

Higher-Order Quantified Boolean Satisfiability

Dmitry Chistikov 

Centre for Discrete Mathematics and its Applications &
Department of Computer Science, University of Warwick, Coventry, UK

Christoph Haase 

Department of Computer Science, University of Oxford, Oxford, UK

Zahra Hadizadeh

Sharif University of Technology, Tehran, Iran

Alessio Mansutti 

Department of Computer Science, University of Oxford, Oxford, UK

Abstract

The Boolean satisfiability problem plays a central role in computational complexity and is often used as a starting point for showing NP lower bounds. Generalisations such as Succinct SAT, where a Boolean formula is succinctly represented as a Boolean circuit, have been studied in the literature in order to lift the Boolean satisfiability problem to higher complexity classes such as NEXP. While, in theory, iterating this approach yields complete problems for k -NEXP for all $k > 0$, using such iterations of Succinct SAT is at best tedious when it comes to proving lower bounds.

The main contribution of this paper is to show that the Boolean satisfiability problem has another canonical generalisation in terms of higher-order Boolean functions that is arguably more suitable for showing lower bounds beyond NP. We introduce a family of problems HOSAT(k, d), $k \geq 0, d \geq 1$, in which variables are interpreted as Boolean functions of order at most k and there are d quantifier alternations between functions of order exactly k . We show that the unbounded HOSAT problem is TOWER-complete, and that HOSAT(k, d) is complete for the weak k -EXP hierarchy with d alternations for fixed $k, d \geq 1$.

We illustrate the usefulness of HOSAT by characterising the complexity of weak Presburger arithmetic, the first-order theory of the integers with addition and equality but without order. It has been a long-standing open problem whether weak Presburger arithmetic has the same complexity as standard Presburger arithmetic. We answer this question affirmatively, even for the negation-free fragment and the Horn fragment of weak Presburger arithmetic.

2012 ACM Subject Classification Theory of computation \rightarrow Logic

Keywords and phrases Boolean satisfiability problem, higher-order Boolean functions, weak k -EXP hierarchies, non-elementary complexity, Presburger arithmetic

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23


Acknowledgements We would like to thank the anonymous reviewers for their thoughtful comments, and in particular for pointing us to [22], which uncovered profound connections between our work and the basic language of set theory and typed λ -calculi studied by Statman [30].

This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 852769, ARiAT).




1 Introduction

The Boolean satisfiability problem (SAT) plays a central role in computational complexity. It was the first problem shown to be NP-complete [8] and has ever since been used a countless number of times to show NP lower bounds for numerous combinatorial problems, a prime example being Karp's list of twenty-one NP-complete problems [17]. What makes SAT stand out is its simplicity: describing an instance of SAT does not require tapes and automata as

 © Dmitry Chistikov, Christoph Haase, Zahra Hadizadeh and Alessio Mansutti;
licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:15

 Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in the case of Turing machines, and neither grids nor dominoes like in tiling problems; it also does not require the introduction of graphs and their properties as in, e.g., graph colouring problems. To obtain a reduction from SAT, it suffices to show how to encode assignments to Boolean variables and how to model connectives. Unfortunately, this simplicity does not transfer over to higher complexity classes.

Succinct representations have been used as a technical tool to lift NP-complete problems to exponential classes [27, 20, 2, 32]. The idea is that the main object of interest is succinctly encoded by a Boolean circuit. For instance, a Boolean formula represented as a DAG whose vertices are labelled by a type (e.g., a Boolean connective or constant value) and a unique integer index can be encoded by a Boolean circuit with outputs. Given a binary encoding of an index, this circuit returns the type of the node with that index and, if present, the indices of its two predecessor nodes. Succinct variants of P- or NP-complete problems become EXP- or NEXP-complete, e.g., the Succinct Circuit Value problem and the Succinct SAT Problem are respectively EXP- and NEXP-complete [26]. In theory, this approach can be iterated indefinitely, and, e.g., the problem of deciding whether a propositional formula encoded by a Boolean circuit that is itself encoded by a Boolean circuit evaluates to true is a “canonical” 2-EXP-complete problem. One does not need too much imagination to see that these iterated succinct problems become unbearable for showing lower bounds.

This loss of simplicity for higher complexity classes compared to classical SAT is at odds with other canonical problems, such as the halting problem for a (non)deterministic Turing machine running in time $f(n)$ on an input of length n , or the problem of tiling a grid of size $f(n) \times f(n)$. Here, the function f can be chosen appropriately in order to obtain hard problems for many complexity classes such as k -EXP and k -NEXP [26]. The same picture emerges in the presence of alternation: whereas QBF [12, Chapter 7.4] elegantly extends SAT to the whole polynomial hierarchy [33], no simple extension of SAT has been defined to capture every level of the weak k -EXP hierarchies, for any $k \geq 1$ (cf. [15]). Again, this is in contrast with Turing machines and tiling problems: both admit extensions to deal with alternation in the context of complexity classes above NP [4, 5, 23].

The main contribution of this paper is to identify a canonical generalisation of SAT, called HOSAT, that gives complete problems for all weak k -EXP hierarchies and that shares the simplicity of classical SAT. Instead of Boolean variables, the building blocks of HOSAT are *function applications*. The functions considered are higher-order Boolean function f of order k , i.e., functions that take as input higher-order functions of order $k-1$ and return a Boolean value, and Boolean values themselves are functions of order zero. HOSAT closes function applications under Boolean connectives as well as quantification.

The development of HOSAT is a result of an attempt of the authors to settle the open problem of the computational complexity of *weak Presburger arithmetic*, the first-order theory of the integers with addition and equality, but without an order predicate (which is provably not definable in this theory). This less expressive theory is in fact the “original” arithmetic theory studied by Presburger in his seminal paper [28], see also [7], and it has been an open problem whether it is computationally as hard as what is nowadays commonly understood as Presburger arithmetic; see, e.g., [6]. The lower bound for Presburger arithmetic given by Berman [3] reduces from an alternating Turing machine running in doubly exponential time with a linear number of alternations. This reduction glosses over some technical details, which is unproblematic in the presence of the sufficient expressive power that the order predicate provides, but becomes problematic for weak Presburger arithmetic. Giving a clean reduction from HOSAT enables us to settle the complexity of weak Presburger arithmetic and to show that it has indeed the same complexity as Presburger arithmetic with the order

predicate. We hope that HOSAT will be as beneficial as it was for us for other researchers for proving their lower bounds, in particular since the question of finding canonical complete problems for weak k -EXP hierarchies frequently comes up; see, e.g., [31, 21].

2 Preliminaries

We write \mathbb{Z} , \mathbb{N} and \mathbb{N}_+ to denote the set of integers, natural numbers including zero, and natural numbers without zero, respectively. Unless otherwise stated, we assume integers to be encoded in binary. Given $l, u \in \mathbb{N}$, we define $[l, u] := \{l, l+1, \dots, u\}$, $[l, u) := [l, u-1]$ and $[u] := [0, u)$. The cardinality of a finite set A is denoted by $\#A$. For $k, n \in \mathbb{N}$, we write $\exp_2^k(n)$ for the tetration function inductively defined as $\exp_2^0(n) := n$ and $\exp_2^k(n) := 2^{\exp_2^{k-1}(n)}$. Intuitively, $\exp_2^k(n)$ is a tower of exponentials of height k , base 2, and top-most exponent n .

We recall complexity classes based on the notion of alternating Turing machines [5]. The class $\Sigma_d^{k\text{-EXP}}$ contains all problems that can be decided by an alternating Turing machine running in time $\exp_2^k(f(n))$ on an input of length n , where $f: \mathbb{N} \rightarrow \mathbb{N}$ is some polynomial that does not depend on the input, and starting in an existential state and making at most $d-1$ alternations on every computation path. By definition, $\Sigma_1^{k\text{-EXP}} = k\text{-NEXP}$. The *weak k -EXP hierarchy* is defined as $\bigcup_{d \geq 0} \Sigma_d^{k\text{-EXP}}$; for $k=1$ see [15].

Given functions $a, s, t: \mathbb{N} \rightarrow \mathbb{N}$, the class $\text{STA}(s(n), t(n), a(n))$ contains all problems that can be decided by an alternating Turing machine in time $t(n)$ using space $s(n)$ making at most $a(n)$ alternations on every computation path, where n is the length of the input. We use $*$ to indicate an unbounded availability of a certain resource. For instance, the polynomial hierarchy can be characterized as $\bigcup_{d \in \mathbb{N}} \text{STA}(*, n^{O(1)}, d)$, and the d -th level of the weak k -exponential hierarchy $\Sigma_d^{k\text{-EXP}}$ corresponds to $\text{STA}(*, \exp_2^k(n^{O(1)}), d)$. The STA complexity measure was introduced by Berman [3] to show the following result.

► **Theorem 1** ([3]). *Presburger arithmetic is complete for $\text{STA}(*, 2^{2^{n^{O(1)}}}, O(n))$.*

A function $f: \mathbb{N} \rightarrow \mathbb{N}$ is said to be *elementary* if there is a $k \in \mathbb{N}$ such that $f(n) < \exp_2^k(n)$ for all $n \in \mathbb{N}$. The (non-elementary) class TOWER [29] contains all problems decidable by a Turing machine in time $\exp_2^{g(n)}(f(n))$ for some fixed polynomial f and elementary function g , on inputs of length n . We have $\bigcup_{k \geq 1} k\text{-EXP} \subsetneq \text{TOWER}$, as TOWER contains problems decidable in time $\exp_2^n(1)$.

3 The higher-order quantified satisfiability problem

We introduce the *higher-order quantified satisfiability problem*, a problem whose instances form a hierarchy $(\text{HOSAT}(k, d))_{k, d}$ of problems that characterise every complexity class $\Sigma_d^{k\text{-EXP}}$.

We write \mathbb{B} to denote the Boolean domain $\{0, 1\}$. We often treat \mathbb{B} as the set of the two nullary Boolean functions $() \rightarrow \{0\}$ and $() \rightarrow \{1\}$. Fix $n \in \mathbb{N}_+$ and let $\mathbb{B}_{0, n} := \mathbb{B}$. Given $k \geq 1$, we write $\mathbb{B}_{k, n}$ for the set of all functions with domain $(\mathbb{B}_{k-1, n})^n$ and codomain \mathbb{B} . For instance, $\mathbb{B}_{1, n}$ is the set of all n -ary Boolean functions $f: \{0, 1\}^n \rightarrow \{0, 1\}$, whereas $\mathbb{B}_{2, n}$ is the set of all n -ary second-order Boolean functions $f: (\{0, 1\}^n \rightarrow \{0, 1\})^n \rightarrow \{0, 1\}$. For $k \geq 1$, the number of functions in $\mathbb{B}_{k, n}$ satisfies the recurrence relation $\#\mathbb{B}_{k, n} = 2^{\#\mathbb{B}_{k-1, n}^n}$, and therefore $\exp_2^{k+1}(n) \leq \#\mathbb{B}_{k, n} \leq \exp_2^{k+1}((k+1) \cdot n)$. We refer to the indices k and n as the *order* and the *arity* of $\mathbb{B}_{k, n}$, respectively. Both n and k are written in unary.

We discuss the encoding $\text{enc}(f)$ of Boolean functions as bit-strings over $\{0, 1\}$, which we often treat as the binary number from \mathbb{N} represented by the bit-string, with least significant digit first. For $b \in \mathbb{B}$, $\text{enc}(b) := b$. Functions $f \in \mathbb{B}_{1, n}$ can be encoded as a string of length

23:4 Higher-Order Quantified Boolean Satisfiability

2^n over \mathbb{B} , whose i -th position encodes the truth value of f on the tuple $(b_1, \dots, b_n) \in \mathbb{B}^n$ such that $b_1 \cdots b_n$ is the binary encoding of i , with least significant digit first. For $k > 1$, functions $f \in \mathbb{B}_{k,n}$ can be encoded as a string over \mathbb{B} whose i -th position encodes the truth value of f on input $(g_1, \dots, g_n) \in (\mathbb{B}_{k-1,n})^n$ such that the concatenation $enc(g_1) \cdots enc(g_n)$ of the encodings of g_1, \dots, g_n is a binary encoding of i , with least significant digit first. The length $|f|$ of a function f from $\mathbb{B}_{k,n}$ is defined as the length of $enc(f)$. Therefore, $|b| = 1$ for all $b \in \mathbb{B}$, and for every $f \in \mathbb{B}_{k,n}$ with $k \geq 1$, we have $|f| = (\#(\mathbb{B}_{k-1,n}))^n \geq (\exp_2^k(n))^n$.

A *quantifier-free generalised Boolean formula* is a formula from the following grammar

$$\Phi := \top \mid f(g_1, \dots, g_n) \mid \neg\Phi \mid \Phi \wedge \Phi$$

where $f(g_1, \dots, g_n)$ is said to be a *function application*, and each f, g_1, \dots, g_n are *function symbols* taken from an infinite alphabet Σ . Each function symbol in Σ is implicitly endowed with a type $\mathbb{B}_{k,n}$, with $k, n \in \mathbb{N}$. A (quantifier-free) generalised Boolean formula is said to be *well-formed* whenever every function application is consistent with the type of the function symbol, i.e., a function application $f(g_1, \dots, g_n)$ requires f to be of arity n , if $f \in \mathbb{B}$ then $n = 0$, and otherwise $f \in \mathbb{B}_{k,n}$ for some $k \geq 1$ and every g_i with $i \in [1, n]$ belongs to $\mathbb{B}_{k-1,n}$.

Given a vector-variable of function symbols $\mathbf{f} = (f_1, \dots, f_m)$ of type $\mathbb{B}_{k,n}$, we write $\exists \mathbf{f} : \mathbb{B}_{k,n}$ as a shorthand for the *existential quantifier block* $\exists f_1 : \mathbb{B}_{k,n} \dots \exists f_m : \mathbb{B}_{k,n}$; the *universal quantifier block* $\forall \mathbf{f} : \mathbb{B}_{k,n}$ stands for $\forall f_1 : \mathbb{B}_{k,n} \dots \forall f_m : \mathbb{B}_{k,n}$. The set of all *generalised Boolean formulae of order 0* and alternation depth d is the set of all formulae $\exists \mathbf{b}_1 : \mathbb{B} \forall \mathbf{b}_2 : \mathbb{B} \dots \exists \mathbf{b}_d : \mathbb{B} . \Phi$, where Φ is a quantifier-free generalised Boolean formula. *Generalised Boolean formulae of order $k \geq 1$* and alternation depth d are formulae

$$\exists \mathbf{f}_1 : \mathbb{B}_{k,n} \forall \mathbf{f}_2 : \mathbb{B}_{k,n} \dots \exists \mathbf{f}_d : \mathbb{B}_{k,n} . \Phi,$$

where the arity n is arbitrary and Φ is a generalised Boolean formula of order $k - 1$, arbitrary alternation depth, and same arity n . The semantics of generalised Boolean formulae is as expected, e.g., $\exists f : \mathbb{B}_{k,n} \Psi$ states that there is a function $f \in \mathbb{B}_{k,n}$ that makes Ψ true. We write $\Phi \equiv \Psi$ to denote that Φ and Ψ are *equivalent*.

The size $|\Phi|$ of a generalised Boolean formula Φ is the number of symbols required to write it down, where “ $:\mathbb{B}_{k,n}$ ” is a lexeme that decorates the function symbols, providing their type, and should not be confused with the actual set $\mathbb{B}_{k,n}$. We write $\text{fv}(\Phi)$ for the set of *free* function symbols of Φ , i.e., the set of those function symbols that do not appear in the scope of a quantifier. A *sentence* is a well-formed generalised Boolean formula Φ where all function symbols are quantified, i.e. $\text{fv}(\Phi) = \emptyset$. We sometimes write $\Phi(f_1, \dots, f_m)$ or $\Phi(\mathbf{f})$, with $\mathbf{f} = (f_1, \dots, f_m)$, for a formula Φ with $\text{fv}(\Phi) = \{f_1, \dots, f_m\}$. Given function symbols g_1, \dots, g_m and a formula $\Phi(f_1, \dots, f_m)$, we write $\Phi(g_1, \dots, g_m)$ for the formula obtained from Φ by replacing each f_i with g_i .

Generalised Boolean formulae are formulae in prenex normal form in which the type $\mathbb{B}_{k,n}$ of Boolean functions of the quantifier prefix weakly decreases with respect to k . For presentational convenience, throughout the remainder of the paper we relax these constraints and consider formulae that are not in prenex normal form. This is done w.l.o.g., as standard ways of efficiently translating formulae in prenex normal form also work for generalised Boolean formulae. Moreover, we use standard Boolean connectives \vee , \rightarrow and \leftrightarrow , and \perp .

Let k and d in \mathbb{N}_+ . We introduce the problem $\text{HOSAT}(k, d)$:

HOSAT(k, d) : d -ALTERNATING SATISFIABILITY PROBLEM OF ORDER k

INPUT: A sentence Φ of order k and alternation depth d .

QUESTION: Is Φ valid?

We define $\text{HOSAT}(k, *) := \bigcup_{d \in \mathbb{N}_+} \text{HOSAT}(k, d)$, i.e., the problem of deciding the validity of

a sentence of order k and arbitrary alternation depth; and $\text{HOSAT} := \bigcup_{k \in \mathbb{N}_+} \text{HOSAT}(k, *)$.

- **Theorem 2.** (I) $\text{HOSAT}(k, d)$ is complete for $\Sigma_d^{k\text{-EXP}}$.
 (II) $\text{HOSAT}(k, *)$ is complete for $\text{STA}(*, \exp_2^k(n^{O(1)}), O(n))$.

The reduction we use to show the lower bound of Theorem 2(I) is uniform for all $k \geq 1$. By uniform polynomial time reduction [29], this implies that HOSAT is TOWER-complete.

Related work. In view of the fact that HOSAT is a natural generalisation of SAT, it comes with no surprise that some of its instances have previously been defined and have found diverse application in the past. In [1], Babai, Fortnow and Lund relied on $\text{HOSAT}(1, 1)$ (called *Oracle-3-satisfiability* in the paper) to show that $\text{NEXP} \subseteq \text{MIP}$, where MIP is the class of all languages with multiple-prover interactive proof systems. In [19], Lohrey relied on $\text{HOSAT}(1, d)$ (called $\text{QO}\Sigma_d\text{-SAT}$ in the paper) to show Σ_d^{EXP} -hardness of model checking $\Sigma_d\text{-MSO}$ sentences over hierarchical graph unfoldings. These works already hint at how considering Boolean functions instead of succinct circuits is already beneficial, in terms of directness of the reductions, for NEXP-hard problems.

Above Σ_d^{EXP} , closely related is the work of Statman on the typed λ -calculus. In [30], he considers the λ -calculus with single ground type $\mathbf{0}$, no constants, only power types (\rightarrow) and β -conversion, and shows that checking whether two λ -terms of the calculus reduce to the same normal form is non-elementary recursive (in fact, it shows that the problem is TOWER-complete). The proof of TOWER-hardness follows thanks to a Church encoding of the basic language of set theory denoted by Statman with Ω . The variables in formulae from Ω are associated with a number type from \mathbb{N} . A variable of type n ranges over \mathcal{D}_n where $\mathcal{D}_0 := \{\mathbf{0}, \mathbf{1}\}$, with $\mathbf{0}$ and $\mathbf{1}$ constants, and \mathcal{D}_{n+1} is the powerset of \mathcal{D}_n . Formulae of Ω are obtained by taking the closure under Boolean connectives and quantification of membership queries of the form $\mathbf{0} \in x$, $\mathbf{1} \in x$ and $y \in z$, where x is of type 1 and y and z are of types n and $n+1$, respectively, for some $n \in \mathbb{N}$. While HOSAT and the satisfiability of formulae from Ω are essentially the same problem, the logic Ω is not suitable to capture any of the levels of the weak k -EXP hierarchies. To see this, fix $k \geq 1$ and let $\Omega(k)$ be the subset of the formulae in Ω having variables of type at most k . Differently from $\text{HOSAT}(k, *)$, the satisfiability problem of $\Omega(k)$ can be shown to be in PSPACE (more precisely, it is equivalent to QBF) since the sets \mathcal{D}_j with $j \leq k$ are now fixed a priori. Here, the reason why $\text{HOSAT}(k, *)$ is instead k -NEXP-hard is because the sets $\mathbb{B}_{j,n}$ with $j \leq k$ still have some degree of freedom given by the unbounded number of choices for $n \in \mathbb{N}$.

According to [30], TOWER-completeness of the satisfiability problem of Ω was announced by Meyer in [24, Theorem 1(7)] as part of a forthcoming paper coauthored with Fischer. To the best of our knowledge, the latter paper was never published. To resolve this issue, in [22] Mairson gives a revision of [30] that provides a standalone proof of the TOWER-hardness of Ω and a simplification to the aforementioned Church encoding.

4 The complexity of $\text{HOSAT}(k, d)$

We prove Theorem 2(I). The proof of the $\Sigma_d^{k\text{-EXP}}$ upper bound is quite simple: given a sentence $\exists \mathbf{f}_1: \mathbb{B}_{k,n} \forall \mathbf{f}_2: \mathbb{B}_{k,n} \cdots \exists \mathbf{f}_d: \mathbb{B}_{k,n} \cdot \Phi$, where Φ is a generalised Boolean formula of order $k-1$ and arbitrary alternation depth, an alternating Turing machine running in k -EXP time and performing d alternations implements the following recursive procedure.

1. Guess functions from $\mathbb{B}_{k,n}$ for each of the function symbols in the vectors $\mathbf{f}_1, \dots, \mathbf{f}_d$, alternating between existential and universal states according to the quantifier prefix.

2. Recursively on Φ , if $\Phi = \exists f : \mathbb{B}_{j,n} \Psi$ (resp. $\Phi = \forall f : \mathbb{B}_{j,n} \Psi$) with $j < k$ then check whether some (resp. each) function of order j satisfies Φ when assigned to f ; if Φ is quantifier-free, then check whether it is satisfied under the current assignment of function symbols.

Since k -exponential time is available, the second step can be performed deterministically by iterating through all function in $\mathbb{B}_{j,n}$, thus without introducing further alternation.

The $\Sigma_d^{k\text{-EXP}}$ lower bound of HOSAT(k, d) is shown by reducing from the d -alternating multi-tiling problem of order k (in short, AMTP(k, d)) considered in [10]. A multi-tiling system \mathcal{S} is a tuple $(\mathcal{T}, \mathcal{T}_0, \mathcal{T}_{\text{acc}}, \mathcal{H}, \mathcal{V}, \mathcal{M}, m)$ such that \mathcal{T} is a finite set of tile types, $\mathcal{T}_0, \mathcal{T}_{\text{acc}} \subseteq \mathcal{T}$ are sets of initial and accepting tiles, respectively, $\mathcal{H}, \mathcal{V}, \mathcal{M} \subseteq \mathcal{T} \times \mathcal{T}$ represent the horizontal, vertical and multi-tiling matching relations, respectively, and $m \in \mathbb{N}_+$ is a number written in unary. We write $|\mathcal{S}|$ for the number of symbols required to encode \mathcal{S} .

Fix $k \in \mathbb{N}_+$, and let $\text{grid}(k, m)$ be the two-dimensional grid $[\exp_2^k(m)] \times [\exp_2^k(m)]$, where we recall that $[u] := [0, u)$. Each $(h, v) \in \text{grid}(k, m)$ is said to be a position of the grid, comprised of a horizontal position h and a vertical position v . Let $d \in \mathbb{N}$ odd (so that the innermost quantifier of the AMTP(k, d) problem we are about to formalise is existential). A d -level \mathcal{S} -tiling for $\text{grid}(k, m)$ is a tuple $(f_1, f_2, f_3, \dots, f_d)$ such that for all $\ell \in [1, d]$:

maps: $f_\ell : \text{grid}(k, m) \rightarrow \mathcal{T}$ assigns a tile type to each position of $\text{grid}(k, m)$;

hori: $(f_\ell(i, j), f_\ell(i, j+1)) \in \mathcal{H}$ for every $j \in [\exp_2^k(m) - 1]$ and $i \in [\exp_2^k(m)]$;

vert: $(f_\ell(i, j), f_\ell(i+1, j)) \in \mathcal{V}$ for every $i \in [\exp_2^k(m) - 1]$ and $j \in [\exp_2^k(m)]$;

multi: if $\ell < d$, then $(f_\ell(i, j), f_{\ell+1}(i, j)) \in \mathcal{M}$ for every $i, j \in [\exp_2^k(m)]$; and

accept: $f_d(\exp_2^k(m) - 1, j) \in \mathcal{T}_{\text{acc}}$, for some $j \in [\exp_2^k(m)]$.

The initial row $I(f)$ of a map $f : \text{grid}(k, m) \rightarrow \mathcal{T}$ is the word $f(0, 0)f(0, 1) \dots f(0, \exp_2^k(m) - 1)$.

AMTP(k, d) : d -ALTERNATING MULTI-TILING PROBLEM OF ORDER k

INPUT: A multi-tiling system $\mathcal{S} = (\mathcal{T}, \mathcal{T}_0, \mathcal{T}_{\text{acc}}, \mathcal{H}, \mathcal{V}, \mathcal{M}, m)$.

QUESTION: Is it true that there is $w_1 \in (\mathcal{T}_0)^{\exp_2^k(m)}$ such that for all $w_2 \in (\mathcal{T}_0)^{\exp_2^k(m)}$ there is \dots there is $w_d \in (\mathcal{T}_0)^{\exp_2^k(m)}$ such that $\text{grid}(k, m)$ has a d -level tiling (f_1, \dots, f_d) where $I(f_\ell) = w_\ell$ for all $\ell \in [1, d]$?

► **Proposition 3** ([10]). *The problem AMTP(k, d) is complete for $\Sigma_d^{k\text{-EXP}}$. When d is given as part of the input instead of being fixed, the problem is STA($*$, $\exp_2^k(n^{O(1)})$, $O(n)$)-complete.*

The AMTP(k, d) problem arose from [4], in which the case $k = 1$ and d not fixed is shown to be STA($*$, $2^{n^{O(1)}}$, $O(n)$)-complete. See [25, Appendix E.7] or [23] for self-contained proofs.

In the remaining part of this section, we describe a reduction from AMTP(k, d) to HOSAT(k, d). First, we introduce a family of generalised Boolean formulae that allows us to compare the bit-strings $\text{enc}(f)$ and $\text{enc}(g)$ of two functions f and g in $\mathbb{B}_{k,n}$, with $n, k \in \mathbb{N}$. Subsequently, we define formulae to encode the tiling conditions (maps)–(accept) as well as the alternation on elements of $(\mathcal{T}_0)^{\exp_2^k(m)}$ required by the AMTP(k, d) problem. We remind the reader that, w.l.o.g., we define generalised Boolean formulae without constraining them to be in prenex normal form, though we need to keep track of the quantifier alternation for function symbols of type k .

Comparing bit-strings. We define formulae $\text{eq}_k(f, g)$, $\text{less}_k(f, g)$ and $\text{succ}_k(f, g)$ stating that $\text{enc}(f) = \text{enc}(g)$, $\text{enc}(f) < \text{enc}(g)$ and $\text{enc}(f) + 1 = \text{enc}(g)$, respectively, and $(h_1, \dots, h_n) <_k (h'_1, \dots, h'_n)$, with all h_i and h'_i in $\mathbb{B}_{k,n}$, which checks if the concatenation $\text{enc}(h_1) \dots \text{enc}(h_n)$ encodes a number smaller than $\text{enc}(h'_1) \dots \text{enc}(h'_n)$. The formula $\text{eq}_k(f, g)$ is defined as:

$$\text{eq}_k(f, g) := \forall h_1, \dots, h_n : \mathbb{B}_{k-1, n} . f(h_1, \dots, h_n) \leftrightarrow g(h_1, \dots, h_n).$$

The formulae $\text{less}_k(f, g)$ and $(h_1, \dots, h_n) <_k (h'_1, \dots, h'_n)$ have a mutually recursive definition: $\text{less}_{k+1}(f, g)$ is defined using $(h_1, \dots, h_n) <_k (h'_1, \dots, h'_n)$, which in turn requires $\text{less}_k(f, g)$. First, we define the base case $(h_1, \dots, h_n) <_0 (h'_1, \dots, h'_n)$, with each h_i and h'_i in $\mathbb{B} = \{0, 1\}$:

$$(h_1, \dots, h_n) <_0 (h'_1, \dots, h'_n) := \bigvee_{i=1}^n \neg h_i \wedge h'_i \wedge \bigwedge_{j=i+1}^n (h'_j \leftrightarrow h_j).$$

Intuitively, this formula forces $\text{enc}(h_1) \dots \text{enc}(h_n) < \text{enc}(h'_1) \dots \text{enc}(h'_n)$ by requiring that there is a bit $i \in [1, n]$ that is set to 0 in $\text{enc}(h_1) \dots \text{enc}(h_n)$, set to 1 in $\text{enc}(h'_1) \dots \text{enc}(h'_n)$, and the binary representations of $\text{enc}(h'_1) \dots \text{enc}(h'_n)$ and $\text{enc}(h_1) \dots \text{enc}(h_n)$ coincide on all bits $j > i$. This is indeed the characterisation of $<$ on binary numbers in least significant digit first encoding. The same idea is used to define the formula $\text{less}_k(f, g)$. Inductively, assume that the formula $(h_1, \dots, h_n) <_k (h'_1, \dots, h'_n)$ was defined. We use it to define $\text{less}_{k+1}(f, g)$:

$$\text{less}_{k+1}(f, g) := \exists \mathbf{h} : \mathbb{B}_{k,n} \cdot \neg f(\mathbf{h}) \wedge g(\mathbf{h}) \wedge \forall \mathbf{h}' : \mathbb{B}_{k,n} \cdot \mathbf{h} <_k \mathbf{h}' \rightarrow (f(\mathbf{h}') \leftrightarrow g(\mathbf{h}')),$$

where $\mathbf{h} = (h_1, \dots, h_n)$ and $\mathbf{h}' = (h'_1, \dots, h'_n)$.

To complete the definitions of $\text{less}_k(f, g)$ and $(h_1, \dots, h_n) <_k (h'_1, \dots, h'_n)$, what is left is to define $(h_1, \dots, h_n) <_{k+1} (h'_1, \dots, h'_n)$ using less_{k+1} . We need to respect the equivalence $(h_1, \dots, h_n) <_{k+1} (h'_1, \dots, h'_n) \equiv \bigvee_{i=1}^n \text{less}_{k+1}(h_i, h'_i) \wedge \bigwedge_{j=i+1}^n \text{eq}_{k+1}(h_j, h'_j)$. However, we cannot define $(h_1, \dots, h_n) <_{k+1} (h'_1, \dots, h'_n)$ as the formula in the right hand side of this equivalence, as this formula uses n occurrences of less_k and would thus lead to both less_k and $(h_1, \dots, h_n) <_k (h'_1, \dots, h'_n)$ being of size exponential in k . To solve this issue and obtain formulae of polynomial size, we rely on a variant of a trick used by Fisher and Rabin in [11], based on the equivalence $\Phi(a) \vee \Phi(b) \equiv \exists c : \Phi(c) \wedge (c = a \vee c = b)$:

$$(h_1, \dots, h_n) <_{k+1} (h'_1, \dots, h'_n) := \\ \exists a, b : \mathbb{B}_{k+1,n} \cdot \text{less}_{k+1}(a, b) \wedge \bigvee_{i=1}^n \text{eq}_{k+1}(a, h_i) \wedge \text{eq}_{k+1}(b, h'_i) \wedge \bigwedge_{j=i+1}^n \text{eq}_{k+1}(h_j, h'_j).$$

► **Lemma 4.** *Let $f, g \in \mathbb{B}_{k,n}$. Then, $\text{less}_k(f, g)$ iff $\text{enc}(f) < \text{enc}(g)$; and $|\text{less}_k(f, g)| \leq O(n^3 k)$.*

The definition of $\text{succ}_k(f, g)$ follows a similar characterisation of successor for binary numbers: we have $a + 1 = b$ whenever (1) there is a bit i that is set to 0 in a and to 1 in b , (2) the binary representations of a and b coincide on every bit $j > i$ (exactly as in the case of $<$) and moreover (3) every bit $j < i$ is set to 1 in a and it is set to 0 in b . For instance, in least significant digit first encoding, $(1111001)_2 + 1 = (0000101)_2$. We have:

$$\text{succ}_{k+1}(f, g) := \exists \mathbf{h} : \mathbb{B}_{k,n} \cdot \neg f(\mathbf{h}) \wedge g(\mathbf{h}) \wedge \forall \mathbf{h}' : \mathbb{B}_{k,n} \cdot \\ (\mathbf{h} <_k \mathbf{h}' \rightarrow (f(\mathbf{h}') \leftrightarrow g(\mathbf{h}'))) \wedge (\mathbf{h}' <_k \mathbf{h} \rightarrow f(\mathbf{h}') \wedge \neg g(\mathbf{h}')),$$

where $\mathbf{h} = (h_1, \dots, h_n)$ and $\mathbf{h}' = (h'_1, \dots, h'_n)$.

► **Lemma 5.** *For $f, g \in \mathbb{B}_{k,n}$, $\text{succ}_k(f, g)$ iff $\text{enc}(f) + 1 = \text{enc}(g)$; and $|\text{succ}_k(f, g)| \leq O(n^3 k)$.*

From AMTP(k, d) to HOSAT(k, d). We are ready to prove the lower bounds of Theorem 2. For $k = 1$, the Σ_d^{EXP} -hardness of HOSAT($1, d$) was already established by Lohrey in [19, Proposition 33] via a reduction from Turing machines. Therefore, we consider $k \geq 2$ (our proof can nonetheless be adapted to the case $k = 1$). Let $\mathcal{S} = (\mathcal{T}, \mathcal{T}_0, \mathcal{T}_{\text{acc}}, \mathcal{H}, \mathcal{V}, \mathcal{M}, m)$ be a multi-tiling system. We assume $\mathcal{T} = [1, r]$ for some $r \in \mathbb{N}_+$ (thus $\mathcal{H}, \mathcal{V}, \mathcal{M} \subseteq [1, r] \times [1, r]$).

Let $n := 2 + \#\mathcal{T} + m$. We write \perp_k for the function in $\mathbb{B}_{k,n}$ such that $\text{enc}(\perp_k) = 0$, i.e. \perp_k is the only solution f of the formula $\text{bot}_k(f) := \forall g_1, \dots, g_n : \mathbb{B}_{k-1,n} \cdot \neg f(g_1, \dots, g_n)$. Similarly, we write \top_k for the function in $\mathbb{B}_{k,n}$ with maximal encoding, that is the only

23:8 Higher-Order Quantified Boolean Satisfiability

solution f to the formula $\text{top}_k(f) := \forall g_1, \dots, g_n : \mathbb{B}_{k-1, n} \cdot f(g_1, \dots, g_n)$. The first step of the reduction consists of encoding the d functions of a d -level \mathcal{S} -tiling (f_1, \dots, f_d) . We encode each of these functions using a function $f \in \mathbb{B}_{k, n}$ satisfying the following two properties:

- 1: If $f(h, v, t_1, \dots, t_r, u_1, \dots, u_m) = 1$ then $\text{enc}(h)$ and $\text{enc}(v)$ belong to $[\exp_2^k(m)]$, each t_i belongs to $\{\perp_{k-1}, \top_{k-1}\}$ and every u_j is \perp_{k-1} .
 - 2: Given $h, v \in \mathbb{B}_{k-1, n}$, if $\text{enc}(h), \text{enc}(v) \in [\exp_2^k(m)]$ then there is exactly one tuple $\mathbf{t} = (t_1, \dots, t_r)$ such that $f(h, v, \mathbf{t}, u_1, \dots, u_m) = 1$; and exactly one among t_1, \dots, t_r is \top_{k-1} .
- Here, the inputs h and v are used to represent horizontal and vertical positions in the grid, the inputs t_1, \dots, t_r are used to encode the tiles, and the inputs u_1, \dots, u_m are only required to make sure we have enough inputs to encode the fact that h and v belong to $[\exp_2^k(m)]$ (see the formula ok_1 below). Together, Properties (1) and (2) characterise the condition (maps) of the \mathcal{S} -tiling (we add the other tiling conditions later). To capture Property 1 above, notice first that h and v should be taken from a space of exactly $\exp_2^k(m)$ functions. As $\#\mathbb{B}_{k-1, n} \geq \exp_2^k(m)$, this obliges us to introduce a formula $\text{ok}_k(g)$ characterising the fact that a function $g \in \mathbb{B}_{k, n}$ is such that $\text{enc}(g) \in [\exp_2^{k+1}(m)]$. We have

$$\begin{aligned} \text{ok}_1(g) &:= \forall h_1, \dots, h_n : \mathbb{B} \cdot g(h_1, \dots, h_n) \rightarrow \bigwedge_{i=m+1}^n \neg h_i \\ \text{ok}_{k+1}(g) &:= \forall h_1, \dots, h_n : \mathbb{B}_{k, n} \cdot g(h_1, \dots, h_n) \rightarrow (\text{ok}_k(h_1) \wedge \bigwedge_{i=2}^n \text{bot}_k(h_i)). \end{aligned}$$

Intuitively, for $k \geq 1$ the formula $\text{ok}_k(g)$ holds whenever for all inputs (h_1, \dots, h_n) and $j := \text{enc}(h_1) \cdots \text{enc}(h_n)$, if the j -th bit of $\text{enc}(g)$ is set to 1 then $j \in [\exp_2^k(m)]$. This means that $\text{enc}(g)$ corresponds to a binary number with $\exp_2^k(m)$ bits, i.e. $\text{enc}(g) \in [\exp_2^{k+1}(m)]$.

In all the formulae below, we let $\mathbf{t} = (t_1, \dots, t_r)$, $\mathbf{u} = (u_1, \dots, u_m)$ and $\mathbf{t}' = (t'_1, \dots, t'_r)$. We define the formula $\text{mapsOne}_k(f)$ stating that $f \in \mathbb{B}_{k, n}$ satisfies Property 1:

$$\begin{aligned} \text{mapsOne}_k(f) &:= \forall h, v, \mathbf{t}, \mathbf{u} : \mathbb{B}_{k-1, n} \cdot f(h, v, \mathbf{t}, \mathbf{u}) \rightarrow \text{ok}_{k-1}(h) \wedge \text{ok}_{k-1}(v) \wedge \\ &\quad \bigwedge_{i=1}^r (\text{top}_{k-1}(t_i) \vee \text{bot}_{k-1}(t_i)) \wedge \bigwedge_{j=1}^m \text{bot}_{k-1}(u_j). \end{aligned}$$

Suppose that $f \in \mathbb{B}_{k, n}$ satisfies $\text{mapsOne}_k(f)$. In a similar fashion, one defines a formula $\text{mapsTwo}_k(f)$ stating that f satisfies Property 2 of the encoding:

$$\begin{aligned} \text{mapsTwo}_k(f) &:= \forall h, v : \mathbb{B}_{k-1, n} \cdot \text{ok}_{k-1}(h) \wedge \text{ok}_{k-1}(v) \rightarrow \exists \mathbf{t}, \mathbf{u} : \mathbb{B}_{k-1, n} \cdot f(h, v, \mathbf{t}, \mathbf{u}) \wedge \\ &\quad \bigvee_{i=1}^r (\text{top}_{k-1}(t_i) \wedge \bigwedge_{\substack{j=1 \\ j \neq i}}^r \text{bot}_{k-1}(t_j)) \wedge \forall \mathbf{t}' : \mathbb{B}_{k-1, n} \cdot f(h, v, \mathbf{t}', \mathbf{u}) \rightarrow \bigwedge_{i=1}^r \text{eq}_{k-1}(t_i, t'_i). \end{aligned}$$

We define $\text{maps}_k(f) := \text{mapsOne}_k(f) \wedge \text{mapsTwo}_k(f)$, and move to the remaining tiling conditions. The conditions (hori) and (vert) can be defined with the help of the formula succ_k . Below, we present the definition of the formula $\text{hori}_k(f)$ that encodes the condition (hori):

$$\begin{aligned} \text{hori}_k(f) &:= \forall h, v, v' : \mathbb{B}_{k-1, n} \cdot \text{ok}_{k-1}(h) \wedge \text{ok}_{k-1}(v) \wedge \text{ok}_{k-1}(v') \wedge \text{succ}_{k-1}(v, v') \\ &\quad \rightarrow \bigvee_{(i, j) \in \mathcal{H}} f(h, v, \underline{i}) \wedge f(h, v', \underline{j}), \end{aligned}$$

where $f(h, v, \underline{i})$ is a shortcut for $\exists \mathbf{t}, \mathbf{u} : \mathbb{B}_{k-1, n} \cdot \text{top}_{k-1}(t_i) \wedge f(h, v, \mathbf{t}, \mathbf{u})$, i.e. a formula stating that the tile $i \in \mathcal{T}$ is assigned to the position $(\text{enc}(h), \text{enc}(v))$ of the grid, under the hypothesis that f satisfies $\text{maps}_k(f)$. The definition of the formula $\text{vert}_k(f)$ encoding the condition (vert) is defined analogously. For the condition (multi), let f and g be two functions of $\mathbb{B}_{k, n}$ satisfying maps_k . We define:

$$\text{multi}_k(f, g) := \forall h, v : \mathbb{B}_{k-1, n} \cdot \text{ok}_{k-1}(h) \wedge \text{ok}_{k-1}(v) \rightarrow \bigvee_{(i, j) \in \mathcal{M}} f(h, v, \underline{i}) \wedge g(h, v, \underline{j}).$$

Lastly, we define the formula $\text{acc}_k(f)$ corresponding to the condition (accept):

$$\begin{aligned} \text{acc}_k(f) := & \exists h, v: \mathbb{B}_{k-1, n} \cdot \text{ok}_{k-1}(h) \wedge \text{ok}_{k-1}(v) \wedge \left(\bigvee_{i \in \mathcal{T}_{\text{acc}}} f(h, v, i) \right) \wedge \\ & \forall h': \mathbb{B}_{k-1, n} \cdot \text{less}_{k-1}(h, h') \rightarrow \neg \text{ok}_{k-1}(h'). \end{aligned}$$

Here, the subformula $\forall h': \mathbb{B}_{k-1, n} \cdot \text{less}_{k-1}(h, h') \rightarrow \neg \text{ok}_{k-1}(h')$ forces $\text{enc}(h) = \exp_2^k(m) - 1$.

To complete the reduction, what is left is to encode the quantifier prefix “ $\exists w_1 \in (\mathcal{T}_0)^{\exp_2^k(m)} \forall w_2 \in (\mathcal{T}_0)^{\exp_2^k(m)} \dots \exists w_d \in (\mathcal{T}_0)^{\exp_2^k(m)}$ ” of the AMTP(k, d) problem. To this end, we introduce two formulae $\text{row}_k(f)$ and $\text{rowCp}_k(f, g)$. The former states that $f \in \mathbb{B}_{k, n}$ satisfies maps_k , and whenever $f(h, v, \mathbf{t}, \mathbf{u})$ holds the only t_i equal to \top_{k-1} is such that $i \in \mathcal{T}_0$:

$$\text{row}_k(f) := \text{maps}_k(f) \wedge \forall h, v, \mathbf{t}, \mathbf{u}: \mathbb{B}_{k-1, n} \cdot f(h, v, \mathbf{t}, \mathbf{u}) \rightarrow \bigvee_{i \in \mathcal{T}_0} \text{top}_k(t_i).$$

We use this formula to quantify on the initial row $I(f)$ of a \mathcal{S} -tiling function \mathfrak{f} , forgetting about the information stored by f elsewhere. This is done thanks to the formula $\text{rowCp}_k(f, g)$, which given $f, g \in \mathbb{B}_{k, n}$ satisfying maps_k , states that f and g agree on the first row:

$$\text{rowCp}_k(f, g) := \exists h: \mathbb{B}_{k-1, n} \cdot \text{bot}_{k-1}(h) \wedge \forall v, \mathbf{t}, \mathbf{u}: \mathbb{B}_{k-1, n} \cdot f(h, v, \mathbf{t}, \mathbf{u}) \leftrightarrow g(h, v, \mathbf{t}, \mathbf{u}),$$

The quantifier prefix of AMTP(k, d) is encoded in HOSAT(k, d) by first quantifying over functions $w_1, \dots, w_d \in \mathbb{B}_{k, n}$ satisfying row_k , appropriately alternating between existential and universal quantification, and then existentially quantifying over functions $f_1, \dots, f_d \in \mathbb{B}_{k, n}$ that encode the \mathcal{S} -tiling functions $(\mathfrak{f}_1, \dots, \mathfrak{f}_d)$. The formula $\text{rowCp}_k(f_i, w_i)$ is used to “copy” the first row of w_i in f_i . This leads to the formula $\text{amtp}_k(\mathcal{S})$:

$$\begin{aligned} & \exists w_1: \mathbb{B}_{k, n} \cdot \text{row}_k(w_1) \wedge \\ & \quad \forall w_2: \mathbb{B}_{k, n} \cdot \text{row}_k(w_2) \rightarrow \\ & \quad \dots \\ & \quad \exists w_d: \mathbb{B}_{k, n} \cdot \text{row}_k(w_d) \wedge \\ & \quad \quad \exists f_1, \dots, f_d: \mathbb{B}_{k, n} \cdot \bigwedge_{i=1}^d (\text{maps}_k(f_i) \wedge \text{rowCp}_k(f_i, w_i) \wedge \text{horik}(f_i) \wedge \text{vert}_k(f_i)) \\ & \quad \quad \wedge \bigwedge_{i=1}^{d-1} \text{multi}_k(f_i, f_{i+1}) \wedge \text{acc}_k(f_d). \end{aligned}$$

► **Proposition 6.** *Let $d \in \mathbb{N}_+$ and $k \geq 2$. The formula $\text{amtp}_k(\mathcal{S})$ is valid if and only if AMTP(k, d) accepts on input \mathcal{S} . Moreover, the formula $\text{amtp}_k(\mathcal{S})$ has size $O(d \cdot k \cdot |\mathcal{S}|^4)$.*

Notice that $\text{amtp}_k(\mathcal{S})$ is a generalised Boolean formula of order k and alternation depth d , since bringing it into prenex normal form yields a formula of the form $\exists w_1: \mathbb{B}_{k, n} \forall w_2: \mathbb{B}_{k, n} \dots \exists w_d, f_1, \dots, f_d: \mathbb{B}_{k, n} \cdot \Phi$, where Φ is a generalised Boolean formula of order $k - 1$ and size in $O(d \cdot k \cdot |\mathcal{S}|^4)$. By Proposition 3 (first part), this completes the proof of Theorem 2(I). Theorem 2(II) is proven analogously, by simply treating d as part of the input instead of being fixed. The upper bound follows the same strategy as the case of HOSAT(k, d), and the lower bound follows from Proposition 6 together with the second part of Proposition 3.

5 Weak Presburger arithmetic is as hard as Presburger arithmetic

In this section, we illustrate the usefulness of the higher order satisfiability problem by employing it to derive a $\text{STA}(*, \exp_2^2(n^{O(1)}), O(n))$ lower bound for the weak fragment of Presburger arithmetic (*Weak PA*), hence showing that this logic matches the complexity of (standard) Presburger arithmetic. Weak PA is the first-order theory of the structure $\langle \mathbb{Z}, (c)_{c \in \mathbb{Z}}, +, = \rangle$, where $(c)_{c \in \mathbb{Z}}$ are constant symbols interpreted as their homographic integer,

23:10 Higher-Order Quantified Boolean Satisfiability

the binary function symbol $+$ is interpreted as addition on \mathbb{Z} and the binary relation $=$ is interpreted as equality on \mathbb{Z} . Subsequently, linear Terms t, t', \dots are of the form $a_1x_1 + \dots + a_dx_d + c$, where d is arbitrary, $a_1, \dots, a_d, c \in \mathbb{Z}$, and x_1, \dots, x_d are first-order variables interpreted over \mathbb{Z} . Formulae of Weak PA close equalities between linear terms $t = t'$ under Boolean connectives \neg, \wedge, \vee , etc., and first-order quantifiers $\exists x$ and $\forall x$. For example, given $m \in \mathbb{N}_+$ and linear terms t, t' , the *modulo constraint* $t \equiv_m t'$ stating that t is congruent to t' modulo m is characterised by the Weak PA formula $\exists x. t - t' = x \cdot m$, where x is a variable not appearing in t or t' .

Surprisingly, our lower bound holds for the following two fragments of Weak PA:

- The *positive fragment* that forbids negation, allowing formulae from the grammar

$$\Phi := a_1x_1 + \dots + a_dx_d = c \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \exists x. \Phi \mid \forall x. \Phi.$$

- The *Horn fragment*, in which formulae are of the form $\exists \mathbf{x}_1 \forall \mathbf{x}_2 \dots \exists \mathbf{x}_m. \bigwedge_{i \in I} (\Phi_i \rightarrow \Psi_i)$, where $\mathbf{x}_1, \dots, \mathbf{x}_m$ are vectors of variables, and each Φ_i and Ψ_i is a system of equalities, that is a conjunction of equalities between linear terms.

► **Theorem 7.** *The positive and Horn fragments of Weak PA are $\text{STA}(*, 2^{2^{n^{O(1)}}}, O(n))$ -hard.*

To prove this theorem we provide a reduction from HOSAT(2, *) to the validity problem for the positive fragment of Weak PA, and a translation from Weak PA to its Horn fragment. The reduction defines arithmetic with multiplication on doubly exponential finite segments of integers, such as $[2^{2^n}]$, and, in a more restricted way, one exponential higher. Here we strengthen Berman's lower bound argument for standard PA [3] (see also [18, Lecture 22]), which relies on the order predicate throughout.

Encoding second-order Boolean functions in Weak PA. Let $n \in \mathbb{N}_+$ be encoded in unary. In a nutshell, the main difficulty in the reduction from HOSAT(2, *) is to understand how to encode the functions in $\mathbb{B}_{1,n} := \{0, 1\}^n \rightarrow \{0, 1\}$ and $\mathbb{B}_{2,n} := (\mathbb{B}_{1,n})^n \rightarrow \{0, 1\}$ using integer numbers, as well as translating the function applications $f(g_1, \dots, g_n)$. For the set $\mathbb{B}_{1,n}$, we rely on the encoding $\text{enc}(\cdot)$ defined in Section 3 that maps every function $f \in \mathbb{B}_{1,n}$ into the number $\text{enc}(f) \in [2^{2^n}]$ written in binary. To encode functions in $\mathbb{B}_{2,n}$, we borrow ideas from [13, 14]. We say that $z \in \mathbb{Z}$ *encodes some function in* $\mathbb{B}_{2,n}$ if and only if for every $i \in [2^{2^n}]$ and all prime numbers $p, q \in [i^3, (i+1)^3]$, $z \equiv_p 0$ or $z \equiv_p 1$, and $z \equiv_p 0$ if and only if $z \equiv_q 0$. Furthermore, we say that $z \in \mathbb{Z}$ (*precisely*) *encodes the function* $f \in \mathbb{B}_{2,n}$ if and only if z encodes some function in $\mathbb{B}_{2,n}$ and moreover for every $\mathbf{g} = (g_0, \dots, g_{n-1}) \in (\mathbb{B}_{1,n})^n$ and $b \in \{0, 1\}$,

$$f(\mathbf{g}) = b \text{ if and only if } z \equiv_p b \text{ for a prime } p \in [\beta(\mathbf{g})^3, (\beta(\mathbf{g}) + 1)^3],$$

where $\beta : (\mathbb{B}_{1,n})^n \rightarrow [0, 2^{2^{2^n}}]$ is the bijection $\beta(g_0, \dots, g_{n-1}) := \sum_{j=0}^{n-1} \text{enc}(g_j) \cdot 2^{j \cdot 2^n}$. The fact that any function in $\mathbb{B}_{2,n}$ is encoded by some $z \in \mathbb{Z}$ follows from the Chinese remainder theorem together with Ingham's theorem [16], a theorem ensuring that for i sufficiently large, $[i^3, (i+1)^3]$ contains at least one prime. As in, e.g., [13, 14], to simplify the presentation we apply Ingham's theorem as if it was known to be true for all $i \in \mathbb{N}$. An alternative would be to use an analogous result on primes between fixed powers [9] that holds for all i , or to add constant offsets throughout.

From HOSAT(2, *) to the positive fragment of Weak PA. We formalise the translation τ that, given a formula from HOSAT(2, *) returns an equivalent formula in the positive fragment of Weak PA. We divide the translation into three parts, depending on whether we are dealing

formula	semantics	formula	semantics
$\varphi_n(x, z)$	$x = 2^{2^n} \cdot z$	$\text{intdiv}_n(x, y, q, r)$	$x = q \cdot y + r, y \in [2^{2^n}], r \in [y]$
$\text{mult}_n(x, y, z)$	$x = y \cdot z, z \in [2^{2^n}]$	$\text{quot}_n(x, y, q)$	$\exists r \in [y] \text{ s.t. } \text{intdiv}_n(x, y, q, r)$
$I_n(x)$	$z \in [2^{2^n}]$	$\text{rem}_n(x, y, r)$	$\exists q \in \mathbb{Z} \text{ s.t. } \text{intdiv}_n(x, y, q, r)$
$\text{power}_n(y, j)$	$y = 2^j \text{ and } j \in [2^n]$	$(\text{not})\text{prime}_n(p)$	$p \in [2^{2^n}] \text{ is } (\text{not}) \text{ prime}$
$\psi_{k,n}(x)$	$x = 2^{k \cdot 2^n}$	$x =_n y^3$	$y \in [2^{2^{2(n+1)}}] \text{ and } x = y^3$

■ **Figure 1** Auxiliary formulae required to reduce HOSAT(2, *) to positive Weak PA.

with a generalised Boolean formula of order 0, 1 or 2. The translation τ is homomorphic for binary Boolean connectives: $\tau(\Phi \wedge \Psi) := \tau(\Phi) \wedge \tau(\Psi)$ and $\tau(\Phi \vee \Psi) := \tau(\Phi) \vee \tau(\Psi)$. Without loss of generality we assume that negations occurring in the quantifier-free part of generalised Boolean formulae only appear in front of function applications (recall that we see Boolean values as 0-ary functions). A (*function application*) *literal* is either a function application $f(g_1, \dots, g_\ell)$ or its negation $\neg f(g_1, \dots, g_\ell)$.

Formulae of order 0. We translate $\exists f: \mathbb{B}, \forall f: \mathbb{B}$ and literals f and $\neg f$, with f of type \mathbb{B} . Let $\mathbb{B}(x) := x = 0 \vee x = 1$, i.e. the formula stating $x \in \mathbb{B}$. Clearly, $\tau(f) := f = 1$ and $\tau(\neg f) := f = 0$. The quantifiers $\exists f: \mathbb{B}$ and $\forall f: \mathbb{B}$ are translated as follows:

$$\tau(\exists f: \mathbb{B} \Phi) := \exists f. \mathbb{B}(f) \wedge \tau(\Phi), \quad \tau(\forall f: \mathbb{B} \Phi) := \forall f' \exists f. f \equiv_2 f' \wedge \mathbb{B}(f) \wedge \tau(\Phi).$$

A few words on the translation of $\forall f: \mathbb{B}$. We cannot translate the universal quantifier $\forall f: \mathbb{B} \Phi$ as $\forall f. \mathbb{B}(f) \rightarrow \tau(\Phi)$, i.e. relying on the \exists - \forall duality, as $\mathbb{B}(f) \rightarrow \tau(\Phi)$ is not in the positive fragment of Weak PA. Our definition circumvents this problem by relying on the equivalence $\forall x. x \in [0, \ell] \rightarrow \Phi(x, z) \equiv \forall x' \exists x. x' \equiv_{\ell+1} x \wedge x \in [0, \ell] \wedge \Phi(x, z)$.

Formulae of order 1. We treat quantifiers $\exists f: \mathbb{B}_{1,n}, \forall f: \mathbb{B}_{1,n}$ and literals $f(g_1, \dots, g_n)$ and $\neg f(g_1, \dots, g_n)$, where f is of type $\mathbb{B}_{1,n}$ and each g_i is of type \mathbb{B} . To achieve this we rely on the auxiliary formulae formally specified in Figure 1 and defined below ($\psi_{k,n}(x)$, $\text{notprime}_n(p)$ and $x =_n y^3$ are defined later as only used for formulae of order 2). The formulae φ_n , mult_n , I_n and intdiv_n are an adaptation of the homonymous formulae provided by Kozen in [18, Lecture 22] in the context of linear real arithmetic. For instance, the formula $\varphi_n(x, z)$ is inductively defined in [18, p. 147] as follows:

$$\varphi_0(x, z) := x = 2z, \quad \varphi_{n+1}(x, z) := \exists y \forall a, b. (a = x \wedge b = y) \vee (a = y \wedge b = z) \rightarrow \varphi_n(a, b).$$

The inductive case of $\varphi_{n+1}(x, z)$ is defined relying on the trick of Fisher and Rabin [11] we already encountered in Section 3, used here to obtain a linear size formula equivalent to $\exists y : \varphi_n(x, y) \wedge \varphi_n(y, z)$. As it stands $\varphi_{n+1}(x, z)$ is not in the positive fragment of Weak PA. We rely on modulo constraints to resolve this issue, redefining $\varphi_{n+1}(x, z)$ as follows:

$$\varphi_{n+1}(x, z) := \exists y \forall i \exists a, b. ((i \equiv_2 0 \wedge a = x \wedge b = y) \vee (i \equiv_2 1 \wedge a = x \wedge b = y)) \wedge \varphi_n(a, b).$$

The definitions of $\text{mult}_n(x, y, z)$, $I_n(x)$ and $\text{intdiv}_n(x, y, q, r)$ require similar adaptations w.r.t. [18, p. 148f], which are omitted due to space constraints. The formulae quot_n and rem_n are simple shortcuts of intdiv_n , e.g., $\text{quot}_n(x, y, q) := \exists r \text{intdiv}_n(x, y, q, r)$.

Finally, $\text{power}_n(y, j)$ is intuitively defined by bit-blasting j into n bits j_0, \dots, j_{n-1} , so that $j = \sum_{i=0}^{n-1} j_i \cdot 2^i$, and forcing $y = 2^j = \prod \{\exp_2^2(i) : i \in [n] \text{ and } j_i = 1\}$ to hold:

$$\text{power}_n(y, j) := \exists (j_0, y_0, z_0), \dots, (j_{n-1}, y_{n-1}, z_{n-1}). j = \sum_{i=0}^{n-1} j_i \cdot 2^i \wedge z_0 = y_0 \wedge y = z_{n-1} \\ \wedge \bigwedge_{i=1}^{n-1} (\text{mult}_n(z_i, z_{i-1}, y_i) \wedge ((j_i = 0 \wedge y_i = 1) \vee (j_i = 1 \wedge \varphi_i(y_i, 1))))).$$

23:12 Higher-Order Quantified Boolean Satisfiability

The subformula $z_0 = y_0 \wedge y = z_{n-1} \wedge \bigwedge_{i=1}^{n-1} \text{mult}_n(z_i, z_{i-1}, y_i)$ computes $y = \prod_{i=0}^{n-1} y_i$. We are ready to translate the generalised Boolean formula of order 1. Recall that we encode a function f in $\mathbb{B}_{1,n}$ with the number $\text{enc}(f) \in [2^{2^n}]$, and we have $f(g_0, \dots, g_{n-1}) = 1$ if and only if the j th bit of $\text{enc}(f)$ is 1, where $j = \sum_{i=0}^{n-1} g_i \cdot 2^i$. With this in mind, the case for existential quantifiers closely follows the definition for formulae of order 0: $\tau(\exists f : \mathbb{B}_{1,n} \Phi) := \exists f. I_n(f) \wedge \tau(\Phi)$. The case of universal quantifiers is more involved: whereas for the case of formulae of order 0 we resorted to reasoning modulo 2, we now reason modulo 2^{2^n} . We can bind 2^{2^n} to a variable ℓ with the formula $\varphi_n(\ell, 1)$. For $m \in \mathbb{Z}$ and $r \in [2^{2^n}]$, we can then check if $m \equiv_\ell r$ holds using the formula $\text{rem}_{n+1}(m, \ell, r)$. Note that we use rem_{n+1} instead of rem_n , as ℓ belongs to the set $[2^{2^{n+1}}] \setminus [2^{2^n}]$. Here is the translation:

$$\tau(\forall f : \mathbb{B}_{1,n} \Phi) := \forall f' \exists f, \ell. \varphi_n(\ell, 1) \wedge \text{rem}_{n+1}(f', \ell, f) \wedge \tau(\Phi).$$

For the function application literal $f(g_0, \dots, g_{n-1})$, where each g_i is mapped through τ into a homonymous Boolean variable according to the translation of formulae of order 0, we consider the number $j = \sum_{i=0}^{n-1} g_i \cdot 2^i$ and check that the j th bit of $\text{enc}(f)$ is set to 1 by verifying that the quotient of the division $f/2^j$ is odd. As a formula:

$$\tau(f(g_0, \dots, g_{n-1})) := \exists j, y, q. j = \sum_{i=0}^{n-1} g_i \cdot 2^i \wedge \text{power}_n(y, j) \wedge \text{quot}_n(f, y, q) \wedge q \equiv_2 1.$$

We treat the literal $\neg f(g_0, \dots, g_{n-1})$ in a similar way, by checking whether $f/2^j$ is even. So, $\tau(\neg f(g_0, \dots, g_{n-1}))$ is obtained from $\tau(f(g_0, \dots, g_{n-1}))$ by replacing $q \equiv_2 1$ with $q \equiv_2 0$.

Formulae of order 2. To complete the definition of τ , we now show how to handle quantifiers $\exists f : \mathbb{B}_{2,n}$ and $\forall f : \mathbb{B}_{2,n}$, and function application literals $f(g_1, \dots, g_n)$ and $\neg f(g_1, \dots, g_n)$, where f is of type $\mathbb{B}_{2,n}$ and every g_i is of type $\mathbb{B}_{1,n}$. As explained at the beginning of the section, functions in $\mathbb{B}_{2,n}$ are encoded as integers $z \in \mathbb{Z}$ having the property that for every $i \in [2^{2^n}]$ and all prime numbers $p, q \in [i^3, (i+1)^3]$, $z \equiv_p 0$ or $z \equiv_p 1$, and $z \equiv_q 0$ if and only if $z \equiv_q 0$. Below, we aim at defining the formula $\text{enc}_n^2(z)$ stating that z encodes some function in $\mathbb{B}_{2,n}$. We start by defining the formula $\text{notprime}_n(p)$ from Figure 1:

$$\text{notprime}_n(p) := I_n(p) \wedge \exists d. I_n(d) \wedge I_n(d-2) \wedge I_n(p-d-1) \wedge \text{rem}_n(p, d, 0).$$

Informally, $\text{notprime}_n(p)$ states that p is not a prime number by finding a divisor $d \in [2, p-1]$. Notice that if we assume $p, d \in [2^{2^n}]$ then $I_n(d-2) \equiv d \geq 2$ and $I_n(p-d-1) \equiv d < p$.

Two further comments on the definition of encoding for functions in $\mathbb{B}_{2,n}$ given above: first, observe that this definition requires the construction of numbers $(i+1)^3$ with $i \in [2^{2^n}]$. Since $(2^{2^n} + 1)^3 < 2^{2^{2(n+1)}}$, all these numbers satisfy the formula $I_{2(n+1)}(x)$. This explains the contribution of $\text{notprime}_{2(n+1)}$ and similar formulae in the forthcoming definition of $\text{enc}_n^2(z)$. Second, the encoding requires to iterate over all $i \in [2^{2^n}]$. As in the definition of τ for the case $\forall f : \mathbb{B}_{1,n}$, this is done by binding 2^{2^n} to a variable ℓ and then considering all numbers in $[\ell]$ by relying on the formula $\text{rem}_{2(n+1)}$. To characterise 2^{2^n} we use the following formula that, given $k \in \mathbb{N}_+$ in unary, is satisfied whenever $x = 2^{k2^n}$:

$$\psi_{k,n}(x) := \exists z_1, \dots, z_k. z_k = 1 \wedge \varphi_k(x, z_1) \wedge \bigwedge_{i=1}^{k-1} \varphi_n(z_i, z_{i+1}).$$

We define $x =_n y^3 := \exists z. \text{mult}_{2(n+1)}(z, y, y) \wedge \text{mult}_{2(n+1)}(x, z, y)$ and the formula $\text{enc}_n^2(z)$:

$$\begin{aligned} \text{enc}_n^2(z) := & \forall i' \exists i, \ell, a, b. \psi_{n,n}(\ell) \wedge \text{rem}_{2(n+1)}(i', \ell, i) \wedge a =_n i^3 \wedge b =_n (i+1)^3 \wedge \\ & \forall p', q' \exists \ell', p, q. \varphi_{2(n+1)}(\ell', 1) \wedge \text{rem}_{2n+3}(p', \ell', p) \wedge \text{rem}_{2n+3}(q', \ell', q) \wedge \\ & \left(\bigvee_{r \in \{p, q\}} (I_{2(n+1)}(a-r-1) \vee I_{2(n+1)}(r-b) \vee \text{notprime}_{2(n+1)}(r)) \right. \\ & \left. \vee \bigvee_{s \in \{0,1\}} (\text{rem}_{2(n+1)}(z, p, s) \wedge \text{rem}_{2(n+1)}(z, q, s)) \right). \end{aligned}$$

Let us dissect this formula line by line. The first line iterates through all $i \in [\ell] = [2^{n2^n}]$, binding a and b to i^3 and $(i+1)^3$ respectively. The second line iterates through all $p, q \in [\ell'] = [2^{2^{2(n+1)}}]$. Following the definition of our encoding of functions in $\mathbb{B}_{2,n}$, we check that

- one among p and q lies outside $[a, b]$ or is not prime (third line of the formula), or
- $z \equiv_p 0$ or $z \equiv_p 1$, and $z \equiv_p 0$ if and only if $z \equiv_q 0$ (fourth line of the formula).

► **Lemma 8.** *A number $z \in \mathbb{Z}$ satisfies $\text{enc}_n^2(z)$ iff z encodes some function in $\mathbb{B}_{2,n}$.*

We can now define τ for \exists quantifiers: $\tau(\exists f : \mathbb{B}_{2,n} \Phi) := \exists f : \text{enc}_n^2(f) \wedge \tau(\Phi)$. For universal quantifiers we reason dually and define a positive Weak PA formula $\text{notenc}_n^2(z)$ stating that z does not encode any function in $\mathbb{B}_{2,n}$. Defining this formula requires a positive Weak PA formula to test for the primality of $p \in [2^{2^n}]$:

$$\text{prime}_n(p) := I_n(p) \wedge \forall z \exists d, r. \text{rem}_n(z, p, d) \wedge (d = 0 \vee d = 1 \vee (\text{rem}_n(p, d, r) \wedge I_n(r - 1))).$$

Afterwards, $\text{notenc}_n^2(z)$ is defined by slightly revisiting $\text{enc}_n^2(z)$:

$$\begin{aligned} \text{notenc}_n^2(z) := & \exists i, \ell, a, b, p, q. \psi_{n,n}(\ell) \wedge I_{2(n+1)}(\ell - i - 1) \wedge a =_n i^3 \wedge b =_n (i + 1)^3 \wedge \\ & \bigwedge_{r \in \{p, q\}} (I_{2(n+1)}(r - a) \wedge I_{2(n+1)}(b - r - 1) \wedge \text{prime}_{2(n+1)}(r)) \wedge \\ & \exists s, t. \text{rem}_{2(n+1)}(z, p, s) \wedge \text{rem}_{2(n+1)}(z, p, t) \wedge (I_{2(n+1)}(s - 2) \vee s \equiv_2 t + 1). \end{aligned}$$

We define the translation for universal quantifiers as $\tau(\forall f : \mathbb{B}_{2,n} \Phi) := \forall f : \text{notenc}_n^2(f) \vee \tau(\Phi)$.

Let $\mathbf{g} = (g_0, \dots, g_{n-1})$. What is left is to treat the literals $f(\mathbf{g})$ and $\neg f(\mathbf{g})$. Once more, recall that every g_i is a function in $\mathbb{B}_{1,n}$, and thus it is translated through τ into a homonymous integer variable that is constrained to be in $[2^{2^n}]$. Suppose that z encodes the function f . To check whether $f(\mathbf{g})$ holds (resp. does not hold), we must check whether $z \equiv_p 1$ (resp. $z \equiv_p 0$) for some prime number p in the interval $[i^3, (i+1)^3)$ with $i := \sum_{j=0}^{n-1} g_j \cdot 2^{j \cdot 2^n}$. Formally, given $d \in \{0, 1\}$ and $i \in [2^{n2^n}]$, we define the macro $z[i] =_n d$ to check this property:

$$\begin{aligned} z[i] =_n d := & \exists a, b, p. a =_n i^3 \wedge b =_n (i + 1)^3 \wedge \text{prime}_{2(n+1)}(p) \wedge \\ & I_{2(n+1)}(p - a) \wedge I_{2(n+1)}(b - p - 1) \wedge \text{rem}_{2(n+1)}(z, p, d). \end{aligned}$$

It then suffices to compute i from $\mathbf{g} = (g_0, \dots, g_{n-1})$, with the following formula

$$\gamma_n(i, \mathbf{g}) := \exists x_0, y_0, \dots, x_{n-1}, y_{n-1}. i = \sum_{j=0}^{n-1} x_j \wedge \bigwedge_{j=0}^{n-1} (\psi_{j,n}(y_j) \wedge \text{mult}_{2(n+1)}(x_j, y_j, g_j)),$$

leading to the following translations for the literals $f(\mathbf{g})$ and $\neg f(\mathbf{g})$:

$$\tau(f(\mathbf{g})) := \exists i. \gamma_n(i, \mathbf{g}) \wedge f[i] =_n 1, \quad \tau(\neg f(\mathbf{g})) := \exists i. \gamma_n(i, \mathbf{g}) \wedge f[i] =_n 0.$$

The correctness of the translation τ is provided by the following proposition, shown by structural induction on the generalised Boolean formula Φ .

► **Proposition 9.** *A generalised Boolean formula Φ of order 2 is valid if and only if so is $\tau(\Phi)$. The size of $\tau(\Phi)$ is polynomial in the size of Φ .*

Horn Weak PA = Weak PA. To complete the proof of Theorem 7, we need to show that Weak PA reduces to its Horn fragment. Briefly, this is done with standard formula manipulations and by relying on the equivalences below (notice the similarities to the trick used to define φ_n):

$$\Phi \wedge \Psi \equiv \forall x. (x \equiv_2 0 \rightarrow \Phi) \wedge (x \equiv_2 1 \rightarrow \Psi), \quad \Phi \vee \Psi \equiv \exists x. (x \equiv_2 0 \rightarrow \Phi) \wedge (x \equiv_2 1 \rightarrow \Psi);$$

where the variable x above does not occur in Φ nor in Ψ .

References

- 1 László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Comput. Complex.*, 1:3–40, 1991. doi:10.1007/BF01200056.
- 2 José L. Balcázar, Antoni Lozano, and Jacobo Torán. *The Complexity of Algorithmic Problems on Succinct Instances*, pages 351–377. 1992. doi:10.1007/978-1-4615-3422-8_30.
- 3 Leonard Berman. The complexity of logical theories. *Theor. Comput. Sci.*, 11(1):71–77, 1980. doi:10.1016/0304-3975(80)90037-7.
- 4 Laura Bozzelli, Alberto Molinari, Angelo Montanari, and Adriano Peron. On the complexity of model checking for syntactically maximal fragments of the interval temporal logic HS with regular expressions. In *International Symposium on Games, Automata, Logics, and Formal Verification, GandALF*, volume 277 of *EPTCS*, pages 31–45, 2017. doi:10.4204/EPTCS.256.3.
- 5 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.
- 6 Dmitry Chistikov and Christoph Haase. On the Power of Ordering in Linear Arithmetic Theories. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 168 of *LIPICs*, pages 119:1–119:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.ICALP.2020.119.
- 7 Christian Choffrut and Achille Frigeri. Deciding whether the ordering is necessary in a Presburger formula. *Discret. Math. Theor. C.*, 12(1):21–38, 2010. URL: <http://dmtcs.episciences.org/510>.
- 8 Stephen A. Cook. The complexity of theorem-proving procedures. In *Symposium on Theory of Computing, STOC*, pages 151–158, 1971. doi:10.1145/800157.805047.
- 9 Adrian W. Dudek. An explicit result for primes between cubes. *Functiones et Approximatio Commentarii Mathematici*, 55(2):177–197, 2016.
- 10 Raul Fervari and Alessio Mansutti. Modal logics and local quantifiers: A zoo in the elementary hierarchy. In *Foundations of Software Science and Computation Structures, FoSSaCS*, volume 13242 of *Lecture Notes in Computer Science*, pages 305–324, 2022. doi:10.1007/978-3-030-99253-8_16.
- 11 Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 122–135, 1998. doi:10.1007/978-3-7091-9459-1_5.
- 12 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 13 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.
- 14 Christoph Haase and Alessio Mansutti. On deciding linear arithmetic constraints over p -adic integers for all primes. In *Mathematical Foundations of Computer Science, MFCS*, volume 202 of *LIPICs*, pages 55:1–55:20, 2021. doi:10.4230/LIPICs.MFCS.2021.55.
- 15 Lane A. Hemachandra. The strong exponential hierarchy collapses. *J. Comput. Syst. Sci.*, 39(3):299–322, 1989. doi:10.1016/0022-0000(89)90025-1.
- 16 Albert E. Ingham. On The Estimation Of $N(\sigma, T)$. *Q. J. Math.*, os-11(1):201–202, 01 1940. doi:10.1093/qmath/os-11.1.201.
- 17 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 18 Dexter Kozen. *Theory of Computation*. Texts in Computer Science. Springer, 2006.
- 19 Markus Lohrey. Model-checking hierarchical structures. *J. Comput. Syst. Sci.*, 78(2):461–490, 2012. Games in Verification. doi:<https://doi.org/10.1016/j.jcss.2011.05.006>.
- 20 Antoni Lozano and José L. Balcázar. The complexity of graph problems fore succinctly represented graphs. In *Graph-Theoretic Concepts in Computer Science, WG*, volume 411 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 1990. doi:10.1007/3-540-52292-1.
- 21 Martin Lück. Canonical models and the complexity of modal team logic. In *Computer Science Logic, CSL*, volume 119 of *LIPICs*, pages 30:1–30:23, 2018. doi:10.4230/LIPICs.CSL.2018.30.

- 22 Harry G. Mairson. A simple proof of a theorem of Statman. *Theor. Comput. Sci.*, 103(2):387–394, 1992. doi:[https://doi.org/10.1016/0304-3975\(92\)90020-G](https://doi.org/10.1016/0304-3975(92)90020-G).
- 23 Alessio Mansutti. Notes on $kAExp(pol)$ problems for deterministic machines. *CoRR*, abs/2110.05630, 2021. arXiv:2110.05630.
- 24 Albert R. Meyer. The inherent computational complexity of theories of ordered sets. In *Proceedings of the International Congress of Mathematicians*, volume 2, pages 477–482, 1974.
- 25 Alberto Molinari. *Model checking: the interval way*. PhD thesis, Università degli Studi di Udine, 2019. URL: <https://arxiv.org/pdf/1901.03880.pdf>.
- 26 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- 27 Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Inf. Control.*, 71(3):181–185, 1986. doi:10.1016/S0019-9958(86)80009-2.
- 28 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 de Mathématiciens des Pays Slaves*, pages 92–101. 1929.
- 29 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.
- 30 Richard Statman. The typed λ -calculus is not elementary recursive. *Theor. Comput. Sci.*, 9(1):73–81, 1979. doi:[https://doi.org/10.1016/0304-3975\(79\)90007-0](https://doi.org/10.1016/0304-3975(79)90007-0).
- 31 user4625. Complexity class $NEXP^{NP}$. Theoretical Computer Science Stack Exchange, 2011. URL: <https://cstheory.stackexchange.com/q/6001>.
- 32 Helmut Veith. Succinct representation, leaf languages, and projection reductions. *Inf. Comput.*, 142(2):207–236, 1998. doi:10.1006/inco.1997.2696.
- 33 Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976. doi:10.1016/0304-3975(76)90062-1.