposition 4 and Lemma 23]:

- **Lines 1–8.** These lines are analysed in the proof of [CMS24, Proposition 4] (Appendix C.3, page 54). The **foreach** loop of line 2 (deterministically) manipulates φ into a quotient system φ'. Let $\boldsymbol{q}$ and $\boldsymbol{r'}$ be the set of all fresh quotient and reminder variables introduced in line 3 (across all iterations of the loop). This part of the algorithm ensure that

$$\begin{bmatrix} x \\ \boldsymbol{r} \end{bmatrix} = \begin{bmatrix} q_x \\ \boldsymbol{q} \end{bmatrix} \cdot 2^y + \begin{bmatrix} r_x \\ \boldsymbol{r'} \end{bmatrix} \quad \text{implies} \quad (\varphi \wedge \theta) \iff (\varphi' \wedge \theta) \tag{34}$$

  In [CMS24], lines 4–14 of Algorithm 2 construct φ' to then pass it to Algorithm 3, which performs (with respect to our pseudocode) the following lines 9–31.

- **Lines 9–31.** These lines are analysed in the proof of [CMS24, Lemma 23] (Appendix C.2, page 40; see in particular the subsection titled "Correctness of Step (i)"). In a nutshell, these lines "divide" each quotient term in φ' by 2^y, by relying on the equivalences in Lemma 3. For example, an equality $a \cdot 2^x + f(\boldsymbol{q}) \cdot 2^y + b \cdot y + \tau' = 0$ is (non-deterministically) rewritten as $a \cdot 2^{x-y} + f(\boldsymbol{q}) + r = 0 \wedge b \cdot y + \tau' = r \cdot 2^y$. Note that $b \cdot y + \tau'$ is the least significant part of the term in the initial equality. Constraints concerning these least significant parts (in our example, $b \cdot y + \tau' = r \cdot 2^y$) are added to the formula ψ (lines 18, 23, 27 and 31), whereas the remaining constraints featuring the variable x (in our example, $a \cdot 2^{x-y} + f(\boldsymbol{q}) + r = 0$) are added to the formula γ (lines 26 and 30; note that the algorithm uses u as a proxy for 2^{x-y}).

Algorithm 10 Step I of the algorithm from [CMS24]. See Lemma 4 for its full specification.

Input: $\theta(\boldsymbol{x})$: ordering of exponentiated variables;
[Below, let 2^x and 2^y be the largest and second-largest terms in this ordering, and let $\boldsymbol{y}$ be the vector obtained by removing x from $\boldsymbol{x}$.]
$\varphi(\boldsymbol{x}, \boldsymbol{r})$: linear-exponential program with divisions, implying $\boldsymbol{r} < 2^x$.
Variables $\boldsymbol{r}$ do not occur in exponentials.

Output of each branch (β):
$\gamma_\beta(q_x, \boldsymbol{q}, u)$: linear program with divisions;
$\psi_\beta(\boldsymbol{y}, r_x, \boldsymbol{r'})$: linear-exponential program with divisions, implying $r_x < 2^y \wedge \boldsymbol{r'} < 2^y$.
Variables r_x and $\boldsymbol{r'}$ do not occur in exponentials.

$\varphi \leftarrow \varphi[w \,/\, (w \bmod 2^x) : w \text{ is a variable}]$
foreach r in $\boldsymbol{r} \cup \{x\}$ **do** ▷ *translate φ into a quotient system induced by θ*
  let q_r and r' be two fresh variables
  $\varphi \leftarrow \varphi \wedge (0 \le r' < 2^y)$
  $\varphi \leftarrow \varphi[r' \,/\, (r \bmod 2^y)]$
  $\varphi \leftarrow \varphi[(r' \bmod 2^w) \,/\, (r \bmod 2^w) : w \text{ is such that } \theta \text{ implies } 2^w \le 2^y]$
  $\varphi \leftarrow \varphi[(q_r \cdot 2^y + r') \,/\, r]$ ▷ *replaces only the linear occurrences of r*
  if r is x **then** $(q_x, r_x) \leftarrow (q_r, r')$ ▷ *the substitution of x in exponentials is delayed*
let u be a fresh variable ▷ *u is an alias for 2^{x-y}*
$\gamma \leftarrow \top$; $\psi \leftarrow \top$ ▷ *new linear-exponential programs constructed from φ*
$\Delta \leftarrow \varnothing$ ▷ *map from linear-exponential terms to $\mathbb{Z}$*
foreach $(\tau \sim 0)$ in φ, where $\sim \in \{=, <, \le, \equiv_d : d \ge 1\}$ **do**
  let τ be $(a \cdot 2^x + f(\boldsymbol{x'}) \cdot 2^y + \rho)$, where ρ is the least significant part of τ
  if $a = 0$ and $f(\boldsymbol{x'})$ is an integer **then** $\psi \leftarrow \psi \wedge (\tau \sim 0)$
  else if the symbol $\sim$ belongs to $\{=, <, \le\}$ **then**
    if $\Delta(\rho)$ is undefined **then**
      guess $h \leftarrow$ integer in $[-\|\rho\|_1, \|\rho\|_1]$
      $\psi \leftarrow \psi \wedge ((h-1) \cdot 2^y < \rho) \wedge (\rho \le h \cdot 2^y)$
      update Δ : add the key–value pair (ρ, h)
    $h \leftarrow \Delta(\rho)$
    if the symbol $\sim$ is $<$ **then**
      guess $\sim' \leftarrow$ sign in $\{=, <\}$
      $\psi \leftarrow \psi \wedge (\rho \sim' h \cdot 2^y)$
      $\sim \leftarrow \le$
      if the symbol $\sim'$ is $=$ **then** $h \leftarrow h + 1$
    $\gamma \leftarrow \gamma \wedge (a \cdot u + f(\boldsymbol{x'}) + h \sim 0)$
    if the symbol $\sim$ is $=$ **then** $\psi \leftarrow \psi \wedge (h \cdot 2^y = \rho)$
  else ▷ *$\sim$ is $\equiv_d$ for some $d \in \mathbb{N}$*
    guess $h \leftarrow$ integer in $[1, mod(\varphi)]$
    $\gamma \leftarrow \gamma \wedge (a \cdot u + f(\boldsymbol{x'}) - h \sim 0)$
    $\psi \leftarrow \psi \wedge (h \cdot 2^y + \rho \sim 0)$
return (γ, ψ)

DECOUPLE $\boldsymbol{q}$, q_z AND u FROM THE OTHER VARIABLES

Since these lines of the algorithm are guided by the equivalences in Lemma 3, one obtains

$$\begin{bmatrix} x \\ \boldsymbol{r} \end{bmatrix} = \begin{bmatrix} q_x \\ \boldsymbol{q} \end{bmatrix} \cdot 2^y + \begin{bmatrix} r_x \\ \boldsymbol{r}' \end{bmatrix} \quad \text{implies} \quad (\varphi' \wedge \theta) \iff \bigvee_\beta \exists u \left(\gamma_\beta \wedge \psi_\beta \wedge (u = 2^{x-y}) \wedge \theta\right), \tag{35}$$

where $\bigvee_\beta$ ranges over all non-deterministic branches β.

The lemma follows by Equations (34) and (35). Consider a solution to $\varphi \wedge \theta$. We can uniquely decompose the values x, y and $\boldsymbol{r}$ take in this solution by following the system $\begin{bmatrix} x \\ \boldsymbol{r} \end{bmatrix} = \begin{bmatrix} q_x \\ \boldsymbol{q} \end{bmatrix} \cdot 2^y + \begin{bmatrix} r_x \\ \boldsymbol{r}' \end{bmatrix}$ and the equation $u = 2^{x-y}$, producing a solution to $\bigvee_\beta \left(\gamma_\beta \wedge \psi_\beta \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x) \wedge \theta\right)$ thanks to Equations (34) and (35). Conversely, from a solution to $\bigvee_\beta \left(\gamma_\beta \wedge \psi_\beta \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x) \wedge \theta\right)$, we can compute (unique) values for the variables $\boldsymbol{r}$ following the system $\boldsymbol{r} = \boldsymbol{q} \cdot 2^y + \boldsymbol{r}'$, producing a solution to $\theta \wedge \varphi$. □

We now move to the complexity of Algorithm 10:

Lemma 22 [CMS24]**.** *The algorithm from Lemma 4 (Step I) runs in non-deterministic polynomial time. Consider its execution on an input (θ, φ) where $\theta(\boldsymbol{x})$ is an ordering of exponentiated variables and $\varphi(\boldsymbol{x}, \boldsymbol{r})$ is a linear exponential program with divisions. In each non-deterministic branch β, the algorithm returns a pair (γ, ψ), where $\gamma(q_x, \boldsymbol{q}, u)$ is a linear program with divisions and $\psi(\boldsymbol{y}, r_x, \boldsymbol{r}')$ a linear-exponential program with divisions, such that (for every $\ell, s, a, c, d \geq 1$):*

$$\textit{if} \begin{cases} \#\mathit{lst}(\varphi, \theta) & \leq \ell \\ \#\varphi & \leq s \\ \|\varphi\|_{\mathfrak{L}} & \leq a \\ \|\varphi\|_1 & \leq c \\ \mathit{mod}(\varphi) & \mid d \end{cases} \textit{then} \begin{cases} \#\mathit{lst}(\psi, \theta') & \leq \ell + 2 \cdot k \\ \#\psi & \leq s + 6 \cdot k + 2 \cdot \ell \\ \|\psi\|_{\mathfrak{L}} & \leq 3 \cdot a \\ \|\psi\|_1 & \leq 4 \cdot c + 5 \\ \mathit{mod}(\psi) & \mid d \end{cases} \textit{and} \begin{cases} \#\gamma & \leq s + 2 \cdot k \\ \|\gamma[2^u \,/\, u]\|_{\mathfrak{L}} & \leq 3 \cdot a \\ \|\gamma\|_1 & \leq 2 \cdot c + 3 \\ \mathit{mod}(\gamma) & \mid d \end{cases}$$

where θ' is the ordering obtained from θ by removing its largest term 2^x, and $k \coloneqq 1 + \#\boldsymbol{r}$.

Proof. We report the complexity analysis from [CMS24, Lemma 6 and Lemma 37]:

- **Lines 1–8.** These lines are analyzed in [CMS24, Lemma 6] (Appendix D.3, page 63). As done in the sketch of the proof of Lemma 4, let us write φ' for the quotient system obtained from $\varphi(\boldsymbol{x}, \boldsymbol{r})$ after executing the **foreach** loop of line 2. [CMS24, Lemma 6] established the following bounds on φ':

$$\text{if} \begin{cases} \#\mathit{lst}(\varphi, \theta) & \leq \ell \\ \#\varphi & \leq s \\ \|\varphi\|_{\mathfrak{L}} & \leq a \\ \|\varphi\|_1 & \leq c \\ \mathit{mod}(\varphi) & \mid d \end{cases} \text{then} \begin{cases} \#\mathit{lst}(\varphi', \theta) & \leq \ell + 2 \cdot k \\ \#\varphi' & \leq s + 2 \cdot k \\ \|\varphi'\|_{\mathfrak{L}} & \leq 3 \cdot a \\ \|\varphi'\|_1 & \leq 2 \cdot (c + 1) \\ \mathit{mod}(\varphi') & \mid d \end{cases} \tag{36}$$

- **Lines 9–31.** These lines are analyzed in [CMS24, Lemma 37] (Appendix D.2, page 59; see in particular the subsection titled "Step (a)"). Let (γ, ψ) be an output of Algorithm 10. The

following bounds are established:

$$\text{if } \begin{cases} \#lst(\varphi', \theta) & \leq \ell \\ \#\varphi' & \leq s \\ \|\varphi'\|_{\mathfrak{L}} & \leq a \\ \|\varphi'\|_1 & \leq c \\ mod(\varphi') & \mid d \end{cases} \text{ then } \begin{cases} \#lst(\psi, \theta') & \leq \ell \\ \#\psi & \leq s + 2 \cdot \ell \\ \|\psi\|_{\mathfrak{L}} & \leq a \\ \|\psi\|_1 & \leq 2 \cdot c + 1 \\ mod(\psi) & \mid d \end{cases} \text{ and } \begin{cases} \#\gamma & \leq s \\ \|\gamma[2^u \,/\, u]\|_{\mathfrak{L}} & \leq a \\ \|\gamma\|_1 & \leq c + 1 \\ mod(\gamma) & \mid d \end{cases} \tag{37}$$

Note that above (and in the statement of the lemma) $\|\gamma[2^u \,/\, u]\|_{\mathfrak{L}}$ is considered instead of $\|\gamma\|_{\mathfrak{L}}$. As reported in [CMS24, Lemma 37], this is because only the coefficients of the variables distinct from u are interesting for the overall analysis of the complexity of the algorithm (in any case, we have a bound of $c + 1$ on this coefficient, given by $\|\gamma\|_1$). Performing the substitution $[2^u \,/\, u]$ makes it so that the coefficients of u are not accounted when computing $\|\cdot\|_{\mathfrak{L}}$.

The bounds in the statement of the lemma are obtained by imply conjoining the bounds in Equations (36) and (37). The fact that Algorithm 10 runs in non-deterministic polynomial time follows again directly from [CMS24, Lemma 6 and Lemma 37]. □

B.2 Step III

Algorithm 11 presents the pseudocode of Step III. It corresponds to lines 24–34 of Algorithm 3 from [CMS24]. Note that line 24 of that algorithm calls a procedure named SolvePrimitive; in our pseudocode, this line is replaced directly with the code of SolvePrimitive (i.e., with lines 1–15 of Algorithm 4 from [CMS24]).

Lemma 6 [CMS24]**.** *There is a non-deterministic procedure with the following specification:*

Input: $\gamma'(q_x, u)$ *: linear program with divisions.*

Output of each branch (β)***:*** $\gamma''_\beta(q_x)$ *: linear program with divisions;*
$\psi''_\beta(y, r_x)$ *: linear-exponential program with divisions.*

The procedure ensures that the equation

$$x = q_x \cdot 2^y + r_x \tag{5}$$

yields a one-to-one correspondence between the solutions of $\gamma' \wedge (u = 2^{x-y}) \wedge (x = q_x \cdot 2^y + r_x)$ *and the solutions of* $\bigvee_\beta (\gamma''_\beta \wedge \psi''_\beta)$*. This correspondence is the identity for the variables these two formulae share (that is,* y*,* q_x *and* r_x*).*

Proof. The correctness of the procedure directly follows from [CMS24, Lemma 21 and Lemma 32] (Appendix C.1, page 36, and Appendix C.2, page 52). □

Here is the complexity of Algorithm 11:

Lemma 23 [CMS24]**.** *The algorithm from Lemma 6 (Step III) runs in non-deterministic polynomial time. Consider its execution on an input linear program with divisions* γ'*. In each non-deterministic branch* β*, the algorithm returns a pair* (γ'', ψ'')*, where* γ'' *is a linear program with divisions and* ψ'' *is a linear-exponential program with divisions, such that (for every* $s, a, c, d \geq 1$*):*

$$\textit{if } \begin{cases} \#\gamma' & \leq s \\ \|\gamma'[2^u \,/\, u]\|_{\mathfrak{L}} & \leq a \\ \|\gamma'\|_1 & \leq c \\ mod(\gamma') & \mid d \end{cases} \textit{ then } \begin{cases} \#\gamma'' & \leq s + 2 \\ \|\gamma''\|_{\mathfrak{L}} & \leq a \\ \|\gamma''\|_1 & \leq \max(2^5 c^3, c \cdot d) \\ mod(\gamma'') & \mid \operatorname{lcm}(d, \phi(d)) \end{cases} \textit{ and } \begin{cases} \#\psi'' & \leq 3 \\ \|\psi''\|_{\mathfrak{L}} & \leq 1 \\ \|\psi''\|_1 & \leq 12 + 4 \cdot \log(\max(c, d)) \\ mod(\psi'') & \mid \phi(d) \end{cases}$$

Algorithm 11 Step III of the algorithm from [CMS24]. See Lemma 6 for its full specification.

Input: $\gamma'(q_x, u)$: linear program with divisions.

Output of each branch (β): $\gamma''_\beta(q_x)$: linear program with divisions;
$\psi''_\beta(y, r_x)$: linear-exponential program with divisions.

▷ *Recall: the procedure assumes both $u = 2^{x-y}$ and $x = q_x \cdot 2^y + r_x$ to hold (see Lemma 6)*

1: **let** γ' be $(\chi \wedge \varphi)$, where χ is the conjunction of all (in)equalities from γ' containing u
2: $(d, n) \leftarrow$ pair of non-negative integers such that $mod(\gamma') = d \cdot 2^n$ and d is odd
3: $C \leftarrow \max\left\{n, 3 + 2 \cdot \left\lceil \log(\frac{|b|+|c|+1}{|a|}) \right\rceil : (a \cdot u + b \cdot q_x + c \sim 0) \text{ in } \chi, \text{ where } \sim \in \{=, <, \leq\}\right\}$
4: **guess** $c \leftarrow$ element of $[0..C-1] \cup \{\star\}$ ▷ *$\star$ signals $x - y \geq C$*

Eliminate x

5: **if** c is not $\star$ **then**
6: $\quad \chi \leftarrow (x - y = c)$
7: $\quad \gamma \leftarrow \gamma'[2^c \,/\, u]$ ▷ *according to $x - y = c$ and $u = 2^{x-y}$*
8: **else** ▷ *assuming $v \geq C$, (in)equalities in χ simplify to $\top$ or $\bot$*
9: $\quad$ **assert**(χ has no equality, and in all its inequalities u has a negative coefficient)
10: $\quad$ **guess** $r \leftarrow$ integer in $[0..d-1]$ ▷ *remainder of 2^{x-y-n} modulo d when $x - y \geq C \geq n$*
11: $\quad$ **assert**(the divisibility $d \mid 2^v - 2^n \cdot r$ is satisfied by some $v \in [0..d-1]$)
12: $\quad r' \leftarrow$ discrete logarithm of $2^n \cdot r$ base 2, modulo d
13: $\quad d' \leftarrow$ multiplicative order of 2 modulo d
14: $\quad \chi \leftarrow (x - y \geq C) \wedge (d' \mid x - y - r')$
15: $\quad \gamma \leftarrow \varphi[2^n \cdot r \,/\, u]$ ▷ *$2^n \cdot r$ is a remainder of 2^{x-y} modulo $mod(\gamma') = d \cdot 2^n$*

16: $\chi \leftarrow \chi[q_x \cdot 2^y + r_x \,/\, x]$ ▷ *apply substitution: x is eliminated*

Decoupling q_x

17: **if** χ is $(-q_x \cdot 2^y - r_x + y + c = 0)$ **then** ▷ *true if χ was constructed in line 6*
18: $\quad$ **guess** $b \leftarrow$ integer in $[0..c]$
19: $\quad \gamma \leftarrow \gamma \wedge (q_x = b)$
20: $\quad \psi \leftarrow b \cdot 2^y = -r_x + y + c$
21: **else** ▷ *true if χ was constructed in line 14*
22: $\quad$ **let** χ be $(-q_x \cdot 2^y - r_x + y + C \leq 0) \wedge (d' \mid q_x \cdot 2^y + r_x - y - r')$, for some $d', r' \in \mathbb{N}$
23: $\quad$ **guess** $(b, g) \leftarrow$ pair of integers in $[0..C] \times [1..d']$
24: $\quad \gamma \leftarrow \gamma \wedge (q_x \geq b) \wedge (d' \mid q_x - g)$
25: $\quad \psi \leftarrow ((b-1) \cdot 2^y < -r_x + y + C) \wedge (-r_x + y + C \leq b \cdot 2^y) \wedge (d' \mid g \cdot 2^y + r_x - y - r')$

26: **return** (γ, ψ)

Proof. The proof of this lemma follows from the proofs of [CMS24, Lemma 36 and Lemma 37]. For completeness, we give below a standalone analysis of the bounds on γ'' and ψ''.

Analysis on γ'': The relevant lines are line 7, line 15, line 20 and line 25.

bound on $\#\gamma''$: The system γ'' is initialized with $\#\gamma'$ constraints in lines 7 or 15. The subsequent lines 20 and 25 add at most 2 constraints.

bound on $\|\gamma''\|_{\mathfrak{L}}$: Recall the γ'' is a linear program with divisions featuring a single variable q_x (see Lemma 6). When this system is initialized in lines 7 or 15, the absolute values of the coefficients of q_x are bounded by $\|\gamma'[2^u \,/\, u]\|_{\mathfrak{L}} \leq a$. Lines 20 and 25 only add constraints in which q_x appears with coefficient $1 \leq a$. Hence, $\|\gamma''\|_{\mathfrak{L}} \leq a$.

bound on $\|\gamma''\|_1$: When the system γ'' is initialized in lines 7 or 15, its norm $\|\cdot\|_1$ is bounded by $\|\gamma'\|_1 \cdot \max(2^C, \mathit{mod}(\gamma'))$, where C is the integer defined in line 3. The (in)equalities added in lines 20 and 25 feature term with norm $\|\cdot\|_1$ bounded by $C+1 \leq 2^C$. Therefore,

$$\begin{aligned}
\|\gamma''\|_1 &\leq c \cdot \max(2^C, d) && \text{⟅by } \|\gamma'\|_1 \leq c \text{ and } \mathit{mod}(\gamma') \leq d\text{⟆}\\
&\leq c \cdot \max(2^{3+2\cdot(\log(c)+1)}, d) && \text{⟅by def. of } C \text{ and } 2^n \leq \mathit{mod}(\gamma')\text{⟆}\\
&\leq c \cdot \max(2^5 c^2, d) \;\leq\; \max(2^5 c^3, c \cdot d).
\end{aligned}$$

bound on $\mathit{mod}(\gamma'')$: When γ'' is initialized in lines 7 or 15, its parameter $\mathit{mod}(\cdot)$ is equal to $\mathit{mod}(\gamma')$. Line 20 adds no divisibility constraints, whereas 25 adds a single divisibility constraints with divisor d'. Here, d' is the multiplicative order of 2 modulo the largest odd factor of $\mathit{mod}(\gamma')$. That is, d' is a divisor of $\phi(\mathit{mod}(\gamma'))$, which in turn is a divisor of $\phi(d)$, where ϕ is Euler's totient function. Therefore, $\mathit{mod}(\gamma')$ divides $\operatorname{lcm}(d, \phi(d))$.

Analysis on ψ'': The relevant lines are line 20 and line 25, which define ψ'' depending on χ.

bound on $\#\psi''$: The system ψ'' has a single equality when defined in line 20, and three constraints when defined in line 25.

bound on $\|\psi''\|_{\mathfrak{L}}$: The variables occurring in ψ'' are r_x and y. Following lines 20 and 25, these variables always appear with coefficients ± 1.

bound on $\|\psi''\|_1$: Following lines 20 and 25, we see that $\|\psi''\|_1 \leq 2 + 2 \cdot C$, where C is the integer defined as in line 3. Therefore,

$$\begin{aligned}
\|\psi''\|_1 &\leq 2 + 2 \cdot C\\
&\leq 2 + 2 \cdot \max(\lceil \log(d) \rceil, 3 + 2\lceil \log(c) \rceil) \quad \text{⟅using } \|\gamma'\|_1 \leq c \text{ and } 2^n \leq \mathit{mod}(\gamma') \leq d\text{⟆}\\
&\leq \max(2\log(d) + 4, 4\log(c) + 12)\\
&\leq 12 + 4 \cdot \log(\max(c, d)).
\end{aligned}$$

bound on $\mathit{mod}(\psi'')$: From line 25 we conclude that $\mathit{mod}(\psi'')$ is a divisor of d'. As discussed in the analysis of $\mathit{mod}(\gamma'')$, this implies that it also divides $\phi(d)$. □

C Proofs of statements from Part I

C.1 Proofs of statements from Section 3

Lemma 7. *Let φ be a linear-exponential program with divisions, and let x be a variable occurring linearly in φ. The set of solutions of φ is $(x, \mathit{mod}(x, \varphi))$-periodic.* *Statement in page 20*

Proof. For brevity, define $p \coloneqq \mathit{mod}(x, \varphi)$. Consider two solution ν and $\nu + [x \mapsto m]$ to φ, with $m \geq p$. (Recall that $\nu + [x \mapsto m]$ denotes the map obtained from ν by increasing the value assigned to x by m.) We prove that $\nu' \coloneqq \nu + [x \mapsto p]$ is also solution to φ. We analyse divisibility constraints and (in)equalities separately. Clearly, ν' satisfies all constraints in which x does not appear, hence below we only consider constraints featuring x (linearly, as per hypothesis).

divisibility constraints. Consider a divisibility constraint $\psi \coloneqq (d \mid c \cdot x + \tau)$. By definition of p, we have $d \mid p$, and therefore $c \cdot \nu(x)$ and $c \cdot (\nu(x) + p)$ have the same reminder modulo d. Hence, ν' satisfies ψ.

equalities and inequalities. We only show the case of inequalities (the proof for equalities is analogous, as they can be seen as a conjunction of two inequalities). Consider an inequality $\chi \coloneqq (c \cdot x + \tau \leq 0)$. Both ν and $\nu + [x \mapsto m]$ satisfy this inequality, so we have $c \cdot \nu(x) + \nu(\tau) \leq 0$ and $c \cdot (\nu(x) + m) + \nu(\tau) \leq 0$. Recall that $m \geq p \geq 1$. If $c < 0$, then $c \cdot (\nu(x) + p) + \nu(\tau) < c \cdot \nu(x) + \nu(\tau) \leq 0$. If instead $c > 0$, then $c \cdot (\nu(x) + p) + \nu(\tau) < c \cdot (\nu(x) + m) + \nu(\tau) \leq 0$. In both cases, we conclude that ν' satisfies χ. □

Lemma 8. *Let $\varphi(\boldsymbol{x})$ be a linear-exponential program with divisions, and let x be a variable occurring linearly in φ. Let $p \coloneqq \mathit{mod}(x, \varphi)$, and let $f(\boldsymbol{x})$ be a (x, p)-monotone function locally to the set of solutions to φ. If the instance (f, φ) has a maximum (analogously, a minimum), then it has one satisfying an equation $a \cdot x + \tau + r = 0$, where $(a \cdot x + \tau) \in \mathrm{terms}(\varphi \wedge x \geq 0)$, $a \neq 0$, and $r \in [0.. |a| \cdot p - 1]$.* *Statement in page 21*

Proof. We prove the lemma for the case of maximization. The case of minimization is analogous. For simplicity, let $\boldsymbol{x} = (x_1, \ldots, x_n)$. Given a solution ν to φ, we write $\Delta^p_x[f](\nu)$ as a shortcut for $\Delta^p_x(\nu(x_1), \ldots, \nu(x_n))$. Assume that an optimal solution to (f, φ) exists.

The lemma is trivially true when x occurs in an equality of φ, as all optima must satisfy that equality. Moreover, if x does not occur in equalities nor inequalities of φ, then the existence of an optimum implies that $\Delta^p_x[f](\nu) = 0$ whenever ν and $\nu + [x \mapsto p]$ are solutions to φ. We only need to make sure to satisfy the divisibility constraints, and therefore it suffices to consider equations $x = r$ with $r \in [0..p - 1]$. These equations are among those considered in the statement of the lemma (since we consider $\mathrm{terms}(\varphi \wedge x \geq 0)$). Below, we assume that x occurs in an inequality of φ but in no equality. We divide the proof depending on the sign of $\Delta^p_x[f]$.

case: $\Delta^p_x[f](\nu) > 0$ whenever ν and $\nu + [x \mapsto p]$ are solutions to φ. Consider a solution ν to φ, and assume that it does not satisfy any equation $a \cdot x + \tau + r = 0$ having $(a \cdot x + \tau) \in \mathrm{terms}(\varphi \wedge x \geq 0)$, $a \neq 0$, and $r \in [0.. |a| \cdot p - 1]$. We show that then $\nu' \coloneqq \nu + [x \mapsto p]$ is still a solution to φ. From $\Delta^p_x[f](\nu) > 0$, the value of the objective function f for the solution ν' is greater than the one for ν. In particular, this means that ν cannot be optimal.

First, observe that ν' still satisfies all inequalities in φ of the form $b \cdot x \leq \tau$ with $b \leq 0$, as well as all divisibility constraints. For the inequalities, this follows from $b \cdot \nu'(x) \leq b \cdot \nu(x) \leq \nu(\tau) = \nu'(\tau)$. For the divisibility constraints, it suffices to observe that $\nu(x)$ and $\nu'(x)$ have the same residue modulo $p = \mathit{mod}(x, \varphi)$. Consider then an inequality $a \cdot x \leq \tau$ with $a > 0$. We have $(a \cdot x - \tau) \in \mathrm{terms}(\varphi)$, and so ν does not satisfy any equation $a \cdot x - \tau + r = 0$ with $r \in [0..a \cdot p - 1]$. Hence, there is a positive integer $k \geq a \cdot b$ such that $a \cdot \nu(x) - \tau + k = 0$. We have $a \cdot \nu'(x) - \tau + k' = 0$, where $k' \coloneqq k - a \cdot b \geq 0$; that is, ν' satisfies $a \cdot x \leq \tau$.

case: $\Delta^p_x[f](\nu) < 0$ whenever ν and $\nu + [x \mapsto p]$ are solutions to φ. Consider a solution ν that does not satisfy any equation $a \cdot x + \tau + r = 0$, where $(a \cdot x + \tau) \in \mathrm{terms}(\varphi \wedge x \geq 0)$, $a \neq 0$, and $r \in [0.. |a| \cdot p - 1]$. This time we show that $\nu' \coloneqq \nu + [x \mapsto -p]$ is still a solution to φ.

From $\Delta^p_x[f](\nu) < 0$, the value that f takes for ν' is greater that the value that it takes for ν; and therefore ν cannot be optimal.

As in the previous case, it is trivial to see that ν' satisfies all inequalities of the form $b \cdot x \leq \tau$, with $b \geq 0$, as well as all divisibility constraints. Consider then an inequality $a \cdot x \leq \tau$ with $a < 0$. Since ν does not satisfy any equation $a \cdot x - \tau + r = 0$ with $r \in [0..\,|a| \cdot p - 1]$, we conclude that $\nu(\tau) \geq a \cdot \nu(x) + |a| \cdot p$. Since $a < 0$, we have $a \cdot \nu(x) + |a| \cdot p = a \cdot (\nu(x) - p)$. Hence, ν' satisfies $a \cdot x \leq \tau$.

case: $\Delta^p_x[f](\nu) = 0$ whenever ν and $\nu + [x \mapsto p]$ are solutions to φ. Let ν be an optimal solution. Recall that we are assuming that x occurs in an inequality $a \cdot x \leq \tau$ of φ, and in no equality. Suppose that $a > 0$ (the case for $a < 0$ is analogous). Let $a_1 \cdot x \leq \tau_1, \ldots, a_j \cdot x \leq \tau_j$ be an enumeration of all inequalities featuring x and such that $a_i > 0$. Let $k \in [1..j]$ satisfying

$$\frac{\nu(\tau_k)}{a_k} = \min\left\{\frac{\nu(\tau_i)}{a_i} : i \in [1..j]\right\}. \tag{38}$$

Let $\ell \in \mathbb{N}$ be the maximum natural number such that $a_k \cdot (\nu(x) + \ell \cdot p) \leq \nu(\tau_k)$, and define $\nu' \coloneqq \nu + [x \mapsto \ell \cdot p]$. We show that ν' is still an optimal solution to φ, and that it satisfies an equation $a_k \cdot x - \tau_k + r = 0$, for some $r \in [0..a_k \cdot p - 1]$.

To prove that ν' is a solution to φ, observe first that, exactly as in the first case of the proof ($\Delta^p_x[f](\nu) > 0$), the map ν' satisfies all inequalities in φ of the form $b \cdot x \leq \tau'$ with $b \leq 0$, as well as all divisibility constraints. Given an inequality $a_i \cdot x \leq \tau_i$ with $i \in [1..j]$, from Equation (38) we conclude that $\nu'(x) \leq \frac{\nu(\tau_k)}{a_k} \leq \frac{\nu(\tau_i)}{a_i}$, and therefore $a_i \cdot \nu'(x) \leq \nu(\tau_i) = \nu'(\tau_i)$. Hence, ν' is a solution to φ. It is also an optimal solution. Indeed, since the set of solutions of φ is (x, p)-periodic (Lemma 7), and $\Delta^p_x[f](\nu'') = 0$ whenever ν'' and $\nu'' + [x \mapsto p]$ are solutions to φ, we conclude that ν and the optimal solution ν' have the same value with respect to the objective function f.

Lastly, ℓ has been defined to be such that $a_k \cdot \nu'(x) \leq \nu(\tau_k) < a_k \cdot (\nu'(x) + p)$. Observe that $a_k \cdot (\nu'(x) + p) - a_k \cdot \nu'(x) = a_k \cdot p$. Therefore, there is an $r \in [0..a_k \cdot p - 1]$ such that $a_k \cdot \nu'(x) + r = \nu(\tau_k)$; that is, ν' satisfies $a_k \cdot x - \tau_k + r = 0$. □

Corollary 2. *Let φ be a linear-exponential program with divisions, and let x be a variable occurring linearly in φ. If φ has a solution, then it has one satisfying an equation $a \cdot x + \tau + r = 0$, where $(a \cdot x + \tau) \in \text{terms}(\varphi \wedge x \geq 0)$, $a \neq 0$, and $r \in [0..\,|a| \cdot \mathit{mod}(x, \varphi) - 1]$.* *Statement in page 21*

Proof. Consider the objective function f that simply returns the value assigned to x. This is clearly $(x, \mathit{mod}(x, \varphi))$-monotone locally to the set of solutions of φ. Since we are looking at non-negative solutions to φ, f ranges over the natural numbers, and so it has a minimum. The corollary follows then immediately from Lemma 8. □

Lemma 9. *Let φ be a linear-exponential program with divisions, and x be a variable occurring linearly in φ. Suppose that the set of solutions to φ has a $(x, \mathit{mod}(x, \varphi))$-monotone decomposition $R_1, \ldots, R_t$ for a function f, where each R_i is the set of solutions of an integer linear-exponential program with divisions ψ_i in which x occurs linearly. If the instance (f, φ) has a maximum (analogously, a minimum), then it has one satisfying an equation $a \cdot x + \tau + r = 0$ such that $a \neq 0$, $(a \cdot x + \tau) \in \text{terms}(\psi_i \wedge x \geq 0)$ and $r \in [0..\,|a| \cdot \mathit{mod}(x, \psi_i) - 1]$, for some $i \in [1..t]$.* *Statement in page 21*

Proof. By definition of monotone decomposition, φ is equivalent to $\psi_1 \vee \cdots \vee \psi_t$. Let ν be a solution to φ that maximizes (or minimizes) the objective function f. There is $i \in [1..t]$ such that ν is

a solution to ψ_i. Since ψ_i implies φ, the solution ν is optimal for (f, ψ_i). By Lemma 8, there is an optimal solution ν' to (f, ψ_i) that satisfies an equation $a \cdot x + \tau + r = 0$, where $a \neq 0$, $(a \cdot x + \tau) \in \text{terms}(\psi_i \wedge x \geq 0)$, and $r \in [0.. |a| \cdot \mathit{mod}(x, \psi_i) - 1]$. The value f takes with respect to the two solutions ν and ν' is the same, and since ψ_i implies φ, the map ν' is also a solution to φ. □

C.2 Proofs of Lemma 10 and Claim 2 from Section 4

Lemma 10. *Let $d, C \in \mathbb{R}$ with $d \geq 1$ and $C \geq 4 \cdot \log_2(d) + 8$. Then, $2^C - 2^{C/2} - d \cdot C \geq 0$.* *Statement in page 24*

Proof. From $C \geq 2$ we get $2^C - 2^{C/2} - d \cdot C \geq 2^{C/2}(2^{C/2} - 1) - d \cdot C \geq 2^{C/2} - d \cdot C$. So, it suffices to prove $2^{C/2} - d \cdot C \geq 0$. Consider the function $f(x) \coloneqq 2^{x/2} - d \cdot x$, as well as its first derivative $f'(x) = \frac{1}{2} \cdot \ln(2) \cdot 2^{x/2} - d$ and second derivative $f''(x) = \frac{1}{4} \cdot \ln(2)^2 \cdot 2^{x/2}$. Note that f'' is positive for all $x \geq 0$, and $f'(4 \cdot \log_2(d) + 8) = 8 \cdot \ln(2) \cdot d^2 - d \geq 4 \cdot d^2 - d \geq 0$. Therefore, f is increasing for every $x \geq 4 \cdot \log_2(d) + 8$, and $f(C) \geq f(4 \cdot \log_2(d) + 8) = 16 \cdot d^2 - d \cdot (4 \cdot \log_2(d) + 8) \geq 16 \cdot d^2 - 12 \cdot d^2 \geq 0$. □

The following claim refers to the objects defined throughout the proof of Proposition 3.

Claim 2. *For every $\sigma \in \mathcal{P}$, the function $C[x_m]$ is (q, p)-monotone locally to $\varphi \wedge \varphi[q + p \,/\, q] \wedge \chi_\sigma$.* *Statement in page 29*

Proof. Let us assume that the quantifier-free part of the formula χ_σ implies $\overline{x}_m \geq x_m$ (a similar argument applies if it instead implies $x_m \geq \overline{x}_m$). We show that for any two solutions ν and $\nu' \coloneqq \nu + [q \mapsto p]$ of $\varphi \wedge \varphi[q + p \,/\, q] \wedge \chi_\sigma$, the objective function $C[x_m]$ evaluated at ν' is at least as large as its value at ν.

Let ν and ν' be such a pair of solutions. We start by focusing on ν. Define $\nu(\boldsymbol{w})$ as the vector of integers obtained by evaluating C and C^{+p} according to ν, to then extract the values corresponding to the variables in $\boldsymbol{w}$ (which include $\overline{x}_m$ and x_m). Since χ_σ contains the equations describing the assignments in C and C^{+p}, the vector $\nu(\boldsymbol{w})$ is the only one that satisfies the quantifier-free part of χ_σ for the solution ν. Let a and $\overline{a}$ represent the values in $\nu(\boldsymbol{w})$ corresponding to x_m and $\overline{x}_m$, respectively. Specifically, a is the value taken by $C[x_m]$ for ν, and $\overline{a}$ is the value taken by $C^{+p}[\overline{x}_m]$ for ν. Since ν' also satisfies φ, Claim 1 implies that the value of $C[x_m]$ for ν' is the same as the value of $(\bar{x}_m, C^{+p})$ for ν, which is $\overline{a}$. Finally, because we assume that $\overline{x}_m \geq x_m$ is implied by the quantifier-free part of χ_σ, we conclude that $\overline{a} \geq a$, completing the proof. □

C.3 Bareiss algorithm

In this appendix we recall the classical Bareiss algorithm from [Bar68] and of some of its standard properties. These properties are then lifted to our variation in Appendix C.4, where we provide the proofs of Lemmas 15 to 18. Throughout this and the next appendix, we follow the notation introduced in Section 5.3, in particular when it comes to defining the subdeterminants $b^{(\ell)}_{i,j}$ and $b^{(\ell)}_{r \leftarrow j}$ of a given matrix with entries denotes as $b_{i,j}$.

We describe the standard Bareiss's algorithm from [Bar68]. Consider an $m \times d$ integer matrix B_0:

$$B_0 \coloneqq \begin{pmatrix} b_{1,1} & \dots & b_{1,d} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \dots & b_{m,d} \end{pmatrix}.$$

As done in Section 5.3, fix $k \in [0.. \min(m, d)]$ (the number of iterations the algorithm will perform), and define $\lambda_0 \coloneqq 1$ and $\lambda_\ell \coloneqq b^{(\ell-1)}_{\ell,\ell}$, for every $\ell \in [1..k]$. We assume that every λ_ℓ is non-zero.

Starting from the matrix B_0, Bareiss algorithm iteratively constructs a sequence of matrices $B_1, \dots, B_k$ as follows. Consider $\ell \in [0..k-1]$, and let B_ℓ be the matrix

$$B_\ell \coloneqq \begin{pmatrix} g_{1,1} & \cdots & g_{1,d} \\ \vdots & \ddots & \vdots \\ g_{m,1} & \cdots & g_{m,d} \end{pmatrix}.$$

The matrix $B_{\ell+1}$ is constructed from B_ℓ by applying the following transformation

1: **for** every row i except row $\ell+1$ **do**
2: multiply the ith row of B_ℓ by $g_{\ell+1,\ell+1}$ ▷ *the entry $(i, \ell+1)$ of B_ℓ is now $g_{\ell+1,\ell+1} \cdot g_{i,\ell+1}$*
3: subtract $g_{i,\ell+1} \cdot (g_{\ell+1,1}, \dots, g_{\ell+1,d})$ from the ith row of B_ℓ ▷ *the entry $(i, \ell+1)$ is now 0*
4: divide each entry of the ith row of B_ℓ by λ_ℓ ▷ *these divisions are without remainder*

The next three results from [Bar68] give a complete description of the entries of $B_0, \dots, B_k$: Lemma 33 contains a trivial observation that we will use many times in the other two lemmas, Lemma 34 describes the last $m-\ell$ rows of these matrices, and Lemma 35 describes the first ℓ rows.

Lemma 33 [Bar68]**.** *For all $\ell \in [0..k-1]$, the $(\ell+1)$th rows of the matrices B_ℓ and $B_{\ell+1}$ are equal.*

Lemma 34 [Bar68]**.** *Consider $\ell \in [0..k]$. For every $i \in [\ell+1..m]$ and $j \in [1..d]$, the entry in position (i,j) of the matrix B_ℓ is $b^{(\ell)}_{i,j}$. In particular, this entry is zero whenever $j \leq \ell$.*

Lemma 35 [Bar68]**.** *Consider $\ell \in [1..k]$. For all $i \in [1..\ell]$ and $j \in [1..d]$, the entry in position (i,j) of B_ℓ is $b^{(\ell)}_{i \leftarrow j}$. In particular, this entry is zero if $j \leq \ell$ and $i \neq j$, and it is instead $b^{(\ell-1)}_{\ell,\ell}$ when $i = j$.*

C.4 Analysis of the variation of Bareiss algorithm

We are now ready to analyze our variation of Bareiss algorithm, and prove Lemmas 15 to 18. Below, let k, B_0, B'_0, μ, U_g and λ_ℓ be defined as in Section 5.3. We recall below how the matrices $B'_1, \dots, B'_k$ are constructed from B'_0. Consider $\ell \in [0..k-1]$, and let B'_ℓ be the matrix

$$B'_\ell \coloneqq \begin{pmatrix} h_{1,1} & \dots & h_{1,d} \\ \vdots & \ddots & \vdots \\ h_{m,1} & \dots & h_{m,d} \end{pmatrix}.$$

The matrix $B'_{\ell+1}$ is constructed from B'_ℓ by applying the following transformation

01: **let** $\pm$ be the sign of $h_{\ell+1,\ell+1}$, and $\alpha \coloneqq \frac{\pm h_{\ell+1,\ell+1}}{\mu}$ ▷ *this division is without remainder*
02: multiply the row $\ell+1$ of B_ℓ by ± 1
03: **for** every row i except row $\ell+1$ **do**
04: **let** $\beta \coloneqq \frac{h_{i,\ell+1}}{\mu}$ ▷ *this division is without remainder*
05: multiply the ith row of B'_ℓ by α ▷ *$B'_\ell(i, \ell+1)$ is now $\alpha \cdot g_{i,\ell+1}$*
06: subtract $\pm\beta \cdot (h_{\ell+1,1}, \dots, h_{\ell+1,d})$ from the ith row of B'_ℓ
07: divide each entry of the ith row of B'_ℓ by $|\lambda_\ell|$ ▷ *these divisions are without remainder*

We now show that the main relationship between the matrices computed with the above transformation, and the sequence of matrices $B_0, \dots, B_k$ computed with Bareiss algorithm.

Lemma 36. *For every $\ell \in [0..k]$, $B'_\ell = \pm B_\ell \cdot U_g$, where $\pm$ is the sign of λ_ℓ.*

Proof. Below, we write $\pm_\ell$ for the sign of λ_ℓ. The proof is by induction on ℓ.

base case: $\ell = 0$. Trivial from the definition of B'_0 (recall that $\lambda_0 = 1$ in this case).

induction hypothesis. Given $\ell \geq 1$, we have $B'_{\ell-1} = \pm_{\ell-1} B_{\ell-1} \cdot U_g$.

induction step: $\ell \geq 1$. From the induction hypothesis, for every $i \in [1..m]$ the ith rows $\boldsymbol{r}_i$ and $\boldsymbol{r}'_i$ of $B_{\ell-1}$ and $B'_{\ell-1}$ are, respectively,

$$\boldsymbol{r}_i = (r_{i,1}, \ldots, r_{i,d}), \qquad \boldsymbol{r}'_i = \pm_{\ell-1}(\mu \cdot r_{i,1}, \ldots, \mu \cdot r_{\ell,g}, r_{i,g+1}, \ldots, r_{i,d}),$$

for some integers $r_{i,1}, \ldots, r_{i,d}$. To show that $B'_\ell = \pm_\ell B_\ell \cdot U_g$, we analyze the pseudocodes used to construct B_ℓ and B'_ℓ, considering each row separately.

Let us start by considering the ℓth row. In the case of B_ℓ, this row coincides with $\boldsymbol{r}_\ell$. In the case of B'_ℓ, this row is instead $\pm\boldsymbol{r}'_\ell$, where $\pm$ is the sign of $\pm_{\ell-1} \cdot \mu \cdot r_{\ell,\ell}$. By Lemma 34 and from the definition of λ_ℓ, we have $r_{\ell,\ell} = \lambda_\ell$. Since $\mu \geq 1$, $\pm 1 = \pm_{\ell-1} \pm_\ell 1$, and therefore $\pm\boldsymbol{r}'_\ell$ is equal to $\pm_\ell(\mu \cdot r_{\ell,1}, \ldots, \mu \cdot r_{\ell,g}, r_{\ell,g+1}, \ldots, r_{\ell,d})$, as required.

Consider now $i \in [1..m]$ with $i \neq \ell$. Following the code of Bareiss algorithm, the ith row of B_ℓ is given by $\frac{1}{\lambda_{\ell-1}} \cdot (r_{\ell,\ell} \cdot \boldsymbol{r}_i - r_{i,\ell} \cdot \boldsymbol{r}_\ell)$. The entry of B_ℓ in position (i,j), with $j \in [1..d]$, is thus

$$t_{i,j} \coloneqq \frac{r_{\ell,\ell} \cdot r_{i,j} - r_{i,\ell} \cdot r_{\ell,j}}{\lambda_{\ell-1}}.$$

The ith row of B'_ℓ is instead $\frac{1}{|\lambda_{\ell-1}|} \cdot (\alpha \cdot \boldsymbol{r}'_i - \pm\beta \cdot \boldsymbol{r}'_\ell)$, where $\pm$ is again the sign of $\pm_{\ell-1}\mu \cdot r_{\ell,\ell}$, and $\alpha \coloneqq \frac{\pm\pm_{\ell-1}\mu \cdot r_{\ell,\ell}}{\mu} = \frac{\pm_\ell \mu \cdot r_{\ell,\ell}}{\mu}$ and $\beta \coloneqq \frac{\pm_{\ell-1}\mu \cdot r_{i,\ell}}{\mu}$ are defined as in line 04 of the pseudocode. For every $j \in [1..g]$, the entry of B'_ℓ in position (i,j) is therefore

$$\begin{aligned} t'_{i,j} &\coloneqq \frac{\alpha \cdot (\pm_{\ell-1}\mu \cdot r_{i,j}) - \pm\beta \cdot (\pm_{\ell-1}\mu \cdot r_{\ell,j})}{|\lambda_{\ell-1}|} \\ &= \frac{\pm_\ell r_{\ell,\ell} \cdot (\pm_{\ell-1}\mu \cdot r_{i,j}) - \pm_\ell r_{i,\ell}(\pm_{\ell-1}\mu \cdot r_{\ell,j})}{\pm_{\ell-1}\lambda_{\ell-1}} = \pm_\ell \mu \cdot \frac{r_{\ell,\ell} \cdot r_{i,j} - r_{i,\ell} \cdot r_{\ell,j}}{\lambda_{\ell-1}} = \pm_\ell \mu \cdot t_{i,j}. \end{aligned}$$

An analogous manipulation shows $t'_{i,j} = \pm_\ell t_{i,j}$, for every $j \in [g+1..d]$. We thus have $(t'_{i,1}, \ldots, t'_{i,d}) = \pm_\ell(\mu \cdot t_{i,1}, \ldots, \mu \cdot t_{i,g}, t_{i,g+1}, \ldots, t_{i,d})$, which concludes the proof. □

By relying on Lemma 36, we can easily rephrase the properties of Bareiss algorithm from Appendix C.3 to our variation, proving Lemmas 15 to 17.

Lemma 15. *For all $\ell \in [0..k-1]$, the $(\ell+1)$th row of $B'_{\ell+1}$ is obtained by multiplying the $(\ell+1)$th row of B'_ℓ by the sign of its $(\ell+1)$th entry.* *Statement in page 41*

Proof. This is a simple observation on the effects of lines 01 and 02 of the procedure. □

Lemma 16. *Consider $\ell \in [0..k]$ and $i \in [\ell+1..m]$, and let $\pm$ be the sign of λ_ℓ. Then:* *Statement in page 41*

1. *For every $j \in [1..g]$, the entry in position (i,j) of the matrix B'_ℓ is $\pm\mu \cdot b^{(\ell)}_{i,j}$. (In particular, this entry is zero whenever $j \leq \ell$.)*
2. *For every $j \in [g+1..d]$, the entry in position (i,j) of the matrix B'_ℓ is $\pm b^{(\ell)}_{i,j}$.*

Proof. Directly from Lemma 36 and Lemma 34. □

Lemma 17. *Consider $\ell \in [1..k]$ and $i \in [1..\ell]$, and let $\pm$ be the sign of λ_ℓ. Then:* *Statement in page 41*

1. *For every $j \in [1..g]$, the entry in position (i,j) of B'_ℓ is $\pm\mu \cdot b^{(\ell)}_{i\leftarrow j}$.*
   (In particular, this entry is zero if $j \le \ell$ and $i \neq j$, and it is instead $\pm\mu \cdot b^{(\ell-1)}_{\ell,\ell}$ when $i = j$.)
2. *For every $j \in [g+1..d]$, the entry in position (i,j) of B'_ℓ is $\pm b^{(\ell)}_{i\leftarrow j}$.*

Proof. Directly from Lemma 36 and Lemma 35. □

Lastly, we move to the proof of Lemma 18, which relies on Laplace expansions. Lemma 37 below recalls this notion using determinants of the form $a^{(\ell)}_{r\leftarrow j}$. These determinants have two straightforward properties:

1. If $j \le \ell$ and $r \neq j$, then the jth and rth columns of the submatrix corresponding to this determinant are identical. Therefore, $a^{(\ell)}_{r\leftarrow j} = 0$.
2. If $j = r$ then $a^{(\ell)}_{j\leftarrow j} = a^{(\ell-1)}_{\ell,\ell}$.

Lemma 37 (Laplace Expansion)**.** *For every $i, j, \ell \in \mathbb{N}$ satisfying $\ell \ge 1$, $\ell < i \le m$ and $\ell < j \le d$,*

$$a^{(\ell)}_{i,j} = a^{(\ell-1)}_{\ell,\ell} \cdot a_{i,j} - \sum\nolimits_{r=1}^{\ell} a^{(\ell)}_{r\leftarrow j} \cdot a_{i,r}. \tag{39}$$

Proof. Equation (39) is not the standard way of writing the Laplace expansion, but it is one that will be very natural for our purposes. A more standard way of writing this identity is:

$$a^{(\ell)}_{i,j} = a^{(\ell-1)}_{\ell,\ell} \cdot a_{i,j} + \sum\nolimits_{r=1}^{\ell} (-1)^{r+\ell+1} m_r \cdot a_{i,r}\,, \tag{40}$$

where $m_r \coloneqq \det \begin{pmatrix} a_{1,1} & \dots & a_{1,r-1} & a_{1,r+1} & \dots & a_{1,\ell} & a_{1,j} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{\ell,1} & \dots & a_{\ell,r-1} & a_{\ell,r+1} & \dots & a_{\ell,\ell} & a_{\ell,j} \end{pmatrix}$.

The matrix M used to compute m_r features the same columns as the matrix M' used to compute $a^{(\ell)}_{r\leftarrow j}$. In particular, M' is obtained from M by applying a cyclic permutation to the last $\ell - r + 1$ columns, so that the column $(a_{1,j}, \dots, a_{\ell,j})$ appears first among those. Recall that swapping two rows of a matrix changes the sign of its determinant. The permutation used to compute M' form M can be realized with $\ell - r$ swaps, hence $(-1)^{\ell-r} m_r = a^{(\ell)}_{r\leftarrow j}$. Therefore, $(-1)^{r+\ell+1} m_r = (-1)^{r+\ell+1+r-r} m_r = -a^{(\ell)}_{r\leftarrow j}$, showing that Equations (39) and (40) are equivalent. □

Lemma 18. *Let $\ell \in [0..k]$ and $i \in [\ell+1..m]$. Consider the following transformation applied to B'_0:* *Statement in page 42*

1: multiply the ith row of B'_0 by $|\lambda_\ell|$
2: ***for*** *r in $[1..\ell]$* ***do*** *subtract $b_{i,r} \cdot \boldsymbol{u}_r$ to the ith row of B'_0, where $\boldsymbol{u}_r$ is the rth row of B'_ℓ*

After the transformation, the ith rows of B'_0 and B'_ℓ are equal.

Proof. Note that if $\ell = 0$, then the transformation does not modify B_0 (recall that $\lambda_0 = 1$), and the lemma is therefore trivially true. Let us consider then $\ell \ge 1$, and write $\boldsymbol{v}$ for the ith row of B'_0 *after* the transformation. Let us compute an expression for the entries in $\boldsymbol{v}$. After executing line 1, the ith

row of B_0 is of the form $\pm b_{\ell,\ell}^{(\ell-1)} \cdot (\mu \cdot b_{i,1}, \ldots, \mu \cdot b_{i,g},\, b_{i,g+1} \ldots, b_{i,d})$, where $\pm$ is the sign of $\lambda_\ell = b_{\ell,\ell}^{(\ell-1)}$. From Lemma 17, the vector $\boldsymbol{u}_r$ from line 2 is $\boldsymbol{u}_r = \pm(\mu \cdot b_{r\leftarrow 1}^{(\ell)}, \ldots, \mu \cdot b_{r\leftarrow g}^{(\ell)},\, b_{r\leftarrow g+1}^{(\ell)}, \ldots, b_{r\leftarrow d}^{(\ell)})$. After performing all subtractions from line 2 to the ith row, we obtain the vector $\boldsymbol{v}$. Its jth entry is:

$$\pm\mu_j \cdot (b_{\ell,\ell}^{(\ell-1)} \cdot b_{i,j} - \sum\nolimits_{r=1}^{\ell} b_{r\leftarrow j}^{(\ell)} \cdot b_{i,r}),$$

where $\mu_j \coloneqq \mu$ whenever $j \leq g$, and otherwise $\mu_j \coloneqq 1$.

We show that the expression $(b_{\ell,\ell}^{(\ell-1)} \cdot b_{i,j} - \sum_{r=1}^{\ell} b_{r\leftarrow j}^{(\ell)} \cdot b_{i,r})$ equals $b_{i,j}^{(\ell)}$, thus establishing that $\boldsymbol{v}$ is the ith row of B'_ℓ by Lemma 16. When $j > \ell$, this results follows directly from Lemma 37. When $j \leq \ell$ instead, observe that $b_{i,j}^{(\ell)} = 0$ (Lemma 16), and it suffices to show that the expression $(b_{\ell,\ell}^{(\ell-1)} \cdot b_{i,j} - \sum_{r=1}^{\ell} b_{r\leftarrow j}^{(\ell)} \cdot b_{i,r})$ is also zero. For every $r \in [1..\ell]$ with $r \neq j$, the determinant $b_{r\leftarrow j}^{(\ell)}$ is zero; as the jth and rth columns of the corresponding matrix are identical. The expression thus simplifies to $(b_{\ell,\ell}^{(\ell-1)} \cdot b_{i,j} - b_{j\leftarrow j}^{(\ell)} \cdot b_{i,j}) \;=\; (b_{\ell,\ell}^{(\ell-1)} \cdot b_{i,j} - b_{\ell,\ell}^{(\ell-1)} \cdot b_{i,j}) \;=\; 0$. □

C.5 Proofs of Lemma 13 from Section 5

Lemma 13. *Given in input a triple $(\boldsymbol{q}_{k-1}, C[x_m], \langle\gamma\,;\psi\rangle)$ with $(C, \langle\gamma\,;\psi\rangle) \in \mathcal{I}_k^\ell$ Algorithm 4 guesses in line 2 a linear term $a \cdot q_{n-\ell} - \tau(u, \boldsymbol{q}_{[\ell+1,k]})$ in which all coefficients of $\boldsymbol{q}_{[\ell,k]}$ are divisible by μ_C.* *Statement in page 36*

Proof. Let us first observe that γ contains an inequality $a \cdot q_{n-\ell} \geq 0$ for some $a \geq 1$, by definition of $\mathcal{I}_k^\ell$, and therefore terms(γ) contains $-a \cdot q_{n-\ell}$. This implies that the guess performed in line 2 is never on an empty set.

Let $\rho \coloneqq (a \cdot q_{n-\ell} - \tau)$ be the guessed term. By definition of $\mathcal{I}_k^\ell$, γ is a linear program in variables u and $\boldsymbol{q}_{[\ell,k]}$, and in which every inequality and equality is such that all the coefficients of the variables in $\boldsymbol{q}_{[\ell,k]}$ are divisible by μ_C. The statement is thus true when ρ belongs to terms($\gamma \wedge \gamma[q_{n-\ell} + p \,/\, q_{n-\ell}]$).

Suppose ρ to be instead computed using Algorithm 2. In this case there are $b, d \in \mathbb{Z}$, and q', q'' from $\boldsymbol{q}_{k-1}$, such that ρ is obtained from $b \cdot u + \mu_C \cdot (q' - q'') + d$ by simultaneously applying two substitutions $[\frac{\tau'}{\lambda} \,/\, \mu_C \cdot q']$ and $[\frac{\tau''}{\lambda} \,/\, \mu_C \cdot q'']$. By definition of simultaneous substitution, ρ is thus of the form $(\lambda \cdot b \cdot u \pm \tau' \mp \tau'' + \lambda \cdot d)$, where the signs $\pm$ and $\mp$ depend on the sign of λ. Here, $\lambda \coloneqq \frac{\eta_C}{\mu_C}$, and the terms τ' and τ'' are computed as described in lines 4–7. From the definition of $\mathcal{I}_k^\ell$, recall that μ_C divides η_C, as well as all coefficients of the variables $\boldsymbol{q}_{[\ell,k]}$ occurring in linear terms $\tau_{n-i}(u, \boldsymbol{q}_{[\ell,k]})$ featured in assignments $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_C}$ of C, with $i \in [0..\ell-1]$. Looking at lines 4–7, it is then easy to see that the terms τ' and τ'' are linear terms in variables u and $\boldsymbol{q}_{[\ell,k]}$, and that in these terms the coefficients of $\boldsymbol{q}_{[\ell,k]}$ are all divisible by μ_C. Then, the same is true for the term $\pm\tau' \mp \tau''$, and in turn also for ρ. □

C.6 Proofs of the claims from Lemma 20 (Section 5)

The following claims refer to objects defined throughout the proof of Lemma 20.

Claim 9. *The ℓth rows of M_ℓ and B'_ℓ are equal. Moreover, $\eta_\ell = \pm a$ and $\alpha = \frac{\eta_\ell}{\mu} = |\lambda_\ell| \neq 0$.* *Statement in page 46*

Proof. By induction hypothesis, the ℓth rows of $M_{\ell-1}$ and $B'_{\ell-1}$ are equal, and by Claim 8, they contain the variable coefficients of $a \cdot q_{n-(\ell-1)} - \tau$. Following Bareiss algorithm (see line 02 and Lemma 15), the ℓth row of B'_ℓ contains the variable coefficients of $\pm a \cdot q_{n-(\ell-1)} - (\pm\tau)$.

In the case of M_ℓ, from its definition (Item (i)), the ℓth row contains the variable coefficients of the term $\eta_\ell \cdot q - \tau'$ such that $q \leftarrow \frac{\tau'}{\eta_\ell}$ is the first assignment in C_ℓ. From line 1 and the last item in line 7 of Algorithm 5, we conclude that this assignment is $q_{n-(\ell-1)} \leftarrow \frac{\pm\tau}{\pm a}$. Therefore, the ℓth rows of M_ℓ and B'_ℓ are equal.

Again from 7 of Algorithm 5, we see that in C_ℓ all assignments to variables in $\boldsymbol{q}_k$ have $\pm a$ as a denominator, that is, $\eta_\ell = \pm a$. Furthermore, from Lemma 17, the entry of B'_ℓ in position (ℓ,ℓ) is $\pm'\mu \cdot b^{(\ell-1)}_{\ell,\ell}$, where $\pm'$ is the sign of $\lambda_\ell = b^{(\ell-1)}_{\ell,\ell}$. By definition of M_ℓ, the ℓth column contains the coefficients of $q_{n-(\ell-1)}$. This means that the entry of M_ℓ in position (ℓ,ℓ) is $\pm a$, and therefore $\pm'\mu \cdot \lambda_\ell = \pm a$. It follows that $\alpha = \frac{\eta_\ell}{\mu} = |\lambda_\ell| \neq 0$. □

Claim 11. *Let $i \in [j+1..j+t]$. The ith rows of M_ℓ and B'_ℓ are equal. Moreover, in all equalities and inequalities $\gamma_{\ell-1}[\frac{\pm\tau}{\alpha} \,/\, \mu \cdot q_{n-\ell-1}]$, all coefficients of the variables $\boldsymbol{q}_{[\ell,k]}$ are divisible by $\eta_{\ell-1}$, and all coefficients of u are divisible by $\frac{\eta_{\ell-1}}{\mu}$.* Statement in page 46

Proof. This proof is similar to the one of Claim 10. By definition, the ith rows of $M_{\ell-1}$ and M_ℓ contain the variable coefficients of the terms in the (in)equalities $\Lambda_{\ell-1}(\rho_{i-j} \sim_{i-j} 0)$ and $\Lambda_\ell(\rho_{i-j} \sim_{i-j} 0)$, respectively. By definition of $\Lambda_{\ell-1}$ and Λ_ℓ, $\Lambda_\ell(\rho_{i-j} \sim_{i-j} 0)$ is the (in)equality obtained from $\Lambda_\ell(\rho_{i-j} \sim_{i-j} 0)$ when running lines 3–6 of Algorithm 5. Below we analyze the updates performed in these lines of the algorithm, and compare them to those Bareiss algorithm performs on the ith row of $B'_{\ell-1}$ in order to produce B'_ℓ.

Let us write $\beta \cdot \mu \cdot q_{n-(\ell-1)} + \tau'$ for the term in the (in)equality $\Lambda_{\ell-1}(\rho_{i-j} \sim_{i-j} 0)$. Line 3 applies the substitution $[\frac{\pm\tau}{\alpha} \,/\, \mu \cdot q_{n-(\ell-1)}]$ on this term, obtaining the term $\pm\beta \cdot \tau + \alpha \cdot \tau'$. Then, following lines 5 and 6, we see that the ith row of M_ℓ holds the variable coefficients of the term $(\pm\beta \cdot \tau + \alpha \cdot \tau')/\frac{\eta_{\ell-1}}{\mu}$ obtained by dividing each integer in $\pm\beta \cdot \tau + \alpha \cdot \tau'$ by $\frac{\eta_{\ell-1}}{\mu}$. As in the proof of Claim 10, we will see below that all variable coefficients of $\pm\beta \cdot \tau + \alpha \cdot \tau'$ are divisible by $\frac{\eta_{\ell-1}}{\mu}$.

Let us look at Bareiss algorithm. By induction hypothesis, the ith row of $B'_{\ell-1}$ contains the variable coefficients of $\beta \cdot \mu \cdot q_{n-(\ell-1)} + \tau'$. The algorithm first multiplies this row by α, and then subtracts $\pm\beta \cdot \boldsymbol{r}_\ell$, where $\boldsymbol{r}_\ell$ is the ℓth row of $B'_{\ell-1}$. By Claim 8, $\boldsymbol{r}_\ell$ holds the variable coefficients of $a \cdot q_{n-(\ell-1)} - \tau$. Hence, after this subtraction, the ith row of $B'_{\ell-1}$ is updated to contain the variable coefficients of the term $\pm\beta \cdot \tau + \alpha \cdot \tau'$. Lastly, each entry of the ith row is divided by $|\lambda_{\ell-1}| = \frac{\eta_{\ell-1}}{\mu}$. From Lemma 16, these divisions are exact. This means that every variable coefficient in $\pm\beta \cdot \tau + \alpha \cdot \tau'$ is divisible by $\frac{\eta_{\ell-1}}{\mu}$; so the divisions performed in lines 5 and 6 of Algorithm 5 are also without remainder. The divisions performed by Bareiss algorithm completes the construction of the ith row of B_ℓ, which thus contain the variable coefficients of $(\pm\beta \cdot \tau + \alpha \cdot \tau')/\frac{\eta_{\ell-1}}{\mu}$. Hence, the ith rows of M_ℓ and B'_ℓ coincide.

Let us discuss the second statement of the claim. Every (in)equality in $\gamma_{\ell-1}[\frac{\pm\tau}{\alpha} \,/\, \mu \cdot q_{n-\ell-1}]$ is obtained by applying the substitution $[\frac{\pm\tau}{\alpha} \,/\, \mu \cdot q_{n-\ell-1}]$ to an (in)equality $\Lambda_{\ell-1}(\rho_{i-j} \sim_{i-j} 0)$, with $i \in [j+1..j+t]$. Following the notation above, let $\pm\beta \cdot \tau + \alpha \cdot \tau'$ be the term resulting from one such substitution. We have already shown that all variable coefficients of this term are divisible by $\frac{\eta_{\ell-1}}{\mu}$. Then, the coefficient of u is divisible by $\frac{\eta_{\ell-1}}{\mu}$. As for the remaining variables, Lemma 16 guarantee that, once divided by $\frac{\eta_{\ell-1}}{\mu}$, their coefficients are still divisible by μ; hence before divisions these coefficients are divisible by $\eta_{\ell-1}$. □

Claim 12. *Let $i \in [\ell+1..j]$. The ith rows of M_ℓ and B'_ℓ are equal.* Statement in page 46

Proof. By definition, the contents of the ith rows of $M_{\ell-1}$ and M_ℓ depend on the type of the equality e_{i-1}. We divide the proof in two cases, depending on this type.

If e_{i-1} is of Type I, then by definition the ith rows of $M_{\ell-1}$ and M_ℓ contain the variable coefficients of the terms in the (in)equalities $\Lambda_{\ell-1}(g_{i-1})$ and $\Lambda_\ell(g_{i-1})$, respectively. Moreover, the generator g_{i-1} of e_{i-1} is an (in)equality of γ_0. Therefore, there is $r \in [j+1..j+t]$ such that the ith and rth rows of $M_{\ell-1}$ (resp. M_ℓ) are equal. Since Bareiss algorithm performs the same updates on every row different from ℓ, the claim then follows from Claim 11.

Suppose now e_{i-1} to be of Type II. Let $g_{i-1} \coloneqq (\rho = 0)$ be the generator of e_{i-1}. Recall that ρ is of the form $b \cdot u + \mu \cdot (q' - q'') + d$, for some $b, d \in \mathbb{Z}$ and q', q'' from $\boldsymbol{q}_k$. By Item (iii) in the definition of M_ℓ, the ith row of M_ℓ contains the coefficients of the variables $\boldsymbol{q}_k$ and u from the term obtained from ρ as follows:

1: multiply every integer in ρ by the quotient of the division of η_ℓ by μ
2: **for** r in $[1..\ell]$ **do** $\rho \leftarrow \rho[\tau_r \,/\, \eta_\ell \cdot q]$, where $q \leftarrow \frac{\tau_r}{\eta_\ell}$ is the rth assignment in C_ℓ

Moreover, again by definition of M_ℓ, the variable coefficients of the term $\eta_\ell \cdot q - \tau_r$ corresponding to the rth assignment in C_ℓ is are stored in the rth row of M_ℓ. By Claims 9 and 10, this row is equal to the rth row of B'_ℓ. Therefore, from Lemma 17, we conclude that q is the variable $q_{n-(r-1)}$, and that τ_r is a linear term in variables u and $\boldsymbol{q}_{[\ell,k]}$. Note that then, the terms $\tau_1, \ldots, \tau_\ell$ do not feature any of the variables $q_{n-(\ell-1)}, \ldots, q_n$, which means that the code above is in fact *simultaneously* applying the substitutions $[\frac{\tau_1}{\alpha} \,/\, \mu \cdot q_n], \ldots, [\frac{\tau_\ell}{\alpha} \,/\, \mu \cdot q_{n-(\ell-1)}]$ to ρ (by Claim 9, $\alpha = \frac{\eta_\ell}{\mu}$). We conclude that the ith row of M_ℓ holds the variable coefficients of

$$\alpha \cdot b \cdot u + \tau' - \tau'' + \alpha \cdot d, \tag{41}$$

where τ' stands for $\eta_\ell \cdot q'$ if C_ℓ assigns no expression to q', and otherwise it is the term such that $q' \leftarrow \frac{\tau'}{\eta_\ell}$ occurs in C_ℓ; and similarly, τ'' stands for $\eta_\ell \cdot q''$ if C_ℓ assigns no expression to q', and otherwise it is the term such that $q'' \leftarrow \frac{\tau''}{\eta_\ell}$ occurs in C_ℓ.

Let us look at Bareiss algorithm. By Lemma 18, the ith row of B'_ℓ can be computed as follows:

1: multiply the ith row of B'_0 by $|\lambda_\ell|$
2: **for** r in $[1..\ell]$ **do** subtract $b_{i,r} \cdot \boldsymbol{u}_r$ to the ith row of B'_0, where $\boldsymbol{u}_r$ is the rth row of B'_ℓ

Above $b_{i,r}$ is the entry of B_0 in position (i, r) —whereas the entry of B'_0 in that position is $\mu \cdot b_{i,r}$. Also, by Claim 9, $|\lambda_\ell| = \alpha$. Following Claims 9 and 10, the ith row of B'_ℓ contains the variable coefficients of the term

$$\alpha \cdot b \cdot u + \eta_\ell \cdot (q' - q'') + \alpha \cdot d - \sum\nolimits_{r=1}^{\ell} b_{i,r} \cdot (\eta_\ell \cdot q_{n-(r-1)} - \tau_r) \tag{42}$$

Now, if $q' = q''$, then all entries $b_{i,s}$ of B_0, where ranges in $s \in [1..k{+}1]$, are zero, and so Equation (42) simplifies to $\alpha \cdot b \cdot u + \alpha \cdot d$; which is equal to the term in Equation (41), since $q' = q''$ implies $\tau' = \tau''$. If instead $q' \neq q''$, then all entries $b_{i,s}$ of B_0 (with $s \in [1..k+1]$) are equal to zero, except for the entry in the position corresponding to q', which is equal to 1, and the entry in the position corresponding to q'', which is equal to -1. Equation (42) can be rewritten as

$$\alpha \cdot b \cdot u + \eta_\ell \cdot (q' - q'') + \alpha \cdot d - \rho' + \rho'', \tag{43}$$

where $\rho' \coloneqq 0$ if the column corresponding to q' is not among the first ℓ, and otherwise $\rho' \coloneqq (\eta_\ell \cdot q' - \tau')$, with $q' \leftarrow \frac{\tau'}{\eta_\ell}$ occurring in C_ℓ; and similarly, $\rho'' \coloneqq 0$ if the column corresponding to q'' is not among the first ℓ, and otherwise $\rho'' \coloneqq (\eta_\ell \cdot q'' - \tau'')$, with $q'' \leftarrow \frac{\tau''}{\eta_\ell}$ occurring in C_ℓ. The terms in Equations (41) and (43) are thus equal, proving the claim. □

C.7 Proof of Claim 6 from Section 5

Claim 6. *The following property is true across all the executions of Algorithm 5 performed in all non-deterministic branches of* ElimVars*, on any of its inputs. In all equalities and inequalities of the formula $\gamma[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ computed in line 3, and in terms $\tau_{n-i}[\frac{\tau}{\alpha} \,/\, \mu_C \cdot q_{n-\ell}]$ computed in line 7, all coefficients of the variables $\boldsymbol{q}_{[\ell+1,k]}$ are divisible by η_C, and all coefficients of u are divisible by $\frac{\eta_C}{\mu_C}$.*

Proof. Following the explanation provided as the start of the induction step of the proof of Lemma 20, this lemma implies that $(C_\ell, \langle \gamma_\ell \,;\, \psi \rangle) \in \mathcal{I}_k^\ell$ for every $\ell \in [0..j]$. In particular, this ensures that the **while** loop of GaussQE iterates at most k times (possibly fewer, if an **assert** command in Algorithm 5 fails). Consequently, by examining all truncations of the non-deterministic branches in Equation (18) of length up to k, we have in fact accounted for all possible non-deterministic executions of GaussQE. Then, the statement from Lemma 20 "if $\ell \geq 1$, then Claim 6 holds when restricted to Algorithm 5 having as input $(C_{\ell-1}[x_m], \langle \gamma_{\ell-1} \,;\, \psi \rangle)$ and the equality $e_{\ell-1}$" generalizes to all inputs of Algorithm 5; that is, Claim 6 holds. □

C.8 Proof of Lemma 21 from Section 5

Lemma 21. *The algorithm from Lemma 14 runs in non-deterministic polynomial time. Consider its execution on an input $(\boldsymbol{q}, C[x_m], \langle \gamma \,;\, \psi \rangle)$, where $(C, \langle \gamma \,;\, \psi \rangle)$ belongs to $\mathcal{I}_k^0$, and define:* *Statement in page 47*

$$L \coloneqq 3 \cdot \mu_C \cdot (4 \cdot \lceil \log_2(2 \cdot \xi_C + \mu_C) \rceil + 8),$$
$$Q \coloneqq \max\{|b| : b \in \mathbb{Z} \text{ is a coefficient of } q_{n-k} \text{ or of a variable in } \boldsymbol{q}, \text{ in a term from } \mathrm{terms}(\gamma)\},$$
$$U \coloneqq \max\{|a| : a = L \text{ or } a \in \mathbb{Z} \text{ is a coefficient of } u \text{ in a term from } \mathrm{terms}(\gamma)\},$$
$$R \coloneqq \max\{|d| : d = L \text{ or } d \in \mathbb{Z} \text{ is a constant of a term from } \mathrm{terms}(\gamma)\}.$$

In each non-deterministic branch β, the algorithm returns a pair $(C'[x_m], \langle \gamma' \,;\, \psi \rangle)$ such that:

1. *γ' features k constraints more than γ, they are all divisibility constraints.*
2. *The circuits C and C' assign the same expressions to $x_{n-k}, \ldots, x_n$ (in particular, $\mu_C = \mu_{C'}$).*
3. *In terms τ either from $\mathrm{terms}(\gamma')$ or in assignments $q_{n-i} \leftarrow \frac{\tau}{\eta_{C'}}$ of C' (where $i \in [0..k-1]$),*
   - *the coefficient of the variable q_{n-k} is $\mu_C \cdot c$, for some $c \in \mathbb{Z}$ with $|c| \leq (k+1)^{k+1}\big(\frac{Q}{\mu_C}\big)^{k+1}$;*
   - *the absolute value of the coefficient of the variable u is bounded by $(k+1)^{k+1}\big(\frac{Q}{\mu_C}\big)^{k} U$;*
   - *the absolute value of the constant is bounded by $\frac{((k+1)\cdot Q)^{2(k+2)^2}}{(\mu_C)^{2k^2}} \cdot \mathit{mod}(\gamma) \cdot R$.*
4. *The positive integer $\mathit{mod}(\gamma')$ divides $c \cdot \mathit{mod}(\gamma)$, for some positive integer $c \leq \frac{(k\cdot Q)^{k^2}}{(\mu_C)^{k(k-1)}}$.*
5. *We have $\eta_{C'} = \mu_C \cdot g$, for some positive integer $g \leq k^k\big(\frac{Q}{\mu_C}\big)^k$.*

Proof. Let $j \in [0..k]$. Throughout the proof, we refer to the pair (C_j, γ_j) and the matrices M_j, B'_j and B_0 defined in Section 5.4. (Recall that M_j is encoding the coefficients that the variables $\boldsymbol{q}_k$ and u have in C_j and γ_j; see Lemma 19.) We let $\mu \coloneqq \mu_C$, and write $\pm$ for the sign of the determinant $\lambda_j \coloneqq b_{j,j}^{(j-1)}$ (postulating $\lambda_0 \coloneqq 1$). We recall that, directly from the Leibniz formula for determinants, one obtains $|\det(A)| \leq d^d \cdot \prod_{i=1}^{d} \alpha_i$ for any $d \times d$ integer matrix A in which the entries of the ith column are bounded, in absolute value, by $\alpha_i \in \mathbb{N}$. Let us start with a simple observation on the entries of B_0, which follows directly from the definition of this matrix:

Claim 21. *For every $i \in [1..j+t]$, $|b_{i,k+2}| \leq U$ and, for every $j \in [1..k+1]$, $|b_{i,j}| \leq \frac{Q}{\mu}$.*

Next, we bound the coefficients of the variables $\boldsymbol{q}_k$ and u occurring in C_j.

Claim 22. *Let $i \in [0..j-1]$, and $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_j}$ be an assignment in C_j. In the term τ_{n-i}:*

- *The coefficients of the variables $\boldsymbol{q}_{[0,j-1]}$ are zero.*

- *Each coefficient of a variable in $\boldsymbol{q}_{[j,k]}$ is $\mu \cdot d$, for some $d \in \mathbb{Z}$ such that $|d| \leq j^j \left(\frac{Q}{\mu}\right)^j$.*
- *The coefficient of the variable u is bounded, in absolute value, by $j^j \left(\frac{Q}{\mu}\right)^{j-1} U$.*

Moreover, $\eta_j = \mu \cdot g$ for some positive integer $g \leq j^j \left(\frac{Q}{\mu}\right)^j$.

Proof. For $j = 0$ the circuit C_0 features no assignment to variables in $\boldsymbol{q}_k$, and thus the claim is trivially true. Assume then $j \geq 1$. Let $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_j}$ be an assignment in C_j. From the proof of correctness of ElimVars (Lemma 14), C_j is a (k, j)-LEAC, and so the coefficients of the variables $\boldsymbol{q}_{[0,j-1]}$ in the term τ_{n-i} are zero. By definition of M_j, the coefficients of the term $\eta_j \cdot q_{n-i} - \tau_{n-i}$ are found in the $(i+1)$th rows of M_j. By Lemma 20, they are also found in the $(i+1)$th row of B'_j.

Given $\ell \in [j..k]$, let us first consider the coefficient of the variable $q_{n-\ell}$, which is located in position $(i+1, \ell+1)$ of B'_j. From Lemma 17.1, the entry of B'_j at that position is $\pm\mu \cdot b^{(j)}_{i+1 \leftarrow \ell+1}$. Since $b^{(j)}_{i+1 \leftarrow \ell+1}$ is a determinant of a $j \times j$ sub-matrix of B_0 not involving its $(k+2)$th column, from Claim 21 we obtain $\left|b^{(j)}_{i+1 \leftarrow \ell+1}\right| \leq j^j (\frac{Q}{\mu})^j$. Similarly, η_j (i.e., the coefficient of q_{n-i}) is located in position $(i+1, i+1)$ of B'_j. Then, by Lemma 17.1, $\eta_j = \pm\mu \cdot b^{(j-1)}_{j,j}$, and $0 \leq \pm b^{(j-1)}_{j,j} \leq j^j (\frac{Q}{\mu})^j$.

Lastly, we consider the coefficient of the variable u, which is located in position $(i+1, k+2)$ of B'_j. From Lemma 17.2, the entry of B'_j at that position is $\pm b^{(j)}_{i+1 \leftarrow k+2}$. This $j \times j$ sub-determinant of B_0 involves the $(k+2)$th column. By Claim 21, $\left|b^{(j)}_{i+1 \leftarrow k+2}\right| \leq j^j (\frac{Q}{\mu})^{j-1} U$. □

Claim 23. *In every equality or inequality of the formula γ_j:*

- *The variables $\boldsymbol{q}_{[0,j-1]}$ do not appear (their coefficients are zero).*
- *Each coefficient of a variable in $\boldsymbol{q}_{[j,k]}$ is $\mu \cdot d$, for some $d \in \mathbb{Z}$ such that $|d| \leq (j+1)^{j+1} \left(\frac{Q}{\mu}\right)^{j+1}$.*
- *The coefficient of the variable u is bounded, in absolute value, by $(j+1)^{j+1} \left(\frac{Q}{\mu}\right)^j U$.*

Proof. The proof is similar to the one of Claim 22, but we now appeal to Lemma 16 instead of Lemma 17. By definition, the coefficients of the (in)equalities of γ_j are located in the last t rows of M_j, or alternatively of B'_j, by Lemma 20. Consider the ith row of B'_j, with $i \in [j+1..j+t]$. From Lemma 16.1, given $r \in [1..k+1]$, the entry of the matrix B'_j in position (i, r) is $\pm\mu \cdot b^{(j)}_{i,r}$; and moreover $b^{(j)}_{i,r} = 0$ whenever $r \leq j$. The first two statements of the claim then follow from the fact that the first $k+1$ columns of B'_j contain coefficients of the variables $q_n, \ldots, q_{n-k}$. In particular, for the second statement, note that $b^{(j)}_{i,r}$ is the determinant of a $(j+1) \times (j+1)$ sub-matrix of B_0 not involving its $(k+2)$th column. From Claim 21, $\left|b^{(j)}_{i,r}\right| \leq (j+1)^{j+1} (\frac{Q}{\mu})^{j+1}$. The coefficient of the variable u is located instead in positions $(i, k+2)$ of B'_j. From Lemma 16.2, the entry in this position is $\pm b^{(j)}_{i,k+2}$. This $(j+1) \times (j+1)$ sub-determinant of B_0 involves the $(k+2)$th column. From Claim 21, we have $\left|b^{(j)}_{i,k+2}\right| \leq (j+1)^{j+1} (\frac{Q}{\mu})^j U$. □

Next, we consider the divisibility constraints in γ_j. We recall that these of the form $d \mid \tau$ where all integers in the term τ belong to $[0..d-1]$. It thus suffices to give a bound on $mod(\psi_k)$ in order to bound all integers in these constraints.

Claim 24. *$mod(\gamma_j)$ divides $c \cdot mod(\gamma)$ for some positive integer $c \leq \frac{(j \cdot Q)^{j^2}}{\mu^{j(j-1)}}$.*

Proof. For a given $\ell \in [1..j]$, we show that $mod(\gamma_\ell) = \mu \cdot \left|b_{\ell,\ell}^{(\ell-1)}\right| \cdot mod(\gamma_{\ell-1})$. The bound then follows by recalling that $\left|b_{\ell,\ell}^{(\ell-1)}\right| \leq \ell^\ell \big(\frac{Q}{\mu}\big)^\ell$. Remark the divisibility constraints are only updated in line 3 of Algorithm 5. In this line, the substitution updates each divisibility constraint $d \mid \rho$ into a constraint of the form $(|a| \cdot d) \mid \rho'$ (where a is the coefficient of $q_{n-\ell+1}$ in the equality $a \cdot q_{n-\ell+1} = \tau$ returned by Algorithm 4). Line 3 also adds a divisibility constraint with divisor $|a|$. From Claim 8, a is the value of the entry in position (ℓ,ℓ) of the matrix $M_{\ell-1}$. From Lemma 20 and Lemma 16.1, $|a| = \mu \cdot \left|b_{\ell,\ell}^{(\ell-1)}\right|$. Therefore, $mod(\gamma_\ell) = \mu \cdot \left|b_{\ell,\ell}^{(\ell-1)}\right| \cdot mod(\gamma_{\ell-1})$, as required. □

Lastly, we bound all constants occurring in equalities and inequalities of γ_j, and in terms τ from the assignments $q_{n-i} \leftarrow \frac{\tau_{n-i}}{\eta_j}$ occurring in C_j, with $i \in [0..j-1]$. Observe that in γ and C, these constants are bounded by R.

Claim 25. *Each constant in terms from* $\text{terms}(\gamma_j)$ *and in terms* τ *from assignments* $q_{n-i} \leftarrow \frac{\tau}{\eta_j}$ *in* C_j *(with* $i \in [0..j-1]$*), is bounded, in absolute value, by* $(j+1)^{2(j+2)^2} \cdot \frac{Q^{2j(j+2)}}{\mu^{2j^2}} \cdot mod(\gamma) \cdot R$.

Proof. For simplicity, let us write:

- R_ℓ for the maximum, in absolute value, of all constants occurring in (in)equalities of γ_ℓ as well as in terms τ from assignments $q_{n-i} \leftarrow \frac{\tau}{\gamma_\ell}$ in C_ℓ, with $\ell \in [0..j]$.
- S_ℓ for the absolute value of the constant in the equality $a_\ell \cdot q_{n-\ell} = \tau_\ell$ returned by Algorithm 4 during the $(\ell+1)$th iteration of ElimVars, with $\ell \in [0..j-1]$.
- g for the positive integer $\mu \cdot (j+1)^{j+1}\big(\frac{Q}{\mu}\big)^{j+1}$. By Claims 8, 22 and 23, this is an upper bound to $\eta_0, \ldots, \eta_j$ and to the absolute values of all the coefficients of variables $\boldsymbol{q}_k$, in all formulae $\gamma_0, \ldots, \gamma_j$, all circuits $C_0, \ldots, C_j$, and all equalities $a_0 \cdot q_n = \tau_0$, … , $a_{j-1} \cdot q_{n-(j-1)} = \tau_{j-1}$.
- h for the positive integer $\frac{(j \cdot Q)^{j^2}}{\mu^{j(j-1)}} \cdot mod(\gamma)$. This is an upper bound to $mod(\gamma_0), \ldots, mod(\gamma_j)$.

We now bound R_ℓ and S_ℓ (first in terms of $R_{\ell-1}$ and $S_{\ell-1}$) by analyzing Algorithms 4 and 5.

bound on R_0: The circuit C_0 has no assignment on the variables $\boldsymbol{q}_{k-1}$, so $R_0 = 0 \leq R$.

bound on R_ℓ for $\ell \geq 1$. We show that $R_\ell \leq g \cdot (R_{\ell-1} + S_{\ell-1})$. Looking at Algorithm 5, we see that after the substitution in line 3 takes place, the constants in the equalities and inequalities in γ are bounded by $\frac{g}{\mu}(R_{\ell-1} + S_{\ell-1})$ (in particular, note that α from line 2 is bounded by $\frac{g}{\mu}$). Afterwards, lines 5 and 6 divide these constants by $\frac{\eta_{\ell-1}}{\mu}$ (line 6 takes the ceiling of this division). By definition of $\mathcal{I}_k^{\ell-1}$, μ divides $\eta_{\ell-1}$. Hence, the constants appearing in the (in)equalities of γ_ℓ are bounded by $\left\lceil \frac{g}{\eta_{\ell-1}}(R_{\ell-1} + S_{\ell-1}) \right\rceil \leq g \cdot (R_{\ell-1} + S_{\ell-1})$. A similar analysis applies to the constants in the terms τ from assignments $q_{n-i} \leftarrow \frac{\tau}{\gamma_\ell}$ in C_ℓ, since the corresponding terms in $C_{\ell-1}$ are updated in the same way as those in equalities of $\gamma_{\ell-1}$ (line 7).

bound on S_ℓ for $\ell \geq 0$. We show that $S_\ell \leq 2 \cdot R_\ell + g \cdot (R + 3 \cdot h)$. If in line 2 Algorithm 4 guesses a term from $\text{terms}(\gamma \wedge \gamma[q_{n-\ell} + p \,/\, q_{n-\ell}])$, then $S_\ell \leq R_\ell + g \cdot h$ and we are done. Otherwise, Algorithm 2 is invoked, which returns a term obtained by simultaneously applying two substitutions ν_1 and ν_2 to a term of the form $a \cdot u + \mu \cdot (q' - q'') + d$, with $a, d \in [-L..L]$ and q', q'' variables in $\boldsymbol{q}_k$. As already discussed during the proof of Claim 8, the substitutions ν_1 and ν_2 are of the form $[\frac{\tau}{\lambda} \,/\, \mu \cdot q]$ where, q is among q' and q'', $\lambda = \frac{\eta_\ell}{\mu}$, and the term τ is *(i)* $\eta_\ell \cdot q$, or *(ii)* $\eta_\ell \cdot q + \eta_\ell \cdot p$ with $p \coloneqq mod(q_{n-\ell}, \gamma_\ell)$, or *(iii)* such that $q \leftarrow \frac{\tau}{\eta_\ell}$ occurs in C_ℓ, or

(iv) of the form $\tau'[q_{n-\ell} + p \,/\, q_{n-\ell}]$ with $q \leftarrow \frac{\tau'}{\eta_\ell}$ occurring in C_ℓ. Therefore, the constant of τ is bounded by $R_\ell + g \cdot h$ (where $g \cdot h$ accounts for the constant in Case *(ii)* and for the increase that the substitution $[q_{n-\ell} + p \,/\, q_{n-\ell}]$ may cause). The constant of the term computed in line 8 of Algorithm 2 is thus bounded, in absolute value, by $2 \cdot (R_\ell + g \cdot h) + \frac{\eta_\ell}{\mu} \cdot |d|$. This constant is then shifted by at most $g \cdot h$ in line 3 of Algorithm 4. Recall that $|d| \leq L \leq R$ and, from Lemma 20, $\frac{\eta_\ell}{\mu} = \left| b_{\ell,\ell}^{(\ell-1)} \right| \leq g$. We conclude that $S_\ell \leq 2 \cdot R_\ell + g \cdot (R + 3 \cdot h)$.

By conjoining the above inequalities for R_ℓ and S_ℓ, we derive the following recurrence relation:

$$R_0 \leq R, \qquad R_\ell \leq 3 \cdot g \cdot R_{\ell-1} + g^2 \cdot (R + 3 \cdot h) \qquad \text{for } \ell \in [1..j].$$

A simple induction shows $R_j \leq (3 \cdot g)^j R + (g^2(R + 3 \cdot h)) \cdot \sum_{i=0}^{j-1} (3 \cdot g)^i$. For $j \geq 1$, we have:

$$\begin{aligned}
R_j &\leq (3 \cdot g)^j + j \cdot (3 \cdot g)^{j-1} g^2 (R + 3 \cdot h) \\
&\leq (j+1) \cdot 3^{j+1} g^{j+1} h \cdot R && \text{⟅we have } R + 3 \cdot h \leq R \cdot 3 \cdot h\text{⟆} \\
&\leq 3^{j+1} \mu^{j+1} (j+1)^{(j+1)^2 + j^2 + 1} \cdot \frac{Q^{(j+1)^2 + j^2}}{\mu^{(j+1)^2 + j(j-1)}} \cdot \mathit{mod}(\gamma) \cdot R && \text{⟅def. of } g \text{ and } h\text{⟆} \\
&\leq \frac{((j+1) \cdot Q)^{2(j+2)^2}}{\mu^{2j^2}} \cdot \mathit{mod}(\gamma) \cdot R. && \text{⟅using } j \geq 1\text{⟆}
\end{aligned}$$

Note that for $j = 0$ the last expression reduces to $Q^8 \mathit{mod}(\gamma) \cdot R$, which is an upper bound to R_0. □

Since we have let j range arbitrarily in $[0..k]$, Claims 22–25 establish that, throughout its execution, ElimVars only constructs objects whose sizes polynomial in the sizes of C and γ. Since k bounds the number of iterations of ElimVars along any non-deterministic branch, we conclude that it runs in non-deterministic polynomial time. This completes the proof of the lemma: Items 1–2 follow from the fact that Algorithm 5 does not update ψ nor any of the expressions in C featuring $x_{n-k}, \ldots, x_n$, whereas Items 3–5 follow directly from Claims 22–25. □

C.9 Proof of Lemma 25 from Section 6

Lemma 25. *Algorithm 6 runs in non-deterministic polynomial time. Consider its execution on an integer linear-exponential program $\varphi(x_1, \ldots, x_n)$ with $n \geq 1$. Let $(\varphi_k, \theta_k, C_k)$ the system, circuit, and ordering obtained at the end of kth iteration of the* ***while*** *loop of line 5, in any non-deterministic branch of the algorithm. Then, the following bounds hold (for every $\ell, s, a, c \geq 1$):*

Statement in page 62

$$\textit{if} \begin{cases} \#\mathit{lst}(\varphi, \theta) & \leq \ell \\ \#\varphi & \leq s \\ \|\varphi\|_{\mathfrak{L}} & \leq a \\ \|\varphi\|_1 & \leq c \\ \mathit{mod}(\varphi) & \mid 1 \end{cases} \quad \textit{then} \quad \begin{cases} \#\mathit{lst}(\varphi_k, \theta_k) & \leq \ell + 3 \cdot k^2 \\ \#\varphi_k & \leq s + 3 \cdot k^3 + 2 \cdot \ell \cdot k \\ \|\varphi_k\|_{\mathfrak{L}} & \leq 3^k a \\ \|\varphi_k\|_1 & \leq 3^{8(k+1)} c \\ \mathit{mod}(\varphi_k) & \leq 3^{2 \cdot k^8} a^{2 \cdot k^7} \\ \xi_{C_k} & \leq 3^{8(k+2)^8} c^{8(k+2)^7} \\ \mu_{C_k} & \leq 3^{k^3} a^{k^2}. \end{cases}$$

Proof. The proof is by induction on k.

base case: $k = 0$. In this case, φ_0 is equal to φ, and C_0 is the empty 0-PreLEAC (hence, by definition $\mu_{C_0} = 1$ and $\xi_{C_0} = 0$). All bounds in the statement trivially follows.

induction hypothesis: For $k \geq 0$, the bounds in the statement hold for the kth loop iteration.

induction step: Given $k \geq 0$, consider a triple $(\varphi_k, \theta_k, C_k)$ obtained at the end of the kth iteration of the loop, and $(\varphi_{k+1}, \theta_{k+1}, C_{k+1})$ be obtained by applying the body of the loop to $(\varphi_k, \theta_k, C_k)$. We bound the parameters of φ_{k+1} and C_{k+1}. For brevity, we write ξ_k and μ_k for ξ_{C_k} and μ_{C_k}, respectively (and use similar notation for $\xi_{C_{k+1}}$ and $\mu_{C_{k+1}}$).

least significant terms of φ_{k+1}:

$$\begin{aligned} \#\mathit{lst}(\varphi_{k+1}, \theta_{k+1}) &\leq \max(\#\mathit{lst}(\varphi_k, \theta_k), 1) + 2 \cdot k + 3 && \text{⟅by Lemma 24⟆} \\ &\leq (\ell + 3 \cdot k^2) + 2 \cdot k + 3 && \text{⟅by I.H.⟆} \\ &\leq \ell + 3 \cdot (k+1)^2 \end{aligned}$$

In the following cases, for simplicity we assume that all parameters of φ_k and C_k are greater than or equal to 1. This assumption is made solely to avoid repeatedly writing expressions involving $\max(\cdot, 1)$, as we did earlier for $\#\mathit{lst}(\varphi_k, \theta_k)$.

number of constraints in φ_{k+1}:

$$\begin{aligned} \#\varphi_{k+1} &\leq \#\varphi_k + 6 \cdot k + 2 \cdot \#\mathit{lst}(\varphi_k, \theta_k) + 3 && \text{⟅by Lemma 24⟆} \\ &\leq (s + 3 \cdot k^3 + 2 \cdot \ell \cdot k) + 6 \cdot k + 2 \cdot (\ell + 3 \cdot k^2) + 3 && \text{⟅by I.H.⟆} \\ &\leq s + 3 \cdot (k+1)^3 + 2 \cdot \ell \cdot (k+1). \end{aligned}$$

linear norm of φ_{k+1}:

$$\begin{aligned} \|\varphi_{k+1}\|_{\mathfrak{L}} &\leq 3 \cdot \|\varphi_k\|_{\mathfrak{L}} && \text{⟅by Lemma 24⟆} \\ &\leq 3^{k+1} a. && \text{⟅by I.H.⟆} \end{aligned}$$

denominator μ_{k+1}:

$$\begin{aligned} \mu_{k+1} &\leq \mu_k (3 \cdot k \cdot \|\varphi_k\|_{\mathfrak{L}})^k && \text{⟅by Lemma 24⟆} \\ &\leq 3^{k^3} a^{k^2} \big(3 \cdot k \cdot (3^k a)\big)^k && \text{⟅by I.H.⟆} \\ &\leq 3^{(k+1)^3} a^{(k+1)^2}. \end{aligned}$$

modulus of φ_{k+1}: Below, $(\varphi_0, C_0), \ldots, (\varphi_{k-1}, C_{k-1})$ denote the formulae and PreLEACs constructed by the algorithm during the first $k-1$ iterations of the **while** loop; (φ_k, C_k) are obtained from (φ_{k-1}, C_{k-1}) by performing a further iteration. In particular, φ_0 is the linear-exponential program given as input to OptILEP, and C_0 is the empty 0-PreLEAC. By Lemma 24, for every $i \in [0..k]$, there is $\alpha_{i+1} \in [1..(3 \cdot i \cdot \mu_i \cdot \|\varphi_i\|_{\mathfrak{L}})^{i^2}]$ such that $\mathit{mod}(\varphi_{i+1})$ is a divisor of $\operatorname{lcm}(\mathit{mod}(\varphi_i), \phi(\alpha_{i+1} \cdot \mathit{mod}(\varphi_i)))$. Let us define $\alpha^* \coloneqq \operatorname{lcm}(\alpha_1, \alpha_2, \ldots, \alpha_{k+1})$, and consider the integers $c_0, \ldots, c_{k+1}$ given by

$$\begin{cases} c_0 \coloneqq 1 \\ c_{i+1} \coloneqq \operatorname{lcm}(c_i, \phi(\alpha^* \cdot c_i)) \quad \text{for } i \in [0..k] \end{cases}$$

Claim 26. *For every $j \in [0..k+1]$, $\mathit{mod}(\varphi_j)$ divides c_j.*

Proof. The proof is by induction on j.

base case: $j = 0$**.** We have $\mathit{mod}(\varphi_0) = 1 = c_0$.

induction step: Assume that the claim holds for $j \in [0..k]$. Then,

$$\begin{aligned}
\mathit{mod}(\varphi_{j+1}) &\coloneqq \operatorname{lcm}(\mathit{mod}(\varphi_j), \phi(\alpha_{j+1} \cdot \mathit{mod}(\varphi_j))) \\
&\mid \operatorname{lcm}(c_j, \phi(\alpha_{j+1} \cdot \mathit{mod}(\varphi_j))) && \text{⟅by I.H., } \mathit{mod}(\varphi_i) \mid c_i\text{⟆} \\
&\mid \operatorname{lcm}(c_j, \phi(\alpha^* \cdot c_j)) && \text{⟅} q \mid r \text{ implies } \phi(q) \mid \phi(r)\text{⟆} \\
&= c_{j+1}. && \square
\end{aligned}$$

Given Claim 26, in order to bound $\mathit{mod}(\varphi_{k+1})$ it suffices to bound c_{k+1}. The next lemma from [CMS24] will help us analyze this integer.

Lemma 26 [CMS24, Lemma 7]**.** *Let $\alpha \geq 1$ be in $\mathbb{N}$. Let $b_0, b_1, \ldots$ be the integer sequence given by the recurrence $b_0 := 1$ and $b_{i+1} := \operatorname{lcm}(b_i, \phi(\alpha \cdot b_i))$. For every $i \in \mathbb{N}$, $b_i \leq \alpha^{2 \cdot i^2}$.*

First, observe that

$$\begin{aligned}
\alpha^* &\leq \textstyle\prod_{i=0}^{k} (3 \cdot i \cdot \mu_i \cdot \|\varphi_i\|_{\mathfrak{L}})^{i^2} \\
&\leq (3 \cdot k \cdot (3^{k^3} a^{k^2}) \cdot (3^k a))^{k^2(k+1)} && \text{⟅by I.H.⟆} \\
&\leq 3^{(k+1)^6} a^{(k+1)^5}.
\end{aligned}$$

Then, c_{k+1} is bounded as follows:

$$\begin{aligned}
c_{k+1} &\leq (\alpha^*)^{2(k+1)^2} && \text{⟅by Lemma 26⟆} \\
&\leq 3^{2(k+1)^8} a^{2(k+1)^7}.
\end{aligned}$$

Note that in Lemma 24 the 1-norm of φ' and the parameter $\xi_{C'}$ are bounded in terms of a relatively complex quantity denoted as β. To simplify the upcoming calculations, we first derive a more manageable upper bound for β, with respect to φ_{k+1} and C_{k+1}. We start by simplifying the subexpression $\log(\xi_k + \mu_k)$:

$$\begin{aligned}
\log(\xi_k + \mu_k) &\leq \log(3^{8(k+2)^8} c^{8(k+2)^7} + 3^{k^3} a^{k^2}) && \text{⟅by I.H.⟆} \\
&\leq 1 + \log\left(3^{8(k+2)^8} c^{8(k+2)^7}\right) && \text{⟅as } a \leq c\text{⟆} \\
&\leq 17 \cdot (k+2)^8 c.
\end{aligned}$$

The quantity β can then be simplified as follows:

$$\begin{aligned}
\beta &\coloneqq \mathit{mod}(\varphi_k)\left(2^7(k+1) \cdot \mu_k \cdot \max(\|\varphi_k\|_1, \log(\xi_k + \mu_k))\right)^{3(k+2)^2} \\
&\leq 3^{2 \cdot k^8} a^{2 \cdot k^7} \left(2^7(k+1) \cdot 3^{k^3} a^{k^2} \max(3^{8(k+1)} c, \log(\xi_k + \mu_k))\right)^{3(k+2)^2} && \text{⟅by I.H⟆} \\
&\leq 3^{2 \cdot k^8} a^{2 \cdot k^7} \left(2^7(k+1) \cdot 3^{k^3} a^{k^2} 3^{8(k+1)} c\right)^{3(k+2)^2} && \text{⟅as } 3^{8(k+1)} \geq 17 \cdot (k+2)^8\text{⟆} \\
&\leq 3^{2 \cdot k^8 + 3 \cdot (k+2)^2 (k^3 + 9k + 13)} c^{2 \cdot k^7 + 3(k+2)^2 (k^2+1)} && \text{⟅as } a \leq c\text{⟆} \\
&\leq 3^{2(k+2)^8} c^{2(k+2)^7}.
\end{aligned}$$

1-norm of φ_{k+1}:

$$\begin{aligned}
\|\varphi_{k+1}\|_1 &\leq 12 + 4 \cdot \max\left(\|\varphi_k\|_1, \log \beta\right) && \text{⟅by Lemma 24⟆} \\
&\leq 12 + 4 \cdot \max(3^{8(k+1)}c, \log \beta) && \text{⟅by I.H.⟆} \\
&\leq 12 + 4 \cdot \max(3^{8(k+1)}c, 4 \cdot (k+2)^8 c) && \text{⟅from bound on } \beta\text{⟆} \\
&\leq 12 + 4 \cdot 3^{8(k+1)}c \leq 3^{8\cdot(k+2)}c.
\end{aligned}$$

parameter ξ_{k+1}:

$$\begin{aligned}
\xi_{k+1} &\leq \xi_k \cdot (3 \cdot k \cdot \|\varphi_k\|_{\mathfrak{L}})^k + 2^6(k+1) \cdot \beta^4 && \text{⟅by Lemma 24⟆} \\
&\leq (3^{8(k+2)^8} c^{8(k+2)^7}) \cdot (3 \cdot k \cdot (3^k a))^k + 2^6(k+1) \cdot (3^{2(k+2)^8} c^{2(k+2)^7})^4 && \text{⟅by I.H.⟆} \\
&\leq 3^{8(k+2)^8} c^{8(k+2)^7}((3^{k+1}k \cdot c)^k + 2^6(k+1)) && \text{⟅}a \leq c\text{⟆} \\
&\leq 3^{8(k+2)^8} c^{8(k+2)^7}(c^k 3^{2k^2+k+5}) \leq 3^{8(k+3)^8} c^{8(k+3)^7}.
\end{aligned}$$

Given the bounds we have just established, it is simple to see that OptILEP runs in non-deterministic polynomial time. Indeed, these bounds ensure that, each time the execution reaches line 5, both the formula φ and circuit C manipulated by the algorithm are of size polynomial in the input. The **while** loop of line 5 iterates n times, and by Lemma 24 each iteration runs in non-deterministic polynomial time. It follows that OptILEP runs in non-deterministic polynomial time. □

D Proofs of statements from Part II

Lemma 27. *Consider an ILESLP $\sigma \coloneqq (x_0 \leftarrow \rho_0, \ldots, x_n \leftarrow \rho_n)$ and let $i \in [0..n]$. One can compute, in time polynomial in the size of σ, an expression E_i of the form $\sum_{j=0}^{i-1} a_{i,j} \cdot 2^{x_j}$ such that $[\![\sigma]\!](E_i) = d(\sigma) \cdot [\![\sigma]\!](x_i)$. For every $j \in [0..i-1]$, the coefficient $a_{i,j}$ is* (i) *an integer whose absolute value is bounded by $2^i \cdot e(\sigma) \cdot d(\sigma)$, and* (ii) *non-zero only if $[\![\sigma]\!](x_j) \geq 0$.*

Statement in page 64

Proof. Given $i \in [0..n]$, let σ_i denote the ILESLP $(x_0 \leftarrow \rho_0, \ldots, x_i \leftarrow \rho_i)$ obtained by truncating σ after $i+1$ assignments. We remark that $d(\sigma_i)$ divides $d(\sigma_j)$ for every $i \leq j$.

We show by induction on i how to compute a vector of rational numbers $\boldsymbol{b}_i = (b_{i,0}, \ldots, b_{i,i-1}) \in \mathbb{Q}^i$ satisfying $[\![\sigma]\!](x_i) = \sum_{j=0}^{i-1} b_{i,j} \cdot 2^{[\![\sigma]\!](x_j)}$. Moreover, each $b_{i,j}$ is of the form $\frac{m}{d(\sigma_i)}$ for some $m \in \mathbb{Z}$ satisfying $|m| \leq 2^i \cdot e(\sigma_i) \cdot d(\sigma_i)$, and $m \neq 0$ only if $[\![\sigma]\!](x_j) \geq 0$. With this result at hand, the expression E_i in the statement of the lemma is computed by multiplying all these rational numbers by $d(\sigma)$ to make them integers. In particular, if $b_{i,j} = \frac{m}{d(\sigma_i)}$, then in E_i the coefficient of 2^{x_j} is $a_{i,j} \coloneqq m \cdot \frac{d(\sigma)}{d(\sigma_i)}$. We then conclude that $|a_{i,j}| \leq 2^i \cdot e(\sigma_i) \cdot d(\sigma_i) \cdot \frac{d(\sigma)}{d(\sigma_i)} \leq 2^i \cdot e(\sigma) \cdot d(\sigma)$. Note that the bit size of each $a_{i,j}$ is thus polynomial in the size of σ. With this in mind, the fact that the whole computation can be performed in polynomial time will be immediate from the inductive proof.

base case: $i = 0$. In this case we simply have $\rho_0 = 0$ and we take $\boldsymbol{b}_0$ to be empty vector.

induction hypothesis. We have computed the vector $\boldsymbol{b}_j \in \mathbb{Q}^j$, for every $j \in [0..i-1]$. Given $k \in [0..j-1]$, the kth entry of $\boldsymbol{b}_j$ is a rational of the form $\frac{m}{d(\sigma_j)}$, for some $m \in \mathbb{Z}$ satisfying $|m| \leq 2^j \cdot e(\sigma_j) \cdot d(\sigma_j)$, and $m \neq 0$ only if $[\![\sigma]\!](x_k) \geq 0$.

induction step: $i \geq 1$. We reason by cases, depending on ρ_i.

case: $\rho_i = 0$. We define $\boldsymbol{b}_i$ to be the zero vector of length i (encoded as the rational $\frac{0}{d(\sigma_i)}$).

case: $\rho_i = 2^{x_j}$**.** We define $\boldsymbol{b}_i$ by setting $b_{i,j} = \frac{d(\sigma_i)}{d(\sigma_i)}$, and $b_{i,\ell} = 0$ for all $\ell \neq j$. Since σ is an ILESLP, we must have $[\![\sigma]\!](x_i) \in \mathbb{Z}$. Therefore, $[\![\sigma]\!](x_j) \geq 0$; which allows us to set a non-zero value to $b_{i,j}$.

case: $\rho_i = x_j + x_k$**.** Following the induction hypothesis, consider the already computed vectors $\boldsymbol{b}_j = (b_{j,0}, \ldots, b_{j,j-1})$ and $\boldsymbol{b}_k = (b_{k,0}, \ldots, b_{k,k-1})$. Let the vectors $(b_{j,0}, \ldots, b_{j,i-1})$ and $(b_{k,0}, \ldots, b_{k,i-1})$ be obtained from $\boldsymbol{b}_j$ and $\boldsymbol{b}_k$ by appending a suitable amount of 0s (encoded as $\frac{0}{d(\sigma_j)}$ and $\frac{0}{d(\sigma_k)}$, respectively). The vector $\boldsymbol{b}_i$ is defined as follows: for every $\ell \in [0..i-1]$, if $b_{j,\ell} = \frac{m}{d(\sigma_j)}$ and $b_{k,\ell} = \frac{r}{d(\sigma_k)}$, then we define $b_{i,\ell} \coloneqq \frac{m \cdot \frac{d(\sigma_i)}{d(\sigma_j)} + r \cdot \frac{d(\sigma_i)}{d(\sigma_k)}}{d(\sigma_i)}$. Clearly, $b_{i,\ell} = b_{j,\ell} + b_{k,\ell}$, and the numerator of $b_{i,\ell}$ is an integer (because $d(\sigma_i)$ is divided by both $d(\sigma_j)$ and $d(\sigma_k)$). For the numerator, we have:

$$
\begin{aligned}
&\left| m \cdot \frac{d(\sigma_i)}{d(\sigma_j)} + r \cdot \frac{d(\sigma_i)}{d(\sigma_k)} \right| \\
\leq &\left| m \cdot \frac{d(\sigma_i)}{d(\sigma_j)} \right| + \left| r \cdot \frac{d(\sigma_i)}{d(\sigma_k)} \right| \\
\leq\ & 2^j \cdot e(\sigma_j) \cdot d(\sigma_j) \cdot \frac{d(\sigma_i)}{d(\sigma_j)} + 2^k \cdot e(\sigma_k) \cdot d(\sigma_k) \cdot \frac{d(\sigma_i)}{d(\sigma_k)} \qquad \text{⟅by induction hypothesis⟆} \\
\leq\ & 2 \cdot 2^{i-1} \cdot e(\sigma_{i-1}) \cdot d(\sigma_i) \\
\leq\ & 2^i \cdot e(\sigma_i) \cdot d(\sigma_i).
\end{aligned}
$$

Lastly, observe that for a variable x among $x_0, \ldots, x_{i-1}$ satisfying $[\![\sigma]\!](x) < 0$, (the numerators of) both corresponding rationals in $\boldsymbol{b}_j$ and $\boldsymbol{b}_k$ are zero (by induction hypothesis). Therefore, the same holds for $\boldsymbol{b}_i$.

case: $\rho_i = \frac{m}{g} \cdot x_j$**.** Similarly to the previous case, consider the vector $(b_{j,0}, \ldots, b_{j,i-1})$ obtained from $\boldsymbol{b}_j$ by appending 0s. The vector $\boldsymbol{b}_i$ is defined as follows: for every $\ell \in [0..i-1]$, if $b_{j,\ell} = \frac{r}{d(\sigma_j)}$ then we define $b_{i,\ell} \coloneqq \frac{m \cdot r \cdot \frac{d(\sigma_{i-1})}{d(\sigma_j)}}{d(\sigma_i)}$. Note that, by definition, $d(\sigma_i) = g \cdot d(\sigma_{i-1})$ and $b_{j,\ell} = \frac{r \cdot \frac{d(\sigma_{i-1})}{d(\sigma_j)}}{d(\sigma_{i-1})}$; and thus $b_{i,\ell} = \frac{m}{g} \cdot b_{i,\ell}$. The numerator is bounded as follows:

$$
\begin{aligned}
\left| m \cdot r \cdot \frac{d(\sigma_{i-1})}{d(\sigma_j)} \right| &\leq 2^j \cdot e(\sigma_j) \cdot d(\sigma_j) \cdot |m| \cdot \frac{d(\sigma_{i-1})}{d(\sigma_j)} && \text{⟅by induction hypothesis⟆} \\
&\leq 2^j \cdot e(\sigma_i) \cdot d(\sigma_{i-1}) && \text{⟅because } e(\sigma_j) \cdot |m| \leq e(\sigma_i)\text{⟆} \\
&\leq 2^i \cdot e(\sigma_i) \cdot d(\sigma_i).
\end{aligned}
$$

Lastly, note that if a variable x among $x_0, \ldots, x_{i-1}$ satisfies $[\![\sigma]\!](x) < 0$, then the corresponding rational in $\boldsymbol{b}_j$ is zero (by induction hypothesis), and so the same holds for $\boldsymbol{b}_j$. □

References

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. Math.*, 2004. doi: 10.4007/annals.2004.160.781.

[AMC+99] Giorgio Ausiello, Alberto Marchetti-Spaccamela, Pierluigi Crescenzi, Giorgio Gambosi, Marco Protasi, and Viggo Kann. *Complexity and approximation: combinatorial optimization problems and their approximability properties*. 1999. doi: 10.1007/978-3-642-58412-1.

[Bar68] Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comput.*, 1968. doi: 10.1090/S0025-5718-1968-0226829-0.

[BBS86] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 1986. doi: 10.1137/0215025.

[BDJ24] Markus Bläser, Julian Dörfler, and Gorav Jindal. PosSLP and sum of squares. 2024. doi: 10.4230/LIPIcs.FSTTCS.2024.13.

[BGW17] Tristram Bogart, John Goodrick, and Kevin Woods. Parametric Presburger arithmetic: logic, combinatorics, and quasi-polynomial behavior. *Discrete Analysis*, 2017. doi: 10.19086/da.1254.

[BH24] Eion Blanchard and Philipp Hieronymi. Decidability bounds for Presburger arithmetic extended by sine. *Ann. Pure Appl. Log.*, 2024. doi: 10.1016/j.apal.2024.103487.

[BJ24] Peter Bürgisser and Gorav Jindal. On the hardness of PosSLP. In *SODA*, 2024. doi: 10.1137/1.9781611977912.75.

[BT76] Itshak Borosh and Leon B. Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Am. Math. Soc.*, 1976. doi: 10.2307/2041711.

[BT18] Clark W. Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*. 2018. doi: 10.1007/978-3-319-10575-8_11.

[BW08] David M. Bressoud and S. Wagon. *A course in computational number theory*. Wiley & Sons, 2008. isbn: 978-0-470-41215-2.

[CJSS21] Peter Chvojka, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Versatile and sustainable timed-release encryption and sequential time-lock puzzles (extended abstract). In *ESORICS*, 2021. doi: 10.1007/978-3-030-88428-4_4.

[CMS24] Dmitry Chistikov, Alessio Mansutti, and Mikhail R. Starchak. Integer linear-exponential programming in NP by quantifier elimination. In *ICALP*, 2024. doi: 10.4230/LIPICS.ICALP.2024.132. See arXiv:2407.07083 for a full version.

[CP89] Pierluigi Crescenzi and Alessandro Panconesi. Completeness in approximation classes. In *FCT*, 1989. doi: 10.1007/3-540-51498-8_11.

[DHM24] Andrei Draghici, Christoph Haase, and Florin Manea. Semënov arithmetic, affine VASS, and string constraints. In *STACS*, 2024. doi: 10.4230/LIPICS.STACS.2024.29.

[DHMP24] Rémy Défossez, Christoph Haase, Alessio Mansutti, and Guillermo A. Pérez. Integer programming with GCD constraints. In *SODA*, 2024. doi: 10.1137/1.9781611977912.128.

[EGKO19] Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. Solving integer quadratic programming via explicit and structural restrictions. In *AAAI*, 2019. doi: 10.1609/aaai.v33i01.33011477.

[FG24] Florian Frohn and Jürgen Giesl. Satisfiability modulo exponential integer arithmetic. In *IJCAR*, 2024. doi: 10.1007/978-3-031-63498-7_21.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective.* 2008. doi: 10.1017/CBO9780511804106.

[HK71] Kenneth Hoffman and Ray Kunze. *Linear Algebra.* 1971. isbn: 978-0135367971.

[JLN+10] Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art.* 2010. doi: 10.1007/978-3-540-68279-0.

[KBT14] Tim King, Clark W. Barrett, and Cesare Tinelli. Leveraging linear and mixed integer programming for SMT. In *FMCAD*, 2014. doi: 10.1109/FMCAD.2014.6987606.

[KLN+25] Toghrul Karimov, Florian Luca, Joris Nieuwveld, Joël Ouaknine, and James Worrell. On the decidability of Presburger arithmetic expanded with powers. In *SODA*, 2025. doi: 10.1137/1.9781611978322.89.

[Kre88] Mark W. Krentel. The complexity of optimization problems. *J. Comput. Syst. Sci.*, 1988. doi: 10.1016/0022-0000(88)90039-6.

[Len83] Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 1983. doi: 10.1287/moor.8.4.538.

[Lib03] Leonid Libkin. Variable independence for first-order definable constraints. *ACM Trans. Comput. Log.*, 4(4):431–451, 2003. doi: 10.1145/937555.937557.

[Lok15] Daniel Lokshtanov. Parameterized integer quadratic programming: Variables and coefficients. *CoRR*, 2015. url: arxiv.org/abs/1511.00310.

[MUW11] Alexei Myasnikov, Alexander Ushakov, and Dong Wook Won. The word problem in the Baumslag group with a non-elementary Dehn function is polynomial time decidable. *J. Algebra*, 2011. doi: 10.1016/j.jalgebra.2011.07.024.

[MUW12] Alexei G. Myasnikov, Alexander Ushakov, and Dong Wook Won. Power circuits, exponential algebra, and time complexity. *Int. J. Algebra Comput.*, 2012. doi: 10.1142/S0218196712500476.

[Nel08] David Nelson. *The Penguin Dictionary of Mathematics: Fourth edition.* 2008. isbn: 978-0-141-92087-0.

[PDM17] Alberto Del Pia, Santanu S. Dey, and Marco Molinaro. Mixed-integer quadratic programming is in NP. *Math. Program.*, 2017. doi: 10.1007/s10107-016-1036-0.

[Pre29] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congrès des Mathématiciens des Pays Slaves*, pages 92–101. 1929.

[RSW96] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, 1996.

[Sem84] Aleksei L. Semenov. Logical theories of one-place functions on the set of natural numbers. *Math. USSR Izv.*, 1984. doi: 10.1070/im1984v022n03abeh001456.

[She18] Bobby Shen. Parametrizing an integer linear program by an integer. *SIAM J. Discret. Math.*, 2018. doi: 10.1137/16M1102458.

[Smo91] Craig Smoryński. *Logical Number Theory I: An Introduction*. 1991. doi: 10.1007/978-3-642-75462-3.

[Sta25] Mikhail R. Starchak. Quantifier elimination for regular integer linear-exponential programming. In *LICS*, 2025. To appear.

[vzGS78] Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proc. Am. Math. Soc.*, 1978. doi: 10.1090/S0002-9939-1978-0500555-0.

[WCW+23] Hao Wu, Yu-Fang Chen, Zhilin Wu, Bican Xia, and Naijun Zhan. A decision procedure for string constraints with string-integer conversion and flat regular constraints. *Acta Inform.*, 2023. doi: 10.1007/s00236-023-00446-4.

[Wei90] Volker Weispfenning. The complexity of almost linear Diophantine problems. *J. Symb. Comput.*, 1990. doi: 10.1016/S0747-7171(08)80051-X.