Integer Linear-Exponential Programming in NP by Quantifier Elimination

Dmitry Chistikov[1]
Centre for Discrete Mathematics and its Applications (DIMAP) &
Department of Computer Science, University of Warwick, Coventry, UK

Alessio Mansutti
IMDEA Software Institute, Madrid, Spain

Mikhail R. Starchak
St. Petersburg State University, Russia

Abstract

This paper provides an NP procedure that decides whether a *linear-exponential system* of constraints has an integer solution. Linear-exponential systems extend standard integer linear programs with exponential terms 2^x and remainder terms $(x \bmod 2^y)$. Our result implies that the existential theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, V_2(\cdot,\cdot), \leq)$ has an NP-complete satisfiability problem, thus improving upon a recent ExpSpace upper bound. This theory extends the existential fragment of Presburger arithmetic with the exponentiation function $x \mapsto 2^x$ and the binary predicate $V_2(x, y)$ that is true whenever $y \geq 1$ is the largest power of 2 dividing x.

Our procedure for solving linear-exponential systems uses the method of quantifier elimination. As a by-product, we modify the classical Gaussian variable elimination into a non-deterministic polynomial-time procedure for integer linear programming (or: existential Presburger arithmetic).

2012 ACM Subject Classification Computing methodologies → Symbolic and algebraic algorithms; Theory of computation → Logic

Keywords and phrases decision procedures, integer programming, quantifier elimination

Funding *Dmitry Chistikov*: Supported in part by the Engineering and Physical Sciences Research Council [EP/X03027X/1].
Alessio Mansutti: funded by the Madrid Regional Government (César Nombela grant 2023-T1/COM-29001), and by MCIN/AEI/10.13039/501100011033/FEDER, EU (grant PID2022-138072OB-I00).
Mikhail R. Starchak: Supported by the Russian Science Foundation, project 23-71-01041.

1 Introduction

Integer (linear) programming is the problem of deciding whether a system of linear inequalities has a solution over the integers ($\mathbb{Z}$). It is a textbook fact that this problem is NP-complete; however, proof of membership in NP is not trivial. It is established [3, 27] by showing that, if a given system has a solution over $\mathbb{Z}$, then it also has a *small* solution. The latter means that the bit size of all components can be bounded from above by a polynomial in the bit size of the system. Integer programming is an important language that can encode many combinatorial problems and constraints from multiple application domains; see, e.g., [20, 32].

In this paper we consider more general systems of constraints, which may contain not only linear inequalities (as in integer programming) but also constraints of the form $y = 2^x$ (exponentiation base 2) and $z = (x \bmod 2^y)$ (remainder modulo powers of 2). Equivalently,

[1] During the work on this paper, DC was a visitor to the Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern and Saarbrücken, Germany, a visiting fellow at St Catherine's College and a visitor to the Department of Computer Science at the University of Oxford, United Kingdom.

and embedding both new operations into a uniform syntax, we look at a conjunction of inequalities of the form

$$\sum\nolimits_{i=1}^{n} \Big(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum\nolimits_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j})\Big) + d \leq 0\,, \tag{1}$$

referred to as an *(integer) linear-exponential system.* In fact, the linear-exponential systems that we consider can also feature equalities $=$ and strict inequalities $<$.[2]

Observe that a linear-exponential system of the form $x_1 = 1 \wedge \bigwedge_{i=1}^{n}(x_{i+1} = 2^{x_i})$ states that x_{n+1} is the tower of 2s of height n. This number is huge, and makes proving an analogue of the small solution property described above a hopeless task in our setting. This obstacle was recently shown avoidable [11], however, and an exponential-space procedure for linear-exponential programs was found, relying on automata-theoretic methods. Our main result is that, in fact, the problem belongs to NP.

► **Theorem 1.** *Deciding whether a linear-exponential system over* $\mathbb{Z}$ *has a solution is in* NP*.*

We highlight that the choice of the base 2 for the exponentials is for the convenience of exposition: our result holds for any positive integer base given in binary as part of the input.

As an example showcasing the power of integer linear-exponential systems, consider computation of discrete logarithm base 2: given non-negative integers $m, r \in \mathbb{N}$, producing an $x \in \mathbb{N}$ such that $2^x - r$ is divisible by m. As sketched in [14], this problem is reducible to checking feasibility (existence of solutions) of at most $\log m$ linear-exponential systems in two variables, by a binary search for a suitable exponent x. Hence, improving Theorem 1 from NP to PTime for the case of linear-exponential systems with a *fixed* number of variables would require a major breakthrough in number theory. In contrast, under this restriction, feasibility of standard integer linear programs can be determined in PTime [19].

For the authors of this paper, the main motivation for looking at linear-exponential systems stems from logic. Consider the first-order theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, V_2(\cdot,\cdot), \leq)$, which we refer to as the *Büchi–Semenov arithmetic.* In this structure, the signature $(0, 1, +, \leq)$ of linear arithmetic is extended with the function symbol $2^{(\cdot)}$, interpreted as the function $x \mapsto 2^x$, and the binary predicate symbol V_2, interpreted as $\{(x, y) \in \mathbb{N} \times \mathbb{N} : y$ is the largest power of 2 that divides $x\}$. Importantly, the predicate V_2 can be replaced in this definition with the function $x \bmod 2^y$, because the two are mutually expressible:

$$\begin{aligned} V_2(x, y) &\iff \exists v \left(2 \cdot y = 2^v \wedge 2 \cdot (x \bmod 2^v) = 2^v\right), \\ (x \bmod 2^y) = z &\iff z \leq 2^y - 1 \wedge \left(x = z \vee \exists u \left(V_2(x - z, 2^u) \wedge 2^y \leq 2^u\right)\right). \end{aligned}$$

Above, the subtraction symbol can be expressed in the theory in the obvious way (perhaps with the help of an auxiliary existential quantifier for expressing $x - z$).

Büchi–Semenov arithmetic subsumes logical theories known as Büchi arithmetic and Semenov arithmetic; see Section 2. As a consequence of Theorem 1, we show:

► **Theorem 2.** *The satisfiability problem of existential Büchi–Semenov arithmetic is in* NP*.*

Theorems 1 and 2 improve upon several results in the literature. The most recent such result is the exponential-space procedure [11] already mentioned above. In 2023, two

[2] While equalities are considered for convenience only (they can be encoded with a pair of inequalities $\leq$), the addition of $<$ is of interest. Indeed, differently from standard integer programming, one cannot define $<$ in terms of $\leq$, since 2^y is not an integer for $y < 0$. Observe that $(x \bmod 2^y) = 0$ when $y < 0$, because over the reals $(a \bmod m) = a - m \left\lfloor \frac{a}{m} \right\rfloor$, where $\lfloor . \rfloor$ is the floor function.

elementary decision procedures were developed concurrently and independently for integer linear programs with exponentiation constraints ($y = 2^x$), or equivalently for the existential fragment of Semenov arithmetic: they run in non-deterministic exponential time [2] and in triply exponential time [42], respectively. Finally, another result subsumed by Theorem 2 is the membership in NP for the existential fragment of Büchi arithmetic [13].

Theorem 1 is established by designing a non-deterministic polynomial-time decision procedure, which, unlike those in papers [11, 13] but similarly to [2, 42], avoids automata-theoretic methods and instead relies on *quantifier elimination.* This is a powerful method (see, e.g., [9] as well as Section 2) that can be seen as a bridge between logic and integer programming. Presburger [30] used it to show decidability of linear integer arithmetic (and Tarski for real arithmetic with addition and multiplication). For systems of linear equations, quantifier elimination is essentially Gaussian elimination. As a little stepping stone, which was in fact one of the springboards for our paper, we extend the PTime integer Gaussian elimination procedure by Bareiss [1, 40] into an NP procedure for solving systems of inequalities over $\mathbb{Z}$ (thus re-proving membership of integer linear programming in NP).

A look ahead. The following Section 2 recalls some relevant related work on logical theories of arithmetic. At the end of the paper (Section 9) this material is complemented by a discussion of future research directions, along with several more key references.

The NP procedure for integer programming is given as Algorithm 1 in Section 4. In this extended abstract, we do not provide a proof of correctness or analysis of the running time, but instead compare the algorithm with the classic Gauss–Jordan variable elimination and with Bareiss' method for systems of equations (that is, equalities). Necessary definitions and background information are provided in the Preliminaries (Section 3).

Our core result is an NP procedure for solving linear-exponential systems over $\mathbb{N}$. Its pseudocode is split into Algorithms 2–4. These are presented in the same imperative style with non-deterministic branching as Algorithm 1, and in fact Algorithm 3 relies on Algorithm 1. Section 5 provides a high-level overview of all three algorithms together. To this end, we introduce several new auxiliary concepts: quotient systems and quotient terms, delayed substitution, and primitive linear-exponential systems. After this, technical details of Algorithms 2–4 are given. Section 6 sketches key ideas behind the correctness argument, and the text within this section is thus to be read alongside the pseudocode of Algorithms. An overview of the analysis of the worst-case running time is presented in Section 7. The basic definitions are again those from Preliminaries, and of particular relevance are the subtleties of the action of term substitutions.

Building on the core procedure, in Section 8 we show how to solve in NP linear-exponential systems not only over $\mathbb{N}$ but also over $\mathbb{Z}$ (Theorem 1) and how to decide Büchi–Semenov arithmetic in NP (Theorem 2). The modifications to this procedure that enable proving both results for a different integer base $b > 2$ for the exponentials are given in Appendix A.

2 Arithmetic theories of Büchi, Semenov, and Presburger

In this section, we review results on arithmetic theories that are the most relevant to our study.

Büchi arithmetic is the first-order theory of the structure $(\mathbb{N}, 0, 1, +, V_2(\cdot,\cdot), \leq)$. By the celebrated Büchi–Bruyère theorem [4, 5], a set $S \subseteq \mathbb{N}^d$ is definable in $(\mathbb{N}, 0, 1, +, V_2(\cdot,\cdot), \leq)$ if and only if the representation of S as a language over the alphabet $\{0,1\}^d$ is recognisable by a deterministic finite automaton (DFA). The theorem is effective, and implies that the satisfiability problem for Büchi arithmetic is in Tower (in fact, Tower-complete) [31, 36].

The situation is different for the existential fragment of this theory. The satisfiability problem is now NP-complete [13], but existential formulae are less expressive [15]. In particular, this fragment fails to capture the binary language $\{10, 01\}^*$. Decision procedures for Büchi arithmetic have been successfully implemented and used to automatically prove many results in combinatorics on words and number theory [35].

Semenov arithmetic is the first-order theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, \leq)$. Its decidability follows from the classical work of Semenov on sparse predicates [33, 34], and an explicit decision procedure was given by Cherlin and Point [6, 28]. Similarly to Büchi arithmetic, Semenov arithmetic is TOWER-complete [8, 29]; however, its existential fragment has only been known to be in NEXPTIME [2]. The paper [42] provides applications of this fragment to solving systems of string constraints with string-to-integer conversion functions.

Büchi–Semenov arithmetic is a natural combination of these two theories. Differently from Büchi and Semenov arithmetics, the $\exists^*\forall^*$-fragment of this logic is undecidable [6]. In view of this, the recent result showing that the satisfiability problem of existential Büchi–Semenov arithmetic is in EXPSPACE [11] is surprising. The proof technique, moreover, establishes the membership in EXPSPACE of the extension of existential Büchi–Semenov arithmetic with arbitrary regular predicates given on input as DFAs. Since this extension can express the intersection non-emptiness problem for DFAs, its satisfiability problem is PSPACE-hard [22]. The decision procedure of [11] was applied to give an algorithm for solving real-world instances of word equations with length constraints.

Both first-order theories of the structures $(\mathbb{N}, 0, 1, +, \leq)$ and $(\mathbb{Z}, 0, 1, +, \leq)$ are usually referred to as *Presburger arithmetic*, because the decision problems for these theories are logspace inter-reducible, meaning that each structure can be interpreted in the other. The procedures that we propose in this paper build upon a version of the quantifier-elimination procedure for the first-order theory of the structure $(\mathbb{Z}, 0, 1, +, \leq)$. Standard procedures for this theory [9, 26, 39] are known to be suboptimal when applied to the existential fragment: throughout these procedures, the bit size of the numbers in the formulae grow exponentially faster than in, e.g., geometric procedures for the theory [7]. A remedy to this well-known issue was proposed by Weispfenning [40, Corollary 4.3]. We develop his observation in Section 4.

3 Preliminaries

We usually write $a, b, c, \dots$ for integers, $x, y, z, \dots$ for integer variables, and $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \dots$ and $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \dots$ for vectors of those. By $\boldsymbol{x} \setminus y$ we denote the vector obtained by removing the variable y from $\boldsymbol{x}$. We denote linear-exponential systems and logical formulae with the letters $\varphi, \chi, \psi, \dots$, and write $\varphi(\boldsymbol{x})$ when the (free) variables of φ are among $\boldsymbol{x}$.

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; note that $n \in \mathbb{N}$ can be represented using $\lceil \log(n+1) \rceil$ bits. For $n, m \in \mathbb{Z}$, denote $[n, m] \coloneqq \{n, n+1, \dots, m\}$. The set $\mathbb{N}$ of non-negative integers contains 0.

Terms. As in Equation (1), a *(linear-exponential) term* is an expression of the form

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

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*. We denote terms by the letters $\rho, \sigma, \tau, \dots$, and write $\tau(\boldsymbol{x})$ if all variables of the term τ are in $\boldsymbol{x}$. For a term τ in Equation (2), its 1-norm is $\|\tau\|_1 \coloneqq \sum_{i=1}^{n}(|a_i| + |b_i| + \sum_{j=1}^{n} |c_{i,j}|) + |d|$.

We use the words 'system' and 'conjunction' of constraints interchangeably. While equalities and inequalities of a linear-exponential system are always of the form $\tau = 0$, $\tau \leq 0$, and $\tau < 0$, for the convenience of exposition we often rearrange left- and right-hand sides and write, e.g., $\tau_1 \leq \tau_2$. In our procedures, linear-exponential systems may contain equalities, inequalities, and also *divisibility constraints* $d \mid \tau$, where τ is a term as in Equation (2), $d \in \mathbb{Z}$ is non-zero, and $\mid$ is the *divisibility predicate*, $\{(d, n) \in \mathbb{Z} \times \mathbb{Z} : n = kd \text{ for some } k \in \mathbb{Z}\}$. We write $\mathit{mod}(\varphi)$ for the (positive) least common multiple of all divisors d appearing in divisibility constraints $d \mid \tau$ of a system φ. For purely syntactic reasons, it is sometimes convenient to see a divisibility constraint $d \mid \tau_1 - \tau_2$ as a *congruence* $\tau_1 \equiv_d \tau_2$, where $d \geq 1$ with no loss of generality. We use the term *divisibility constraint* also for these congruences.

Substitutions. Our procedure uses several special kinds of substitutions. Consider a linear-exponential system φ, a term τ, two variables x and y, and $a \in \mathbb{Z} \setminus \{0\}$.

- We write $\varphi[\tau \,/\, x]$ for the system obtained from φ by replacing every *linear occurrence* of x outside modulo operators with τ. To clarify, this substitution only modifies the "$a_i \cdot x_i$" parts of the term in Equation (2), but not the "$c_{i,j} \cdot (x_i \bmod 2^{x_j})$" parts.
- We write $\varphi[\tau \,/\, x \bmod 2^y]$ and $\varphi[\tau \,/\, 2^x]$ for the system obtained from φ by replacing with τ every occurrence of $(x \bmod 2^y)$ and 2^x, respectively.
- We write $\varphi[[\frac{\tau}{a} \,/\, x]]$ for the *vigorous substitution* of $\frac{\tau}{a}$ for x. This substitution works as follows. **1:** Multiply every equality and inequality by a, flipping the signs of inequalities if $a < 0$; this step also applies to inequalities in which x does not occur. **2:** Multiply both sides of divisibility constraints in which x occurs by a, i.e., $d \mid \tau$ becomes $a \cdot d \mid a \cdot \tau$. **3:** Replace with τ every linear occurrence of $a \cdot x$ outside modulo operators. Note that, thanks to step 1, each coefficient of x in the system can be factorised as $a \cdot b$ for some $b \in \mathbb{Z}$.

We sometimes see substitutions $[\tau \,/\, \tau']$ as first-class citizens: functions mapping systems to systems. When adopting this perspective, $\varphi[\tau \,/\, \tau']$ is seen as a function application.

4 Solving systems of linear inequalities over $\mathbb{Z}$

In this section we present Algorithm 1 (GaussQE), a non-deterministic polynomial time quantifier elimination (QE) procedure for solving systems of linear inequalities over $\mathbb{Z}$, or in other words for integer programming. A constraint (equality, inequality, or divisibility) is *linear* if it only contains linear terms, as defined in Section 3. Our algorithm assumes that all inequalities are non-strict ($\tau \leq 0$).

We already mentioned in Section 1 that Integer Programming $\in$ NP is a standard result. Intuitively, the range of each variable is infinite, which necessitates a proof that a suitable (and *small*) range suffices; see, e.g., [3, 27, 38]. Methods developed in these references, however, do not enjoy the flexibility of quantifier elimination: e.g., they either do not preserve formula equivalence or are not actually removing quantifiers.

▶ **Theorem 3.** *Algorithm 1 (GaussQE) runs in non-deterministic polynomial time and, given a linear system $\varphi(\boldsymbol{x}, \boldsymbol{z})$ and variables $\boldsymbol{x}$, produces in each non-deterministic branch β a linear system $\psi_\beta(\boldsymbol{z})$ such that $\bigvee_\beta \psi_\beta$ is equivalent to $\exists \boldsymbol{x}\, \varphi$.*

GaussQE is based on an observation by Weispfenning, who drew a parallel between a weak form of QE and Gaussian variable elimination [40]. Based on this observation and relying on an insight by Bareiss [1] (to be discussed below), Weispfenning sketched a non-deterministic procedure for deciding *closed* existential formulae of Presburger arithmetic in polynomial time. Although the idea of weak QE [40] has since been developed further [23], the NP observation has apparently remained not well known.

Algorithm 1 GaussQE: Gauss–Jordan elimination for integer programming.

Input: $\boldsymbol{x}$: sequence of variables; $\varphi(\boldsymbol{x}, \boldsymbol{z})$: system of linear constraints without $<$.
Output of each branch (β): system $\psi_\beta(\boldsymbol{z})$ of linear constraints.
Ensuring: $\bigvee_\beta \psi_\beta$ is equivalent to $\exists \boldsymbol{x}\, \varphi$.

1: replace each inequality $\tau \leq 0$ in φ with $\tau + y = 0$, where y is a fresh slack variable
2: $\ell \leftarrow 1$; $s \leftarrow ()$ ▷ *s is an empty sequence of substitutions*
3: **foreach** x in $\boldsymbol{x}$ **do**
4: **if** no equality of φ contains x **then continue**
5: **guess** $ax + \tau = 0$ (with $a \neq 0$) $\leftarrow$ equality in φ that contains x
6: $p \leftarrow \ell$; $\ell \leftarrow a$ ▷ *previous and current lead coefficients*
7: **if** τ contains a slack variable y not assigned by s **then**
8: **guess** $v \leftarrow$ integer in $[0, |a| \cdot \mathit{mod}(\varphi) - 1]$
9: append $[v \,/\, y]$ to s
10: $\varphi \leftarrow \varphi[[\frac{-\tau}{a} \,/\, x]]$
11: divide each constraint in φ by p ▷ *in divisibility constraints, both sides are affected*
12: $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
13: **foreach** equality $\eta = 0$ of φ that contains some slack variable y not assigned by s **do**
14: replace $\eta = 0$ with $\eta[0 \,/\, y] \leq 0$ **if** the coefficient at y is positive **else** with $\eta[0 \,/\, y] \geq 0$
15: apply substitutions of s to φ
16: **foreach** x in $\boldsymbol{x}$ that occurs in φ **do**
17: **guess** $r \leftarrow$ integer in $[0, \mathit{mod}(\varphi) - 1]$
18: $\varphi \leftarrow \varphi[r \,/\, x]$
19: **return** φ

Due to space constraints, we omit the proof of Theorem 3, and explain instead only the key ideas. We first consider the specification of GaussQE, in particular non-deterministic branching. We then recall the main underlying mechanism: Gaussian variable elimination (thus retracing and expanding Weispfenning's observation). After that, we discuss extension of this mechanism to tackle inequalities over $\mathbb{Z}$.

Input, output, and non-determinism. The input to GaussQE is a system φ of linear constraints, as well as a sequence $\boldsymbol{x}$ of variables to eliminate. The algorithm makes non-deterministic guesses in lines 5, 8, and 17. Output of each branch (of the non-deterministic execution) is specified at the top: it is a system ψ_β of linear constraints, in which all variables x in $\boldsymbol{x}$ have been eliminated. For any specific non-deterministic branch, call it β, the output system ψ_β may not necessarily be equivalent to $\exists \boldsymbol{x}\, \varphi$, but the disjunction of all outputs across all branches must be: $\bigvee_\beta \psi_\beta$ has the same set of satisfying assignments as $\exists \boldsymbol{x}\, \varphi$.[3]

The number of non-deterministic branches (individual paths through the execution tree) is usually exponential, but each of them runs in polynomial time. (This is true for all algorithms presented in this paper.) If all variables of the input system φ are included in $\boldsymbol{x}$, then each branch returns a conjunction of numerical assertions that evaluates to true or false.

[3] Formally, an assignment is a map ν from (free) variables to $\mathbb{Z}$. It satisfies a constraint if replacing each z in the domain of ν with $\nu(z)$ makes the constraint a true numerical assertion.

Gaussian elimination and Bareiss' method. Consider a system φ of linear equations (i.e., equalities) over fields, e.g., $\mathbb{R}$ or $\mathbb{Q}$, and let $\boldsymbol{x}$ be a vector of variables that we wish to eliminate from φ. We recall the Gauss–Jordan variable elimination algorithm, proceeding as follows:

01: $\ell \leftarrow 1$
02: **foreach** x in $\boldsymbol{x}$ **do**
03:     **if** no equality of φ contains x **then continue**
04:     **let** $ax + \tau = 0$ (with $a \neq 0$) $\leftarrow$ an arbitrary equality in φ that contains x
05:     $p \leftarrow \ell$; $\quad \ell \leftarrow a$
06:     $\varphi \leftarrow \varphi[[\frac{-\tau}{a} \,/\, x]]$
07:     divide each constraint in φ by p
08: **return** φ

By removing from this code all lines involving p and ℓ (lines 01, 05 and 07), we obtain a naive version of the procedure: an equation is picked in line 04 and used to remove one of its occurring variables in line 06. Indeed, applying a vigorous substitution $[[\frac{-\tau}{a} \,/\, x]]$ to an equality $bx + \sigma = 0$ is equivalent to first multiplying this equality by the lead coefficient a and then subtracting $b \cdot (ax + \tau) = 0$. The result is $-b\tau + a\sigma = 0$, and x is eliminated.

An insightful observation due to Bareiss [1] is that, after multiple iterations, coefficients accumulate non-trivial common factors. Lines 01, 05, and 07 take advantage of this. Indeed, line 07 divides every equation by such a common factor. Importantly, if all numbers in the input system φ are integers, then the division is without remainder. To show this, Bareiss uses a linear-algebraic argument based on an application of the Desnanot–Jacobi identity (or, more generally, Sylvester's identity) for determinants [1, 10, 21]. Over $\mathbb{Q}$, this makes it possible to perform Gaussian elimination (its 'fraction-free one-step' version) in PTime. (This is not the only polynomial-time method; cf. [32, Section 3.3].)

Gaussian elimination for systems of equations can be extended to solving over $\mathbb{Z}$, by introducing divisibility constraints: line 06 becomes $\varphi \leftarrow \varphi[[\frac{-\tau}{a} \,/\, x]] \wedge (a \mid \tau)$. However, while the running time of the procedure remains polynomial, its effect becomes more modest: the procedure reduces a system of linear equations over $\mathbb{Z}$ to an equivalent system of equations featuring variables not in $\boldsymbol{x}$ and multivariate linear congruences that may still contain variables from $\boldsymbol{x}$. To completely eliminate $\boldsymbol{x}$, further computation is required. For our purposes, non-deterministic guessing is a good enough solution to this problem; see the final **foreach** loop in lines 16–18 of GaussQE.

From equalities to inequalities. GaussQE extends Bareiss' method to systems of inequalities over $\mathbb{Z}$. As above, the method allows us to control the (otherwise exponential) growth of the bit size of numbers. Gaussian elimination is, of course, still at the heart of the algorithm (see lines 2–6, 10, and 11), and we now discuss two modifications:

- Line 1 introduces *slack variables* ranging over $\mathbb{N}$. These are internal to the procedure and are removed at the end (lines 13–15).
- In line 5 the equality $ax + \tau = 0$ is selected non-deterministically.

The latter modification is required for the correctness (more precisely: completeness) of GaussQE. Geometrically, for a satisfiable system of inequalities over $\mathbb{Z}$ consider the convex polyhedron of all solutions over $\mathbb{R}$ first. At least one of solutions over $\mathbb{Z}$ must lie in or near a facet of this polyhedron. Line 5 of Algorithm 1 attempts to guess this facet. The amount of slack guessed in line 8 corresponds to the distance from the facet. Observe that if $ax + \tau = 0$ corresponds to an equality of the original system φ, then every solution of φ needs to satisfy $ax + \tau = 0$ exactly, and so there is no slack (lines 8–9 are not taken).

The values chosen for the slack variables in line 8 have, in fact, a counterpart in the standard decision procedures for Presburger arithmetic. When the latter pick a term ρ to substitute, the substitutions in fact introduce $\rho + k$ for k ranging in some $[0, \ell]$, where ℓ depends on $mod(\varphi)$. The amount of slack considered in GaussQE corresponds to these values of k. (Because of this parallel, making the range of guesses in line 8 symmetric, i.e., $|v| \leq |a| \cdot mod(\varphi) - 1$, extends our procedure to the entire existential Presburger arithmetic.)

5 Solving linear-exponential systems over $\mathbb{N}$: an overview

In this section we give an overview of our non-deterministic procedure to solve linear-exponential systems over $\mathbb{N}$. The procedure is split into Algorithms 2–4. A more technical analysis of these algorithms is given later in Section 6.

Whenever non-deterministic Algorithms 1–4 call one another, the return value is always just the output of a single branch, rather than (say) the disjunction over all branches.

Algorithm 2 (LinExpSat). This is the main procedure. It takes as input a linear-exponential system φ without divisibility constraints and decides whether φ has a solution over $\mathbb{N}$. The procedure relies on first (non-deterministically) fixing a linear ordering θ on the exponential terms 2^x occurring in φ (line 2). For technical convenience, this ordering contains a term 2^{x_0}, with x_0 fresh variable, and sets $2^{x_0} = 1$. Variables are iteratively eliminated starting from the one corresponding to the leading exponential term in θ (i.e., the biggest one), until reaching x_0 (lines 3–16). The elimination of each variable is performed by first rewriting the system (in lines 8–14) into a form admissible for Algorithm 3 discussed below. This rewriting introduces new variables, which will never occur in exponentials throughout the entire procedure and are later eliminated when the procedure reaches x_0. Overall, the termination of the procedure is ensured by the decreasing number of exponentiated variables. After LinExpSat rewrites φ, it calls Algorithm 3 to eliminate the currently biggest variable.

Algorithm 3 (ElimMaxVar). This procedure takes as input an ordering θ, a *quotient system induced by* θ and a *delayed substitution*. Let us introduce these notions.

Quotient systems. Let $\theta(\boldsymbol{x})$ be the ordering $2^{x_n} \geq 2^{x_{n-1}} \geq \dots \geq 2^{x_0} = 1$, where $n \geq 1$. A *quotient system induced by* θ is a system $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$ of equalities, inequalities, and divisibility constraints $\tau \sim 0$, where $\sim \in \{<, \leq, =, \equiv_d : d \geq 1\}$ and τ is an *quotient term (induced by θ)*, that is, a term of the form

$$a \cdot 2^{x_n} + f(\boldsymbol{x}') \cdot 2^{x_{n-1}} + b \cdot x_{n-1} + \tau'(x_0, \dots, x_{n-2}, \boldsymbol{z}'),$$

where $a, b \in \mathbb{Z}$, $f(\boldsymbol{x}')$ is a linear term, and τ' is a linear-exponential term in which the variables from $\boldsymbol{z}'$ do not occur exponentiated. Furthermore, for every variable z' in $\boldsymbol{z}'$, the quotient system φ features the inequalities $0 \leq z' < 2^{x_{n-1}}$. The variables in $\boldsymbol{x}$, $\boldsymbol{x}'$ and $\boldsymbol{z}'$ form three disjoint sets, which we call the *exponentiated variables*, the *quotient variables* and the *remainder variables* of the system φ, respectively. We also refer to the term $b \cdot x_{n-1} + \tau'(x_0, \dots, x_{n-2}, \boldsymbol{z}')$ as the *least significant part* of the quotient term τ.
Importantly, quotient terms are **not** linear-exponential terms.

Here is an example of a quotient system induced by $2^{x_3} \geq 2^{x_2} \geq 2^{x_1} \geq 2^{x_0} = 1$, and having quotient variables $\boldsymbol{x}' = (x_1', x_2')$ and remainder variables $\boldsymbol{z}' = (z_1', z_2')$

$$\begin{aligned} -2^{x_3} + (2 \cdot x_1' - x_2' - 1) \cdot 2^{x_2} + \big\{ -2 \cdot x_2 + 2^{x_1} - (z_1' \bmod 2^{x_1}) \big\} &\leq 0, & 0 \leq z_1' < 2^{x_2}, \\ x_1' \cdot 2^{x_2} + \big\{ x_1 + z_2' - 5 \big\} &= 0, & 0 \leq z_2' < 2^{x_2}. \end{aligned}$$

The curly brackets highlight the least significant parts of two terms of the system, the other parts being $\pm z_1'$ and $\pm z_2'$ stemming from the inequalities on the right.

Delayed substitution. This is a substitution of the form $[x' \cdot 2^{x_{n-1}} + z' \,/\, x_n]$, where 2^{x_n} is the leading exponential term of θ. Our procedure delays the application of this substitution until x_n occurs linearly in the system φ. One can think of this substitution as an equality $(x_n = x' \cdot 2^{x_{n-1}} + z')$ in φ that must not be manipulated for the time being.

Back to ElimMaxVar, given a quotient system $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$ induced by θ and the delayed substitution $[x' \cdot 2^{x_{n-1}} + z' \,/\, x_n]$, the goals of this procedure are to (i) eliminate the quotient variables $\boldsymbol{x}' \setminus x'$; (ii) eliminate all occurrences of the leading exponential term 2^{x_n} of θ and apply the delayed substitution to eliminate the variable x_n; (iii) finally, remove x'. Upon exit, ElimMaxVar gives back to LinExpSat a (non-quotient) linear-exponential system where x_n has been eliminated; i.e., a system with one fewer exponentiated variable.

For steps (i) and (iii), the procedure relies on the Algorithm 1 (GaussQE) for eliminating variables in systems of inequalities, from Section 4. This is where flexibility of QE is important: in line 22 some variables are eliminated and some are not. Step (ii) is instead implemented by Algorithm 4.

Algorithm 4 (SolvePrimitive). The goal of this procedure is to rewrite a system of constraints where x_n occurs exponentiated with another system where all constraints are linear. The specification of the procedure restricts the output further. At its core, SolvePrimitive tailors Semenov's proof of the decidability of the first-order theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, \leq)$ [34] to a small syntactic fragment, which we now define.

Primitive linear-exponential systems. Let u, v be two variables. A linear-exponential system is said to be (u, v)-*primitive* whenever all its (in)equalities and divisibility constraints are of the form $a \cdot 2^u + b \cdot v + c \sim 0$, with $a, b, c \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \equiv_d : d \geq 1\}$.

The input to SolvePrimitive is a (u, v)-primitive linear-exponential system. This procedure removes all occurrences of 2^u in favour of linear constraints, working under the assumption that $u \geq v$. This condition is ensured when ElimMaxVar invokes SolvePrimitive. The variable u of the primitive system in the input corresponds to the term $x_n - x_{n-1}$, and the variable v stands for the variable x' in the delayed substitution $[x' \cdot 2^{x_{n-1}} + z' \,/\, x_n]$. ElimMaxVar ensures that $x_n - x_{n-1} \geq x'$.

6 Algorithms 2–4: a walkthrough

Having outlined the interplay between Algorithms 2–4, we move to their technical description, and present the key ideas required to establish the correctness of our procedure for solving linear-exponential systems over $\mathbb{N}$.

6.1 Algorithm 2: the main loop

Let $\varphi(x_1, \dots, x_n)$ be an input linear-exponential system (with no divisibility constraints). As explained in the summary above, LinExpSat starts by guessing an ordering $\theta(x_0, \dots, x_n)$ of the form $t_1 \geq t_2 \geq \dots \geq t_n \geq 2^{x_0} = 1$, where $(t_1, \dots, t_n)$ is a permutation of the terms $2^{x_1}, \dots, 2^{x_n}$, and x_0 is a fresh variable used as a placeholder for 0. Note that if φ is satisfiable (over $\mathbb{N}$), then θ can be guessed so that $\varphi \land \theta$ is satisfiable; and conversely no such θ exists if φ is unsatisfiable. For the sake of convenience, we assume in this section that the ordering $\theta(x_0, \dots, x_n)$ guessed by the procedure is $2^{x_n} \geq 2^{x_{n-1}} \geq \dots \geq 2^{x_1} \geq 2^{x_0} = 1$.

Algorithm 2 LinExpSat: A procedure to decide linear-exponential systems over $\mathbb{N}$.

Input: $\varphi(x_1,\dots,x_n)$: linear-exponential system (without divisibility constraints).
Output: True ($\top$) if φ has a solution over $\mathbb{N}$, and otherwise false ($\bot$).

1: **let** x_0 be a fresh variable ▷ *placeholder for 0*
2: **guess** $\theta \leftarrow$ ordering of the form $t_1 \geq t_2 \geq \dots \geq t_n \geq 2^{x_0} = 1$, where $(t_1,\dots,t_n)$ is a permutation of the terms $2^{x_1},\dots,2^{x_n}$
3: **while** θ is not the ordering $(2^{x_0} = 1)$ **do**
4: $\quad 2^x \leftarrow$ leading exponential term of θ ▷ *in the i-th iteration, 2^x is t_i*
5: $\quad 2^y \leftarrow$ successor of 2^x in θ ▷ *and 2^y is t_{i+1}*
6: $\quad \varphi \leftarrow \varphi[w \,/\, (w \bmod 2^x) : w \text{ is a variable}]$
7: $\quad \boldsymbol{z} \leftarrow$ all variables z in φ such that z is x or z does not appear in θ
8: $\quad$ **foreach** z in $\boldsymbol{z}$ **do** ▷ *form a quotient system induced by θ*
9: $\qquad$ **let** x' and z' be two fresh variables
10: $\qquad \varphi \leftarrow \varphi \land (0 \leq z' < 2^y)$
11: $\qquad \varphi \leftarrow \varphi[z' \,/\, (z \bmod 2^y)]$
12: $\qquad \varphi \leftarrow \varphi[(z' \bmod 2^w) \,/\, (z \bmod 2^w) : w \text{ is such that } \theta \text{ implies } 2^w \leq 2^y]$
13: $\qquad \varphi \leftarrow \varphi[(x' \cdot 2^y + z') \,/\, z]$ ▷ *replaces only the linear occurrences of z*
14: $\qquad$ **if** z is x **then** $(x_0', z_0') \leftarrow (x', z')$ ▷ *for delayed substitution, see next line*
15: $\quad \varphi \leftarrow$ ElimMaxVar$(\theta, \varphi, [x_0' \cdot 2^y + z_0' \,/\, x])$
16: $\quad$ remove 2^x from θ
17: **return** $\varphi(\mathbf{0})$ ▷ *evaluates to $\top$ or $\bot$*

The **while** loop starting in line 3 manipulates φ and θ, non-deterministically obtaining at the end of the ith iteration a system $\varphi_i(\boldsymbol{x},\boldsymbol{z})$ and an ordering $\theta_i(\boldsymbol{x})$, where $\boldsymbol{x} = (x_0,\dots,x_{n-i})$ and $\boldsymbol{z}$ is a vector of i fresh variables. The non-deterministic guesses performed by LinExpSat are such that the following properties (I1)–(I3) are loop invariants across all branches, whereas (I4) is an invariant for at least one branch (below, $i \in [0,n]$ and $(\varphi_0,\theta_0) \coloneqq (\varphi,\theta)$):

I1. All variables that occur exponentiated in φ_i are among $x_0,\dots,x_{n-i}$.
I2. θ_i is the ordering $2^{x_{n-i}} \geq 2^{x_{n-i-1}} \geq \dots \geq 2^{x_1} \geq 2^{x_0} = 1$.
I3. All variables z in $\boldsymbol{z}$ are such that $z < 2^{x_{n-i}}$ is an inequality in φ_i.
I4. $\varphi_i \land \theta_i$ is equisatisfiable with $\varphi \land \theta$ over $\mathbb{N}$.

More precisely, writing $\bigvee_\beta \psi_\beta$ for the disjunction of all the formulae $\varphi_i \land \theta_i$ obtained across all non-deterministic branches, we have that $\bigvee_\beta \psi_\beta$ and $\varphi \land \theta$ are equisatisfiable. Therefore, whenever $\varphi \land \theta$ is satisfiable, (I4) holds for at least one branch. If $\varphi \land \theta$ is instead unsatisfiable, then (I4) holds instead for all branches.

The invariant above is clearly true for φ_0 and θ_0, with $\boldsymbol{z}$ being the empty set of variables. Item (I2) implies that, after n iterations, θ_n is $2^{x_0} = 1$, which causes the **while** loop to exit. Given θ_n, properties (I1) and (I3) force the values of x_0 and of all variables in $\boldsymbol{z}$ to be zero, thus making $\varphi \land \theta$ equisatisfiable with $\varphi_n(\mathbf{0})$ in at least one branch of the algorithm, by (I4). In summary, this will enable us to conclude that the procedure is correct.

Let us now look at the body of the **while** loop. Its objective is simple: manipulate the current system, say φ_i, so that it becomes a quotient system induced by θ_i, and then call Algorithm 3 (ElimMaxVar). For these systems, note that 2^x and 2^y in lines 4–5 correspond to $2^{x_{n-i}}$ and $2^{x_{n-i-1}}$, respectively. Behind the notion of quotient system there are two goals. One of them is to make sure that 2^x and 2^y are not involved in modulo operations.

(We will discuss the second goal in Section 6.2.) The **while** loop achieves this goal as follows:

- Since 2^x is greater than every variable in φ_i, every $(w \bmod 2^x)$ can be replaced with w.
- For 2^y instead, we "divide" every variable z that might be larger than it. Observe that z is either x or from the vector $\boldsymbol{z}$ in (I3) of the invariant. The procedure replaces every linear occurrence of z with $x' \cdot 2^y + z'$, where x' and z' are fresh variables and z' is a residue modulo 2^y, that is, $0 \leq z' < 2^y$.

The above-mentioned replacement simplifies all modulo operators where z appears: $(z \bmod 2^y)$ becomes z', and every $(z \bmod 2^w)$ such that θ_i entails $2^w \leq 2^y$ becomes $(z' \bmod 2^w)$. We obtain in this way a quotient system induced by θ_i, and pass it to ElimMaxVar.

Whilst the goal we just discussed is successfully achieved, we have not in fact eliminated the variable x completely. Recall that, according to our definition of substitution, occurrences of 2^x in the system φ are unaffected by the application of $[x' \cdot 2^y + z' \,/\, x]$ in line 13 of LinExpSat. Because of this, the procedure keeps this substitution as a *delayed substitution* for future use, to be applied (by ElimMaxVar) when x will finally occur only linearly.

6.2 Algorithm 3: elimination of leading variable and quotient variables

Let $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$ be a quotient system induced by an ordering $\theta(\boldsymbol{x})$, with $\boldsymbol{x}$ exponentiated, $\boldsymbol{x}'$ quotient and $\boldsymbol{z}'$ remainder variables, and consider a delayed substitution $[x' \cdot 2^y + z' \,/\, x]$. ElimMaxVar removes $\boldsymbol{x}'$ and x, obtaining a linear-exponential system ψ that adheres to the loop invariant of LinExpSat. This is done by following the three steps described in the summary of the procedure, which we now expand.

Step (i): lines 3–22. This step aims at calling Algorithm 1 (GaussQE) to eliminate all variables in $\boldsymbol{x}' \setminus x'$. There is, however, an obstacle: these variables are multiplied by 2^y. Here is where the second goal behind the notion of quotient system comes into play: making sure that least significant parts of quotient terms can be bounded in terms of 2^y. To see what we mean by this and why it is helpful, consider below an inequality $\tau \leq 0$ from φ, where $\tau = a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + \rho(\boldsymbol{x} \setminus x, \boldsymbol{z}')$ and ρ is the least significant part of τ.

Since φ is a quotient system induced by θ, all variables and exponential terms 2^w appearing in ρ are bounded by 2^y, and thus every solution of $\varphi \wedge \theta$ must also satisfy $|\rho| \leq \|\rho\|_1 \cdot 2^y$. More precisely, the value of ρ must lie in the interval $[(r-1) \cdot 2^y + 1, r \cdot 2^y]$ for some $r \in [-\|\rho\|_1, \|\rho\|_1]$. The procedure guesses one such value r (line 9). The inequality $\tau \leq 0$ can be rewritten as

$$\big(a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + r \cdot 2^y \leq 0\big) \;\wedge\; \big((r-1) \cdot 2^y < \rho \leq r \cdot 2^y\big). \tag{3}$$

Fundamentally, $\tau \leq 0$ has been split into a "left part" and a "right part", shown with big brackets around. The "right part" $(r-1) \cdot 2^y < \rho \leq r \cdot 2^y$ is made of two linear-exponential inequalities featuring none of the variables we want to eliminate ($\boldsymbol{x}'$ and x). Following the same principle, the procedure produces similar splits for all strict inequalities, equalities, and divisibility constraints of φ. In the pseudocode, the "left parts" of the system are stored in the formula γ, and the "right parts" are stored in the formula ψ.

Let us focus on a "left part" $a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + r \cdot 2^y \leq 0$ in γ. Since θ implies $2^x \geq 2^y$, we can factor out 2^y from this constraint, obtaining the inequality $a \cdot 2^{x-y} + f(\boldsymbol{x}') + r \leq 0$. There we have it: the variables $\boldsymbol{x}' \setminus x'$ occur now linearly in γ and can be eliminated thanks to GaussQE. For performing this elimination, the presence of 2^{x-y} is unproblematic. In fact, the procedure uses a placeholder variable u for 2^{x-y} (line 1), so that γ is in fact a linear system with, e.g., inequalities $a \cdot u + f(\boldsymbol{x}') + r \leq 0$. Observe that inequalities $\boldsymbol{x}' \geq \mathbf{0}$ are added to γ in line 22, since GaussQE works over $\mathbb{Z}$ instead of $\mathbb{N}$. This concludes Step (i).

Algorithm 3 ElimMaxVar: Variable elimination for quotient systems.

Input: $\theta(\boldsymbol{x})$: ordering of exponentiated variables;
$\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$: quotient system induced by θ, with $\boldsymbol{x}$ exponentiated, $\boldsymbol{x}'$ quotient, and $\boldsymbol{z}'$ remainder variables;
$[x' \cdot 2^y + z'/x]$: delayed substitution for φ.
Output of each branch (β): $\psi_\beta(\boldsymbol{x} \setminus x, \boldsymbol{z}')$: linear-exponential system such that for every z in $\boldsymbol{z}'$, z does not occur in exponentials and $0 \leq z < 2^y$ occurs in ψ_β.
Ensuring: $(\exists x\, \theta) \wedge \bigvee_\beta \psi_\beta$ is equivalent to $\exists x \exists \boldsymbol{x}'(\theta \wedge \varphi \wedge x = x' \cdot 2^y + z')$ over $\mathbb{N}$.

let u be a fresh variable ▷ *u is an alias for 2^{x-y}*
$\gamma \leftarrow \top$; $\psi \leftarrow \top$

Elimination of $\boldsymbol{x}' \setminus x'$

$\Delta \leftarrow \varnothing$ ▷ *map from linear-exponential terms to $\mathbb{Z}$*
foreach $(\tau \sim 0)$ in φ, where $\sim \in \{=, <, \leq, \equiv_d : d \geq 1\}$ **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 $\{=, <, \leq\}$ **then**
    if $\Delta(\rho)$ is undefined **then**
      guess $r \leftarrow$ integer in $[-\|\rho\|_1, \|\rho\|_1]$
      $\psi \leftarrow \psi \wedge ((r-1) \cdot 2^y < \rho) \wedge (\rho \leq r \cdot 2^y)$
      update Δ : add the key–value pair (ρ, r)
    $r \leftarrow \Delta(\rho)$
    if the symbol $\sim$ is $<$ **then**
      guess $\sim' \leftarrow$ sign in $\{=, <\}$; $\psi \leftarrow \psi \wedge (\rho \sim' r \cdot 2^y)$; $\sim \leftarrow \leq$
      if the symbol $\sim'$ is $=$ **then** $r \leftarrow r + 1$
    $\gamma \leftarrow \gamma \wedge (a \cdot u + f(\boldsymbol{x}') + r \sim 0)$
    if the symbol $\sim$ is $=$ **then** $\psi \leftarrow \psi \wedge (r \cdot 2^y = \rho)$
  else ▷ *$\sim$ is $\equiv_d$ for some $d \in \mathbb{N}$*
    guess $r \leftarrow$ integer in $[1, \mathit{mod}(\varphi)]$
    $\gamma \leftarrow \gamma \wedge (a \cdot u + f(\boldsymbol{x}') - r \sim 0)$
    $\psi \leftarrow \psi \wedge (r \cdot 2^y + \rho \sim 0)$
$\gamma \leftarrow$ GaussQE $(\boldsymbol{x}' \setminus x', \gamma \wedge \boldsymbol{x}' \geq \boldsymbol{0})$

$\gamma \leftarrow \gamma[2^u / u]$ ▷ *u now is an alias for $x - y$*
$(\chi, \gamma) \leftarrow$ SolvePrimitive(u, x', γ)
$\chi \leftarrow \chi[x - y / u][x' \cdot 2^y + z' / x]$ ▷ *apply delayed substitution: x is eliminated*

Elimination of x'

if χ is $(-x' \cdot 2^y - z' + y + c = 0)$ for some $c \in \mathbb{N}$ **then**
  guess $b \leftarrow$ integer in $[0, c]$
  $\gamma \leftarrow \gamma \wedge (x' = b)$
  $\psi \leftarrow \psi \wedge (b \cdot 2^y = -z' + y + c)$
else
  let χ be $(-x' \cdot 2^y - z' + y + c \leq 0) \wedge (d \mid x' \cdot 2^y + z' - y - r)$, with $d, r \in \mathbb{N}$, $c \geq 3$
  guess $(b, g) \leftarrow$ pair of integers in $[0, c] \times [1, d]$
  $\gamma \leftarrow \gamma \wedge (x' \geq b) \wedge (d \mid x' - g)$
  $\psi \leftarrow \psi \wedge ((b-1) \cdot 2^y < -z' + y + c) \wedge (-z' + y + c \leq b \cdot 2^y) \wedge (d \mid g \cdot 2^y + z' - y - r)$
assert(GaussQE(x', γ) is equivalent to $\top$) ▷ *upon failure, Algorithm 2 returns $\bot$*
return ψ

Before moving on to Step (ii), we justify the use of the map Δ from line 3. If the procedure were to apply Equation (3) and replace every inequality $\tau \leq 0$ with three inequalities, then multiple calls to ElimMaxVar would produce a system with exponentially many constraints. A solution to this problem is to guess $r \in [-\|\rho\|_1, \|\rho\|_1]$ only once, and use it in all the "left parts" stemming from inequalities in φ having ρ as their least significant part. The "right part" $(r-1) \cdot 2^y < \rho \leq r \cdot 2^y$ is added to ψ only once. The map Δ implements this memoisation, avoiding the aforementioned exponential blow-up. Indeed, the number of least significant parts grows very slowly throughout LinExpSat, as we will see in Section 7.

Step (ii): lines 23–25. The goal of this step is to eliminate all occurrences of the term 2^{x-y}. For convenience, the procedure first reassigns u to now be a placeholder for $x - y$ (line 23). Because of this reassignment, the system γ returned by GaussQE at the end of Step (i) is a (u, x')-primitive linear-exponential system.

The procedure calls Algorithm 4 (SolvePrimitive), which constructs from γ a pair of systems $(\chi_\beta(u), \gamma_\beta(x'))$, which is assigned to (χ, γ). Both are linear systems, and thus all occurrences of 2^{x-y} (rather, 2^u) have been removed. At last, all promised substitutions can be realised (line 25): u is replaced with $x - y$, and the delayed substitution replaces x with $x' \cdot 2^y + z'$. This eliminates x. The only variable that is yet to be removed is x' (Step (iii)).

It is useful to recall at this stage that SolvePrimitive is only correct under the assumption that $u \geq x' \geq 0$. This assumption is guaranteed by the definition of θ, the delayed substitution, and the fact that u is a placeholder for $x - y$ (and we are working over $\mathbb{N}$). Indeed, if $x' = 0$, then the inequality $2^x \geq 2^y$ in θ ensures $u = x - y \geq 0 = x'$. If $x' \geq 1$,

$$
\begin{aligned}
u = x - y = x' \cdot 2^y + z' - y & \qquad \text{delayed substitution} \\
\geq x' \cdot (y+1) + z' - y & \qquad 2^y \geq y + 1, \text{ for every } y \in \mathbb{N} \\
= y \cdot (x' - 1) + x' + z' \geq x'. & \qquad \text{since } x' \geq 1.
\end{aligned}
$$

Step (iii): lines 26–35. This step deals with eliminating the variable x' from the formula $\gamma(x') \wedge \chi(x', z', y) \wedge \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}')$, where ψ contains the "right parts" of φ computed during Step (i). The strategy to eliminate x' follows closely what was done to eliminate the other quotient variables from $\boldsymbol{x}'$ during Step (i): the algorithm first splits the formula $\chi(x', z', y)$ into a "left part", which is added to γ and features the variable x', and a "right part", which is added to ψ and features the variables z' and y. It then eliminates x' by calling GaussQE on γ. To perform the split into "left part" and "right part", observe that χ is a system of the form either $-x' \cdot 2^y - z' + y + c = 0$ or $(-x' \cdot 2^y - z' + y + c \leq 0) \wedge (d \mid x' \cdot 2^y + z' - y - r)$ (see the spec of SolvePrimitive). By arguments similar to the ones used for ρ in Step (i), $-z' + y + c$ can be bounded in terms of 2^y. (Notice, e.g., the similarities between the inequalities in line 34 and the ones in line 10.) After the elimination of x', if GaussQE does not yield an unsatisfiable formula, ElimMaxVar returns the system ψ to LinExpSat.

Before moving on to the description of SolvePrimitive, let us clarify the semantics of the **assert** statement occurring in line 35. It is a standard semantics from programming languages. If an assertion b evaluates to true at runtime, **assert**(b) does nothing. If b evaluates to false instead, the execution aborts and the main procedure (LinExpSat) returns $\perp$. This semantics allows for assertions to query NP problems, as done in line 35 (and in line 11 of SolvePrimitive), without undermining the membership in NP of LinExpSat.

Algorithm 4 SolvePrimitive: A procedure to decompose and linearise primitive systems.

Input: u, v : two varaibles; φ : (u,v)-primitive linear-exponential system.
Output of each branch (β): a pair of linear systems $(\chi_\beta(u), \gamma_\beta(v))$ such that $\chi_\beta(u)$ is either of the form $(u = a)$ or of the form $(u \geq b) \land (d \mid u - r)$, where $a, d, r \in \mathbb{N}$ and $b \geq 3$.
Ensuring: $(u \geq v \geq 0)$ entails that $\bigvee_\beta (\chi_\beta \land \gamma_\beta)$ is equivalent to φ.

1: **let** φ be $(\chi \land \psi)$, where χ is the conjunction of all (in)equalities from φ containing 2^u
2: $(d, n) \leftarrow$ pair of non-negative integers such that $mod(\varphi) = d \cdot 2^n$ and d is odd
3: $C \leftarrow \max\left\{n, 3 + 2 \cdot \lceil \log(\frac{|b|+|c|+1}{|a|}) \rceil : (a \cdot 2^u + b \cdot v + c \sim 0) \text{ in } \chi, \text{ where } \sim \in \{=, <, \leq\}\right\}$
4: **guess** $c \leftarrow$ element of $[0, C-1] \cup \{\star\}$ ▷ *$\star$ signals $u \geq C$*
5: **if** c is not $\star$ **then**
6: $\quad \chi \leftarrow (u = c)$
7: $\quad \gamma \leftarrow \varphi[2^c \,/\, 2^u]$
8: **else** ▷ *assuming $u \geq C$, (in)equalities in χ simplify to $\top$ or $\bot$*
9: $\quad$ **assert**(χ has no equality, and in all its inequalities 2^u has a negative coefficient)
10: $\quad$ **guess** $r \leftarrow$ integer in $[0, d-1]$ ▷ *remainder of 2^{u-n} modulo d when $u \geq C \geq n$*
11: $\quad$ **assert**($d \mid 2^u - 2^n \cdot r$ is satisfiable)
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: $\quad \chi \leftarrow (u \geq C) \land (d' \mid u - r')$
15: $\quad \gamma \leftarrow \psi[2^n \cdot r / 2^u]$ ▷ *$2^n \cdot r$ is a remainder of 2^u modulo $mod(\psi) = d \cdot 2^n$*
16: **return** (χ, γ)

6.3 Algorithm 4: from primitive systems to linear systems

Consider an input (u,v)-primitive linear-exponential system φ, and further assume we are searching for solutions over $\mathbb{N}$ where $u \geq v$. The goal of SolvePrimitive is to decompose φ (in the sense of monadic decomposition [16, 24]) into two *linear* systems: a system χ only featuring the variable u, and a system γ only featuring v.

To decompose φ, the key parameter to understand is the threshold C for the variable u (line 3). This positive integer depends on two quantities, one for "linearising" the divisibility constraints, and one for "linearising" the equalities and inequalities of φ. Below we first discuss the latter quantity. Throughout the discussion, we assume $u \geq C$, as otherwise the procedure simply replaces u with a value in $[0, C-1]$ (see lines 6 and 7).

Consider an inequality $a \cdot 2^u + b \cdot v + c \leq 0$. Regardless of the values of u and v, as long as $|a \cdot 2^u| > |b \cdot v + c|$ holds, the truth of this inequality will solely depend on the sign of the coefficient a. Since we are assuming $u \geq v$ and $u \geq C \geq 1$, $|a \cdot 2^u| > |b \cdot v + c|$ is implied by $|a| \cdot 2^u > (|b| + |c|) \cdot u$. In turn, this inequality is implied by $u \geq C$, because both sides of the inequalities are monotone functions, $|a| \cdot 2^u$ grows faster than $(|b| + |c|) \cdot u$, and, given $C' := 3 + 2 \cdot \left\lceil \log(\frac{|b|+|c|+1}{|a|}) \right\rceil$ (which is at most C), we have

$$|a| \cdot 2^{C'} \geq |a| \cdot 2^3 \cdot \left(\frac{|b| + |c| + 1}{|a|}\right)^2 > \big(|b| + |c|\big) \cdot 2^{\left\lceil \log(\frac{|b|+|c|+1}{|a|}) \right\rceil + 2} > \big(|b| + |c|\big) \cdot C' \,,$$

where to prove the last inequalities one uses the fact that $2^{x+1} > 2 \cdot x + 1$ for every $x \geq 0$. Hence, when $u \geq C$, every inequality in φ simplifies to either $\top$ or $\bot$, and this is also true for strict inequalities. The Boolean value $\top$ arises when a is negative. The Boolean $\bot$ arises when a is positive, or when instead of an inequality we consider an equality.

It remains to handle the divisibility constraints, again under the assumption $u \geq C$. This is where the second part of the definition of C plays a role. Because $u \geq C \geq n$ (see the definition of (d, n) in line 2), we can guess $r \in [0, d-1]$ such that $\mathit{mod}(\varphi) \mid 2^u - 2^n \cdot r$ (line 10). This constraint is equivalent to $d \mid 2^{u-n} - r$ and, since 2^n and d are coprime, it is also equivalent to $d \mid 2^u - 2^n \cdot r$. It might be an unsatisfiable constraint: the procedure checks for this eventuality in line 11, by solving a discrete logarithm problem (which can be done in NP, see [18]). Suppose a solution is found, say r' (as in line 12). We can then represent the set of solutions of $d \mid 2^u - 2^n \cdot r$ as an arithmetic progression: it suffices to compute the multiplicative order of 2 modulo d, i.e., the smallest positive integer d' such that $d \mid 2^{d'} - 1$. This is again a discrete logarithm problem, but differently from the previous case d' always exists since d and 2 are coprime. The set of solutions of $d \mid 2^u - 2^n \cdot r$ is given by $\{r' + \lambda \cdot d' : \lambda \in \mathbb{Z}\}$, that is, $\mathit{mod}(\varphi) \mid 2^u - 2^n \cdot r$ is equivalent to $d' \mid u - r'$. The procedure then returns $\chi(u) \coloneqq (u \geq C \land d' \mid u - r')$ and $\gamma(v) \coloneqq \psi[2^n \cdot r \,/\, 2^u]$ (see lines 14 and 15), where ψ (defined in line 1) is the system obtained from φ by removing all equalities and inequalities featuring 2^u.

Elaborating the arguments sketched in this section, we can prove that Algorithms 2–4 comply with their specifications.

▶ **Proposition 4.** *Algorithm 2 (LinExpSat) is a correct procedure for deciding the satisfiability of linear-exponential systems over* $\mathbb{N}$.

7 Complexity analysis

We analyse the procedure introduced in Sections 5 and 6 and show that it runs in non-deterministic polynomial time. This establishes Theorem 1 restricted to $\mathbb{N}$.

▶ **Proposition 5.** *Algorithm 2 (LinExpSat) runs in non-deterministic polynomial time.*

To simplify the analysis required to establish Proposition 5, we assume that Algorithms 2–4 store the divisibility constraints $d \mid \tau$ of a system φ in a way such that the coefficients and the constant of τ are always reduced modulo $\mathit{mod}(\varphi)$. For example, if $\mathit{mod}(\varphi) = 5$, the divisibility $5 \mid (7 \cdot x + 6 \cdot 2^x - 1)$ is stored as $5 \mid (2 \cdot x + 2^x + 4)$. Any divisibility can be updated in polynomial time to satisfy this requirement, so there is no loss of generality. Observe that Algorithm 1 (GaussQE) is an exception to this rule, as the divisibility constraints it introduces in line 12 must respect some structural properties throughout its execution. Thus, line 23 of Algorithm 3 (ElimMaxVar) implicitly reduces the output of GaussQE modulo $m = \mathit{mod}(\varphi)$ as appropriate. Since GaussQE runs in non-deterministic polynomial time, the reduction takes polynomial time too.

As is often the case for arithmetic theories, the complexity analysis of our algorithms requires tracking several parameters of linear-exponential systems. Below, we assume an ordering $\theta(\boldsymbol{x}) = (2^{x_n} \geq \dots \geq 2^{x_0} = 1)$ and let φ be either a linear-exponential system or a quotient system induced by θ. Here are the parameters we track:

- The least common multiple of all divisors $\mathit{mod}(\varphi)$, defined as in Section 3.
- The number of equalities, inequalities and divisibility constraints in φ, denoted by $\#\varphi$. (Similarly, given a set T, we write $\#T$ for its cardinality.)
- The 1-norm $\|\varphi\|_1 \coloneqq \max\{\|\tau\|_1 : \tau \text{ is a term appearing in an (in)equality of } \varphi\}$. For linear-exponential terms, $\|\tau\|_1$ is defined in Section 3. For quotient terms τ induced by θ, the 1-norm $\|\tau\|_1$ is defined as the sum of the absolute values of all the coefficients and constants appearing in τ. The definition of $\|\varphi\|_1$ excludes integers appearing in divisibility constraints since, as explained above, those are already bounded by $\mathit{mod}(\varphi)$.

- The *linear norm* $\|\varphi\|_{\mathfrak{L}} \coloneqq \max\{\|\tau\|_{\mathfrak{L}} : \tau$ is a term appearing in an (in)equality of $\varphi\}$. For a linear-exponential term $\tau = \sum_{i=1}^{n}\big(a_i \cdot x_i + b_i \cdot 2^{x_i} + \sum_{j=1}^{n} c_{i,j} \cdot (x_i \bmod 2^{x_j})\big) + d$, we define $\|\tau\|_{\mathfrak{L}} \coloneqq \max\{|a_i|, |c_{i,j}| : i,j \in [1,n]\}$, that is, the maximum of all coefficients of x_i and $(x_i \bmod 2^{x_j})$, in absolute value. For a quotient term induced by θ, of the form $\tau = a \cdot 2^{x_n} + (c_1 \cdot x_1' + \dots + c_m \cdot x_m' + d) \cdot 2^{x_{n-1}} + b \cdot x_{n-1} + \rho(x_0, \dots, x_{n-2}, \boldsymbol{z}')$, we define $\|\tau\|_{\mathfrak{L}} \coloneqq \max\big(|b|, \|\rho\|_{\mathfrak{L}}, \max\{|c_i| : i \in [1,m]\}\big)$, thus also taking into account the coefficients of the quotient variables $x_1', \dots, x_m'$.
- The set of the least significant terms $lst(\varphi,\theta)$ defined as $\big\{ \pm\rho : \rho$ is the least significant part of a term τ appearing in an (in)equality $\tau \sim 0$ of φ, with respect to $\theta \big\}$. We have already defined the notion of the least significant part for a quotient term induced by θ in Section 5. For a (non-quotient) linear-exponential system φ, the least significant part of a term $a \cdot 2^{x_n} + b \cdot x_n + \tau'(x_1, \dots, x_{n-1}, \boldsymbol{z})$ is the term $b \cdot x_n + \tau'$.

Two observations are in order. First, the bit size of a system $\varphi(x_1,\dots,x_n)$ (i.e., the number of bits required to write down φ) is in $O(\#\varphi \cdot n^2 \cdot \log(\max(\|\varphi\|_1, mod(\varphi), 2)))$. Second, together with the number of variables in the input, our parameters are enough to bound all guesses in the procedure. For instance, the value of $c \neq \star$ guessed in line 4 of Algorithm 4 (SolvePrimitive) can be bounded as $O(\log(\max(mod(\gamma), \|\chi\|_1)))$.

The analysis of the whole procedure is rather involved. Perhaps a good overall picture of this analysis is given by the evolution of the parameters at each iteration of the main **while** loop of LinExpSat, described in Lemma 6 below. This loop iterates at most n times, with n being the number of variables in the input system. Below, Φ stands for Euler's totient function, arising naturally because of the computation of multiplicative orders in SolvePrimitive.

▶ **Lemma 6.** *Consider the execution of* LinExpSat *on an input $\varphi(x_1,\dots,x_n)$, with $n \geq 1$. For $i \in [0,n]$, let (φ_i, θ_i) be the pair of a system and ordering obtained after the ith iteration of the **while** loop of line 3, where $\varphi_0 = \varphi$ and θ_0 is the ordering guessed in line 2. Then, for every $i \in [0,n-1]$, φ_{i+1} has at most $n+1$ variables, and for every $\ell, s, a, c, d \geq 1$,*

$$\text{if} \begin{cases} \#lst(\varphi_i,\theta_i) & \leq \ell \\ \#\varphi_i & \leq s \\ \|\varphi_i\|_{\mathfrak{L}} & \leq a \\ \|\varphi_i\|_1 & \leq c \\ mod(\varphi_i) & \mid d \end{cases} \quad \text{then} \begin{cases} \#lst(\varphi_{i+1},\theta_{i+1}) & \leq \ell + 2(i+2) \\ \#\varphi_{i+1} & \leq s + 6(i+2) + 2\cdot\ell \\ \|\varphi_{i+1}\|_{\mathfrak{L}} & \leq 3\cdot a \\ \|\varphi_{i+1}\|_1 & \leq 2^5(i+3)^2(c+2) + 4\cdot\log(d) \\ mod(\varphi_{i+1}) & \mid \operatorname{lcm}(d, \Phi(\alpha_i \cdot d)) \end{cases}$$

*for some $\alpha_i \in [1, (3\cdot a+2)^{(i+3)^2}]$. The $(i+1)$st iteration of the **while** loop of line 3 runs in non-deterministic polynomial time in the bit size of φ_i.*

We iterate the bounds in Lemma 6 to show that, for every $i \in [0,n]$, the bit size of φ_i is polynomial in the bit size of the initial system φ. A challenge is to bound $mod(\varphi_i)$, which requires studying iterations of the map $x \mapsto \operatorname{lcm}(x, \Phi(\alpha \cdot x))$, where α is some positive integer. We show the following lemma:

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

Given Lemma 6, one can show $\alpha_j \leq (\|\varphi\|_{\mathfrak{L}} + 2)^{O(j^3)}$ for every $j \in [0,n-1]$. Then, since $mod(\varphi_0) = 1$, for a given $i \in [0,n-1]$ we apply Lemma 7 with $\alpha = \operatorname{lcm}(\alpha_0,\dots,\alpha_i)$ to derive $mod(\varphi_{i+1}) \leq (\|\varphi\|_{\mathfrak{L}} + 2)^{O(i^6)}$. Once a polynomial bound for the bit size of every φ_i is established, Proposition 5 follows immediately from the last statement of Lemma 6.

8 Proofs of Theorem 1 and Theorem 2

In this section, we discuss how to reduce the task of solving linear-exponential systems over $\mathbb{Z}$ to the non-negative case, thus establishing Theorem 1. We also prove Theorem 2.

Solving linear-exponential systems over $\mathbb{Z}$ (proof of Theorem 1). Let $\varphi(x_1, \dots, x_n)$ be a linear-exponential system $\varphi(x_1, \dots, x_n)$ (without divisibility constraints). We can non-deterministically guess which variables will, in an integer solution $\boldsymbol{u} \in \mathbb{Z}^n$ of φ, assume a non-positive value. Let $I \subseteq [1, n]$ be the set of indices corresponding to these variables. Given $i \in I$, all occurrences of $(x \bmod 2^{x_i})$ in φ can be replaced with 0, by definition of the modulo operator. We can then replace each linear and exponentiated occurrence of x_i with $-x_i$. Let $\chi(\boldsymbol{x})$ be the system obtained from φ after these replacements.

The absolute value of all entries of $\boldsymbol{u}$ is a solution for χ over $\mathbb{N}$. However, χ might feature terms of the form 2^{-x_i} for some $i \in I$ and thus not be a linear-exponential system. We show how to remove such terms. Consider an inequality of the form $\tau \leq \sigma$, where the term τ contains no 2^{-x} and $\sigma \coloneqq \sum_{i \in I} a_i \cdot 2^{-x_i}$ with some a_i non-zero. Since each x_i is a non-negative integer, we have $\left|\sum_{i \in I} a_i \cdot 2^{-x_i}\right| \leq \sum_{i \in I} |a_i| \eqqcolon B$. Therefore, in order to satisfy $\tau \leq \sigma$, any solution $\boldsymbol{v}$ of χ must be such that $\tau(\boldsymbol{v}) \leq B$. We can then non-deterministically add to χ either $\tau < -B$ or $\tau = g$, for some $g \in [-B, B]$.

Case $\tau < -B$. The inequality $\tau \leq \sigma$ is entailed by $\tau < -B$ and can thus be eliminated.

Case $\tau = g$ for some $g \in [-B, B]$. We replace $\tau \leq \sigma$ with $g \leq \sigma$, and multiply both sides of this inequality by $2^{\Sigma_{i \in I} x_i}$. The resulting inequality is rewritten as $g \cdot 2^z \leq \sum_{i \in I} a_i \cdot 2^{z_i}$, where z and all z_i are fresh variables (over $\mathbb{N}$) that are subject to the equalities $z = \sum_{i \in I} x_i$ and $z_i = \sum_{j \in I \setminus \{i\}} x_j$. We add these equalities to χ.

In the above cases we have removed from χ the inequality $\tau \leq \sigma$ in favour of inequalities and equalities only featuring linear-exponential terms. Strict inequalities $\tau < \sigma$ can be handled analogously; and for equalities $\tau = \sigma$ one can separately consider $\tau \leq \sigma$ and $-\tau \leq -\sigma$. The fresh variables z and z_i can be introduced once and reused for all inequalities.

Repeating the process above for each equality and inequality yields (in non-deterministic polynomial time) a linear-exponential system ψ that is satisfiable over $\mathbb{N}$ if and only if the input system φ is satisfiable over $\mathbb{Z}$. The satisfiability of ψ is then checked by calling LinExpSat. Hence, correctness and NP membership follow by Propositions 4 and 5, respectively. ◀

Deciding existential Büchi–Semenov arithmetic (proof of Theorem 2). Let φ be a formula in the existential theory of the structure $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, V_2(\cdot,\cdot), \leq)$ (i.e., Büchi–Semenov arithmetic). By De Morgan's laws, we can bring φ to negation normal form. Negated literals can then be replaced by positive formulae: $\neg V_2(\tau, \sigma)$ becomes $V_2(\tau, z) \land \neg(z = \sigma)$ where z is a fresh variable, $\neg(\tau = \sigma)$ becomes $(\tau < \sigma) \lor (\sigma < \tau)$, and $\neg(\tau \leq \sigma)$ becomes $\sigma < \tau$. Next, occurrences of $V_2(\cdot,\cdot)$ and $2^{(\cdot)}$ featuring arguments other than variables can be "flattened" by introducing extra (non-negative integer) variables: e.g., an occurrence of 2^τ can be replaced with 2^z, where z is fresh, subject to conjoining to the formula φ the constraint $z = \tau$. Lastly, recall that $V_2(x, y)$ can be rephrased in terms of the modulo operator via a linear-exponential system $2 \cdot y = 2^v \land 2 \cdot (x \bmod 2^v) = 2^v$, where v is a fresh variable.

After the above transformation, we obtain a formula ψ of size polynomial with respect to the original one. This formula is a positive Boolean combination of linear-exponential systems. A non-deterministic polynomial-time algorithm deciding ψ first (non-deterministically) rewrites each disjunction $\varphi_1 \lor \varphi_2$ occurring in ψ into either φ_1 or φ_2. After this step, each non-deterministic branch contains a linear-exponential system. The algorithm then calls LinExpSat. Correctness and NP membership then follow by Propositions 4 and 5. ◀

9 Future directions

We have presented a quantifier elimination procedure that decides in non-deterministic polynomial time whether a linear-exponential system has a solution over $\mathbb{Z}$. As a by-product, this result shows that satisfiability for existential Büchi–Semenov arithmetic belongs to NP. We now discuss further directions that, in view of our result, may be worth pursuing.

As mentioned in Section 2, the $\exists^*\forall^*$-fragment of Büchi–Semenov arithmetic is undecidable. Between the existential and the $\exists^*\forall^*$-fragments lies, in a certain sense, the optimisation problem: minimising or maximising a variable subject to a formula. It would be interesting to study whether the natural optimisation problem for linear-exponential systems lies within an optimisation counterpart of the class NP.

With motivation from verification questions, problems involving integer exponentiation have recently been approached with satisfiability modulo theories (SMT) solvers [12]. The algorithms developed in our paper may be useful to further the research in this direction.

Our work considers exponentiation with a single base. In a recent paper [17], Hieronymi and Schulz prove the first–order theory of $(\mathbb{N}, 0, 1, +, 2^{\mathbb{N}}, 3^{\mathbb{N}}, \leq)$ undecidable, where $k^{\mathbb{N}}$ is the predicate for the powers of k. Therefore, the first-order theories of the structures $(\mathbb{N}, 0, 1, +, V_2, V_3, \leq)$ and $(\mathbb{N}, 0, 1, +, 2^{(\cdot)}, 3^{(\cdot)}, \leq)$, which capture $2^{\mathbb{N}}$ and $3^{\mathbb{N}}$, are undecidable. Decidability for the existential fragments of all the theories in this paragraph is open.

Lastly, it is unclear whether there are interesting relaxed versions of linear-exponential systems, i.e., over $\mathbb{R}$ instead of $\mathbb{Z}$. Observe that, in the existential theory of the structure $(\mathbb{R}, 0, 1, +, 2^{(\cdot)}, \leq)$, the formula $x = 2^{y'+z'} \land y = 2^{y'} \land z = 2^{z'}$ defines the graph of the multiplication function $x = y \cdot z$ for positive reals. This "relaxation" seems then only to be decidable subject to (a slightly weaker version of) Schanuel's conjecture [25]. To have an unconditional result one may consider systems where only one variable occurs exponentiated. These are, in a sense, a relaxed version of (u, v)-primitive systems. Under this restriction, unconditional decidability was previously proved by Weispfenning [41].

References

1 Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving Gaussian elimination. *Math. Comput.*, 22:565–578, 1968. doi:10.1090/S0025-5718-1968-0226829-0.

2 Michael Benedikt, Dmitry Chistikov, and Alessio Mansutti. The complexity of Presburger arithmetic with power or powers. In *ICALP*, pages 112:1–112:18, 2023. doi:10.4230/LIPICS.ICALP.2023.112.

3 Itshak Borosh and Leon Bruce Treybig. Bounds on positive integral solutions of linear Diophantine equations. *Proc. Am. Math. Soc.*, 55(2):299–304, 1976. doi:10.2307/2041711.

4 Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *Bull. Belgian Math. Soc.*, 1(2):191–238, 1994. Corrigendum: *Bull. Belgian Math. Soc.*, 1, 1994, 577. doi:10.36045/bbms/1103408547.

5 J. Richard Büchi. Weak second-order arithmetic and finite automata. *Math. Logic Quart.*, 6(1-6):66–92, 1960. doi:10.1002/malq.19600060105.

6 Gregory Cherlin and Françoise Point. On extensions of Presburger arithmetic. In *4th Easter Conference on Model Theory*, volume 86 of *Humboldt-Univ. Berlin Seminarberichte*, pages 17–34, 1986. URL: https://webusers.imj-prg.fr/~francoise.point/papiers/cherlin_point86.pdf.

7 Dmitry Chistikov, Christoph Haase, and Alessio Mansutti. Geometric decision procedures and the VC dimension of linear arithmetic theories. In *LICS*, pages 59:1–59:13, 2022. doi:10.1145/3531130.3533372.

8 Kevin J. Compton and C. Ward Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Log.*, 48(1):1–79, 1990. doi:10.1016/0168-0072(90)90080-L.

9 David C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7(91-99):300, 1972.

10 Charles L. Dodgson. Condensation of determinants, being a new and brief method for computing their arithmetical values. *Proc. R. Soc. Lond.*, 15:150–155, 1867. doi:10.1098/rspl.1866.0037.

11 Andrei Draghici, Christoph Haase, and Florin Manea. Semënov arithmetic, affine VASS, and string constraints. In *STACS*, pages 29:1–29:19, 2024. doi:10.4230/LIPICS.STACS.2024.29.

12 Florian Frohn and Jürgen Giesl. Satisfiability modulo exponential integer arithmetic, 2024. To appear in IJCAR. arXiv:2402.01501.

13 Florent Guépin, Christoph Haase, and James Worrell. On the existential theories of Büchi arithmetic and linear p-adic fields. In *LICS*, pages 1–10, 2019. doi:10.1109/LICS.2019.8785681.

14 Christoph Haase. Approaching arithmetic theories with finite-state automata. In *LATA*, pages 33–43, 2020. doi:10.1007/978-3-030-40608-0_3.

15 Christoph Haase and Jakub Różycki. On the expressiveness of Büchi arithmetic. In *FoSSaCS*, pages 310–323, 2021. doi:10.1007/978-3-030-71995-1_16.

16 Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Monadic decomposition in integer linear arithmetic. In *IJCAR*, pages 122–140, 2020. doi:10.1007/978-3-030-51074-9_8.

17 Philipp Hieronymi and Christian Schulz. A strong version of Cobham's theorem. In *STOC*, pages 1172–1179, 2022. doi:10.1145/3519935.3519958.

18 Antoine Joux, Andrew Odlyzko, and Cécile Pierrot. The past, evolving present, and future of the discrete logarithm. In *Open Problems in Mathematics and Computational Science*, pages 5–36. Springer, 2014. doi:10.1007/978-3-319-10683-0_2.

19 Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

20 Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey. *50 years of integer programming 1958–2008: From the early years to the state-of-the-art.* Springer, 2009.

21 Anna Karapiperi, Michela Redivo-Zaglia, and Maria Rosaria Russo. Generalizations of Sylvester's determinantal identity, 2015. arXiv:1503.00519.

22 Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, pages 254–266, 1977. doi:10.1109/SFCS.1977.16.

23 Aless Lasaruk and Thomas Sturm. Weak integer quantifier elimination beyond the linear case. In *CASC*, pages 275–294, 2007. doi:10.1007/978-3-540-75187-8_22.

24 Leonid Libkin. Variable independence for first-order definable constraints. *ACM Trans. Comput. Log.*, 4(4):431–451, 2003. doi:10.1145/937555.937557.

25 Angus Macintyre and Alex J. Wilkie. On the decidability of the real exponential field. In Piergiorgio Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 441–467. A K Peters, 1996.

26 Derek C. Oppen. Elementary bounds for Presburger arithmetic. In *STOC*, pages 34–37, 1973. doi:10.1145/800125.804033.

27 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

28 Françoise Point. On decidable extensions of Presburger arithmetic: from A. Bertrand numeration sytems to Pisot numbers. *J. Symb. Log.*, 65(3):1347–1374, 2000. doi:10.2307/2586704.

29 Françoise Point. On the expansion $(\mathbb{N}; +, 2^x)$ of Presburger arithmetic. 2007. Preprint. URL: https://webusers.imj-prg.fr/~francoise.point/papiers/Pres.pdf.

30 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.

31 Sylvain Schmitz. Complexity hierarchies beyond Elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.

32 Alexander Schrijver. *Theory of linear and integer programming.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

33 Aleksei L. Semenov. On certain extensions of the arithmetic of addition of natural numbers. *Math. USSR Izv.*, 15(2):401–418, 1980. doi:10.1070/im1980v015n02abeh001252.

34 Aleksei L. Semenov. Logical theories of one-place functions on the set of natural numbers. *Math. USSR Izv.*, 22(3):587–618, 1984. doi:10.1070/im1984v022n03abeh001456.

35 Jeffrey Shallit. *The Logical Approach to Automatic Sequences: Exploring Combinatorics on Words with Walnut.* Cambridge University Press, 2022. doi:10.1017/9781108775267.

36 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973. doi:10.1145/800125.804029.

37 Gilbert Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, 3rd edition, 2003.

38 Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *P. Am. Math. Soc.*, 72(1):155–158, 1978. doi:10.1090/S0002-9939-1978-0500555-0.

39 Volker Weispfenning. The complexity of almost linear Diophantine problems. *J. Symb. Comput.*, 10(5):395–404, 1990. doi:10.1016/S0747-7171(08)80051-X.

40 Volker Weispfenning. Complexity and uniformity of elimination in Presburger arithmetic. In *ISSAC*, pages 48–53, 1997. doi:10.1145/258726.258746.

41 Volker Weispfenning. Deciding linear-exponential problems. *SIGSAM Bull.*, 34(1):30–31, 2000. doi:10.1145/373500.373513.

42 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.

A Theorem 1 holds for any positive integer base given in binary

Algorithm 2 and Algorithm 3 are agnostic with regard to the choice of the base $k \geq 2$. They do not inspect k and see exponential terms k^x as purely syntactic objects. Their logic does not need to be updated to accommodate a different base. To add support for a base k given in input to these two algorithms, it suffices to replace in the pseudocode every 2 with k.

Algorithm 4 is different, as it uses properties of exponentiation. In that algorithm, line 2 must be updated as follows. The pair (d, n) is redefined to be such that d is the largest integer coprime with k dividing $\mathit{mod}(\gamma)$, and k^n is the smallest power of k divisible by $\frac{\mathit{mod}(\gamma)}{d}$. For example, in the case when $k = 6$ and $\mathit{mod}(\gamma) = 60$, we obtain $d = 5$ and $n = 2$, because 36 is the smallest power of 6 divisible by $\frac{60}{5} = 12$. It is clear that $n \leq \lceil\log(\mathit{mod}(\gamma))\rceil$, and the pair (d, n) can be computed in deterministic polynomial time.

Apart from this update, it suffices to replace every occurrence of $2^n \cdot r$ with $\frac{\mathit{mod}(\varphi)}{d} \cdot r$, and every remaining occurrence of 2 with k (except for the constant 2 appearing in the expression $3 + 2 \cdot \lceil\log_k(\frac{|b|+|c|+1}{|a|})\rceil$). This means that the discrete logarithm problems of lines 11–13 must be solved with respect to k instead of 2 (but this can still be done in non-deterministic polynomial time). No other change is necessary.

B Proofs from Section 4: solving systems of linear inequalities over $\mathbb{Z}$

This appendix provides a proof of Theorem 3. We first introduce (in Section B.1) a matrix representation of the input system of constraints and also fix some other notation needed in the subsequent sections. The correctness proof is split into three parts. Key properties of the matrices in the variable elimination process are gathered in Lemma 9 (fundamental lemma) in Section B.2. Based on these properties, we next prove that the divisions in line 11 of Algorithm 1 (GaussQE) are without remainder, in Section B.3. In Section B.4 we prove that steps of the algorithm keep the system of constraints equivalent to the input system. We then provide the complexity analysis of the algorithm in Section B.5, and Theorem 3 will be a direct consequence of the main statements proved in this appendix.

B.1 Introduction to Algorithm 1 (GaussQE) and its analysis

Throughout this section, we refer to Algorithm 1 (GaussQE) on page 6. The input to GaussQE is assumed to be a system (conjunction) φ of equalities ($\tau = 0$), non-strict linear inequalities ($\tau \leq 0$), and divisibility constraints ($d \mid \rho$). Strict inequalities can be handled by the addition of $+1$ to the left-hand sides. Variables of the system φ are partitioned into $\boldsymbol{x}$, to be eliminated, and $\boldsymbol{z}$, to remain in the output.

Slack variables (line 1). In addition to variables $\boldsymbol{x}$ and $\boldsymbol{z}$, the algorithm also uses *slack* variables, $\boldsymbol{y}$, which are auxiliary. The intended domain of slack variables is $\mathbb{N}$. These are internal to the procedure and are eliminated at the end.

Slack variables are not picked in the header (line 3) of the first **foreach** loop (below, we refer to this loop as the *main **foreach** loop*). Instead, a slack variable y gets eliminated when the substitution $[v \,/\, y]$ is applied to φ. This substitution is set up in line 9, right after the constraint from which it arises is used to eliminate some variable from $\boldsymbol{x}$. Thus, each iteration of the loop eliminates one variable from $\boldsymbol{x}$ and, if the chosen constraint stems from an inequality of the input formula, also one slack variable.

Lazy addition. GaussQE performs the substitution $[v \,/\, y]$ lazily: the choice of the value v for the variable y is recorded, but no replacement is carried out in the constraints until the second **foreach** loop, so that variable y continues to be used. Thus, the addition of original constant terms in the constraints and the integers arising from the replacement of y by v is delayed. This laziness turns out convenient when proving the correctness of the algorithm. It is, however, easy to see that the *non-lazy* (eager) version of the algorithm in which each substitution $[v \,/\, y]$ is applied straight away (at line 9) can also be proved correct as long as the lazy version is correct.

Matrix representation of systems of constraints. A system of constraints (equalities, inequalities, and divisibilities) can be written in a matrix. Formally, the *matrix associated to a system* γ has rows that correspond to constraints and columns that correspond to variables and constant terms in γ. An entry in this matrix is the coefficient of the variable (or the constant term) in the constraint. We assume that there are no inequalities in γ, as they have been replaced with equalities following the introduction of slack variables in line 1 as described above. Each divisibility constraint $d \mid \rho$ is represented as the equality $du + \rho = 0$, where u is a fresh dummy variable (different for each constraint). We denote $\boldsymbol{u}$ the vector of all such dummy variables.

Representation of divisibility constraints in the matrix does not influence in any way the execution of GaussQE, which manipulates such constraints as described in the pseudocode.

In other words, the associated matrix is a concept used in the analysis only.

To sum up, after the introduction of slack variables at line 1 and at each iteration of the main **foreach** loop of GaussQE, the system φ will have the form

$$\left[\ A \mid \boldsymbol{c} \mid I' \mid D\ \right] \cdot \left[\ \boldsymbol{x}^\top \mid \boldsymbol{z}^\top \mid -1 \mid \boldsymbol{y}^\top \mid \boldsymbol{u}^\top\ \right]^\top = \mathbf{0}, \tag{4}$$

where:

- $\boldsymbol{x}$ is the vector of input variables to be eliminated;
- $\boldsymbol{z}$ is the vector of all other input variables;
- $\boldsymbol{y}$ is the vector of slack variables (internal to the procedure);
- $\boldsymbol{u}$ is the vector of dummy variables for the analysis of divisibility constraints; and
- the factor $B \coloneqq \left[\ A \mid \boldsymbol{c} \mid I' \mid D\ \right]$ is the *matrix associated to the system* φ*:*
  - A (the main part) has as many columns as there are variables in $\boldsymbol{x}$ and $\boldsymbol{z}$ combined,
  - $\boldsymbol{c}$ (the constant-term block) consists of one column only,
  - I' (the slack part) is the identity matrix interspersed with some zero rows, and
  - D (the dummy part) is the diagonal matrix interspersed with some zero rows.

Main foreach loop of the algorithm (lines 3–12). We call an iteration of the main **foreach** loop *nontrivial* if, at line 4, some equality of the system φ contains the variable x. For $k \geq 0$, we denote by φ_k the system of constraints φ immediately after k nontrivial iterations of the main **foreach** loop. For example, φ_0 is the system obtained after the introduction of slack variables at line 1.

In the non-deterministic execution of GaussQE, each branch computes the sequence of systems $\varphi_0, \varphi_1, \dots, \varphi_k, \dots$, and we denote by $B_0, B_1, \dots, B_k, \dots$ the matrices associated with them. Thus, each non-deterministic branch can be depicted using a commutative diagram:

$$\begin{tikzcd} \varphi \arrow[r] & \varphi_0 \arrow[r] \arrow[d] & \varphi_1 \arrow[r] \arrow[d] & \cdots \arrow[r] & \varphi_k \arrow[r] \arrow[d] & \cdots \\ & B_0 \arrow[r] & B_1 \arrow[r] & \cdots \arrow[r] & B_k \arrow[r] & \cdots \end{tikzcd} \tag{5}$$

Each horizontal arrow in the diagram involves a non-deterministic choice of an equation (line 5), as well as possibly a non-deterministic choice of the amount of slack (line 8). Naturally, systems φ_k and matrices B_k across different branches will, in general, be different (depending on these guesses).

▶ Remark 8. Practically, constraints that originated as equalities (and thus, variables that appear in such constraints) should probably be handled first. Theoretically, the order in which variables are chosen does not matter. However, if a chosen variable appears in a constraint that originated as equality, then in line 5 we may restrict the guessing to such equalities only. This restriction of choice eliminates all non-deterministic branching (guessing) in this iteration of the main **foreach** loop.

Assumptions for the proof. In our analysis it will be convenient to make the following two assumptions:

(A1) the main **foreach** loop picks (in line 5) rows of the matrix in the natural order, i.e., from top to bottom; and

(A2) the variables are eliminated (handled in lines 5–12) in the natural order of the columns of the matrix.

Both assumptions are made only for the sake of convenience of notation, with no loss of generality.

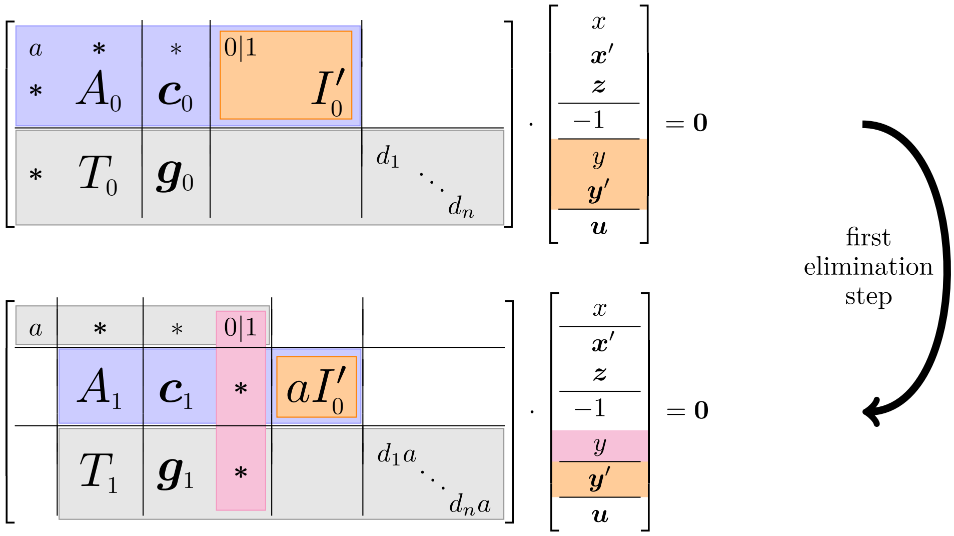


Figure 1 Matrix representation of the linear system in input of GaussQE (above), and matrix obtained after the first iteration of the (main) **foreach** loop of line 3 (below). Submatrices that are shown as empty only contain zeros. Gray rectangles represent divisibility constraints $d \mid \tau$, given in the matrix representation as rows $du + \rho = 0$, where u is a fresh variable ranging over $\mathbb{Z}$. Blue rectangles represent the equalities in the systems; line 5 guesses equalities only from these lines. Inequalities are translated into equalities by introducing slack variables ranging over $\mathbb{N}$ (line 1). The matrix highlighted with the orange rectangle is the identity matrix interspersed with zero rows; its non-zero rows correspond to inequalities in the original system. When a variable is eliminated, the procedure may "lazily" assign a value to a slack variable (see variable y highlighted in magenta; the corresponding magenta column should be understood as a "constant column" once the procedure assigns a value to y). The variables in this figure have the following roles: $\boldsymbol{u}$ are the auxiliary variables encoding divisibilities, x and $\boldsymbol{x}'$ are the variables to be eliminated, $\boldsymbol{z}$ are free variables in the input system that will not be eliminated, y and $\boldsymbol{y}'$ are the slack variables. The -1 in the column vector corresponds to the column of constants.

Operations on nontrivial iterations (lines 5–11). The process followed by a single non-deterministic branch is the *Bareiss-style fraction-free one-step elimination* [1]. We compare it against the standard Gauss–Jordan process from linear algebra for variable elimination (see, e.g., [37]). An example with $k = 1$, that is, the first nontrivial iteration of the process, is depicted in Figure 1.

To begin with, note that applying a substitution $[[\frac{-\tau}{a} \,/\, x]]$ to an equality $bx + \sigma = 0$ in line 10 is equivalent to first multiplying the equality by the lead coefficient a and then subtracting the equality $ax + \tau = 0$ multiplied by b. The result is $-b\tau + a\sigma = 0$. (For example, for $a = b$ and $\tau = \sigma$, we have $(ax + \tau = 0)[[\frac{-\tau}{a} \,/\, x]] = (-a\tau + a\tau = 0)$, which simplifies to true.) Thus, the formula $\varphi[[\frac{-\tau}{a} \,/\, x]]$ is the result of applying this operation to all constraints in φ. We use vigorous substitutions instead of standard substitutions in the main elimination process. This is in order to simplify the handling of (and reasoning about) "Bareiss factors", explained in more detail in Section B.2.

Let us give a matrix representation to these operations. Note that, by Assumption (A1), nontrivial iteration k uses the kth row of the matrix B_k as the "lead row". Thus, in effect, on iteration k, GaussQE applies the following row operations to B_k. Line 10 multiplies all rows by the lead coefficient, then subtracts from each row indexed i the lead row (indexed k) with the multiplier equal to the original coefficient of x in row i. The lead row gets temporarily "zeroed out"; we discuss this in more detail below. Line 11 divides each row by p; Lemma 15 below will prove that this division is without remainder. The substitution in line 9 corresponds to subtracting the y-column of the matrix multiplied by v from the constant-term column; but this operation is only carried out later in line 15 (effectively, this operation is $\boldsymbol{c} \leftarrow \boldsymbol{c} - v\boldsymbol{s}$, where $\boldsymbol{s}$ is the vector of coefficients of the slack variable y).

Notice that the Gauss–Jordan elimination process would, in comparison, leave the lead row (indexed k) unaffected by these operations. (In particular, it would subtract the lead row from all other rows (indexed $i \neq k$) only, not from itself.)

We remark that for the rows of B_k that correspond to divisibility constraints the same reasoning applies. For example, recall from Section 3 that multiplication and division in GaussQE are applied to both sides of divisibility constraints: for λ a nonzero integer, $d \mid \rho$ may evolve (at line 10) into $\lambda d \mid \lambda\rho$ or, if d and all numbers occurring in ρ are divisible by d (at line 11), into $\lambda^{-1}d \mid \lambda^{-1}\rho$. When represented as an equation: $du + \rho = 0$ may evolve into $(\lambda d)u + \lambda\rho = 0$ or $(\lambda^{-1}d)u + \lambda^{-1}\rho = 0$, respectively.

Recasting the lead row as a divisibility constraint (line 12). We already saw that, in line 10, $(ax + \tau = 0)[[\frac{-\tau}{a} \,/\, x]] = (-a\tau + a\tau = 0)$, which simplifies to true. Thus, this line removes the equality $ax + \tau = 0$ from φ_k. Almost immediately, line 12 reinstates it, although recasting it as a divisibility constraint. This is indeed possible: in the Gauss–Jordan elimination process the equality $ax + \tau = 0$ would still be present, but at the same time the variable x would now only be occurring in this equality. Therefore, a suitable value from $\mathbb{Z}$ can be assigned to this variable if and only if the lead coefficient a divides the value of τ.

The fact that no other constraint apart from $ax + \tau = 0$ contains the variable x depends on the fact that all divisibility constraints within φ_k are affected by the vigorous substitution. The alternative — leaving them unaffected — would correspond to the Gauss-style elimination process, which brings the matrix to non-reduced row echelon form, rather than reduced row echelon form; see, e.g., [37]. However, care need to be taken in such a modified process to keep the same values of subdeterminants to establish an analogue of Lemma 15.

B.2 Fundamental lemma of the Gauss–Jordan–Bareiss process

This subsection is devoted to a key property of the Gauss–Jordan variable elimination process with modifications à la Bareiss [1]. We first discuss a similar property of the *standard* Gaussian elimination process and then carry the idea over to our algorithm.

Consider the standard process for a system of equations (equalities) over $\mathbb{R}$ or $\mathbb{Q}$ in which rows are never permuted or multiplied by constants (but only added to one another, possibly with some real multipliers). Then, subject to assumptions related to (A1) and (A2) above, the following two properties hold; we refer the reader to, e.g., [32, Section 3.3]:

- The leading principal minors[4] $\overline{\mu}_1, \overline{\mu}_2, \dots$ of the matrix (i.e., those formed by the first k rows and first k columns, for some k) remain unchanged throughout. For convenience, we denote $\overline{\mu}_0 = 1$.
- After $k \geq 0$ steps of the process, the entry in position (i, j) for $i, j > k$ is the ratio $a_{ij}^{(k)}/\overline{\mu}_k$, where $a_{ij}^{(k)}$ is the kth leading principal minor $\overline{\mu}_k$ bordered by the ith row and jth column,

[4] A *minor* of a matrix is the determinant of a square submatrix. A synonym is *subdeterminant*.

that is,

$$a_{ij}^{(k)} := \begin{vmatrix} a_{11} & \dots & a_{1k} & a_{1j} \\ \vdots & \ddots & \vdots & \vdots \\ a_{k1} & \dots & a_{kk} & a_{kj} \\ a_{i1} & \dots & a_{ik} & a_{ij} \end{vmatrix} .$$

This is because, after k steps, the entries in positions $(i, 1)$, …, (i, k) are all 0.

The statement of the following (fundamental) lemma refers to the bordered minor $b_{ij}^{(k)}$, which is determined, as above (and as in Bareiss' paper [1]), by the entries of the original matrix $B_0 = (b_{ij})$ of the system, from Equation (5):

$$b_{ij}^{(k)} := \begin{vmatrix} b_{11} & \dots & b_{1k} & b_{1j} \\ \vdots & \ddots & \vdots & \vdots \\ b_{k1} & \dots & b_{kk} & b_{kj} \\ b_{i1} & \dots & b_{ik} & b_{ij} \end{vmatrix} .$$

The index j in this notation may refer to a column within any of the four blocks A, $-\boldsymbol{c}$, I', D. A particular case of bordered minors is the kth leading principal minor:

$$\mu_k := b_{k,k}^{(k-1)},$$

that is, the determinant formed by the first k rows and the first k columns.

▶ **Lemma 9** (fundamental lemma)**.** *Consider a branch of non-deterministic execution of* GaussQE*. The following statements hold:*

(a) *For all $k \geq 0$, all $i > k$ and $j > k$, the entry in position (i, j) of the matrix B_k is $b_{ij}^{(k)}$.*
(b) *For all $k \geq 1$, the lead coefficient in the kth nontrivial iteration is μ_k.*
(c) *The division in line 11 is without remainder, except possibly in constraints introduced in some earlier iteration(s) by line 12.*

Proof. Induction on k. Part (c) for iteration k is proved inductively along with parts (a) and (b).

For $k = 0$:

(a) For all $i \geq 1$, $j \geq 1$, the entry in position (i, j) is simply b_{ij}.
(b) There is nothing to prove.
(c) Trivial.

Let $k \geq 1$. Notice that part (b) is a consequence of the inductive hypothesis (for $k - 1$), namely of part (a) with $i = j = k$. We show how, in an inductive step, to arrive at the statement of part (a). Observe that row and column operations applied to the initial matrix B_0 can be represented by rational square matrices L_k and U_k such that $B_k = L_k \cdot B_0 \cdot U_k$. We track the evolution of the matrix $F_k := L_k \cdot B_0$, without considering column operations (i.e., substitutions of integers for slack variables). This corresponds to the *laziness* of our algorithm.

By the inductive hypothesis for $k-1$, part (b), and thanks to Assumptions (A1) and (A2), the lead coefficient in iteration $k \geq 1$ is $b_{k,k}^{(k-1)} = \mu_k$. According to the definition of vigorous substitution, each row $i > k$ in the system is first multiplied by $a = \mu_k$, and then row k is subtracted from it b times where b is the coefficient at x in the equality that corresponds to row i. Let F be the matrix associated to the system $\varphi[[\frac{-\tau}{a} \,/\, x]]$. (Note that F does not reflect any substitution of values for slack variables during the execution of the algorithm.) Observe that, for $i > k$ and $j > k$,

$$f_{ij} = \begin{vmatrix} b_{kk}^{(k-1)} & b_{kj}^{(k-1)} \\ b_{ik}^{(k-1)} & b_{ij}^{(k-1)} \end{vmatrix} = \mu_{k-1} \cdot b_{ij}^{(k)}.$$

Here, the entries of the 2×2 determinant are the entries of the matrix obtained on the previous step of the algorithm (whilst skipping all column operations); this follows from the inductive hypothesis for $k-1$, part (a). The fact that f_{ij} is equal to this determinant can be verified directly (and is crucial, e.g., for Bareiss' paper [1]). Finally, the final equality is the Desnanot–Jacobi identity (or the Sylvester determinantal identity); see, e.g., [1, 10, 21].

Note that the argument above applies regardless of whether the index j is referring to a column of A, to the slack part of the matrix, or to the constant-term column. This completes the proof of part (a).

Finally, we now see that every entry of each row of the matrix F indexed $i > k$ is μ_{k-1} multiplied by an integer. By the inductive hypothesis, part (b), the lead coefficient in iteration $k-1$ is μ_{k-1}, and therefore at iteration k the algorithm has $p = \mu_{k-1}$. Notice that line 5 at iteration $k-1$ ensures that the coefficient at x is non-zero, so $\mu_{k-1} \neq 0$. Therefore, the division in line 11 is without remainder, proving part (c). ◀

In the following Section B.3 we strengthen Lemma 9(c), showing that in fact there cannot be any exceptions at all: division in line 11 is always without remainder.

It would be fair to dub the divisor p in line 11 of GaussQE, or equivalently the factor μ_{k-1} identified in the final paragraph of the proof above, 'the Bareiss factor'. (We will not actually need this term in the paper.)

▶ Remark 10. As seen from the proof of Lemma 9, Assumptions (A1) and (A2) imply that $\mu_k \neq 0$ as long as the main **foreach** loop runs for at least k iterations. This is because equations in which the coefficient at x equals zero are not considered in line 5 of GaussQE.

The following two basic facts about GaussQE, which can be established independently, are in fact direct consequences of Lemma 9.

▶ **Lemma 11.** *At the beginning of each iteration of the main* ***foreach*** *loop, each slack variable that is not yet assigned a value by s occurs in a single constraint of φ (it is an equality).*

Proof. Let column index j correspond to the slack variable in question, which we denote by y. If y appears in a divisibility constraint that was originally introduced by line 12, then it must have been introduced to that constraint by a substitution of line 10, but then line 8 must have assigned a value to y. Therefore, if the slack variable is not yet assigned a value, then it cannot appear in the divisibility constraints introduced by line 12.

We now consider equalities present in the system; this restricts us to rows of the current matrix with indices $i > k$, where k denotes the number of completed nontrivial iterations so far. Lemma 9(a) applies to these rows. Take the constraint corresponding to row i. If variable y was not present in it originally, then the jth column of the bordered minor $b_{ij}^{(k)}$ is zero, and thus $b_{ij}^{(k)} = 0$. In other words, y is still absent from this constraint after k iterations. Since originally each slack variable only features in one constraint, the lemma follows. ◀

Lemma 11 shows that the operation in line 9 is unambiguous, as there cannot be more than one suitable variable y in the preceding line 7. This is because there is originally at most one slack variable in each equality, and the lemma holds throughout the entire run of the algorithm. For the same reason, line 14 is unambiguous as well.

▶ **Lemma 12.** *At every iteration of the main **foreach** loop, all slack variables that are not yet assigned a value by s occur in the constraints of φ with identical coefficients, namely μ_k after k nontrivial iterations.*

Proof. By Lemma 9(b), the lead coefficient in nontrivial iteration i is μ_i; $1 \le i \le k$. This is also the divisor in line 11 in nontrivial iteration $i+1 \le k$. Therefore, after k iterations the slack variable y occurs with coefficient

$$1 \cdot \frac{\mu_1 \cdot \mu_2 \dots \mu_k}{1 \cdot \mu_1 \dots \mu_{k-1}} = \mu_k. \qquad ◀$$

B.3 Integers that appear in the run of Algorithm 1 (GaussQE)

An important milestone is to prove that division in line 11 of GaussQE is without remainder. Lemma 9, part (c), already proves this for all constraints except those introduced by line 12. In this section, we analyse these constraints, as well as other divisibility constraints.

We use the same notation, namely B_k and μ_k, as in the previous Section B.2.

▶ **Lemma 13.** *Assume that all divisions in line 11 in the first k nontrivial iterations of the main **foreach** loop are without remainder. Let $d \mid \rho$ be a divisibility constraint in φ_k. Then:*

- *$d = d^\circ \cdot \mu_k$ if this constraint has evolved from a constraint $d^\circ \mid \rho^\circ$ present in φ_0, and*
- *$d = \mu_k$ if this constraint was introduced to the system at line 12.*

Proof. Let $d^\circ \mid \rho^\circ$ be a constraint present in φ_0. By our definition of vigorous substitutions, during the run of GaussQE the divisor (modulus) d° is multiplied by

$$1, \mu_1, 1^{-1}, \mu_2, \mu_1^{-1}, \dots, \mu_k, \mu_{k-1}^{-1}.$$

Here we used Lemma 9, part (b), which shows that the lead coefficients are principal minors of the matrix B_0; and Remark 10. The product of the factors listed above is μ_k; thus, the divisor (modulus) evolves from d° into $d^\circ \cdot \mu_k$.

Now consider a divisibility constraint introduced to the system by line 12, say in the ith nontrivial iteration. By Lemma 9, part (b), the divisor in this constraint is μ_i. In (nontrivial) iterations $i+1$ through k, this divisor is multiplied by μ_{i+1}, μ_i^{-1}, μ_{i+2}, μ_{i+1}^{-1}, $\dots$, μ_k, μ_{k-1}^{-1}. The product of these factors is μ_k/μ_i, and so the result is μ_k.

We remark that, in both scenarios, the assumption of the lemma is required so that d and ρ are well-defined. ◀

Our next result is an analogue of Lemma 9, part (a), for integers that appear in divisibility constraints. Part (b) of Lemma 14 will be key for the inductive proof that all divisions in line 11 are without remainder.

▶ **Lemma 14.** *Assume that all divisions in line 11 in the first k nontrivial iterations of the main **foreach** loop are without remainder.*

(a) *After these iterations, for all $i \le k$ and all j, the entry in position (i, j) of the obtained matrix is equal to the minor of B_0 formed by the first k rows and columns $1, \dots, i-1, j, i+1, \dots, k$.*

(b) *In the $(k+1)$th nontrivial iteration (if it exists), just before line 11, for all $i \le k+1$ and all j, the entry in position (i, j) of the obtained matrix is equal to μ_k times the minor of B_0 formed by the first $k+1$ rows and columns $1, \dots, i-1, j, i+1, \dots, k+1$.*

Before proving this lemma, we give two restatements of its first part for the reader's convenience. For the proof as well as for these restatements, we will need the standard notion of the *adjugate* of a $k \times k$ rational matrix M. It is the transpose of the cofactor matrix of M: $\mathrm{adj}(M) = (m_{ij})$, where m_{ij} is the determinant of the matrix obtained from M by removing the jth row and the ith column. If M is invertible, then $\mathrm{adj}(M) = M^{-1} \cdot \det M$.

Now, let M_k be the submatrix of B_0 formed by the first k rows and first k columns. Suppose line 12 of GaussQE introduces, on the ith nontrivial iteration of the main **foreach** loop, a divisibility constraint which becomes $d \mid \rho$ after the kth nontrivial iteration ($1 \leq i \leq k$). Then, assuming as above that all the performed divisions are without remainder:

- The term ρ is the linear combination of the first k constraints of the original system with coefficients from the ith row of $\mathrm{adj}(M_k)$, and with the first k handled variables removed. In other words, row ℓ is taken with coefficient equal to the (ℓ, i) cofactor of M_k.
- For every variable w occurring in ρ, its coefficient is equal to the determinant of the matrix obtained from M_k by replacing the ith column with the vector of coefficients of w in the first k constraints of the original system.
- The constant term of ρ is equal to the determinant of the matrix obtained from M_k by replacing the ith column with the vector of constant terms of the first k constraints of the original system.

Proof. We focus on part (a) first. If all the divisions mentioned in the statement of the lemma are without remainder, then, by Lemma 13 (second part), after the k nontrivial iterations, the square submatrix formed by the first k rows and first k columns is equal to $\mu_k I$, where I is the $k \times k$ identity matrix. This is because our algorithm is a variant of the Gauss–Jordan variable elimination: in particular, after each nontrivial iteration (say e) the entire eth column of the matrix becomes zero, with the exception of the entry in position (e, e).

Let $B^{(1..k)}$ denote the submatrix of B_0 formed by the first k rows. Consider the effect of the (nontrivial) k iterations of the main **foreach** loop on $B^{(1..k)}$. These operations amount to manipulating the rows of $B^{(1..k)}$, namely to multiplication of $B^{(1..k)}$ from the left by a square matrix, which we denote by L. Denote, as above, by M_k the submatrix of $B^{(1..k)}$ formed by the first k columns. We know that $L \cdot M_k = \mu_k I$. Since $\mu_k = \det M_k \neq 0$ by Lemma 9(b) and Remark 10, it follows that $L = \mathrm{adj}(M_k)$.

We now consider three cases, depending on the position of the entry (i, j) in the matrix.

Case $i = j \leq k$. This is a diagonal entry of the obtained matrix, within the first k rows. We already saw above that, by Lemma 13 (second part), this entry is equal to μ_k. And indeed, the minor of B_0 formed by the first k rows and columns $1, \dots, i-1, j, i+1, \dots, k$ is in this case simply $\det M_k$.

Case $i \neq j \leq k$. This is an off-diagonal entry within the first k columns. Again, we have already seen that, by the definition of the Gauss–Jordan process, this entry must be 0. Indeed, the minor of B_0 formed by the first k rows and columns $1, \dots, i-1, j, i+1, \dots, k$ in this case contains a repeated column, namely column j.

Case $j > k$. This is the main case, when the entry in question lies to the right of the $\mu_k I$ submatrix. Let $\boldsymbol{b}$ denote the vector formed by entries of B_0 in column j and rows 1 through k. The entry in question is the ith component of $\mathrm{adj}(M_k) \cdot \boldsymbol{b} = M_k^{-1} \boldsymbol{b} \cdot \det M_k$, or in other words of the solution to the system of equations $M_k \cdot \boldsymbol{w} = \boldsymbol{b} \cdot \det M_k$, where $\boldsymbol{w}$ is a vector of fresh variables. By Cramer's rule, this component is equal to the determinant of the matrix obtained from M_k by replacing the ith column by $\boldsymbol{b} \cdot \det M_k$, divided by $\det M_k$. This is exactly the minor from the statement of the lemma.

This completes the proof of part (a). To justify part (b), we observe that all of our arguments remain valid for the $(k+1)$th iteration, except that the submatrix formed by the first

$k+1$ rows and first $k+1$ columns is now $\mu_{k+1}\mu_k I$. This is because the $(k+1)$st iteration multiplies $\mu_k I$ by the new lead coefficient, which is μ_{k+1} by Lemma 9, part (b). The remaining reasoning goes through almost unchanged: we now have $L' \cdot M_{k+1} = \mu_{k+1}\mu_k I$, where I is $(k+1) \times (k+1)$, and so $L' = \mu_k \cdot \mathrm{adj}(M_{k+1})$ instead of $L = \mathrm{adj}(M_k)$. This introduces the extra factor of μ_k, matching the statement of the present Lemma 14, part (b). ◀

▶ **Lemma 15.** *The division in line 11 is without remainder.*

Proof. By Lemma 9, part (c), we can focus on constraints introduced by line 12 only. The proof is by induction on k, the index of a nontrivial iteration of the main **foreach** loop.

The base case is $k = 1$. No constraints have been introduced priori to the 1st iteration, and thus there is nothing to prove. In the inductive step, we assume that the statement holds for the first k nontrivial iterations. Thus, Lemma 14, part (b), applies. But the factor μ_k from its statement is, by Lemma 9, part (b), the lead coefficient of the kth nontrivial iteration: $p = \mu_k$. Therefore, the division by p in line 12 is indeed without remainder. ◀

B.4 Correctness of Algorithm 1 (GaussQE)

In this subsection we show that GaussQE correctly implement its specification.

We first make a basic observation underpinning the proof of correctness of GaussQE. Fix some assignment to target variables $\boldsymbol{x}$ and free variables $\boldsymbol{z}$. It is clear that the input conjunction φ of equalities, inequalities, and divisibility constraints is true if and only if there are nonnegative amounts of slack (that is, an assignment of values from $\mathbb{N}$ to slack variables $\boldsymbol{y}$) that make the equations with slack produced in line 1 of GaussQE as well as all the divisibility constraints true. This equivalence justifies line 1 of GaussQE.

We now prove a sequence of three lemmas, which will later be combined into a proof of correctness of GaussQE. For all lemmas in this section, it is convenient to think of φ as a logical *formula*.

▶ **Lemma 16.** *An assignment satisfies formula φ after lines 10–12 if and only if it has an extension to x that satisfies φ just before these lines. In other words:*

$$\exists x\, \varphi_k \iff \varphi_{k+1}\,, \qquad k \geq 0.$$

Proof. Let ν be a satisfying assignment for the formula φ just before line 10. In this context, ν assigns values to all variables in $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$. All constraints in the formula $\varphi[[\frac{-\tau}{a} \,/\, x]]$ are obtained by multiplying constraints of φ by integers as well as adding such constraints together. Therefore, all these constraints are also satisfied by ν. Dividing both sides of a constraint by a non-zero integer does not change the set of satisfying assignments either. Finally, for line 12 we note that if $\nu(ax + \tau) = 0$, then certainly a divides $\nu(\tau)$.

In the other direction, let ν' be a satisfying assignment for the formula obtained after line 12. Here ν assigns values to all variables to $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ except x. Observe that reversing the application of line 11 preserves the satisfaction, because it amounts to multiplying both sides of each constraint by a non-zero integer. We now show that the value for x can be chosen such that the formula φ at hand *just before* line 10 is satisfied. Indeed, thanks to line 12 the integer $\nu'(\tau)$ is a multiple of a. We will show that assigning $-\nu'(\tau)/a$ to x satisfies φ; note that this choice of value ensures that $\nu'(ax + \tau) = 0$.

Consider any nontrivial (different from Boolean true) constraint in $\varphi[[\frac{-\tau}{a} \,/\, x]]$. If it stems from a constraint $bx + \sigma = 0$, for some (possibly zero) integer b, then this (satisfied) constraint is actually $-b\tau + a\sigma = 0$, simply by the definition of substitution. Therefore, we have $-b\nu'(\tau) + a\nu'(\sigma) = 0$ and $a\nu'(x) + \nu'(\tau) = 0$, from which we conclude $ab\nu'(x) + a\nu'(\sigma) = 0$. Since $a \neq 0$, the constraint $bx + \sigma = 0$ is indeed satisfied by our choice for the value of x. ◀

The reader should not be lulled into a false sense of security by the seemingly very powerful equivalence in the displayed equation in Lemma 16. While lines 10–12 of GaussQE indeed eliminate x, the primary objective after the introduction of slack (line 1) is to bound the range of slack variables. This is not achieved by Lemma 16, but is the subject of the following Lemma 17.

For a sequence of substitutions s, we denote by $\mathrm{vars}(s)$ the set of variables that s assigns values to. Also, by $\boldsymbol{x} \setminus \{y_1, \dots, y_k\}$ we denote the vector obtained from $\boldsymbol{x}$ by removing $y_1, \dots, y_k$. Lemma 17 shows that the range of the guessed slack in line 8 of GaussQE suffices for completeness.

▶ **Lemma 17.** *Consider the $(k+1)$th nontrivial iteration of the main **foreach** loop, for $k \geq 0$. Fix an arbitrary assignment ξ to $\boldsymbol{z}$, $\boldsymbol{x} \setminus x$, and $\mathrm{vars}(s)$. Then, under ξ, the formula φ_k is satisfiable if and only if, for some choice of guesses in lines 5–9, the formula $\varphi_{k+1}[v \,/\, y]$ (or φ_{k+1} if there is no slack at line 7) is satisfiable. In other words:*

$$\exists x \exists \boldsymbol{y'} \, (\boldsymbol{y'} \geq \boldsymbol{0} \wedge \varphi_k) \iff \bigvee_{i \in I} \exists \boldsymbol{y}_i'(\boldsymbol{y}_i' \geq \boldsymbol{0} \wedge \varphi_{k+1}^{(i)}[v_i \,/\, y_i]) \vee \bigvee_{j \in J} \exists \boldsymbol{y'}(\boldsymbol{y'} \geq \boldsymbol{0} \wedge \varphi_{k+1}^{(j)}),$$

where I and J are sets of indices corresponding to the guesses at lines 5–9, with I corresponding to those featuring equalities with slack variables, and $\boldsymbol{y'}$ and $\boldsymbol{y}_i'$ are those slack variables that are not assigned by s before and after the $(k+1)$th iteration takes place.

Proof. Consider the set of equalities in φ_k in which x appears. If in some equality of φ_k involving x all slack variables have already been assigned values by substitutions of s, then we may restrict the choice at line 5 to such equalities only. Indeed, in this case there is no slack at line 7 and line 8 is not executed at all. Thus, in this case the statement follows directly from Lemma 16.

We can therefore assume without loss of generality that, in every equality of φ_k that contains x, there is some slack variable that has not been assigned a value by s yet. With no loss of generality, we assume these slack variables are exactly $\boldsymbol{y'} = (y_1, \dots, y_r)$. (In general, the vector $\boldsymbol{y'}$ from the statement of the lemma may contain more slack variables, but these will play no further role as their values will remain unchanged.) Each y_i belongs to the vector of all slack variables, $\boldsymbol{y}$, but $\boldsymbol{y}$ may well contain further slack variables too. By Lemma 11, each y_e appears in φ exactly once.

The nontrivial direction of the proof is left to right ("only if"). In line with the statement of the lemma, we fix an assignment ξ that assigns values to free variables $\boldsymbol{z}$, each (remaining) variable from $\boldsymbol{x} \setminus x$, and each slack variable from $\mathrm{vars}(s)$. These values will remain fixed throughout the proof.

We assume that φ_k is satisfiable at the beginning of the iteration, and accordingly we can additionally assign a value to the variable x as well as to all slack variables that have not been yet assigned an integer value, so that φ_k is satisfied. We let ν be the assignment that extends ξ accordingly.

All components of the vector $\nu(\boldsymbol{y'}) = (\nu(y_1), \dots, \nu(y_r))$ belong to $\mathbb{N}$. Consider the auxiliary rational vector

$$\boldsymbol{q}^\nu = (q_1, \dots, q_r) \coloneqq \left(\frac{\nu(y_1)}{|a_1'|}, \dots, \frac{\nu(y_r)}{|a_r'|} \right),$$

where a_e' is the coefficient at x in the equality in which the variable y_e appears. Suppose ν is one of the assignments which is an extension of ξ, which satisfies φ_k, and for which the *smallest* component of $\boldsymbol{q}^\nu$ is minimal. Assume without loss of generality that q_1 is this component.

Consider the non-deterministic branch of the algorithm that guesses in line 5 the equality in which the variable y_1 is present. We will now show that $q_1 < mod(\varphi_k)$ or, in other words, $\nu(y_1) < |a_1'| \cdot mod(\varphi_k)$. The number $mod(\varphi_k)$ is the one appearing in the expression for the right endpoint of the interval in line 8 of GaussQE, before the formula manipulation at line 10.

An informal aside. Recall that the standard argument dating back to Presburger picks an interval to which the value assigned to x belongs, and moving this value so that it is close to an endpoint of this interval. In the scenario where the interval is bounded (and not an infinite ray), the two endpoints of the interval correspond to two constraints that have the smallest slack. (More precisely, the set of constraints can be partitioned into two — think left and right — so that, among all the constraints on each side, the "endpoint constraint" has the smallest slack.) This justifies our choice, above, of the assignment ν that minimises the slack in constraints that involve x. In particular, if these constraints include an equation without slack, then this equation is chosen. We remark that slack for different equations is measured *relative* to the absolute value of the coefficient at x: indeed, the inequalities that define the above-mentioned intervals for x are obtained by first dividing each equation (and thus, intuitively, the corresponding slack variable) by the coefficient of x in it.

More formally, denote $a = a_1'$ and assume for the sake of contradiction that $\nu(y_1) \geq |a| \cdot mod(\varphi_k)$. Let m be the coefficient of the slack variable y_1 in the equality $ax + \tau = 0$. Denote by ν' the assignment that agrees with ν on all variables except x and $\boldsymbol{y'}$ as well as on all slack variables eliminated previously, and such that

$$\begin{aligned} &\nu'(x) = \nu(x) \pm m \cdot mod(\varphi_k), \\ &\nu'(y_1) = \nu(y_1) \mp a \cdot mod(\varphi_k) < \nu(y_1), \\ \text{and}\quad &\nu'(y_e) = \nu(y_e) \mp a_e' \cdot mod(\varphi_k), \qquad\qquad e > 1, \end{aligned}$$

where the signs are chosen depending on $a > 0$ or $a < 0$, so that the inequality constraining $\nu'(y_1)$ is satisfied. (As in the definition of $\boldsymbol{q}^\nu$ above, we use a_e' to denote the coefficient at x in the equality in which the variable y_e is present.) We are now going to show that ν' is a satisfying assignment for the formula φ_k.

Observe that if ν satisfies all divisibility constraints of φ_k, then so does ν'. Let us verify that ν' respects the range of each slack variable. Indeed, since $\nu(y_1) \geq |a| \cdot mod(\varphi_k)$, we have $\nu'(y_1) \geq 0$. We now consider the amount of slack in other constraints. Note that

$$\begin{aligned} \nu'(y_e) &= \nu(y_e) \mp a_e' \cdot mod(\varphi_k) && \text{(by the choice of } \nu') \\ &= |a_e'| \cdot q_e \mp a_e' \cdot mod(\varphi_k) && \text{(by the definition of } q_e) \\ &\geq |a_e'| \cdot (q_e - mod(\varphi_k)) && \text{(by cases)} \\ &\geq |a_e'| \cdot (q_1 - mod(\varphi_k)) && \text{(since } q_1 \text{ is the smallest component of } \boldsymbol{q}^\nu) \\ &\geq 0. && \text{(by assumption)} \end{aligned}$$

Therefore, $\nu'(y_e) \geq 0$ for all $e = 1, \dots, r$.

It remains to verify that the assignment ν' satisfies all equalities from φ_k. By Lemma 11, there is only one equality that contains y_1. For this equality, we have

$$\nu'(ax + \tau) = \nu(ax + \tau) \pm a \cdot m \cdot mod(\varphi_k) \mp m \cdot a \cdot mod(\varphi_k) = 0,$$

so under the new assignment this equality remains satisfied. Take any other equality involving x from φ_k, say $a_2'x + \sigma = 0$ in which the unassigned slack variable is y_2. By Lemma 12, the coefficient at y_2 in this equality is equal to m, the coefficient at y_1 in $ax + \tau = 0$. We have

$$\nu'(a_2'x + \sigma) = \nu(a_2'x + \sigma) \pm a_2' \cdot m \cdot \mathit{mod}(\varphi_k) \mp m \cdot a_2' \cdot \mathit{mod}(\varphi_k) = 0.$$

Thus, ν' is also a satisfying assignment for the formula φ_k. However, $\nu'(y_1) < \nu(y_1)$ by the choice of ν', and therefore $\nu'(y_1)/|a| < \nu(y_1)/|a| = q_1$. This inequality contradicts our choice of ν, because we assumed that the smallest component of the vector $\boldsymbol{q}^\nu$ is minimal. Thus, we conclude that the inequality $q_1 < \mathit{mod}(\varphi_k)$ must hold, that is, $\nu(y_1) < |a_1'| \cdot \mathit{mod}(\varphi_k)$. This means that the range specified in line 8 suffices to keep the formula satisfiable after at least one of the possible substitutions.

It remains to prove the other direction (right to left, "if"). A satisfying assignment to the formula $\varphi_{k+1}[v \,/\, y]$ can, by Lemma 16, always be extended to x so that it also satisfies φ_k. This completes the proof. ◀

We turn our attention to lines 13–14 of GaussQE. We need to prove that, when the slack variables are removed in line 14, there is no need to keep the divisibility constraints on the slack. More precisely, suppose that, when the algorithm reaches line 13, φ contains an equality with a slack variable y which is not assigned any value by substitutions from s. Let us assume that the coefficient at y is positive; the negative case is analogous. Thus, the equality has the form $\rho + gy = 0$, $g > 0$. The range of y is $\mathbb{N}$, and by Lemma 11 this variable occurs in no other constraint of φ; therefore, this constraint can be replaced with a conjunction $(\rho \leq 0) \land (g \mid \rho)$. Line 14 only introduces the inequality $\rho \leq 0$, omitting the divisibility constraint $g \mid \rho$.

The following lemma shows that this divisibility is *implied* by other constraints and can thus, indeed, be removed safely. (In practice, it might be beneficial to keep and use the constraint.)

▶ **Lemma 18.** *Denote by φ' the formula obtained at the end of the first **foreach** loop (in lines 3–12), and by ψ' the one after the second **foreach** loop (in lines 13–14), which removes variables $\boldsymbol{y}'$. Then every assignment ν that satisfies ψ' can be extended to $\boldsymbol{y}'$ so that the resulting assignment ν' has $\nu'(y) \in \mathbb{N}$ for all y in $\boldsymbol{y}'$ and moreover ν' satisfies φ'. In other words:*

$$\exists \boldsymbol{y}' \, (\boldsymbol{y}' \geq \mathbf{0} \land \varphi') \iff \psi'.$$

Proof. Take an assignment ν that satisfies the assumptions of the lemma (in particular, ν satisfies ψ'). Take an inequality $\eta[0 \,/\, y] \leq 0$ that is introduced by line 14 of the algorithm. (The case $\eta[0 \,/\, y] \geq 0$ is analogous and we skip it.) Suppose this inequality originates from a constraint $\eta = 0$ picked by line 13, where y is the one slack variable in η that is not assigned a value by substitutions of the sequence s.

Let k be the number of iterations of the first **foreach** loop. We need the following auxiliary notation. Let $x_1, \ldots, x_k$ be the variables picked by the header of the loop in line 3. Assume without loss of generality that, for each i, at the beginning of iteration i the variable x_i is present in at least one of the equalities in φ (i.e., we consider k nontrivial iterations). Suppose the constraint in question, η, arose from the sequence of transformations depicted below, where each η_i is obtained after i iterations, and in particular $\eta = \eta_k$:

$$\begin{array}{lccccccccc}
\text{Equality:} & \eta_0 = 0 & \to & \eta_1 = 0 & \to & \cdots & \to & \eta_{k-1} = 0 & \to & \eta_k = 0 \\
\text{Term without } y\text{:} & \eta_0' & & \eta_1' & & \cdots & & \eta_{k-1}' & & \eta_k' \\
\text{Slack:} & 1 \cdot y & & a_1 \cdot y & & \cdots & & a_{k-1} \cdot y & & a_k \cdot y
\end{array}$$

Each term η'_i is obtained from η_i by dropping the slack variable: $\eta'_i = \eta_i[0 \,/\, y]$. The slack row shows the coefficients of the variable y in $\eta_0, \eta_1, \dots, \eta_k$; see Lemma 12. Here we assume that, on iteration i, the equality picked in line 5 is $a_i x_i + \tau_i = 0$.

Our goal is to prove that $\nu(\eta'_k)$ is divisible by a_k. (This enables us to extend ν to y by assigning $\nu(y) \coloneqq -\nu(\eta'_k)/a_k$.) Denote by p_i the divisor in line 11 in iteration i; then $p_i = a_{i-1}$ for all $i \geq 2$ and $p_1 = 1$. For each $i = k, \dots, 1$ we will show that

$$\nu(\eta'_{i-1}) = \frac{p_i}{a_i} \cdot \nu(\eta'_i).$$

It will then follow that $\nu(\eta'_0) = \nu(\eta'_k) \cdot \prod_{i=1}^k p_i / \prod_{i=1}^k a_i = \nu(\eta'_k)/a_k$. Since $\nu(\eta'_0) \in \mathbb{Z}$, this will conclude the proof.

To justify the equation $\nu(\eta'_{i-1}) = p_i/a_i \cdot \nu(\eta'_i)$, notice that, from the pseudocode of the algorithm, we obtain $\nu(a_i x_i + \tau_i) = 0$. We now develop this intuition into a formal argument. Consider the ith iteration of the first **foreach** loop, and in particular lines 10–12. Suppose that, just before line 10, the constraint $\eta_{i-1} = 0$ has the form $b_i x_i + \hat{\eta}_{i-1} + m_{i-1} y = 0$ for some $m_{i-1} \in \mathbb{Z}$, so that $\eta'_{i-1} = b_i x_i + \hat{\eta}_{i-1}$ for some term $\hat{\eta}_{i-1}$. The result of applying substitution $[[\frac{-\tau_i}{a_i} \,/\, x_i]]$ to η'_{i-1} in line 10 is $-b_i \tau_i + a_i \hat{\eta}_{i-1}$. Since $\eta'_{i-1} = \eta_{i-1}[0 \,/\, y]$, and the variable y does not occur in the term τ_i thanks to Lemma 11, we can apply Lemma 9, part (c), concluding that all coefficients in the term $-b_i \tau_i + a_i \hat{\eta}_{i-1}$ are divisible by $p_i = a_{i-1}$. (Notice that Lemma 9 does not involve any assignments: it applies directly to the syntactic objects that GaussQE works with.)

Thus, we can, in fact, write $\eta'_i = (-b_i \tau_i + a_i \hat{\eta}_{i-1})/p_i$, and moreover $\nu(\eta'_i) \in \mathbb{Z}$. Notice that we can assume $\nu(a_i x_i + \tau_i) = 0$, because even though the formula ψ' (satisfied by ν) does not contain the equality $a_i x_i + \tau_i = 0$, it contains the divisibility constraint $a_i \mid \tau$, and no occurrences of x_i, so we may as well stipulate $\nu(a_i x_i + \tau_i) = 0$. Therefore,

$$\nu(\eta'_i) = \frac{a_i b_i \nu(x_i) + a_i \nu(\hat{\eta}_{i-1})}{p_i} \in \mathbb{Z},$$

and hence $a_i \nu(\hat{\eta}_{i-1}) = p_i \nu(\eta'_i) - a_i b_i \nu(x_i)$. Since $a_i \neq 0$, the number $p_i \nu(\eta'_i)$ must be divisible by a_i, and moreover

$$\nu(\hat{\eta}_{i-1}) = \frac{p_i}{a_i} \cdot \nu(\eta'_i) - b_i \nu(x_i).$$

Recalling that $\eta'_{i-1} = b_i x_i + \hat{\eta}_{i-1}$, we conclude that

$$\nu(\eta'_{i-1}) = \nu(b_i x_i) + \nu(\hat{\eta}_{i-1}) = b_i \nu(x_i) + \frac{p_i}{a_i} \cdot \nu(\eta'_i) - b_i \nu(x_i) = \frac{p_i}{a_i} \cdot \nu(\eta'_i).$$

This completes the proof. ◀

▶ **Lemma 19.** *Algorithm 1 (GaussQE) complies with its specification.*

Proof. Consider an input system $\varphi(\boldsymbol{x}, \boldsymbol{z})$, with $\boldsymbol{x}$ being the set of variables to be eliminated.

We combine the auxiliary results obtained earlier in this section. The main argument shows that, for a given assignment to free variables $\boldsymbol{z}$, if the input system φ is satisfiable, then at least one non-deterministic branch β produces a satisfiable output ψ_β.

1. The guessing in line 8 restricts the choice to a finite set. This 'amount of slack' is shown sufficient in Lemma 17. Thus, the guesses performed in the main (first) **foreach** loop of the algorithm are sufficient.
2. Lemma 18 handles the removal of remove the remaining slack variables, showing that the second **foreach** loop correctly recasts the equalities that still contain slack variables into inequalities.

3. The algorithm also contains a third **foreach** loop (lines 16–18). When this **foreach** loop is reached, the variables from $\boldsymbol{x}$ that still appear in the formula φ do so in divisibility constraints only. Thus, assigning to these variables values in the interval $[0, mod(\varphi) - 1]$, as done in line 17, suffices.

We now formalise the sketch given above. As above, we write φ_0 for the system obtained from φ after executing line 1. We denote by $\mathcal{B}$ a set of indices for the possible non-deterministic branches of the algorithm. In particular, to each $\beta \in \mathcal{B}$ corresponds a sequence of guesses done in lines 5–9. We write $\varphi_\beta(\boldsymbol{x}_\beta, \boldsymbol{z})$ and s_β for the system and for the sequence of substitutions obtained in the non-deterministic branch of β, when the algorithm completes the main **foreach** loop. Here, $\boldsymbol{x}_\beta$ are the variables from $\boldsymbol{x}$ that are removed in the third **foreach** loop; this means that, in φ_β, these variables only occur in divisibility constraints. Similarly, we write ψ'_β for the system obtained when the non-deterministic computation completes the second **foreach** loop. As in the specification, we denote by ψ_β the output of the branch. Lastly, $\boldsymbol{y}_\beta \coloneqq \boldsymbol{y} \setminus \text{vars}(s_\beta)$, where $\boldsymbol{y}$ is the set of slack variables introduced in line 1. We have

$$
\begin{aligned}
\exists \boldsymbol{x}\, \varphi &\iff \exists \boldsymbol{x}\, \exists \boldsymbol{y}\, (\boldsymbol{y} \geq \boldsymbol{0} \wedge \varphi_0) && \text{(introduction of slack variables)}\\
&\iff \bigvee_{\beta \in \mathcal{B}} \exists \boldsymbol{x}_\beta\, \exists \boldsymbol{y}_\beta\, (\boldsymbol{y}_\beta \geq \boldsymbol{0} \wedge \varphi_\beta) s_\beta && \text{(by Lemma 17)}\\
&\iff \bigvee_{\beta \in \mathcal{B}} \exists \boldsymbol{x}_\beta\, (\psi'_\beta s_\beta) && \text{(by Lemma 18)}\\
&\iff \bigvee_{\beta \in \mathcal{B}} \psi_\beta && \text{(lines 15–18).} \quad \blacktriangleleft
\end{aligned}
$$

B.5 Analysis of complexity of Algorithm 1 (GaussQE)

The following Lemma 20 is not required for the proof of Theorem 3, but rather for a more fine-grained analysis in the subsequent proof that Algorithm 2 (LinExpSat) runs in non-deterministic polynomial time (Proposition 5).

▶ **Lemma 20.** *Consider a linear system $\varphi(\boldsymbol{x}, \boldsymbol{z})$ having $n \geq 1$ variables. Let $\alpha \in \mathbb{N}$ be the largest absolute value of coefficients of variables from $\boldsymbol{x}$ in equalities and inequalities of φ. Let ψ be the output of Algorithm 1 on input $(\boldsymbol{x}, \varphi)$. Then, $\|\psi\|_1 \leq (\|\varphi\|_1 + 2)^{4 \cdot (n+1)^2} \cdot mod(\varphi)$, $\#\psi \leq \#\varphi$, and $mod(\psi)$ divides $c \cdot mod(\varphi)$ for some positive $c \leq (\alpha + 2)^{(n+1)^2}$.*

Proof. The inequality $\#\psi \leq \#\varphi$ follows directly from the description of the algorithm: new constraints are only introduced at line 12, which means that they replace equalities that are eliminated previously by substitutions at line 10. As previously, we use the observation that $(ax + \tau = 0)[[\frac{-\tau}{a} \,/\, x]] = (-a\tau + a\tau = 0)$, which simplifies to true.

To prove that $mod(\psi)$ divides $c \cdot mod(\varphi)$ for some positive $c \leq (\alpha + 2)^{(n+1)^2}$, we apply Lemma 13: the least common multiple of all moduli in φ gets multiplied (during the course of the procedure) by μ_k, or possibly by a divisor of this integer. Here k is the total number of nontrivial iterations.

As the next step, we show that $|\mu_k| \leq (\alpha + 2)^{(n+1)^2}$. Indeed, the square submatrix whose determinant is μ_k is formed by the first k rows and first k columns of the matrix B_k from Equation (5). The columns, in particular, correspond to the variables from $\boldsymbol{x}$ eliminated by the procedure; so all entries of the submatrix are coefficients of some k variables from $\boldsymbol{x}$ in equalities and inequalities of φ. The absolute value of each entry is at most α. Therefore,

$$
|\mu_k| \leq k! \cdot \alpha^k \leq n! \cdot \alpha^n \leq n^n \alpha^n \leq 2^{n^2} \alpha^n \leq (\alpha + 2)^{(n+1)^2}, \tag{6}
$$

and so indeed $mod(\psi)$ divides $c \cdot mod(\varphi)$ for some $c \leq (\alpha+2)^{(n+1)^2}$.

It remains to prove the upper bound on the 1-norm of the output, namely $\|\psi\|_1 \leq (\|\varphi\|_1+2)^{4\cdot(n+1)^2} \cdot mod(\varphi)$. Let $C := (\|\varphi\|_1+2)^{(n+1)^2}$. Take an arbitrary equality $\eta = 0$ present in the output system ψ. The following quantities contribute to the 1-norm of η:

- The coefficients of all variables in η. Observe that, since the equality $\eta = 0$ is present in ψ, it was never picked at line 5 of the procedure. Thus, for each *non-zero* coefficient of variables in η we can apply Lemma 9, part (a): this coefficient is equal to a $(k+1)\times(k+1)$ minor of the matrix B_0. The rows for the corresponding square submatrix are associated with equalities and inequalities of the original input system φ, and the columns with the k eliminated variables and the one variable, say w, that we are considering. This variable w must belong to $\boldsymbol{x}$ or $\boldsymbol{z}$, because all slack variables are removed by lines 13–14. Thus, $k+1 \leq n$ and all entries of this submatrix have absolute value at most α. Therefore, by a calculation similar to Equation (6), the coefficient at w is at most $(\alpha+2)^{(n+1)^2} \leq C$.
- The constant term of the equality $\eta' = 0$ that is present in the system just before line 15 and later rewritten into $\eta = 0$. To estimate this term, we can again use Lemma 9(a). The columns of the square submatrix are now associated with the k eliminated variables and with the constant terms. It follows that all entries of the submatrix have absolute value at most $\|\varphi\|_1$, so, in analogy with the chained inequality (6), we obtain the upper bound of C for this term.
- Values of slack variables and their coefficients. Line 15 updates the constant terms in equalities; let us estimate the effect. Firstly, the coefficients of slack variables in these equalities are, again by Lemma 9(a), minors of the matrix B_0. The final column of the $(k+1)\times(k+1)$ submatrix is now the column of coefficients of the slack variable in question. Clearly, this column must be a 0–1 vector with at most one '1', so the upper bound of $(\alpha+2)^{(n+1)^2}$ from Equation (6) remains valid. To estimate the values assigned to slack variables, consider line 8 of the procedure. By Lemma 13, for each $i \geq 0$, the system φ_i obtained after i nontrivial steps of the procedure satisfies $mod(\varphi_i) \mid c \cdot mod(\varphi)$, with $c \leq C$. By Lemma 9(b), the lead coefficients in the iterations are $\mu_1, \ldots, \mu_k$, and they are also at most c. Therefore, no value assigned to a slack variable can exceed $c^2 \cdot mod(\varphi)$, and the contribution of one slack variable with its coefficient is at most $c^3 \cdot mod(\varphi) \leq C^3 \cdot mod(\varphi)$.

Let us now put these contributions together. Observe that there are at most n variables in $\boldsymbol{x}$ in $\boldsymbol{z}$ combined; and in fact only $k \leq n$ slack variables may have been assigned values by s: all other slack variables are removed by lines 13–14. So

$$\|\varphi\|_1 \leq n\cdot C + C + n\cdot C^3 \cdot mod(\varphi) \leq (n+1)\cdot C^3 \cdot mod(\varphi) \leq C^4 \cdot mod(\varphi),$$

because $n+1 \leq 2^{(n+1)^2} \leq (\|\varphi\|_1+2)^{(n+1)^2} = C$ for all $n \geq 1$. This completes the proof. ◀

B.6 Proof of Theorem 3

The correctness of GaussQE is provided by Lemma 19. All the integers appearing in the run have polynomial bit size by Lemma 9 and by Lemma 14 (parts (a) and (b)) combined with Lemma 15. Therefore, the running time is also polynomial in the bit size of the input.

C Proofs from Section 6: correctness of Algorithm 2 (LinExpSat)

This appendix provides a proof of Proposition 4. This is done by first showing the correctness of Algorithm 4 (SolvePrimitive), then the correctness of Algorithm 3 (ElimMaxVar), and lastly the correctness of Algorithm 2 (LinExpSat). The interaction between Algorithms 2–4 has been described in Sections 5 and 6. The flowchart in Figure 2 is provided to remind the reader of the key steps of these algorithms.

Non-deterministic branches. In this appendix, for the convenience of exposition it is sometimes useful to fix a representation for the non-deterministic branches of our procedures. As done throughout the body of the paper, we write β for a non-deterministic branch. It is represented as its *execution trace*, that is a list of entries (tuples) following the control flow of the program. Each entry contains the name of the algorithm being executed, the line number that is currently being executed, and, for lines featuring the non-deterministic **guess** statement, the performed guess. As an example, to line 2 of LinExpSat correspond entries of the form (LinExpSat, "line 2", θ), where θ is the ordering that has been guessed.

We write $\beta = \beta_1\beta_2$ whenever β can be decomposed on a prefix β_1 and suffix β_2.

Global outputs. Our proofs often consider the **global outputs** of Algorithms 3 and 4. This is defined as the set containing all outputs of all branches β of a procedure. For Algorithm 3, the **global output** is a set of linear-exponential systems. For Algorithm 4, this is a set of pairs of linear-exponential systems. Observe that the **ensuring** part of the specification of these algorithms provides properties of their **global outputs**. For instance, in the case of Algorithm 3, the **ensuring** part specifies some properties of the disjunction $\bigvee_\beta \psi_\beta$, which ranges over all branches of the procedure, and the **global output** is just the set of all disjuncts ψ_β appearing in that formula.

In Appendix C.2 we also define **global outputs** for each of the three steps of Algorithm 3. For Steps (i) and (ii), they will be sets of pairs of specific linear-exponential systems (similarly to Algorithm 4). Step (iii) will have a **global output** of the same type as Algorithm 3.

C.1 Correctness of Algorithm 4 (SolvePrimitive)

▶ **Lemma 21.** *Algorithm 4 (SolvePrimitive) complies with its specification.*

Proof. Recall that (u, v)-primitive systems are composed of equalities, (non)strict inequalities, and divisibilities of the form $(a \cdot 2^u + b \cdot v + c \sim 0)$, where u, v are non-negative integer variables, the predicate symbol $\sim$ is from the set $\{=, <, \leq, \equiv_d : d \geq 1\}$, and a, b, c are integers.

Consider an input (u, v)-primitive system φ. Algorithm 4 first splits φ into a conjunction of two subsystems $\chi \wedge \gamma$ (line 1) such that

- the system χ comprises all (in)equalities $(a \cdot 2^u + b \cdot v + c \sim 0)$ with $a \neq 0$;
- the system γ is composed of all other constraints of φ.

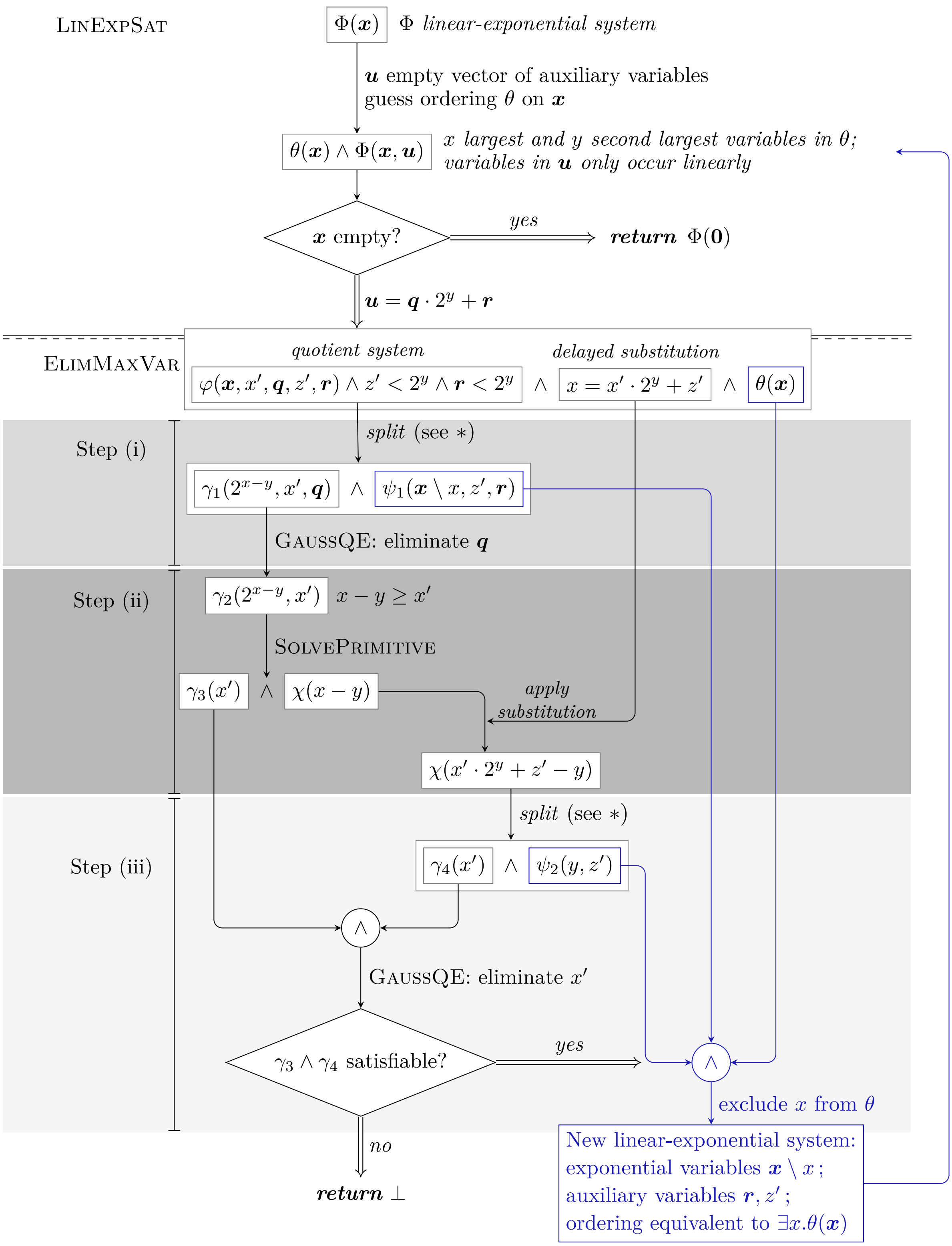
In the sequel, most of our attention is given to the formula χ.

As explained in the body of the paper, the integer constant C defined in line 3 plays a key role in the correctness proof. The following equivalence is immediate:

$$\varphi \iff \Big(\bigvee_{c=0}^{C-1} (u = c) \wedge \varphi[2^c \,/\, 2^u]\Big) \vee \Big(u \geq C \wedge \varphi\Big). \tag{7}$$

In the subformula $\bigvee_{c=0}^{C-1}(u = c) \wedge \varphi[2^c \,/\, 2^u]$ the variable u has already been linearised. This is reflected in the procedure in lines 4–7: if the procedure guesses $c \in [0, C-1]$ in line 4, then it returns a pair $\chi(u) \coloneqq (u = c)$ and $\gamma(v) \coloneqq \varphi[2^c \,/\, 2^u]$, which corresponds to one of the disjuncts of $\bigvee_{c=0}^{C-1}(u = c) \wedge \varphi[2^c \,/\, 2^u]$. The case when $u \geq C$ (equivalently, when the algorithm guesses $c = \star$ in line 4) is more involved and dealt with in the remaining part of the proof. Most of the work is concerned with manipulations of the formula $(u \geq C \wedge \varphi)$.

Figure 2 Flowchart of Algorithms 2 and 3.



$*$: In a nutshell, assuming $z' < 2^y$, $\boldsymbol{r} < 2^y$ and $z < 2^y$ for all $z \in \boldsymbol{x} \setminus x$, we can rewrite an inequality $a \cdot 2^x + f(x', \boldsymbol{q}) \cdot 2^y + \rho(\boldsymbol{x} \setminus x, z', \boldsymbol{r}) \leq 0$ into $\bigvee_{r=-\|\rho\|_1}^{\|\rho\|_1} (\underbrace{a \cdot 2^{x-y} + f(x', \boldsymbol{q}) + r \leq 0}_{\text{part of } \gamma_1/\gamma_4} \wedge \underbrace{(r-1) \cdot 2^y < \rho \leq r \cdot 2^y}_{\text{part of } \psi_1/\psi_2})$.
For equalities, strict inequalities and divisibility constraints, see Lemmas 24 and 25.

Recall that we are working under the assumption that $u \geq v \geq 0$ (as stated in the signature of the procedure), which is essential for handling this case. Let us start by considering the (in)equalities in χ. Take one such (in)equality $a \cdot 2^u + b \cdot v + c \sim 0$, with $\sim \in \{<, \leq, =\}$ and $a \neq 0$. Define $C' \coloneqq 2 \cdot \lceil \log(\frac{|b|+|c|+1}{|a|}) \rceil + 3$. For this constant, we can prove that $|a| \cdot 2^{C'} > \big(|b| + |c|\big) \cdot C'$.

$$
\begin{aligned}
|a| \cdot 2^{C'} = |a| \cdot 2^3 \cdot \left(\frac{|b| + |c| + 1}{|a|}\right)^2 &> \big(|b| + |c|\big) \cdot 2^{\log(\frac{|b|+|c|+1}{|a|})+3} \\
&> \big(|b| + |c|\big) \cdot 2^{\lceil \log(\frac{|b|+|c|+1}{|a|}) \rceil + 2} \\
&> \big(|b| + |c|\big) \cdot \Big(2 \cdot \Big(\Big\lceil \log\big(\frac{|b| + |c| + 1}{|a|}\big)\Big\rceil + 1\Big) + 1\Big) \\
&= \big(|b| + |c|\big) \cdot \Big(2 \cdot \Big\lceil \log\big(\frac{|b| + |c| + 1}{|a|}\big)\Big\rceil + 3\Big) \\
&= \big(|b| + |c|\big) \cdot C',
\end{aligned}
$$

where for the second to last derivation we have used the fact that $2^{x+1} > 2x + 1$ for every $x \in \mathbb{N}$. Note that both $|a| \cdot 2^x$ and $(|b| + |c|) \cdot x$ are monotonous functions. Then, since 2^x grows much faster than x and (by definition) $C \geq C'$, we conclude that $|a| \cdot 2^x > \big(|b| + |c|\big) \cdot x$ for every $x \geq C$. From this fact, we observe that given two integers u, v such that $u \geq v \geq 0$ and $u \geq C \geq 1$, the absolute value of $a \cdot 2^u$ is greater than the absolute value of $b \cdot v + c$:

$$
|a| \cdot 2^u > (|b| + |c|) \cdot u \geq |b| \cdot v + |c| \geq |b \cdot v + c|.
$$

This means that the inequality $a \cdot 2^u + b \cdot v + c \sim 0$ from χ simplifies:

(i) if a is negative and $\sim$ is either $<$ or $\leq$, then the inequality simplifies to $\top$;

(ii) if a is positive or $\sim$ is $=$, then the inequality simplifies to $\bot$.

Let χ' be the formula obtained from χ by applying the two rules above to each inequality. Note that this formula simplifies to either $\top$ or $\bot$. We have:

$$
(u \geq v \geq 0) \implies \Big((u \geq C \land \varphi) \iff (u \geq C \land \chi' \land \gamma)\Big). \tag{8}
$$

The restriction $u \geq v \geq 0$ implies that if χ' is false, then so is $u \geq C \land \varphi$. Thus, in terms of the procedure, the guess $c = \star$ in line 4 should lead to an unsuccessful execution. This case is covered by line 9, where the procedure checks that the condition (ii) above is never satisfied.

In the formula $u \geq C \land \chi' \land \gamma$, the exponential term 2^u does not occur in (in)equalities, but it might still occur in the divisibility constraints of γ. The next step is thus to "linearise" these constraints. Back to the definition of C in line 3, note that $C \geq n$ where $n \in \mathbb{N}$ is the maximum natural number such that 2^n divides $\mathit{mod}(\gamma)$ (note that $\mathit{mod}(\gamma) = \mathit{mod}(\varphi)$). Let d be the positive odd number $\frac{\mathit{mod}(\gamma)}{2^n}$, as defined in line 2. Assuming $u \geq C$, the fact that $C \geq n$ implies that the remainder of 2^u modulo $\mathit{mod}(\varphi)$ is of the form $2^n \cdot r$ for some $r \in [0, d-1]$ (in particular, note that $r = 0$ when $d = 1$). The following tautology holds:

$$
u \geq C \implies \Big(\gamma \iff \bigvee_{r=0}^{d-1} \big((\mathit{mod}(\varphi) \mid 2^u - 2^n \cdot r) \land \gamma[2^n \cdot r \,/\, 2^u]\big)\Big). \tag{9}
$$

We have thus eliminated all occurrences of u from γ, and we are left with the divisibility constraints $\mathit{mod}(\varphi) \mid 2^u - 2^n \cdot r$. Note that these constraints can be reformulated as follows:

$$
\begin{aligned}
\mathit{mod}(\varphi) \mid 2^u - 2^n \cdot r &\iff d \mid 2^{u-n} - r \\
&\iff d \mid 2^u - 2^n \cdot r \qquad \text{since } d \text{ and } 2^n \text{ are coprime}
\end{aligned}
$$

We now want to "linearise" $d \mid 2^u - 2^n \cdot r$. There are two cases.

The first case corresponds to $(d \mid 2^u - 2^n \cdot r)$ not having a solution. That is, the discrete logarithm of $2^n \cdot r$ modulo d does not exist. This happens, for instance, if $r = 0$ and $d \geq 3$. In this case, the corresponding disjunct in Equation (9) simplifies to $\bot$, and accordingly the execution of the procedure aborts, following the **assert** command in line 11.

In the other case, $(d \mid 2^u - 2^n \cdot r)$ has a solution. We remark (even though this will also be discussed later in the complexity analysis of the procedure) that computing a solution for this discrete logarithm problem can be done in non-deterministic polynomial time in the bit size of d and $2^n \cdot r$ (see [18]). In this case, we define ℓ_r to be the discrete logarithm of $2^n \cdot r$ modulo d (ℓ_r corresponds to r' in line 12). Let $R \subseteq [0, d-1]$ be the set of all $r \in [0, d-1]$ such that ℓ_r is defined. Furthermore, let d' be the multiplicative order of 2 modulo d (which exists because d is odd), that is, d' is the smallest integer from the interval $[1, d-1]$ such that $d \mid 2^{d'} - 1$ (as defined in line 13). Let us for the moment assume the following result:

▷ Claim 22. For every $r \in R$ and $u \in \mathbb{Z}$, $(d \mid 2^u - 2^n \cdot r)$ if and only if $(d' \mid u - \ell_r)$.

Thanks to this claim, form Equation (9), we obtain

$$u \geq C \implies \Big(\gamma \iff \bigvee_{r \in R} \big((d' \mid u - \ell_r) \land \gamma[2^n \cdot r \,/\, 2^u]\big)\Big). \tag{10}$$

For the non-deterministic procedure, in the case when ℓ_r defined, the procedure returns $\chi(u) \coloneqq u \geq C \land (d' \mid u - \ell_r)$ and $\gamma[2^n \cdot r \,/\, 2^u]$, as shown in lines 14 and 15. Across all non-deterministic branches, the **ensuring** part of the specification of Algorithm 4 follows by Equations (7), (8), and (10):

$$(u \geq v \geq 0) \implies \tag{11}$$

$$\Big(\varphi \iff \bigvee_{c=0}^{C-1} (u = c) \land \varphi[2^c \,/\, 2^u] \lor \bigvee_{r \in R} (\chi' \land (u \geq C \land d' \mid u - \ell_r) \land \gamma[2^n \cdot r \,/\, 2^u])\Big). \tag{12}$$

Notice that the right-hand side of the double implication (12) corresponds to the set of pairs

$$\{(u = c, \varphi[2^c \,/\, 2^u]) : c \in [0, C-1]\} \cup \{(u \geq C \land d' \mid u - \ell_r, \gamma[2^n \cdot r \,/\, 2^u])\},$$

which is exactly the **global output** of Algorithm 4. This completes the proof Lemma 21, subject to the proof of Claim 22 which is given below. ◀

Proof of Claim 22. Let $r \in R$ and $u \in \mathbb{Z}$. First, recall that by definition of ℓ_r and d' we have $d \mid 2^{\ell_r} - 2^n \cdot r$ and $d \mid 2^{d'} - 1$. This directly implies that, for every $\lambda \in \mathbb{Z}$, the divisibility $d \mid 2^{\lambda \cdot d' + \ell_r} - 2^n \cdot r$ holds. Then, the right-to-left direction is straightforward, as $d' \mid u - \ell_r$ is indeed equivalent to the statement "there is $\lambda \in \mathbb{Z}$ such that $u = \lambda \cdot d' + \ell_r$".

For the left-to-right direction, we need to prove that if $d \mid 2^u - 2^n \cdot r$ then u is of the form $u = \lambda \cdot d' + \ell_r$, for some $\lambda \in \mathbb{Z}$. Consider $t_1, t_2 \in [0, d'-1]$ and $\lambda_1, \lambda_2 \in \mathbb{Z}$ such that $u = \lambda_1 \cdot d' + t_1$ and $\ell_r = \lambda_2 \cdot d' + t_2$. We show that $t_1 - t_2$ is a multiple of d', which in turn implies $u \equiv_{d'} \ell_r$ and therefore that u is of the required form $u = \lambda \cdot d' + \ell_r$, for some $\lambda \in \mathbb{Z}$.

Suppose $t_1 \geq t_2$ (the case of $t_1 < t_2$ is analogous). From $d \mid 2^u - 2^n \cdot r$ and $d \mid 2^{\ell_r} - 2^n \cdot r$ we derive $d \mid 2^u - 2^{\ell_r}$. Then,

$$\begin{aligned}
d \mid 2^u - 2^{\ell_r} &\iff d \mid 2^{\lambda_1 \cdot d' + t_1} - 2^{\lambda_2 \cdot d' + t_2} && \\
&\iff d \mid 2^{t_1} - 2^{t_2} && \text{by definition of } d' \\
&\iff d \mid 2^{t_2}(2^{t_1 - t_2} - 1) && \text{since } t_1 \geq t_2 \\
&\iff d \mid 2^{t_1 - t_2} - 1 && \text{since 2 and } d \text{ are coprime.}
\end{aligned}$$

Hence, since d' is the multiplicative order of 2 modulo d, $t_1 - t_2$ is a multiple of d'. ◀

C.2 Correctness of Algorithm 3 (ElimMaxVar)

In this section of Appendix C, we prove that Algorithm 3 (ElimMaxVar) complies with its specification (see Lemma 34 at the end of the section; page 54).

Before moving to the correctness proof, we would like to remind the reader that Algorithm 3 (ElimMaxVar) is a *non-deterministic quantifier-elimination procedure*. Non-determinism is used to guess integers (see lines 9, 19, 27, and 32) as well as in the calls to Algorithms 1 and 4. The result of computations in each non-deterministic branch β is a linear-exponential system ψ_β. For the set B of all such non-deterministic branches, the **global output** of Algorithm 3 is the set $\{\psi_\beta\}_{\beta \in B}$. According to the specification given to Algorithm 3, for the disjunction $\bigvee_{\beta \in B} \psi_\beta$ (we will prove that) the following equivalence holds:

$$\exists x \exists \boldsymbol{x'} \big(\theta(\boldsymbol{x}) \land \varphi(\boldsymbol{x}, \boldsymbol{x'}, \boldsymbol{z'}) \land (x = x' \cdot 2^y + z')\big) \tag{13}$$

$$\iff \big(\exists x\, \theta(\boldsymbol{x})\big) \land \bigvee_{\beta \in B} \psi_\beta(\boldsymbol{x} \setminus x, \boldsymbol{z'}). \tag{14}$$

This equivalence demonstrates that the variables $x, \boldsymbol{x'}$ have been eliminated from the input formula φ. Since the ordering θ is a simple formula that plays only a structural role in the elimination process, we do not modify it explicitly (this is done in Algorithm 2 directly after the call to Algorithm 3). Because 2^x is the leading exponential term of the ordering θ, it is sufficient to exclude the inequality $(2^y \leq 2^x)$ from this formula. This leads to the full elimination of the variables x and $\boldsymbol{x'}$. Of course, while Algorithm 3 can be regarded as a standard quantifier-elimination procedure, it only works for formulae of a very specific language (namely, the existential formulae of the form (13)).

Three steps: their inputs and outputs

Following the discussion in the body of the paper, Algorithm 3 can be split into three steps, each achieving a specific goal:

- **(i)** lines 1–22: elimination of the quotient variables $\boldsymbol{q} \coloneqq \boldsymbol{x'} \setminus x'$.
- **(ii)** lines 23–25: linearisation of the variable x and then its elimination by application of the delayed substitution $[x' \cdot 2^y + z'/x]$.
- **(iii)** lines 26–35: elimination of the quotient variable x'.

Steps (i) and (iii) are quite similar. The intermediate Step (ii) is essentially made of a single call to Algorithm 4.

Each step can be considered as an independent non-deterministic subroutine with its input and branch/global outputs. Specification 1, Specification 2 and Specification 3 formally define these **inputs** and **global outputs**, for Step (i), Step (ii) and Step (iii), respectively. With these specifications at hand, we organise the proof of correctness of Algorithm 3 as follows. We start by showing that Algorithm 3 is correct as soon as one assumes that the three steps comply with their specification. Afterwards, we prove (in independent subsections) that each step does indeed follow its specification.

Let us first briefly discuss the three specifications. From Specification 1, we see that the input of Step (i) is the same as of Algorithm 3. Its **global output** is a set of pairs of systems $\{(\gamma_i, \psi_i)\}_{i \in I}$, where the ψ_i are linear-exponential systems featuring only variables from the vector $(\boldsymbol{x} \setminus x, \boldsymbol{z'})$, and the γ_i are linear systems only featuring the variables x' and u (which is a placeholder for 2^{x-y}). By **branch output** of this step we mean a particular pair (γ, ψ) of systems from the **global output**. The systems γ and ψ correspond to the content of the homonymous "program variables" in the pseudocode, after line 22 has been executed.

Specification 1 Step (i): Lines 1–22 of Algorithm 3

Input: $\theta(\boldsymbol{x})$: ordering of exponentiated variables;
$\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$: quotient system induced by θ, with $\boldsymbol{x}$ exponentiated, $\boldsymbol{x}'$ quotient, and $\boldsymbol{z}'$ remainder variables;
$[x' \cdot 2^y + z'/x]$: delayed substitution for φ. (Recall: $\boldsymbol{q} \coloneqq \boldsymbol{x}' \setminus x'$.)

Global output: $\{(\gamma_i(u, x'), \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'))\}_{i \in I}$: set of pairs such that for every $i \in I$,

- γ_i is a linear system containing the inequality $x' \geq 0$.
- ψ_i is a linear-exponential system containing inequalities $0 \leq z < 2^y$ for every z in $\boldsymbol{z}'$.

Over $\mathbb{N}$, the formula $\exists \boldsymbol{q}\big(\theta(\boldsymbol{x}) \land \varphi(\boldsymbol{x}, x', \boldsymbol{q}, \boldsymbol{z}') \land (x = x' \cdot 2^y + z')\big)$ is equivalent to

$$\bigvee_{i \in I} \theta(\boldsymbol{x}) \land (x = x' \cdot 2^y + z') \land \exists u\big(\gamma_i(u, x') \land u = 2^{x-y}\big) \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'). \tag{15}$$

Specification 2 Step (ii): Lines 23–25 of Algorithm 3

Input: $\theta(\boldsymbol{x})$: ordering of exponentiated variables (same as in Step (i));
$(\gamma(u, x'), \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}'))$: branch output of Step (i);
$[x' \cdot 2^y + z'/x]$: delayed substitution (same as in Step (i)).

Global output: $\{(\chi_j(y, x', z'), \gamma_j(x'), \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}'))\}_{j \in J}$: set of triples such that for $j \in J$

1. $\chi_j(y, x', z')$ is either $(x' \cdot 2^y + z' - y = a)$ or $(x' \cdot 2^y + z' - y \geq b) \land (d \mid x' \cdot 2^y + z' - y - r)$, for some $a, b, d, r \in \mathbb{N}$ and $b > 2$ (that depend on j);
2. $\gamma_j(x')$ is a linear system containing the inequality $x' \geq 0$.
3. $\psi(\boldsymbol{x} \setminus x, \boldsymbol{z}')$ is the system in the input of the algorithm (it is not modified).

Over $\mathbb{N}$, given $\Psi(\boldsymbol{x}, x', z') \coloneqq \theta(\boldsymbol{x}) \land (x = x' \cdot 2^y + z') \land \exists u\big(\gamma(u, x') \land u = 2^{x-y}\big)$, we have

$$(\exists x \Psi(\boldsymbol{x}, x', z')) \iff \bigvee_{j \in J} (\exists x \theta(\boldsymbol{x})) \land \chi_j(y, x', z') \land \gamma_j(x'). \tag{16}$$

Specification 3 Step (iii): Lines 26–35 of Algorithm 3

Input: $\theta(\boldsymbol{x})$: ordering of exponentiated variables (as in Steps (i));
$(\chi(y, x', z'), \gamma(x'), \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}'))$: branch output of Step (ii).

Global output: $\{\psi_k(y, z') \land \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}')\}_{k \in K}$: set of linear-exponential systems (the system $\psi(\boldsymbol{x} \setminus x, \boldsymbol{z}')$ is as in the input of the algorithm). Over $\mathbb{N}$, the following formula holds:

$$0 \leq z' < 2^y \implies \Big(\exists x'\big(\chi(y, x', z') \land \gamma(x')\big) \iff \bigvee_{k \in K} \psi_k(y, z')\Big). \tag{17}$$

To visualise the specification of Step (ii), consider the diagram given in Figure 2. In this diagram, Step (ii) is enclosed within the darker background rectangle. Four arrows enter this rectangle. Two of them correspond to a branch output of Step (i), and the other two come from the substitution for the variable x and the ordering $\theta(\boldsymbol{x})$. Observe that computations at Step (ii) do not affect the second parameter of the branch output of Step (i). The linear-exponential system $\psi(\boldsymbol{x} \setminus x, \boldsymbol{z}')$ is just propagated to the next step. The **branch output** of Step (ii) is a triple of systems (χ, γ, ψ), corresponding to the content of the homonymous program variables, after line 25 has been executed.

The specification of Step (iii) also follows the diagram from Figure 2. This step takes as input a branch output of the previous step (three arrows that correspond to the systems γ_3, χ, ψ_1) and the ordering $\theta(\boldsymbol{x})$. Its **branch output** is a single linear-exponential system that corresponds to the output of Algorithm 3.

We prove the aforementioned conditional statement about the correctness of Algorithm 3.

▶ **Lemma 23.** *If Steps (i), (ii), and (iii) comply with, respectively, Specifications 1, 2, and 3 then Algorithm 3 (*ElimMaxVar*) complies with its specification.*

Proof. In a nutshell, the proof of the equivalence between the formulas (13) and (14) follows by chaining the equivalences appearing in Specifications 1, 2, and 3.

Consider an input to Algorithm 3: an ordering $\theta(\boldsymbol{x})$, a quotient system $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$ induced by θ, and a delayed substitution $[x' \cdot 2^y + z'/x]$ for φ. According to the specification of Algorithm 3, it suffices to show the aforementioned equivalence and that each **branch output** ψ_β of the algorithm contains the inequalities $0 \le z < 2^y$, for every z in $\boldsymbol{z}'$. Below we focus on proving the equivalence, and derive the additional property on ψ_β as a by-product.

Following Specification 1, with this input, Step (i) constructs a set of pairs of systems $\{(\gamma_i(u, x'), \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'))\}_{i \in I}$. Given $i \in I$, we define the formula

$$\Psi_i(\boldsymbol{x}, x', z') \coloneqq \theta(\boldsymbol{x}) \land (x = x' \cdot 2^y + z') \land \exists u\big(\gamma_i(u, x') \land u = 2^{x-y}\big).$$

The disjunction (15) appearing in Specification 1 can be rewritten in a compact way by using this formula. From the **global output** of Specification 1, we obtain the equivalence:

$$\begin{aligned} &\exists x \exists \boldsymbol{x}'\big(\theta(\boldsymbol{x}) \land \varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}') \land (x = x' \cdot 2^y + z')\big) && \text{(i.e. (13))} \\ &\qquad \iff \bigvee_{i \in I} \big(\exists x \exists x' \Psi_i(\boldsymbol{x}, x', z')\big) \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'). && (18) \end{aligned}$$

Following Specification 2, in addition to the ordering $\theta(\boldsymbol{x})$ and the delayed substitution $[x' \cdot 2^y + z'/x]$, Step (ii) takes as input a branch output $(\gamma_i(u, x'), \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'))$ of Step (i), for some $i \in I$. Note that this pair of linear-exponential systems corresponds to a single disjunct of the formula (18). The **global output** of Step (ii) on this input is a set of triples of linear-exponential systems $\{(\chi_{i,j}(y, x', z'), \gamma_{i,j}(x'), \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'))\}_{j \in J_i}$ such that, according to the equivalence (16) in Specification 2, for every $i \in I$,

$$(\exists x \exists x' \Psi_i(\boldsymbol{x}, x', z')) \iff \bigvee_{j \in J_i} (\exists x \theta(\boldsymbol{x})) \land \exists x'\big(\chi_{i,j}(y, x', z') \land \gamma_{i,j}(x')\big). \qquad (19)$$

Consider the combination of Steps (i) and (ii). The output of the two steps combined is given by the following set: $\big\{(\chi_{i,j}, \gamma_{i,j}, \psi_i) : i \in I, j \in J_i\big\}$. From the equivalences in (18) and (19), we obtain the following chain of equivalences:

$$\begin{aligned}
&\exists x \exists \boldsymbol{x}' \big(\theta(\boldsymbol{x}) \land \varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}') \land (x = x' \cdot 2^y + z')\big) \\
&\qquad\iff \bigvee_{i \in I} \big(\exists x \exists x' \Psi_i(\boldsymbol{x}, x', z')\big) \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}') \\
&\qquad\iff \bigvee_{i \in I} \bigvee_{j \in J_i} \big(\exists x \theta(\boldsymbol{x})\big) \land \exists x' \big(\chi_{i,j}(y, x', z') \land \gamma_{i,j}(x')\big) \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'). \qquad (20)
\end{aligned}$$

Following Specification 3, the input of Step (iii) is the ordering $\theta(\boldsymbol{x})$ together with a branch output $(\chi_{i,j}, \gamma_{i,j}, \psi_i)$ of Step (ii), for some $(i, j) \in I \times J_i$. The **global output** of Step (iii) on this input is a set $\{\psi_{i,j,k} \land \psi_i\}_{k \in K_{i,j}}$ of linear-exponential systems. Since, given the specification of Step (i), each ψ_i contains the inequality $0 \le z' < 2^y$, thanks to the formula (17) in Specification 3 we have

$$\exists x' \big(\chi_{i,j}(y, x', z') \land \gamma_{i,j}(x')\big) \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}') \iff \bigvee_{k \in K_{i,j}} \psi_{i,j,k}(y, z') \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'). \qquad (21)$$

Let $\{\psi_\beta\}_{\beta \in B}$ be the **global output** of Algorithm 3, where B is a set of non-deterministic branches. This corresponds to the **global output** of Step (iii), i.e.,

$$\Big\{\psi_\beta \,:\, \beta \in B\Big\} = \Big\{\psi_{i,j,k}(y, z') \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}') \,:\, i \in I, j \in J_i, \text{ and } k \in K_{i,j}\Big\}.$$

(Notice that this means that ψ_β features variables from $(\boldsymbol{x} \setminus x, \boldsymbol{z}')$.) Combining (20) and (21), we obtain the desired equivalence between (13) and (14):

$$\begin{aligned}
&\exists x \exists \boldsymbol{x}' \big(\theta(\boldsymbol{x}) \land \varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}') \land (x = x' \cdot 2^y + z')\big) && \text{(i.e. (13))} \\
\iff &\bigvee_{i \in I} \bigvee_{j \in J_i} \big(\exists x \theta(\boldsymbol{x})\big) \land \exists x' \big(\chi_{i,j}(y, x', z') \land \gamma_{i,j}(x')\big) \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}') \\
\iff &\big(\exists x \theta(\boldsymbol{x})\big) \land \bigvee_{i \in I} \bigvee_{j \in J_i} \bigvee_{k \in K_{i,j}} \big(\psi_{i,j,k}(y, z') \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}')\big) \\
\iff &\big(\exists x \theta(\boldsymbol{x})\big) \land \bigvee_{\beta \in B} \psi_\beta(\boldsymbol{x} \setminus x, \boldsymbol{z}') && \text{(i.e. (14))}.
\end{aligned}$$

To conclude the proof, observe that each ψ_β features a system ψ_i, for some $i \in I$. From the specification of Step (i), each ψ_i contains the inequalities $0 \le z < 2^y$ for every z in $\boldsymbol{z}'$. ◀

To make Lemma 23 unconditional, it now suffices to prove correctness of Steps (i), (ii), and (iii). This is done in the following subsections.

Correctness of Step (i)

The goal of this subsection is to prove that the non-deterministic algorithm that corresponds to lines 1–22 of Algorithm 3 complies with Specification 1.

Our main concern is the **foreach** loop of line 4, which considers sequentially all constraints $(\tau \sim 0)$ of the input quotient system $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$. As discussed in the body of the paper, the goal of this loop is to split each constraint into a "left part" and a "right part" (see, respectively, γ_1 and ψ_1 in the diagram of Figure 2). The left part corresponds to a linear constraint over the quotient variables $\boldsymbol{x}'$ and the auxiliary variable u (which is an alias for 2^{x-y}). The right part is a linear-exponential system over the variables $\boldsymbol{x} \setminus x$ and $\boldsymbol{z}'$. In a nutshell, the split into these two parts is possible because of the three equivalences given in the following two lemmas.

▶ **Lemma 24.** *Let $d, M \in \mathbb{N}$, with $M \geq d \geq 1$. Given $y \in \mathbb{N}$ and $z, w \in \mathbb{Z}$, we have*

$$z \cdot 2^y + w \equiv_d 0 \iff \bigvee_{r=1}^{M} \big(z - r \equiv_d 0 \land r \cdot 2^y + w \equiv_d 0\big).$$

Proof. Informally, this lemma states that z can be replaced with a number in $[1, M]$ congruent to z modulo d. The proof is obvious. ◀

▶ **Lemma 25.** *Let $C, D \in \mathbb{Z}$, with $C \leq D$. For $y \in \mathbb{N}$, $z \in \mathbb{Z}$, and $w \in [C \cdot 2^y, D \cdot 2^y]$, the following equivalences hold*

1. $z \cdot 2^y + w = 0 \iff \bigvee_{r=C}^{D} \big(z + r = 0 \land w = r \cdot 2^y\big)$,
2. $z \cdot 2^y + w \leq 0 \iff \bigvee_{r=C}^{D} \big(z + r \leq 0 \land (r-1) \cdot 2^y < w \leq r \cdot 2^y\big)$,
3. $z \cdot 2^y + w < 0 \iff \bigvee_{r=C}^{D} \big(z + r + 1 \leq 0 \land w = r \cdot 2^y\big) \lor \big(z + r \leq 0 \land (r-1) \cdot 2^y < w < r \cdot 2^y\big)$.

Proof. Firstly, notice that since $w \in [C \cdot 2^y, D \cdot 2^y]$, there is $r^* \in [C, D]$ such that $\lceil \frac{w}{2^y} \rceil = r^*$.

Proof of (1). For the left-to-right direction, note that $z \cdot 2^y + w = 0$ forces w to be divisible by 2^y. Hence $w = r^* \cdot 2^y$, and we have $z \cdot 2^y + r^* \cdot 2^y = 0$, i.e., $z + r^* = 0$. Since r^* belongs to $[C, D]$, the right-hand side is satisfied. The right-to-left direction is trivial.

Proof of (2). Observe that given $r \in [C, D]$ satisfying $(r-1) \cdot 2^y < w \leq r \cdot 2^y$, we have $r = r^*$. Therefore, it suffices to show the equivalence $z \cdot 2^y + w \leq 0 \iff z + r^* \leq 0$. If w is divisible by 2^y, then $w = r^* \cdot 2^y$ and the equivalence easily follows:

$$z \cdot 2^y + w \leq 0 \iff z \cdot 2^y + r^* \cdot 2^y \leq 0 \iff z + r^* \leq 0.$$

Otherwise, when w is not divisible by 2^y, it holds that $\lfloor \frac{w}{2^y} \rfloor = r^* - 1$. Below, given $t \in \mathbb{R}$ we let $\{t\} \coloneqq t - \lfloor t \rfloor$, i.e., $\{t\}$ is the fractional part of t. Observe that $0 \leq \{t\} < 1$. We have:

$$\begin{aligned}
z \cdot 2^y + w \leq 0 &\iff z \cdot 2^y + \left(\left\lfloor \frac{w}{2^y} \right\rfloor + \left\{ \frac{w}{2^y} \right\}\right) \cdot 2^y \leq 0 \\
&\iff z + \left\lfloor \frac{w}{2^y} \right\rfloor + \left\{ \frac{w}{2^y} \right\} \leq 0 \\
&\iff z + \left\lfloor \frac{w}{2^y} \right\rfloor < 0 && \text{since } w \text{ is not divisible by } 2^y \\
&\iff z + r^* - 1 < 0 \\
&\iff z + r^* \leq 0. && (22)
\end{aligned}$$

Proof of (3). While their equivalences look different, this and the previous case have very similar proofs. This similarity stems from the fact that, when w is not divisible by 2^y, then $z \cdot 2^y + w$ cannot be 0, and thus the cases of $z \cdot 2^y + w \leq 0$ and $z \cdot 2^y + w < 0$ become identical. Below, we formalise the full proof for completeness.

Since $w \in [C \cdot 2^y, D \cdot 2^y]$, there must be $r \in [C, D]$ such that either $w = r \cdot 2^y$ or $(r-1) \cdot 2^y < w < r \cdot 2^y$. In both cases, $r = r^*$, and thus to conclude the proof it suffices to establish that:

- $w = r^* \cdot 2^y$ implies $z \cdot 2^y + w < 0 \iff z + r^* + 1 \leq 0$, and
- $(r^* - 1) \cdot 2^y < w < r^* \cdot 2^y$ implies $z \cdot 2^y + w < 0 \iff z + r^* \leq 0$.

The proof of the first item is straightforward. Assuming $w = r^* \cdot 2^y$, we get:

$$z \cdot 2^y + w < 0 \iff z \cdot 2^y + r^* \cdot 2^y < 0 \iff z + r^* < 0 \iff z + r^* + 1 \leq 0.$$

For the second item, assume that $(r^* - 1) \cdot 2^y < w < r^* \cdot 2^y$. In this case w is not divisible by 2^y, and $\lfloor \frac{w}{2^y} \rfloor = r^* - 1$. Hence, $z \cdot 2^y + w$ cannot be 0, which in turn means $z \cdot 2^y + w < 0 \iff z \cdot 2^y + w \leq 0$. Therefore, we can apply the same sequence of equivalences from (22) to show that $z \cdot 2^y + w < 0 \iff z + r^* \leq 0$. ◀

Looking at the pseudocode of Algorithm 3, one can see that the **foreach** loop of line 4 does indeed follow the equivalences in Lemmas 24 and 25. The equivalences in Lemma 25 are applied in lines 8–17, setting $[C, D] = [-\|\rho\|_1, \|\rho\|_1]$. The equivalence in Lemma 24 is applied in lines 19–21, setting $M = \mathit{mod}(\varphi)$. Ultimately, the correctness of Step (i), which we now formalise, follows from these equivalences (and from the correctness of Algorithm 1).

We divide the proof of correctness into the following four steps:

1. We show that the map Δ has no influence in the correctness of the algorithm and can be ignored during the analysis. This is done to simplify the exposition of the next step.
2. We analyse the body of the **foreach** loop of line 4. Here we use Lemmas 24 and 25.
3. We analyse the complete execution of the **foreach** loop, hence obtaining a specification for lines 1–21 of Step (i).
4. We incorporate the call to Algorithm 1 (GaussQE) performed in line 22 into the analysis, proving that Step (i) follows Specification 1.

The map Δ is not needed for correctness. For the correctness of Step (i), the first simplifying step consists in doing a program transformation that removes the uses of the map Δ. This map is introduced exclusively for complexity reasons, and the correctness of the algorithm is preserved if one removes it. More precisely, instead of guessing the integer r in line 9 only once for each least significant part ρ, one can perform one such guess every time ρ is found.

▶ **Lemma 26.** *Consider the code obtained from Step (i) by replacing lines 8–12 with lines 9 and 10. If it complies with Specification 1, then so does Step (i).*

Proof. Suppose that the modified Step (i) complies with Specification 1, which in particular means that its **global output** is a set $\{(\gamma_i(u, x'), \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'))\}_{i \in I}$: set of pairs such that

1. γ_i is a linear system containing the inequality $x' \geq 0$,
2. ψ_i is a linear-exponential system containing inequalities $0 \leq z < 2^y$ for every z in $\boldsymbol{z}'$,

and, over $\mathbb{N}$, the formula $\exists \boldsymbol{q}\big(\theta(\boldsymbol{x}) \land \varphi(\boldsymbol{x}, x', \boldsymbol{q}, \boldsymbol{z}') \land (x = x' \cdot 2^y + z')\big)$ is equivalent to

$$\bigvee_{i \in I} \theta(\boldsymbol{x}) \land (x = x' \cdot 2^y + z') \land \exists u\big(\gamma_i(u, x') \land u = 2^{x-y}\big) \land \psi_i(\boldsymbol{x} \setminus x, \boldsymbol{z}'). \tag{23}$$

Observe that the modification done to the algorithm only influences the number of constraints of the form $(r-1) \cdot 2^y < \rho \land \rho \leq r \cdot 2^y$ that are added to the system ψ, and corresponding to the guesses of $r \in [-\|\rho\|_1, \|\rho\|_1]$. More precisely, when a single ρ is encountered multiple times during the procedure, the modified Step (i) is allowed to guess multiple values for r, whereas the original Step (i) reuses the same r. So, for every pair of systems (γ, ψ) in the **global output** of the original Step (i), there is a system ψ' such that

- (γ, ψ') belongs to the **global output** of the modified Step (i),
- ψ' can be obtained from ψ by duplicating a certain number of times formulae of the form $(r-1) \cdot 2^y < \rho \land \rho \leq r \cdot 2^y$ that already appear in ψ.

Then, clearly, also the **global output** of the original Step (i) satisfies Items 1 and 2 above. It also satisfies the equivalence involving the formula (23). This is because all the systems ψ_i (with $i \in I$) in the **global output** of the modified Step (i) that do not correspond, in the sense we have just discussed, to a ψ in the **global output** of the original Step (i), are unsatisfiable. The reason for their unsatisfiability is that these systems ψ_i feature constraints $(r_1 - 1) \cdot 2^y < \rho \land \rho \leq r_1 \cdot 2^y$ and $(r_2 - 1) \cdot 2^y < \rho \land \rho \leq r_2 \cdot 2^y$ with $r_1 \neq r_2$. As the same term ρ appears in these constraints, their conjunction is unsatisfiable. Thus, the disjunct of formula (23) corresponding to such a ψ_i can be dropped without changing the truth of the equivalence, and we conclude that the original Step (i) complies with Specification 1. ◀

▶ Remark 27. Following Lemma 26, for the remaining part of the proof of correctness of Step (i) we assume this step to only feature lines 9 and 10 in place of lines 8–12.

Analysis of the body of the foreach loop (of line 4). For the rest of Appendix C.2, we simply say *"the **foreach** loop"* without referring to its line number, as there are no other loops in Algorithm 3. We start the analysis by considering a single iteration of this loop. Given an input $(\theta, \varphi, [x' \cdot 2^y + z'/x])$ of Algorithm 3 (which corresponds to the input of Step (i), see Specification 1), and a constraint $(\tau \sim 0)$ from φ, below we say that *executing the **foreach** body on the state $(\tau \sim 0, \gamma, \psi)$ yields as a **global output** a set S* whenever:

- $\tau \sim 0$ is the constraint selected in line 4, and γ and ψ are the systems stored in the homonymous variables when $\tau \sim 0$ is selected (these systems are initially $\top$, see line 2).
- S is the union over all non-deterministic branches of the pairs of systems (γ', ψ') stored in the variables (γ, ψ) after the **foreach** loop completes its iteration on $\tau \sim 0$ (i.e., the body of the loop is executed exactly once, and the program reaches line 4 again).

The following lemma describes the effects of one iteration of the **foreach** loop.

▶ **Lemma 28.** *Let $(\theta, \varphi, [x' \cdot 2^y + z'/x])$ be an input of Step (i) described in Specification 1. Let u be the fresh variable defined in line 1, and let $(\tau \sim 0)$ be a constraint from φ. Executing the **foreach** body on the state $(\tau \sim 0, \gamma, \psi)$ yields as a **global output** a set of pairs $\{(\gamma \land \gamma_r, \psi \land \psi_r)\}_{r \in R}$, for some finite set of indices R, such that*

1. *γ_r is a linear (in)equality or divisibility constraint over the variables $\boldsymbol{x}'$ and u.*
2. *ψ_r is a linear-exponential constraint over the variables $\boldsymbol{x} \setminus x$ and $\boldsymbol{z}'$.*
3. *If τ only features variables from $\boldsymbol{x} \setminus x$ and $\boldsymbol{z}'$, then $R = \{0\}$, and $\gamma_0 = \top$ and $\psi_0 = (\tau \sim 0)$.*
4. *Over $\mathbb{N}$, $\theta \land \bigwedge_{z \in \boldsymbol{z}'}(0 \leq z < 2^y) \implies \big((\tau \sim 0) \iff \bigvee_{r \in R}(\gamma_r[2^{x-y}/u] \land \psi_r)\big)$.*

Proof. In every constraint $(\tau \sim 0)$ of the system φ, the term τ is a quotient term induced by θ, and $\sim$ is a predicate symbol from the set $\{<, \leq, =, \equiv_d : d \geq 1\}$. Line 5 "unpacks" the term τ, according to the definition of quotient term given in Section 5, as

$$a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + \rho(\boldsymbol{x} \setminus x, \boldsymbol{z}'), \tag{24}$$

where 2^x is the leading exponential term of the ordering θ and 2^y is its successor in this ordering (observe that this agrees with the delayed substitution $[x' \cdot 2^y + z'/x]$). In the expression in (24), a is an integer, $f(\boldsymbol{x}')$ is a linear term over the quotient variables $\boldsymbol{x}'$, and ρ is the least significant part of τ. The latter means that ρ is of the form

$$b \cdot y + \sum\nolimits_{i=1}^{\ell} \Big(a_i \cdot x_i + c_i \cdot (x_i \bmod 2^y) + \sum\nolimits_{j=1}^{m} \big(b_j \cdot 2^{x_j} + c_{i,j} \cdot (x_i \bmod 2^{x_j})\big)\Big) + d, \tag{25}$$

where the coefficients $b, a_i, c_i, b_j, c_{i,j}$ and the constant d are all integers; $m \leq \ell$, the variables $x_1, \dots, x_m$ are from $\boldsymbol{x} \setminus \{x, y\}$, and the variables $x_{m+1}, \dots, x_\ell$ are from $\boldsymbol{z}'$. (The notation $\boldsymbol{x} \setminus \{x, y\}$ is short for $(\boldsymbol{x} \setminus x) \setminus y$.) Finally, since φ is a quotient system induced by θ, it features the inequalities $0 \leq z < 2^y$ for every z in $\boldsymbol{z}'$.

We divide the proof into three cases, following which of the three branches of the chain of **if-else** statements starting in line 6 triggers.

The guard of the if statement in line 6 triggers. In this case, $(\tau \sim 0)$ only features variables from $\boldsymbol{x} \setminus x$ and $\boldsymbol{z}'$, and the iteration of the **foreach** loop on $\tau \sim 0$ completes yielding as a **global output** a set with only one pair of systems: $(\gamma, \psi \land (\tau \sim 0))$. Properties 1–4 in the statement are trivially satisfied, and the lemma is proven.

The else-if statement in line 7 triggers. In this case, $\sim$ is a symbol from $\{=, \leq, <\}$ and with respect to the expression in (24), either $a \neq 0$ or $f(\boldsymbol{x}')$ is not an integer. Notice that then Property 3 trivially holds, as the antecedent of the implication in this property is false. Below we focus on Properties 1, 2 and 4.

We remind the reader that, following Remark 27, we are considering the version of Step (i) featuring lines 9 and 10 in place of lines 8–12. Therefore, in this case the iteration of the **foreach** loop executes lines 9, 10 and 13–17.

Observe that, under the assumption that $\theta(\boldsymbol{x}) \land \bigwedge_{z \in \boldsymbol{z}'}(0 \leq z < 2^y)$ holds, in the expression in (25) all variables x_i and terms 2^{x_j}, with $i \in [1, \ell]$ and $j \in [1, m]$, are bounded by 2^y, which in turns implies $\rho \in \big[-\|\rho\|_1 \cdot 2^y, \|\rho\|_1 \cdot 2^y\big]$. We thus derive the following implication:

$$\theta \land \bigwedge_{z \in \boldsymbol{z}'} (0 \leq z < 2^y) \implies -\|\rho\|_1 \cdot 2^y \leq \rho \leq \|\rho\|_1 \cdot 2^y. \tag{26}$$

Algorithm 3 takes advantage of this implication to estimate the least significant part ρ. In line 9, it guesses an integer $r \in \big[-\|\rho\|_1, \|\rho\|_1\big]$, and in line 10, it adds to ψ the formula

$$\psi'_r \coloneqq ((r-1) \cdot 2^y < \rho) \land (\rho \leq r \cdot 2^y). \tag{27}$$

Essentially, in adding ψ'_r to ψ, the algorithm is guessing that $r = \big\lceil \frac{\rho}{2^y} \big\rceil$.

We now inspect lines 13–17, carefully analysing the three cases of $\sim \in \{=, \leq, <\}$ separately.

case: $=$**.** Let $R \coloneqq \big[-\|\rho\|_1, \|\rho\|_1\big]$. Given $r \in R$, let us define

$$\gamma_r \coloneqq (a \cdot u + f(\boldsymbol{x}') + r = 0),$$
$$\psi_r \coloneqq \psi'_r \land (r \cdot 2^y = \rho).$$

Following lines 10, 16 and 17, we deduce that executing the **foreach** body on the state $(\tau \sim 0, \gamma, \psi)$ yields as a **global output** the set of pairs $\{(\gamma \land \gamma_r, \psi \land \psi_r)\}_{r \in R}$. Properties 1, 2 and 4 are easily seen to be satisfied:

- Properties 1 and 2 trivially follow from the definitions of γ_r and ψ_r.
- Observe that the expression in (24) can be rewritten as $(a \cdot 2^{x-y} + f(\boldsymbol{x}')) \cdot 2^y + \rho(\boldsymbol{x} \setminus x, \boldsymbol{z}')$. From the formula (26) together with Equivalence 1 from Lemma 25, we obtain

$$\theta \land \bigwedge_{z \in \boldsymbol{z}'} (0 \leq z < 2^y) \implies \Big(\tau = 0 \iff \bigvee_{r=-\|\rho\|_1}^{\|\rho\|_1} (a \cdot 2^{x-y} + f(\boldsymbol{x}') + r = 0 \land \rho = r \cdot 2^y)\Big).$$

  The subformula $a \cdot 2^{x-y} + f(\boldsymbol{x}') + r = 0$ is equal to $\gamma_r[2^{x-y} \,/\, u]$. The subformula $\rho = r \cdot 2^y$ is equivalent to ψ_r. We thus have

$$\theta \land \bigwedge_{z \in \boldsymbol{z}'} (0 \leq z < 2^y) \implies \Big(\tau = 0 \iff \bigvee_{r \in R} (\gamma_r[2^{x-y}/u] \land \psi_r)\Big),$$

  that is, Property 4 holds.

case: $\leq$**.** Let $R \coloneqq \big[-\|\rho\|_1, \|\rho\|_1\big]$. Given $r \in R$, let us define

$$\gamma_r \coloneqq (a \cdot u + f(\boldsymbol{x}') + r \leq 0),$$
$$\psi_r \coloneqq \psi'_r.$$

Following lines 10 and 16, we deduce that executing the **foreach** body on the state $(\tau \sim 0, \gamma, \psi)$ yields as a **global output** the set of pairs $\{(\gamma \land \gamma_r, \psi \land \psi_r)\}_{r \in R}$. The proof that Properties 1, 2 and 4 are satisfied follows as in the previous case (relying on Equivalence 2 from Lemma 25 to prove Property 4).

case: **$<$.** Let $R \coloneqq \big[-\|\rho\|_1, \|\rho\|_1\big] \times \{=,<\}$. Given $r \in R$, we define

$$\begin{aligned}
\gamma_{(r,=)} &\coloneqq (a \cdot u + f(\boldsymbol{x'}) + r + 1 \leq 0),\\
\psi_{(r,=)} &\coloneqq \psi'_r \wedge (\rho = r \cdot 2^y),\\
\gamma_{(r,<)} &\coloneqq (a \cdot u + f(\boldsymbol{x'}) + r \leq 0),\\
\psi_{(r,<)} &\coloneqq \psi'_r \wedge (\rho < r \cdot 2^y).
\end{aligned}$$

Following lines 10 and 13–16, we deduce that executing **foreach** body on the state $(\tau \sim 0, \gamma, \psi)$ yields as a **global output** the set of pairs $\{(\gamma \wedge \gamma_{(r,\sim')}, \psi \wedge \psi_{(r,\sim')})\}_{(r,\sim')\in R}$. The proof of Properties 1 and 2 is trivial. For the proof of Property 4, from the formula (26) and Equivalence 3 from Lemma 25 we have

$$\begin{aligned}
\theta \wedge \bigwedge_{z \in \boldsymbol{z'}} (0 \leq z < 2^y) \implies \Big(\tau < 0 \iff &\\
&\bigvee_{r=-\|\rho\|_1}^{\|\rho\|_1} \big(a \cdot 2^{x-y} + f(\boldsymbol{x'}) + r + 1 \leq 0 \wedge \rho = r \cdot 2^y\big)\\
&\vee \bigvee_{r=-\|\rho\|_1}^{\|\rho\|_1} \big(a \cdot 2^{x-y} + f(\boldsymbol{x'}) + r \leq 0 \wedge (r-1) \cdot 2^y < \rho < r \cdot 2^y\big)\Big).
\end{aligned}$$

The subformulae $a \cdot 2^{x-y} + f(\boldsymbol{x'}) + r + 1 \leq 0$ and $a \cdot 2^{x-y} + f(\boldsymbol{x'}) + r \leq 0$ are equal to $\gamma_{(r,=)}[2^{x-y} \,/\, u]$ and $\gamma_{(r,<)}[2^{x-y} \,/\, u]$, respectively. The subformulae $\rho = r \cdot 2^y$ and $(r-1) \cdot 2^y < \rho < r \cdot 2^y$ are equivalent to $\psi_{(r,=)}$ and $\psi_{(r,<)}$, respectively. We thus have

$$\theta \wedge \bigwedge_{z \in \boldsymbol{z'}} (0 \leq z < 2^y) \implies \Big(\tau < 0 \iff \bigvee_{(r,\sim') \in R} \gamma_{(r,\sim')}[2^{x-y} \,/\, u] \wedge \psi_{(r,\sim')}\Big),$$

that is, Property 4 holds.

The else statement of in line 18 triggers. In this case, $\sim$ is $\equiv_d$ for some $d \geq 1$, and the algorithm executes lines 19–21. Let $R \coloneqq \big[1, \mathit{mod}(\varphi)\big]$. In line 19, it guesses an integer $r \in R$. Recall that $\mathit{mod}(\varphi)$ is the least common multiple of all divisors appearing in divisibility constraints of the system φ, and therefore $d \leq \mathit{mod}(\varphi)$. For an integer $r \in R$, we define

$$\begin{aligned}
\gamma_r &\coloneqq (a \cdot u + f(\boldsymbol{x'}) - r \equiv_d 0), && \text{(see line 20)}\\
\psi_r &\coloneqq (r \cdot 2^y + \rho \equiv_d 0). && \text{(see line 21)}
\end{aligned}$$

Following lines 20 and 21, we deduce that executing **foreach** body on the state $(\tau \sim 0, \gamma, \psi)$ yields as a **global output** the set of pairs $\{(\gamma \wedge \gamma_r, \psi \wedge \psi_r)\}_{r \in R}$. Properties 1–4 are again satisfied:

- Properties 1 and 2 follow directly by definition of γ_r and ψ_r.
- Property 3 is true, as the antecedent of the implication in this property is false (as in the previous case, we have either $a \neq 0$ or $f(\boldsymbol{x'})$ non-constant).
- Property 4 follows by Lemma 24, since $d \leq \mathit{mod}(\varphi)$ and θ implies $(x - y) \in \mathbb{N}$. ◀

Analysis of the complete execution of the foreach loop. We extend the analysis performed in Lemma 28 to multiple iterations of the body of the **foreach** loop. We define the **global output** of the **foreach** loop of line 4 to be the set of all pairs (γ, ψ) where γ and ψ are the systems stored in the homonymous variables when, in a non-deterministic branch of the program, line 22 is reached (and before this line is executed). We prove the following lemma.

▶ **Lemma 29.** *Let $(\theta(\boldsymbol{x}), \varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}'), [x' \cdot 2^y + z'/x])$ be an input of Step (i), as described in Specification 1. Let u be the fresh variable defined in line 1. Executing the **foreach** loop of line 4 on this input yields as a **global output** a set of pairs $\{(\gamma_i, \psi_i)\}_{i \in I}$ such that*

A. *γ_i is a linear system over the variables $\boldsymbol{x}'$ and u.*

B. *ψ_i is a linear-exponential system over the variables $\boldsymbol{x} \setminus x$ and $\boldsymbol{z}'$. Moreover, ψ_i contains inequalities $0 \leq z < 2^y$, for every z in $\boldsymbol{z}'$.*

C. *Over $\mathbb{N}$, $\theta \land \varphi$ is equivalent to $\theta \land \exists u\big(u = 2^{x-y} \land \bigvee_{i \in I}(\gamma_i \land \psi_i)\big)$.*

Proof. This lemma follows by first applying Lemma 28, and then arguing that every formula ψ_i contains the inequalities $0 \leq z < 2^y$, for every z in $\boldsymbol{z}'$. Roughly speaking, this allows to remove the hypothesis $\bigwedge_{z \in \boldsymbol{z}'}(0 \leq z < 2^y)$ from Property 4 in Lemma 28, resulting in the equivalence required by Property C.

Let us formalise the above sketch. For simplicity, let $\varphi = \bigwedge_{i=1}^m \tau_i \sim_i 0$, and assume that the guard of the **foreach** loop selects the constraints in φ in the order $\tau_1 \sim_1 0, \tau_2 \sim_2 0, \dots$ We denote by $\gamma^{(k)}$ and $\psi^{(k)}$ two systems that, in a single non-deterministic branch of the program that executes the body of the **foreach** loop exactly k times, are stored in the variables γ and ψ, respectively; and denote by $\{(\gamma_t^{(k)}, \psi_t^{(k)})\}_{t \in T_k}$ the set of all such pairs of systems, across all non-deterministic branches. Note that, from line 2, we have $\gamma^{(0)} = \psi^{(0)} = \top$ (and T_0 contains a single index). Lastly, let $\varphi^{(k)} := \bigwedge_{i=k+1}^m \tau_i \sim_i 0$ (hence, $\varphi^{(m)} = \top$).

By relying on Lemma 28, we conclude that the following is an invariant for the **foreach** loop: for every $k \in [0, m]$ and $t \in T_k$,

(i) $\gamma_t^{(k)}$ is a linear (in)equality or divisibility constraints over the variables $\boldsymbol{x}'$ and u,

(ii) $\psi_t^{(k)}$ is a linear-exponential constraint over the variables $\boldsymbol{x} \setminus x$ and $\boldsymbol{z}'$,

(iii) if $\tau_k \sim_k 0$ is $0 \leq z$ (resp. $z < 2^y$) for some z in $\boldsymbol{z}'$, then $\psi_t^{(k)}$ contains $0 \leq z$ (resp. $z < 2^y$). Here, recall that $0 \leq z$ and $z < 2^y$ are shorthands for $-z \leq 0$ and $z - 2^y < 0$, respectively.

(iv) Over $\mathbb{N}$, $\theta \land \bigwedge_{z \in \boldsymbol{z}'}(0 \leq z < 2^y) \implies \big(\varphi \iff \bigvee_{t \in T_k}(\varphi^{(k)} \land \gamma_t^{(k)}[2^{x-y}/u] \land \psi_t^{(k)})\big)$.

Consider the case of $k = m$. Since φ is a quotient system induced by θ, it contains $\bigwedge_{z \in \boldsymbol{z}'}(0 \leq z < 2^y)$ as a subsystem. Hence, from Item (iii) of the invariant, for every $t \in T_m$, $\psi_t^{(m)}$ contains $\bigwedge_{z \in \boldsymbol{z}'}(0 \leq z < 2^y)$ as a subsystem. Together with Items (i) and (ii), this shows that Properties A and B hold. By Item (iv), we also have the following equivalence:

$$(\theta \land \varphi) \iff \big(\theta \land \bigvee_{t \in T_k}(\varphi^{(m)} \land \gamma_t^{(m)}[2^{x-y}/u] \land \psi_t^{(m)})\big).$$

Above, $\varphi^{(m)} = \top$, and $\gamma_t^{(m)}[2^{x-y}/u]$ is equivalent to $\exists u(u = 2^{x-y} \land \gamma_t^m)$. Since θ implies $x \geq y$, the variable u can be existentially quantified over $\mathbb{N}$. Hence, Property C holds. ◀

Incorporating the call to GaussQE and completing the analysis of Step (i). By chaining Lemma 29 and Lemma 19, we are now able to prove the correctness of Step (i).

▶ **Lemma 30.** *Step (i) of Algorithm 3 complies with Specification 1.*

Proof. The input of this step corresponds to the input of Algorithm 3, that is a triple $(\theta(\boldsymbol{x}), \varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}'), [x' \cdot 2^y + z'/x])$ satisfying the properties described in Specification 1.

By Lemma 29, the **global output** of the **foreach** loop of line 4 corresponding to this input is a set of pairs of systems $\{(\gamma_j, \psi_j)\}_{j \in J}$ satisfying Properties A–C. Hence, every formula γ_j is a linear system over the variables $\boldsymbol{x}'$ and u, where u is the fresh variable defined in line 1. Let us fix some $j \in J$, and consider the non-deterministic branch in which the **foreach** loop produces the pair (γ_j, ψ_j). In line 22, the algorithm calls Algorithm 1 (GaussQE) to remove the variables $\boldsymbol{q} \coloneqq \boldsymbol{x}' \setminus x'$ from the formula $\gamma_j \land \boldsymbol{x}' \geq 0$.

By Lemma 19, the **global output** of Algorithm 1 on input $(\boldsymbol{q}, \gamma_j \land \boldsymbol{x}' \geq 0)$ is a set of linear systems $\{\gamma_{j,k}(u, x')\}_{k \in K_j}$ such that $\bigvee_{k \in K_j} \gamma_{j,k}(u, x')$ is equivalent to $\exists \boldsymbol{q}(\gamma_j \land \boldsymbol{x}' \geq 0)$ over $\mathbb{Z}$. Because of the inequalities $\boldsymbol{x}' \geq 0$, the quantification $\exists \boldsymbol{q}$ can be restricted to the non-negative integers, and therefore we conclude that, over $\mathbb{N}$,

$$\exists \boldsymbol{q} \gamma_j \iff \bigvee_{k \in K_j} \gamma_{j,k}(u, x'). \tag{28}$$

The **global output** of Step (i) is the set:

$$\big\{(\gamma_i, \psi_i) \,:\, i \in I\big\} \coloneqq \big\{(\gamma_{j,k}, \psi_j) \,:\, j \in J, k \in K_j\big\}.$$

We show that this set satisfies the requirements of Specification 1.

- Obviously, every γ_i a linear system in variables u, and x', as it corresponds to some $\gamma_{j,k}$. This system contains the inequality $x' \geq 0$. Indeed, this inequality is present in the system $\gamma_j \land \boldsymbol{x}' \geq 0$ in input of Algorithm 1. Looking at its pseudocode, observe that Algorithm 1 leaves untouched all inequalities that do not feature variables that are to be eliminated. Hence, since x' is not among the eliminated variables $\boldsymbol{q}$, the output formula $\gamma_{j,k}$ contains $x' \geq 0$.
- By Property B of Lemma 29, every ψ_i is a linear-exponential system containing the inequalities $0 \leq z < 2^y$, for every z in $\boldsymbol{z}'$.
- By Property C of Lemma 29 and the equivalence (28), over $\mathbb{N}$ we have,

$$\begin{aligned}
& \exists \boldsymbol{q}\big(\theta \land \varphi \land x = x' \cdot 2^y + z'\big) \\
\iff & \exists \boldsymbol{q}\big(\theta \land \exists u\big(u = 2^{x-y} \land \bigvee_{j \in J} (\gamma_j \land \psi_j)\big) \land x = x' \cdot 2^y + z'\big) && \text{by Property C} \\
\iff & \bigvee_{j \in J} \big(\theta \land \exists u\big(u = 2^{x-y} \land (\exists \boldsymbol{q} \gamma_j) \land \psi_j\big) \land x = x' \cdot 2^y + z'\big) && \boldsymbol{q} \text{ only occurs in } \gamma_j \\
\iff & \bigvee_{i \in I} \big(\theta \land \exists u\big(u = 2^{x-y} \land \gamma_i \land \psi_i\big) \land x = x' \cdot 2^y + z'\big) && \text{by def. of } I \text{ and (28)} \\
\iff & \bigvee_{i \in I} \big(\theta \land \exists u\big(u = 2^{x-y} \land \gamma_i\big) \land \psi_i \land x = x' \cdot 2^y + z'\big) && u \text{ does not occur in } \psi_i. \blacktriangleleft
\end{aligned}$$

Correctness of Step (ii)

▶ **Lemma 31.** *Step (ii) of Algorithm 3 complies with Specification 2.*

Proof. To prove the lemma, we analyse Step (ii). This step takes as input the ordering $\theta(\boldsymbol{x})$ and the delayed substitution $[x' \cdot 2^y + z'/x]$ that are part of the input of Algorithm 3 (where x and y in the delayed substitution are the largest and second-to-largest variables with respect to θ), together with a branch output of Step (i). By Lemma 30 and according to Specification 1, the latter is a pair (γ, ψ) where $\gamma(u, x')$ is a linear system and $\psi(\boldsymbol{x} \setminus x, \boldsymbol{z}')$ is a linear-exponential system.

Step (ii) starts with the replacement of the auxiliary variable u. However, in line 23 we do not only perform the replacement of u with 2^{x-y}, but immediately replace $(x-y)$ with u. That is, the formula $\exists u(\gamma(u,x') \land u = 2^{x-y})$ appearing in Ψ from Specification 2 is updated into the *equivalent* formula $\exists u(\gamma(2^u,x') \land u = x-y)$. The system $\gamma(2^u,x')$ is a (u,x')-primitive linear-exponential system.

After this "change of alias" for the variable u, the algorithm proceeds with linearising its occurrences in $\gamma(2^u,x')$. This is done by invoking Algorithm 4 (SolvePrimitive) on input $(u,x',\gamma(2^u,x'))$; see line 24. By correctness of Algorithm 4 (Lemma 21), its **global output** is a set of pairs $\{(\hat{\chi}_j(u),\gamma_j(x'))\}_{j\in J}$, where every $\hat{\chi}_j$ and γ_j is a linear system, such that

$$(u \geq x' \geq 0) \implies \Big(\gamma(2^u,x') \iff \bigvee_{j\in J} \hat{\chi}_j(u) \land \gamma_j(x')\Big). \tag{29}$$

To use the double implication of (29), we show next that $\theta(\boldsymbol{x}) \land (x = x' \cdot 2^y + z') \land (u = x - y)$ entails $u \geq x'$ (recall that x' ranges over $\mathbb{N}$).

When $x' = 0$, the inequality $u \geq x'$ follows from the inequality $2^x \geq 2^y$ appearing in the ordering $\theta(\boldsymbol{x})$. If x' is positive, then we have

$$\begin{aligned} u = x - y &= x' \cdot 2^y + z' - y && \text{delayed substitution} \\ &\geq x' \cdot (y+1) + z' - y && \text{since } (2^y \geq y+1) \text{ for every } y \in \mathbb{N} \\ &= y \cdot (x'-1) + x' + z' \geq x'. && \text{since } x' \geq 1. \end{aligned}$$

Therefore, from the formula (29) we obtain the equivalence

$$\Psi(\boldsymbol{x},x'z') \iff \theta(\boldsymbol{x}) \land (x = x' \cdot 2^y + z') \land \exists u\big(\big(\bigvee_{j\in J} \hat{\chi}_j(u) \land \gamma_j(x')\big) \land (u = x-y)\big) \tag{30}$$

Observe that, from the specification of Algorithm 4, every system $\hat{\chi}_j(u)$ has a very simple form: it is either an equality $(u = a)$ or a conjunction $(u \geq b) \land (d \mid u - r)$, for some $a,b,d,r \in \mathbb{N}$, with $b > 2$. We will use this fact twice in the remaining part of the proof.

In line 25, the algorithm performs on $\hat{\chi}_j(u)$ the substitutions $[x-y\,/\,u]$ and $[x' \cdot 2^y + z'/x]$, in this order. For the moment, let us focus on the effects of the first substitution. Because of the form of $\hat{\chi}_j$, the system $\hat{\chi}_j[x-y\,/\,u]$ entails $x \geq y$. This allows us to exclude $2^x \geq 2^y$ from the ordering $\theta(\boldsymbol{x})$; or alternatively quantifying it away as $(\exists x\theta(\boldsymbol{x}))$. Starting from the equivalence (30), we thus obtain

$$\Psi(\boldsymbol{x},x'z') \iff (\exists x\theta(\boldsymbol{x})) \land (x = x' \cdot 2^y + z') \land \bigvee_{j\in J} \hat{\chi}_j[x-y\,/\,u] \land \gamma_j(x'), \tag{31}$$

where we highlight the fact that x still occurs free in both sides of the equivalence.

We now consider the application of the second substitution $[x' \cdot 2^y + z'/x]$. For every $j \in J$, define $\chi_j(y,x',z') \coloneqq \hat{\chi}_j[x-y\,/\,u][x' \cdot 2^y + z'/x]$, i.e., the system $\hat{\chi}_j(x' \cdot 2^y + z' - y)$. As Step (ii) ends in line 25, its **global output** is the set of triples $\{(\chi_j(y,x',z'),\gamma_j(x'),\psi(\boldsymbol{x} \setminus x,\boldsymbol{z}'))\}_{j\in J}$. It is easy to see that this set satisfies Items 1–3 in Specification 2:

- Item 1 follows from the form of $\hat{\chi}_j(u)$ and the substitutions applied to it.
- The first statement of Item 2 follows from the specification of Algorithm 4. For the second statement, observe that by Lemma 30 and according to Specification 1, the formula $\gamma(2^u,x')$ contains the inequality $x' \geq 0$. Let us study the evolution of this inequality through Algorithm 4. In line 1, this inequality is part of the formula ψ. Then, following the updates in lines 7 and 15, $x' \geq 0$ appears in the formula γ in output of Algorithm 4. Therefore, for every $j \in J$, $x' \geq 0$ appears in γ_j, as required.

- Item 3 is direct from the fact that Step (ii) does not manipulate ψ.

Lastly, from the equivalence (31) and the definition of χ_j, we establish the equivalence (16) in the specification:

$$\begin{aligned}
&\exists x \Psi(\boldsymbol{x}, x' z') \\
\iff\ &\exists x \big((\exists x \theta(\boldsymbol{x})) \land (x = x' \cdot 2^y + z') \land \bigvee_{j \in J} \hat{\chi}_j[x - y \,/\, u] \land \gamma_j(x')\big) \\
\iff\ &\bigvee_{j \in J} (\exists x \theta(\boldsymbol{x})) \land \chi_j(y, x', z') \land \gamma_j(x').
\end{aligned}$$

◀

Correctness of Step (iii)

We start by giving a high-level overview of Step (iii). As discussed in the body of the paper, this step can be seen as a simplified version of Step (i). Recall that Step (i) manipulates the linear-exponential system $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$ from the input of Algorithm 3. Step (iii) manipulates instead the formula $\chi(y, x', z')$ that is part of the output of Step (ii). The similarity between these two manipulations is reflected in the diagram from Figure 2. The systems γ and ψ in input of Step (iii) are denoted in the diagram by γ_3 and ψ_1, respectively. As shown in the diagram, within Step (iii) the system $\chi(y, x', z')$, which is in fact a quotient system induced by the ordering $\theta(\boldsymbol{x})$, is (non-deterministically) split into its most significant part $\gamma_4(x')$ and least significant part $\psi_2(y, z')$. The former is conjoined with the formula $\gamma_3(x')$. Since both systems are linear systems with a single variable x', by calling Algorithm 1 (GaussQE) we can check whether $\gamma_3(x') \land \gamma_4(x')$ is satisfiable over $\mathbb{N}$. If the answer is negative, the computations in this non-deterministic branch do not contribute to the global output. (We recall in passing that an empty disjunction is equivalent to the formula $\bot$. So, if the **global output** of Step (iii) is the empty set, its input corresponds to an unsatisfiable formula.) If the answer is positive, the step returns the linear-exponential system $\psi_1(\boldsymbol{x} \setminus x, \boldsymbol{z}') \land \psi_2(y, z')$.

► **Lemma 32.** *Step (iii) of Algorithm 3 complies with Specification 3.*

Proof. This step takes as input the ordering $\theta(\boldsymbol{x})$ that is part of the input of Algorithm 3, together with a branch output of Step (ii). By Lemma 31 and according to Specification 2, the latter is a triple $(\chi(y, x', z'), \gamma(x'), \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}'))$ satisfying Items 1–3 from Specification 2.

To prove the lemma, consider first lines 26–34 (that is, Step (iii) without the call to Algorithm 1 performed in line 35). Define the **global output** of these lines as the union over all non-deterministic branches of the pairs (γ, ψ), where γ and ψ are the contents of the homonymous variables when line 35 is reached (before this line is executed). Let us for the moment assume the following result:

▷ Claim 33. Let $(\chi(y, x', z'), \gamma(x'), \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}'))$ be an input of Step (iii), as described in Specification 3. Then, the **global output** of lines 26–34 is a set of pairs $\{(\gamma_j \land \gamma, \psi_j \land \psi)\}_{j \in J}$ such that, for every $j \in J$ we have

A. γ_j is a linear system in the single variable x'.

B. ψ_j is a linear-exponential system over the variables (y, z').

C. Over $\mathbb{N}$, $(0 \leq z' < 2^y)$ entails that $\chi(y, x', z')$ is equivalent to $\bigvee_{j \in J} \gamma_j(x') \land \psi_j(y, z')$.

With the above claim at hand, it is simple to complete the proof. Let $K \subseteq J$ be the subset of indices $j \in J$ such that $\gamma_j \land \gamma$ is satisfiable (over $\mathbb{N}$). Observe that, by Item A and Lemma 31, $\gamma_j \land \gamma$ is a linear system, and moreover $\gamma(x')$ contains the inequality $x' \geq 0$ (this is ensured by Specification 2). Therefore, despite working over $\mathbb{Z}$, Algorithm 1 can be used to perform this satisfiability check over $\mathbb{N}$ (as it is done in line 35).

The **global output** of Step (iii) is the set $\{\psi_k(y, z') \wedge \psi(\boldsymbol{x} \setminus x, \boldsymbol{z}') : k \in K\}$. By Item B and Specification 2, this is a set of linear-exponential systems, as required by Specification 3. Lastly, formula (17) follows from the fact that $0 \leq z' < 2^y$ entails

$$\begin{aligned}
\exists x'(\chi(y, x', z') \wedge \gamma(x')) &\iff \exists x' \bigvee_{j \in J} \gamma_j(x') \wedge \psi_j(y, z') \wedge \gamma(x') && \text{by Item C} \\
&\iff \bigvee_{j \in J} \psi_j(y, z') \wedge \exists x'\big(\gamma_j(x') \wedge \gamma(x')\big) \\
&\iff \bigvee_{k \in K} \psi_k(y, z') && \text{by def. of } K.
\end{aligned}$$

This completes the proof Lemma 32, subject to the proof of Claim 33 which is given below. ◀

Proof of Claim 33. The quotient system $\chi(y, x', z')$ induced by the ordering θ may have one of the following two forms, which are handled in different lines of Step (iii):

(a) lines 27–29: $(x' \cdot 2^y + z' - y - c = 0)$ for $c \in \mathbb{N}$; or

(b) lines 31–34: $(-x' \cdot 2^y - z' + y + c \leq 0) \wedge (d \mid x' \cdot 2^y + z' - y - r)$ for $c, d, r \in \mathbb{N}$ and $c \geq 3$.

Below, we consider the cases (a) and (b) separately. Beforehand, we estimate the term $(-z' + y + c)$ assuming that the variables z' and y are such that $(0 \leq z' < 2^y)$. We have:

$$\begin{aligned}
-2^y < -z' &\leq (-z' + y + c) && \text{since } y, c \geq 0 \\
&\leq y + c < (1 + c) \cdot 2^y.
\end{aligned}$$

Therefore, under the hypothesis that $0 \leq z' < 2^y$, we have $(-z' + y + c) \in [0 \cdot 2^y, c \cdot 2^y]$, which allows us to handle cases (a) and (b) by relying on Lemma 25.

Case (a). For $b \in [0, c]$, define $\gamma_b \coloneqq (x' = b)$ and $\psi_b \coloneqq (b \cdot 2^y = -z' + y + c)$. Following the guess done in line 27, the **global output** of lines 26–34 is the set of pairs $\{(\gamma_b \wedge \gamma, \psi_b \wedge \psi)\}_{b \in [0,c]}$. Then, Items A and B of the claim are obviously satisfied. By Lemma 25, we have

$$(0 \leq z' < 2^y) \implies (\chi \iff \textstyle\bigvee_{b=0}^{c} (\gamma_b \wedge \psi_b)),$$

which shows Item C.

Case (b). In this case χ is a conjunction of an inequality $(-x' \cdot 2^y - z' + y + c \leq 0)$ and a divisibility constraint $(d \mid x' \cdot 2^y + z' - y - r)$. Given $(b, g) \in [0, c] \times [1, d]$, define

$$\begin{aligned}
&\gamma_{(b,g)} \coloneqq x' \geq b \wedge (d \mid x' - g), \\
&\psi_{(b,g)} \coloneqq ((b - 1) \cdot 2^y < -z' + y + c) \wedge (-z' + y + c \leq b \cdot 2^y) \wedge (d \mid g \cdot 2^y + z' - y - r).
\end{aligned}$$

Following the guess done by the algorithm in line 32, the **global output** of lines 26–34 is the set of pairs $\{(\gamma_K \wedge \gamma, \psi_K \wedge \psi)\}_{K \in [0,c] \times [1,d]}$. Then, Items A and B of the claim are obviously satisfied. By Lemma 24 we have

$$(d \mid x' \cdot 2^y + z' - y - r) \iff \bigvee_{g=1}^{d} (d \mid x' - g) \wedge (d \mid g \cdot 2^y + z' - y - r),$$

and by Lemma 25, we have

$$(0 \leq z' < 2^y) \implies \Big(-x' \cdot 2^y - z' + y + c \leq 0 \iff \bigvee_{b \in [0,c]} (x' \geq b) \wedge ((b - 1) \cdot 2^y < -z' + y + c \leq b \cdot 2^y) \Big).$$

Therefore, we conclude that

$$(0 \leq z' < 2^y) \implies \Big(\chi \iff \textstyle\bigvee_{K \in [0,c] \times [1,d]} (\gamma_K \land \psi_K)\Big),$$

which shows Item C, completing the proof of the claim. ◁

The correctness of ElimMaxVar now follows by combining Lemmas 30–32 with Lemma 23.

▶ **Lemma 34.** *Algorithm 3 (ElimMaxVar) complies with its specification.*

C.3 Correctness of Algorithm 3 (LinExpSat)

▶ **Proposition 4.** *Algorithm 2 (LinExpSat) is a correct procedure for deciding the satisfiability of linear-exponential systems over* $\mathbb{N}$.

Proof. We prove that Algorithm 2 (LinExpSat) complies with its specification.

Consider an input linear-exponential system $\varphi(x_1, \dots, x_n)$ (with no divisibility constraints). Let Θ be the set of all orderings $\theta(x_0, \dots, x_n) = t_1 \geq t_2 \geq \dots \geq t_n \geq 2^{x_0} = 1$ such that $(t_1, \dots, t_n)$ is a permutation of the terms $2^{x_1}, \dots, 2^{x_n}$, and x_0 is a fresh variable used as a placeholder for 0. The following equivalence is immediate:

$$\varphi \iff \bigvee_{\theta \in \Theta} \exists x_0 (\theta \land \varphi).$$

Therefore, φ is satisfiable if and only if so is some $\exists x_0(\theta \land \varphi)$ with $\theta \in \Theta$. The algorithm starts by guessing such a θ (line 2). Without loss of generality, let us assume for simplicity that the procedure guesses the ordering $\theta(x_0, \dots, x_n) = 2^{x_n} \geq 2^{x_{n-1}} \geq \dots \geq 2^{x_1} \geq 2^{x_0} = 1$.

Throughout the proof, we write β for a non-deterministic branch of the procedure, represented as a list of line numbers of the algorithm decorated with guesses for lines featuring the non-deterministic **guess** statement.

We remind the reader that we represent a non-deterministic branch β as a list of entries containing the name of the algorithm being executed, the line number being executed, and, for lines featuring the non-deterministic **guess** statement, the performed guess. We write $\beta = \beta_1 \beta_2$ whenever β can be decomposed on a prefix β_1 and suffix β_2.

We define B_i for the set of all non-deterministic branches of the procedure ending with the entry (LinExpSat, "line 3") after iterating i times the loop starting at line 3 (that is, the entry (LinExpSat, "line 3") appears $i+1$ times in the representation of the non-deterministic branch). Note that every $\beta \in B_i$ uniquely corresponds to a system φ_β and an ordering θ_β, that are stored in the homonymous program variables. Because of the above assumption on θ, we can suppose B_0 to contain a single non-deterministic branch β_0 such that $\varphi_{\beta_0} = \varphi$, and $\theta_{\beta_0} = \theta$.

We show that the loop of line 3 enjoys the following loop invariant (where $i \in \mathbb{N}$, assuming that the loop executes at least i times). For every $\beta \in B_i$,

1. the variables in φ_β are from either $\{x_0, \dots, x_{n-i}\}$ or from $\boldsymbol{z}_\beta$, where variables in $\boldsymbol{z}_\beta$ are different from $x_0, \dots, x_n$;
2. all variables that occur exponentiated in φ_β are among $x_0, \dots, x_{n-i}$ (that is, the variables in $\boldsymbol{z}_\beta$ do not occur in exponentials);
3. all variables z belonging to $\boldsymbol{z}_\beta$ are such that $(z < 2^{x_{n-i}})$ is an inequality in φ_β;
4. the vector $\boldsymbol{z}_\beta$ contains at most i variables;
5. θ_β is the ordering $2^{x_{n-i}} \geq 2^{x_{n-i-1}} \geq \dots \geq 2^{x_1} \geq 2^{x_0} = 1$.
6. Moreover, $\bigvee_{\beta \in B_i} \theta_\beta \land \varphi_\beta$ is equisatisfiable with $\theta \land \varphi$ over $\mathbb{N}$.

First, observe that this invariant is trivially true for $B_0 = \{\beta_0\}$, for which $\boldsymbol{z}_{\beta_0}$ is the empty set of variables. Second, observe that from Item 5, for every $\beta_n \in B_n$, the ordering θ_{β_n} equals $2^{x_0} = 1$. This causes the loop in line 3 to exit after n iterations. Then, the definition of θ_{β_n}, together with Items 2 and 3 of the invariant, force x_0 and all variables in $\boldsymbol{z}_{\beta_n}$ to be equal to zero, which in turn imply $\varphi \land \theta$ to be equisatisfiable with $\varphi_{\beta_n}(\boldsymbol{0})$, by Item 6. Therefore, in order to conclude the proof it suffices to show that the above invariant is indeed preserved at every iteration of the loop.

Below, we assume the loop invariant to be true for some $(i-1) \in \mathbb{N}$, and that the loop is executed at least i times. For $\beta \in B_{i-1}$, we define $B_i^\beta \coloneqq \{\beta' \in B_i : \beta' = \beta\beta'' \text{ for some } \beta''\}$, that is, B_i^β contains the non-deterministic branches of B_i that are obtained by executing the body of the loop in line 3 once, starting from β. We show that every $\beta' \in B_i^\beta$ satisfies Items 1–5, and that $\bigvee_{\beta' \in B_i^\beta} \theta_{\beta'} \land \varphi_{\beta'}$ is equisatisfiable with $\theta_\beta \land \varphi_\beta$ over $\mathbb{N}$. Observe that the latter statement implies Item 6, because of the identity $B_i = \bigcup_{\beta \in B_{i-1}} B_i^\beta$ that follows directly from the definition of B_i.

Let $\beta \in B_{i-1}$. Note that $2^{x_{n-i+1}}$ is the leading exponential term of θ_β and $2^{x_{n-i}}$ is its successor (using the terminology in line 5). For notational convenience, we rename the variables x_{n-i+1} and x_{n-i} with, respectively, x and y (these are the names used in the pseudocode, see lines 4 and 5). During the ith iteration, the loop manipulates φ_β so that it becomes a quotient system induced by θ_β, which is then fed to Algorithm 3 in order to remove the variable x. By Item 3, all occurrences of the modulo operator $(w \bmod 2^x)$ can be simplified, i.e.,

$$\varphi_\beta \iff \varphi'_\beta\,, \qquad \text{where } \varphi'_\beta \coloneqq \varphi_\beta[w \,/\, (w \bmod 2^x)]. \tag{32}$$

The procedure uses this equivalence in line 6. Following line 7, we now consider all variables z in φ_β such that z is x or z does not appear in θ_β. In other words, these are all the variables $\boldsymbol{z}_\beta$ from Item 1 of the invariant, say $z_1, \dots, z_m$, plus the variable x (note: by Item 4, $\boldsymbol{z}_\beta$ has at most $i-1$ variables). For each variable z_j, with $j \in [1, m]$, consider two fresh variables x'_j and z'_j. Furthermore, let x'_0 and z'_0 be two fresh variables to be used for replacing x. We see each variable z'_j, with $j \in [0, m]$, as remainders modulo 2^y, whereas each x'_j is a quotient of the division by 2^y. Let $\boldsymbol{x}' = (x'_0, \dots, x'_m)$ and $\boldsymbol{z}' = (z'_0, \dots, z'_m)$. The following equivalence (over $\mathbb{N}$) is straightforward:

$$\begin{aligned}\varphi'_\beta \iff \exists \boldsymbol{x}' \exists \boldsymbol{z}' \Big(\varphi'_\beta \land \bigwedge\nolimits_{j=1}^m (z_j = x'_j \cdot 2^y + z'_j \land 0 \leq z'_j < 2^y) \\ \land (x = x'_0 \cdot 2^y + z'_0 \land 0 \leq z'_0 < 2^y)\Big).\end{aligned}$$

By Item 1, the variables $z_1, \dots, z_m$ only occur linearly in φ'_β and can thus be eliminated by substitution thanks to the equalities $z_j = x'_j \cdot 2^y + z'_j$ above. We can also substitute the linear occurrences of x in φ'_β with $x'_0 \cdot 2^y + z'_0$, but since this variable may occur exponentiated we must preserve the equality $x = x'_0 \cdot 2^y + z'_0$. Over $\mathbb{N}$, we have,

$$\begin{aligned}\exists \boldsymbol{z}_\beta \varphi'_\beta \iff \exists \boldsymbol{x}' \exists \boldsymbol{z}' \Big(\varphi'_\beta[x'_j \cdot 2^y + z'_j \,/\, z_j : j \in [1, m]][x'_0 \cdot 2^y + z'_0 \,/\, x] \land \\ \bigwedge\nolimits_{j=0}^m 0 \leq z'_j < 2^y \land (x = x'_0 \cdot 2^y + z'_0)\Big).\end{aligned}$$

Define $\varphi''_\beta \coloneqq \varphi'_\beta[x'_j \cdot 2^y + z'_j \,/\, z_j : j \in [1, m]][x'_0 \cdot 2^y + z'_0 \,/\, x]$. Following the ordering θ_β, in φ''_β all occurrences of the modulo operator featuring terms $x'_j \cdot 2^y + z'_j$, with $j \in [0, m]$, can be simplified. In particular, $((x'_j \cdot 2^y + z'_j) \bmod 2^y)$ can be rewritten to z'_j. Similarly, $((x'_j \cdot 2^y + z'_j) \bmod 2^w)$, where w is a variable distinct form y and such that θ_β implies $2^w \leq 2^y$, can be rewritten to $(z_j' \bmod 2^w)$. Let φ_β''' be the system obtained from φ_β'' after all these modifications. In the pseudocode, all these updates to φ_β' are done by the **foreach** loop from line 8 of the procedure. Over $\mathbb{N}$, we have,

$$\begin{aligned}&\exists \boldsymbol{z}_\beta(\theta_\beta \land \varphi_\beta') \iff \\ &\qquad \exists \boldsymbol{x}' \exists \boldsymbol{z}' \Big(\theta_\beta \land (\varphi_\beta''' \land \bigwedge\nolimits_{j=0}^{m} 0 \le z_j' < 2^y) \land (x = x_0' \cdot 2^y + z_0')\Big).\end{aligned} \tag{33}$$

Observe that $\varphi_\beta''' \land \bigwedge_{j=0}^{m} 0 \le z_j' < 2^y$ is a quotient system induced by θ_β. In line 15, the procedure calls Algorithm 3 on input θ_β, $(\varphi_\beta''' \land \bigwedge_{j=0}^{m} 0 \le z_j' < 2^y)$, and $[x_0' \cdot 2^y + z_0' \,/\, x]$. Over $\mathbb{N}$, by Lemma 34 and following the **global output** of Algorithm 3, we obtain

$$\begin{aligned}&\exists x \exists \boldsymbol{x}' \Big(\theta_\beta \land (\varphi_\beta''' \land \bigwedge\nolimits_{j=0}^{m} 0 \le z_j' < 2^y) \land (x = x_0' \cdot 2^y + z_0')\Big) \iff \\ &\qquad\qquad (\exists x \theta_\beta) \land \bigvee\nolimits_{\beta' \in B_i^\beta} \psi_{\beta'},\end{aligned} \tag{34}$$

where each **branch output** $\psi_{\beta'}$ is a linear-exponential system in variables $x_0, \dots, x_{n-i-1}, y$ and $\boldsymbol{z}'$, where variables in $\boldsymbol{z}'$ do not occur in exponentials, and $\psi_{\beta'}$ features inequalities $0 \le z_j' < 2^y$ for every $j \in [0, m]$.

By Equations (32)–(34), over $\mathbb{N}$ we have:

$$\exists \boldsymbol{z}_\beta \exists x(\theta_\beta \land \varphi_\beta) \iff \exists \boldsymbol{z}' \left((\exists x \theta_\beta) \land \bigvee\nolimits_{\beta' \in B_i^\beta} \psi_{\beta'}\right). \tag{35}$$

The algorithm then excludes 2^x from θ_β (line 16). The iteration of the loop has been performed.

To conclude the proof, it suffices to observe that, for every $\beta' \in B_i^\beta$, $\theta_{\beta'}$ and $\varphi_{\beta'}$ satisfy the invariant of the loop in line 3. Recall that by x and y we denoted the variables x_{n-i+1} and x_{n-i}, and that we are assuming the loop invariant to hold for $i-1$. Then, with respect to $\varphi_{\beta'}$, the sequence of variables $\boldsymbol{z}_{\beta'}$ is $\boldsymbol{z}'$ and

- Items 1–3 follow directly from the definition of the formulae ψ returned by Algorithm 3.
- Item 4 follows by definition of $\boldsymbol{z}'$ and from the fact that $\boldsymbol{z}_\beta$ has at most $i-1$ variables.
- Item 5 follows directly from the definition of θ_β.
- Item 6 follows by the equivalence in (35), which implies that $\bigvee_{\beta' \in B_i^\beta} \theta_{\beta'} \land \varphi_{\beta'}$ is equisatisfiable with $\theta_\beta \land \varphi_\beta$ over $\mathbb{N}$ (as already discussed above, this suffices to establish Item 6). ◀

D Proofs from Section 7: the complexity of Algorithm 2 (LinExpSat)

To simplify the exposition, we borrow from [2] the use of "parameter tables" to describe the growth of the parameters. These tables have the following shape.

	$f_1(\cdot)$	$f_2(\cdot)$	$\cdots$	$f_n(\cdot)$
φ	a_1	a_2	$\cdots$	a_n
ψ_1	$f_{1,1}(a_1, \dots, a_n)$	$f_{1,2}(a_1, \dots, a_n)$	$\cdots$	$f_{1,n}(a_1, \dots, a_n)$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
ψ_m	$f_{m,1}(a_1, \dots, a_n)$	$f_{m,2}(a_1, \dots, a_n)$	$\cdots$	$f_{m,n}(a_1, \dots, a_n)$

In this table, φ,ψ_1,...,ψ_m are systems, $f_1(\cdot)$,...,$f_n(\cdot)$ are parameter functions from systems to $\mathbb{N}$, $a_1, \dots, a_n \in \mathbb{N} \setminus \{0\}$, and all $f_{j,k}$ are functions from $\mathbb{N}^n$ to $\mathbb{N}$. The system φ should be seen as the "input", whereas $\psi_1, \dots, \psi_m$ should be seen as "outputs". The table states that if $f_i(\varphi) \leq a_i$ for all $i \in [1,n]$, then $f_k(\psi_j) \leq f_{j,k}(a_1,\dots,a_n)$ for all $j \in [1,m]$ and $k \in [1,n]$.

However, we will have **two exceptions to this semantics**:

- for the parameter $mod(\varphi)$, the table must be read as if $mod(\varphi)$ *divides* a (with a value in the row of φ corresponding to $mod(\varphi)$), then $mod(\psi_j)$ *divides* $f_j(a_1,\dots,a_n)$ for all $j \in [1,m]$. To repeat, the parameter table always encodes a $\leq$ relationship between input and output, except for $mod(\varphi)$ where this relationship concerns divisibility. An example of this is given in Lemma 36, where the $mod(\cdot)$ column should be read as if $mod(\varphi) \mid d$, then $mod(\chi) \mid \Phi(d)$ and $mod(\gamma) \mid d$. Here Φ stands for Euler's totient function, see below.
- we will sometimes track the parameter $\#(lst(\cdot,\cdot))$, which takes two arguments instead of one as in the parameters $f_1,\dots,f_n$ above. In this case we will indicate what is the second argument inside the cells of the column of $\#(lst(\cdot,\cdot))$. An example of this is given in Lemma 37, where $lst(\cdot,\cdot)$ should be evaluated with respect to the ordering θ when considering φ, and with respect to the ordering θ' when considering ψ.

Unfilled cells in the tables correspond to quantities that are not relevant for ultimately showing Proposition 5 (see, e.g., the cells for $\|\varphi\|_{\mathfrak{L}}$ and $\|\gamma\|_{\mathfrak{L}}$ in Lemma 36).

Our analysis requires the use of Euler's totient function, which we indicate with the capital phi symbol Φ to not cause confusion with homonymous linear-exponential systems. Given a natural number $a \geq 1$ whose prime decomposition is $\prod_{i=1}^n p_i^{k_i}$ (here $p_1,\dots,p_n$ are distinct primes and all k_i are at least 1), $\Phi(a) \coloneqq \prod_{i=1}^n (p_i^{k_i-1}(p_i-1))$. Remark that $\Phi(1) = 1$.

D.1 Analysis of Algorithm 4 (SolvePrimitive)

We will need the following folklore result (which we prove for completeness):

▶ **Lemma 35.** *Let d be an odd number. Consider $x \in \mathbb{N} \setminus \{0\}$ satisfying $d \mid 2^x - 1$. Let $\ell \in [1, d-1]$ be the multiplicative order of 2 modulo d. Then ℓ divides x.*

Proof. Recall once more that ℓ is the smallest natural number in $[1, d-1]$ satisfying the divisibility constraint $d \mid 2^x - 1$. *Ad absurdum*, suppose that ℓ does not divide x. Then, there are $r \in [1, \ell-1]$ and $\lambda \in \mathbb{N}$ such that $x = \lambda \cdot \ell + r$. We have

$$d \mid 2^x - 1 \iff d \mid 2^{\lambda\ell+r} - 1 \iff d \mid 2^{\lambda\ell} \cdot 2^r - 1 \iff d \mid 2^r - 1.$$

Since $r \in [1, \ell-1]$, this contradicts the minimality of ℓ. Hence, ℓ divides x. ◀

▶ **Lemma 36.** *Consider a (u,v)-primitive linear-exponential system φ. On input (u,v,φ), Algorithm 4 (SolvePrimitive) returns a pair of linear-exponential systems (χ,γ) with bounds as shown below:*

	$\#(\cdot)$	$\|\cdot\|_{\mathfrak{L}}$	$mod(\cdot)$	$\|\cdot\|_1$
φ	s		d	c
χ	2	1	$\Phi(d)$	$6 + 2 \cdot \log(\max(c,d))$
γ	s		d	$2^6 \cdot \max(c,d)^3$

The algorithm runs in non-deterministic polynomial time in the bit size of φ.

Proof. Let us start by showing the bounds in the table, column by column.

Number of constraints. Regarding $\#(\cdot)$, the bound on $\#\chi$ is trivial and already given by the specification of the algorithm (proven correct in Lemma 21). For $\#\gamma$, note that this system is defined either in line 7 or in line 15. In both cases, γ is obtained by substitution into a subsystem of φ (in the first case it is exactly φ). So, $\#\gamma \leq s$.

Linear norm. We are only interested in $\|\chi\|_{\mathfrak{L}}$. Again following the specification of the algorithm, $\|\chi\|_{\mathfrak{L}} = 1$.

Least common multiple of the divisibility constraints. Regarding $mod(\cdot)$, the case of γ is trivial. Again this system is obtained by substituting terms 2^u form a subsystem of φ, so $mod(\gamma) = mod(\varphi)$. For χ, we note that the only divisibility constraint this system can have is the divisibility $d' \mid u - r'$ from line 14. In that line, d' is the multiplicative order of 2 modulo $\frac{mod(\varphi)}{2^n}$, where n is the largest natural number dividing $mod(\varphi)$. We have $mod(\chi) = d'$. By Euler's theorem, $(\frac{mod(\varphi)}{2^n}) \mid 2^{\Phi(\frac{mod(\varphi)}{2^n})} - 1$, and therefore, by Lemma 35, d' divides $\Phi(\frac{mod(\varphi)}{2^n})$. Observe that for every $a, b \in \mathbb{N} \setminus \{0\}$ if $a \mid b$ then $\Phi(a) \mid \Phi(b)$. Then, $\Phi(\frac{mod(\varphi)}{2^n})$ divides $\Phi(mod(\varphi))$, which in turn divides $\Phi(d)$. We conclude that $mod(\chi) \mid \Phi(d)$, as required.

1-norm. In the case of χ, $\|\chi\|_1$ is bounded by $C+1$, where C is defined as in line 3. Then,

$$\begin{aligned}
\|\chi\|_1 &\leq \max(n, 3 + 2 \cdot \lceil\log(\|\varphi\|_1)\rceil) + 1\\
&\leq \max(\lceil\log(mod(\varphi))\rceil, 3 + 2 \cdot \lceil\log(\|\varphi\|_1)\rceil) + 1\\
&\leq 4 + 2 \cdot \lceil\log(\max(\|\varphi\|_1, mod(\varphi)))\rceil\\
&\leq 6 + 2 \cdot \log(\max(c, d)).
\end{aligned}$$

For $\|\gamma\|_1$, we need to study the effects of the two substitutions $[2^c \,/\, 2^u]$ and $[2^n \cdot r \,/\, 2^u]$ of lines 7 and 15. Note that $2^n \cdot r \leq mod(\varphi)$ and $2^c \leq 2^C$, and thus we have

$$\begin{aligned}
\|\gamma\|_1 &\leq \|\varphi\|_1 \cdot \max(mod(\varphi), 2^C)\\
&\leq c \cdot \max(d, 2^{\max(n, 3+2\cdot\lceil\log(\|\varphi\|_1)\rceil)})\\
&\leq c \cdot \max(d, 2^{6+2\cdot\log(\max(c,d))}) && \text{from the computation of } \|\chi\|_1\\
&\leq c \cdot \max(d, 2^6 \cdot \max(c,d)^2) \leq 2^6 \cdot \max(c,d)^3.
\end{aligned}$$

This completes the proof of the parameter table.

Let us now discuss the runtime of the procedure. Lines 1–3 clearly run in (deterministic) polynomial size in the bit size of φ (to construct the pair (d, n) simply compute $mod(\gamma)$ and then iteratively divide it by 2 until obtaining an odd number). To guess of c in line 4 it suffices to guess $\lceil\log(C)\rceil + 1$ many bits (the $+1$ is used to encode the case of $\star$); which can be done in non-deterministic polynomial time. Lines 6, 7 and 9 only require polynomial time in the bit size of φ. To guess r in line 10 it suffices guessing at most $\lceil mod(\varphi)\rceil$ many bits. We then arrive to lines 11–13. These three lines correspond to discrete logarithm problems, which can be solved in non-deterministic polynomial time (in the bit size of φ), see [18]. For lines 11 and 12, one has to find $x \geq 0$ such that $d \mid 2^x - 2^n \cdot r$. For line 13, one has to find $x \in [1, d-1]$ *minimal* such that $d \mid 2^x - 1$. Note that asking for the minimality here is not a problem and x can be found in non-deterministic polynomial time: the non-deterministic algorithm simply guesses a number $\ell \in [1, d-1]$ and returns it together with its prime factorisation. By Lemma 35, certifying in polynomial time that ℓ is the multiplicative order of 2 modulo d is trivial:

1. check if $(2^\ell \bmod d)$ equals 1 with a fast modular exponentiation algorithm;
2. check if the given prime factorisation is indeed the factorisation of ℓ;
3. check if dividing ℓ by any prime in its prime factorisation results in a number ℓ' such that $(2^\ell \bmod d)$ does not equal 1.

After line 13, the algorithm runs in polynomial time in the bit size of φ. We conclude that, overall, Algorithm 4 runs in non-deterministic polynomial time in the bit size of φ. ◀

D.2 Analysis of Algorithm 3 (ElimMaxVar)

▶ **Lemma 37.** *Consider the execution of Algorithm 3 (*ElimMaxVar*) on input $(\theta, \varphi, \sigma)$, where $\theta(\boldsymbol{x})$ is an ordering of exponentiated variables, $\varphi(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{z}')$ is a quotient system induced by θ, with n exponentiated variables $\boldsymbol{x}$, m quotient variables $\boldsymbol{x}'$, m remainder variables $\boldsymbol{z}'$, and $\sigma = [x' \cdot 2^y + z' \,/\, x]$ is a delayed substitution. Let θ' be the ordering obtained from θ by removing its leading term 2^x. The linear-exponential systems ψ and γ defined in the algorithm when its execution reaches line 35 satisfy the following bounds:*

	$\#(\cdot)$	$\#lst(\cdot,\cdot)$	$\lVert\cdot\rVert_{\mathfrak{L}}$	$mod(\cdot)$	$\lVert\cdot\rVert_1$
φ	s	$\theta: \quad \ell$	a	d	c
ψ	$s + 2 \cdot \ell + 3$	$\theta': \quad \ell + 2$	a	$\operatorname{lcm}(d, \Phi(\alpha \cdot d))$	$16 \cdot (m+2)^2 \cdot (c+2) + 4 \cdot \log(d)$
γ	$s + m + 2$			$\operatorname{lcm}(\alpha \cdot d, \Phi(\alpha \cdot d))$	$(c+3)^{14 \cdot (m+2)^2} \cdot d^3$

where $\alpha \leq (a+2)^{(m+2)^2}$. Moreover, the algorithm runs in non-deterministic polynomial time in the bit size of the input.

Proof. We remark that the formula ψ in the statement of the lemma also corresponds to the formula in output of Algorithm 3.

Because of the length of the proof, it is useful to split Algorithm 3 into three steps.

(a) We consider lines 1–21, and write down the parameter table with respect to the systems ψ and γ when the execution reaches line 22, before calling Algorithm 1.

(b) We consider lines 22–25 to discuss how γ and the auxiliary system χ evolve after the calls to Algorithm 1 and Algorithm 4.

(c) We analyse lines 26–35, completing the proof.

Observe that this division into Steps (a), (b), and (c) is slightly different from the one into Steps (i), (ii), and (iii), which we used to prove correctness of the algorithm. Now the first step does not include line 22. Since we have already performed the complexity analysis of Algorithm 1 and Algorithm 4, it is here convenient to analyse the two in a single step.

Step (a). Let ψ and γ be as defined in the algorithm when its execution reaches line 22, before Algorithm 1 is called. They satisfy the following bounds:

	$\#(\cdot)$	$\#lst(\cdot,\cdot)$	$\lVert\cdot\rVert_{\mathfrak{L}}$	$mod(\cdot)$	$\lVert\cdot\rVert_1$
φ	s	$\theta: \quad \ell$	a	d	c
ψ	$s + 2 \cdot \ell$	$\theta': \quad \ell$	a	d	$2 \cdot c + 1$
$\gamma[2^u \,/\, u]$	s		$a^\star$	d	$c + 1$

$\star$: for γ, we are only interested in the coefficients of variables distinct from u. Hence the "weird" substitution of u with 2^u.

As remarked below the table, in fact more than being interested in γ, we are interested in $\gamma[2^u \,/\, u]$. The reason for this is simple: after calling Algorithm 1 on γ, the variable u (which is not eliminated by this algorithm) is replaced with 2^u in line 23. Because u is a placeholder for an exponentiated, not a linearly occurring variable, the coefficient of u in γ should not be taken into account as part of $\lVert\gamma\rVert_{\mathfrak{L}}$. As done for Lemma 36, we show that the bounds in this table are correct column by column, starting form the leftmost one.

Number of constraints. The bound $\#\gamma \leq s$ is simple to establish. The system γ is initially defined as $\top$ in line 2. Afterwards, in every iteration of the **foreach** loop of line 4, γ is conjoined with an additional constraint, either an inequality (line 16) or a divisibility constraint (line 20). Therefore, when the execution reaches line 22, the number of constraints in γ is bounded by the number of iterations of the **foreach** loop, that is, $\#\varphi$. Let us move to $\#\psi$. Again, ψ is initially defined as $\top$ in line 2. Similarly to γ, the updates in line 14 (for strict inequalities), line 17 (for equalities), and line 21 (for divisibilities) add to ψ at most $\#\varphi$ many constraints. Although it is easy to see, notice that strict inequalities become non-strict in line 14; hence, in this case, the **if** statement of line 17 is clearly not true, and the formula ψ is only updated in line 14.

We then need to account for the update done in line 10. This update adds two inequalities whenever the guard "$\Delta(\rho)$ is undefined" in the **if** statement of line 8 is true. Here ρ is the least significant part of an (in)equality of φ. Each least significant part is considered only once, because of the update done to the map δ in line 11. Therefore, during the **foreach** loop of line 4, at most $2 \cdot \#\mathit{lst}(\varphi)$ many inequalities can be added to ψ because of line 10. We conclude that $\#(\psi) \leq s + 2 \cdot \ell$.

Least significant terms. We are only interested in bounding $\#\mathit{lst}(\psi, \theta')$. Note that, with respect to the code of the algorithm, the leading exponential term in θ' is 2^y. Then, let us look once more at the (in)equalities added to ψ in lines 10, 14, and 17. They have all the form $g \cdot 2^y \pm \rho \sim 0$, with $\sim \{<, \leq, =\}$, $g \in \mathbb{Z}$ and ρ being the least significant part of some term in φ (with respect to θ). By definition, $\pm\rho$ is the least significant part of $g \cdot 2^y \pm \rho \sim 0$ with respect to θ'. We conclude that $\mathit{lst}(\psi, \theta')$ is the set containing all such ρ and $-\rho$. By definition of lst, both ρ and $-\rho$ occur in $\mathit{lst}(\varphi, \theta)$. Therefore, $\#\mathit{lst}(\psi, \theta') \leq \#\mathit{lst}(\varphi, \theta) \leq \ell$.

Linear norm. For the aforementioned reasons, we look at $\gamma[2^u \,/\, u]$ instead of γ. The linear norm of $\gamma[2^u \,/\, u]$ is solely dictated by the update $\gamma \leftarrow \gamma \land (a \cdot u + f(\boldsymbol{x}') + r \sim 0)$ done in line 16. Here, $f(\boldsymbol{x}')$ is a linear term $a_1 \cdot x_1' + \dots + a_m \cdot x_m' + a_{m+1}$ that occur in a quotient term $a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + \rho$ of φ. In φ, the linear term f is accounted for in defining the linear norm. We conclude that $\|\gamma[2^u \,/\, u]\|_{\mathfrak{L}} \leq \|\varphi\|_{\mathfrak{L}} \leq a$.

One bounds $\|\psi\|_{\mathfrak{L}}$ in a similar way. In the updates done to ψ in lines 10, 14, and 17, the linear norm of the added (in)equalities is the linear norm of ρ. Here, ρ is a subterm of φ, and therefore we have $\|\psi\|_{\mathfrak{L}} \leq \|\varphi\|_{\mathfrak{L}} \leq a$.

Least common multiple of the divisibility constraints. Bounding this parameter is simple. Divisibility constraints $d \mid \tau$ are added to γ and ψ in lines 20 and 21. The divisor d is also a divisor of some divisibility constraint of φ. So, $\mathit{mod}(\psi) \mid \mathit{mod}(\varphi)$ and $\mathit{mod}(\gamma) \mid \mathit{mod}(\varphi)$.

1-norm. Regarding $\|\gamma\|_1$, again we need to look at the updates caused by line 16. There, notice that the 1-norm of the term $a \cdot u + f(\boldsymbol{x}') + \rho$ is bounded by $\|\varphi\|_1$, and the constant r is bounded in absolute value by $\|\rho\|_1 + 1$. We conclude that $\|\gamma\|_1 \leq c + 1$.

For $\|\psi\|_1$ the analysis is similar. The inequalities $g \cdot 2^y \pm \rho \sim 0$ added in lines 10, 14, and 17 are such that $\|\rho\|_1 \leq \|\varphi\|_1$ and $|g| \leq \|\varphi\|_1 + 1$. Then, $\|\psi\|_1 \leq 2 \cdot c + 1$.

We have now established that ψ and γ satisfy the bounds of the table for line 22 of the algorithm. We move to the second step of the proof.

Step (b). This step only involves the formula γ, which is first manipulated by Algorithm 1 in order to remove all quotient variables in $\boldsymbol{x}'$ that are different from the quotient variable x' appearing in the delayed substitution, and then passed to Algorithm 4 in order to "linearise" the occurrences of u.

Note that before the execution of line 22, γ has $m+1$ variables (i.e., u plus the m quotient variables $\boldsymbol{x}'$). Note moreover that in line 22 we conjoin γ with the system $\boldsymbol{x}' \geq \boldsymbol{0}$ featuring m inequalities. Then, starting form the bounds on γ obtained in the previous part of the proof, by Lemma 20, line 22 yields the following new bounds for γ:

	$\#(\cdot)$	$\|\cdot\|_{\mathfrak{L}}$	$\mathit{mod}(\cdot)$	$\|\cdot\|_1$
φ	s	a	d	c
γ	$s+m$		$\alpha \cdot d$	$(c+3)^{4\cdot(m+2)^2} \cdot d$

where $\alpha \in [1, (\|\varphi\|_{\mathfrak{L}}+2)^{(m+2)^2}] \subseteq [1, (a+2)^{(m+2)^2}]$.

Line 23 replaces u with 2^u and does not change the bounds above. The procedure then calls Algorithm 4 on γ (line 24). The output of this algorithm updates γ and produces the auxiliary system χ. By Lemma 36, these systems enjoy the following bounds:

	$\#(\cdot)$	$\|\cdot\|_{\mathfrak{L}}$	$\mathit{mod}(\cdot)$	$\|\cdot\|_1$
φ	s		d	c
χ	2	1	$\Phi(\alpha \cdot d)$	$6+2\cdot\log(\max((c+3)^{4\cdot(m+2)^2}\cdot d, \alpha\cdot d))$
γ	$s+m$		$\alpha \cdot d$	$2^6 \cdot \max((c+3)^{4\cdot(m+2)^2}\cdot d, \alpha\cdot d)^3$

Note that, by definition, $\|\varphi\|_{\mathfrak{L}} \leq \|\varphi\|_1 \leq c$. After Algorithm 4, the procedure updates χ to $\chi[x-y\,/\,u][x' \cdot 2^y + z'\,/\,x]$. In χ, the coefficient of u in inequalities is always ± 1. Then, this update cause the 1-norm of χ to increase by 2. This completes Step (b).

Before moving to Step (c), let us clean up a bit the bounds on $\|\chi\|_1$ and $\|\gamma\|_1$. We have,

$$\begin{aligned}\max((c+3)^{4\cdot(m+2)^2}\cdot d, \alpha\cdot d) &\leq \max((c+3)^{4\cdot(m+2)^2}\cdot d, (c+2)^{(m+2)^2}\cdot d)\\ &\leq (c+3)^{4\cdot(m+2)^2}\cdot d\,,\end{aligned}$$

and so,

$$\begin{aligned}\|\chi\|_1 &\leq 6+2\cdot\log((c+3)^{4\cdot(m+2)^2}\cdot d)+2\\ &= 8+8\cdot(m+2)^2\cdot\log(c+3)+2\cdot\log(d)\\ &= 8\cdot(1+(m+2)^2\cdot\log(c+3))+2\cdot\log(d)\,;\end{aligned}$$

$$\begin{aligned}\|\gamma\|_1 &\leq 2^6\cdot((c+3)^{4\cdot(m+2)^2}\cdot d)^3\\ &\leq (c+3)^{12\cdot(m+2)^2+6}\cdot d^3\\ &\leq (c+3)^{14\cdot(m+2)^2}\cdot d^3\,.\end{aligned}$$

Step (c). We now analyse lines 26–35. Note that in these lines the definition of χ is used to update γ and ψ. Since ψ was not updated during Step (b), it still has the bounds from Step (a). We show that, when the execution of the procedure reaches line 35, ψ and γ satisfy the bounds in the statement of the lemma, here reproposed:

	$\#(\cdot)$	$\#\mathit{lst}(\cdot,\cdot)$	$\|\cdot\|_{\mathfrak{L}}$	$\mathit{mod}(\cdot)$	$\|\cdot\|_1$
φ	s	$\theta\colon\ \ell$	a	d	c
ψ	$s+2\cdot\ell+3$	$\theta'\colon\ \ell+2$	a	$\mathrm{lcm}(d, \Phi(\alpha\cdot d))$	$16\cdot(m+2)^2\cdot(c+2)+4\cdot\log(d)$
γ	$s+m+2$			$\mathrm{lcm}(\alpha\cdot d, \Phi(\alpha\cdot d))$	$(c+3)^{14\cdot(m+2)^2}\cdot d^3$

where $\alpha \leq (a+2)^{(m+2)^2}$.

Observe that the procedure executes either lines 27–29 or lines 32–34, depending on whether χ is an equality.

Number of constraints. If χ is an equality, then a single constraint is added to ψ. If χ is not an equality, the procedures adds 3 constraints. Together with the bounds in the table of Step (a), we conclude that $\#\psi \leq s + 2 \cdot \ell + 3$ when the procedure reaches line 35.
In the case of γ, at most 2 more constraints are added (this corresponds to the case of χ not being an equality). Then, $\#\gamma \leq s + m + 2$.

Least significant terms. We are only interested in $\mathit{lst}(\psi, \theta')$. Independently on whether χ is an equality, the updates done to ψ only add the least significant terms $\pm(-z' + y + c)$. Therefore, $\#\mathit{lst}(\psi, \theta') \leq \ell + 2$.

Linear norm. Again, we are only interested in ψ. Independently on the shape of χ, in the inequalities added to ψ by the procedure all linearly occurring variables (z' and y in the pseudocode) have ± 1 as a coefficient. Since we are assuming $a \geq 1$, from the table of Step (a) we conclude $\|\psi\|_{\mathfrak{L}} \leq a$.

Least common multiple of the divisibility constraints. When χ is an equality, no divisibility constraints are added to ψ and γ. When χ is not an equality, a single divisibility constraint is added to both ψ and γ, with divisor $\mathit{mod}(\chi)$.
From the tables in Step (a) and Step (b), we conclude that $\mathit{mod}(\psi)$ divides $\operatorname{lcm}(d, \Phi(\alpha \cdot d))$, and $\mathit{mod}(\gamma)$ divides $\operatorname{lcm}(\alpha \cdot d, \Phi(\alpha \cdot d))$.

1-norm. The 1-norm of the (in)equalities added to γ is bounded by $\|\chi\|_1$. Therefore,

$$\begin{aligned}\|\gamma\|_1 &\leq \max\left(8 \cdot (1 + (m+2)^2 \cdot \log(c+3)) + 2 \cdot \log(d)\,,\, (c+3)^{14 \cdot (m+2)^2} \cdot d^3\right) \\ &\leq (c+3)^{14 \cdot (m+2)^2} \cdot d^3.\end{aligned}$$

The 1-norm of the (in)equalities added to ψ is bounded by $2 \cdot \|\chi\|_1$. Hence,

$$\begin{aligned}\|\psi\|_1 &\leq \max\left(16 \cdot (1 + (m+2)^2 \cdot \log(c+3)) + 4 \cdot \log(d)\,,\, 2 \cdot c + 1\right) \\ &\leq 16 \cdot (m+2)^2 \cdot (c+2) + 4 \cdot \log(d).\end{aligned}$$

This completes the proof of the parameter table in the statement of the lemma.

Let us now discuss the runtime of the procedure. First, recall that the bit size of a system $\varphi'(x_1, \dots, x_k)$ belongs to $O(\#\varphi \cdot k^2 \cdot \log(\|\varphi\|_1) \cdot \log(\mathit{mod}(\varphi)))$. This means that all the formula we have analysed above have a bit size polynomial in the bit size of φ. It is then quite simple to see that Algorithm 3 runs in non-deterministic polynomial time:

- the **foreach** loop in line 4 simply iterates over all constraints of φ, and thus runs in polynomial time in the bit size of φ.
- For the two guesses in lines 9 and 19 requires guessing at most $\lceil \log(\|\varphi\|_1) \rceil + 1$ and $\lceil \log(\mathit{mod}(\varphi)) \rceil$ many bits, respectively. This number is bounded by the bit size of φ.
- From the tables above, the inputs of Algorithm 1 and Algorithm 4 in lines 22 and 24 have a polynomial bit size with respect to the bit size of φ. Therefore, these two algorithms run in non-deterministic polynomial time in the bit size of φ, by Theorem 3 and Lemma 36.
- The call to Algorithm 1 done in line 35 takes again as input a formula with bit size polynomial in the bit size of φ. So, once more, this step runs in non-deterministic polynomial time.
- Every other line in the code runs in polynomial time in the bit size of φ. ◀

D.3 Analysis of Algorithm 2 (LinExpSat)

We first compute the bounds for a single iteration of the **while** loop of line 3.

▶ **Lemma 6.** *Consider the execution of* LinExpSat *on an input* $\varphi(x_1, \dots, x_n)$*, with* $n \geq 1$*. For* $i \in [0, n]$*, let* (φ_i, θ_i) *be the pair of a system and ordering obtained after the ith iteration of the* ***while*** *loop of line 3, where* $\varphi_0 = \varphi$ *and* θ_0 *is the ordering guessed in line 2. Then, for every* $i \in [0, n-1]$*,* φ_{i+1} *has at most* $n+1$ *variables, and for every* $\ell, s, a, c, d \geq 1$*,*

$$\text{if} \begin{cases} \#\mathit{lst}(\varphi_i, \theta_i) & \leq \ell \\ \#\varphi_i & \leq s \\ \|\varphi_i\|_{\mathfrak{L}} & \leq a \\ \|\varphi_i\|_1 & \leq c \\ \mathit{mod}(\varphi_i) & \mid d \end{cases} \quad \text{then} \begin{cases} \#\mathit{lst}(\varphi_{i+1}, \theta_{i+1}) & \leq \ell + 2(i+2) \\ \#\varphi_{i+1} & \leq s + 6(i+2) + 2 \cdot \ell \\ \|\varphi_{i+1}\|_{\mathfrak{L}} & \leq 3 \cdot a \\ \|\varphi_{i+1}\|_1 & \leq 2^5(i+3)^2(c+2) + 4 \cdot \log(d) \\ \mathit{mod}(\varphi_{i+1}) & \mid \operatorname{lcm}(d, \Phi(\alpha_i \cdot d)) \end{cases}$$

for some $\alpha_i \in [1, (3 \cdot a + 2)^{(i+3)^2}]$*. The* $(i+1)$*st iteration of the* ***while*** *loop of line 3 runs in non-deterministic polynomial time in the bit size of* φ_i*.*

Proof. We prove this result by first analysing lines 4–14 and then appealing to Lemma 37.

Observe that, from the proof of Proposition 4, the variables in φ_i are from either $\{x_0, \dots, x_{n-i}\}$, with x_0 being the fresh variable introduced in line 1, or from a vector $\boldsymbol{z}_i$ of at most i variables (see Item 1 and Item 4 of the invariant in the proof of Proposition 4). This already implies that φ_i has at most $n+1$ variables, for every $i \in [0, n]$.

Let $i \in [0, n-1]$. Starting from the system φ_i, we consider the execution of the **while** loop of line 3, until reaching the call to Algorithm 3 in line 15 (and before executing such a call). Let φ_i' be the quotient system induced by θ_i passed in input to Algorithm 3 at that point of the execution of the program. We show the following parameter table:

	$\#(\cdot)$	$\#\mathit{lst}(\cdot,\cdot)$	$\|\cdot\|_{\mathfrak{L}}$	$\mathit{mod}(\cdot)$	$\|\cdot\|_1$
φ_i	s	$\theta:\ \ell$	a	d	c
φ_i'	$s + 2 \cdot (i+1)$	$\theta:\ \ell + 2 \cdot (i+1)$	$3 \cdot a$	d	$2 \cdot (c+1)$

Number of constraints. Constraints are only added in line 10, as part of the **foreach** loop in line 8. Each iteration of this loop adds 2 constraints, and the number of iterations depends on the cardinality of $\boldsymbol{z}$. As explained in the proof of Proposition 4, $\boldsymbol{z}$ has at most $i+1$ variables, and therefore $\#\varphi_i' \leq \#\varphi_i + 2 \cdot (i+1)$.

Least significant terms. Note that while ψ_i is a linear-system, ψ_i' is a quotient system induced by θ, so in the latter any least significant part of a term is of the form $b \cdot y + \rho(\boldsymbol{x} \setminus \{x, y\}, \boldsymbol{z})$, where $\boldsymbol{x} \setminus \{x, y\}$ is the vector obtained from $\boldsymbol{x}$ by removing x and y. It is simple to see that the substitutions performed in lines 6, 11 and 13 do not change the cardinality of *lst*. Indeed, if two inequalities had the same least significant part before one of these substitutions, they will have the same least significant part also after the substitution. Least significant parts are however introduced in line 10. Indeed, the least significant part of $0 \leq z'$ is $-z'$ and the least significant part of $z' < 2^y$ is z'. Since the **foreach** loop in line 8 executes $i+1$ times, we conclude that $\#\mathit{lst}(\varphi_i', \theta_i) \leq \#\mathit{lst}(\varphi_i, \theta_i) + 2 \cdot (i+1)$.

Linear norm. We only consider the lines of the loop that might influence the linear norm. Line 6 performs on φ_i the substitution $[w \,/\, (w \bmod 2^x) : w \text{ is a variable}]$. This can (at most) double the linear norm, as terms of the form $a_1 \cdot w + a_2 \cdot (w \bmod 2^x)$ will be rewritten into $(a_1 + a_2) \cdot w$. We observe however that the maximum absolute value of a coefficient of some modulo operator $(w' \bmod 2^{w''})$ is still bounded by $\|\varphi_i\|_{\mathfrak{L}}$ after this update. The terms in the inequalities of line 10 have a linear norm of 1. Since we are assuming $a \geq 1$, this does not influence our bound for the linear norm of φ_i'. Line 11 does not influence the linear norm, as at this stage point in the execution of the procedure z' only occur in the inequalities added to the system in line 10. Line 12 does not increase the linear norm, since before the substitution performed in this line the variable z' does not occur inside the modulo operator. Line 13 does increase the linear norm: the substitution $[(x' \cdot 2^y + z') \,/\, z]$ causes $\|\varphi_i'\|_{\mathfrak{L}}$ to be bounded by $3 \cdot a$ (after all iterations of the loop of line 8). This is because, before performing the substitution, the coefficients of z in inequalities of φ_i' are bounded by $2 \cdot a$ (this stems from the update performed in line 6), and the coefficient of z' is bounded by a (this stems from the substitution in line 11).

Least common multiple of the divisibility constraints. No line increases $mod(\cdot)$.

1-norm. Recall that we are assuming $c \geq 1$. Observe that lines 6, 11 and 12 do not modify the 1-norm. The inequalities added in line 10 have a 1-norm of 2, and are thus bounded by $c + 1$. The only other line that affects the 1-norm is therefore line 13, which doubles it.

We have now established that φ_i' satisfies the bounds on the parameter table. Observe also that φ_i' has at most $(i+1)$ quotient variables and $(i+1)$ remainder variables (since $\boldsymbol{z}$ has at most $(i+1)$ many variables). Then, the bounds on φ_{i+1} provided in the statement of the lemma follow directly from Lemma 37.

We conclude the proof of the lemma by discussing the runtime of the body of the **while** loop. Observe that lines 4–14 run in deterministic polynomial time. Moreover, from the parameter table above, φ_i' is of bit size polynomial in the bit size of φ_i. Therefore, when accounting for the execution of Algorithm 3, we conclude by Lemma 37 that the body of the **while** loop runs in non-deterministic polynomial time in the bit size of φ_i. ◀

Before iterating the bounds of Lemma 6, thus completing the analysis of the complexity of Algorithm 2, we prove Lemma 7. The proof of this lemma requires a few auxiliary definitions that we now introduce.

Let $a \in \mathbb{N} \setminus \{0\}$ and let $\prod_{i=1}^n p_i^{k_i}$ be its prime factorisation, where $p_1, \dots, p_n$ are distinct primes. We recall that the *radical* of a is defined as $\mathrm{rad}(a) \coloneqq \prod_{i=1}^n p_i$. We introduce the notion of *shifted radical* of a, defined as $\mathrm{S}(a) \coloneqq \prod_{i=1}^n (p_i - 1)$. Note that the map $x \mapsto \mathrm{S}(x)$ over $\mathbb{N}$ corresponds to the sequence A173557 in Sloane's *On-Line Encyclopedia of Integer Sequences* (OEIS).

By definition of Euler's totient function, we have $\Phi(a) = \frac{a}{\mathrm{rad}(a)} \mathrm{S}(a)$. Let us state a few properties of the shifted radical that follow directly from its definition. For every $a, b \in \mathbb{N} \setminus \{0\}$:

A. $\mathrm{S}(a) \leq a$;
B. if $a \mid b$ then $\mathrm{S}(a) \mid \mathrm{S}(b)$;
C. $\mathrm{S}(a^k \cdot b) = \mathrm{S}(a \cdot b)$;
D. $\mathrm{S}(a \cdot b) \mid \mathrm{S}(a) \cdot \mathrm{S}(b)$. More interestingly, note that $\mathrm{S}(a \cdot b) = \mathrm{S}(\mathrm{lcm}(a, b)) = \frac{\mathrm{S}(a) \cdot \mathrm{S}(b)}{\mathrm{S}(\gcd(a,b))}$.

We also need the notion of the *iterated shifted radical* function $\mathbb{S}(x, k)$:

$$\begin{aligned} \mathbb{S}(x, 0) &\coloneqq 1 \\ \mathbb{S}(x, k+1) &\coloneqq \mathrm{S}(x \cdot \mathbb{S}(x, k)). \end{aligned}$$

Regarding this function, we need the following two properties (Lemmas 38 and 39).

▶ **Lemma 38.** *For all $n \in \mathbb{N} \setminus \{0\}$ and $k \in \mathbb{N}$, $\mathbb{S}(n, k)$ divides $\mathbb{S}(n, k+1)$.*

Proof. By induction on k.

base case $k = 0$: $\mathbb{S}(n, 0) = 1$ divides $\mathrm{S}(n) = \mathbb{S}(n, 1)$.
induction step $k \geq 1$:

$$\begin{aligned}
\mathbb{S}(n, k) &= \mathrm{S}(n \cdot \mathbb{S}(n, k-1)) \\
&\mid \ \mathrm{S}(n \cdot \mathbb{S}(n, k)) && \text{by I.H. and } a \mid b \text{ implies } \mathrm{S}(a) \mid \mathrm{S}(b) \\
&= \mathbb{S}(n, k+1).
\end{aligned}$$
◀

▶ **Lemma 39.** *For all $n \in \mathbb{N} \setminus \{0\}$ and $k \in \mathbb{N}$, $\mathbb{S}(n, k) \leq n^k$.*

Proof. By induction on k.
base case $k = 0$: $\mathbb{S}(n, 0) = 1 = n^0$.
induction step $k \geq 1$:

$$\begin{aligned}
\mathbb{S}(n, k) &= \mathrm{S}(n \cdot \mathbb{S}(n, k-1)) \\
&\leq \mathrm{S}(n) \cdot \mathrm{S}(\mathbb{S}(n, k-1)) && \text{by } \mathrm{S}(a \cdot b) \leq \mathrm{S}(a) \cdot \mathrm{S}(b) \\
&\leq n \cdot \mathbb{S}(n, k-1) \leq n^k && \text{by } \mathrm{S}(a) \leq a \text{ and I.H.}
\end{aligned}$$
◀

The proof of Lemma 7 uses an auxiliary recurrence defined in the next lemma.

▶ **Lemma 40.** *Let $\alpha \geq 1$ be in $\mathbb{N}$. Let the integer sequence $d_1, d_2, \ldots$ be defined by the recurrence $d_1 := \alpha \cdot S(\alpha)$ and $d_{i+1} := \alpha \cdot d_i \cdot S(d_i)$ for $i \geq 1$. For all $i \in \mathbb{N} \setminus \{0\}$, $d_i = \alpha^i \cdot \prod_{k=1}^{i} \mathbb{S}(\alpha, k)$.*

Proof. By induction on $i \geq 1$.
base case $i = 1$. $d_1 = \alpha \cdot \mathrm{S}(\alpha) = \alpha^1 \cdot \mathbb{S}(\alpha, 1)$.
induction step. Assume that $d_{i-1} = \alpha^{i-1} \cdot \prod_{k=1}^{i-1} \mathbb{S}(\alpha, k)$ for some $i > 1$. Then we have:

$$\begin{aligned}
d_i = \alpha \cdot d_{i-1} \cdot \mathrm{S}(d_{i-1}) &= \alpha^i \cdot \prod_{k=1}^{i-1} \mathbb{S}(\alpha, k) \cdot \mathrm{S}(\alpha^{i-1} \cdot \prod_{k=1}^{i-1} \mathbb{S}(\alpha, k)) && \text{by I.H.} \\
&= \alpha^i \cdot \prod_{k=1}^{i-1} \mathbb{S}(\alpha, k) \cdot \mathrm{S}(\alpha \cdot \prod_{k=1}^{i-1} \mathbb{S}(\alpha, k)) && \text{by } \mathrm{S}(a^k \cdot b) = \mathrm{S}(a \cdot b) \\
&= \alpha^i \cdot \prod_{k=1}^{i-1} \mathbb{S}(\alpha, k) \cdot \mathrm{S}(\alpha \cdot \mathbb{S}(\alpha, i-1)) && \text{by Lemma 38} \\
&= \alpha^i \cdot \prod_{k=1}^{i-1} \mathbb{S}(\alpha, k) \cdot \mathbb{S}(\alpha, i) && \text{by def. of } \mathbb{S}(x, n) \\
&= \alpha^i \cdot \prod_{k=1}^{i} \mathbb{S}(\alpha, k).
\end{aligned}$$
◀

We are now ready to prove Lemma 7.

▶ **Lemma 7.** *Let $\alpha \geq 1$ be in $\mathbb{N}$. Consider the integer sequence $b_0, b_1, \ldots$ given by the recurrence $b_0 := 1$ and $b_{i+1} := \mathrm{lcm}(b_i, \Phi(\alpha \cdot b_i))$. For every $i \in \mathbb{N}$, $b_i \leq \alpha^{2 \cdot i^2}$.*

Proof. We prove that for every $i \in \mathbb{N} \setminus \{0\}$, b_i divides d_i (defined in Lemma 40).
The proof is by induction on $i \geq 1$.

base case $i = 1$. We have $b_1 = \mathrm{lcm}(b_0, \Phi(\alpha \cdot b_0)) = \mathrm{lcm}(1, \Phi(\alpha \cdot 1)) = \Phi(\alpha) = \frac{\alpha}{\mathrm{rad}(\alpha)} \cdot \mathrm{S}(\alpha)$.
Therefore, b_1 divides $\alpha \cdot \mathrm{S}(\alpha) = d_1$, proving the base case.

induction step. Assume that b_i divides d_i. A well-known property of Euler's totient function applied to the product $\alpha \cdot d_i$ gives us the equation

$$\Phi(\alpha \cdot d_i) = \gcd(\alpha, d_i) \cdot \Phi(\operatorname{lcm}(\alpha, d_i)) = \alpha \cdot \Phi(d_i), \tag{36}$$

where the second equality follows from the fact that α is a divisor of d_i. Now we show that b_{i+1} divides d_{i+1}.

$$\begin{aligned}
b_{i+1} &= \operatorname{lcm}(b_i,\, \Phi(\alpha \cdot b_i)) && \\
&\mid\ \operatorname{lcm}(d_i,\, \Phi(\alpha \cdot d_i)) && \text{by I.H. and } a \mid b \text{ implies } \Phi(a) \mid \Phi(b)\\
&= \operatorname{lcm}(d_i,\, \alpha \cdot \Phi(d_i)) && \text{by Equation (36)}\\
&\mid\ \alpha \cdot \operatorname{lcm}(d_i,\, \Phi(d_i)) && \\
&= \alpha \cdot \operatorname{lcm}(d_i,\, \frac{d_i}{\operatorname{rad}(d_i)} \cdot \mathrm{S}(d_i)) && \text{def. of } \Phi\\
&\mid\ \alpha \cdot d_i \cdot \mathrm{S}(d_i) = d_{i+1}. &&
\end{aligned}$$

Therefore, we conclude that for every $i \geq 1$, b_i is bounded as:

$$\begin{aligned}
b_i \leq d_i &= \alpha^i \cdot \prod_{k=1}^{i} \mathbb{S}(\alpha, k) && \text{by Lemma 40}\\
&\leq \alpha^i \cdot \mathbb{S}(a, i)^i \leq \alpha^i \cdot \alpha^{i^2} \leq \alpha^{2 \cdot i^2}. && \text{by Lemmas 38 and 39}
\end{aligned}$$

◀

We are now ready to complete the complexity analysis of Algorithm 2.

▶ **Proposition 5.** *Algorithm 2 (LinExpSat) runs in non-deterministic polynomial time.*

Proof. Consider the execution of Algorithm 2 on an input $\varphi(x_1, \dots, x_n)$, with $n \geq 1$. For $i \in [0, n]$, let (φ_i, θ_i) be the pair of system and ordering obtained after the ith iteration of the **while** loop of line 3, where $\varphi_0 = \varphi$ and θ_0 is the ordering guessed in line 2.

By virtue of Lemma 6, in order to prove the proposition it suffices to show that, for every $i \in [0, n]$, the formula φ_i is of bit size polynomial in the bit size of the input formula φ. Indeed, recall that the **while** loop of line 3 is executed n times, and during its ith iteration, the loop body runs in non-deterministic polynomial time in the bit size of φ_i.

To prove that the bit size of φ_i is polynomial in the bit size of φ, it suffices to iterate i times the bounds stated in Lemma 6. That is, we establish the following parameter table:

	$\#lst(\cdot,\cdot)$	$\#(\cdot)$	$\lVert\cdot\rVert_{\mathfrak{L}}$	$mod(\cdot)$	$\lVert\cdot\rVert_1$
φ	$\theta\colon\ \ell$	s	a	1	c
φ_i	$\theta_i\colon\ \ell + 5 \cdot i^2$	$s + 16 \cdot i \cdot (i+2)^2 + 2 \cdot i \cdot \ell$	$3^i a$	β_i	$(i+3)^{8 \cdot i}(\log(a+1) + c)$

where $\beta_i \in [1, (3^i \cdot a + 2)^{18 \cdot i^5}]$.

Observe that the parameter table states that $\#\varphi_i$, $\log(\lVert\varphi_i\rVert_1)$ and $\log(mod(\varphi_i))$ are bounded polynomially in the bit size of φ. This does indeed imply that the bit size of φ_i is polynomial in the bit size of φ, since the former is in $O(\#\varphi_i \cdot n^2 \cdot \log(\lVert\varphi_i\rVert_1) \cdot \log(mod(\varphi_i)))$.

To show that the parameter table above is correct, we first prove the $\lVert\cdot\rVert_{\mathfrak{L}}$ column by induction on i. Then, we switch to $mod(\cdot)$, which is proved by using Lemma 7. Lastly, we tackle the cases of the remaining parameters by induction on i.

Linear norm. The base case is simple: $\lVert\varphi_0\rVert_{\mathfrak{L}} = \lVert\varphi\rVert_{\mathfrak{L}} \leq a$. In the induction step, assume $\lVert\varphi_i\rVert_{\mathfrak{L}} \leq 3^i a$ by induction hypothesis ($i \geq 0$). By Lemma 6, $\lVert\varphi_{i+1}\rVert_{\mathfrak{L}} \leq 3^{i+1} a$.

Least common multiple of the divisibility constraints. First, let us bound the value of α_i in the statement of Lemma 6. In that lemma, α_i is stated to be in $[1, (3 \cdot a' + 2)^{(i+3)^2}]$ where a' is an upper bound to $\|\varphi_i\|_{\mathfrak{L}}$. Hence, from the previous step of the proof, we have $\alpha_i \in [1, (3^{i+1}a + 2)^{(i+3)^2}]$. Now, following again the statement of Lemma 6, we conclude that there is a sequence of integers $b_0, b_1, \ldots$ such that

$$\begin{aligned} &\mathit{mod}(\varphi_0) \mid b_0 && b_0 = 1 \\ &\mathit{mod}(\varphi_{i+1}) \mid b_{i+1} && b_{i+1} = \operatorname{lcm}(b_i, \Phi(\alpha_i \cdot b_i)). \end{aligned}$$

We almost have the sequence in Lemma 7, the only difference being that the value α_i changes for each b_i. Let $\overline{\alpha}_0 \coloneqq 1$ and for every positive $j \in \mathbb{N}$, let $\overline{\alpha}_j \coloneqq \operatorname{lcm}(\alpha_0, \ldots, \alpha_{j-1})$. For every $i \in \mathbb{N}$, we consider the sequence of $i+1$ terms $c_0, \ldots, c_i$ given by

$$c_0 \coloneqq 1, \qquad c_{j+1} \coloneqq \operatorname{lcm}(c_j, \Phi(\overline{\alpha}_i \cdot c_j)) \text{ for } j \in [0, i-1].$$

By Lemma 7, we conclude that $c_i \leq (\overline{\alpha}_i)^{2 \cdot i^2}$.
Let us show that, for every $j \in [0, i]$, $b_j \mid c_j$. This is done with a simple induction hypothesis. The base case for b_0 and c_0 is trivial. For the induction step, pick $j \in [0, i-1]$. We have:

$$\begin{aligned} b_{j+1} &= \operatorname{lcm}(b_j, \Phi(\alpha_j \cdot b_j)) \\ &\mid \operatorname{lcm}(c_j, \Phi(\alpha_j \cdot c_j)) && \text{by I.H. and from “} a \mid b \text{ implies } \Phi(a) \mid \Phi(b)\text{”} \\ &\mid \operatorname{lcm}(c_j, \Phi(\overline{\alpha}_i \cdot c_j)) && \alpha_j \text{ divides } \overline{\alpha}_i \text{, and again “} a \mid b \text{ implies } \Phi(a) \mid \Phi(b)\text{”} \\ &= c_{j+1}. \end{aligned}$$

Then, from the definition of $\overline{\alpha}_i$ and the bound $c_i \leq (\overline{\alpha}_i)^{2 \cdot i^2}$, we conclude that $\mathit{mod}(\varphi_i)$ divides some $\beta_i \in [1, (3^i \cdot a + 2)^{18 \cdot i^5}]$.

We now show by induction on i the bounds on the remaining three parameters in the table. The base case $i = 0$ is trivial, as $\varphi_0 = \varphi$. For the induction step, assume that the bounds in the table are correct for φ_i. We show that then φ_{i+1} follows its respective table.

Least significant terms. By induction hypothesis, $\#\mathit{lst}(\varphi_i, \theta_i) \leq \ell + 5 \cdot i^2$. We apply the bounds in Lemma 6, and obtain

$$\begin{aligned} \#\mathit{lst}(\varphi_{i+1}, \theta_{i+1}) &\leq (\ell + 5 \cdot i^2) + 2 \cdot (i+2) \\ &\leq \ell + 5 \cdot i^2 + 2 \cdot i + 4 \\ &\leq \ell + 5 \cdot (i^2 + 2 \cdot i + 1) \\ &\leq \ell + 5 \cdot (i+1)^2. \end{aligned}$$

Number of constraints. By induction hypothesis $\#\varphi_i \leq s + 16 \cdot i \cdot (i+2)^2 + 2 \cdot i \cdot \ell$ and, as in the previous case, $\#\mathit{lst}(\varphi_i, \theta_i) \leq \ell + 5 \cdot i^2$. We apply the bounds in Lemma 6, and obtain

$$\begin{aligned} \#\varphi_{i+1} &\leq (s + 16 \cdot i \cdot (i+2)^2 + 2 \cdot i \cdot \ell) + 6(i+2) + 2 \cdot (\ell + 5 \cdot i^2) \\ &= s + (16 \cdot i \cdot (i+2)^2 + 6(i+2) + 10 \cdot i^2) + 2 \cdot (i+1) \cdot \ell \\ &\leq s + (16 \cdot i \cdot (i+2)^2 + 16(i+2)^2) + 2 \cdot (i+1) \cdot \ell \\ &\leq s + 16 \cdot (i+1) \cdot (i+3)^2 + 2 \cdot (i+1) \cdot \ell. \end{aligned}$$

1-norm. By applying the induction hypothesis, we have $\|\varphi_i\|_1 \leq (i+3)^{8\cdot i}(\log(a+1)+c)$ and $mod(\varphi_i) \leq (3^i \cdot a + 2)^{18\cdot i^5}$. Then, following Lemma 6,

$$\begin{aligned}
\|\varphi_{i+1}\|_1 &\leq 2^5(i+3)^2\big((i+3)^{8\cdot i}(\log(a+1)+c)+2\big) + 4\cdot\log((3^i\cdot a+2)^{18\cdot i^5}) \\
&\leq 2^5(i+3)^{8i+2}(\log(a+1)+c) \;+\; 2^6(i+3)^2 \;+\; 4\cdot 18\cdot 3\cdot i^6\log(a+1) \\
&\qquad\qquad\qquad\qquad\text{note: } \log(3a+2)\leq 3\log(a+1) \text{ for } a\geq 1 \\
&\leq 3\cdot 2^5(i+3)^{8i+2}(\log(a+1)+c) \\
&\qquad\qquad\qquad\qquad\text{last two summands are smaller than the first} \\
&\leq (i+3)^{8(i+1)}(\log(a+1)+c).
\end{aligned}$$

◀