An Introduction to the Theory of Linear Integer Arithmetic

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

Abstract

Presburger arithmetic, or linear integer arithmetic (LIA), is a logic that allows one to express linear constraints on integers: equalities, inequalities, and divisibility by nonzero $n \in \mathbb{Z}$. More formally, it is the first-order theory of integers with addition and ordering. This paper offers a short introduction: what can be expressed in this logical theory, decision problems, and automated reasoning methods.

We begin with an elementary introduction, explaining the language of linear arithmetic constraints by examples. We adopt a theoretical perspective, focusing on the decision problem: determining the truth value of a logical sentence. The following three views on Presburger arithmetic give us three effective methods for decision procedures: a view from geometry (using semi-linear sets), from automata theory (using finite automata and recognizable sets), and from symbolic computation (using quantifier elimination).

The decision problem for existential formulas of Presburger arithmetic is essentially the feasibility problem of integer linear programming. By a fundamental result due to Borosh and Treybig [*Proc. Am. Math. Soc.* 55(2), 1976] and Papadimitriou [*J. ACM* 28(4), 1981], it belongs to the complexity class NP. Echoing the three views discussed above, we sketch three proofs of this result and discuss how these ideas have been used and developed in the recent research literature.

This is a companion paper for a conference talk focused on the three views on Presburger arithmetic and their applications. The reader will require background knowledge at the level of undergraduate computer science curricula. The discussion of complexity aspects is more advanced.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Theory of computation → Logic

Keywords and phrases Logical theories of arithmetic, decision procedures

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2024.1

Category Invited Paper

Funding Supported in part by the Engineering and Physical Sciences Research Council [EP/X03027X/1].

Acknowledgements I would like to thank Christoph Haase for introducing me to arithmetic theories and all my co-authors for sharing inspiration, ideas, and kindness. I am also grateful to Ranko Lazić and Shrisha Rao for useful discussions of the Steinitz lemma and its applications. Last but not least, I would like to thank everyone who gave feedback on early drafts of this paper.

1 What is linear integer arithmetic?

Linear integer arithmetic, also known as *Presburger arithmetic,* combines linear Diophantine equations with logic (Boolean connectives and quantifiers). Intuitively, a logic is a language in which we can express things that are true or false. In the language of Presburger arithmetic we can:

- talk about integers (referred to as variables x, y, …),
- assert linear inequalities involving these integers,

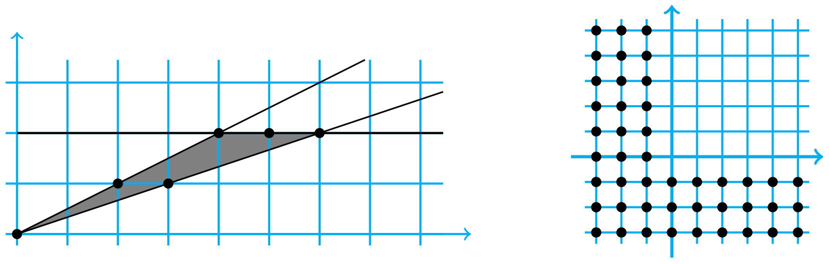

Figure 1 Black dots show integer points (assignments to x, y) that make the formulas in Eqs. (1) and (2) true. These assignments form sets T (left) and U (right), respectively. On the left, the set of solutions in $\mathbb{R}^2$ to the system of 3 inequalities in Eq. (1) is the intersection of 3 half-planes, shown in grey.

- form Boolean combinations (logical AND, OR, NOT, denoted by $\wedge$, $\vee$, $\neg$, respectively) of these assertions, and
- quantify over (all) integers, using "for all" ($\forall$) and "there exists" ($\exists$).

We develop the intuition first, deferring rigorous definitions to Section 2.

A *formula* in linear integer arithmetic is a syntactic object. A formula expresses (*defines*) a set of points with integer coordinates, that is, a subset of $\mathbb{Z}^d$ for some d. For example, the formula

$$(x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2) \tag{1}$$

defines the set $T = \{(0,0),(2,1),(3,1),(4,2),(5,2),(6,2)\}$ of integer points in a triangle. Indeed, $\{(x,y) \in \mathbb{Z}^2 : (x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2)\} = T$, see Fig. 1 (left). Informally, the formula talks about x and y, and thus defines a set in 2D. A set may be equivalently expressed by more than one logical formula. For example, the two formulas

$$(x < 0) \vee (y < 0) \qquad \text{and} \qquad \neg((x \geqslant 0) \wedge (y \geqslant 0)) \tag{2}$$

define the same set, call it U, depicted in Fig. 1 (right).

Let us consider some formulas with quantifiers. As an elementary example, the formula

$$\forall x \ [(\exists y \ (x = 2y)) \vee (\exists z \ (x = 2z + 1))] \tag{3}$$

can be interpreted as saying that every integer is either even or odd (possibly both). Notice that we could equivalently write $\forall x \ [(\exists y \ (x = 2y)) \vee (\exists y \ (x = 2y + 1))]$, because the choice of name (y or z) for the auxiliary variable does not change the meaning of the formula, as long as there is no "name clash": usage of a name that is already in use at this point in the formula.

Formally, variables that a formula φ "talks about" are called *free variables* of φ. For example, the formula

$$E(x): \quad \exists y \ (x = 2y) \tag{4}$$

talks about a single variable, x, and asserts that x is even. Note that we denoted this formula $E(x)$. More generally, we write, for instance, $\varphi(x, y, z)$ implying that φ has no free variables except x, y, and z. It is *not* implied that x, y, and z are mentioned by φ: some or even all of them may not appear in it. In the formula from Eq. (3), all variables are "quantified away", that is, these formulas have no free variables. Each of these formulas evaluates to just true or false. Formulas without free variables are called *sentences*.

▶ **Example 1.** Fix two integers $a, b > 0$. Consider the sentence

$$C_{a,b}\colon \quad \forall s\ \exists x_1\ \exists x_2\ \ s = ax_1 - bx_2.$$

We will see in Section 3 that $C_{a,b}$ is true if and only if a and b are coprime. Intuitively, $C_{a,b}$ is implied by, and in fact equivalent to, a similar formula $\forall s_1\ \forall s_2\ \exists x_1\ \exists x_2\ \ s_2 - s_1 = ax_1 - bx_2$. Rearranging the terms, we can rewrite the equation as $s_1 + ax_1 = s_2 + bx_2$. So the formula in fact asserts that the two arithmetic progressions $s_1, s_1 + a, s_1 + 2a, \ldots$ and $s_2, s_2 + b, s_2 + 2b, \ldots$ always have a number appearing in both, no matter how the initial terms s_1 and s_2 are chosen. ⌟

▶ Remark. In Example 1, a and b are fixed parameters. If they were, in fact, variables, then the given formula would not be a sentence and would talk about a and b. Denote it $C(a, b)$. Assuming $a, b > 0$, the formula $C(a, b)$ would still assert that a and b are co-prime, but would no longer be a formula of linear integer arithmetic, because of the multiplication of two variables a and x_1. Logicians would say that $C(a, b)$ is a formula in the language of rings.

All variables are always quantified over the same set, namely $\mathbb{Z}$. That is, the syntax of Presburger arithmetic disallows formulas such as $\forall x \in S\ \varphi(x)$. Can we still express such an assertion in our logic? Assuming that the set S itself can be defined in Presburger arithmetic, namely as $S = \{a \in \mathbb{Z} : \psi(a) \text{ is true}\}$ for some formula ψ with one free variable, we can write $\forall x\ (\neg\psi(x) \vee \varphi(x))$ or, equivalently, $\forall x\ (\psi(x) \to \varphi(x))$, where $\to$ denotes logical implication. Presburger arithmetic can quantify over individual numbers (*first-order* quantification), but not over sets, relations, etc.

Some assertions cannot be written as a finite Boolean combination of linear inequalities without the help of quantifiers. An example is the formula $E(x)$ from Eq. (4). It is thus often convenient to extend the syntax of the logic by assertions such as "x is even", "$x - 3y + 5$ is divisible by 7", etc. We can write them as divisibility constraints $k \mid \ldots$, reading "k divides $\ldots$", where k is a fixed nonzero integer (that is, k cannot be a variable or an expression involving variables). Yet another alternative syntax is congruences $t_1 \equiv t_2 \pmod{k}$, with t_1, t_2 linear functions. Such constraints, also known as modulo constraints, can be used alongside linear inequalities as basic building blocks of linear integer arithmetic.

▶ **Example 2.** By the Chinese remainder theorem, formula

$$(x \equiv 2 \,(\mathrm{mod}\ 3)) \wedge (x \equiv 1 \,(\mathrm{mod}\ 5)) \wedge (x \equiv 3 \,(\mathrm{mod}\ 7)) \wedge (x \geqslant 0)$$

defines the set of natural numbers congruent to 101 modulo $105 = 3 \cdot 5 \cdot 7$. An equivalent formula avoids modulo constraints at the cost of extra quantified variables: $\exists u\, \exists v\, \exists w\ (x - 2 = 3u) \wedge (x - 1 = 5v) \wedge (x - 3 = 7w) \wedge (x \geqslant 0)$. ⌟

Linear integer arithmetic, and logical theories of arithmetic more generally (not necessarily linear), provide a common framework for expressing problems from various domains: for example, many classical combinatorial optimisation problems can be encoded directly.

▶ **Example 3.** The subset sum problem asks, given natural numbers $a_1, \ldots, a_n$, whether there exists a subset that sums up to a given target, t:

$$\exists x_1 \ldots \exists x_n \bigwedge_{i=1}^{k} [(x_i = 0) \vee (x_i = 1)] \wedge \sum_{i=1}^{n} a_i x_i = t.$$

Notice that, since all quantification is over integers, the subformula in square brackets can be rewritten as $[(0 \leqslant x_i) \wedge (x_i \leqslant 1)]$, making the entire sentence an existentially quantified conjunction of equalities and inequalities. In terms of computational complexity, the subset sum problem is NP-complete [136, Section 7.5]. ⌟

▶ **Example 4.** The Frobenius coin problem asks for the largest whole amount that *cannot* be formed using coins with denominations $a_1, \dots, a_n$, all in unbounded supply [144, 5]. If the greatest common divisor of $a_1, \dots, a_n$ is 1, then such a number exists and is referred to as the Frobenius number of $a_1, \dots, a_n$, denoted $F(a_1, \dots, a_n)$. For every $x \in \mathbb{Z}$, we have $F(a_1, \dots, a_n) \leqslant x$ if and only if the following formula of Presburger arithmetic is true:

$$\Phi(x)\colon \quad \forall y\, \exists x_1 \dots \exists x_n \left[(y \leqslant x) \lor \left(\bigwedge_{i=1}^{k} (x_i \geqslant 0) \land \left(\sum_{i=1}^{n} a_i x_i = y \right) \right) \right].$$

Thus, $F(a_1, \dots, a_n)$ is the number that *satisfies* (makes true) the formula $\Phi(x) \land \neg\Phi(x-1)$ or, equivalently, the formula

$$\Phi(x) \land \forall z_1 \dots \forall z_n \left[\bigvee_{i=1}^{k} (z_i < 0) \lor \left(\sum_{i=1}^{n} a_i z_i < x \right) \lor \left(\sum_{i=1}^{n} a_i z_i > x \right) \right].$$

Deciding whether $F(a_1, \dots, a_n) \leqslant t$ is NP-hard [4] and belongs to the complexity class $\mathrm{coNP}^{\mathrm{NP}} = \Pi_2\mathrm{P}$ (see, e.g., the definition of the polynomial(-time) hierarchy [136, Section 10.3]). ⌟

Decision problems and decision procedures

Is it possible to determine the truth value of a given sentence of Presburger arithmetic? This problem is traditionally referred to as the *decision problem*:

Input: Sentence φ.
Output: Is φ true or false?

There exists an algorithm (a *decision procedure*) that solves the decision problem for Presburger arithmetic; see Section 3. Thus, one says that (the theory of) Presburger arithmetic is *decidable*. This statement requires proof, because quantifiers in φ range over an infinite set, $\mathbb{Z}$.

The existence of algorithms that solve the decision problem for this and for other logics enables the field of *automated reasoning*: we can outsource to a computer the determination of whether a formal mathematical statement (even if expressed in a relatively simple language) is true or false! At the same time, sentences of Presburger arithmetic are not deep mathematical truths: this logic is rather restrictive. Extending the syntax by allowing not just linear but arbitrary polynomial constraints makes the problem undecidable. (This follows from Gödel's incompleteness theorems and Tarski's undefinability theorem. The argument is shown in, e.g., [23, Chapter 17] and [135, Chapter 10].) In fact, even for Presburger arithmetic the worst-case computational complexity of the problem is high, meaning that big input sentences φ may require prohibitively large computation time. Still, Presburger arithmetic offers a useful language for expressing assertions that arise in applications and can be checked in an automated fashion, striking a balance between expressiveness and decidability.

Nowadays powerful software tools are available that implement decision procedures for many logical theories (involving arithmetic or not). A big class of such tools is *satisfiability modulo theories (SMT) solvers*, developed since the early 2000s (see, e.g., [2, 12, 11]). SMT solvers build on the earlier boom of Boolean satisfiability (SAT) solvers, adding to SAT more powerful logics; hence the "modulo [logical] theories" in the name.

The present paper adopts a theoretical perspective, focusing on the pure decision problem as defined above. In practice, the success of software tools depends on many other features taking theoretical ideas further (see, e.g., [26, Chapter 1]). For example, solvers can be asked to produce explanations: proofs that a given sentence is true or false. Also key is incremental solving: when a new constraint is appended to an input sentence, the solver can benefit from information gained in its previous run, instead of restarting from scratch. SMT solvers implement not only algorithms (in the strict sense of the word) for decision problems but also heuristic approaches (for decidable as well as undecidable theories). This way the tools can successfully handle formulas coming from applications with thousands of variables, avoiding the worst-case computational complexity or even undecidability.

2 Syntax and semantics (formal definitions)

We give formal definitions of syntax and semantics of linear integer arithmetic. A description of the syntax of a logic specifies which syntactic expressions are admissible ("belong" to the logic), and the semantics prescribes their meaning. The goal here is not to re-define or revise fundamental mathematical notions, but rather to determine unambiguously:
(syntax) what kind of formulas a decision procedure must be able to handle (as its input);
(semantics) what the correct output (truth value) for each possible input formula is.

Let $\mathcal{V}$ be the set (alphabet) of variables that a formula may use. A small example might have $\mathcal{V} = \{x, y\}$, but in general $\mathcal{V}$ may well be infinite.

Syntax. We first define *terms*. Intuitively, a term in our logic is an expression that can evaluate (under an assignment of values to variables) to an integer. Formally, a term is a formal expression of the form $a_0 + a_1x_1 + \ldots + a_nx_n$, where $a_0, \ldots, a_n \in \mathbb{Z}$, $x_1, \ldots, x_n \in \mathcal{V}$, and $n \geqslant 0$.

We now define formulas of the logic. Formulas, like terms, are syntactic objects. Unlike terms, the intuition is that a formula should evaluate (again under an assignment of values to variables) to true or false. The definition is in two "stages".

An *atomic formula*, or an *atom*, is either a comparison of the form $t_1 < t_2$, $t_1 \leqslant t_2$, or $t_1 = t_2$, where t_1 and t_2 are terms, or a congruence constraint of the form $t_1 \equiv t_2 \pmod{m}$, where t_1 and t_2 are terms and $m \in \mathbb{Z} \setminus \{0\}$.

▶ Remark. Although this definition excludes comparisons of the form $t_1 > t_2$ and $t_1 \geqslant t_2$, we can always rewrite such comparisons backwards: $t_2 < t_1$ and $t_2 \leqslant t_1$, respectively. Therefore, we can regard symbols $>$ and $\geqslant$ as "syntactic sugar". Similarly, a divisibility constraint of the form $k \mid t$, where $k \in \mathbb{Z} \setminus \{0\}$ and t is a term, can be rewritten as $t \equiv 0 \pmod{k}$.

The main definition is inductive. A *formula* is:
- either an atomic formula,
- or an expression of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, or $\neg\varphi$, where φ and ψ are formulas,
- or an expression of the form $\exists x\ \varphi$ or $\forall x\ \varphi$, where φ is a formula.

Only expressions of the above three kinds are considered formulas. We use brackets to disambiguate the composition of formulas from its sub-formulas. The reader can verify that all formulas from Section 1 are indeed formulas according to this definition.

Note that the third case does not restrict φ, and in particular we do not specify whether or not the variable x bound by the quantifier even appears in φ.

▶ Remark. A reader unfamiliar with mathematical logic may find it convenient to assume **no variable reuse**. That is, whenever a formula Φ contains a subformula $\exists x\ \varphi$ or $\forall x\ \varphi$, we can assume that *all* occurrences of x in Φ are within subformulas of such two forms, and moreover these subformulas cannot contain one another. This convention forbids, e.g., nested quantifiers that bind the same variable.

Semantics. We now show how to "assign meaning" to formulas, which have been defined above as purely syntactic objects.

An *assignment* is a map from $\mathcal{V}$, our alphabet of variables, to $\mathbb{Z}$. Let $\nu\colon \mathcal{V} \to \mathbb{Z}$ be an assignment and take some integer $a \in \mathbb{Z}$ and variable $x \in \mathcal{V}$. By $\nu[a/x]$ we denote another assignment, ν', that agrees with ν on all variables from $\mathcal{V}$ except x and sets $\nu(x)$ to a:

$$\nu'(u) = \begin{cases} \nu(u) & \text{if } u \in \mathcal{V} \setminus \{x\}, \\ a & \text{if } u \text{ is } x. \end{cases}$$

For example, if $\mathcal{V} = \{x, y\}$ and $\nu(x) = \nu(y) = 6$, then for $\nu' = \nu[3/y]$ we have $\nu'(x) = 6$ and $\nu'(y) = 3$.

If t is a term and ν an assignment, then by $\nu(t)$ we denote the value of t under ν. Formally, if t is $a_0{+}a_1x_1{+}\ldots{+}a_nx_n$ where $x_1, \ldots, x_n \in \mathcal{V}$, then $\nu(t) = a_0{+}a_1\nu(x_1){+}\ldots{+}a_n\nu(x_n)$. Note that $\nu(t)$ is a (specific) integer. In the example from the previous paragraph, $\nu(x{-}3y{+}5) = -7$ and $\nu[3/y](x - 3y + 5) = 2$.

We are now ready to assign truth values to formulas, given an assignment. This definition is also inductive, following the inductive definition of a formula. Let ν be an assignment. For every formula we determine whether it *holds under* ν (also: φ *is true on* ν; ν *satisfies* φ):

- Take an atomic formula $t_1 < t_2$, $t_1 \leqslant t_2$, $t_1 = t_2$, or $t_1 \equiv t_2 \pmod{m}$, where t_1 and t_2 are terms and $m \in \mathbb{Z} \setminus \{0\}$. Then $t_1 < t_2$ holds under ν if and only if $\nu(t_1) < \nu(t_2)$. Here $\nu(t_1) < \nu(t_2)$ is just a comparison between two specific numbers from $\mathbb{Z}$. The definition for the cases of $t_1 \leqslant t_2$ and $t_1 = t_2$ is analogous. For the case of $t_1 \equiv t_2 \pmod{m}$, the condition is that $\nu(t_1) - \nu(t_2)$ is a multiple of m.
- A formula $\varphi \wedge \psi$ holds under ν if both φ and ψ hold under ν. For $\varphi \vee \psi$, the condition is that at least one of φ and ψ holds. For $\neg\varphi$, the condition is that φ does not hold.
- A formula $\exists x\ \varphi$ holds under ν if for some $a \in \mathbb{Z}$ the formula φ is true under the assignment $\nu[a/x]$. A formula $\forall x\ \varphi$ holds under ν if for every $a \in \mathbb{Z}$ the formula φ is true under the assignment $\nu[a/x]$.

This definition may appear tautological (and even unnecessary), but it ensures that the meaning of formulas is determined unambiguously.

▶ Remark. In logic, the standard notation for "φ holds under ν" would be "$(\mathbb{Z}, +, \leqslant), \nu \models \varphi$", as the satisfaction relation is really a ternary relation. Here $(\mathbb{Z}, +, \leqslant)$ is the structure that we have fixed throughout: $\mathbb{Z}$ is the domain of discourse (universe), $+\colon \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is binary addition on $\mathbb{Z}$, and $\leqslant$ is the standard "less than or equal to" relation (predicate) on $\mathbb{Z}$. Intuitively, a structure "realises" the syntax of a logic, giving a "valuation" for each symbol.

An occurrence of a variable $x \in \mathcal{V}$ in a formula Φ is *free* if it lies outside all subformulas $\exists x\ \varphi$ and $\forall x\ \varphi$, and *bound* otherwise. Now, $x \in \mathcal{V}$ is a *free variable* of Φ if it has a free occurrence, and a *bound variable* otherwise. A bound variable might have no occurrences whatsoever. *Sentences* are exactly formulas with no free variables.

▶ **Proposition 5.** *The truth value of a sentence does not depend on the choice of assignment.*

Proof idea. Reduce to the following special case. Let x be a bound variable of Φ and assume $\nu_2 = \nu_1[a/x]$ for some $a \in \mathbb{Z}$. Then Φ holds under ν_1 if and only if it holds under ν_2. (In fact, Φ may be assumed to be an arbitrary formula, not necessarily a sentence.) ◀

▶ Remark. In logic, the *theory of a structure* is the set of all sentences true in this structure. Presburger arithmetic is, formally, the (first-order) theory of the structure $(\mathbb{Z}, +, \leqslant)$. Consider the sentence $\Phi\colon\ \forall x\, \exists y\ (x = 2y)$, which is false. According to this definition, Φ is therefore not a sentence of Presburger arithmetic. It is, however, a sentence (written) in the syntax of Presburger arithmetic.

In the present paper, whenever we refer to sentences of Presburger arithmetic (linear integer arithmetic), as well as of other logics, the meaning is purely syntactic. We will not use the concept of theory as such, but we have decided to mention the distinction so that the reader does not get confused when consulting literature.

Taking a formula $\varphi(x_1, \ldots, x_n)$, that is, one where all free variables are among $x_1, \ldots, x_n$, it is occasionally convenient to write $\varphi(a_1, \ldots, a_n)$ for the truth value of φ on any assignment ν such that $\nu(x_i) = a_i$ for all i. For a sentence φ, it should be clear from the context whether, when writing φ, we are referring to the syntactic object or to its truth value.

The set $\{\boldsymbol{a} \in \mathbb{Z}^n : \varphi(\boldsymbol{a}) \text{ is true}\}$ is the set *defined* by formula φ; sets for which a suitable formula exists are *definable* sets. Here and elsewhere, we use boldface letters to denote elements of $\mathbb{Z}^n$. We do this for tuples of variables too, writing $\boldsymbol{x} = (x_1, \ldots, x_n)$. Two formulas $\varphi(\boldsymbol{x})$ and $\psi(\boldsymbol{x})$ are *equivalent* if they define the same set; we write $\varphi(\boldsymbol{x}) \iff \psi(\boldsymbol{x})$.

3 Three views on linear integer arithmetic

Sets $S \subseteq \mathbb{Z}^d$ definable in linear integer arithmetic (also: *Presburger-definable* sets) can also be represented with the help of geometry, or with the help of strings. The following three representations are available.

Semi-linear sets. A set can be specified by referring to its geometric features. In an analogy, a triangle in $\mathbb{R}^2$ can be specified by referring to its vertices. In the case of linear integer arithmetic, we also need to describe periodic patterns to specify a set.

Finite automata. To represent a subset of natural numbers, we can use a finite automaton over $\{0, 1\}$ that recognises (accepts) the set of binary expansions of these numbers. This idea extends to negative integers, as well as to tuples of integers.

Logical formulas. To represent a set, we can use a formula that is true exactly for the elements of the set. We already saw this representation in Section 1, following the syntax given in Section 2.

Thus, Presburger-definable sets can be viewed in three different ways: from the perspective of geometry, automata theory, and symbolic computation, respectively. Each of the three representations can be employed to solve the decision problem for Presburger arithmetic. The three resulting *methods*, generalised appropriately, can also be used for other logics.

In this section, we describe these three views on linear integer arithmetic in more detail. Actual *algorithms* are not given in this short introduction, only the ideas behind them.

3.1 A view from geometry: semi-linear sets

Semi-linear sets are a generalisation to $\mathbb{Z}^d$ of ultimately periodic sets of natural numbers. Let $S \subseteq \mathbb{N}$.[1] The set S is called:

- *periodic* if there is $p > 0$ such that, for all $x \in \mathbb{N}$, we have $x \in S$ if and only if $x + p \in S$;
- *ultimately periodic* if there are N and $p > 0$ such that, for all $x \geqslant N$, we have $x \in S$ if and only if $x + p \in S$.

We call the numbers p and N the *period* and the *offset*, respectively. A multiple of a period is also a period, and any number exceeding an offset is also an offset.

▶ **Example 6.** In Fig. 2, the top set is periodic with period 3, and the other two sets are not periodic. The top two sets are ultimately periodic, also with period 3. ⌟

[1] Whether $0 \in \mathbb{N}$ or not is a matter of convention. In logic, it is more common to include zero in the set of natural numbers. We follow this convention.

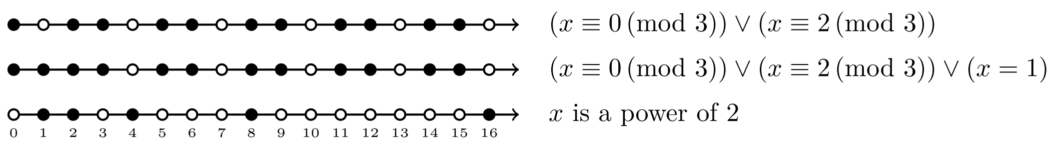
$(x \equiv 0 \pmod 3) \vee (x \equiv 2 \pmod 3)$
$(x \equiv 0 \pmod 3) \vee (x \equiv 2 \pmod 3) \vee (x = 1)$
x is a power of 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Figure 2 Three sets of natural numbers as sets of points on the number line with coordinate $x \in \mathbb{N}$. Black and white dots show numbers included in and excluded from the set, respectively. The top two sets are definable in linear integer arithmetic and the bottom set is not.

For the following Proposition, our definition of an arithmetic progression is a set

$$\{a, a+m, a+2m, \ldots\} = \{x \in \mathbb{Z} : \exists t \in \mathbb{Z}\ (x = a + tm) \wedge (t \geqslant 0)\}, \qquad a, m \in \mathbb{N}. \tag{5}$$

A special case is a singleton, if $m = 0$. From Eq. (5), or by the formula $(x \equiv a \pmod m) \wedge (x \geqslant a)$, every arithmetic progression is definable in linear integer arithmetic.

▶ **Proposition 7.** *A set $S \subseteq \mathbb{N}$ is ultimately periodic if and only if it is a union of finitely many arithmetic progressions.*

Ultimately periodic sets are closed under the following operations: complement (because p and N stay unchanged); intersection (the least common multiple of the two periods is a new period, and the maximum of the two offsets is a new offset); and union (e.g., by De Morgan's law).

There is more than one way to generalise ultimately periodic sets to higher dimensions. For example, should the two sets shown in Fig. 1 be considered ultimately periodic or not? For us, the most useful analogue will be the following definition due to Parikh [114], which is probably the most inclusive.

We first define an analogue of arithmetic progressions. A set $S \subseteq \mathbb{Z}^d$ is called *linear* if there is $\boldsymbol{b} \in \mathbb{Z}^d$ and a finite set $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\} \subseteq \mathbb{Z}^d$ such that

$$S = L(\boldsymbol{b}, P) \overset{\text{def}}{=} \left\{ \boldsymbol{b} + \sum_{i=1}^{k} \lambda_i \boldsymbol{p}_i : \lambda_1, \ldots, \lambda_k \in \mathbb{N} \right\}. \tag{6}$$

The term "linear" is traditional; the integer programming community uses the term "integer cone" (if $\boldsymbol{b}$ is $\boldsymbol{0} \overset{\text{def}}{=} (0, \ldots, 0)$) instead (see, e.g., [48, 80]). Notice that k may be different from d. If $P = \varnothing$, then $L(\boldsymbol{b}, P) = \{\boldsymbol{b}\}$.

A set is *semi-linear* if it is a union of finitely many linear sets. Points $\boldsymbol{b}$ are referred to as *base points*, *offsets*, or *constants*. Vectors $\boldsymbol{p} \in P$ are *period vectors*, or *periods*. Base points and periods can be collectively referred to as *generators*, and the representation $S = \bigcup_{i \in I} L(\boldsymbol{b}_i, P_i)$ as *generator representation* of S. Generator representation is not unique, unless S is finite.

▶ **Example 8.** Linear set $L(\boldsymbol{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\})$ with $\boldsymbol{p}_1 = (2, 1)$ and $\boldsymbol{p}_2 = (1, 2)$ is shown in Fig. 3 (left). Both sets in Fig. 1 are semi-linear. Indeed, $T = \bigcup_{\boldsymbol{t} \in T} L(\{\boldsymbol{t}\}, \varnothing)$ and $U = L(\{-\boldsymbol{e}_1\}, \{-\boldsymbol{e}_1, \boldsymbol{e}_2, -\boldsymbol{e}_2\}) \cup L(\{-\boldsymbol{e}_2\}, \{-\boldsymbol{e}_2, \boldsymbol{e}_1, -\boldsymbol{e}_1\})$, where $\boldsymbol{e}_1 = (1, 0)$ and $\boldsymbol{e}_2 = (0, 1)$. ⌟

The following theorem shows that nice closure properties of ultimately periodic sets in $\mathbb{N}$ extend to semi-linear sets in $\mathbb{Z}^d$. As it turns out, in dimension 1, ultimately periodic sets are exactly subsets of $\mathbb{N}$ definable in linear integer arithmetic. In higher dimension, definable sets are exactly semi-linear sets.

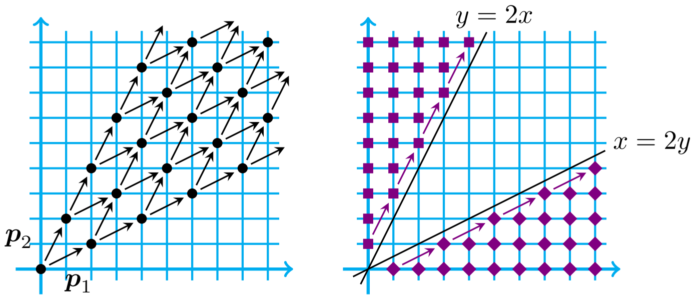

Figure 3 Five sets forming a partition of $\mathbb{N}^2$, for Example 11. Left: linear set $L(\mathbf{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\})$ with $\boldsymbol{p}_1 = (2, 1)$ and $\boldsymbol{p}_2 = (1, 2)$. Middle: linear sets V_1 (in diamond shape) and V_2 (in square shape). Period vectors not shown are unit vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$, respectively. Right: linear sets W_1 (in empty blue circles) and W_2 (in empty green squares), which are shifts of $L(\mathbf{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\})$ by $(1, 1)$ and $(2, 2)$, respectively.

▶ **Theorem 9** (Ginsburg and Spanier [57, 58]). *For a set $S \subseteq \mathbb{Z}^d$, the following are equivalent:*

(a) *$S = \bigcup_{i \in I} L(\boldsymbol{b}_i, P_i)$ for some finite set I, offsets $\boldsymbol{b}_i \in \mathbb{Z}^d$ and finite sets of periods $P_i \subseteq \mathbb{Z}^d$;*

(b) *$S = \{ (a_1, \ldots, a_d) \in \mathbb{Z}^d : \varphi(a_1, \ldots, a_d)$ is true$\}$ for some formula φ of linear integer arithmetic.*

Moreover, this equivalence is effective: there exist algorithms for conversion between the generator representation in (a) *and the formula in* (b)*.*

Proof idea.

(a) ⇒ (b). For given $\boldsymbol{b}$ and $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\}$, the condition $\boldsymbol{x} \in L(\boldsymbol{b}, P)$ can be expressed using a formula with k existentially quantified variables for $\lambda_1, \ldots, \lambda_k$ in Eq. (6). The quantifier-free part of the formula is a conjunction of d equalities, one per coordinate, and d inequalities $\lambda_i \geqslant 0$. A union of $|I|$ sets corresponds to a disjunction of k formulas.

(b) ⇒ (a). The proof is by induction on the structure of the formula φ, following the definition of the syntax from Section 2.

- For an atomic formula (inequality, equality, or congruence) we construct the generator representation directly. (It suffices to prove this for a non-strict inequality, because other atoms can be avoided. Indeed: $t_1 < t_2$ if and only if $\neg(t_1 \geqslant t_2)$; $t_1 = t_2$ if and only if $(t_1 \leqslant t_2) \wedge (t_2 \leqslant t_1)$; and $t_1 \equiv t_2 \pmod{m}$ if and only if $\exists x\ (t_1 - t_2 = mx)$.)
- For formulas of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, or $\neg\varphi$, we prove the closure of the family of semi-linear sets under Boolean operations.
- For an existential quantifier, let $\boldsymbol{y} = (y_1, \ldots, y_d)$ be the vector of free variables. Observe that the set $\{\boldsymbol{b} \in \mathbb{Z}^d$: formula $\exists x\ \varphi(x, \boldsymbol{y})$ holds on $\boldsymbol{b}\}$ can be obtained from the set $\{(a, \boldsymbol{b}) \in \mathbb{Z} \times \mathbb{Z}^d$: formula $\varphi(x, \boldsymbol{y})$ holds on $(a, \boldsymbol{b})\}$ by orthogonal projection: namely by removing the x-component (coordinate) from each element. In the generator representation, we simply cross out this component in all generators. Universal quantifiers can be handled by observing that a formula $\forall x\ \varphi$ is equivalent to $\neg\exists x\ \neg\varphi$.

In truth, this plan requires a tweak: we need to handle *systems* of inequalities as a primitive. (We sketch the construction in Section 5.1.) This is because proofs of the closure of semi-linear sets under intersection use semi-linearity of sets of solutions to such systems. ◀

The algorithm (b) ⇒ (a) of Theorem 9, run on sentences, is a decision procedure for linear integer arithmetic.

▶ **Example 10.** Let us follow the sketch above for the co-primality formula $C_{a,b}$ from Example 1, Section 1. To make the argument concrete, fix $a = 4$ and $b = 6$.

The first step is to find a generator representation of the set of solutions in x_1, x_2, s to $s = 4x_1 - 6x_2$. A suitable one is $L(\mathbf{0}, \{-\boldsymbol{v}_1, \boldsymbol{v}_1, -\boldsymbol{v}_2, \boldsymbol{v}_2\})$ with $\boldsymbol{v}_1 = (1, 0, 4)$ and $\boldsymbol{v}_2 = (0, 1, -6)$, where the coordinates are written in the order x_1, x_2, s. Indeed, it is clear that all vectors of this linear set are solutions; conversely, for a solution (x_1, x_2, s) we have $x_1 \cdot \boldsymbol{v}_1 + x_2 \cdot \boldsymbol{v}_2 = (x_1, x_2, 4x_1 - 6x_2) = (x_1, x_2, s)$, and thus $(x_1, x_2, s) \in L(\mathbf{0}, \{-\boldsymbol{v}_1, \boldsymbol{v}_1, -\boldsymbol{v}_2, \boldsymbol{v}_2\})$.

In the second step, we handle the existential quantifiers that bind x_1 and x_2. We cross out the corresponding coordinates, which leads to the set $S_1 \stackrel{\text{def}}{=} L(0, \{-4, 4, -6, 6\}) = \{4z_1 + 6z_2 : z_1, z_2 \in \mathbb{Z}\}$. To handle the universal quantifier, we rely on the "equivalence" $\forall = \neg\exists\neg$: indeed, the formula $C_{4,6}$ is true if and only if the set $\mathbb{Z} \setminus S_1$ is empty. To complement S_1, observe that $S_1 \cap \mathbb{N}$ is periodic with period 4 as well as with period 6. It is therefore periodic with period $\gcd(4, 6) = 2$. The same is true for $S_1 \cap \{-n : n \in \mathbb{N}\}$. Since $0 \in S_1$ and $\pm 1 \notin S_1$, we have $S_1 = L(0, \{-2, 2\})$ and $\mathbb{Z} \setminus S_1 = L(1, \{-2, 2\}) \neq \varnothing$. Thus, $C_{4,6}$ is false. ⌟

The construction of a semi-linear set for an inequality $a_0 + a_1x_1 + \ldots + a_dx_d \leqslant 0$, where all $a_i \in \mathbb{Z}$, generalises what we saw for equality $s = 4x_1 - 6x_2$ in Example 10.

The following example illustrates the complementation of a semi-linear set.

▶ **Example 11.** Consider $L(\mathbf{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\})$ with $\boldsymbol{p}_1 = (2, 1)$ and $\boldsymbol{p}_2 = (1, 2)$, in Fig. 3. Its complement can be decomposed as $\mathbb{Z}^2 \setminus L(\mathbf{0}, \{\boldsymbol{p}_1, \boldsymbol{p}_2\}) = U \cup V_1 \cup V_2 \cup W_1 \cup W_2$, where:

$$\begin{aligned}
U &= \{(x, y) \in \mathbb{Z}^2 : (x < 0) \vee (y < 0)\} \qquad \text{(Example 8)},\\
V_1 &= \{(x, y) \in \mathbb{Z}^2 : (y \geqslant 0) \wedge (2y < x)\} = L(\boldsymbol{e}_1, \{\boldsymbol{e}_1, \boldsymbol{p}_1\}),\\
V_2 &= \{(x, y) \in \mathbb{Z}^2 : (x \geqslant 0) \wedge (2x < y)\} = L(\boldsymbol{e}_2, \{\boldsymbol{e}_2, \boldsymbol{p}_2\}),\\
W_1 &= L((1, 1), \{\boldsymbol{p}_1, \boldsymbol{p}_2\}), \qquad \text{and} \qquad W_2 = L((2, 2), \{\boldsymbol{p}_1, \boldsymbol{p}_2\}).
\end{aligned}$$

Set U is shown in Fig. 1 (right), and sets V_1, V_2, W_1, W_2 in Fig. 3. ⌟

Let us introduce the Minkowski sum notation for sets: $A + B \stackrel{\text{def}}{=} \{a + b : a \in A, b \in B\}$. We write $a + B$ instead of $\{a\} + B$. In Example 11, sets $W_i = (i, i) + L(\mathbf{0}, P)$ for $i \in \{1, 2\}$ are shifts of the set $L(\mathbf{0}, P)$, where $P = \{\boldsymbol{p}_1, \boldsymbol{p}_2\}$. Moreover, these three sets form a partition of the set of integer points in the sector $\{(x, y) \in \mathbb{R}^2 : (x \leqslant 2y) \wedge (y \leqslant 2x)\}$; see Fig. 3 (right). The sector itself is definable in linear *real* arithmetic. We see on this example a useful rule of thumb: reasoning about integer points combines reasoning about linear *real* arithmetic (intuitively: geometric constraints in $\mathbb{R}^d$) and reasoning about integer *lattices* (intuitively: divisibility properties, or periodic patterns; see, e.g., [101, Section 2.2]).

Further reading. Over the years, multiple algorithms have appeared for operations on semi-linear sets. Early papers [57, 58] and a monograph [56, Chapter 5] paved the way and are still helpful for developing intuition. Huynh's paper [78] is geometric and optimises the size of description. More recent constructions [33, 35] optimise and analyse the dependence of the size on multiple parameters; these papers provide further references.

If the dimension d is fixed, the decision problem for *existential* Presburger arithmetic (where all quantifiers are $\exists$ and appear at the beginning of the formula) can be solved in polynomial time [128]. In fact, all satisfying assignments (if finitely many) can be efficiently enumerated if the formula, moreover, contains no occurrences of $\vee$ and $\neg$ and no congruence constraints. These are consequences of fundamental results of Lenstra [95] and Barvinok [14, 13]; see also, e.g., monographs [108, 39]. Nguyen and Pak [106] look at not

$$\begin{array}{lr} x: & 54\,321 \\ y: & +\quad 98\,765 \\ \hline z: & 153\,086 \end{array}$$

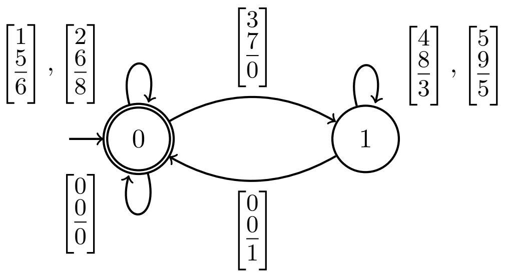
$\begin{bmatrix}1\\5\\\hline 6\end{bmatrix}, \begin{bmatrix}2\\6\\\hline 8\end{bmatrix}$
$\begin{bmatrix}0\\0\\\hline 0\end{bmatrix}$
0
$\begin{bmatrix}3\\7\\\hline 0\end{bmatrix}$
$\begin{bmatrix}0\\0\\\hline 1\end{bmatrix}$
1
$\begin{bmatrix}4\\8\\\hline 3\end{bmatrix}, \begin{bmatrix}5\\9\\\hline 5\end{bmatrix}$

Figure 4 Left: long addition base 10. Right: finite automaton that checks $x + y = z$, reading triplets of digits right to left. Only transitions traversed on the example on the left, as well as the transition for the triplet of leading zeros, are shown.

necessarily orthogonal projections of semi-linear sets (cf. proof idea for Theorem 9) and find a way to compute generating functions for these projections, à la Barvinok. In a certain technical sense, the idea extends to formulas with quantifiers in which d bounds the total number of variables (quantified or not).

In recent years, extensions of semi-linear sets with applications to verification of Petri nets have been considered in the literature [97, 63].

3.2 A view from automata theory: k-automatic sets

For simplicity, throughout this section arithmetic is over $\mathbb{N}$ instead of $\mathbb{Z}$. Representation of sets of numbers using finite automata is a far-reaching development of the following observation. Consider natural numbers divisible by 3. It is well-known that these are precisely the numbers whose sum of decimal digits is divisible by 3. A finite automaton can read the decimal expansion of $n \in \mathbb{N}$ digit by digit (the input alphabet is $\{0, 1, \dots, 9\}$), maintaining the remainder modulo 3 of all digits read so far. Then $3 \mid n$ if and only if the final remainder is 0. In fact, there is nothing special about 3: to capture divisibility by, say, 28, the automaton maintains the remainder modulo 28 of the number read so far.

To move from properties of individual numbers to properties of pairs and triplets of numbers, etc., we use automata over larger input alphabets.

▶ **Example 12.** We describe an automaton that checks addition $x + y = z$ for $x, y, z \in \mathbb{N}$. The idea is to mimic the elementary method of long addition (Fig. 4, left). The automaton will read triplets of decimal digits one at a time, so the input alphabet is $\{0, 1, \dots, 9\}^3$, with 1000 letters. Intuitively, we could say the input tape has three tracks, one for each of x, y, and z. Numbers can be padded with leading zeros (on the left), so that all tracks have the same length. A deterministic finite automaton (DFA) can read the tape from right to left, keeping the carry in its control state; see Fig. 4, right. The state with carry 0 is accepting.

Our automata will usually be incomplete: for example, there are no transitions on input letter $(0, 0, 5) \in \{0, 1, \dots, 9\}^3$, no matter the current carry. To make the DFA complete, we can send all such transitions to a rejecting sink state. ⌟

Let us move from base 10 to base 2. Perhaps counter-intuitively, it will be more convenient for us to use automata that read input from left to right instead, i.e., most significant bit first. (In Example 12, we simply reverse all transitions in the automaton.)

For $d \geqslant 1$, a set $S \subseteq \mathbb{N}^d$ is 2-*automatic* (or: 2-*recognizable*) if there is a deterministic finite automaton (DFA) that accepts the language

$$\{(w_1, \ldots, w_d) \in \left(\{0,1\}^d\right)^* : \text{for some } (n_1, \ldots, n_d) \in S, \text{ each } w_i \text{ is a binary expansion of } n_i\}. \quad (7)$$

Notice that each $n \in \mathbb{N}$ has infinitely many binary expansions: e.g., 110, 0110, 00110, etc. for $6 \in \mathbb{N}$. Replacing the base 2 with a larger integer $k \geqslant 3$, one gets k-automatic sets.

The definition refers to DFA, but any equivalent formalism for regular languages would do just as well, e.g., nondeterministic finite automata (NFA) or regular expressions.

Automata that represent finite sets (as in Fig. 1, left) are not unlike binary decision diagrams (BDDs), but can accept strings of varying length.

▶ **Theorem 13** (Büchi–Bruyère [30, 29], corollary)**.**
1. *Every set $S \subseteq \mathbb{N}^d$ definable in linear integer arithmetic is 2-automatic. The containment is effective: there is an algorithm that, given a formula φ defining S, constructs an automaton accepting the language from Eq.* (7)*.*
2. *There exists a 2-automatic set $S \subseteq \mathbb{N}$ that is not definable in linear integer arithmetic.*

Proof idea. The first part is proved by induction on the structure of the formula. We follow the definition of the syntax from Section 2:

- We need to find an automaton (DFA) for every set defined by an atomic formula (inequality, equality, or congruence). As in the proof of Theorem 9, we can focus on inequalities with no loss of generality. In fact, we reduce the reasoning to the simplest possible case: equalities of the form $x + y = z$ and $2x = z$. For inequalities, introduce *slack variables*: for instance, assuming that all variables range over $\mathbb{N}$,

  $$y \leqslant 2 \qquad \Longleftrightarrow \qquad \exists y' \; (y + y' = 2).$$

  Equalities are further rewritten. For example:

  $$s + 2u = 3v \quad \Longleftrightarrow \quad \exists y_1 \, \exists y_2 \, \exists y_3 \; (s + y_1 = y_3) \wedge (y_1 = 2u) \wedge (y_2 = 2v) \wedge (y_3 = v + y_2).$$

  We can now use Example 12 with two amendments: input is in base 2 instead of base 10, with most significant bit read first (instead of last).
- Sets defined by formulas of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, or $\neg\varphi$ are 2-automatic by the inductive hypothesis and by the (effective) closure of the family of regular languages under Boolean operations.
  For the complementation to work correctly, it is important that leading zeros not affect acceptance by the given automaton. Otherwise if, say, for $d = 1$ the string 110 is accepted but 0110 is not, then the correspondence between sets of numbers and languages breaks: the complement of the language will still contain a binary expansion of 6.
- We describe the idea how an existential quantifier can be handled. (For universal quantifiers, observe that $\forall x \; \varphi$ is equivalent to $\neg \exists x \; \neg\varphi$.) Let $\boldsymbol{y} = (y_1, \ldots, y_d)$ be the vector of free variables. Intuitively, if binary expansions of $\{(a, \boldsymbol{b}) \in \mathbb{N} \times \mathbb{N}^d : \text{formula } \varphi(x, \boldsymbol{y}) \text{ holds on } (a, \boldsymbol{b})\}$ can be recognised by a DFA over the alphabet $\{0,1\}^{d+1}$, then removing the x-track from all letters leads to the set $\{\boldsymbol{b} \in \mathbb{N}^d : \text{formula } \exists x \; \varphi(x, \boldsymbol{y}) \text{ holds on } \boldsymbol{b}\}$.
  We need to be careful, however: in some accepted strings the erased track may be strictly longer than all other tracks; that is, a has more binary digits than each component of $\boldsymbol{b}$. (In the formula $\exists z \; (x + y = z)$, this happens for the variable z: see Fig. 4 (left).) Thus, after removing x-components from all letters on the transitions of the original DFA, we make a state q accepting if and only if that DFA had an accepting run from q on which all letters have 0 on the d "surviving" tracks.

For the second part of the theorem, observe that the set $P_2 = \{n \in \mathbb{N} \colon n = 2^k \text{ for some } k \in \mathbb{N}\}$ is 2-automatic: binary expansions of powers of 2 are exactly strings matched by regular expression 0^*10^*. Since P_2 is not ultimately periodic, it is not definable in linear integer arithmetic (see Section 3.1). ◀

The first part of Theorem 13, applied to sentences, gives a decision procedure for Presburger arithmetic.

The sketch above offers relatively little intuition as to how automata for various arithmetic constraints really work. In the following example, we show an alternative, direct construction.

▶ **Example 14.** Consider the co-primality formula $C_{a,b}$ from Example 1 (Section 1). Fix $a = 3$ and $b = 2$. Unlike in Section 1, we let all variables range over $\mathbb{N}$ not $\mathbb{Z}$.

For uniformity of notation, let us rename s to x_0. Take the atomic formula $x_0 = 3x_1 - 2x_2$. The idea is similar to the divisibility examples at the beginning of this section. As an automaton reads triplets of bits left to right, it keeps in memory the current *error* (rather than remainder modulo 3 or 28): the integer $e = x_0 - 3x_1 + 2x_2$. (For the moment, we can think of an infinite-state automaton, to be made finite-state shortly.) If $e = 0$, the current state is accepting. Initially $e = 0$, because our equation $x_0 = 3x_1 - 2x_2$ is homogeneous. How does this error change over time?

Suppose the next bit triplet is $(\alpha_0, \alpha_1, \alpha_2) \in \{0,1\}^3$. If x_0, x_1, x_2 are the binary numbers read so far, then after reading the new bits these numbers become

$$x_0' = 2x_0 + \alpha_0, \qquad x_1' = 2x_1 + \alpha_1, \qquad x_2' = 2x_2 + \alpha_2.$$

Thus, the new error is

$$e' = x_0' - 3x_1' + 2x_2' = 2(x_0 - 3x_1 + 2x_2) + (\alpha_0 - 3\alpha_1 + 2\alpha_2) = 2e + (\alpha_0 - 3\alpha_1 + 2\alpha_2).$$

Denote $u \overset{\text{def}}{=} \alpha_0 - 3\alpha_1 + 2\alpha_2$ and observe that $u \in [-3,3]$. Thus, if $|e| \geqslant 3$, then $|e'| \geqslant 3$ as well and so state 0 is no longer reachable. Thus, just five states for $e \in \{-2,-1,0,1,2\}$ suffice, and all other values can be glued into a rejecting sink.

The automaton for the quantifier-free formula $x_0 = 3x_1 - 2x_2$ has a lot of transitions. The table in Fig. 5 (left) shows the update u for each $(\alpha_0, \alpha_1, \alpha_2) \in \{0,1\}^3$. Observe that the destination of transitions depends only on u and not on $\alpha_0, \alpha_1, \alpha_2$. In principle, each bit triplet can be read from each control state, but if the new error $e' = 2e + u$ is outside $[-2,2]$, then the transition goes into the rejecting sink. Based on this automaton (which we do not depict), we construct an automaton (NFA) for the formula $\varphi(x_0) \colon \exists x_1 \, \exists x_2 \, (x_0 = 3x_1 - 2x_2)$: for each transition, the label $(\alpha_0, \alpha_1, \alpha_2)$ is replaced by just α_0; see Fig. 5 (right).

We omit the acceptance analysis. It turns out that all states except -2 are accepting, and the language recognised by the NFA is $\{0,1\}^*$. Thus, formula $C_{3,2}$ from Example 1 is true even if all variables are interpreted over $\mathbb{N}$. ⌟

Further reading. In connection with the second part of Theorem 13, it is meaningful to ask whether a given set $S \subseteq \mathbb{N}^d$ represented by an automaton is definable in Presburger arithmetic. As it turns out, this definability can be determined in polynomial [96] and, in dimension $d = 1$, even almost linear time [100, 21]. The first of these algorithms was implemented in TaPAS [98], a software framework for the theory of mixed real-integer linear arithmetic [143, 19]. Paper [96] also contains a collection of references for contemporaneous software for linear integer arithmetic. In a different direction, sets represented by automata can be characterised by another logic, so-called Büchi arithmetic (see, e.g., [29, 18]).

α_0	α_1	α_2	u
0	0	0	0
0	0	1	2
0	1	0	-3
0	1	1	-1
1	0	0	1
1	0	1	3
1	1	0	-2
1	1	1	0

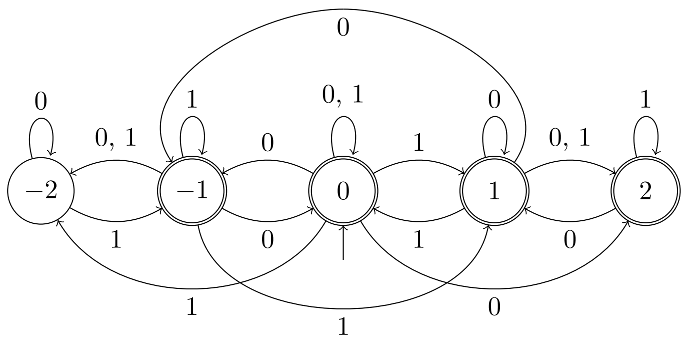

Figure 5 Left: aggregate effect u for each bit triplet. Right: nondeterministic finite automaton for the formula $\varphi(x_0)\colon \exists x_1\, \exists x_2\, (x_0 = 3x_1 - 2x_2)$, with variables interpreted over $\mathbb{N}$.

Let us mention a tool MONA [76, 86], which implements an automata-based decision procedure for a more powerful logic: the weak monadic second-order logic with one successor (WS1S). Informally speaking, this logic can "encode" Presburger arithmetic.

A more detailed exploration of the subject of this section (and, more generally, the view of automata as data structures) can be found in Esparza and Blondin's textbook [50, Chapter 9]. Another source is a survey by Boigelot and Wolper [22]. Survey [29] reviews Büchi's idea (including a bugfix) and subsequent developments up to the 1990s. Several chapters in the book [117] discuss topics such as interface of automata theory with number theory (see also Rigo's earlier survey [125]); automatic sequences; and automatic structures. The latter are structures (in logic) in which, informally speaking, relations can be represented by automata (generalising Example 12); see, e.g., Grädel's tutorial [60]. A recent monograph by Shallit [134] explores the use of logic to study automatic sequences (a closely related concept), with applications in combinatorics on words. For a further sample of cutting-edge developments, the reader is referred to papers [66, 139, 70, 43].

3.3 A view from symbolic computation: quantifier elimination

In symbolic computation, we rewrite a given formula iteratively, using a set of predefined rules. The resulting formulas get progressively simpler structurally, usually at the cost of increase in size. Often the objective is to remove quantifiers, so the approach is known as *quantifier elimination*. We first give an example outside Presburger arithmetic.

▶ **Example 15.** Given $p, q \in \mathbb{R}$, a quadratic equation $x^2 + px + q = 0$ has a real solution if and only if its discriminant is nonnegative: $p^2 - 4q \geqslant 0$. Thus, the formulas $\exists x\, [x^2 + px + q = 0]$ and $p^2 - 4q \geqslant 0$ (of a suitable logic over $\mathbb{R}$) are equivalent, that is, we have eliminated the quantifier $\exists x$. (Both formulas have free variables p and q.) ⌟

Importantly, we would like the resulting formula to stay within the syntax of the same logic that we start from. For Presburger arithmetic, this is possible. (As we saw in Section 1 on the example of formula $E(x)\colon \exists y\, (x = 2y)$, congruence constraints cannot be dispensed with.) Geometrically, we saw in Section 3.1 that elimination of an existential quantifier corresponds to orthogonal projection. This is very easy to handle for semi-linear sets in generator representation; for logical formulas, a little more work is required.

A formula is *quantifier-free* if symbols $\exists$ and $\forall$ do not occur in it.

▶ **Theorem 16** (Presburger [121]). *There exists an algorithm that, given a quantifier-free formula φ, outputs a quantifier-free formula φ' equivalent to $\exists x\, \varphi$.*

Theorem 16 gives a decision procedure for linear integer arithmetic, in fact historically the first one. English translation of and commentary on Presburger's 1929 paper are available [137, 122].

Instead of giving the proof in full generality, we will consider several examples. We roughly follow an algorithm given by Cooper [37], later reproduced (under the heading "The present algorithm") in Cooper [38].

▶ **Example 17.** Consider the formula $\exists x\, [(x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2)]$, which defines the projection of the set T from Fig. 1 (page 2) on the y axis. Only the first two inequalities in the formula mention x. Putting them into a chained inequality $2y \leqslant x \leqslant 3y$, we see that a suitable value for x exists (in $\mathbb{Z}$) if and only if $2y \leqslant 3y$, that is, $y \geqslant 0$. Thus, the entire formula is equivalent to $(y \geqslant 0) \wedge (y \leqslant 2)$. This is exactly the projection $\{0, 1, 2\}$ of the set T on the y axis. ⌟

If the formula has several inequalities bounding x from below (say, $s_1 \leqslant x, \ldots, s_k \leqslant x$, where $s_1, \ldots, s_k$ are some terms not involving x) and several inequalities bounding it from above (say, $x \leqslant t_1, \ldots, x \leqslant t_m$, with $t_1, \ldots, t_m$ not involving x), then a suitable value for x exists (in $\mathbb{Z}$) if and only if $s_i \leqslant t_j$ for all pairs i, j. In words, the greatest lower bound does not exceed the least upper bound.

▶ **Example 18.** Now consider the formula $\exists y\, [(x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2)]$, which defines the projection of the same set T from Fig. 1 (page 2) on the x axis. To use the same principle as in Example 17, we multiply both sides of each inequality by a positive integer so as to make all the coefficients at y identical. In this case, they will be $6 = \operatorname{lcm}(3, 2, 1)$. For all $x, y \in \mathbb{Z}$, we have $x \leqslant 3y$ if and only if $2x \leqslant 6y$; the other two inequalities are handled similarly. We obtain one lower bound and two upper bounds on $6y$; thus, the formula is true if and only if $2x \leqslant 6y \leqslant \min\{3x, 12\}$, or equivalently $(2x \leqslant 6y \leqslant 3x) \wedge (2x \leqslant 6y \leqslant 12)$.

It is, however, not true that a suitable integer value for y exists if and only if $2x \leqslant \min\{3x, 12\}$. Indeed, $x = 1$ gives a counterexample. Rather, such a value exists if and only if there is an integer divisible by 6 between $2x$ and $\min\{3x, 12\}$. To express this statement without existential quantification ("there is an integer"), we consider several cases depending on the remainder of $2x$ modulo 6. For example, if $2x$ has remainder 2 modulo 6, then instead of inequality $2x \leqslant \min\{3x, 12\}$ the condition to use is $2x + 4 \leqslant \min\{3x, 12\}$, "rounding up" the lower bound to a multiple of 6. (This is because $y = 2x + 4$ is always a suitable choice of y if $2x + 4 \leqslant \min\{3x, 12\}$ and $2x + 4 \equiv 0 \pmod 6$.) The formula is thus equivalent to

$$\bigvee_{r=0}^{5} [(2x + r \equiv 0 \pmod 6) \wedge (2x + r \leqslant \min\{3x, 12\})]$$

or, slightly less succinctly,

$$\bigvee_{r=0}^{5} [(2x + r \equiv 0 \pmod 6) \wedge (2x + r \leqslant 3x) \wedge (2x + r \leqslant 12)]\,.$$

This eliminates the existential quantifier. Observing that only even r are possible and collecting like terms, we can simplify the result further into

$$\begin{aligned} &[(x \equiv 0 \pmod 3)) \wedge (0 \leqslant x) \wedge (2x \leqslant 12)] \vee \\ &[(x + 1 \equiv 0 \pmod 3)) \wedge (2 \leqslant x) \wedge (2x \leqslant 10)] \vee \\ &[(x + 2 \equiv 0 \pmod 3)) \wedge (4 \leqslant x) \wedge (2x \leqslant 8)]. \end{aligned}$$

In the three cases, we get $x \in \{0, 3, 6\}$, $x \in \{2, 5\}$, and $x \in \{4\}$, so the formula defines the set $\mathbb{Z} \cap [0, 6] \setminus \{1\}$. This is the projection of the set T in Fig. 1 (left) on the x axis. ⌟

In Example 18, it is clear that the "hole" $x = 1$ in the projection of T arises because of divisibility constraints, which are necessary because we are eliminating an integer variable. In comparison, for the theory of linear *real* arithmetic, the conjunction $\max s_i \leqslant \min t_j$ already does the job. In fact, over $\mathbb{R}$ this method shows that the orthogonal projection of a convex polyhedron (that is, of the set of real solutions to a system of linear inequalities) is also a convex polyhedron. The method is known as the *Fourier–Motzkin quantifier elimination* for linear real arithmetic (see, e.g., [112, Chapter 1] and [92, Chapter 1]), a useful instrument not only in the development of theory but also in modern practical tools [105].

As in Section 3.1, we see that reasoning about integers requires a combination of an argument for reals and an approach for handling divisibility constraints.

Proof idea for Theorem 16. Using De Morgan's laws and equivalences such as

$$\neg(t_1 \leqslant t_2) \iff (t_2 < t_1), \qquad (t_1 < t_2) \iff (t_1 + 1 \leqslant t_2),$$
$$\neg(t_1 \equiv t_2 \pmod{m}) \iff \bigvee_{r=1}^{m-1} (t_1 + r \equiv t_2 \pmod{m}),$$

bring the given formula φ to disjunctive normal form (DNF), or rather (more precisely) an OR of ANDs of non-strict linear inequalities and congruence constraints (no negations). As $[\exists x\ (A \vee B)] \iff (\exists x\ A) \vee (\exists x\ B)$, the problem reduces to handling one such AND. For this special case, the idea is shown in Examples 17 and 18, and here is a more structured sketch:

1. Multiply both sides of each constraint so that all the coefficients at x become identical, say to $m \in \mathbb{Z}$. (A natural choice for m is the least common multiple of all coefficients at x.)
2. Replace all occurrences of mx by x', where x' is a new (*fresh*) variable. Conjoin (AND) the congruence $x' \equiv 0 \pmod{m}$ to the formula. We are now handling $\exists x'$ instead of $\exists x$. (In Example 18, the auxiliary variable would be y' standing for $6y$.)
3. The formula is now an AND of constraints not involving x', several lower bounds on x', several upper bounds on x', and several congruence constraints on x'. We split into cases (OR) according to which of the lower bounds $s_1, \dots, s_k$ is the greatest and according to the remainder (of this greatest lower bound) with respect to the least common multiple of moduli in congruence constraints. Within each case, x' can be eliminated. ◀

▶ Remark 19. A different take on quantifier elimination for Presburger arithmetic follows Cooper's algorithm ("the new algorithm") in [38], not requiring conversion to DNF. After all coefficients at variable x are made identical, the inequalities involving x split the set $\mathbb{Z}$ into a finite number of intervals. A suitable value for x exists if and only if one of these intervals contains an integer that satisfies (or perhaps fails) a certain subset of the inequalities and has some divisibility properties (as prescribed by the Boolean structure of the formula).

Suppose the least common multiple of all divisors in the divisibility constraints is $M > 0$. ($M = 1$ if there are no such constraints.) Let $[\alpha, \beta]$ be one of the intervals, where α and β are terms in which x does not appear. Then a suitable value for x (importantly, one with the right divisibility properties) exists in $[\alpha, \beta]$ if and only if one of $\alpha, \alpha + 1, \dots, \alpha + (M - 1)$ is suitable and does not exceed β; this is because shifting x by M leaves the truth value of all divisibility constraints unchanged. The case $\alpha = -\infty$ is similar and simpler. ⌟

▶ **Observation 20.** Let a formula φ' be output by Cooper's quantifier elimination procedure on input φ, which has eliminated variable x. Constraints of φ' arise from equating pairs of expressions for (or rather bounds on) x. If φ contains inequalities $s \leqslant ax$ and $bx \leqslant t$, where $a, b > 0$, then their conjunction entails $bs \leqslant at$, or equivalently $at - bs \geqslant 0$. A rescaling by nonzero $\lambda \in \mathbb{Q}$ may be necessary if $\operatorname{lcm}(a,b) \neq ab$, or if other coefficients are taken into account when computing the least common multiple. If, as in Example 18, divisibility constraints come into the picture, then a constant $\alpha \in \mathbb{Z}$ may be added or subtracted.

In fact, *all* inequalities in φ' can be shown to arise this way. Suppose that an atomic formula $\tau \leqslant 0$ mentions two or more variables and appears in φ'. Then there are numbers $\lambda \in \mathbb{Q}$, $\alpha \in \mathbb{Z}$ such that τ is $\lambda \cdot (a_1\tau_2 - a_2\tau_1) + \alpha$ for some terms $a_1x + \tau_1$ and $a_2x + \tau_2$ which appear in the original formula φ. (For brevity, we do not make precise the notion of appearance of a term in a formula.)

Further reading. Cooper's paper [38] is a lucid introduction to quantifier elimination, and his decision procedures have been analysed in more detail and extended (see, e.g., Oppen [111] as well as Section 4.1 of the present paper). A formalisation in Isabelle/HOL is available [109].

The "big" disjunctions in the formulas resulting from quantifier elimination in Example 18 can instead be replaced by a bounded version of the existential quantifier ($\exists r \in [0,5]$). Such quantifiers are not part of the syntax (Section 2) but can be added. This leads to so-called *uniform* (generalisation of) Presburger arithmetic and the idea of *weak* quantifier elimination [142], developed further [90] and implemented in Redlog. Redlog [42] is part of the general-purpose computer algebra system REDUCE and offers quantifier elimination methods for multiple arithmetic theories, Presburger arithmetic being one of them.

Chapter 4 of Kreisel and Krivine's book [88] introduces quantifier elimination for several arithmetic theories, such as Presburger arithmetic (although this name is not mentioned). The reader should beware that this book uses symbols $\bigwedge$ and $\bigvee$ in place of now ubiquitous $\forall$ and $\exists$. Unlike our presentation, the book also discusses not only model-theoretic aspects of quantifier elimination but also proof-theoretic aspects (which axioms can justify the equivalence of formulas).

Multiple further versions and extensions of quantifier elimination, for a variety of logical theories, have been proposed in the literature and implemented in software. Most famously, Tarski's quantifier elimination procedure for the first-order theory of $\mathbb{R}$ with addition, multiplication, and ordering led in the 1970s to Collins's cylindrical algebraic decomposition, a fundamental concept in computer algebra. In the first-order theory of the reals, atomic formulas are inequalities between multivariate polynomials; see, e.g., [44] and [103, Section 5].

Many extensions of Presburger arithmetic are known to have quantifier elimination (see, e.g., [132, 99, 133, 130]). For Presburger arithmetic itself, Weispfenning [141] analyses the size of formulas output by (his) quantifier elimination procedure. Bounds that he obtains in terms of the number of quantifier blocks (alternation) and the number of variables in each block are a versatile instrument for other problems. An early implementation of such a procedure is the Omega test [123]. In the SMT solver cvc5, quantifier elimination for linear arithmetic uses so-called counterexample-guided instantiation [124, 9].

4 Alternation of quantifiers and computational complexity

The three views shown in Section 3 give rise to three (kinds of) decision procedures for Presburger arithmetic. In the current section, we look further into the computational complexity aspects.

If we analyse the decision procedures sketched above directly, we get overly pessimistic estimates of the running time and memory requirements. These estimates can be improved substantially (Section 4.1). Nevertheless, the decision problem turns out inherently hard. Clever formulas in linear integer arithmetic can express rather intricate sets, and it is possible to prove worst-case lower bounds on the complexity of the problem itself (Section 4.2).

Table 1 Effect of logical connectives and quantifiers on the size of representation (informally). For all three views, $\forall$ can be replaced by $\neg\exists\neg$.

View	Geometry	Automata theory	Symbolic computation (quantifier elimination)
Representation	Semi-linear sets	Automata	Logical formulas
$\neg$	expensive	NFA $\rightsquigarrow$ DFA	CNF $\rightsquigarrow$ DNF*
$\vee$	trivial	trivial	trivial
$\wedge$	OK	easy	easy*
$\exists$	trivial	easy	expensive

* For some but not all quantifier elimination procedures. Trivial for other procedures.

As we will see, the high computational complexity is in some sense linked with the alternation of quantifiers of type $\exists$ and $\forall$ in the logical formulas. The present section focuses on formulas with alternation, and Section 5 on formulas without it.

4.1 Handling quantifiers in decision procedures

As seen from Table 1, in each of the three views on Presburger arithmetic some of the logical connectives and quantifiers may require a big increase in the size required to represent the object – a semi-linear set, an automaton, or a logical formula. In slightly more detail:

For semi-linear sets, complementation is the most difficult operation. Intersection requires work but the increase in size is smaller in comparison. (In dimension 1, the new period can be chosen as the least common multiple of old periods; in higher dimension a suitable generalisation is required.)

For automata, the existential quantifier requires an update to the set of accepting states (which is easy) and, most importantly, makes the automaton nondeterministic (NFA not DFA). Complementation of the NFA is then expensive, e.g., requiring the subset construction.

For logical formulas, some quantifier elimination algorithms require conversion to DNF. For them, negation is difficult (conversion from CNF to DNF) and conjunction requires the application of the distributive law. Other algorithms can handle arbitrary Boolean structure. In either case, the existential quantifier presents the main challenge.

In summary, for each method some operation requires an exponential growth in size, also referred to as an exponential blow-up, in the worst case.

If n is the size of the input sentence, then stacking n exponentials leads to a worst-case upper bound on the running time of decision procedures that has the form $2^{\cdot^{\cdot^{\cdot^{2}}}}$, where the height of the tower grows as n. Differences between functions such as 2^n, 2^{cn}, 2^{n^2}, etc., can be ignored for the purpose of this crude estimate. For all three views, a sequence of n alternating quantifiers $\exists x_1 \forall x_2 \exists x_3 \ldots$ presents a challenge.

Importantly, better decision procedures (and sometimes better analyses) are available.

A function $f\colon \mathbb{N} \to \mathbb{N}$ is called *elementary* if $f(n) \leqslant 2^{\cdot^{\cdot^{\cdot^{2^n}}}}$, where the tower has some fixed height $k \in \mathbb{N}$. A yes–no problem belongs to the complexity class ELEMENTARY if it has an algorithm with elementary running time. For example, all problems in complexity classes P, NP, PSPACE, EXP, 2-EXP, etc., also belong to this huge class. According to English Wikipedia, “The name was coined by László Kalmár, in the context of recursive functions and undecidability; most problems in it are far from elementary.” For computationally hard problems in logic, achieving polynomial running time or even polynomial space (memory usage) is often not possible, and the dichotomy between elementary and non-elementary bounds can indicate problems for which practical implementations can hope to handle some medium-size inputs in reasonable time.

▶ **Theorem 21** (Oppen [111]). *There is an algorithm that solves the decision problem for Presburger arithmetic in triply exponential time.*

As we discuss below, in fact all three approaches (based on semi-linear sets, on automata, and on the elimination of quantifiers) provide elementary decision procedures.

Proof idea. Cooper's quantifier elimination algorithm [38] sketched in Section 3.3 can be shown to require at most $2^{2^{2^{pn}}}$ basic computational steps on sentences of size n, where $p > 0$ is some universal constant. To improve upon the crude non-elementary upper bound, we need to measure the complexity of formulas using several parameters, rather than just one (size of the formula). The dependence of these parameters on one another may still seem to lead to a tower of exponentials, but the idea is to find a parameter that enjoys a tighter (elementary) estimate. We can think of it as a *controlling parameter*, because the other parameters can then be bounded once we know this key one is under control.

Our controlling parameter will be the number of distinct coefficients of variables in the formula. For a formula φ, let us denote this quantity by $z(\varphi)$. We only sketch the core of the argument, leaving many details to the reader. Consider the elimination of a single existential quantifier. Let φ be a formula given to Cooper's algorithm, and let φ' be the output formula equivalent to $\exists x\ \varphi$. Let us review Observation 20 from Section 3.3. Each inequality in φ' arises from two inequalities of φ by cross-multiplication. Given terms $ax + by + t'$ and $cx + dy + t''$ that appear in inequalities of φ, the result is

$$(ad - bc)\, y + (at'' - ct') \sim 0, \tag{8}$$

where $\sim$ is one of the signs $\leqslant$, $\geqslant$; possibly rescaled by some nonzero $\lambda \in \mathbb{Q}$ and with an additive shift $\alpha \in \mathbb{Z}$. We ignore the scaling factor $\lambda \in \mathbb{Q}$ in this proof. So, for a variable y, every coefficient at y in the new formula φ' is a combination of at most 4 coefficients in φ, two of them at y and two at the variable x that is being eliminated. Thus, $z(\varphi') \leqslant z(\varphi)^4$.

Now denote by φ_k the formula obtained from φ after eliminating k quantifiers; $\varphi_0 = \varphi$. In the worst case, $z(\varphi_k)$ grows with k as follows:

$$z(\varphi) = z \quad \rightsquigarrow \quad z^4 \quad \rightsquigarrow \quad z^{4^2} \quad \rightsquigarrow \quad z^{4^3} \quad \rightsquigarrow \quad \ldots \quad \rightsquigarrow \quad z^{4^k}.$$

Formally, one can prove by induction an upper bound $z(\varphi_k) \leqslant M_k \overset{\text{def}}{=} z(\varphi)^{4^k}$. Since k is bounded from above by the size of the input formula φ, the number of distinct coefficients throughout the entire procedure is at most doubly exponential in the size of φ. With this elementary bound in hand, we can bound other parameters, such as the magnitude of coefficients, of moduli in congruence constraints, of constant terms, etc. For instance, the number of *linear terms* of the form $a_1x_1 + \ldots + a_mx_m$ which may arise in the formula is bounded by $(M_k)^m$, where the number of variables, m, is again at most the size of φ. This is also an elementary bound. (We note that the total number of inequalities might be much higher than the number of linear terms, because of additive constants.) Bounds can then be combined into an elementary bound on the size of formulas. ◀

There are a number of factors that we have glossed over. For example, in the sketch above we have considered two constraints in isolation, whereas the least common multiple of all coefficients at x may be much bigger than just ac. However, the principle remains sound.

Starting with Oppen's theorem, elementary decision procedures have been given for all three views on linear integer arithmetic. Oppen's result was extended to automata [85, 45, 46] and recently to semi-linear sets [35]. Weispfenning's quantifier elimination procedure [141] also runs in triply exponential time, as best procedures for all three views do.

We briefly comment on the geometric view. It is convenient to extend the $L(\boldsymbol{b}, P)$ notation (Eq. (6) in Section 3.1) to $L(B, P) \overset{\text{def}}{=} \bigcup_{\boldsymbol{b} \in B} L(\boldsymbol{b}, P)$. Sets of this form have been studied by Ginsburg and Spanier [57, 56], and we refer to them as *hybrid linear* sets. As we will see in Section 5.1, these are sets of integer solutions to systems of linear inequalities. Semi-linear sets are exactly unions

$$S = \bigcup_{i \in I} L(B_i, P_i), \qquad |I|, |B_i|, |P_i| < \infty. \tag{9}$$

As it turns out, in generator representation the cardinality $|I|$ can be chosen as the controlling parameter for elementary bounds. By a discrete version of Carathéodory's theorem (see, e.g., [48] and [33, Proposition 5]), the reasoning can be reduced to the case where each set P_i contains linearly independent vectors only. The reader may now see the link with the number of *linear terms* in the logical formulas during quantifier elimination. A useful ingredient in the constructions is the observation, for every fixed $d \geqslant 1$, that n hyperplanes in $\mathbb{R}^d$ split it into at most $O(n^d)$ regions [101, Chapter 6].

4.2 Using quantifiers to write succinct formulas

We typically use arbitrary integers in Presburger formulas. One can ask if the succinctness of the logic changes if only small integers (say, between -2 and 2) are allowed. The answer to this question is negative: for instance, the formula $y = 2^n \cdot x$ is equivalent to

$$\exists y_0\, \exists y_1\, \ldots \exists y_n\ (y_0 = x) \wedge (y_1 = 2y_0) \wedge (y_2 = 2y_1) \wedge \ldots \wedge (y_n = 2y_{n-1}) \wedge (y = y_n).$$

Thus, numbers up to $2^{\mathrm{poly}(n)}$ can be expressed by formulas of size $\mathrm{poly}(n)$ with coefficients from $[-2, 2]$. Can we express even larger numbers using small formulas?

▶ **Example 22** (Fischer and Rabin [53]). There is a sequence of formulas $F_n(x, y)$ of size $O(n)$ such that $F_n(x, y)$ holds if and only if $x = 2^{2^n} \cdot y$. According to Fischer and Rabin [53], "[this] device is a special case of a more general theorem due to M. Fischer and A. Meyer. It was rediscovered independently by several people including V. Strassen."

We can take $F_0(x, y)\colon\ x = 2y$, since $2 = 2^{2^0}$. The sequence is then constructed inductively. The idea is to have $F_{n+1}(x, y)$ equivalent to $\exists z_n\ [F_n(x, z_n) \wedge F_n(z_n, y)]$. This direct definition, however, would unravel into a very big formula, of size $O(2^n)$ for index n. Instead, the following construction has only one instance of F_n and thus unravels into a formula of size $O(n)$:

$$F_{n+1}(x, y)\colon\ \exists z_n\, \forall u_n\, \forall v_n\ \big[\big(\big((u_n = x) \wedge (v_n = z_n)\big) \vee \big((u_n = z_n) \wedge (v_n = y)\big)\big) \to F_n(u_n, v_n)\big].$$

The *bit size* of the formula is actually $O(n \log n)$, with the $\log n$ factor due to the need to refer to at least n distinct variables. Interestingly, F_n works just as well over $\mathbb{R}$. ⌟

Combining Example 22 with the Chinese remainder theorem (generalising Example 2), Fischer and Rabin also give a sequence of formulas that define triply exponential rather than doubly exponential numbers; see Kozen's textbook [87, Lecture 24] for details.

Formulas from Example 22 and more sophisticated ones are then used to "define" and manipulate long bit strings. This led Fischer and Rabin [53] to lower bounds on the computational complexity of the decision problem for linear integer (as well as real) arithmetic.

▶ **Theorem 23** (Fischer and Rabin [53]). *The decision problem for Presburger arithmetic requires at least doubly exponential time in the worst case, even for nondeterministic algorithms.*

For linear real arithmetic, the lower bound is a single exponential. A precise characterisation of the computational complexity of the problem was subsequently given by Berman [17]: the decision problem for Presburger arithmetic is complete for $\mathrm{STA}(*, 2^{2^{O(n)}}, n)$. The acronym STA stands for "space, time, alternation"; this class corresponds to Turing machines that run in time $2^{2^{cn}}$ for some $c > 0$ on inputs of length n, using up to $n-1$ alternations between nondeterministic and universal modes (with $n = 0$ corresponding to deterministic algorithms). Kozen [87, Lectures 23–24] offers a modern exposition of results of this kind.

Theorem 23 and Berman's strengthening of it suggest that triply exponential algorithms (Section 4.1) are optimal in the worst case: known simulation results of 2-NEXP and $\mathrm{STA}(*, 2^{2^{O(n)}}, n)$ machines by deterministic algorithms run in triply exponential time.

In logical formulas arising from applications, *alternation depth* is often low, i.e., formulas may have the form $\Phi(\boldsymbol{u})\colon \forall \boldsymbol{x}^1 \exists \boldsymbol{x}^2 \dots \mathsf{Q}\boldsymbol{x}^k\ \varphi(\boldsymbol{x}^1, \dots, \boldsymbol{x}^k, \boldsymbol{u})$ with small k. We can think of k as being fixed. Here φ is quantifier-free, Q is either $\exists$ or $\forall$ depending on the parity of k, and all variables in each block $\boldsymbol{x}^i$ are bound by the same kind of quantifier (existential or universal). As an example, $\forall^*\exists^*$ formulas have alternation depth $k = 2$. Naturally, sentences may also start with the existential block.

▶ **Example 24** (Grädel [59], see also Haase [64]). There exists a constant $c > 0$, a sequence of integers $(r_n)_{n \geqslant 1}$, and a sequence of $\forall\exists^*$-formulas $G_n(z)$ of size $O(n)$ such that (i) $G_n(z)$ holds if and only if r_n divides z and (ii) $r_n \geqslant 2^{2^{cn}}$ for all n.

The construction consists of three steps. In the first step, we construct a sequence of formulas $M_n(x, y, z)$ for *bounded multiplication:* under the assumption $0 \leqslant x < 2^n$, the formula $M_n(x, y, z)$ will be true if and only if $x \cdot y = z$. For other values of x, the truth value of M_n is unconstrained. Here is the formula:

$$\begin{aligned}
M_n(x,y,z)\colon \quad & \exists x_1 \dots \exists x_n\ \exists z_1 \dots \exists z_n \\
& \qquad (x = x_1 \cdot 2^{n-1} + x_2 \cdot 2^{n-2} + \dots + x_n) \wedge \bigwedge_{i=1}^{n} [(0 \leqslant x_i) \wedge (x_i \leqslant 1)] \wedge \\
& \qquad (z = z_1 \cdot 2^{n-1} + z_2 \cdot 2^{n-2} + \dots + z_n) \wedge \\
& \bigwedge_{i=1}^{n} \big[\big((x_i = 0) \rightarrow (z_i = 0)\big) \wedge \\
& \qquad \big((x_i = 1) \rightarrow (z_i = y)\big) \big].
\end{aligned}$$

This is an *existential* formula of size $O(n)$.

The second step constructs the formula $D_n(x, z)\colon \exists y\ M_n(x, y, z)$. Under the assumption $0 \leqslant x < 2^n$, this formula asserts that z is a multiple of x. In the third step we arrive at $G_n(z)\colon \forall x\ [((0 < x) \wedge (x < 2^n)) \rightarrow D_n(x, z))]$. Now $r_n = \mathrm{lcm}(1, 2, \dots, 2^n - 1)$. ⌟

Example 24 leads to Grädel's lower bound in the following theorem:

▶ **Theorem 25** (Grädel [59] and Haase [64]). *The decision problem for $\forall^*\exists^*$-Presburger arithmetic is complete for* coNEXP.

Class coNEXP contains complements of problems that can be decided in nondeterministic time $O(2^{n^c})$, for some $c > 0$. The upper bound is due to Haase, who also proved that, for each $k \geqslant 2$, the fragment of Presburger arithmetic with alternation depth k is complete for the $(k-1)$st level of the so-called weak EXP hierarchy [64].

More recently, Nguyen and Pak have studied the computational complexity of further restricted fragments. By encoding problems related to continued fractions, they were able to show, for instance, that the decision problem for sentences of the form $\exists z\, \forall y_1\, \forall y_2\, \exists x_1 \exists x_2 \colon \varphi(x_1, x_2, y_1, y_2, z)$ is NP-hard even for φ with 10 inequalities [107]. In this problem, the sentence is fixed entirely except for the numbers in atomic formulas.

5 Three views on integer programming

The feasibility problem of *integer (linear) programming* is the following problem:

Input: A system $A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}$ of m linear inequalities in n variables, where $A \in \mathbb{Z}^{m\times n}$ and $\boldsymbol{c} \in \mathbb{Z}^m$.
Output: Does the system have a solution in $\mathbb{Z}^n$?

Here the relation $\leqslant$ between vectors is component-wise: for $\boldsymbol{a} = (a_1, \ldots, a_m)$ and $\boldsymbol{b} = (b_1, \ldots, b_m)$ we write $\boldsymbol{a} \leqslant \boldsymbol{b}$ if and only if $a_i \leqslant b_i$ for each i between 1 and m.

In the present paper, "integer programming" will refer to this problem (that is, we will not consider optimisation versions thereof). It is a special case of the decision problem for Presburger arithmetic, and even for *existential* Presburger arithmetic, where all quantifiers are existential and appear at the beginning of the formula.

▶ **Theorem 26** (Borosh and Treybig [24] and Papadimitriou [113]). *Integer programming is* NP*-complete.*

▶ **Corollary 27** (see, e.g., [111]). *Existential Presburger arithmetic is* NP*-complete.*

In these statements, we omit the words "the decision problem for". NP-hardness of both problems is easy to justify, by an encoding of the subset sum problem (Example 3, conjunctive formula) or Boolean satisfiability (SAT). We now derive Corollary 27 from Theorem 26.

Proof of Corollary 27. Given a sentence in Presburger arithmetic, where all quantifiers are existential and appear at the beginning of the formula, we eliminate all negations (except on congruence constraints) using De Morgan's laws and the equivalence $(t_1 < t_2) \iff (t_1 + 1 \leqslant t_2)$. We can similarly eliminate all atomic formulas except inequalities and the remaining negated congruences. The quantifier-free part of the formula is now a monotone Boolean combination of atoms of two types: non-strict linear inequalities and negated congruences. Such a sentence is true if and only if there exists a subset of these atoms and, for each negated congruence $\neg(t_1 \equiv t_2 \pmod{m})$, a remainder $r \in \{1, \ldots, m-1\}$ such that:

(1) if all the atoms in the subset are true, then the formula is true;
(2) if each negated congruence in the subset is replaced by the equality $t_1 + r = t_2 + mw$, where w is a fresh variable, and then by two inequalities $t_1 + r \leqslant t_2 + mw$ and $t_1 + r \geqslant t_2 + mw$, then the resulting system of inequalities is a positive instance of the integer programming problem.

The first condition is checked directly, and the second is an NP condition by Theorem 26. ◀

Thus, existential Presburger arithmetic is very close to integer programming. Membership of integer programming in NP is not obvious. It is indeed the case that a guessed solution can be verified quickly; but a separate argument is required to show that, whenever some solution exists, a *small* solution exists too, where "small" means "of polynomial bit size relative to the bit size of the system".

At first sight, appeal to the three views on Presburger arithmetic does not seem to resolve the problem: the size of representation in the decision procedures might well increase to doubly exponential. Nevertheless, all three views *can* actually be used to prove the NP upper bound of Theorem 26. This is the subject of three subsections below.

For several perspectives on integer programming, one can recommend books [81, 129, 145].

5.1 A view from geometry: discrete convex polyhedra

In this section we take the geometric view, following the ideas of von zur Gathen and Sieveking [55]. The resulting proof of Theorem 26 makes use of the principle we have already seen: first consider constraints over $\mathbb{R}$ (or $\mathbb{Q}$) instead of $\mathbb{Z}$.

We recall basic definitions from polyhedral geometry and convex analysis. For an introduction, the reader is referred to Lauritzen's undergraduate textbook [92] or lecture notes [91], and to Paffenholz's lecture notes [112]. More advanced material can be found in De Loera, Hemmecke, and Köppe [39], Rockafellar [127], and Schrijver [129].

A linear combination $\lambda_1 \boldsymbol{v}_1 + \ldots + \lambda_n \boldsymbol{v}_n$ of $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n \in \mathbb{R}^n$ is called:

- *conic* (or: positive) if all $\lambda_i \geqslant 0$,
- *affine* if $\lambda_1 + \ldots + \lambda_n = 1$, and
- *convex* if it is conic and affine.

Given a set $S \subseteq \mathbb{R}^d$, the set of all its conic (affine, convex) combinations is the *cone* generated by S, the *affine hull* and the *convex hull* of S, denoted $\operatorname{cone} S$, $\operatorname{aff} S$, and $\operatorname{conv} S$, respectively. We will use the Minkowski sum and product notation: $A + B = \{a + b : a \in A, b \in B\}$ and $A \cdot B = \{a \cdot b : a \in A, b \in B\}$.

The following famous theorem gives two equivalent characterisations of *convex polyhedra:*

▶ **Theorem 28** (Minkowski–Weyl, 1896, 1935). *For $P \subseteq \mathbb{R}^d$, the following are equivalent:*
(H) $P = \{\boldsymbol{x} : A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}\}$ *for some matrix A and vector $\boldsymbol{c}$;*
(V) $P = \operatorname{conv} F + \operatorname{cone} G$ *for some finite sets F and G.*

In words of Rockafellar [127, Section 19]: "This classical result is an outstanding example of a fact which is completely obvious to geometric intuition, but which wields important algebraic content and is not trivial to prove." The (V) $\Rightarrow$ (H) direction can in fact be obtained by a direct application of Fourier–Motzkin quantifier elimination (Section 3.3) over $\mathbb{R}$.

Representations (H) and (V) are referred to as *H-representation* and *V-representation* of P. In the former, P is the intersection of a finite collection of **h**alf-spaces. In the latter, elements of F can be thought of as **v**ertices and elements of G as directions in which P "is infinite" (directions of recession).

The *size* of each representation is the number of bits required to write it (that is, $A, \boldsymbol{c}$ or F, G) down. Let us assume henceforth that we deal with rational numbers only. An effective version of the Minkowski–Weyl theorem over $\mathbb{Q}$ states that the two representations can be translated from one to another; see, e.g., [129, Chapter 10] and [112, Chapter 5]:

▶ **Lemma 29.** *In Theorem 28:*
1. *A and $\boldsymbol{c}$ can be made rational if and only if F and G can be made rational.*
2. *Let P be nonempty and suppose either representation is given, of bit size s. Then the other can be computed, with the bit size of each number at most* $\operatorname{poly}(s)$.

We do not give a proof of Lemma 29. One of the possible paths is through the theory of structure of convex polyhedra: intuitively, in $\mathbb{R}^d$ each vertex (element of F) is determined as the intersection of d hyperplanes (facets), and each "infinite direction" (element of G) by the intersection of $d - 1$ hyperplanes. By Cramer's rule from linear algebra, solutions to systems of linear equations are formed by ratios of appropriately formed determinants. This yields a polynomial bound on the bit size of numbers. However, in degenerate situations (e.g., linear dependencies or underdetermined systems) some more work is needed.

In both translations (H) $\Rightarrow$ (V) and (V) $\Rightarrow$ (H), the total blow-up in size can be exponential while the size of individual numbers stays polynomial. An example is the "box" $[0,1]^d = \{(x_1, \dots, x_d) \in \mathbb{R}^d : 0 \leqslant x_i \leqslant 1 \text{ for all } i\}$, a convex polyhedron which has a short H-representation but 2^d vertices. A realisation of the translations is the double description method [92, Chapter 5], see also [112, Chapters 2–5] and [54, Chapter 5].

Let us move from $\mathbb{Q}$ to $\mathbb{Z}$. Recall the definition of hybrid linear sets $L(B, P)$ from the end of Section 4.1.

▶ **Theorem 30** (von zur Gathen and Sieveking [55]). *For $S \subseteq \mathbb{Z}^d$, the following are equivalent:*
(a) *S is a projection of $\{\boldsymbol{x} \in \mathbb{Z}^k : A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}\}$ for some $A \in \mathbb{Z}^{m \times k}$, $\boldsymbol{c} \in \mathbb{Z}^m$, and $k \geqslant d$;*
(b) *$S = L(C, Q)$ for some finite sets $C \subseteq \mathbb{Z}^d$ and $Q \subseteq \mathbb{Z}^d$.*
Moreover, suppose $P \neq \varnothing$ and either representation is given of bit size s. Then the other can be computed, with the bit size of each number at most $\operatorname{poly}(s)$.

The projection in Theorem 30 is orthogonal to the principal axes: some $k - d$ coordinates are simply crossed out. This theorem entails the NP upper bound of Theorem 26. It highlights and uses the idea that hybrid linear sets are a discrete analog of convex polyhedra. Indeed, the set

$$\Big\{ \sum \lambda_i \boldsymbol{b}_i + \sum \mu_j \boldsymbol{p}_j : \quad \sum \lambda_i = 1,\ \lambda_i \geqslant 0,\ \mu_j \geqslant 0 \Big\} \tag{10}$$

is the convex polyhedron $\operatorname{conv} B + \operatorname{cone} P$ if $\lambda_i, \mu_j \in \mathbb{R}$; and the hybrid linear set $L(B, P)$ if $\lambda_i, \mu_j \in \mathbb{Z}$.

Proof of Theorem 30. Direction (b) to (a) is straightforward. Auxiliary variables λ_i and μ_j in Eq. (10) correspond to coordinates that are crossed out (projected away).

For (a) to (b), we apply Lemma 29. Taking an arbitrary $\boldsymbol{x} \in \mathbb{Z}^k$ from the set $\{\boldsymbol{x} : A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}\} = \operatorname{conv} F + \operatorname{cone} G$, we can write it, for some $\lambda_i, \mu_j \geqslant 0$ such that $\sum \lambda_i = 1$, as

$$\boldsymbol{x} = \sum \lambda_i \boldsymbol{f}_i + \sum \mu_j \boldsymbol{g}_j = \Big(\sum \lambda_i \boldsymbol{f}_i + \sum (\mu_j - \lfloor \mu_j \rfloor) \boldsymbol{g}_j \Big) + \sum \lfloor \mu_j \rfloor \, \boldsymbol{g}_j,$$

where $\lfloor a \rfloor$ stands for the greatest integer not exceeding $a \in \mathbb{R}$, and $\boldsymbol{f}_i$, $\boldsymbol{g}_j$ are elements of F and G, respectively. Note that we can assume with no loss of generality that all vectors in G belong to $\mathbb{Z}^k$: indeed, they must be in $\mathbb{Q}^k$ by Lemma 29, and therefore each of them can be multiplied by the least common multiple of the denominators of its components. This shows that each vector $\lfloor \mu_j \rfloor \boldsymbol{g}_j$ is also in $\mathbb{Z}^k$. But then, since $\boldsymbol{x} \in \mathbb{Z}^k$, the difference of these two vectors (which is the vector in the big round brackets) is also integral. Therefore, $\boldsymbol{x}$ belongs to $L(C', Q')$, where Q' is the rescaled version of G and C' is the set of integer points in the Minkowski sum $\operatorname{conv} F + \sum [0, 1) \cdot \{\boldsymbol{g}_j\}$. Since F and G are finite sets, this sum is bounded and so C' is also finite. Finally, sets C and Q are obtained from C' and Q', respectively, by crossing out some of the components. We leave the verification of the "moreover" part of the statement to the reader. ◀

▶ **Example 31.** Consider the triangle example from Eq. (1), page 2, with formula $\varphi(x, y)$: $(x \leqslant 3y) \wedge (2y \leqslant x) \wedge (y \leqslant 2)$. All vertices of the triangle are actually integer points, see Fig. 1 (left). Consider, however, the modified formula $\varphi(x, y) \wedge (x \geqslant 1)$. The origin $(0, 0)$ is no longer a feasible point, and in fact the set of rational solutions is $\operatorname{conv}(\boldsymbol{u}', \boldsymbol{u}'', \boldsymbol{v}, \boldsymbol{w})$, where $\boldsymbol{v} = (4, 2)$ and $\boldsymbol{w} = (6, 2)$ are unchanged and $\boldsymbol{u}' = (1, 1/3)$ and $\boldsymbol{u}'' = (1, 1/2)$. The two new vertices are not integer points, and it is not difficult to imagine all of the “corners” of our polytope (polygon in this example) to be cut off. Feasibility of the integer program would thus hinge on combining a rational point from the convex hull, $\sum \lambda_i \boldsymbol{f}_i$ in the proof of Theorem 30, with a rational point from the cone, $\sum(\mu_j - \lfloor \mu_j \rfloor)\boldsymbol{g}_j$. ⌟

We now give an example showing that the number of generators of the hybrid linear set $L(C, Q)$ in Theorem 30 may be big, namely comparable to the magnitude of integers in the system $A \cdot \boldsymbol{x} \leqslant \boldsymbol{c}$.

▶ **Example 32.** Consider the set $S = \{(x, y, z) \in \mathbb{Z}^3 : (x+y-50z = 0)\wedge(x \geqslant 0)\wedge(y \geqslant 0)\}$. We have $S = L(\boldsymbol{0}, \{\boldsymbol{v}^0, \boldsymbol{v}^1, \dots, \boldsymbol{v}^{50}\})$, where $\boldsymbol{v}^i = (i, 50-i, 1)$. This blow-up in size is unavoidable: if $S = \bigcup_{i\in I} L(\boldsymbol{b}_i, P_i)$, then $|I| + |\bigcup_{i\in I} P_i| \geqslant 52$. Indeed, notice that $\bigcup_{i\in I} P_i \subseteq S \subseteq \mathbb{N}^3$. The first containment holds because S is defined by a conjunction of homogeneous constraints. Thus, $\boldsymbol{v}^m \notin L(\boldsymbol{0}, \bigcup_{i\in I} P_i \setminus \{\boldsymbol{v}^m\})$ for all m. So, whenever $\boldsymbol{v}^m \in L(\boldsymbol{b}_i, P_i)$, we have either $\boldsymbol{v}^m = \boldsymbol{b}_i$, or $\boldsymbol{v}^m \in P_i$ and $\boldsymbol{b}_i = \boldsymbol{0}$. This proves the inequality stated above.

In the absence of inequalities $x \geqslant 0$ and $y \geqslant 0$, we would instead have $S' = \{(x, y, z) \in \mathbb{Z}^3 : x + y - 50z\} = L(\boldsymbol{0}, \{\pm\boldsymbol{u}, \pm\boldsymbol{v}\})$, where $\boldsymbol{u} = (50, 0, 1)$ and $\boldsymbol{v} = (-1, 1, 0)$. Intuitively, in the definition of the set S the two inequalities prevent the vectors $\pm\boldsymbol{v}$ from becoming periods. More generally, a nonempty set $C \subseteq \mathbb{Z}^d$ defined by a system (conjunction) of linear equations with integer coefficients is a coset of a finitely generated *lattice* (an analog of an affine subspace, but over $\mathbb{Z}$ not $\mathbb{R}$). The number of generators of the lattice is $d - r$, where r is the rank of the system, which means that C can be written as a linear set with one base vector and $2(d - r)$ periods. ⌟

The set P of periods of a linear set $L(\boldsymbol{0}, P)$ is also known as the *Hilbert basis*. If the positive cone of the set is pointed, then there is a unique Hilbert basis of minimal size [39, Corollary 2.6.4]. Software tools and libraries that can compute Hilbert bases are available, e.g., Normaliz [28, 27] and polymake [7].

Ideas presented in this section tend to be very useful, and so is Theorem 30. Another presentation is [39, Section 2.6]. Some recent extensions and applications can be found in [94, 34, 40, 36]. Further development of ideas presented in this section have recently led to a new quantifier elimination procedure for Presburger arithmetic [68].

5.2 A view from automata theory: pumping lemma

In Section 3.2 we have described and used representation of integers base $k \geqslant 2$, for so-called k-automatic sets. The author is not aware of an argument proving the NP upper bound of Theorem 26 using such representations. Indeed, extension of this NP upper bound to existential Büchi arithmetic [62, 66] rather required *building upon* Theorem 26 (or, more precisely, Theorem 30). We present a slightly different approach instead, which can also be interpreted as automata-theoretic, but in the unary representation (base 1). That is, $4 \in \mathbb{N}$ is represented as $1 + 1 + 1 + 1$ in this numeration system.

It is a well-known fact that the integer programming problem can equivalently be formulated as follows:

Input: A system $A \cdot \boldsymbol{x} = \boldsymbol{b}$ of linear equations, with A an integer matrix and $\boldsymbol{b}$ an integer vector.
Output: Does the system have a solution over $\mathbb{N}$?

As a reminder, we use $\mathbb{N} = \{0, 1, \dots\}$ in this paper. Our goal is to bound the norm of smallest solutions from above: this is easy if all entries of A have the same sign, but not necessarily obvious in general.

For $\boldsymbol{v} = (v_1, \ldots, v_d) \in \mathbb{R}^d$, denote by $\|\boldsymbol{v}\|$ its Euclidean length. In fact, any norm would work just as well.

The following classical theorem is sometimes known as the Steinitz lemma.

▶ **Theorem 33** (Steinitz lemma). *Let $d \geqslant 1$ and let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be a sequence of vectors from $\mathbb{R}^d$. Assume $\|\boldsymbol{v}_i\| \leqslant 1$ for all i. If $\sum_{i=1}^n \boldsymbol{v}_i = \boldsymbol{0}$, then by reordering the vectors we can obtain another sequence $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ such that $\|\sum_{i=1}^k \boldsymbol{u}_i\| \leqslant d$ for all $k = 0, \ldots, n$.*

It is not immediately clear why this is true. Think of each $\boldsymbol{v}_i$ as a vector of travel (movement), and of a sequence of vectors as a travel itinerary. If we start from the origin $\boldsymbol{0}$, then following the entire sequence takes us back to the origin. Reordering the vectors will not change this, but can we reorder so that we never go far from $\boldsymbol{0}$ on the way?

It is rather remarkable that the distance can be kept small, at most d, no matter how big or small n is. Matoušek's wonderful book on applications of linear algebra in combinatorics, geometry, and computer science presents a short proof [102, Miniature 20]. The upper bound of d is due to Grinberg and Sevastyanov [61].

Let us apply the Steinitz lemma to integer programming, following Eisenbrand and Weismantel [49]. For simplicity, consider a homogeneous equation $A \cdot \boldsymbol{x} = \boldsymbol{0}$ first. To show an upper bound on the norm of *smallest* nonzero solutions over $\mathbb{N}$, take *some* solution $\boldsymbol{x} = (p_1, \ldots, p_s)$, where s is the number of variables in the system and all $p_i \in \mathbb{N}$. Let $n = p_1 + \ldots + p_s$ and let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be the sequence of columns of A with the ith column taken p_i times. (This is the “unary” representation of solution $(p_1, \ldots, p_s)$.) As we know that $A \cdot (p_1, \ldots, p_s) = \boldsymbol{0}$, we can apply the Steinitz lemma to scaled vectors $\boldsymbol{p}_i / \max_i \|\boldsymbol{p}_i\|$. In the reordered sequence $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ (without scaling: each $\boldsymbol{u}_i$ is some $\boldsymbol{v}_j$), all partial sums $\sum_{i=1}^k \boldsymbol{u}_i$ do not deviate much from $\boldsymbol{0}$; here $k = 0, \ldots, n$. Instead of $\leqslant d$ in Theorem 33, we get $\leqslant H \overset{\text{def}}{=} d \cdot \max_i \|\boldsymbol{p}_i\|$ instead.

The remaining part of the argument is simple. Recall that all $\boldsymbol{u}_i$ are integer vectors, and there are not too many integer points in $\mathbb{R}^m$ with norm at most H; let us say at most $N_m(H)$ such points. (Here m is the number of equations in the system.)

- If $n > N_m(H)$, then some point apart from $\boldsymbol{0}$ is visited twice (or $\boldsymbol{0}$ is visited three times or more). Hence a part of the sequence $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ can be removed without affecting the total sum. In other words, the solution $\boldsymbol{x}$ was not minimal to start with.
- Otherwise $n \leqslant N_m(H)$, and this is already an upper bound on the norm of solution.

We omit the calculations (see, e.g., [49]), but the bounds are sufficient for the NP upper bound of Theorem 26. It remains to remark that nonzero $\boldsymbol{b}$ in $A \cdot \boldsymbol{x} = \boldsymbol{b}$ can be incorporated in the argument too, for example by putting $-\boldsymbol{b}$ into the sequence.

The argument in the bullet list is reminiscent of the pumping lemma in automata theory: in a finite-state automaton, every sufficiently long accepting run must repeat a state and can therefore be shortened.

For further reading and applications, we refer to [8, 110].

5.3 A view from symbolic computation: Gaussian elimination

In 2024, two new approaches to quantifier elimination for existential Presburger arithmetic were proposed. Each can be thought of as a nondeterministic polynomial-time rewriting procedure for existential formulas. Run deterministically, the procedures output quantifier-free formulas of exponential size; polynomial-size formulas can instead be produced in an appropriately extended syntax. One of the procedures, by Chistikov, Mansutti, and Starchak [36], is based on Gaussian elimination and the other, by Haase, Krishna, Madnani, Mishra, and Zetzsche [68], on geometry of convex polyhedra. We outline key ideas behind the former, focusing on integer programming (conjunctive formulas).

Fraction-free Gaussian elimination. The approach develops a remark by Weispfenning [142, Corollary 4.3]. The basis of the algorithm is Gaussian elimination, a standard method for solving systems of linear equations. It is well known that, over rational numbers ($\mathbb{Q}$), it can be performed in polynomial time. An elegant argument showing that each number appearing in the computation is a ratio of two minors (sub-determinants) of the original matrix is given in Schrijver's book [129, Section 3.3]. Thus, the bit size of numbers stays bounded by a polynomial in the bit size of the input.

We need instead a version of Gaussian elimination for integers ($\mathbb{Z}$). The simplest *division-free* algorithm works as follows: given equations $ax + by + \ldots = 0$ and $cx + dy + \ldots = 0$, to eliminate x we "cross-multiply" them by c and a, respectively, and then subtract one from the other (cf. Observation 20). The result is

$$(ad - bc)\, y + \ldots = 0. \tag{11}$$

The bit size of coefficients may double, and repeated cross-multiplication is known to lead (in the worst case) to numbers of doubly exponential magnitude, i.e., of exponential bit size.

As it turns out, there exists a method avoiding this blow-up, which appears in the works of Edmonds [47] and Bareiss [10] and was apparently known as early as 1888 to Clasen [6]. The following question is instructive. The coefficient at y in Eq. (11) is a 2×2 determinant. Gaussian elimination can be seen as computing many such 2×2 determinants, inductively. *How* does the determinant of a big (say $n \times n$) square matrix, which can be computed using Gaussian elimination, arise in this process?

The answer appears in a beautiful paper by Dodgson [41]. Let $A = (a_{ij})$ be a 3×3 matrix. Let us apply the first two steps of Gaussian elimination: after the first step, the coefficient matrix becomes

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{11}a_{22} - a_{12}a_{21} & a_{11}a_{23} - a_{13}a_{21} \\ 0 & a_{11}a_{32} - a_{12}a_{31} & a_{11}a_{33} - a_{13}a_{31} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ 0 & \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} \end{pmatrix}$$

and then, after the second elimination step, the entry in position $(3, 3)$ becomes

$$\begin{vmatrix} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} \end{vmatrix}.$$

By (a special case of) an identity due to Sylvester and Jacobi–Desnanot (see, e.g., [10, 41, 82]), this coefficient is actually equal to the product $a_{11} \cdot \det A$. So, when division-free Gaussian elimination is run on a bigger matrix, after *two* initial steps all coefficients in row 3 and below become *divisible* by a_{11}, as long as all original entries of A are integers. We can divide by a_{11}, and carry on, performing similar divisions after each step. The result is a *fraction-free* Gaussian elimination [47, 10].

Handling inequalities with slack variables and nondeterminism. A second ingredient is required to handle inequalities. We turn each of them into an equality, introducing *slack variables* as, e.g., in our proof sketch for Theorem 13. In the run of Gaussian elimination, each iteration takes an equality from the system and eliminates one actual (non-slack) variable.

If this equality contains a slack variable not handled so far, a small value for this slack variable is guessed. The fact that small values (i.e., of polynomial bit size) suffice requires a proof. These guessed values have the same nature as remainders r in Example 18, or shifts $+0, \dots, +(M-1)$ in Remark 19.

For further details, we refer the reader to the paper [36]. The algorithm can be extended to arbitrary existential Presburger formulas, in which case (i) the shifts assume both positive and negative values, and (ii) the equation to pick at each iteration is guessed nondeterministically. Unlike [68], the paper [36] does not explicitly describe the formula produced by elimination process but shows how to integrate the algorithm into an NP decision procedure for an extension of existential Presburger arithmetic with nonlinear constraints of the forms $y = 2^x$ and $z = (x \bmod 2^y)$. (This extension was previously shown decidable by automata-theoretic methods [43].) Paper [68] draws a multitude of other consequences from efficient quantifier elimination.

6 Further directions

There are several sources to help new people enter this field. Bradley and Manna's textbook [25] offers a detailed introduction to computational logic, including logical theories of arithmetic, and applications in program verification. Kroening and Strichman's textbook [89] focuses on decision procedures and includes a chapter on quantifier-free linear real and integer arithmetic and another on handling quantified formulas. Kirby's recent textbook on model theory [84] targets an audience of undergraduate and Master's students, assuming little in the way of background knowledge beyond the basics of abstract algebra.

Haase's survey [65] provides a variety of references on Presburger arithmetic. An earlier survey by Bès [18] gives an overview of definability and decidability questions for arithmetic theories more broadly. Michaux and Villemaire's survey of open questions [104], with a focus on Presburger and Büchi arithmetic (see Section 3.2), instigated several subsequent developments. For arithmetic theories that include both addition and multiplication, beyond the literature mentioned at the end of Section 3.3 we refer the reader to Poonen's lecture notes on Hilbert's 10th problem over rings [120] and a recent article by Pasten [115]. An early reference on the computational complexity of logical theories is Ferrante and Rackoff's monograph [52].

We already highlighted (in Section 1) several sources on satisfiability modulo theories (SMT) solving. Among further such sources focusing on logical theories of arithmetic are an overview [3] and an evolving electronic book on SAT and SMT [147] containing many worked examples and aimed at programmers. A new self-contained overview of the arithmetic engine of Z3 (a successful SMT solver and theorem prover) has recently appeared [20].

Applications in verification. An early application of Presburger arithmetic appeared in the work of Parikh [114] and Ginsburg and Spanier [57, 58]. Parikh considered the commutative image, nowadays also known as Parikh image, of formal languages. For example, the commutative image of a word $w \in \{a, b\}^*$ is a 2-dimensional vector $\psi(w) = (m, n)$ such that the letters a and b occur in w exactly m and n times, respectively: $\psi(abbab) = (2, 3)$. The commutative image of a language $L \subseteq \{a, b\}^*$ is $\psi(L) = \{\psi(w) : w \in L\}$. Parikh's theorem shows that the commutative image of every context-free language forms a *semi-linear set*, and in fact for every semi-linear set S there exists a regular language with commutative image S. Ginsburg and Spanier later proved that semi-linear sets are exactly sets definable in Presburger arithmetic [58]; see Section 3.1.

Intuitively, context-free languages can be used to model procedure calls and recursion in software. More generally, linear arithmetic constraints can capture periodic and ultimately periodic behaviours. A well-known result constructs a polynomial-size existential Presburger formula for the Parikh image of a context-free language, see [131, 140] and [75]. This has further applications and extensions [67, 51, 74, 79, 72]. Thanks to the closure properties of semi-linear sets, in the algorithmic foundations of verification "expressible in Presburger arithmetic" often means tractable: complicated behaviours of systems can be approximated and analysed with the help of this logic (see, e.g., [1, 73, 16]).

Extensions and other arithmetic theories. Ongoing research on arithmetic theories extends decidability to more expressive theories and characterises the complexity of decidable ones.

Some extensions of Presburger arithmetic do not, in fact, change the family of definable sets, but increase the succinctness of the logic. For Presburger arithmetic with counting quantifiers [130, 71] and with the star operator [118, 69], complexity questions are open.

As already mentioned, adding arbitrary polynomial constraints to Presburger arithmetic makes the theory undecidable. There is, however, a multitude of decidable theories. Such is the extension of linear integer arithmetic with exponentiation base 2 [133, 32, 119], see also [146, 15, 36]. Effectively, this means a new type of constraints is allowed, namely those of the form $y = 2^x$, where x, y are variables. The choice of integer base $k \geqslant 2$ is immaterial but needs to remain the same throughout the formula.

Alternatively, the logic can permit, in addition to linear integer constraints, assertions of the form "x is a Fibonacci number" as basic building blocks. Such building blocks are referred to as predicates. Many other *sparse* predicates can be added instead of the Fibonacci one, whilst keeping the resulting theory decidable [132, 99]. For the predicate "x is a power of 2", an algorithm with elementary running time has been found [15]. On the theme of adding several "powers" predicates simultaneously, recent references are [77] and [83].

Recall that linear integer arithmetic includes predicates for divisibility by fixed integers. Adding the full divisibility predicate "x divides y" (where x and y are variables) renders the theory undecidable [126, 18], whilst keeping decidable its *existential fragment* (sentences with existential quantifiers only, all of which must appear at the beginning of the formula without negations in between). Whether the decision problem for this fragment belongs to NP is an open question; see, e.g., [94, 138, 116, 40]. In comparison, if instead of divisibility $(x \mid y)$ we add multiplication $(x \cdot y = z)$, then even the existential fragment becomes undecidable, by a celebrated result due to Davis, Putnam, Robinson, and Matiyasevich (negative resolution of Hilbert's 10th problem); see, e.g., [120].

Famous open problems in number theory can be expressed using sentences in linear integer arithmetic extended with a predicate $P(x)$ asserting that x is prime. Indeed, the twin prime conjecture is expressed with the sentence $\forall x \, \exists y \, [(y \geqslant x) \wedge P(y) \wedge P(y+2)]$, and the Goldbach conjecture with the sentence $\forall x \, \exists y \, [(x \leqslant 2) \vee (P(y) \wedge P(x-y))]$. It remains open whether Presburger arithmetic with the primes predicate (P) is decidable; see, e.g., [31, 93].

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Roland Meyer, and Mehdi Seyed Salehi. What's decidable about availability languages? In *FSTTCS*, pages 192–205, 2015. doi:10.4230/LIPICS.FSTTCS.2015.192.

2 Erika Ábrahám, József Kovács, and Anne Remke. SMT: something you must try. In *iFM*, pages 3–18, 2023. doi:10.1007/978-3-031-47705-8_1.

3 Erika Ábrahám and Gereon Kremer. SMT solving for arithmetic theories: Theory and tool support. In *SYNASC*, pages 1–8, 2017. `doi:10.1109/SYNASC.2017.00009`.

4 Jorge L. Ramírez Alfonsín. Complexity of the Frobenius problem. *Combinatorica*, 16(1):143–147, 1996. `doi:10.1007/BF01300131`.

5 Jorge L. Ramírez Alfonsín. *The Diophantine Frobenius problem.* Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. `doi:10.1093/acprof:oso/9780198568209.001.0001`.

6 Steven C. Althoen and Renate McLaughlin. Gauss-Jordan reduction: A brief history. *Am. Math. Mon.*, 94(2):130–142, 1987.

7 Benjamin Assarf, Ewgenij Gawrilow, Katrin Herr, Michael Joswig, Benjamin Lorenz, Andreas Paffenholz, and Thomas Rehn. Computing convex hulls and counting integer points with `polymake`. *Math. Program. Comput.*, 9(1):1–38, 2017. `doi:10.1007/S12532-016-0104-Z`.

8 Imre Bárány. On the power of linear dependencies. In *Building Bridges: Between Mathematics and Computer Science*, volume 19 of *Bolyai Society Mathematical Studies*, pages 31–45. Springer, 2008. `doi:10.1007/978-3-540-85221-6_1`.

9 Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS*, pages 415–442, 2022. `doi:10.1007/978-3-030-99524-9_24`.

10 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`.

11 Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability — Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 1267–1329. IOS Press, 2021. `doi:10.3233/FAIA201017`.

12 Clark W. Barrett, Cesare Tinelli, Haniel Barbosa, Aina Niemetz, Mathias Preiner, Andrew Reynolds, and Yoni Zohar. Satisfiability modulo theories: A beginner's tutorial. In *FM*, pages 571–596, 2024. `doi:10.1007/978-3-031-71177-0_31`.

13 Alexander Barvinok. *Integer points in polyhedra.* EMS Press, 2008. `doi:10.4171/052`.

14 Alexander I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Math. Oper. Res.*, 19(4):769–779, 1994. `doi:10.1287/MOOR.19.4.769`.

15 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`.

16 Pascal Bergsträßer, Moses Ganardi, Anthony W. Lin, and Georg Zetzsche. Ramsey quantifiers in linear arithmetics. *Proc. ACM Program. Lang.*, 8(POPL):1–32, 2024. `doi:10.1145/3632843`.

17 Leonard Berman. The complexity of logical theories. *Theor. Comput. Sci.*, 11:71–77, 1980. `doi:10.1016/0304-3975(80)90037-7`.

18 Alexis Bès. A survey of arithmetical definability. *A tribute to Maurice Boffa. B. Belg. Math. Soc.-Sim.*, suppl.:1–54, 2001. URL: `http://lacl.u-pec.fr/bes/publi/survey.pdf`.

19 Alexis Bès and Christian Choffrut. Theories of real addition with and without a predicate for integers. *Log. Methods Comput. Sci.*, 17(2), 2021. `doi:10.23638/LMCS-17(2:18)2021`.

20 Nikolaj S. Bjørner and Lev Nachmanson. Arithmetic solving in Z3. In *CAV, part I*, pages 26–41, 2024. `doi:10.1007/978-3-031-65627-9_2`.

21 Bernard Boigelot, Isabelle Mainz, Victor Marsault, and Michel Rigo. An efficient algorithm to decide periodicity of b-recognisable sets using MSDF convention. In *ICALP*, pages 118:1–118:14, 2017. `doi:10.4230/LIPICS.ICALP.2017.118`.

22 Bernard Boigelot and Pierre Wolper. Representing arithmetic constraints with finite automata: An overview. In *ICLP*, LNCS, pages 1–19, 2002. `doi:10.1007/3-540-45619-8_1`.

23 George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic*. Cambridge University Press, 5th edition, 2007. doi:10.1017/CBO9780511804076.

24 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:10.1090/S0002-9939-1976-0396605-3.

25 Aaron R. Bradley and Zohar Manna. *The calculus of computation: Decision procedures with applications to verification*. Springer, 2007. doi:10.1007/978-3-540-74113-8.

26 Martin Bromberger. *Decision Procedures for Linear Arithmetic*. PhD thesis, Saarland University, Saarbrücken, Germany, 2019. URL: https://tel.archives-ouvertes.fr/tel-02427371.

27 Winfried Bruns, Bogdan Ichim, and Christof Söger. The power of pyramid decomposition in Normaliz. *J. Symb. Comput.*, 74:513–536, 2016. doi:10.1016/J.JSC.2015.09.003.

28 Winfried Bruns, Bogdan Ichim, Christof Söger, and Ulrich von der Ohe. Normaliz. Algorithms for rational cones and affine monoids. URL: https://www.normaliz.uni-osnabrueck.de/.

29 Véronique Bruyère, Georges Hansel, Christian Michaux, and Roger Villemaire. Logic and p-recognizable sets of integers. *B. Belg. Math. Soc.-Sim.*, 1(2):191–238, 1994. doi:10.36045/bbms/1103408547.

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

31 Patrick Cegielski, Denis Richard, and Maxim Vsemirnov. On the additive theory of prime numbers II. In *CSIT 2005 (Computer Science and Information Technologies, September 19–23, 2005, Yerevan, Armenia)*, pages 39–47, 2005. doi:10.48550/arXiv.math/0609554.

32 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.

33 Dmitry Chistikov and Christoph Haase. The taming of the semi-linear set. In *ICALP*, pages 128:1–128:13, 2016. doi:10.4230/LIPICS.ICALP.2016.128.

34 Dmitry Chistikov and Christoph Haase. On the complexity of quantified integer programming. In *ICALP*, pages 94:1–94:13, 2017. doi:10.4230/LIPICS.ICALP.2017.94.

35 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.

36 Dmitry Chistikov, Alessio Mansutti, and Mikhail R. Starchak. Integer linear-exponential programming in NP by quantifier elimination. In *ICALP*, pages 132:1–132:20, 2024. doi:10.4230/LIPICS.ICALP.2024.132.

37 David C. Cooper. Programs for mechanical program verification. In *Machine Intelligence*, volume 6, pages 43–59. Edinburgh University Press, 1971.

38 David C. Cooper. Theorem proving in arithmetic without multiplication. In *Machine Intelligence*, volume 7, pages 91–99. Edinburgh University Press, 1972.

39 Jesús A. De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and geometric ideas in the theory of discrete optimization*, volume MO14 of *MOS-SIAM Series on Optimization*. SIAM and MOS, 2013. doi:10.1137/1.9781611972443.

40 Rémy Défossez, Christoph Haase, Alessio Mansutti, and Guillermo A. Pérez. Integer programming with GCD constraints. In *SODA*, pages 3605–3658, 2024. doi:10.1137/1.9781611977912.128.

41 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.

42 Andreas Dolzmann and Thomas Sturm. REDLOG: computer algebra meets computer logic. *SIGSAM Bull.*, 31(2):2–9, 1997. doi:10.1145/261320.261324.

43 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.

44 Lou van den Dries. Alfred Tarski's elimination theory for real closed fields. *J. Symb. Log.*, 53(1):7–19, 1988. doi:10.1017/S0022481200028899.

45 Antoine Durand-Gasselin and Peter Habermehl. On the use of non-deterministic automata for Presburger arithmetic. In *CONCUR*, pages 373–387, 2010. doi:10.1007/978-3-642-15375-4_26.

46 Antoine Durand-Gasselin and Peter Habermehl. Ehrenfeucht-Fraïssé goes elementarily automatic for structures of bounded degree. In *STACS*, pages 242–253, 2012. doi:10.4230/LIPIcs.STACS.2012.242.

47 Jack Edmonds. Systems of distinct representatives and linear algebra. *J. Res. NBS-B. Math. Sci.*, 71B(4):241–245, 1967. doi:10.6028/jres.071B.033.

48 Friedrich Eisenbrand and Gennady Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006. doi:10.1016/J.ORL.2005.09.008.

49 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020. doi:10.1145/3340322.

50 Javier Esparza and Michael Blondin. *Automata theory: An algorithmic approach.* MIT Press, 2023.

51 Javier Esparza and Pierre Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL*, pages 499–510, 2011. doi:10.1145/1926385.1926443.

52 Jeanne Ferrante and Charles W. Rackoff. *The computational complexity of logical theories*, volume 718 of *Lecture Notes in Mathematics.* Springer, 1979. doi:10.1007/BFb0062837.

53 Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In *Complexity of Computation*, volume 7 of *SIAM-AMS Proceedings*, pages 27–41. AMS, 1974.

54 Jean Gallier and Jocelyn Quaintance. Aspects of convex geometry: Polyhedra, linear programming, shellings, Voronoi diagrams, Delaunay triangulations, 2022. URL: https://www.cis.upenn.edu/~jean/gbooks/convexpoly.html.

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

56 Seymour Ginsburg. *The mathematical theory of context-free languages.* McGraw-Hill, 1966.

57 Seymour Ginsburg and Edwin H. Spanier. Bounded ALGOL-like languages. *Trans. Am. Math. Soc.*, 113(2):333–368, 1964. doi:10.1090/S0002-9947-1964-0181500-1.

58 Seymour Ginsburg and Edwin H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966. doi:10.2140/pjm.1966.16.285.

59 Erich Grädel. Dominoes and the complexity of subclasses of logical theories. *Ann. Pure Appl. Log.*, 43(1):1–30, 1989. doi:10.1016/0168-0072(89)90023-7.

60 Erich Grädel. Automatic structures: Twenty years later. In *LICS*, pages 21–34, 2020. doi:10.1145/3373718.3394734.

61 Victor S. Grinberg and Sergey V. Sevast'yanov. Value of the Steinitz constant. *Funct. Anal. Appl.*, 14(2):125–126, 1980. doi:10.1007/BF01086559.

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

63 Roland Guttenberg, Mikhail A. Raskin, and Javier Esparza. Geometry of reachability sets of vector addition systems. In *CONCUR*, pages 6:1–6:16, 2023. doi:10.4230/LIPICS.CONCUR.2023.6.

64 Christoph Haase. Subclasses of Presburger arithmetic and the weak EXP hierarchy. In *CSL-LICS*, pages 47:1–47:10, 2014. doi:10.1145/2603088.2603092.

65 Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. URL: https://www.cs.ox.ac.uk/people/christoph.haase/home/publication/haa-18/haa-18.pdf, doi:10.1145/3242953.3242964.

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

67 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *CONCUR*, pages 369–383, 2009. doi:10.1007/978-3-642-04081-8_25.

68 Christoph Haase, Shankara Narayanan Krishna, Khushraj Madnani, Om Swostik Mishra, and Georg Zetzsche. An efficient quantifier elimination procedure for Presburger arithmetic. In *ICALP*, pages 142:1–142:17, 2024. doi:10.4230/LIPICS.ICALP.2024.142.

69 Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. In *LICS*, 2019. doi:10.1109/LICS.2019.8785850.

70 Peter Habermehl, Vojtech Havlena, Michal Hecko, Lukás Holík, and Ondrej Lengál. Algebraic reasoning meets automata in solving linear integer arithmetic. In *CAV, part I*, pages 42–67, 2024. doi:10.1007/978-3-031-65627-9_3.

71 Peter Habermehl and Dietrich Kuske. On Presburger arithmetic extended with non-unary counting quantifiers. *Log. Methods Comput. Sci.*, 19(3), 2023. doi:10.46298/LMCS-19(3:4)2023.

72 Matthew Hague, Artur Jez, and Anthony W. Lin. Parikh's theorem made symbolic. *Proc. ACM Program. Lang.*, 8(POPL):1945–1977, 2024. doi:10.1145/3632907.

73 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.

74 Matthew Hague and Anthony Widjaja Lin. Model checking recursive programs with numeric data types. In *CAV*, pages 743–759, 2011. doi:10.1007/978-3-642-22110-1_60.

75 Matthew Hague and Anthony Widjaja Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV*, pages 260–276, 2012. Full version: https://anthonywlin.github.io/papers/cav12-long.pdf. doi:10.1007/978-3-642-31424-7_22.

76 Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona: Monadic second-order logic in practice. In *TACAS*, pages 89–110, 1995. doi:10.1007/3-540-60630-0_5.

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

78 Thiet-Dung Huynh. The complexity of semilinear sets. *Elektron. Inf.verarb. Kybern.*, 18(6):291–338, 1982.

79 Petr Janku and Lenka Turonová. Solving string constraints with approximate Parikh image. In *EUROCAST (I)*, volume 12013, pages 491–498, 2019. doi:10.1007/978-3-030-45093-9_59.

80 Klaus Jansen and Kim-Manuel Klein. About the structure of the integer cone and its application to bin packing. *Math. Oper. Res.*, 45(4):1498–1511, 2020. doi:10.1287/MOOR.2019.1040.

81 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. doi:10.1007/978-3-540-68279-0.

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

83 Toghrul Karimov, Florian Luca, Joris Nieuwveld, Joël Ouaknine, and James Worrell. On the decidability of Presburger arithmetic expanded with powers. In *SODA*, 2025. To appear. arXiv:2407.05191.

84 Jonathan Kirby. *An invitation to model theory*. Cambridge University Press, 2019. doi:10.1017/9781316683002.

85 Felix Klaedtke. Bounds on the automata size for Presburger arithmetic. *ACM Trans. Comput. Log.*, 9(2):11:1–11:34, 2008. doi:10.1145/1342991.1342995.

86 Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual.* BRICS, Department of Computer Science, University of Aarhus, January 2001. Notes Series NS-01-1. URL: http://www.brics.dk/mona/.

87 Dexter Kozen. *Theory of computation.* Texts in Computer Science. Springer, 2006. doi:10.1007/1-84628-477-5.

88 Georg Kreisel and Jean-Louis Krivine. *Elements of mathematical logic (model theory).* North Holland, 1967.

89 Daniel Kroening and Ofer Strichman. *Decision procedures: An algorithmic point of view.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2nd edition, 2016. doi:10.1007/978-3-662-50497-0.

90 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.

91 Niels Lauritzen. Lectures on convex sets, 2009. URL: https://users.fmf.uni-lj.si/lavric/lauritzen.pdf.

92 Niels Lauritzen. *Undergraduate convexity: From Fourier and Motzkin to Kuhn and Tucker*. World Scientific, 2013. doi:10.1142/8527.

93 Thierry Lavendhomme and Arnaud Maes. Note on the undecidability of $\langle \omega; +, P_{m,r} \rangle$. In *Definability in Arithmetics and Computability*, volume 11 of *Cahiers du Centre de logique*, pages 61–68. Academia-Bruylant, 2000. URL: http://www.cahiersdelogique.be/cahier11angl.html.

94 Antonia Lechner, Joël Ouaknine, and James Worrell. On the complexity of linear arithmetic with divisibility. In *LICS*, pages 667–676, 2015. doi:10.1109/LICS.2015.67.

95 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.

96 Jérôme Leroux. A polynomial time Presburger criterion and synthesis for number decision diagrams. In *LICS*, pages 147–156, 2005. doi:10.1109/LICS.2005.2.

97 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL*, pages 307–316, 2011. doi:10.1145/1926385.1926421.

98 Jérôme Leroux and Gérald Point. TaPAS: The Talence Presburger arithmetic suite. In *TACAS*, pages 182–185, 2009. doi:10.1007/978-3-642-00768-2_18.

99 Arnaud Maes. Revisiting Semenov's results about decidability of extensions of Presburger arithmetic. In *Definability in Arithmetics and Computability*, volume 11 of *Cahiers du Centre de logique*, pages 11–59. Academia-Bruylant, 2000. URL: http://www.cahiersdelogique.be/cahier11angl.html.

100 Victor Marsault and Jacques Sakarovitch. Ultimate periodicity of b-recognisable sets: A quasilinear procedure. In *DLT*, pages 362–373, 2013. doi:10.1007/978-3-642-38771-5_32.

101 Jiří Matoušek. *Lectures on discrete geometry.* Graduate Texts in Mathematics. Springer, 2002. doi:10.1007/978-1-4613-0039-7.

102 Jiří Matoušek. *Thirty-three miniatures: Mathematical and algorithmic applications of linear algebra*, volume 53 of *Student Mathematical Library.* AMS, 2010. doi:10.1090/stml/053.

103 Jirí Matoušek. Intersection graphs of segments and $\exists\mathbb{R}$, 2014. arXiv:1406.2636.

104 Christian Michaux and Roger Villemaire. Open questions around Büchi and Presburger arithmetics. In *Logic: from Foundations to Applications: European logic colloquium*, pages 353–384. Oxford University Press, 1996. doi:10.1093/oso/9780198538622.003.0015.

105 Jasper Nalbach, Valentin Promies, Erika Ábrahám, and Paul Kobialka. FMplex: A novel method for solving linear real arithmetic problems. In *GandALF*, pages 16–32, 2023. doi:10.4204/EPTCS.390.2.

106 Danny Nguyen and Igor Pak. Enumerating projections of integer points in unbounded polyhedra. *SIAM J. Discret. Math.*, 32(2):986–1002, 2018. doi:10.1137/17M1118907.

107 Danny Nguyen and Igor Pak. Short Presburger arithmetic is hard. *SIAM J. Comput.*, 51(2):1–31, 2022. doi:10.1137/17M1151146.

108 Phong Q. Nguyen and Brigitte Vallée, editors. *The LLL algorithm: Survey and applications.* Information Security and Cryptography. Springer, 2010. doi:10.1007/978-3-642-02295-1.

109 Tobias Nipkow. Linear quantifier elimination. *J. Autom. Reason.*, 45(2):189–212, 2010. doi:10.1007/S10817-010-9183-0.

110 Timm Oertel, Joseph Paat, and Robert Weismantel. A colorful Steinitz lemma with application to block-structured integer programs. *Math. Program.*, 204(1):677–702, 2024. doi:10.1007/S10107-023-01971-3.

111 Derek C. Oppen. A $2^{2^{2^{pn}}}$ upper bound on the complexity of Presburger arithmetic. *J. Comput. Syst. Sci.*, 16(3):323–332, 1978. doi:10.1016/0022-0000(78)90021-1.

112 Andreas Paffenholz. Polyhedral geometry and linear optimization (summer semester 2010), 2013. URL: http://www.mathematik.tu-darmstadt.de/~paffenholz/data/preprints/geometry_of_optimization.pdf.

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

114 Rohit Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364.

115 Hector Pasten. Definability and arithmetic. *Notices of the AMS*, 70(9):1385–1393, 2023. doi:10.1090/noti2777.

116 Guillermo A. Pérez and Ritam Raha. Revisiting parameter synthesis for one-counter automata. In *CSL*, pages 33:1–33:18, 2022. doi:10.4230/LIPICS.CSL.2022.33.

117 Jean-Éric Pin, editor. *Handbook of automata theory. Volume II. Automata in mathematics and selected applications.* EMS Press, 2021. doi:10.4171/AUTOMATA-2.

118 Ruzica Piskac and Viktor Kuncak. Linear arithmetic with stars. In *CAV*, pages 268–280, 2008. doi:10.1007/978-3-540-70545-1_25.

119 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.

120 Bjorn Poonen. Hilbert's Tenth Problem over rings of number-theoretic interest. *Notes from the lectures at the Arizona Winter School "Logic and Number Theory"*, 2003. URL: http://math.mit.edu/~poonen/papers/aws2003.pdf.

121 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, Varsovie, 1929*, pages 92–101. Skład Główny, 1930.

122 Mojżesz Presburger and Dale Jabcquette. On the completeness of a certain system of arithmetic of whole numbers in which addition occurs as the only operation. *History and Philosophy of Logic*, 12(2):225–233, 1991. Translation of ref. [121] and commentary. doi:10.1080/014453409108837187.

123 William W. Pugh. A practical algorithm for exact array dependence analysis. *Commun. ACM*, 35(8):102–114, 1992. doi:10.1145/135226.135233.

124 Andrew Reynolds, Tim King, and Viktor Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods Syst. Des.*, 51(3):500–532, 2017. doi:10.1007/S10703-017-0290-Y.

125 Michel Rigo. Numeration systems: A link between number theory and formal language theory. In *DLT*, pages 33–53, 2010. doi:10.1007/978-3-642-14455-4_6.

126 Julia Robinson. Definability and decision problems in arithmetic. *J. Symb. Log.*, 14(2):98–114, 1949. doi:10.2307/2266510.

127 R. Tyrrell Rockafellar. *Convex analysis.* Princeton University Press, 1970. doi:10.1515/9781400873173.

128 Bruno Scarpellini. Complexity of subcases of Presburger arithmetic. *Trans. Am. Math. Soc.*, 284(1):203–218, 1984. doi:10.1090/S0002-9947-1984-0742421-9.

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

130 Nicole Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Trans. Comput. Log.*, 6(3):634–671, 2005. `doi:10.1145/1071596.1071602`.

131 Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In *ICALP*, pages 1136–1149, 2004. `doi:10.1007/978-3-540-27836-8_94`.

132 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`.

133 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`.

134 Jeffrey Shallit. *The logical approach to automatic sequences: Exploring combinatorics on words with* `Walnut`, volume 482 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, 2022. `doi:10.1017/9781108775267`.

135 Alexander Shen and Nikolay K. Vereshchagin. *Computable functions*, volume 19 of *Student Mathematical Library.* AMS, 2003. `doi:10.1090/stml/019`.

136 Michael Sipser. *Introduction to the theory of computation.* Cengage Learning, 2013. 3rd ed.

137 Ryan Stansifer. Presburger's article on integer arithmetic: Remarks and translation. Technical Report TR 84-639, Department of Computer Science, Cornell University, Ithaca, New York, September 1984. URL: `https://hdl.handle.net/1813/6478`.

138 Mikhail R. Starchak. Positive existential definability with unit, addition and coprimeness. In *ISSAC*, pages 353–360, 2021. `doi:10.1145/3452143.3465515`.

139 Mikhail R. Starchak. On the existential arithmetics with addition and bitwise minimum. In *FoSSaCS*, pages 176–195, 2023. `doi:10.1007/978-3-031-30829-1_9`.

140 Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational Horn clauses. In *CADE*, pages 337–352, 2005. `doi:10.1007/11532231_25`.

141 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`.

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

143 Volker Weispfenning. Mixed real-integer linear quantifier elimination. In *ISSAC*, pages 129–136, 1999. `doi:10.1145/309831.309888`.

144 Herbert S. Wilf. A circle-of-lights algorithm for the "money-changing problem". *Am. Math. Mon.*, 85(7):562–565, 1978. `doi:10.2307/2320864`.

145 H. Paul Williams. *Logic and integer programming*, volume 130 of *International Series in Operations Research & Management Science.* Springer, 2009. `doi:10.1007/978-0-387-92280-5`.

146 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.*, 61(1):23–52, 2024. `doi:10.1007/S00236-023-00446-4`.

147 Dennis Yurichev. SAT/SMT by example. Version of May 12, 2024. URL: `https://smt.st/`.