Subclasses of Presburger Arithmetic and the Weak EXP Hierarchy

Christoph Haase [*]

Laboratoire Spécification et Vérification (LSV), CNRS
École Normale Supérieure (ENS) de Cachan, France
haase@lsv.ens-cachan.fr

Abstract

It is shown that for any fixed $i > 0$, the Σ_{i+1}-fragment of Presburger arithmetic, *i.e.*, its restriction to $i + 1$ quantifier alternations beginning with an existential quantifier, is complete for Σ_i^{EXP}, the i-th level of the weak EXP hierarchy, an analogue to the polynomial-time hierarchy residing between NEXP and EXPSPACE. This result completes the computational complexity landscape for Presburger arithmetic, a line of research which dates back to the seminal work by Fischer & Rabin in 1974. Moreover, we apply some of the techniques developed in the proof of the lower bound in order to establish bounds on sets of naturals definable in the Σ_1-fragment of Presburger arithmetic: given a Σ_1-formula $\Phi(x)$, it is shown that the set of non-negative solutions is an ultimately periodic set whose period is at most doubly-exponential and that this bound is tight.

Categories and Subject Descriptors F.4.1 [*Mathematical logic*]: Computational logic

Keywords Presburger arithmetic, bounded quantifier alternation, weak EXP hierarchy, ultimately periodic sets, context-free commutative grammars

1. Introduction

Presburger arithmetic is the first-order theory of the structure $\langle \mathbb{N}, 0, 1, +, < \rangle$. This theory was shown to be decidable by Presburger in his seminal paper in 1929 by providing a quantifier-elimination procedure [31]. Presburger arithmetic is central to a vast number of different areas in computer science and is often employed as a tool for showing decidability and complexity results.

The central decision problem for Presburger arithmetic is *validity*, *i.e.*, to decide whether a given sentence is true with respect to the standard interpretation in arithmetic. The two most prominent ways to decide validity are either quantifier-elimination based [8] or automata based [9, 23, 40]. Any decision procedure for Presburger arithmetic is inherently tied to the computational complexity of Presburger arithmetic; for that reason the complexity of Presburger arithmetic has extensively been studied in the literature from the 1970's onwards. In order to fully capture the computational complexity of Presburger arithmetic, Berman even introduced the STA measure on the complexity of a decision problem, since Presburger arithmetic *"may not have precise complexity characterisations in terms of the usual time and tape measures"* [3]. The class $\mathsf{STA}(s(n), t(n), a(n))$ is the class of all problems of length n that can be decided by an alternating Turing machine in space $s(n)$ and time $t(n)$ using $a(n)$ alternations, where "$*$" acts as a wildcard in order to indicate an unbounded availability of a certain resource. Based on the work by Fischer & Rabin [11] and Ferrante & Rackoff [10], Berman established the following result.

Proposition 1 (Berman [3])**.** Presburger arithmetic is complete for $\mathsf{STA}(*, 2^{2^{n^{O(1)}}}, n)$.

In terms of the usual time and space measures, this settles Presburger arithmetic between 2-NEXP and 2-EXPSPACE. Despite these high computational costs, on the positive side when looking at fragments Presburger arithmetic becomes more manageable. There are two dimensions in which we can constraint formulas in order to obtain fragments of Presburger arithmetic: the number of quantifier alternations and the number of variables in each quantifier block. For $i, j \in \mathbb{N} \cup \{*\}$, let PA($i$,$j$) denote the set of formulas of the Σ_i-fragment of Presburger arithmetic[1] such that at most j different variables occur in each quantifier block, where "$*$" is used as a wildcard for an unbounded number. Hence, Proposition 1 characterises the computational complexity of PA($*$,$*$) with n being the number of symbols required to write down the formula. Subsequently, PA and PA(i) abbreviate PA($*$,$*$) and PA(i,$*$), respectively.

One of the most prominent fragments of Presburger arithmetic is its existential or quantifier-free fragment, which is computationally not more expensive than standard Boolean satisfiability.

Proposition 2 (Scarpellini [33], Borosh & Treybing [5])**.** For any fixed $j \in \mathbb{N}$, PA(1,j) is in P [33]. PA(1) is NP-complete [5].

Due to its comparably low computational complexity, quantifier-free Presburger arithmetic is the fragment that is most commonly found in application areas which aim at a practical impact. The existential fragment of Presburger arithmetic can even be extended with a full divisibility predicate while retaining decidability [25, 26].

Another subclass of Presburger arithmetic which has extensively been studied is obtained by allowing for an arbitrary but fixed number of quantifier alternations.

[*] The author is supported by the French Agence Nationale de la Recherche (ANR), REACHARD (grant ANR-11-BS02-001).

[1] All results obtained are symmetric when considering Π_i-formulas.

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.
Copyright © 2014 ACM 978-1-4503-2886-9…$15.00.
http://dx.doi.org/10.1145/2603088.2603092

Proposition 3 (Grädel [17], Schöning [34], Reddy & Loveland [32])**.** For any fixed $i > 0$ and $j > 2$, PA($i+1$,j) is $\mathsf{\Sigma}_i^{\mathsf{P}}$-complete[2] [17, 34]. PA(i) is in $\mathsf{STA}(*, 2^{n^{O(i)}}, i)$ [32].

Thus, when fixing the number of quantifier alternations, the complexity of Presburger arithmetic decreases roughly by one exponent, and when additionally fixing the number of variables, we obtain every level of the polynomial-time hierarchy. Notice that there is an obvious gap: a completeness result for Presburger arithmetic with a fixed number of quantifier alternations and an arbitrary number of variables in each quantifier block is missing.

The study of lower bounds for PA(i) goes back to the work of Fürer [13], who showed a NEXP lower bound for some fixed $i > 1$. Later, Grädel showed NEXP-hardness and $\mathsf{EXPSPACE}$ membership of PA(2), but tight lower and upper bounds for the whole class of PA(i) formulas have not yet been established. The purpose of the first part of this paper is to close this gap and establish the following theorem.

Theorem 1. For any fixed $i > 0$, the Σ_{i+1}-fragment of Presburger arithmetic is $\mathsf{\Sigma}_i^{\mathsf{EXP}}$-complete.

Here, $\mathsf{\Sigma}_i^{\mathsf{EXP}}$ denotes the i-th level of the *weak EXP hierarchy* [19], an analogue to the polynomial-time hierarchy [36] residing between NEXP and $\mathsf{EXPSPACE}$; a formal definition will be provided in Section 2.4. Equivalently, we obtain that PA($i+1$) is complete for $\mathsf{STA}(*, 2^{n^{O(i)}}, i)$. Determining the precise complexity of the Σ_i-fragment of Presburger arithmetic for a fixed i has been listed as a problem that *"deserves to be investigated"* by Compton & Henson [7, Prob. 10.14]. However, as pointed out in [7], their generic methods for proving lower bounds do not seem to be applicable to this fragment, and our hardness result is based on rather specific properties of, for instance, distributions of prime numbers.

The second part of the paper diverts from the first part and focuses on the Σ_1-fragment of Presburger arithmetic. More specifically, we consider sets of naturals definable by formulas in the Σ_1-fragment of Presburger arithmetic open in one variable. Given a Σ_1-formula $\Phi(x)$, denote by $[\![\Phi(x)]\!]$ the set of those $a \in \mathbb{N}$ such that replacing x with a in $\Phi(x)$ is valid. It is well-known that $[\![\Phi(x)]\!]$ is an ultimately periodic set, see *e.g.* [4]. A set $N \subseteq \mathbb{N}$ is *ultimately periodic* if there exists a *threshold* $t \in \mathbb{N}$, a *base* $B \subseteq \{0, \ldots t-1\}$, a *period* $p \in \mathbb{N}$, and a set of *residue classes* $R \subseteq \{0, \ldots p-1\}$ such that $N = U(t, p, B, R)$ with

$$U(t, p, B, R) \overset{\text{def}}{=} B \cup \{t + r + kp : r \in R, k \geq 0\}\,.$$

Given a Σ_1-formula $\Phi(x)$, by applying some insights from the first part, we can establish a doubly-exponential upper bound on the period of the ultimately periodic set equivalent to $[\![\Phi(x)]\!]$ and show that this bound is tight, which is captured by the second main theorem of this paper.

Theorem 2. There exists a family of Σ_1-formulas of Presburger arithmetic $(\Phi_n(x))_{n>0}$ such that each $\Phi_n(x)$ is a PA(1,$O(n)$) formula with $|\Phi_n(x)| \in O(n^2)$ and $[\![\Phi_n(x)]\!]$ is an ultimately periodic set with period $p_n \in 2^{2^{\Omega(n)}}$. Moreover for any Σ_1-formula $\Phi(x)$, we have $[\![\Phi(x)]\!] = U(t, p, B, R)$ such that $t \in 2^{\mathsf{poly}(|\Phi(x)|)}$ and $p \in 2^{2^{\mathsf{poly}(|\Phi(x)|)}}$.

The most interesting part about this theorem is the doubly-exponential lower bound of the period of ultimately periodic sets definable by PA(1) formulas. Establishing bounds on constants of ultimately periodic sets naturally occurs when analysing the computational complexity of decision problems for infinite-state systems [15] or in formal language theory [20]. For instance, analysing such bounds has been crucial in order to obtain optimal complexity results for model-checking problems of a class of one-counter automata in [15]. In more detail, in [15] it has been shown that the set of non-negative weights of paths between two nodes in a weighted graph is ultimately periodic with a period that is at most *singly-exponential* bounded. A result by Seidl *et al.* [35] on Parikh images of non-deterministic finite-state automata implicitly states that those ultimately periodic sets are definable in the Σ_1-fragment of Presburger arithmetic. It would thus be desirable to establish a generic upper bound for ultimately periodic sets definable in PA(1) yielding the same optimal bounds. In this context, Theorem 2 provides a negative result in that it shows that a general bound on ultimately periodic sets definable in PA(1) cannot yield the optimal bounds required for natural concrete ultimately periodic sets like those considered in [15].

This paper is structured as follows. In Section 2 we provide most of the formal definitions required in this paper; however the reader is expected to have some level of familiarity with standard notions and concepts from linear algebra, integer programming, first-order logic and computational complexity. Even though we provide a slightly more elaborated account on succinct encodings via Boolean circuits, it will be beneficial to the reader to be familiar with Chapters 8 and 20 in Papadimitriou's book on computational complexity [28]. Section 3 is then going to establish the lower and upper bounds of Theorem 1, and Theorem 2 is shown in Section 4. The paper concludes in Section 5. Subsequent to the bibliography, a proof of a technical characterisation of the weak EXP hierarchy is outlined in the appendix for the sake of completeness.

2. Preliminaries

2.1 General notation

By $\mathbb{Z}$ and $\mathbb{N}$ we denote the set of integers and natural numbers, respectively. We will usually use a, b, c for numbers in $\mathbb{Z}$ and $\mathbb{N}$. Given $a \in \mathbb{N}$, we define $[a] \overset{\text{def}}{=} \{0, \ldots a-1\}$. Given sets $M, N \subseteq \mathbb{N}$, as is standard $M + N \overset{\text{def}}{=} \{m + n : m \in M, n \in N\}$ and $M \cdot N \overset{\text{def}}{=} \{mn : m \in M, n \in N\}$. Moreover, we will use standard notation for integer intervals and, *e.g.*, for $a \leq b \in \mathbb{N}$ denote by $[a, b)$ the set $\{a, \ldots b-1\}$. For vectors $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$, we will denote by $\|\mathbf{a}\|$ the *norm of* $\mathbf{a}$, which is the maximum absolute value of all components of $\mathbf{a}$, *i.e.*, $\|\mathbf{a}\| \overset{\text{def}}{=} \max\{|a_i|\}_{1 \leq i \leq n}$. For $m \times n$ integer matrices A, $\|A\|$ denotes the maximum absolute value of all components of A. Finally, given a set $M \subseteq \mathbb{Z}^n$, we denote by $\|M\|$ the maximum of the norm of all elements of M. All functions in this paper are assumed to map non-negative integers to non-negative integers. Unless stated otherwise, we assume all integers in this paper to be encoded in binary, *i.e.*, the size or length to write down $a \in \mathbb{Z}$ is $O(\log|a|)$.

2.2 Presburger Arithmetic

Usually, x, y, z will denote first-order variables, and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ vectors or tuples of first-order variables. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be an n-tuple of first-order variables. In this paper, formulas of Presburger arithmetic are standard first-order formulas over the structure $\langle \mathbb{N}, 0, 1, +, < \rangle$ obtained from atomic expressions of the form $p(\mathbf{x}) < b$, where $p(\mathbf{x})$ is a linear multivariate polynomial with integer coefficients and absolute term zero, and $b \in \mathbb{Z}$. If the dimension of $\mathbf{x}$ is clear from the context, for brevity we will often omit stating it explicitly. Let $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{N}^n$ and $\Phi(\mathbf{x})$ be open in the first-order variables $\mathbf{x}$, we denote by $\Phi(\mathbf{a}/\mathbf{x})$ the closed formula obtained from replacing every occurrence of x_i in $\Phi(\mathbf{x})$ with a_i. By $[\![\Phi(\mathbf{x})]\!]$ we denote the set $\{\mathbf{a} \in \mathbb{N}^n : \Phi(\mathbf{a}/\mathbf{x}) \text{ is valid}\}$. The *size* $|\Phi|$ of a formula of Presburger arithmetic is defined as the number of

[2] In order to establish hardness, $j > 2$ is only required on the innermost quantifier.

symbols required to write it down, and the *norm* $\|\Phi\|$ is the largest absolute value of all constants occurring in Φ.

Remark. For notational convenience, when stating concrete formulas we will permit ourselves to use atomic formulas $p(\mathbf{x}) < q(\mathbf{x})$ for linear polynomials $p(\mathbf{x}), q(\mathbf{x})$. Moreover, all results on the complexity of validity of formulas of Presburger arithmetic carry over if we assume unary encoding of numbers, since binary encoding of numbers can be "simulated" by the introduction of additional first-order variables and repeated multiplication by two, causing only a sub-quadratic blowup in the formula size. In addition, an equality predicate "=" can be expressed in terms of $<$ causing a linear blowup, since $x = y \leftrightarrow x < y + 1 \wedge y < x + 1$. Likewise, $x > y$ and $x < y < z$ abbreviate $y < x$ and $x < y \wedge y < z$, respectively.

2.3 Semi-Linear Sets and Systems of Linear Diophantine Inequalities

A central result due to Ginsburg and Spanier states that the sets of natural numbers definable by a formula of Presburger arithmetic open in n variables are the *n-dimensional semi-linear sets* [14], which we just call semi-linear sets if the dimension is clear from the context. A semi-linear set is a finite union of *linear sets*. The latter are defined in terms of a *base vector* $\mathbf{b} \in \mathbb{N}^n$ and a finite set of *period vectors* $P = \{\mathbf{p}_1, \dots \mathbf{p}_k\} \subseteq \mathbb{N}^n$, and define the set

$$L(\mathbf{b}; P) \overset{\text{def}}{=} \mathbf{b} + \lambda_1 \mathbf{p}_1 + \cdots \lambda_k \mathbf{p}_k,\ \lambda_i \in \mathbb{N}, 1 \leq i \leq k.$$

Let A be an $m \times n$ integer matrix and $\mathbf{c} \in \mathbb{Z}^m$. A *system of linear Diophantine inequalities* is given as $S : A\mathbf{x} \geq \mathbf{c}$. The *size* $|S|$ *of* S is the number of symbols required to write down S assuming binary encoding of numbers. The *set of positive solutions of* S is denoted by $[\![S]\!] \subseteq \mathbb{N}^n$ and is the set of all n-tuples such that the inequalities in every row of S hold.

The following proposition is due to Frank & Tardos and establishes a strongly polynomial-time algorithm for the feasibility problem of a system of linear Diophantine inequalities in a fixed dimension, *i.e.*, deciding whether $[\![S]\!] \neq \emptyset$.

Proposition 4 (Frank & Tardos [12])**.** Let $S : A\mathbf{x} \geq \mathbf{c}$ be a system of linear Diophantine inequalities such that A is an $m \times n$ matrix. Then feasibility of S can be decided using $n^{2.5n+o(n)}|S|$ arithmetic operations and space polynomial in $|S|$.

When we are interested in representing the set of all solutions of S, we will employ the following proposition, which provides bounds on the semi-linear representation of $[\![S]\!]$ and is a consequence of Corollary 1 in [30].

Proposition 5 (Pottier [30])**.** Let $S : A\mathbf{x} \geq \mathbf{c}$ be a system of linear Diophantine inequalities such that A is an $m \times n$ matrix. Then $[\![S]\!] = \bigcup_{i \in I} L(\mathbf{b}_i; P_i)$ such that for all $i \in I$,

$$\|\mathbf{b}_i\|, \|P_i\| \leq (n\|A\| + \|\mathbf{c}\| + 2)^{m+n}.$$

2.4 Time Hierarchies

Let us recall the definitions of the *polynomial-time hierarchy* PH [36] and the *weak EXP hierarchy* EXPH [19] in terms of oracle complexity classes. As usual,

$$\mathsf{P} = \bigcup_{k>0} \mathsf{DTIME}(n^k) \qquad \mathsf{EXP} = \bigcup_{k>0} \mathsf{DTIME}(2^{n^k})$$
$$\mathsf{NP} = \bigcup_{k>0} \mathsf{NTIME}(n^k) \qquad \mathsf{NEXP} = \bigcup_{k>0} \mathsf{NTIME}(2^{n^k}).$$

The aforementioned time hierarchies are now defined as

$$\begin{aligned} \Sigma_0^{\mathsf{P}} \overset{\text{def}}{=} \Pi_0^{\mathsf{P}} &\overset{\text{def}}{=} \mathsf{P} & \Sigma_0^{\mathsf{EXP}} \overset{\text{def}}{=} \Pi_0^{\mathsf{EXP}} &\overset{\text{def}}{=} \mathsf{EXP} \\ \Sigma_{i+1}^{\mathsf{P}} &\overset{\text{def}}{=} \mathsf{NP}^{\Sigma_i^{\mathsf{P}}} & \Sigma_{i+1}^{\mathsf{EXP}} &\overset{\text{def}}{=} \mathsf{NEXP}^{\Sigma_i^{\mathsf{P}}} \\ \Pi_{i+1}^{\mathsf{P}} &\overset{\text{def}}{=} \mathsf{coNP}^{\Sigma_i^{\mathsf{P}}} & \Pi_{i+1}^{\mathsf{EXP}} &\overset{\text{def}}{=} \mathsf{coNEXP}^{\Sigma_i^{\mathsf{P}}} \\ \mathsf{PH} &\overset{\text{def}}{=} \bigcup_{i \geq 0} \Sigma_i^{\mathsf{P}} & \mathsf{EXPH} &\overset{\text{def}}{=} \bigcup_{i \geq 0} \Sigma_i^{\mathsf{EXP}}. \end{aligned}$$

For our lower bounds, we will rely on the following equivalent characterisation of Σ_i^{EXP}.

Lemma 1. For any $i > 0$, a language $L \subseteq \{0,1\}^*$ is in Σ_i^{EXP} iff there exists a polynomial q and a predicate $R \subseteq (\{0,1\}^*)^{i+1}$ such that for any $w \in \{0,1\}^n$,

$$w \in L \text{ iff } \exists w_1 \in \{0,1\}^{2^{q(n)}}.\forall w_2 \in \{0,1\}^{2^{q(n)}} \cdots$$
$$\cdots Q_i w_i \in \{0,1\}^{2^{q(n)}}.R(w, w_1, \dots, w_i)$$

and $R(w, w_1, \dots, w_i)$ can be decided in deterministic polynomial time.

Despite being in the spirit of an elementary result on computational complexity, the author was unable to find a formal proof of Lemma 1 in the standard literature. It is somewhat stated informally without a proof in [19]. In order to keep this paper self-contained and for the reader's convenience, a proof sketch of Lemma 1 based on a proof of an analogue characterisation of the polynomial-time hierarchy given in [2] is provided in the appendix.

2.5 Boolean Circuits

A standard approach to raise the complexity of a problem known to be complete for a complexity class by one exponent is to succinctly represent the input, see *e.g.* [16, 29]. A well-known concept is to represent the input by Boolean circuits. In this paper, for technical convenience we adapt the definition provided in [16].

Definition 1. A *Boolean circuit* $\mathcal{C}$ *of size* r *with* $n \leq r$ *inputs* is a function $f : [r] \to \{\&, \|, \sim, \uparrow, \downarrow_1\} \times [r] \times [r]$, where $f(i) = (t, j, k)$ iff the gate with *index* i is of *type* t, *i.e.*, an *and*, *or*, *not*, *input* or *constant gate*, respectively, and $j, k < i$ are inputs of the gate, unless $t = \sim$ in which case we require $j = k$.

We identify each gate of $\mathcal{C}$ with an index from $[r]$, and by convention the first $n \leq r$ gates are *input*-gates, and the r-th gate, *i.e.*, the gate with index $r - 1$, is treated as the *output gate* of $\mathcal{C}$. Moreover for technical convenience, we sometimes identify the various types of the gates by natural numbers ordered as in Definition 1, *i.e.*, & is identified as 0, $\|$ as 1, *etc.* By using constant gates as gates with constant value 1, an input $w \in \{0,1\}^n$ to $\mathcal{C}$ induces a unique evaluation mapping $e_w : [r] \to \{0,1\}$ defined in the obvious way, and $\mathcal{C}$ evaluates to true (false) on input w if $e_w(r-1) = 1$ ($e_w(r-1) = 0$). For brevity, we define $\mathcal{C}(w) \overset{\text{def}}{=} e_w(r-1)$, and if $m_1, \dots, m_k \in \mathbb{N}$ then $\mathcal{C}(m_1, \dots m_k)$ is the output of $\mathcal{C}(w_1 \cdots w_k)$, where each $w_i \in \{0,1\}^{\lceil \log m_i \rceil}$ is the binary, if necessary padded, representation of m_i.

For the remainder of this section, we will briefly recall and elaborate on some results and concepts about circuits and succinct encodings from Papadimitriou's book [28] on computational complexity. Given a circuit $\mathcal{C}$ and an input $w \in \{0,1\}^n$ for some $n \geq 0$, it is well-known that determining $\mathcal{C}(w)$ is P-complete [28, Thm. 8.1]. In [28], the proof of P-hardness is established by showing that the computation table of a polynomial-time Turing machine can be encoded as a Boolean circuit. For an $f(n)$-time-bounded Turing machine M, a computation table is an $f(n) \times f(n)$ grid of cells $T_{i,j}$ from an alphabet that allows for uniquely encoding configurations of

⋮	⋮	⋮	⋮	⋮	⋮
$\triangleright$	1	1	0_{q_0}	□	□
$\triangleright$	1	1_{q_1}	1	□	□
$\triangleright$	1_{q_2}	0	1	□	□
$\triangleright_{q_0}$	1	0	1	□	□

Figure 1. Graphical illustration of a computation table of a time-bounded Turing machine M. For example, here we have $T_{2,2} = 0_{q_2}$. The control state and the head position of M is indicated by tape symbols with some q_i as subscript. The four gray-shaded cells illustrate that successive cells only depend on three preceding cells.

M such that the configuration of M in step i while running on w is encoded in the i-th row. Figure 1 graphically illustrates the concept of a computation table, where 0 and 1 are alphabet symbols of M, and $\triangleright$ and $\Box$ are left delimiters and blank symbols, respectively. The crucial fact for encoding computation tables as Boolean circuits is that for $i, j > 1$, the symbol at $T_{i,j}$ only depends on a fixed number of cells, namely $T_{i-1,j-1}$, $T_{i-1,j}$ and $T_{i-1,j+1}$, illustrated by the gray-shaded cells in Figure 1. It is then clear that the alphabet of a computation table can be encoded into a binary alphabet of truth values, and that a constant basic circuit can be constructed from M which ensures that the values of the cells are correctly propagated along the y-axis. It then follows that M accepts w iff there exists a computation table ending in an accepting state iff the circuit encoding this computation table evaluates to true.

In the next section, for our lower bound we will apply Lemma 1, which entails deciding $(w, w_1, \ldots, w_i) \in R$, where the w_j are of size exponential in $n = |w|$. Let M_w be a polynomial-time Turing machine deciding R for a fixed w. The $w_1, \ldots w_j$ will implicitly be coded into natural numbers, so it will not be possible to construct a Boolean circuit $\mathcal{C}_w$ upfront that can evaluate M_w on the input $w_1, \ldots w_j$ of exponential size, since we are required to establish a polynomial-time reduction. Instead, we will *succinctly encode* $\mathcal{C}_w$ via another Boolean circuit $\mathcal{D}_w$. More precisely, $\mathcal{C}_w$ is encoded via $\mathcal{D}_w$ as follows: $\mathcal{D}_w$ has $3r(n) + 3$ input gates for some fixed polynomial r depending on M_w such that for $i, j, k \in [2^{r(n)}]$ and $t \in [5]$, $\mathcal{D}_w(t, i, j, k) = 1$ iff the defining function f of $\mathcal{C}_w$ gives $f(i) = (t, j, k)$, *i.e.*, that the gate with index i of $\mathcal{C}_w$ is of type t and has input gates with index j and k. In particular, $\mathcal{D}_w$ and henceforth $\mathcal{C}_w$ only depend on w and M, and are *independent* of $w_1, \ldots w_i$. Note that we can view an assignment of truth values to the gates of $\mathcal{C}_w$ as a string of length $2^{r(n)}$.

More generally, it is known that if $\mathcal{C}$ with no input gates is succinctly given by some circuit $\mathcal{D}$, determining whether $\mathcal{C}$ evaluates to true is EXP-complete [28, Thm. 20.2 & Cor. 2]. The idea underlying the hardness proof is a straight-forward generalisation of the approach outlined in the paragraph above. The circuit $\mathcal{C}$ encodes the computation table of an EXP Turing machine M. Since the indices of the gates of $\mathcal{C}$ can be represented in *binary*, via $\mathcal{D}$ we can encode $\mathcal{C}$ by implicitly encoding an exponential number of the constant basic circuit ensuring proper propagation between consecutive cells. This approach can now be adapted for our purpose, *i.e.*, to evaluate a polynomial-time Turing machine on an input of exponential size. The major challenge is to transfer input to the succinctly encoded circuit on-the-fly.

Referring to Lemma 1 and given M_w as above, we can construct in logarithmic space a Boolean circuit $\mathcal{D}_w$ encoding $\mathcal{C}_w$ such that the input $(w_1, \ldots, w_i)$ to $\mathcal{C}_w$ is obtained from the first $i2^{q(n)}$ gates, and $\mathcal{C}_w$ encodes a computation table of M_w on this input. Figure 2 illustrates how this can be realised. Each box in Figure 2 is a gate, and a cell of the computation table of M_w while being executed on $(w_1, \ldots, w_n)$ is encoded into the dashed boxes, or more specifically, into the three framed gray-shaded boxes on the bottom of the dashed boxes. Here, we assume that three bits are sufficient to represent the alphabet of the computation table of M_w, and that $\triangleright_{q_0}$ is encoded as 111. Consequently, the gates with index $(3, 4)$, $(3, 5)$ and $(3, 6)$, representing the cell $T_{1,1}$ of the computation table of M_w, are gates with constant value one, as indicated in Figure 2. Now we want the values of the cells $T_{1,2}$, $T_{1,3}$, *etc.* of the computation table of M_w to be equivalent to $w_1 \cdots w_n$, which are represented by the gates with indices $(0, 0), \ldots (0, i2^{q(n)})$. The gates in $\mathcal{D}_w$ corresponding to $T_{1,2}$ and $T_{1,3}$ have indices $(3, 15), (3, 16), (3, 17)$ and $(3, 26), (3, 27), (3, 28)$, respectively. Those gates have the gates $(0, 0)$ and $(0, 1)$ as their inputs, respectively. Suppose that in our encoding 1 is represented as 101 and 0 as 010, the sequence of $\|$, $\sim$ and $\|$ gates ensures that 1 is mapped to 101 and 0 to 010. Consequently, the gates $(3, 15), (3, 16), (3, 17)$ can correctly transfer the alphabet symbols $\{0, 1\}$ of M_w into the internal representation of the computation table, and in particular copy the first symbol of the input string $w_1 \cdots w_i$ into the internal representation of the computation table. In the example in Figure 2, the gates with index $(3, 15), (3, 16), (3, 17)$ would output 1, 0 and 1, respectively, since the gate $(0, 0)$ has value 1 which corresponds to the first symbol of the input string w_1. As stated before, in our reduction this value is provided on-the-fly. The rest of the reduction follows standard arguments. Each dashed box contains circuits $\mathcal{T}_1, \mathcal{T}_2$ and $\mathcal{T}_3$ which compute the consecutive cell of the simulated computation table of M_w, *i.e.*, the values of the three gates representing this cell. The dashed boxes on the left use different circuits $\mathcal{U}_1, \mathcal{U}_2$ and $\mathcal{U}_3$ since they do not have a left neighbor. All unused gates can assumed to be dummy gates, *i.e.* gates with constant value 1, as indicated in Figure 2. It follows that M_w accepts $(w_1, \ldots, w_i)$ iff $\mathcal{C}_w$ evaluates to true on the input provided, *i.e.*, the value of the gate with the highest index of $\mathcal{C}_w$ is equal to 1.

In order to encode $\mathcal{C}_w$ succinctly, it is clear that due to the regular structure of $\mathcal{C}_w$, the type and input gates to any gate can be computed from a given index of a gate by a polynomial-time algorithm. The circuit $\mathcal{D}_w$ can now be taken as the circuit corresponding to this algorithm.

3. Completeness of the Σ_{i+1}-Fragment of Presburger Arithmetic for Σ_i^{EXP}

In this section, we show that PA($i + 1$) is Σ_i^{EXP}-complete for every fixed $i > 0$. We begin with the lower bound and first note that it is not possible to adapt Berman's hardness proof [3] in order to get the desired result, since it relies on a trick by Fischer & Rabin [11] in order to perform arithmetic operations on a bounded interval over large numbers which linearly increases the number of quantifier alternations. Instead, we will partly adapt concepts and ideas introduced by Grädel in his hardness proof for PA(2) in [18] and Gottlob, Leone & Veith in [16]. Roughly speaking, we aim for "implementing" Lemma 1 via a PA($i + 1$) formula, which will entail encoding bit strings of exponential size into natural numbers and evaluating Boolean circuits in Presburger arithmetic on-the-fly. The upper bound does not follow immediately and requires combining solution intervals established by Weispfenning in [38] with Proposition 4.

3.1 Lower Bounds

The goal of this section is to prove the following proposition.

Proposition 6. Let $L \subseteq \{0, 1\}^*$ be a language in Σ_i^{EXP}, $i > 0$ and $w \in \{0, 1\}^*$. There exists a polynomial-time computable PA($i + 1$) formula $\Phi_{L,w}$ such that $w \in L$ iff $\Phi_{L,w}$ is valid.

To this end, we employ the characterisation of Σ_i^{EXP} in Lemma 1. Let M be the deterministic polynomial-time Turing machine decid-

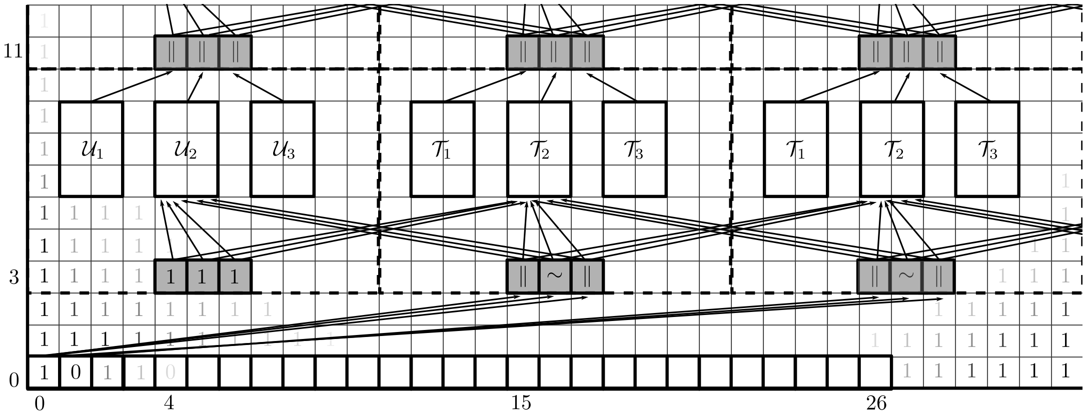

Figure 2. Illustration of the approach of how to succinctly encode a Boolean circuit encoding the computation table of a polynomial-time Turing machine on an input of exponential size. Each square represents a gate, all gates not surrounded by boxes are assumed to be gates with constant value 1.

ing R from Lemma 1, and let M_w be such a Turing machine deciding R for a fixed input $w \in \{0,1\}^n$, which can be computed from M in logarithmic space. The bit strings w_1 to w_i from Lemma 1 constituting the input to M_w are represented in our reduction via natural numbers assigned to first-order variables $\mathbf{x} = (x_1, \ldots, x_i)$. The precise encoding of a w_j via x_j is discussed below. For now, it is only important to mention that not every natural number encodes a bit string. Let us focus on the high-level structure of $\Phi_{L,w}$:

$$\Phi_{L,w} \overset{\text{def}}{=} \exists x_1.\forall x_2 \cdots Q_i x_i. \bigwedge_{1\le j\le i,\ j \text{ odd}} \Psi_{valid,r(n)}(x_j) \wedge$$
$$\wedge \Big(\bigwedge_{1\le j\le i,\ j \text{ even}} \Psi_{valid,r(n)}(x_j) \Big) \to \Psi_{M_w}(x_1, \ldots, x_i). \quad (1)$$

Unsurprisingly, the alternation of quantifiers in Lemma 1 is reflected by the alternation of quantifiers in (1), so $Q_i = \exists$ if i is odd and $Q_i = \forall$ if i is even. The formula $\Psi_{valid,r(n)}(x_i)$ is a Π_1-formula, and $\Psi_{M_w}(x_1, \ldots, x_i)$ is a formula in the Boolean closure of Σ_1 if i is odd and a Σ_1-formula if i is even. The first conjunct ensures that the existentially quantified variables represent encodings of bit strings and the second conjunct that, under the additional assumption that the universally quantified variables encode valid bit strings as well, M_w accepts the bit strings encoded in $x_1, \ldots, x_i$. For the given $w \in \{0,1\}^n$, those formulas are concrete instances of a family of formulas, and $r(n)$ is an index in this family for some polynomial $r(n)$ which dominates $q(n)$ in Lemma 1 and is made more precise at a later stage. Consequently, for a fixed $i > 0$, we have that $\Phi_{L,w}$ is a PA($i+1$) formula.

In our reduction, we have to take extra care to prevent the "accidental" introduction of quantifier alternations. In general when providing formulas, we adapt Grädel's approach in [18] and provide *neutral formulas*, which are open polynomially equivalent Σ_1- and Π_1-formulas. This ensures that, for instance, we do not have to care about whether we could possibly introduce a new quantifier alternation if a formula is used on the left-hand side of an implication. When providing a neutral Σ_1-formula $\Phi(\mathbf{x}) = \exists \mathbf{y}.\varphi(\mathbf{x}, \mathbf{y})$, we will denote by $\bar{\Phi}(\mathbf{x}) = \forall \mathbf{y}.\bar{\varphi}(\mathbf{x}, \mathbf{y})$ its neutral equivalent Π_1 counterpart. For the sake of consistent naming, whenever $\Phi(\mathbf{x})$ occurs as a subformula in some other formula, we *implicitly* assume that it is appropriately replaced such that the resulting formula is either a Σ_1- or a Π_1-formula, depending on the context. Likewise, if for instance $\Phi(\mathbf{x})$ occurs as a negated subformula in a formula that is supposed to be existentially quantified, we assume that this subformula is *implicitly* replaced by $\exists \mathbf{x}.\neg(\bar{\varphi}(\mathbf{x}, \mathbf{y}))$, and $\neg(\bar{\varphi}(\mathbf{x}, \mathbf{y}))$ is treated in the same way if it is not yet quantifier-free. In this way, we can always make sure to result in Σ_1- or Π_1-formulas.

We now turn towards the details of our reduction and begin with discussing the encoding of bit strings as natural numbers we use subsequently. The encoding we use is due to Grädel [18]. In his NEXP lower bound for PA(2) he exploits a result due to Ingham [6, 22] that for any sufficiently large[3] $i \in \mathbb{N}$ there is at least one prime in the interval $[i^3, (i+1)^3)$. Given a bit string $w = b_1 \cdots b_n \in \{0,1\}^n$, a natural number $a \in \mathbb{N}$ encodes w if for all $1 \le i \le n$ and

$$\text{for all primes } p \in [i^3, (i+1)^3) : a \equiv b_i \bmod p.$$

The existence of such an a is then guaranteed by the Chinese remainder theorem. Given a fixed $n > 0$, we call $a \in \mathbb{N}$ a *valid encoding* if for every $1 \le i \le n$, either $a \equiv 0 \bmod p$ or $a \equiv 1 \bmod p$ for all prime numbers $p \in [i^3, (i+1)^3)$.

In order to enable the extraction of bits of bit strings encoded as naturals, we show how to check for divisibility with a natural number whose number of bits is fixed. Next, we show how to evaluate a Boolean circuit in Presburger arithmetic. This serves two purposes: first, it allows for deciding if a given number lies in an interval $[i^3, (i+3)^3)$ and for testing whether a given number is a prime due to the AKS primality test [1]. Second, it allows for simulating M_w discussed above on an input of exponential size using its succinct encoding via a circuit as discussed in Section 2.5. Putting everything together eventually yields the desired reduction.

We begin with a family of quantifier-free formulas $\Phi_{bin,n}(\mathbf{x}, x)$ such that given $\mathbf{b} \in \{0,1\}^n$ and $b \in \mathbb{N}$, $\Phi_{bin,n}(\mathbf{b}, b)$ holds if $\mathbf{b}$ is the binary representation of b. Consequently, this formula implicitly constraints b such that $b \in [2^n]$:

$$\Phi_{bin,n}(\mathbf{x}, x) \overset{\text{def}}{=} \bigwedge_{i \in [n]} (x_i = 0 \vee x_i = 1) \wedge x = \sum_{i \in [n]} 2^i x_i. \quad (2)$$

Next, we provide a family of neutral formulas $\Phi_{mod,n}(x, y)$ such that for a, b with $b \in [2^n]$, $\Phi_{mod,n}(a, b)$ holds iff $a \equiv 0 \bmod b$. Essentially, $\Phi_{mod,n}(x, y)$ realises a formula for bounded multiplication. In contrast to a formula with the same purpose given in [18], it

[3] Cheng [6] provides explicit bounds on Ingham's result [22] and shows that this statement holds for all $i \in \mathbb{N}$ such that $i > 2^{2^{15}}$. As in [18], for brevity we will use Ingham's result as if it were true for all $i > 0$. It will be clear that we could add Cheng's offset to all numbers involved, causing a constant blowup only.

is not recursively defined and of size $O(n)$ as opposed to $O(n \log n)$ when binary encoding of numbers is assumed. The latter fact will be useful in Section 4. The underlying idea of the subsequent definitions is that if the binary expansion of b is $b = \sum_{i \in [n]} 2^i b_i$ and $bk = a$ for some $k \geq 0$ then a can be written as $a = \sum_{i \in [n]} 2^i a_i$ with $a_i = k b_i$:

$$\Phi_{dig,n}(\mathbf{x}, \mathbf{y}, k) \overset{\text{def}}{=} \bigwedge_{i \in [n]} (y_i = 0 \to x_i = 0 \land y_i = 1 \to x_i = k)$$

$$\begin{aligned} \Phi_{mod,n}(x, y) \overset{\text{def}}{=} {} & \exists \mathbf{x}.\exists \mathbf{y}.\exists k.\Phi_{bin,n}(\mathbf{y}, y) \land \\ & \land \Phi_{dig,n}(\mathbf{x}, \mathbf{y}, k) \land x = \sum_{i \in [n]} 2^i x_i \end{aligned} \tag{3}$$

$$\begin{aligned} \bar{\Phi}_{mod,n}(x, y) \overset{\text{def}}{=} {} & \forall \mathbf{x}.\forall \mathbf{y}.\forall k.\big(\Phi_{bin,n}(\mathbf{y}, y) \land \\ & \land \Phi_{dig,n}(\mathbf{x}, \mathbf{y}, k)\big) \to x = \sum_{i \in [n]} 2^i x_i. \end{aligned}$$

We now turn towards evaluating Boolean circuits with suitable formulas in Presburger arithmetic. The subsequent formulas for evaluating a circuit $\mathcal{C}$ with n input and r gates in total are essentially an adaption of a construction given by Gottlob, Leone & Veith in [16]. It is easily checked that for $a \in [2^n]$, $\Phi_{\mathcal{C}}(a)$ holds iff $\mathcal{C}(a) = 1$. In $\Phi_{\mathcal{C}}$, the input to $\mathcal{C}$ is encoded via a dimension n vector of first-order variables $\mathbf{x}$ and the Boolean assignment to the gates via a dimension r vector of first-order variables $\mathbf{y}$, which are implicitly assumed to range over $\{0, 1\}$. First, we provide a formula ensuring that the structure of the gates of $\mathcal{C}$ is correctly encoded in $\mathbf{y}$:

$$\begin{aligned} & \Phi_{\mathcal{C},gates}(\mathbf{x}, \mathbf{y}) \overset{\text{def}}{=} \\ & \bigwedge_{i \in [r]} \begin{cases} y_i = 1 \leftrightarrow (y_j = 1 \land y_k = 1) & \text{if } f(i) = (\&, j, k) \\ y_i = 1 \leftrightarrow (y_j = 1 \lor y_k = 1) & \text{if } f(i) = (\|, j, k) \\ y_i = 1 \leftrightarrow y_j = 0 & \text{if } f(i) = (\sim, j, k) \\ y_i = x_i & \text{if } f(i) = (\uparrow, 0, 0) \\ y_i = 1 & \text{if } f(i) = (\downarrow_1, 0, 0). \end{cases} \end{aligned} \tag{4}$$

Next, the formula $\Phi_{\mathcal{C}}(x)$ defined below now enables us to determine whether $\mathcal{C}$ accepts a given input encoded into the first-order variable x:

$$\begin{aligned} \Phi_{\mathcal{C}}(x) \overset{\text{def}}{=} {} & \exists \mathbf{x}.\exists \mathbf{y}.\exists y.\Phi_{bin,n}(\mathbf{x}, x) \land \Phi_{bin,r}(\mathbf{y}, y) \land \\ & \land \Phi_{\mathcal{C},gates}(\mathbf{x}, \mathbf{y}) \land y_r = 1 \end{aligned} \tag{5}$$

$$\begin{aligned} \bar{\Phi}_{\mathcal{C}}(x) \overset{\text{def}}{=} {} & \forall \mathbf{x}.\forall \mathbf{y}.\forall y.\big(\Phi_{bin,n}(\mathbf{x}, x) \land \Phi_{bin,r}(\mathbf{y}, y) \land \\ & \land \Phi_{\mathcal{C},gates}(\mathbf{x}, \mathbf{y})\big) \to y_r = 1. \end{aligned} \tag{6}$$

We now show how a predicate determining whether a given number $a \in \mathbb{N}$ is a prime number in the interval $[b^3, (b+1)^3)$ for some $b > 0$ representable by n bits can be realised. It is easily verified that any number in this interval can be represented by at most $m = 3(n+1)$ bits. Moreover as discussed above, both conditions can be decided in polynomial time. Therefore we can construct in logarithmic space a Boolean circuit $\mathcal{C}_{prime,n}$ with $m + n$ input gates implementing this predicate [28, Thm. 8.1] and define

$$\begin{aligned} \Phi_{prime,n}(x, y) \overset{\text{def}}{=} {} & \exists \mathbf{x}.\exists \mathbf{y}.\exists z.\Phi_{bin,m}(\mathbf{x}, x) \land \Phi_{bin,n}(\mathbf{y}, y) \land \\ & \land z = \sum_{i \in [m]} 2^i x_i + 2^m \sum_{i \in [n]} 2^i y_i \land \Phi_{\mathcal{C}_{prime,n}}(z) \end{aligned} \tag{7}$$

$$\begin{aligned} \bar{\Phi}_{prime,n}(x, y) \overset{\text{def}}{=} {} & \forall \mathbf{x}.\forall \mathbf{y}.\forall z.\big(\Phi_{bin,m}(\mathbf{x}, x) \land \Phi_{bin,n}(\mathbf{y}, y) \land \\ & \land z = \sum_{i \in [m]} 2^i x_i + 2^m \sum_{i \in [n]} 2^i y_i\big) \to \Phi_{\mathcal{C}_{prime,n}}(z). \end{aligned} \tag{8}$$

The first line of $\Phi_{prime,n}(x, y)$ converts x and y into their binary representation. Next, the second line first concatenates these bit representations via the additional variable z by appropriately shifting the value of y by m bits, and finally z is passed to $\mathcal{C}_{prime,n}$. Consequently, we have that $\Phi_{prime,n}(a, b)$ holds iff a is prime and $a \in [b^3, (b+1)^3)$.

We are now in a position in which we can define a family of Π_1-formulas $\Psi_{valid,n}(x)$ used in (1) that allow for testing whether some $a \in \mathbb{N}$ represents a valid respectively invalid encoding of a bit string of length 2^n. For valid encodings, we wish to make sure that all primes in every relevant interval $[b^3, (b+1)^3)$ have uniform residue classes in a for all $1 \leq b \leq 2^n$, *i.e.*, for any two primes $p_1, p_2 \in [b^3, (b+1)^3)$ we either have $a \equiv 0 \bmod p_1$ and $a \equiv 0 \bmod p_2$, or $a - 1 \equiv 0 \bmod p_1$ and $a - 1 \equiv 0 \bmod p_2$. Let m be as above,

$$\begin{aligned} \Psi_{valid,n}(x) \overset{\text{def}}{=} {} & \forall y.\forall p_1.\forall p_2.\big(1 \leq y \leq 2^n \land \\ & \land \Phi_{prime,n+1}(p_1, y) \land \Phi_{prime,n+1}(p_2, y)\big) \to \\ & \to \big((\Phi_{mod,m+3}(x, p_1) \land \Phi_{mod,m+3}(x, p_2)) \lor \\ & \lor (\Phi_{mod,m+3}(x-1, p_1) \land \Phi_{mod,m+3}(x-1, p_2))\big). \end{aligned}$$

In order to complete our hardness proof for Σ_i^{EXP} for a subsequently fixed $i > 0$ via its characterisation in Lemma 1 and $\Phi_{L,w}$ in (1), we will now define the remaining Π_1-formula $\Psi_{M_w}(x_1, \ldots, x_i)$ for a given $w \in \{0,1\}^n$. Let $\mathcal{C}_w$ be the Boolean circuit succinctly encoded by a Boolean circuit $\mathcal{D}_w(t, y, z_1, z_2)$ deciding M_w on an input of length $2^{q(n)} i$ such that $\mathcal{C}_w$ consists of $2^{r(n)}$ gates for some polynomial $r : \mathbb{N} \to \mathbb{N}$. Recall that we can view an assignment of truth values to the gates of the succinctly encoded circuit $\mathcal{C}_w$ as a bit string, or sequence of bit strings, of appropriate length. In the following let $\mathbf{a} = (a_1, \ldots, a_i) \in \mathbb{N}^i$ be a valuation, for any $1 \leq j < i$ each a_j will be used to encode the values of the input gates with index $2^{q(n)}(j-1)$ up to $2^{q(n)} j - 1$ of $\mathcal{C}_w$, and a_i will encode the values of the gates with index $2^{q(n)}(i-1)$ up to the gate with index $2^{r(n)} - 1$ of $\mathcal{C}_w$. So in particular the internal gates of $\mathcal{C}_w$ are encoded in a_i.

In order to extract encodings of bit strings from natural numbers, as a first step we provide neutral formulas $\Phi_{\mathcal{C}_w,0}(x, y)$ and $\bar{\Phi}_{\mathcal{C}_w,0}(x, y)$ which assume x to be a valid encoding. These formulas enable us to test whether a bit of a bit string whose index is given by y is encoded to be zero in x. Formally, for a valid encoding $a \in \mathbb{N}$ and for $b \in [2^{r(n)}]$, we have $\Phi_{\mathcal{C}_w,0}(a, b)$ iff there is a prime $p \in [(b+1)^3, (b+2)^3)$ and $a \equiv 0 \bmod p$, or $a \equiv 0 \bmod p$ for all primes $p \in [(b+1)^3, (b+2)^3)$, respectively[4]. Let $r'(x) = r(x) + 1$, we define:

$$\Phi_{\mathcal{C}_w,0}(x, y) \overset{\text{def}}{=} \exists p.\Phi_{prime,r'(n)}(p, y+1) \land \Phi_{mod,r'(n)}(x, p)$$

$$\bar{\Phi}_{\mathcal{C}_w,0}(x, y) \overset{\text{def}}{=} \forall p.\Phi_{prime,r'(n)}(p, y+1) \to \Phi_{mod,r'(n)}(x, p).$$

The formulas $\Phi_{\mathcal{C}_w,1}(x, y)$ and $\bar{\Phi}_{\mathcal{C}_w,1}(x, y)$ testing whether the bit with index y is set to 1 in the encoding x can be defined analogously by negating $\Phi_{\mathcal{C}_w,0}(x, y)$. The previously constructed formulas now enable us to define formulas that allow for evaluating the succinctly encoded $\mathcal{C}_w$ on an input that is provided on-the-fly via $\mathbf{a}$. Given an index b implicitly less than $2^{r(n)}$ of a gate of $\mathcal{C}_w$, represented by the first-order variable y, and a vector of valid encodings $\mathbf{a} = (a_1, \ldots, a_i)$ represented by the first-order variables $\mathbf{x}$, the following formula $\Phi_{\mathcal{C}_w,\top}(\mathbf{x}, y)$ checks whether the value of the gate with index b is set to true under the valuation $\mathbf{a}$ according

[4] In order to properly handle the case $b = 0$, we have to shift the interval we use for the encoding by one from $[b^3, (b+1)^3)$ to $[(b+1)^3, (b+2)^3)$.

to the convention described before:

$$\Phi_{\mathcal{C}_w,\top}(\mathbf{x},y) \overset{\text{def}}{=} \bigwedge_{1\leq j<i} \Big(\big(2^{q(n)}(j-1) \leq y < 2^{q(n)}j \rightarrow$$
$$\rightarrow \Phi_{\mathcal{C}_w,1}(x_j,y)\big) \wedge \big(2^{q(n)}(i-1) \leq y \rightarrow \Phi_{\mathcal{C}_w,1}(x_i,y)\big)\Big).$$

A formula $\Phi_{\mathcal{C}_w,\bot}(\mathbf{x},y)$ testing whether the value of a gate is set to false can be defined analogously by negating $\Phi_{\mathcal{C}_w,\top}(\mathbf{x},y)$. Building upon those formulas, we can now construct Boolean connectives that allow for checking that the gates of $\mathcal{C}_w$ are consistently encoded. Given $\mathbf{a} \in \mathbb{N}^i$ as above, $\Phi_{\mathcal{C}_w,\&}(\mathbf{a},b,c_1,c_2)$ holds if the logical and-connective holds for the truth values of the gates with index b, c_1 and c_2 encoded via $\mathbf{a}$:

$$\Phi_{\mathcal{C}_w,\&}(\mathbf{x},y,z_1,z_2) =$$
$$\big(\Phi_{\mathcal{C}_w,\top}(\mathbf{x},y) \leftrightarrow \Phi_{\mathcal{C}_w,\top}(\mathbf{x},z_1) \wedge \Phi_{\mathcal{C}_w,\top}(\mathbf{x},z_2)\big).$$

The remaining Boolean connectives found in Definition 1 can be reflected via the additional formulas

$$\Phi_{\mathcal{C}_w,\|}(\mathbf{x},y,z_1,z_2),\ \Phi_{\mathcal{C}_w,\sim}(\mathbf{x},y,z_1,z_2) \text{ and } \Phi_{\mathcal{C}_w,\downarrow_1}(\mathbf{x},y,z_1,z_2)$$

which are defined analogously to $\Phi_{\mathcal{C}_w,\&}(\mathbf{x},y,z_1,z_2)$. These formulas now enable us to define a Π_1-analogue to $\Phi_{\mathcal{C},gates}(\mathbf{x},\mathbf{y})$ in (4) in order to check if the Boolean assignment of the succinctly encoded circuit $\mathcal{C}_w$ is consistent:

$$\Psi_{\mathcal{C}_w,gates}(\mathbf{x}) = \forall t.\forall y.\forall z_1.\forall z_2.\Phi_{\mathcal{D}_w}(t,y,z_1,z_2) \rightarrow$$
$$\rightarrow \bigwedge \begin{cases} t=0 \rightarrow \Phi_{\mathcal{C}_w,\&}(\mathbf{x},y,z_1,z_2) \\ t=1 \rightarrow \Phi_{\mathcal{C}_w,\|}(\mathbf{x},y,z_1,z_2) \\ t=2 \rightarrow \Phi_{\mathcal{C}_w,\top}(\mathbf{x},y) \leftrightarrow \Phi_{\mathcal{C}_w,\bot}(\mathbf{x},z_1) \\ t=4 \rightarrow \Phi_{\mathcal{C}_w,\top}(\mathbf{x}). \end{cases}$$

Here, $\Phi_{\mathcal{D}_w}(t,y,z_1,z_2)$ is an instantiation of $\Phi_{\mathcal{C}}(x)$ defined in (5) and (6) for the circuit $\mathcal{D}_w$. The use of $\Phi_{\mathcal{D}_w}(t,y,z_1,z_2)$ is not totally clean as $\Phi_{\mathcal{C}}(x)$ it is only open in x. However, this can easily be fixed by concatenating t, y, z_1 and z_2 into a single first-order variable as it was done in (7) and (8), and details have only been omitted for the sake of readability. Also note that the values of t, y, z_1 and z_2 are then implicitly bounded through $\Phi_{\mathcal{D}_w}(t,y,z_1,z_2)$.

Finally, we can define $\Psi_{M_w}(x_1,\ldots,x_i)$, the last remaining formula from (1), as follows

$$\Psi_{M_w}(x_1,\ldots,x_i) \overset{\text{def}}{=}$$
$$\begin{cases} \Psi_{\mathcal{C}_w,gates}(\mathbf{x}) \wedge \Phi_{\mathcal{C}_w,\top}(\mathbf{x},2^{r(n)}-1) & \text{if } i \text{ is odd} \\ \Psi_{\mathcal{C}_w,gates}(\mathbf{x}) \rightarrow \Phi_{\mathcal{C}_w,\top}(\mathbf{x},2^{r(n)}-1) & \text{if } i \text{ is even} \end{cases}$$

Inspecting the construction outlined in this section, it is not difficult to see that for a given $w \in \{0,1\}^n$ the construction of $\Phi_{L_w}(x_1,\ldots,x_i)$ is tedious, but can be performed in polynomial time with respect to n. We leave it as an open problem whether this reduction can actually be performed in logarithmic space, though there do not seem to be any major obstacles. Following the argumentation of this section, we conclude that $\Phi_{L,w}$ is the formula required in Proposition 6.

3.2 Upper Bounds

We will now show that the previously obtained lower bounds have corresponding upper bounds. Let us first recall an improved version of a result by Reddy & Loveland [32] established by Weispfenning [38, Thm. 2.2], which bounds the solution intervals of Presburger formulas.

Proposition 7 (Weispfenning [38])**.** There exists a constant $c > 0$ such that for any PA(i,j) formula Φ and $N = \{0, \ldots 2^{c|\Phi|^{(3j)^i}}\}$, Φ is valid iff Φ is valid when restricting the first-order variables of Φ to be interpreted over elements from N.

Algorithm 1 Deciding $\exists\mathbf{x}_{i+1}.\varphi(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1})$ for a given instantiation $\mathbf{a}_1,\ldots\mathbf{a}_i \in \mathbb{N}^j$ of $\mathbf{x}_1,\ldots\mathbf{x}_i$.

```
1:  $\varphi(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1})$ :=DNF($\varphi(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1})$)
2:  for all clauses $\psi(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1})$ of $\varphi$ do
3:      for all literals $t = p(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1}) < b$ of $\psi$ do
4:          replace $t$ in $\psi$ with $-p(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1}) \geq -b+1$
5:      end for
6:      for all literals $t = \neg(p(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1}) < b)$ do
7:          replace $t$ in $\psi(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1})$ with $p(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1}) \geq b$
8:      end for
9:      $(S : A\mathbf{x}_{i+1} \geq \mathbf{c}) := \psi[\mathbf{a}_1/\mathbf{x}_1,\ldots,\mathbf{a}_i/\mathbf{x}_i]$
10:     if $[\![S]\!] \neq \emptyset$ then
11:         return true
12:     end if
13: end for
14: return false
```

Together with Lemma 1, this immediately gives that for any fixed $i > 0$, validity in PA($i+1$) is in $\mathsf{\Sigma}_{i+1}^{\mathsf{EXP}}$. We now show how to decrease the number of oracle calls by one.

To this end, let Φ be a PA($i+1$,j) formula in prenex normal form for a fixed $i > 0$ and some j, *i.e.*,

$$\Phi = \exists\mathbf{x}_1.\forall\mathbf{x}_2\cdots Q_{i+1}\mathbf{x}_{i+1}.\varphi(\mathbf{x}_1,\ldots,\mathbf{x}_{i+1}).$$

In order to decide validity of Φ, by application of Proposition 7, a $\mathsf{\Sigma}_i^{\mathsf{EXP}}$-algorithm can alternatingly guess valuations $\mathbf{a}_1,\ldots\mathbf{a}_i \in \mathbb{N}^j$ for the $\mathbf{x}_1,\ldots\mathbf{x}_i$ such that $\|\mathbf{a}_k\| \leq 2^{c|\Phi|^{(3j)^{i+1}}}$ for all $1 \leq k \leq i$ and some constant $c > 0$, and by additionally padding valuations with leading zeros, we may assume that any number in every $\mathbf{a}_k$ is represented using $2^{\mathsf{poly}(|\Phi|)}$ bits. Consequently, it remains to show that validity of $Q_{i+1}\mathbf{x}_{i+1}.\varphi(\mathbf{a}_1/\mathbf{x}_1,\ldots,\mathbf{a}_i/\mathbf{x}_i,\mathbf{x}_{i+1})$ can be decided in polynomial time. This is, of course, not the case under standard assumptions from complexity theory. However, the final call to the $\mathsf{\Sigma}_0^{\mathsf{P}}$-oracle of a $\mathsf{\Sigma}_i^{\mathsf{EXP}}$ algorithm gets φ and all $\mathbf{a}_k$ as input, the latter being of *exponential size* in $|\Phi|$. Informally speaking, this provides us with sufficient additional time in order to decide validity of

$$Q_{i+1}\mathbf{x}_{i+1}.\varphi(\mathbf{a}_1/\mathbf{x}_1,\ldots,\mathbf{a}_i/\mathbf{x}_i,\mathbf{x}_{i+1}) \tag{9}$$

in polynomial time with respect to the size of the input.

Algorithm 1, which takes φ and the $\mathbf{a}_k$ as input, is a pseudo algorithm deciding validity of a formula as in (9) for even i, *i.e.*, $Q_{i+1} = \exists$. The case $Q_{i+1} = \forall$ can be derived symmetrically. Let us discuss Algorithm 1 and analyse its running time. In Line 1, the algorithm converts φ into disjunctive normal form. This step can be performed in exponential time $\mathsf{DTIME}(2^{O(|\Phi|)})$ and thus takes polynomial time with respect to the input. Starting in Line 2, the algorithm iterates over all clauses ψ of φ, and since there are at most $2^{O(|\Phi|)}$ clauses this iteration is performed at most a polynomial number of times with respect to the size of the input. In each iteration, in Lines 3–8 the algorithm transforms the disjuncts of ψ into linear inequalities by eliminating negation. After Line 8, ψ is a conjunction of linear inequalities and thus gives rise to an equivalent system of linear Diophantine inequalities S in which the first-order variables $\mathbf{x}_1,\ldots\mathbf{x}_i$ are instantiated by the $\mathbf{a}_1,\ldots\mathbf{a}_i$. Clearly, Lines 3–9 can be executed in polynomial time with respect to the size of the input. Finally, in Line 10 feasibility of S is checked. To this end, we invoke Proposition 4 from which it follows that feasibility of each S can be decided in $\mathsf{DTIME}(2^{p(|\Phi|)}|S|)$ for some polynomial p. This step is again polynomial with respect to the input to the oracle call. If S is feasible the algorithm returns true in Line 11. Otherwise, if no S is feasible for all disjuncts of φ, the algorithm returns false in Line 14. Consequently, we have shown the following proposition, which together with Proposition 6 completes the proof of Theorem 1.

Proposition 8. For any fixed $i > 0$, PA($i+1$) is decidable in Σ_i^{EXP}.

3.3 Discussion

We conclude this part of the paper with a short discussion on the relationship of our proof of the lower bound of PA($i+1$) to the proof of a NEXP lower bound for PA(2) by Grädel [18], and applications of and results derivable from Proposition 8.

As it emerged in Section 3.1, at many places we can apply and reuse ideas of Grädel's NEXP-hardness proof for PA(2) given in [18] for our lower bound. One main difference is that for his hardness proof, Grädel reduces from a NEXP-complete tiling problem that he specifically introduces in order to show hardness for PA(2). In our paper, we are in the lucky position of having access to twenty-five additional years of developments in computational complexity, in which it turned out that succinct encodings via Boolean circuits provide a canonical way in order to show hardness results for complexity classes that include EXP, see *e.g.* [16, 28, 29]. Moreover, the discovery of a polynomial-time algorithm for deciding primality [1] also enables us to use Boolean circuits encoded into Σ_1- respectively Π_i-formulas in order to decide primality of a positive integer of a bounded bit size, while in [18] this is achieved by an application of the Lucas primality criterion, *cf.* Lehmer's more general proof [24]. In addition, Grädel's stronger statement that PA(2) is NEXP-hard already for an $\exists\forall^*$-quantifier prefix can be recovered from our lower bound. Even more generally for $i > 1$, we can derive Σ_i^{EXP}-hardness from our construction for a $(\exists\forall)^{((i-1)/2)}\exists^*\forall^*$ quantifier prefix if i is odd, and for a $\exists(\forall\exists)^{(i/2-1)}\forall^*\exists^*$ quantifier prefix if i is even. Even though essentially all technical results required to prove Theorem 1 were available when [18] was published, as we have seen in this section the proof of the lower bound requires some substantial technical efforts, which is probably a reason why this result has not been obtained earlier.

With regards to applications of Proposition 8, we give an example of a result which can be obtained as a corollary of this proposition. In [21], Huynh investigates the complexity of the inclusion problem for context-free commutative grammars. Given context-free grammars G_1, G_2, this problem is to determine whether the Parikh image[5] of the language defined by G_1 is included in the Parikh image of the language defined by G_2. Building upon a careful analysis of the semi-linear sets obtained from Parikh images of context-free grammars due to Ginsburg [14] and by establishing a Carathéodory-type theorem for integer cones, Huynh shows that the complement of this problem is in NEXP. This result can however now easily be obtained as a corollary of Proposition 8: Verma *et al.* have shown that the Parikh image of a context-free grammar can be defined in terms of a Σ_1-formula of Presburger arithmetic linear in the size of the grammar [37]. Non-inclusion then reduces to checking validity of a Σ_2-sentence, which yields the following corollary.

Corollary 1. Non-inclusion between Parikh images of context-free grammars is in NEXP.

Of course, the "hard work" of the upper bound is done in Proposition 4, but nevertheless we are able to obtain a succinct proof of Huynh's result. In general, the NEXP upper bound for PA(2) provides a generic upper bound for non-inclusion problems that can be reduced to checking inclusion between semi-linear sets definable via PA(1) formulas. For context-free commutative grammars, it should however be noted that it is not known whether this upper bound is tight, the best known lower bound being Σ_2^{P} [21].

4. Ultimately-Periodic Sets Definable in the Σ_1-fragment of Presburger Arithmetic

We will now apply some techniques developed in Section 3 in order to prove Theorem 2, *i.e.*, give bounds on the representation of projections of PA(1) formulas open in one variable as ultimately-periodic sets. Formally, given a PA(1) formula $\Phi(x)$, we are interested in the representation of the set

$$\llbracket \Phi(x) \rrbracket = \{a \in \mathbb{N} : \Phi(a/x) \text{ is valid}\}.$$

Subsequently, we show that this set is an ultimately periodic set whose period is at most doubly-exponential and that this bound is tight. Throughout this section we assume binary encoding of numbers in $\Phi(\mathbf{x})$

We begin with the first part of Theorem 2 and prove the following proposition.

Proposition 9. There exists a family of Σ_1-formulas of Presburger arithmetic $(\Phi_n(x))_{n>0}$ such that each $\Phi_n(x)$ is a PA(1,$O(n)$) formula with $|\Phi_n(x)| \in O(n^2)$ and $\llbracket \Phi_n(x) \rrbracket$ is an ultimately periodic set with period $p_n \in 2^{2^{\Omega(n)}}$.

To this end, we combine $\Phi_{mod,n}(x, y)$ from (3) in Section 3.1 with the following statement.

Proposition 10 (Nair [27]). Let $n \geq 9$, then $2^n \leq \mathrm{lcm}\{1, \ldots n\} \leq 2^{2n}$.

We define

$$\Phi_n(x) \overset{\text{def}}{=} \exists y . \Phi_{mod,n}(x, y) \wedge y > 1.$$

We have $|\Phi_n(x)| \in O(n^2)$, and, since numbers are encoded in binary, that $\Phi_n(x)$ is a PA(1,$O(n)$) formula. Now $a \in \llbracket \Phi(x) \rrbracket$ iff there is $1 < m < 2^n$ such that $a \equiv 0 \bmod m$, and consequently

$$\begin{aligned} \llbracket \Phi_n(x) \rrbracket &= \bigcup_{1<m<2^n} U(0, m, \emptyset, \{0\}) \\ &= U(0, p, \emptyset, \{a : a \in [p], m | a, 1 < m < 2^n\}), \end{aligned}$$

where $p \overset{\text{def}}{=} \mathrm{lcm}\{1, \ldots 2^n - 1\}$. By Proposition 10, $p \in 2^{2^{\Omega(n)}}$, which yields the lower bound for Theorem 2.

Turning now towards the upper bound, the remainder of this section is devoted to proving the second part of Theorem 2, *i.e.*, the following statement.

Proposition 11. For any Σ_1-formula $\Phi(x)$, we have $\llbracket \Phi(x) \rrbracket = U(t, p, B, R)$ such that $t \in 2^{\mathsf{poly}(|\Phi(x)|)}$ and $p \in 2^{2^{\mathsf{poly}(|\Phi(x)|)}}$.

As a first step, we consider projections of sets of solutions of systems of linear Diophantine inequalities. To this end, let $S : A\mathbf{x} \geq \mathbf{c}$ be such a system. From Proposition 5, we have that $\llbracket S \rrbracket = \bigcup_{i \in I} L(\mathbf{b}_i; P_i)$ for some index set I. Let M_i be the projection of $L(\mathbf{b}_i; P_i)$ on the first component. We get that M_i can be obtained as

$$\begin{aligned} M_i &= \{b_i + \lambda_{i,1} p_1 + \cdots \lambda_r p_{i,r_i} : \lambda_k \in \mathbb{N}\} \\ &= b_i + g_i \cdot \{\lambda_1 p_{i,1}/g_i + \cdots \lambda_r p_{i,r_i}/g_i : \lambda_k \in \mathbb{N}\} \end{aligned}$$

for some $b_i, p_{i,1} < \cdots < p_{i,r_i}$ and $g_i = \gcd\{p_{i,1}, \ldots p_{i,r_i}\}$. Since $\gcd\{p_{i,1}/g_i, \ldots p_{i,r_i}/g_i\} = 1$, it is folklore that

$$M_i = b_i + g_i \cdot U(t_i' + 1, 1, B_i', \{0\})$$

for some $B_i' \subseteq [t_i']$ and $t_i' \in \mathbb{N}$ known as the Frobenius number of $p_{i,1}/g_i, \ldots p_{i,r_i}/g_i$. Given co-prime positive integers $1 < a_1 < a_2 < \cdots < a_k \in \mathbb{N}$, the *Frobenius number* $f \in \mathbb{N}$ is the largest positive integer not expressible as a positive linear combination of $a_1, \ldots a_k$ and can be bounded as follows.

[5] The Parikh image of a word $w \in \Sigma^*$ is a vector of naturals of dimension $|\Sigma|$ counting the number of times each alphabet symbol occurs in w.

Proposition 12 (Wilf [39])**.** Let $1 < a_1 < a_2 < \cdots < a_k \in \mathbb{N}$ be pairwise co-prime. Then the Frobenius number f is bounded by $f \leq a_k^2$.

Hence, for some $t_i \leq b_i + g_i(p_{i,r_i}/g_i)^2 \leq b_i + p_{i,r_i}^2$ we consequently have

$$M_i = U(t_i, p_i, B_i, \{0\}) \tag{10}$$

for some $p_i \leq p_{i,r_i}$.

Let $\Phi(x) = \exists \mathbf{x}.\varphi(x, \mathbf{x})$ be a PA(1) formula. From Algorithm 1 we can derive that

$$[\![\varphi(x, \mathbf{x})]\!] = \bigcup_{j \in J} [\![S_j]\!] = \bigcup_{i \in I} L(\mathbf{b}_i; P_i),$$

where each $S_j : A_j(x, \mathbf{x}) \geq \mathbf{c}_j$ is a system of linear Diophantine inequalities obtained from one disjunct of the disjunctive normal form of φ, similar as in Line 9 of Algorithm 1. Clearly, $\|A_j\|, \|\mathbf{c}_j\| \leq \|\Phi(x)\| + 1$ for all $i \in I$. Moreover, from Proposition 5 we derive that $[\![S_j]\!] = \bigcup_{i \in I_j} L(\mathbf{b}_i; P_i)$ for some index set I_j such that for every $i \in I_j$

$$\|\mathbf{b}_i\|, \|P_i\| \leq (|\Phi(x)| \|A_i\| + \|\mathbf{c}_i\| + 1)^{O(|\Phi(x)|)} \in 2^{\mathsf{poly}(|\Phi(x)|)}.$$

Let M_i be as above, from (10) we have $M_i = U(t_i, p_i, B_i, \{0\})$. Now define $p \overset{\text{def}}{=} \operatorname{lcm}\{p_i\}_{i \in I}$, combining the estimations in (10) with Proposition 10 we have $p \in 2^{2^{\mathsf{poly}(|\Phi(x)|)}}$. It follows that $[\![\Phi(x)]\!] = U(t, p, B, R)$ for some $t \in 2^{\mathsf{poly}(|\Phi(x)|)}$ and $p \in 2^{2^{\mathsf{poly}(|\Phi(x)|)}}$ as above, which concludes the proof of Theorem 2.

5. Conclusion

In the first part of this paper we have shown that Presburger arithmetic with a fixed number of $i + 1$ quantifier alternations and an arbitrary number of variables in each quantifier block is complete for $\mathbf{\Sigma}_i^{\mathsf{EXP}}$ for every $i > 0$. This result closes a gap that has been left open in the literature, and in particular improves and generalises results obtained by Fürer [13], Grädel [18] and Reddy & Loveland [32]. Moreover, it provides an interesting natural problem which is complete for the weak EXP hierarchy, a complexity class for which not that many natural complete problems have been known so far.

In the second part, we established bounds on ultimately periodic sets definable in the Σ_1-fragment of Presburger arithmetic and showed that in particular the period of those sets is at most doubly-exponential and that this bound is tight. As already discussed in the introduction, there are however natural ultimately periodic sets definable in this fragment that admit periods that are at most singly-exponential, *cf.* [15]. An interesting open question is whether it is possible to identify a fragment of Σ_1-Presburger arithmetic for which such a singly-exponential upper bound can be established in general and that captures sets such as those considered in [15].

Acknowledgments

The author would like to thank the anonymous referees for their thoughtful comments on the first version of this paper. In addition, the author is grateful to Benedikt Bollig, Stefan Göller, Felix Klaedtke, Sylvain Schmitz and Helmut Veith for encouraging discussions and helpful suggestions.

References

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 2:781–793, 2002.

[2] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[3] Leonard Berman. The complexity of logical theories. *Theoretical Computer Science*, 11(1):71–77, 1980.

[4] Alexis Bès. A survey of arithmetical definability. In *A tribute to Maurice Boffa*, pages 1–54. Société Mathématique de Belgique, 2002.

[5] Itshak Borosh and Leon B. Treybing. Bounds on positive integral solutions of linear Diophantine equations. *Proceedings oft the American Mathematical Society*, 55:299–304, 1976.

[6] Yuan-You Fu-Rui Cheng. Explicit estimate on primes between consecutive cubes. *Rocky Mountain Journal of Mathematics*, 40(1):117–153, 2010.

[7] Kevin J. Compton and C. Ward Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Annals of Pure and Applied Logic*, 48(1):1 – 79, 1990.

[8] D.C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7:91–99, 1972.

[9] Antoine Durand-Gasselin and Peter Habermehl. Ehrenfeucht-Fraïssé goes elementarily automatic for structures of bounded degree. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 242–253, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[10] Jeanne Ferrante and Charles Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

[11] Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In Bob F. Caviness and Jeremy R. Johnson, editors, *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and Monographs in Symbolic Computation, pages 122–135. Springer Vienna, 1998.

[12] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[13] Martin Fürer. The complexity of Presburger arithmetic with bounded quantifier alternation depth. *Theoretical Computer Science*, 18(1):105–111, 1982.

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

[15] Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Branching-time model checking of parametric one-counter automata. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures*, volume 7213 of *Lecture Notes in Computer Science*, pages 406–420. Springer, 2012.

[16] Georg Gottlob, Nicola Leone, and Helmut Veith. Second order logic and the weak exponential hierarchies. In Jiří Wiedermann and Petr Hájek, editors, *Mathematical Foundations of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1995.

[17] Erich Grädel. Subclasses of Presburger arithmetic and the polynomial-time hierarchy. *Theoretical Computer Science*, 56(3):289–301, 1988.

[18] Erich Grädel. Dominoes and the complexity of subclasses of logical theories. *Annals of Pure and Applied Logic*, 43(1):1–30, 1989.

[19] Lane A. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.

[20] Dung T. Huynh. Deciding the inequivalence of context-free grammars with 1-letter terminal alphabet is Σ_2^{P}-complete. *Theoretical Computer Science*, 33(23):305–326, 1984.

[21] Dung T. Huynh. The complexity of equivalence problems for commutative grammars. *Information and Control*, 66(12):103–121, 1985.

[22] Albert E. Ingham. On the estimation of $N(\sigma, T)$. *The Quarterly Journal of Mathematics*, os-11(1):201–202, 1940.

[23] Felix Klaedtke. Bounds on the automata size for Presburger arithmetic. *ACM Transactions on Computational Logic*, 9(2):11:1–11:34, 2008.

[24] Derrick H. Lehmer. Tests for primality by the converse of Fermat's theorem. *Bulletin of the American Mathematical Society*, 33(3):327–340, 1927.

[25] Leonard M. Lipshitz. The Diophantine problem for addition and divisibility. *Transactions of the American Mathematical Society*, 235:271–283, 1978.

[26] Leonard M. Lipshitz. Some remarks on the Diophantine problem for addition and divisibility. In *Proceedings of the Model Theory Meeting*, volume 33, pages 41–52, 1981.

[27] Mohan Nair. On Chebyshev-type inequalities for primes. *The American Mathematical Monthly*, 89(2):126–129, 1982.

[28] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[29] Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

[30] Loïc Pottier. Minimal solutions of linear Diophantine systems : bounds and algorithms. In Ronald V. Book, editor, *Rewriting Techniques and Applications*, volume 488 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 1991.

[31] 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 congres de Mathematiciens des Pays Slaves*, pages 92–101. 1929.

[32] C. R. Reddy and Donald W. Loveland. Presburger arithmetic with bounded quantifier alternation. In *Proceedings of the 10th annual ACM Symposium on Theory of Computing*, pages 320–325, New York, NY, USA, 1978. ACM.

[33] Bruno Scarpellini. Complexity of subcases of Presburger arithmetic. *Transactions of the American Mathematical Society*, 284:203–218, 1984.

[34] Uwe Schöning. Complexity of Presburger arithmetic with fixed quantifier dimension. *Theory of Computing Systems*, 30(4):423–428, 1997.

[35] Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 1136–1149. Springer, 2004.

[36] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[37] Kumar N. Verma, Helmut Seidl, and Thomas Schwentick. On the complexity of equational Horn clauses. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352. Springer, 2005.

[38] Volker Weispfenning. The complexity of almost linear Diophantine problems. *Journal of Symbolic Computation*, 10(5):395–403, 1990.

[39] Herbert S. Wilf. A circle-of-lights algorithm for the "money-changing problem". *The American Mathematical Monthly*, 85(7):562–565, 1978.

[40] Pierre Wolper and Bernard Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In Alan Mycroft, editor, *Static Analysis*, volume 983 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 1995.

A. Missing proofs

In the following, let $\mathbb{B} = \{0, 1\}$. Let us recall the following characterisation of the polynomial-time hierarchy.

Lemma 2 (Def. 5.3 and Thm. 5.12 in [2])**.** For $i > 0$, a language $L \subseteq \mathbb{B}^*$ is in $\mathsf{\Sigma}_i^{\mathsf{P}}$ iff there exists a polynomial r and a deterministic polynomial-time computable predicate $S \subseteq (\mathbb{B}^*)^{i+1}$ such that $w \in L$ iff

$$\exists w_1 \in \mathbb{B}^{r(n)}.\forall w_2 \in \mathbb{B}^{r(n)} \cdots Q_i w_i \in \mathbb{B}^{r(n)}.S(w, w_1, \ldots, w_i).$$

Lemma 3 (Lem. 1 in the main text)**.** For any $i > 0$, a language $L \subseteq \{0,1\}^*$ is in $\mathsf{\Sigma}_i^{\mathsf{EXP}}$ iff there exists a polynomial q and a predicate $R \subseteq (\{0,1\}^*)^{i+1}$ such that for any $w \in \{0,1\}^n$,

$$w \in L \text{ iff } \exists w_1 \in \{0,1\}^{2^{q(n)}}.\forall w_2 \in \{0,1\}^{2^{q(n)}} \cdots$$
$$\cdots Q_i w_i \in \{0,1\}^{2^{q(n)}}.R(w, w_1, \ldots, w_i)$$

and $R(w, w_1, \ldots, w_i)$ can be decided in deterministic polynomial time.

Proof. ("$\Leftarrow$") We describe a $\mathsf{NEXP}^{\mathsf{\Sigma}_{i-1}^{\mathsf{P}}}$ Turing machine M deciding for a given $w \in \mathbb{B}^n$ whether $w \in L$. First, M performs a NEXP guess in order to guess $w_1 \in \mathbb{B}^{2^{q(n)}}$. Define $L' \subseteq \mathbb{B}^n \times \mathbb{B}^{2^{q(n)}}$ such that $(w, w_1) \in L'$ iff

$$\exists w_2 \in \mathbb{B}^{2^{q(n)}} \cdots Q_i' w_i \in \mathbb{B}^{2^{q(n)}}.\neg R(w, w_1, \ldots, w_i),$$

where $Q_i' = \exists$ if $Q_i = \forall$ and *vice versa*. By Lemma 2, we have that L' is a language in $\mathsf{\Sigma}_i^{\mathsf{P}}$ since we can check if the input is sufficiently large and immediately reject if this is not the case, choose $r : w \mapsto |w| - n$, and decide $\neg R(w, w_1, \ldots, w_i)$ in deterministic polynomial time. Thus, after M has guessed w_1, it invokes the $\mathsf{\Sigma}_{i-1}^{\mathsf{P}}$ oracle to check $(w, w_1) \in L'$ and accepts if $(w, w_1) \notin L'$.

("$\Rightarrow$") Let L be decided by a $\mathsf{NEXP}^{\mathsf{\Sigma}_{i-1}^{\mathsf{P}}}$ Turing machine M. Given $w \in \mathbb{B}^n$, an accepting run of M has length at most 2^{n^k} for some $k > 0$ on which it resolves $c_1, \ldots, c_m \in \mathbb{B}, m \leq 2^{n^k}$ non-deterministic choices. Moreover, M makes ℓ oracle queries "$v_j \in L'$?" for some L' in $\mathsf{\Sigma}_{i-1}^{\mathsf{P}}$ such that $n_j = |v_j|, \ell \leq 2^{n^k}$, and M receives answers $a_j \in \mathbb{B}$ to those queries. By Lemma 2, we have $v_j \in L'$ iff

$$\exists w_{2,j} \in \{0,1\}^{r(n_j)} \cdots Q_i w_{i,j} \in \{0,1\}^{r(n_j)}.S(v_j, w_{2,j}, \ldots, w_{i,j}).$$

If M receives $a_j = 1$ as an answer to an oracle call it can guess the corresponding certificate $w_{2,j} \in \mathbb{B}^{r(n_j)}$. Otherwise, if $a_j = 0$ this result can be verified using one quantifier alternation. Consequently, we can guess the answers to the oracle queries and then verify at once whether those guesses were correct. Hence, $w \in L$ iff

$$\exists c_1, \ldots c_m \in \mathbb{B}, v_1, \ldots v_\ell \in \mathbb{B}^{2^{n^k}}, a_1, \ldots a_\ell \in \mathbb{B},$$
$$w_{2,1}, \ldots w_{2,\ell} \in \mathbb{B}^{r(2^{n^k})}.\forall w_2 \in \mathbb{B}^{2^{q(n)}} \cdots Q_i w_i \in \mathbb{B}^{2^{q(n)}}.$$
$$R((w \cdot c_1 \cdots c_m \cdot v_1 \cdots v_\ell \cdot a_1 \cdots a_\ell \cdot w_{2,1} \cdots w_{2,\ell}), w_2, \ldots, w_i)$$

for some appropriately chosen polynomial q and appropriately constructed $R \subseteq (\mathbb{B}^*)^{i+1}$ combining S with checking that the c_i resolve the non-determinism of M correctly and that the guessed answers to the oracle calls are correct. □