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

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.132

Category Track B: Automata, Logic, Semantics, and Theory of Programming

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].

[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.

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, 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 $\{(x, y) \in \mathbb{N} \times \mathbb{N} : y \text{ is the largest power of 2 that divides } x\}$. Importantly, the predicate V_2 can be replaced in this definition with the function $x \bmod 2^y$, because the two are mutually expressible:

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

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.

[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.

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 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 [41], 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, 41], 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, 39] 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 [41] 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, 38] 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 [39, Corollary 4.3]. We develop his observation in Section 4.

3 Preliminaries

We usually write $a, b, c, \ldots$ for integers, $x, y, z, \ldots$ for integer variables, and $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \ldots$ and $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \ldots$ 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 $\varphi, \chi, \psi, \ldots$, 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 $[n, m] \coloneqq \{n, n + 1, \ldots, m\}$. 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 $\rho, \sigma, \tau, \ldots$, 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.

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, 37]. 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 [39]. 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 [39] 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.
Output of each branch ($\beta$): 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 $\varphi$ 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 $\varphi$ contains $x$ then continue
5:     guess $ax + \tau = 0$ (with $a \neq 0$) $\leftarrow$ equality in $\varphi$ that contains $x$
6:     $p \leftarrow \ell$;   $\ell \leftarrow a$                 ▷ previous and current lead coefficients
7:     if $\tau$ contains a slack variable $y$ not assigned by $s$ then
8:         guess $v \leftarrow$ integer in $[0, |a| \cdot mod(\varphi) - 1]$
9:         append $[v \,/\, y]$ to $s$
10:    $\varphi \leftarrow \varphi[[\frac{-\tau}{a} \,/\, x]]$
11:    divide each constraint in $\varphi$ by $p$   ▷ in divisibility constraints, both sides are affected
12:    $\varphi \leftarrow \varphi \wedge (a \mid \tau)$
13: foreach equality $\eta = 0$ of $\varphi$ 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 $\varphi$
16: foreach $x$ in $\boldsymbol{x}$ that occurs in $\varphi$ do
17:     guess $r \leftarrow$ integer in $[0, mod(\varphi) - 1]$
18:     $\varphi \leftarrow \varphi[r \,/\, x]$
19: return $\varphi$
```

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 $\varphi$ contains $x$ **then continue**
04:     **let** $ax + \tau = 0$ (with $a \neq 0$) $\leftarrow$ an arbitrary equality in $\varphi$ that contains $x$
05:     $p \leftarrow \ell$;   $\ell \leftarrow a$
06:     $\varphi \leftarrow \varphi[[\frac{-\tau}{a} \,/\, x]]$
07:     divide each constraint in $\varphi$ by $p$
08: **return** $\varphi$
```

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 $2^{x_n} \geq 2^{x_{n-1}} \geq \cdots \geq 2^{x_0} = 1$, 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 $\sim \in \{<, \leq, =, \equiv_d \colon d \geq 1\}$ and τ is an *quotient term (induced by* θ*)*, that is, a term of the form

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

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 $\sim \in \{<, \leq, =, \equiv_d \colon d \geq 1\}$.

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, 3, 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 $\varphi \wedge \theta$ 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 $\theta$ is not the ordering $(2^{x_0} = 1)$ do
4:      $2^x \leftarrow$ leading exponential term of $\theta$                  ▷ in the $i$-th iteration, $2^x$ is $t_i$
5:      $2^y \leftarrow$ successor of $2^x$ in $\theta$                           ▷ and $2^y$ is $t_{i+1}$
6:      $\varphi \leftarrow \varphi[w \,/\, (w \bmod 2^x) : w$ is a variable$]$
7:      $\boldsymbol{z} \leftarrow$ all variables $z$ in $\varphi$ such that $z$ is $x$ or $z$ does not appear in $\theta$
8:      foreach $z$ in $\boldsymbol{z}$ do                              ▷ form a quotient system induced by $\theta$
9:          let $x'$ and $z'$ be two fresh variables
10:         $\varphi \leftarrow \varphi \wedge (0 \leq z' < 2^y)$
11:         $\varphi \leftarrow \varphi[z' \,/\, (z \bmod 2^y)]$
12:         $\varphi \leftarrow \varphi[(z' \bmod 2^w) \,/\, (z \bmod 2^w) : w$ is such that $\theta$ implies $2^w \leq 2^y]$
13:         $\varphi \leftarrow \varphi[(x' \cdot 2^y + z') \,/\, z]$          ▷ replaces only the linear occurrences of $z$
14:         if $z$ is $x$ then $(x_0', z_0') \leftarrow (x', z')$        ▷ for delayed substitution, see next line
15:     $\varphi \leftarrow$ElimMaxVar$(\theta, \varphi, [x_0' \cdot 2^y + z_0' \,/\, x])$
16:     remove $2^x$ from $\theta$
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 $(\varphi_0, \theta_0) := (\varphi, \theta)$):

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. $\varphi_i \wedge \theta_i$ is equisatisfiable with $\varphi \wedge \theta$ over $\mathbb{N}$.

More precisely, writing $\bigvee_\beta \psi_\beta$ for the disjunction of all the formulae $\varphi_i \wedge \theta_i$ obtained across all non-deterministic branches, we have that $\bigvee_\beta \psi_\beta$ and $\varphi \wedge \theta$ are equisatisfiable. Therefore, whenever $\varphi \wedge \theta$ is satisfiable, (I4) holds for at least one branch. If $\varphi \wedge \theta$ 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 $\varphi \wedge \theta$ 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 $\boldsymbol{x}' \geq \boldsymbol{0}$ 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 $0 \le z < 2^y$ occurs in ψ_β.

Ensuring: $(\exists x\, \theta) \land \bigvee_\beta \psi_\beta$ is equivalent to $\exists x \exists \boldsymbol{x}'(\theta \land \varphi \land x = x' \cdot 2^y + z')$ over $\mathbb{N}$.

```
 1: let $u$ be a fresh variable                                   ▷ $u$ is an alias for $2^{x-y}$
 2: $\gamma \leftarrow \top$; $\psi \leftarrow \top$
    Elimination of $\boldsymbol{x}' \setminus x'$
 3: $\Delta \leftarrow \varnothing$                          ▷ map from linear-exponential terms to $\mathbb{Z}$
 4: foreach $(\tau \sim 0)$ in $\varphi$, where $\sim \in \{=, <, \le, \equiv_d : d \ge 1\}$ do
 5:     let $\tau$ be $(a \cdot 2^x + f(\boldsymbol{x}') \cdot 2^y + \rho)$, where $\rho$ is the least significant part of $\tau$
 6:     if $a = 0$ and $f(\boldsymbol{x}')$ is an integer then $\psi \leftarrow \psi \land (\tau \sim 0)$
 7:     else if the symbol $\sim$ belongs to $\{=, <, \le\}$ then
 8:         if $\Delta(\rho)$ is undefined then
 9:             guess $r \leftarrow$ integer in $[-\|\rho\|_1, \|\rho\|_1]$
10:             $\psi \leftarrow \psi \land ((r-1) \cdot 2^y < \rho) \land (\rho \le r \cdot 2^y)$
11:             update $\Delta$ : add the key–value pair $(\rho, r)$
12:         $r \leftarrow \Delta(\rho)$
13:         if the symbol $\sim$ is $<$ then
14:             $\sim \leftarrow \le$
15:             $r \leftarrow r + 1$                        ▷ $(v < w)$ is equivalent to $(v + 1 \le w)$
16:         $\gamma \leftarrow \gamma \land (a \cdot u + f(\boldsymbol{x}') + r \sim 0)$
17:         if the symbol $\sim$ is $=$ then $\psi \leftarrow \psi \land (r \cdot 2^y = \rho)$
18:     else                                                ▷ $\sim$ is $\equiv_d$ for some $d \in \mathbb{N}$
19:         guess $r \leftarrow$ integer in $[1, mod(\varphi)]$
20:         $\gamma \leftarrow \gamma \land (a \cdot u + f(\boldsymbol{x}') - r \sim 0)$
21:         $\psi \leftarrow \psi \land (r \cdot 2^y + \rho \sim 0)$
22: $\gamma \leftarrow$ GaussQE $(\boldsymbol{x}' \setminus x', \gamma \land \boldsymbol{x}' \ge \mathbf{0})$
23: $\gamma \leftarrow \gamma[2^u / u]$                              ▷ $u$ now is an alias for $x - y$
24: $(\chi, \gamma) \leftarrow$ SolvePrimitive$(u, x', \gamma)$
25: $\chi \leftarrow \chi[x - y / u][x' \cdot 2^y + z' / x]$        ▷ apply delayed substitution: $x$ is eliminated
    Elimination of $x'$
26: if $\chi$ is $(-x' \cdot 2^y - z' + y + c = 0)$ for some $c \in \mathbb{N}$ then
27:     guess $b \leftarrow$ integer in $[0, c]$
28:     $\gamma \leftarrow \gamma \land (x' = b)$
29:     $\psi \leftarrow \psi \land (b \cdot 2^y = -z' + y + c)$
30: else
31:     let $\chi$ be $(-x' \cdot 2^y - z' + y + c \le 0) \land (d \mid x' \cdot 2^y + z' - y - r)$, with $d, r \in \mathbb{N}$, $c \ge 3$
32:     guess $(b, g) \leftarrow$ pair of integers in $[0, c] \times [1, d]$
33:     $\gamma \leftarrow \gamma \land (x' \ge b) \land (d \mid x' - g)$
34:     $\psi \leftarrow \psi \land ((b-1) \cdot 2^y < -z' + y + c) \land (-z' + y + c \le b \cdot 2^y) \land (d \mid g \cdot 2^y + z' - y - r)$
35: assert(GaussQE$(x', \gamma)$ is equivalent to $\top$)      ▷ upon failure, Algorithm 2 returns $\bot$
36: return $\psi$
```

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 $\bot$. 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 $\varphi$ be $(\chi \land \psi)$, where $\chi$ is the conjunction of all (in)equalities from $\varphi$ 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:     $\chi \leftarrow (u = c)$
7:     $\gamma \leftarrow \varphi[2^c \,/\, 2^u]$
8: else        ▷ assuming $u \geq C$, (in)equalities in $\chi$ simplify to $\top$ or $\bot$
9:     assert($\chi$ has no equality, and in all its inequalities $2^u$ has a negative coefficient)
10:    guess $r \leftarrow$ integer in $[0, d-1]$        ▷ remainder of $2^{u-n}$ modulo $d$ when $u \geq C \geq n$
11:    assert($d \mid 2^u - 2^n \cdot r$ is satisfiable)
12:    $r' \leftarrow$ discrete logarithm of $2^n \cdot r$ base 2, modulo $d$
13:    $d' \leftarrow$ multiplicative order of 2 modulo $d$
14:    $\chi \leftarrow (u \geq C) \land (d' \mid u - r')$
15:    $\gamma \leftarrow \psi[2^n \cdot r / 2^u]$        ▷ $2^n \cdot r$ is a remainder of $2^u$ modulo $mod(\psi) = d \cdot 2^n$
16: return $(\chi, \gamma)$
```

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 [24, 16]) 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 $C' \coloneqq 3 + 2 \cdot \left\lceil \log(\frac{|b|+|c|+1}{|a|}) \right\rceil$ (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 $mod(\varphi) \mid 2^u - 2^n \cdot r$ (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, $mod(\varphi) \mid 2^u - 2^n \cdot r$ 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 $mod(\varphi)$. For example, if $mod(\varphi) = 5$, 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 $m = mod(\varphi)$ 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 $\theta(\boldsymbol{x}) = (2^{x_n} \geq \cdots \geq 2^{x_0} = 1)$ 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 $mod(\varphi)$, 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 $\|\varphi\|_1 \coloneqq \max\{\|\tau\|_1 : \tau$ 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 $mod(\varphi)$.

- The *linear norm* $\|\varphi\|_{\mathfrak{L}} \coloneqq \max\{\|\tau\|_{\mathfrak{L}} : \tau \text{ is a term appearing in an (in)equality of } \varphi\}$. 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 $\mathit{lst}(\varphi, \theta)$ defined as $\big\{ \pm\rho : \rho \text{ is the least significant part of a term } \tau \text{ appearing in an (in)equality } \tau \sim 0 \text{ of } \varphi, \text{ with respect to } \theta \big\}$. 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 $O(\#\varphi \cdot n^2 \cdot \log(\max(\|\varphi\|_1, \mathit{mod}(\varphi), 2)))$. 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 $O(\log(\max(\mathit{mod}(\gamma), \|\chi\|_1)))$.

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 $\ell, s, a, c, d \geq 1$,*

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

*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.*

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 $\mathit{mod}(\varphi_i)$, 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 $\mathit{mod}(\varphi_0) = 1$, for a given $i \in [0, n-1]$ we apply Lemma 7 with $\alpha = \operatorname{lcm}(\alpha_0, \dots, \alpha_i)$ to derive $\mathit{mod}(\varphi_{i+1}) \leq (\|\varphi\|_{\mathfrak{L}} + 2)^{O(i^6)}$. 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 $\left|\sum_{i \in I} a_i \cdot 2^{-x_i}\right| \leq \sum_{i \in I} |a_i| =: B$. 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 $2^{\sum_{i \in I} x_i}$. 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 $V_2(\tau, z) \wedge \neg(z = \sigma)$ where z is a fresh variable, $\neg(\tau = \sigma)$ becomes $(\tau < \sigma) \vee (\sigma < \tau)$, 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 $2 \cdot y = 2^v \wedge 2 \cdot (x \bmod 2^v) = 2^v$, 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 $\varphi_1 \vee \varphi_2$ 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 [40].

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 1–21, 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. Warsaw, 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 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.

38 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.

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

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

41 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.