HAL
open science

The Universal Fragment of Presburger Arithmetic with Unary Uninterpreted Predicates is Undecidable

Matthias Horbach, Marco Voigt, Christoph Weidenbach

▶ **To cite this version:**

Matthias Horbach, Marco Voigt, Christoph Weidenbach. The Universal Fragment of Presburger Arithmetic with Unary Uninterpreted Predicates is Undecidable. 2017. hal-01592177

HAL Id: hal-01592177

https://inria.hal.science/hal-01592177v1

Preprint submitted on 22 Sep 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

arXiv:1703.01212v1 [cs.LO] 3 Mar 2017

The Universal Fragment of Presburger Arithmetic with Unary Uninterpreted Predicates is Undecidable

Matthias Horbach
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Marco Voigt
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany, Saarbrücken Graduate School of Computer Science

Christoph Weidenbach
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

The first-order theory of addition over the natural numbers, known as Presburger arithmetic, is decidable in double exponential time. Adding an uninterpreted unary predicate to the language leads to an undecidable theory. We sharpen the known boundary between decidable and undecidable in that we show that the purely universal fragment of the extended theory is already undecidable. Our proof is based on a reduction of the halting problem for two-counter machines to unsatisfiability of sentences in the extended language of Presburger arithmetic that does not use existential quantification. On the other hand, we argue that a single $\forall\exists$ quantifier alternation turns the set of satisfiable sentences of the extended language into a Σ^1_1-complete set.

Some of the mentioned results can be transfered to the realm of linear arithmetic over the ordered real numbers. This concerns the undecidability of the purely universal fragment and the Σ^1_1-hardness for sentences with at least one quantifier alternation.

Finally, we discuss the relevance of our results to verification. In particular, we derive undecidability results for quantified fragments of separation logic, the theory of arrays, and combinations of the theory of equality over uninterpreted functions with restricted forms of integer arithmetic. In certain cases our results even imply the absence of sound and complete deductive calculi.

1 Introduction

In 1929 Mojżesz Presburger presented a quantifier elimination procedure that decides validity of first-order sentences over the natural numbers with addition [31] (see [40] for an English translation and [14] for a textbook exposition). Today, this theory is known as *Presburger* arithmetic. In 1974 the computational time complexity of deciding validity of its sentences was shown to be double exponential by Fischer and Rabin [15]. It has been proved in several ways that the addition of a single uninterpreted unary predicate symbol to the language renders the validity problem undecidable. In 1957 Putnam [32] discussed this theory as one example of an undecidable theory that is somewhat stronger than the decidable theory of natural numbers with the successor function and a uninterpreted unary predicates. Lifshits mentioned in a note [27] (without giving a proof) that the addition of one predicate—of unspecified arity—to Presburger arithmetic leads to undecidability. In a technical report [13] from 1972 Downey gave an encoding of two-counter machines and their halting problem in Presburger arithmetic with a single unary predicate symbol. Moreover, undecidability is also implied by a general result due to Garfunkel and Schmerl [18] published in 1974. Seventeen years later Halpern [23] strengthened the undecidability result in that he proved

Π^1_1-completeness of this problem. Only recently, Speranski [39] gave an alternative characterization of the analytical hierarchy that is based on a reduction of Π^1_n-formulas with multiplication to Π^1_n-formulas without multiplication. Halpern's Π^1_1-completeness can be read as a special case of this more general point of view.

Halpern's proof rests on a result by Harel, Pnueli and Stavi (Proposition 5.1 in [24]), which states that the set of Gödel numbers of recurring Turing machines is Σ^1_1-complete.[1] A nondeterministic Turing machine is considered to be *recurring* if, started on an empty input tape, it is able to perform a nonterminating computation in which it infinitely often reaches its initial state (but not necessarily its initial configuration). The encoding of recurring Turing machines that Halpern employs in his proof results in formulas with two quantifier alternations. More precisely, the used sentences start with a $\forall^*\exists^*\forall^*$-prefix of first-order quantifiers when written in prenex normal form. The reduction by Speranski [39] relies on the same pattern of quantifier alternations. However, the required quantifier alternation in Halpern's proof can be simplified to $\forall^*\exists^*$, as pointed out by Speranski in [38]. Formally, Downey's encoding of two-counter machines in [13] exhibits a $\forall\exists$ alternation as well. However, in this case, suitable modifications lead to an encoding that does not require existential quantification. A crucial difference between Downey's encoding and ours is that the former concentrates on reachability of configurations, while the latter also considers the temporal order in which configurations are reached. One consequence is that our encoding facilitates the formalization of *recurrence* for nondeterministic two-counter machines. This requires some chronological information regarding the configurations that occur in a run that goes beyond reachability.

In the main part of the present paper we restrict the admitted language so that only universal first-order quantifiers may be used. Yet, the resulting validity and satisfiability problems remain undecidable.[2] To be more precise, we show Σ^0_1-completeness of the set of unsatisfiable sentences from the universal fragment of Presburger arithmetic extended with a single uninterpreted unary predicate symbol (cf. Theorems 3 and 10). As it turns out, this result is still valid when we use the reals as the underlying domain (Theorem 5).

Our proof proceeds by a reduction of the (negated) halting problem for two-counter machines (cf. [28]) to the satisfiability problem in the described language. A run of such a machine started with a certain input can be represented by a (potentially infinite) sequence of configurations $\langle \ell, c_1, c_2 \rangle$—triples of natural numbers—, where ℓ describes the control state of the machine and c_1, c_2 are the current values of the machine's counters. It is not very hard to imagine that such a sequence of configurations can be encoded by a (potentially infinite) string of bits. On the other hand, we can conceive any interpretation of a unary predicate over the natural numbers as a bit string. Given this basic idea, it remains to devise a translation of the program of an arbitrary two-counter machine into a suitable set of sentences from the universal fragment of Presburger arithmetic with an additional uninterpreted unary predicate symbol P. Suitable in this case means that any model of the resulting set of formulas interprets P such that it faithfully represents a run of the given machine on the given input. Section 3 is devoted to exactly this purpose. In Section 2 we recap the necessary preliminaries.

In Section 4 we relax our language restrictions a bit and show that allowing one quantifier alternation entails a high degree of undecidability. More precisely, the set of satisfiable $\forall^*\exists^2$-sentences is Σ^1_1-complete. The proof rests on a lemma, due to Alur and Henzinger [2], that rephrases Harel et al.'s Σ^1_1-hardness result for recurring Turing machines in terms of recurring two-counter machines. In order to apply this lemma, we have to adapt the encoding presented in Section 3 only slightly. All we need to do is to add the possibility of nondeterministic branching of the control flow and to replace the check for the reachability of the `halt` instruction by a condition

[1] Halpern's proof shifts the perspective from the validity problem to the problem of satisfiability. A Σ^1_1-complete satisfiability problem entails a Π^1_1-complete validity problem and vice versa, given that the considered languages are closed under negation. For the definition of the analytical hierarchy and the sets Π^1_1 and Σ^1_1, see, e.g., Chapter IV.2 in [30] or Chapter 16 in [36].

[2] In fact, this result can be obtained from Downey's proof [13]—even for Horn clauses—after suitable modifications to his encoding of two-counter machines. Apparently, Downey was not concerned with minimizing quantifier prefixes.

that formalizes the recurrence property.

Moreover, we observe that our undecidability and Σ_1^1-hardness results for settings over the integer domain can be transfered to corresponding results in the realm of real numbers. We do so at the end of Sections 3 and 4, respectively.

Finally, we discuss the relevance of our findings to the field of verification in Section 5. In particular, we derive undecidability results for quantified fragments of separation logic (Section 5.1), the theory of arrays (Section 5.2), and combinations of the theory of equality over uninterpreted functions with restricted forms of integer arithmetic (Sections 5.3 and 5.4). In certain cases our results even imply the absence of sound and complete deductive calculi.

The authors would like to stress that all of the results outlined above are obtained based on refinements of the encoding of two-counter machines presented in Section 3.2. To the authors' knowledge, a similarly general applicability is not documented for any other encoding of hard problems in the language of Presburger arithmetic augmented with uninterpreted predicate symbols.

2 Preliminaries

2.1 The universal fragment of Presburger arithmetic

We define the language of *Presburger arithmetic* to comprise all first-order formulas with equality over the signature $\langle 0, 1, + \rangle$. We use the following abbreviations, where s and t denote arbitrary terms over the signature $\langle 0, 1, + \rangle$:

- $s \neq t$ abbreviates $\neg(s = t)$,
- $s \leq t$ abbreviates $\exists z.\, s + z = t$,
- $s < t$ abbreviates $s + 1 \leq t$,
- for any integer $k \geq 1$ we use the constant k as abbreviation for the sum $1 + \ldots + 1$ with k summands,
- for any integer $k \geq 1$ and any variable x we write kx to abbreviate $x + \ldots + x$ with k summands.

For notational convenience, we shall also use the relation symbols $\leq, <$ in their symmetric variants $\geq$ and $>$, respectively. We follow the convention that negation binds strongest, that conjunction binds stronger than disjunction, and that all of the aforementioned bind stronger than implication. The scope of quantifiers shall stretch as far to the right as possible.

The *universal fragment of Presburger arithmetic* confines the language of Presburger arithmetic to sentences in prenex normal form in which universal quantification is allowed but existential quantification may not occur. The abbreviations $s \leq t$ and $s < t$ are exempt from the rule, i.e. we pretend that they do not stand for a formula that contains a quantifier.

This exemption does not constitute a serious weakening of the restriction to universal quantification, because any atom $s \leq t$ can be replaced with an atom $\neg(t < s)$, which is equivalent to $\forall z.\, \neg(t + 1 + z = s)$. For instance, the sentence

$$\varphi \;:=\; \forall xy.\, x = y \;\longrightarrow\; x \leq y$$

belongs to the universal fragment of Presburger arithmetic, although it is actually a short version of $\forall xy.\, x = y \;\longrightarrow\; \exists z.\, x + z = y$. However, φ is equivalent to $\forall xy.\, x = y \;\longrightarrow\; \neg(y < x)$, which stands for $\forall xy.\, x = y \;\longrightarrow\; \neg(\exists z.\, y + 1 + z = x)$ and is thus equivalent to

$$\forall xyz.\, x = y \;\longrightarrow\; y + 1 + z \neq x\ .$$

2.2 Minsky's two-counter machines

Minsky has introduced the two-counter machine as a Turing-complete model of computation (Theorem 14.1-1 in [28]). We shall only briefly recap the basic architecture of this kind of computing device.

A *two-counter machine* $\mathcal{M}$ consists of two counters C_1, C_2 and a finite program whose lines are labeled with integers $0, \ldots, K$. Each program line contains one of five possible instructions with the following meaning:

`inc(`C_1`)`	increment counter C_1 and proceed with the next instruction;
`inc(`C_2`)`	increment counter C_2 and proceed with the next instruction;
`test&dec(`C_1`,`ℓ`)`	if $C_1 > 0$ then decrement C_1 and proceed with the next instruction, otherwise proceed with instruction ℓ and leave the counters unchanged;
`test&dec(`C_2`,`ℓ`)`	if $C_2 > 0$ then decrement C_2 and proceed with the next instruction, otherwise proceed with instruction ℓ and leave the counters unchanged;
`halt`	halt the computation.

We tacitly assume that the last program line of any two-counter machine contains the `halt` instruction. In the initial state of a given two-counter machine the input is stored in the two counters. The computation of the machine starts at the first program line, labeled 0.

Notice that the described machine model leads to deterministic computation processes. Since the described machine model is strong enough to simulate any deterministic Turing machine, the halting problem for two-counter machines is undecidable.

Proposition 1 (corollary of Theorem 14.1-1 from [28])**.** It is impossible to devise an algorithm that is able to decide for every two-counter machine $\mathcal{M}$ and every input $\langle m, n\rangle \in \mathbb{N} \times \mathbb{N}$ whether $\mathcal{M}$ ever reaches a program line containing the `halt` instruction when started on $\langle m, n\rangle$.

3 Encoding a deterministic two-counter machine

Since validity of Presburger arithmetic sentences is decidable[3], we need some additional language element in order to encode the computations of a two-counter machine on a given input. It turns out that it is sufficient to add an uninterpreted unary predicate symbol P to the underlying signature and thus consider first-order sentences over the extended signature $\langle 0, 1, +, P\rangle$. As soon as we have constructed a sentence φ that encodes a given machines $\mathcal{M}$ together with a given input pair $\langle m, n\rangle$, we are interested in the (un)satisfiability of φ. Hence, we pose the question: Is there an interpretation $\mathcal{I}$ with $P^{\mathcal{I}} \subseteq \mathbb{N}$ such that $\mathcal{I} \models \varphi$, or is there no such interpretation?

3.1 Informal description of the encoding

Since any interpretation $P^{\mathcal{I}}$ of the predicate symbol P is a subset of the natural numbers, we can view $P^{\mathcal{I}}$ as an infinite sequence of bits $b_0 b_1 b_2 \ldots$, where for every $n \in \mathbb{N}$ we have

$$b_n := \begin{cases} 0 & \text{if } n \notin P^{\mathcal{I}}, \\ 1 & \text{if } n \in P^{\mathcal{I}}. \end{cases}$$

Given a two-counter machine $\mathcal{M}$ with $K+1$ program lines, labeled $0, \ldots, K$, and two input values m, n, we shall encode all the configurations that occur during the run of $\mathcal{M}$ when started on input $\langle m, n\rangle$. One such configuration consists of the address of the program line that is to be executed in the current step, the value of the first counter C_1, and the value of the second counter

[3] Regarding sentences, i.e. closed formulas, of Presburger arithmetic, validity and satisfiability coincide. The reason is that the domain is fixed to the nonnegative integers and all language elements in the underlying signature $\langle 0, 1, +\rangle$ have a fixed interpretation. As soon as we add uninterpreted operations or relations, the two notions differ. In this latter case, we shall consider decidability of the satisfiability problem rather than of the validity problem.

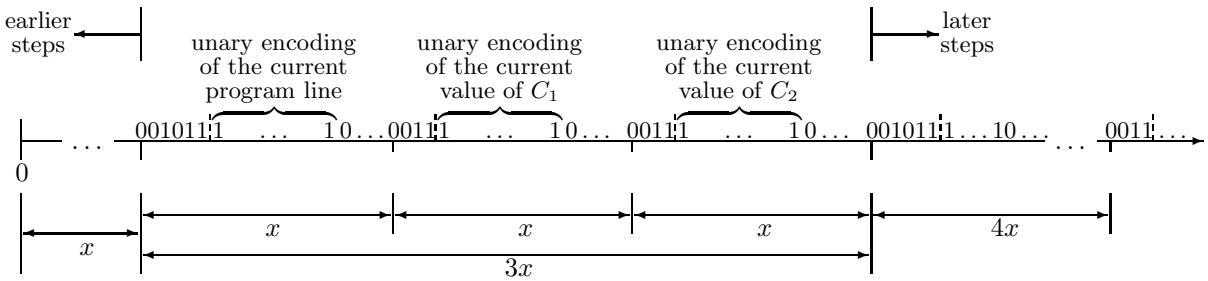
C_2. We divide the bit sequence $P^{\mathcal{I}}$ into chunks of growing length, each delimited by the bit sequence 001011. Such a chunk is divided into three subchunks, using the bit sequence 0011 as a delimiter. The first subchunk holds the current program line encoded in unary. The second and third subchunk store the current values of the counters C_1, C_2, respectively, also encoded in unary notation. Hence, every chunk has the form

$$\underbrace{0010111}_{\substack{\text{left de-}\\\text{limiter}}}1^{\ell}0\ldots0\ \underbrace{00111}_{\substack{\text{first sub-}\\\text{delimiter}}}1^{c_1}0\ldots0\ \underbrace{00111}_{\substack{\text{second}\\\text{subde-}\\\text{limiter}}}1^{c_2}0\ldots0\ ,$$

where ℓ is the address of the program line to be executed, c_1 is the value currently stored in counter C_1, and c_2 is the value currently stored in counter C_2. The subsequences 1^{ℓ}, 1^{c_1} and 1^{c_2} are followed by blocks of zeros that fill up the gap before the next 0011-delimiter (indicating the start of the subsequent subchunk) or the next 001011-delimiter (indicating the beginning of the successor configuration).

The length of each chunk and its subchunks increases with the number of computation steps that have already been performed. This makes sure that there is always enough space available to store the current counter values, which may thus become arbitrarily large. Of course, we have to provide sufficient space in the beginning such that the address of any program line and the initial counter values m and n may be stored. In order to achieve this, we define the constant $d := \max\{K, m, n\}+6$ and require that the leftmost chunk starts at position d, i.e. there is a 001011-delimiter starting at position d but none starting left of d. The first three subchunks have length d each. Thus, the second chunk starts at position $4c$. The subchunks of the second chunk, however, have a length of $4d$.[4] Hence, the total length of the second chunk is $12d$. This scheme continues indefinitely, i.e. the starting points of the chunks in the bit sequence are $d, 4d, 16d, 64d, 256d$, and so on. Consequently, all the chunks are large enough to store all possibly occurring counter values, as these can increase by at most one in every step of the computation.

The following figure illustrates the structure of a single chunk in the sequence, starting at position x.



3.2 Formal encoding of two-counter machine computations

Recall that we assume to be given a two-counter machine $\mathcal{M}$ with $K+1$ program lines, labeled $0, \ldots, K$, and two input values m and n. We use the following abbreviations for arbitrary terms t:

$$\begin{aligned}
\psi_{001011}(t) &:= \neg P(t) \ \wedge\ \neg P(t+1) \ \wedge\ P(t+2) \ \wedge\ \neg P(t+3) \ \wedge\ P(t+4) \ \wedge\ P(t+5)\\
\psi_{0011}(t) &:= \neg P(t) \ \wedge\ \neg P(t+1) \ \wedge\ P(t+2) \ \wedge\ P(t+3)\\
\psi_{01}(t) &:= \neg P(t) \ \wedge\ P(t+1)\\
\psi_{10}(t) &:= P(t) \ \wedge\ \neg P(t+1)\\
\chi_j(t) &:= \psi_{10}(t+5+j) \qquad\qquad \text{for } j = 0, \ldots, K
\end{aligned}$$

[4] Technically, a length of $d+1$ for the subchunks of the second chunk would suffice. After all, the value of a counter can increase by at most one in a single computation step. However, we have chosen to increase the length in an exponential fashion rather than a linear one, in order to keep the encoding simple.

First of all, we set up the general structure of the predicate P. Let d denote the integer with the value $d := \max\{K+6, m+4, n+4\}$. We use d as the starting point of our encoding.

$$
\begin{aligned}
\varphi_1 := \; & \psi_{001011}(d) && (1)\\
& \wedge \; \big(\forall x.\, x < d \;\longrightarrow\; \neg P(x)\big) && (2)\\
& \wedge \; \big(\forall x.\, \psi_{001011}(x) \;\longrightarrow\; \psi_{0011}(2x) \;\wedge\; \psi_{0011}(3x) \;\wedge\; \psi_{001011}(4x)\big) && (3)\\
& \wedge \; \big(\forall xy.\, \psi_{001011}(x) \;\wedge\; \psi_{001011}(y) \;\wedge\; x \le y \;\wedge\; y < 4x \;\longrightarrow\; x = y\big) && (4)\\
& \wedge \; \big(\forall xy.\, \psi_{001011}(x) \;\wedge\; \psi_{0011}(y) \;\wedge\; x \le y \;\longrightarrow\; y \ge 2x\big) && (5)\\
& \wedge \; \big(\forall xy.\, \psi_{001011}(x) \;\wedge\; \psi_{0011}(y) \;\wedge\; 2x < y \;\longrightarrow\; y \ge 3x\big) && (6)\\
& \wedge \; \big(\forall xy.\, \psi_{001011}(x) \;\wedge\; \psi_{0011}(y) \;\wedge\; 3x < y \;\longrightarrow\; y \ge 4x\big) && (7)\\
& \wedge \; \big(\forall xyu.\, \psi_{001011}(x) \;\wedge\; \psi_{01}(y) \;\wedge\; x+5 < y \;\wedge\; y < 4x \;\wedge\; u+1 = y \;\longrightarrow\; \psi_{0011}(u)\big) && (8)
\end{aligned}
$$

Formula (1) sets the first 001011-delimiter at position d and Formula (2) ensures that this is indeed the leftmost such delimiter. Formula (3) sets up all the other delimiters and Formulas (4) to (7) guarantee that there are no spurious delimiters in between them. Formula (8) stipulates that every 01 substring is part of one of the delimiters, i.e. there cannot be a substring 01 that lies outside of a 001011- or 0011-delimiter. This does also entail that between one delimiter (001011 or 0011) and the subsequent one there is exactly one substring 10, possibly overlapping with the last or first bit of one of the delimiters. Hence, this substring uniquely marks the end of the number encoded in the respective subchunk.

There is one peculiarity in Formula (8) that is worth noticing, namely, the role of the variable u. We need to introduce it to facilitate the formulation of the term $y-1$, since the signature of Presburger arithmetic does not contain the minus operation. Hence, informally, Formula (8) stands for

$$\forall xy.\, \psi_{001011}(x) \;\wedge\; \psi_{01}(y) \;\wedge\; x+5 < y \;\wedge\; y < 4x \;\longrightarrow\; \psi_{0011}(y-1)\ .$$

We will use this pattern again later, when we shall encode the decrement operation for counters.

The following formula sets the initial values of the counters. Moreover, it sets the initial program line, which we assume to be the very first one:

$$\varphi_2 := \;\; \chi_0(d) \;\;\wedge\;\; \psi_{10}(2d+3+m) \;\;\wedge\;\; \psi_{10}(3d+3+n)\ .$$

Regarding the encoding of program lines, we have to enforce that the current program line never exceeds K. This is easily done with the formula

$$\varphi_3 := \;\; \forall xy.\, \psi_{001011}(x) \;\;\wedge\;\; \psi_{10}(y) \;\;\wedge\;\; x+5 \le y \;\;\wedge\;\; y \le 2x \;\;\longrightarrow\;\; y \le x+5+K\ .$$

The previous formulas already ensure that exactly one address of a program line is encoded.

Next we encode the control flow of $\mathcal{M}$. We assume that the following instructions occur in program line j for some $j \in \{0, \ldots, K\}$.

Encoding of the instruction j : `inc(`C_1`)`:

$$
\begin{aligned}
& \forall xyz.\, \psi_{001011}(x) \;\wedge\; 2x \le y \;\wedge\; y \le 3x \;\wedge\; \psi_{10}(y) \;\wedge\; 3x \le z \;\wedge\; z \le 4x \;\wedge\; \psi_{10}(z) \;\wedge\; \chi_j(x)\\
& \qquad \longrightarrow\; \psi_{10}(6x+y+1) \;\wedge\; \psi_{10}(9x+z) \;\wedge\; \chi_{j+1}(4x)
\end{aligned}
$$

The subfomula $\psi_{001011}(x)$ in the premise of the implication states that the chunk encoding the currently regarded configuration starts at position x. The other preconditions make clear that y and z correspond to the positions at which we find 10-substrings in the two subchunks storing the current counter values:

$$
\begin{array}{ccccccccc}
x & & 2x & & y & & 3x & & z\\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow\\
\underbrace{0010111}_{\text{left delimiter}}1^{\ell}0\ldots0 & & \underbrace{00111}_{\text{first subdelimiter}}1^{c_1-1}10\ldots0 & & & & \underbrace{00111}_{\text{second subdelimiter}}1^{c_2-1}10\ldots0 & &
\end{array}
$$

Hence, C_1 and C_2 currently hold the values $c_1 = y - 2x - 3$ and $c_2 = z - 3x - 3$, respectively. Since the subsequent chunk starts at position $4x$ and its second and third subchunks start at positions $8x$ and $12x$, respectively, we know that there must be one 10-substring at position $8x + 3 + c_1 + 1 = 6x + y + 1$—the first counter is incremented by 1—and one at position $12x + 3 + c_2 = 9x + z$—the value of the second counter remains unchanged. Moreover, the machine currently executes program line j and is to continue at program line $j+1$. Therefore, we put the formula $\chi_j(x)$ in the premise and the formula $\chi_{j+1}(4x)$ into the consequent of the implication.

Encoding of the instruction j : `inc(`C_2`)`:

$$\forall xyz.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x) \\ \longrightarrow\ \psi_{10}(6x + y) \ \wedge\ \psi_{10}(9x + z + 1) \ \wedge\ \chi_{j+1}(4x)$$

Encoding of the instruction j : `test&dec(`C_1`,`ℓ`)`:

The case of C_1 storing 0:

$$\forall xyz.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \\ \wedge\ \chi_j(x) \ \wedge\ y = 2x + 3 \\ \longrightarrow\ \psi_{10}(6x + y) \ \wedge\ \psi_{10}(9x + z) \ \wedge\ \chi_\ell(4x)$$

The condition $y = 2x + 3$ ensures that the first counter stores the value 0.

The case of C_1 storing a value greater than 0:

$$\forall xyzu.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \\ \wedge\ \chi_j(x) \ \wedge\ y > 2x + 3 \ \wedge\ u + 1 = 6x + y \\ \longrightarrow\ \psi_{10}(u) \ \wedge\ \psi_{10}(9x + z) \ \wedge\ \chi_{j+1}(4x)$$

The condition $y > 2x + 3$ ensures that the first counter stores a value strictly greater than 0. Notice that u stands for the term $6x + y - 1$ and thus facilitates the decrement operation.

Encoding of the instruction j : `test&dec(`C_2`,`ℓ`)`:

The case of C_2 storing 0:

$$\forall xyz.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \\ \wedge\ \chi_j(x) \ \wedge\ z = 3x + 3 \\ \longrightarrow\ \psi_{10}(6x + y) \ \wedge\ \psi_{10}(9x + z) \ \wedge\ \chi_\ell(4x)$$

The case of C_2 storing a value greater than 0:

$$\forall xyzu.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \\ \wedge\ \chi_j(x) \ \wedge\ z > 3x + 3 \ \wedge\ u + 1 = 9x + z \\ \longrightarrow\ \psi_{10}(6x + y) \ \wedge\ \psi_{10}(u) \ \wedge\ \chi_{j+1}(4x)$$

Encoding of the instruction j : `halt`:

$$\forall xyz.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x) \\ \longrightarrow\ \psi_{10}(6x + y) \ \wedge\ \psi_{10}(9x + z) \ \wedge\ \chi_K(4x)$$

The consequent of the implication ensures that the counters remain unchanged and that the computation continues at program line K. Since we assumed the K-th program line to contain the instruction `halt`, the rest of the bit sequence will repeat the same chunk structure again and again, as the counter values will remain unchanged and the encoded program line will also repeat indefinitely.

Finally, we pose the central question concerning the halting behavior of the machine: Does the machine ever reach a program line containing the `halt` instruction? The question is posed as a requirement in a negative fashion:

$$\varphi_4 := \ \forall x.\ \psi_{001011}(x) \ \longrightarrow \ \neg\chi_K(x)\ .$$

Technically speaking, we require that the machine never reaches the K-th program line. Due to the encoding of the `halt` instruction in arbitrary program lines j, it is clear that whenever any program line containing `halt` is reached the K-th program line is reached in the subsequent step. Hence, the above formula is satisfied if and only if the machine will never reach the instruction `halt` when started on the given input.

Lemma 2. The two-counter machine $\mathcal{M}$ with $K+1$ program lines, labeled $0,\ldots,K$, started on input $\langle m,n\rangle$ eventually reaches a program line containing the instruction `halt` if and only if the described encoding of $\mathcal{M}$ in Presburger arithmetic with an additional uninterpreted unary predicate symbol is unsatisfiable.

Proof sketch. We first observe the following technical properties of every interpretation $\mathcal{I}$ with $\mathcal{I} \models \varphi_1$.

(a) For every integer $r \in \mathbb{N}$ we have $\mathcal{I}, [x{\mapsto}r] \models \psi_{001011}(x)$ if and only if $r = 4^i d$ for some $i \in \mathbb{N}$.

(b) For every integer $r \in \mathbb{N}$ we have $\mathcal{I}, [x{\mapsto}r] \models \psi_{0011}(x)$ if and only if $r = 2 \cdot 4^i d$ or $r = 3 \cdot 4^i d$ for some $i \in \mathbb{N}$.

(c) For every integer $r \in \mathbb{N}$ we have $\mathcal{I}, [x{\mapsto}r] \models \psi_{01}(x)$ if and only if

$$r \in \bigcup_{i\in\mathbb{N}} \big\{4^i d+1, 4^i d+3, 2\cdot 4^i d+1, 3\cdot 4^i d+1\big\}\ .$$

(d) Suppose there are integers $i, r, q \in \mathbb{N}$ such that $4^i d+5 \leq r, q < 2\cdot 4^i d$. If we have $\mathcal{I}, [x{\mapsto}r] \models \psi_{10}(x)$ and $\mathcal{I}, [x{\mapsto}q] \models \psi_{10}(x)$, then it follows that $r = q$.

(e) Suppose there are integers $i, r, q \in \mathbb{N}$ such that $2\cdot 4^i d+3 \leq r, q < 3\cdot 4^i d$. If we have $\mathcal{I}, [x{\mapsto}r] \models \psi_{10}(x)$ and $\mathcal{I}, [x{\mapsto}q] \models \psi_{10}(x)$, then it follows that $r = q$.

(f) Suppose there are integers $i, r, q \in \mathbb{N}$ such that $3\cdot 4^i d+3 \leq r, q < 4\cdot 4^i d$. If we have $\mathcal{I}, [x{\mapsto}r] \models \psi_{10}(x)$ and $\mathcal{I}, [x{\mapsto}q] \models \psi_{10}(x)$, then it follows that $r = q$.

(g) For every integer $i \in \mathbb{N}$ there are integers $r_1, r_2, r_3 \in \mathbb{N}$ such that

- $4^i d+5 \leq r_1 < 2\cdot 4^i d$ and $\mathcal{I}, [x{\mapsto}r_1] \models \psi_{10}(x)$,
- $2\cdot 4^i d+3 \leq r_2 < 3\cdot 4^i d$ and $\mathcal{I}, [x{\mapsto}r_2] \models \psi_{10}(x)$, and
- $3\cdot 4^i d+3 \leq r_3 < 4\cdot 4^i d$ and $\mathcal{I}, [x{\mapsto}r_3] \models \psi_{10}(x)$.

Due to the above observations, it is clear that any model $\mathcal{I}$ of φ_1 interprets P in such a way that it uniquely represents an infinite sequence of triples of nonnegative integers encoded in unary, just as we have described it earlier. If, in addition, $\mathcal{I}$ satisfies φ_2 and φ_3, then the first triple of the sequence has the form $\langle 0, m, n\rangle$ and the first component of every triple in the sequence does not exceed K.

Given the program of $\mathcal{M}$, we denote by $\varphi_{\mathcal{M}}$ the sentence that encodes $\mathcal{M}$'s program in accordance with the described formula schemes. Hence, for any model $\mathcal{I} \models \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{\mathcal{M}}$ the interpretation $P^{\mathcal{I}}$ of P does not only represent a sequence of triples of integers but also establishes relations between the triples in the sequence, such that they mimic the operations that $\mathcal{M}$ would perform on its counters in accordance with the instructions in its program. The only technical difference is that whenever $\mathcal{M}$ enters a configuration $\langle \ell, c_1, c_2\rangle$ such that instruction ℓ in $\mathcal{M}$'s

program is `halt`, then all later configurations have the form $\langle K, c_1, c_2\rangle$. All in all, $P^{\mathcal{I}}$ is a faithful encoding of some run of $\mathcal{M}$ starting from the input $\langle m, n\rangle$.

On the other hand, since $\mathcal{M}$ is deterministic, there is a unique sequence

$$\tau := \langle 0, m, n\rangle\langle \ell_1, c_{1,1}, c_{2,1}\rangle\langle \ell_2, c_{1,2}, c_{2,2}\rangle\langle \ell_3, c_{1,3}, c_{2,3}\rangle \dots$$

of configurations that represents the *run of $\mathcal{M}$ started on input $\langle m, n\rangle$*. If τ is finite and thus contains a halting configuration $\langle \ell, c_1, c_2\rangle$ as its last triple, we concatenate the infinite sequence $\langle K, c_1, c_2\rangle\langle K, c_1, c_2\rangle \dots$ and thus obtain an infinite sequence again. This infinite sequence (be it originally infinite or made so artificially) can be translated into an interpretation $\mathcal{I}_\tau$ such that $\mathcal{I}_\tau \models \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{\mathcal{M}}$.

So far, we have seen that $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{\mathcal{M}}$ is satisfiable and that every model represents the unique run τ of $\mathcal{M}$ started on input $\langle m, n\rangle$. Clearly, we now observe for any model $\mathcal{I} \models \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{\mathcal{M}}$ that $\mathcal{I} \models \varphi_4$ holds if and only if τ *does not* contain a triple $\langle K, c_1, c_2\rangle$ for any $c_1, c_2 \in \mathbb{N}$. Hence, $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{\mathcal{M}} \wedge \varphi_4$ is unsatisfiable if and only if $\mathcal{M}$ reaches the `halt` instruction when started on the input $\langle m, n\rangle$. □

Together with the fact that the halting problem for two-counter machines is undecidable (cf. Proposition 1), we have shown the following theorem.

Theorem 3. (Un)satisfiability of the universal fragment of Presburger arithmetic with a single additional uninterpreted unary predicate symbol is undecidable.

An alternative proof of Lemma 2 uses *hierarchic superposition* [4, 1, 26, 5]. Transforming the two-counter machine encoding $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_{\mathcal{M}} \wedge \varphi_4$ into *conjunctive normal form (CNF)* results in a first-order clause set Φ over Presburger arithmetic without any uninterpreted function or constant symbols. The only uninterpreted symbol in Φ is the predicate P. Therefore, hierarchic superposition is *refutationally complete* for this clause set. That is, Φ unsatisfiable if and only if hierarchic superposition derives a contradiction, the empty clause, out of the clauses in Φ. Hierarchic superposition first resolves away all P literals from a clause and in case the clause only contains arithmetic literals, it checks their unsatisfiability.

By an inductive argument, hierarchic superposition derives exactly the states of the two-counter machine via ground literals of the form $[\neg]P(k)$, $k \in \mathbb{N}$. Let $\Psi_{001011}(s)$, $\Psi_{10}(s)$, and $X_j(s)$ be the sets of clauses—unit clauses in this case—that correspond to the formulas $\psi_{001011}(s)$, $\psi_{10}(s)$, $\chi_j(s)$, respectively, for any term s.

Suppose that the ground clauses in $\Psi_{001011}(k)$, $X_j(k)$, $\Psi_{10}(2k+3+c_1)$, and $\Psi_{10}(3k+3+c_2)$ (with $2k+3+c_1 < 3k$ and $3k+3+c_2 < 4k$) have been derived already. They represent the two-counter machine at program line j with counter values c_1 and c_2. Without loss of generality, we assume that the instruction of the machine at line j is an increment on the second counter. The other operations are argued analogously. Consider the clauses that result from the formula

$$\begin{aligned}&\forall xyz.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x)\\ &\qquad \longrightarrow\ \psi_{10}(6x+y) \ \wedge\ \psi_{10}(9x+z+1) \ \wedge\ \chi_{j+1}(4x)\ .\end{aligned}$$

All literals in $\psi_{001011}(x)$, $\psi_{10}(y)$, $\psi_{10}(z)$, and $\chi_j(x)$ can be resolved away via superposition and the substitution $\sigma = \{x \mapsto k, y \mapsto 2k+3+c_1, z \mapsto 3k+3+c_2\}$, thus generating the new ground clauses in $\Psi_{10}(6x+y)\sigma$, $\Psi_{10}(9x+z+1)\sigma$, $X_{j+1}(4x)\sigma$, which represent the next state of the two-counter machine. The underlying strategy always selects all negative literals resulting out of $\psi_{001011}(x)$, $\psi_{10}(y)$, $\psi_{10}(z)$, and $\chi_j(x)$. Resolving those away with the respective ground unit literals, fixes already the values for x, y, and z. Thus all other P literals resulting from the delimiter encoding can then be reduced away by subsumption resolution. The ground literals in $\Psi_{001011}(4x)\sigma$ are generated by this strategy with the clauses resulting from (3). This finishes the proof that hierarchic superposition generates the states of the two-counter machine by the derivation of respective P ground literals.

However, it remains to be shown that there are no derivations with other clauses that might lead to a contradiction. Any clause resulting from the encoding of a two-counter machine instruction

contains the literals in $X_\ell(x)$ for the respective line of the program ℓ. Now if the encoding of the instruction of line ℓ is resolved with ground P literals representing a line $j \neq \ell$ via the above selection strategy, then all resulting clauses turn into tautologies as $\chi_\ell(x)$ does not hold for the respective positions as a result of the different delimiter bit sequences that cannot be confused. Hence all these inferences become redundant and can be neglected. This is a consequence of the clauses resulting from (1) to (8). So given ground literals $\psi_{001011}(k)$, $\chi_j(k)$, $\psi_{10}(2k+3+c_1)$, $\psi_{10}(3k+3+c_2)$, there is exactly one line encoding formula that can be resolved with that does not lead to tautologies: the encoding of program line j.

3.3 Reducing the number of variables

We can formulate the encoding with at most two variables per formula, if we are willing to incorporate $\leq$ into the signature of Presburger arithmetic rather than conceiving expressions of the form $s \leq t$ as an abbreviation of $\exists z. s + z = t$. If we accept this extended signature, we still have to modify the encoding formulas a little bit. As a matter of fact, the criterion is already met by most of the formulas in the previous subsection. Only Subformula (8) of φ_1 and the encodings of the program instructions have to be modified as follows.

Modified variant of Subformula (8):

$$\forall xy.\ \psi_{001011}(x) \ \wedge\ \psi_{01}(y+1) \ \wedge\ x+5 < y+1 \ \wedge\ y+1 < 4x \ \longrightarrow\ \psi_{0011}(y)$$

In this variant of the subformula we get rid of the variable u that we have introduced in order to simulate subtraction. We do so by using $y+1$ in the premise rather than y. In this way subtraction by one in the consequent amounts to leaving away the $+1$. We will reuse this pattern in the encodings of the decrement operation later.

Modified encoding of the instruction j : `inc(`C_1`)`:

$$\forall xy.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ \chi_j(x) \ \longrightarrow\ \psi_{10}(6x+y+1) \ \wedge\ \chi_{j+1}(4x)$$

$$\forall xz.\ \psi_{001011}(x) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x) \ \longrightarrow\ \psi_{10}(9x+z)$$

For this instruction and most of the others we split the encoding formula into two parts: the first formula realizes the y-part of the original encoding and the second formula realizes the z-part.

Modified encoding of the instruction j : `inc(`C_2`)`:

$$\forall xy.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ \chi_j(x) \ \longrightarrow\ \psi_{10}(6x+y) \ \wedge\ \chi_{j+1}(4x)$$

$$\forall xz.\ \psi_{001011}(x) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x) \ \longrightarrow\ \psi_{10}(9x+z+1)$$

Modified encoding of the instruction j : `test&dec(`C_1`,`ℓ`)`:

The case of C_1 storing 0:

$$\forall xz.\ \psi_{001011}(x) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x) \ \wedge\ \psi_{10}(2x+3)$$
$$\longrightarrow\ \psi_{10}(8x+3) \ \wedge\ \psi_{10}(9x+z) \ \wedge\ \chi_\ell(4x)$$

The subformula $\psi_{10}(2x+3)$ in the premise ensures that the counter C_1 currently stores a 0 and the subformula $\psi_{10}(8x+3)$ requires that C_1 still stores 0 in the next step. Notice that we do not need a variable y to address the corresponding bit positions, since we can directly calculate these positions from x.

The case of C_1 storing a value greater than 0:

$$\forall xy.\ \psi_{001011}(x) \ \wedge\ 2x \leq y+1 \ \wedge\ y+1 \leq 3x \ \wedge\ \psi_{10}(y+1) \ \wedge\ \chi_j(x) \ \wedge\ y+1 > 2x+3 \\ \longrightarrow\ \psi_{10}(6x+y) \ \wedge\ \chi_{j+1}(4x)$$

$$\forall xz.\ \psi_{001011}(x) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x) \ \wedge\ \neg\psi_{10}(2x+3) \\ \longrightarrow\ \psi_{10}(9x+z)$$

In the first formula $y+1 > 2x+3$ ensures that the value of C_1 is greater than zero. We need the variable y to refer to C_1's value for the decrement operation. In the second formula C_1's exact value is not important and thus $\neg\psi_{10}(2x+3)$ is sufficient for ensuring that C_1's value is strictly positive.

Modified encoding of the instruction j : `test&dec(`C_2`,`ℓ`)`:

The case of C_2 storing 0:

$$\forall xy.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ \chi_j(x) \ \wedge\ \psi_{10}(3x+3) \\ \longrightarrow\ \psi_{10}(6x+y) \ \wedge\ \psi_{10}(12x+3) \ \wedge\ \chi_\ell(4x)$$

The case of C_2 storing a value greater than 0:

$$\forall xy.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ \chi_j(x) \ \wedge\ \neg\psi_{10}(3x+3) \\ \longrightarrow\ \psi_{10}(6x+y) \ \wedge\ \chi_{j+1}(4x)$$

$$\forall xz.\ \psi_{001011}(x) \ \wedge\ 3x \leq z+1 \ \wedge\ z+1 \leq 4x \ \wedge\ \psi_{10}(z+1) \ \wedge\ \chi_j(x) \ \wedge\ z+1 > 3x+3 \\ \longrightarrow\ \psi_{10}(9x+z)$$

Modified encoding of the instruction j : `halt`:

$$\forall xy.\ \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ \chi_j(x) \ \longrightarrow\ \psi_{10}(6x+y) \ \wedge\ \chi_K(4x)$$

$$\forall xz.\ \psi_{001011}(x) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x) \ \longrightarrow\ \psi_{10}(9x+z)$$

Theorem 4. Extend the standard signature $\langle 0, 1, + \rangle$ of Presburger arithmetic by the relation $\leq$ and an uninterpreted unary predicate symbol P to obtain $\langle 0, 1, +, \leq, P \rangle$. (Un)satisfiability of the universal fragment of Presburger arithmetic over this extended signature is undecidable, if we allow at least two variables per clause.

To the authors' knowledge, the case of a single variable per clause is not known to be (un)decidable. From Downey's encoding [13] it follows that one variable is sufficient, if we further extend the signature by modulo operators mod k for finitely many integer constants k.

3.4 Using the reals as underlying domain

Presburger arithmetic is defined on the natural numbers and we have shown that adding a unary uninterpreted predicate symbol leads to undecidability. It is known that validity and satisfiability over real numbers exhibits a different behavior when decidability is concerned.

In the context of the reals, we consider $\leq$ and $<$ to be part of the signature and not abbreviations for more complicated terms. Of course, we assume that they have the standard semantics over the reals.

We can directly use the encoding that we have presented for the integers in order to show undecidability over the real domain. The crucial point is that we have encoded the reachability of the `halt` instruction in a negative fashion. If the machine $\mathcal{M}$ reaches a `halt` instruction, then we cannot find a model of the encoding formula set, since any interpretation that faithfully represents

the run of $\mathcal{M}$ on the given input must violate the condition $\neg\chi_K(n)$ for some integer n for which $\psi_{001011}(n)$ is true. We have used this observation to prove Lemma 2. The described conflict does not vanish when we assume a larger domain. If, on the other hand, the machine $\mathcal{M}$ does not reach a `halt` instruction, then there is a model of the encoding formula set. In particular, there is a model in which P is interpreted such that it exclusively contains integers and no reals at all. Hence, the fact that we are dealing with an extended domain does not affect the circumstances under which the encoding formula set is unsatisfiable or not. Consequently, we have the following undecidability result.

Theorem 5. (Un)satisfiability of the universal fragment of linear arithmetic over the reals with an additional uninterpreted unary predicate symbol is undecidable.

3.5 Unary function symbols and the Horn fragment

The uninterpreted unary predicate P in our encoding of two-counter machines can be replaced by an uninterpreted unary function $f : \mathbb{N} \to \mathbb{N}$ over the natural numbers. We simply add the assertion $\forall x.\, f(x) \leq 1$ and substitute every negative literal $\neg P(t)$ with $f(t) = 0$ and every positive literal $P(t)$ with $f(t) = 1$, where t is any term. (Implicitly, we exploit the fact that f is interpreted by a total function $f^{\mathcal{I}}$ in any interpretation $\mathcal{I}$.) After this substitution, transforming the encoding formula set from Section 3.2 into *conjunctive normal norm (CNF)* yields a clause set that is *Horn*, i.e. every clause contains at most one positive literal. By this line of argument we obtain the following theorem.

Theorem 6. (Un)satisfiability of the universal Horn fragment of Presburger arithmetic with a single additional uninterpreted unary function symbol is undecidable.

Over the domain of the reals, we can replace the predicate symbol P in the same spirit, yet in a slightly different way. For one thing, we add the assertion $\forall x.\, 0 \leq f(x) \;\wedge\; f(x) \leq 1$ to the encoding, which also introduces an explicit lower bound to the values of f. As this assertion alone does not guarantee that in any model the image of $f : \mathbb{R} \to \mathbb{R}$ contains at most two values, we replace any occurrence of $\neg P(t)$ with $f(t) = 0$ and any occurrence of $P(t)$ with $f(t) > 0$. Again, a CNF transformation leads to a Horn clause set.

Theorem 7. (Un)satisfiability of the universal Horn fragment of linear arithmetic over the reals with a single additional uninterpreted unary function symbol is undecidable.

3.6 On the degree of unsolvability

We have shown that the unsatisfiability problem of the universal fragment of Presburger arithmetic with additional uninterpreted predicate symbols is undecidable. Next, we shall argue that this set is recursively enumerable. In order to prove this, it suffices to give a sound calculus that, given an unsatisfiable sentence over the language in question, derives the empty clause, i.e. *falsity*, in finitely many steps. This property is known as *refutational completeness*. In fact, the mentioned calculus would constitute a semi decision procedure for unsatisfiable sentences.

Indeed, *hierarchic superposition* is refutationally complete for all sets of (hierarchic) clauses that are *sufficiently complete*, if the background theory is *compact* (cf. Theorem 24 in [4]). In the case of the universal fragment of Presburger arithmetic without uninterpreted constant or function symbols, the two requirements are satisfied. Sufficient completeness (cf. Definition 20 in [4]) concerns uninterpreted constant or function symbols that reach into the background sort $\mathbb{N}$. Since we do not allow such symbols in our language, all sets of sentences are sufficiently complete. For the same reason, the background theory (over universal sentences built from the signature $\langle 0, 1, + \rangle$) is compact. This means, every unsatisfiable set of universal first-order sentences over $\langle 0, 1, + \rangle$ has some finite subset that is unsatisfiable. Indeed, every unsatisfiable set of such sentences has an unsatisfiable subset that contains exactly one sentence. Hence, the following proposition holds.

Proposition 8. The set of unsatisfiable sentences over the universal fragment of Presburger arithmetic with additional uninterpreted predicate symbols (not necessarily unary ones) is recursively enumerable.

From the literature on the *arithmetical hierarchy* (see, e.g. [36, 37, 30]) we know the following.

Proposition 9.

(i) The set Σ_1^0 captures exactly the recursively enumerable sets.

(ii) The set Π_1^0 captures exactly the sets whose complement is recursively enumerable.

(iii) The halting problem for (ordinary) Turing machines is Σ_1^0-complete.

Proof. (i) and (ii) are reformulations of Theorems II.1.2 and IV.1.3 in [37], respectively. (iii) combines the following parts of [37]: Definitions I.3.1, I.4.1, I.4.5, Theorem II.4.2 and the discussion after Definition IV.2.1 on page 64. □

Since we have completed a chain of reductions from the halting problem of Turing machines via the halting problem of two-counter machines to the unsatisfiability problem of the universal fragment of Presburger arithmetic with uninterpreted predicate symbols, we conclude Σ_1^0-completeness of the latter problem by Lemma 2 together with Propositions 8 and 9.

Theorem 10. The set of unsatisfiable sentences over the universal fragment of Presburger arithmetic with additional uninterpreted predicate symbols is Σ_1^0-complete.

It is worth to notice that the theorem can be translated to the realm of linear arithmetic over the reals. The reason is that hierarchic superposition is also refutationally complete over the universal fragment of this language, if there are no uninterpreted constant or function symbols involved.

Since any reduction of a problem S to a problem T (both read as a set of Gödel numbers) at the same time yields a reduction from $\overline{S}$ to $\overline{T}$, the complement of a Σ_1^0-complete set is complete for Π_1^0. Hence, Theorem 10 entails Π_1^0-completeness of the set of satisfiable sentences over the same language.

There are strong ties between (un)satisfiability in the universal fragment of the language we consider and (in)validity in the dual language, the existential fragment. The bottom line is that the obtained completeness results can be transfered to the corresponding (in)validity problems. The overall situation is depicted in Table 1.

For the sake of completeness, we briefly discuss (un)satisfiability for the existential fragment. Kruglov and Weidenbach [26] have presented a general result regarding the satisfiability problem for hierarchic clause sets that are ground. More precisely, they have devised a decision procedure for the problem that is based on a hierarchic superposition calculus.

Proposition 11 (corollary of Theorem 23 from [26])**.** Satisfiability of the existential fragment of Presburger arithmetic with additional uninterpreted predicate symbols is decidable.

With this knowledge we can complete the overview in Table 1 and thus reveal the full picture of where the (un)satisfiability and (in)validity problems of the universal and existential fragments of Presburger arithmetic augmented with uninterpreted predicate symbols reside in the arithmetical hierarchy.

4 One $\forall\exists$ quantifier alternation leads to Σ_1^1-completeness

Halpern has shown that the satisfiability problem for Presburger arithmetic with any choice of additional uninterpreted function symbols and predicate symbols lies in Σ_1^1 (Theorem 3.1 in [23]). This result is independent of the number of occurring quantifier alternations. In the present section, we show that already a single quantifier alternation suffices to make the problem complete for Σ_1^1. We leverage the following result, due to Alur and Henzinger.

	Satisfiability	Unsatisfiability	Validity	Invalidity
$\forall^*$-fragment	Π^0_1-complete	Σ^0_1-complete	Σ^0_0	Σ^0_0
$\exists^*$-fragment	Σ^0_0	Σ^0_0	Σ^0_1-complete	Π^0_1-complete

Table 1: Overview regarding the degree of unsolvability of the (un)satisfiability and (in)validity problems for the purely universal and purely existential fragment of Presburger arithmetic with additional uninterpreted predicate symbols. Notice that membership in Σ^0_0 (which coincides with Π^0_0) entails decidability of the respective problem.

Proposition 12. [Lemma 8 in [2]] The problem of deciding whether a given nondeterministic two-counter machine has a recurring computation is Σ^1_1-hard.

A *nondeterministic two-counter machine* differs from the deterministic model described in Section 2 in that it allows nondeterministic branching after a program line has been executed. This means that after the execution of a program line j (which does not result in a jump induced by a `test&dec` instruction) the machine does not necessarily proceed to the $(j+1)$-st line, but may have the choice between two specified options.

This kind of nondeterminism can easily be incorporated into the encoding presented in Section 3. For instance, the nondeterministic version of the instruction $j : \texttt{inc}(C_1)$ can be represented by the formula

$$\begin{aligned} &\forall xyz.\, \psi_{001011}(x) \ \wedge\ 2x \leq y \ \wedge\ y \leq 3x \ \wedge\ \psi_{10}(y) \ \wedge\ 3x \leq z \ \wedge\ z \leq 4x \ \wedge\ \psi_{10}(z) \ \wedge\ \chi_j(x) \\ &\qquad \longrightarrow\ \ \psi_{10}(6x+y+1) \ \wedge\ \psi_{10}(9x+z) \ \wedge\ \big(\chi_{j'}(4x) \vee \chi_{j''}(4x)\big)\ . \end{aligned}$$

The last conjunct $\big(\chi_{j'}(4x) \vee \chi_{j''}(4x)\big)$ now offers a choice between program lines j' and j'' as the ones that are to be executed next.

Consequently, we can reuse major parts of our encoding in order to prove Σ^1_1-hardness. For any nondeterministic two-counter machine $\mathcal{M}$ we write $\varphi'_{\mathcal{M}}$ to address the encoding of $\mathcal{M}$'s program in accordance with Section 3 and the just described adaptations due to the nondeterministic setting.

A computation performed by a nondeterministic two-counter machine is considered to be *recurring* if and only if it starts with both counters set to zero and if it reaches the program line with address 0 infinitely often. This means, we have to remove φ_4 from the encoding set of sentences and replace it with a proper formalization of the recurrence condition:

$$\varphi'_5 := \ \ \forall x \exists y.\, x \leq y \ \wedge\ \psi_{001011}(y) \ \wedge\ \chi_0(y)\ .$$

φ'_5 formulates recurrence in a positive fashion by saying that at any point in time program line 0 will be reached eventually. Formally speaking, φ'_5 exhibits a $\forall\exists\exists$ quantifier prefix, since $x \leq y$ is an abbreviation for $\exists z.\, x + z = y$.

Finally, in order to account for the specific input requirements posed in the definition of recurrence, we construct φ'_2 from φ_2 by setting $m = n = 0$, i.e.

$$\varphi'_2 := \ \ \psi_{10}(2d+3) \wedge \psi_{10}(3d+3) \wedge \chi_0(d)\ .$$

Lemma 13. The nondeterministic two-counter machine $\mathcal{M}$ has a recurring run if and only if $\varphi_1 \wedge \varphi'_2 \wedge \varphi_3 \wedge \varphi'_{\mathcal{M}} \wedge \varphi'_5$ is satisfiable.

By Proposition 12, this yields Σ^1_1-hardness. Due to the result by Halpern [23], we know that the set of satisfiable Presburger arithmetic sentences with additional uninterpreted predicate symbols lies in Σ^1_1. Hence, the following theorem holds.

Theorem 14. The set of satisfiable sentences of the $(\forall^* \wedge \forall\exists^2)$-fragment of Presburger arithmetic with a single additional uninterpreted unary predicate symbol is Σ^1_1-complete.

Due to the strong relations between satisfiability in one fragment and invalidity in its dual, the above theorem entails Π^1_1-completeness of the set of invalid sentences in the $(\exists^* \vee \exists\forall^2)$-fragment.

Moreover, notice that the theorem can be reformulated in terms of uninterpreted unary functions instead of uninterpreted unary predicates. However, in contrast to Theorem 6, we lose the property that the encoding results in a set of Horn clauses when transformed into CNF. The reason is the involved nondeterminism and the way we have encoded nondeterministic branching.

Over the domain of the reals, we can only show Σ^1_1-hardness of the satisfiability problem, since Halpern's upper bound does only cover the realm of the natural numbers.

Theorem 15. The set of satisfiable sentences of the $(\forall^* \wedge \forall\exists^2)$-fragment of linear arithmetic over the reals with a single additional uninterpreted unary predicate symbol is Σ^1_1-hard.

5 On the relevance to verification

In verification one usually abstracts from some of the limitations that apply to real-world computing devices. In particular, memory is often regarded as an infinite resource in one way or another. This can be due to infinitely many memory locations—similarly to the infinite tape of a Turing machine—or due to the capability of storing arbitrarily large integers in one memory location—similarly to the counters of counter machines.

In our encoding of two-counter machines the uninterpreted predicate symbol P serves as a representation of an unbounded memory. As we have pointed out, any interpretation $P^{\mathcal{I}} \subseteq \mathbb{N}$ can be conceived as an infinite sequence of bits. And these bits can be accessed by integer addresses. We have also pointed out in Section 3.5 that the same applies to uninterpreted function symbols over the integers or some co-domain with at least two distinct elements.

Clearly, this means that our results are relevant to all verification approaches in which an infinite memory is modeled and in which there are sufficiently strong means available to access individual memory locations. We shall discuss several exemplary settings: separation logic over an integer-indexed heap, logics formalizing integer-indexed arrays or similar data structures, logics with restricted forms of linear integer arithmetic.

Verification is one driving force behind attempts to the combination of theories, such as integer or real arithmetic and the *theory of equality over uninterpreted functions (EUF)*.[5] For quantifier-free cases the Nelson-Oppen procedure [29] provides a general-purpose framework that yields a decision procedure for combined theories from decision procedures for the constituent theories. Over the course of the last decade numerous approaches have been proposed to go beyond the quantifier-free setting and handle quantification [16, 12, 19, 20, 6, 35, 34]. Typically, some kind of heuristic is applied to guide instantiation to ground formulas. Often the methods are incomplete in the sense that unsatisfiable sentences are not necessarily recognized as such. Nevertheless, the proposed methods have been implemented and successfully applied, e.g. in the tools Verifun, Simplify, and the CVC family.

These approaches inevitably face undecidability when they allow too liberal syntax that combines arithmetic with uninterpreted function or predicate symbols. The hardness results presented in this paper draw a sharp line around what is possible in such settings. In the remaining sections, we give reasons why incomplete heuristics is sometimes the best one can expect.

5.1 Separation logic

In [33] the Bernays–Schönfinkel–Ramsey fragment ($\exists^*\forall^*$-sentences) of separation logic is investigated. The quantifiers range over memory locations. Although Reynolds, Iosif, and Serban also present a refinement of Halpern's undecidability result [23] for Presburger arithmetic with an additional uninterpreted unary predicate symbol, their approach differs from ours in an important aspect. In their setting it is sufficient to consider models wherein the unary predicate is interpreted over *finite* subsets of $\mathbb{N}$. In our case finite subsets do not suffice. It is due to this difference,

[5] For references see, e.g., Chapter 10 in [9], or Chapters 10 and 12 in [25].

that their strategy can be used to also show undecidability of the satisfiability problem for $\exists^*\forall^*$-sentences of separation logic over a heap with *finitely* many integer-indexed memory locations, each capable of storing one integer of arbitrary size.

Our results in Sections 3 and 4 have implications for settings with integer-indexed heaps that comprise a countably infinite number of memory locations, each capable of distinguishing at least two values (e.g. 0 and 1) or states (e.g. *allocated* and *not allocated*). However, a slight modification of the encoding in Section 3.2 leads to a result that subsumes Theorem 3 in [33] and also entails undecidability of the satisfiability problem for the $\exists^*\forall^*$-fragment of separation logic with integer-indexed heaps that comprise finitely many memory locations, each capable of storing at least one bit of information.

Lemma 16. Let $\mathcal{M}$ be a two-counter machine with $K+1$ program lines, labeled $0, \ldots, K$, and let $\langle m, n\rangle$ be a pair of nonnegative integers. There is a sentence φ from the $(\exists\forall^*)$-fragment of Presburger arithmetic with an additional unary predicate symbol P, such that the following statements are equivalent:

(a) φ is satisfied by an interpretation $\mathcal{I}$ under which $P^{\mathcal{I}}$ is a finite subset of $\mathbb{N}$,

(b) $\mathcal{M}$ reaches the `halt` instruction when started on the input $\langle m, n\rangle$.

Proof sketch. Let $\varphi''_{\mathcal{M}}$ be the encoding of $\mathcal{M}$'s program in accordance with Section 3.2 with the exception that we do not encode the instruction in program line K. Due to our conventions, this program line contains the `halt` instruction.

Let $\varphi''_1(z)$ be the result of replacing the subformula (3) in φ_1 with

$$\forall x.\ x < z \ \wedge\ \psi_{001011}(x) \ \longrightarrow\ \psi_{0011}(2x) \ \wedge\ \psi_{0011}(3x) \ \wedge\ \psi_{001011}(4x)\ .$$

Moreover, let

$$\varphi''_4(z) := \ \psi_{001011}(z) \wedge \chi_K(z)\ .$$

Notice that both formulas $\varphi''_1(z)$ and $\varphi''_4(z)$ contain the free variable z. We now set

$$\varphi := \ \exists z.\ \varphi''_1(z) \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi''_{\mathcal{M}} \wedge \varphi''_4(z)\ .$$

There exists a model $\mathcal{I}'$ of φ if and only if $\mathcal{M}$ reaches program line K when started on the input $\langle m, n\rangle$. Due to the modifications in φ''_1, the formula $\psi_{001011}(x)$ does not have to be satisfied for arbitrarily large values of x. One consequence is that the run of $\mathcal{M}$ represented by a model of φ can be aborted at the point when program line K is reached. This means, in contrast to the proof of Lemma 2, we do not have to artificially continue $\mathcal{M}$'s run beyond that point. Hence, any model of φ can be modified in such a way that from a certain point on the bit sequence represented by the interpretation of P contains only zeros. □

5.2 Verification of data structures

There are undecidability results in the context of verification of programs that use integer-indexed arrays as data structures. Examples can be found in [10] (Section 5), [8] (Sections 2.4 and 2.6.3), [22] (Section 3). The reductions presented therein are based on arrays with infinite co-domains, such as the integers or the reals. Moreover, they typically use at least one quantifier alternation (but face other restrictions of syntax). Usually, several arrays are used for convenience, but could be merged into one. For our proof approach a single array is sufficient as well.

Read operations on integer-indexed arrays can be formalized as uninterpreted functions with an integer domain. Hence, our results, Theorems 6 and 7 in particular, show that reasoning about integer- or real-indexed arrays over a *finite co-domain* with at least two elements can lead to undecidability, if constraints on array indices allow the necessary syntactic means. Notice that for the proof it is not necessary to have write operations on arrays. This means, a single integer-indexed read-only array over a Boolean co-domain suffices.

The mentioned results and arguments hold for arrays that comprise an infinite number of elements. However, due to Lemma 16, undecidability arises also in the context of finite arrays (over finite co-domains), as long as their length is not bounded by a concrete number.

Remark 17. The above arguments are also applicable to recursively defined data structures, such as lists or trees, as soon as there are sufficiently strong syntactic means available to access the stored information. This means, if one can essentially simulate arrays using a recursive data structure, then our results apply immediately. Examples of such setting are lists where the stored elements can be addressed by integers, or where one can access the sublist starting at the position that is x nodes away from the head (for some integer-sort variable x that may be universally quantified).

5.3 Verification using counter arithmetic

In [11] the fragment *CLU* is introduced, which constitutes a strongly restricted fragment of Presburger arithmetic with additional uninterpreted function and predicate symbols. A less syntactically sugared subfragment is treated in [17] and in [3]. Regarding arithmetic, there are only two operators available in CLU: the successor operator `succ` and the predecessor operator `pred`. The language of CLU does not contain an interpreted constant that addresses zero. On the other hand, some syntactic elements are added for convenience, such as lambda abstraction and an `if-then-else` operator. The fragment was chosen for its expressiveness and the fact that it facilitates efficient reasoning. Although quantifier-free in its original definition, the authors state about their verification tool *UCLID* that they "have built some support for quantifiers in CLU using automatic quantifier instantiation heuristics" ([11], Section 7).

In what follows, we consider the extension of CLU by universal quantification of integer variables. We shall refer to this extended language as *uCLU*. By a result due to Gurevich [21] (see also [7], Theorems 4.1.8 and 4.1.11), satisfiability of EUF sentences with universal quantification is undecidable. Hence, satisfiability of uCLU sentences is undecidable as well.

Proposition 18 (corollary of the Main Theorem in [21])**.** (Un)satisfiability for uCLU sentences is undecidable.

On the other hand, the unsatisfiable sentences of pure first-order logic (and thus also of quantified EUF) are recursively enumerable. We next argue that uCLU does not possess this property.

The encoding of two-counter machines from Section 3 and 4 cannot immediately be translated into uCLU. First of all, we need to fix a point of reference that serves as zero (CLU does not contain 0 as a built-in constant). Moreover, expressions of the form kx for some integer k and some integer-sort variable x require a form of addition that is not available as a built-in operation in uCLU. However, with unrestricted universal quantification over integer variables at hand, we can easily define addition as a function. Hence, we need only the following uninterpreted symbols to encode two-counter machines: one constant c_0 serving as zero, one binary function realizing addition, one uninterpreted unary function or predicate symbol serving as memory.

We define the addition function as follows, where we use c_0 as zero:

$$\begin{array}{lll} \forall x. & & \mathrm{add}(x, c_0) = x \\ \forall xy. & \mathrm{succ}(y) > c_0 \longrightarrow & \mathrm{add}\big(x, \mathrm{succ}(y)\big) = \mathrm{add}\big(\mathrm{succ}(x), y\big) \\ \forall xy. & \mathrm{succ}(y) < c_0 \longrightarrow & \mathrm{add}\big(x, \mathrm{succ}(y)\big) = x \ . \end{array}$$

All constants that we use in the encoding shall be written as $\mathrm{succ}^k(c_0)$ instead of just k. Moreover, we add guards $x \geq c_0 \rightarrow \ldots$ to each sentence for every universally quantified variable x that occurs in the sentence.

As we have seen in Section 4, in particular in Theorem 14, $\forall\exists$ quantifier alternations lead to (un)satisfiability problems that are not even recursively enumerable. Since CLU allows uninterpreted function symbols, uCLU essentially allows $\forall^*\exists^*$ quantifier prefixes. Hence, we may introduce a fresh unary Skolem function f_{init} and translate the sentence φ_5' from Section 4 into the uCLU formula

$$\forall x.\, x \geq 0 \ \longrightarrow \ x \leq f_{\mathrm{init}}(x) \ \wedge \ \psi_{001011}\big(f_{\mathrm{init}}(x)\big) \ \wedge \ \chi_0\big(f_{\mathrm{init}}(x)\big) \ .$$

This means, we can transfer Theorem 14 to uCLU and thus obtain the following result.

Proposition 19. Neither the set of satisfiable uCLU sentences nor the set of unsatisfiable uCLU sentences is recursively enumerable. In particular, there cannot be any sound and refutationally complete calculus for uCLU.

In [3] the authors present a combination result (Theorem 4.6) for the ground theories of *integer-offsets* (the arithmetic subfragment of CLU embodied by the operators `succ` and `pred`), arrays, and/or EUF (as long as the signature of uninterpreted functions does not contain the array sort). The result states that the satisfiability of sentences in such combined theories can be decided using term-rewriting methods. By a similar line of argument that led us to Proposition 19, it follows that Theorem 4.6 in [3] cannot be generalized to cases which admit quantification over integer-sort variables. But we do not only lose decidability, we also lose semi-decidability. In other words, it is impossible to devise sound and complete calculi for combinations of EUF and arithmetic—even in such a restricted form as in CLU—if universal quantification of integer variables is available.

Remark 20. Note that lists plus an append, a length operator, and a $<$ predicate can be used to simulate natural numbers with addition.[6] Hence, combining such a theory with EUF leads to undecidability, if universal quantification is allowed.

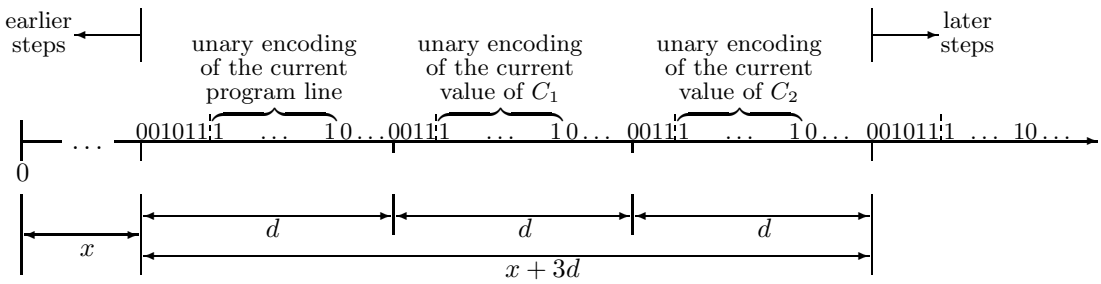
Similarly, sets with disjoint union (or standard union plus a disjointness predicate or a membership predicate) and a cardinality function can simulate natural numbers with addition.

5.4 Almost uninterpreted formulas with offsets

In [20] Ge and de Moura define the fragment of *almost uninterpreted formulas*. It constitutes a combination of subfragments of first-order logic, EUF, and linear arithmetic over the integers. Its language admits uninterpreted predicate symbols, function symbols and constant symbols. Formulas are assumed to be given in CNF. All variables are universally quantified, but may only occur as arguments of uninterpreted function or predicate symbols with the following exceptions. Literals of the form $\neg(x \leq y)$, $\neg(x \leq t)$, $\neg(x \geq t)$, $\neg(x = t)$, $\neg(x \leq y + t)$, $x = t$ with integer-sort variables x, y are allowed for all ground terms t of the integer sort. Moreover, terms of the form $f(\ldots, x + t, \ldots)$ and $P(\ldots, x + t, \ldots)$ are allowed for ground terms t of the integer sort, function symbols f and predicate symbols P. In what follows we shall be more liberal with the syntax than this. However, the formulas that we will present can be rewritten into equivalent ones that obey the above restrictions.

The encoding of two-counter machines in Section 3.2 requires different syntactic means than the ones available in Ge and de Moura's *almost uninterpreted* fragment. Hence, a proof of undecidability in the syntax of [20] needs a slight shift of paradigm. We start from the reduced form outlined in Section 3.3, since this encoding requires at most two integer-sort variables in atomic constraints. In our previous encodings the length of the chunks (substrings) storing a single configuration $\langle \ell, c_1, c_2 \rangle$ increases over time. This behavior is necessary to formalize non-terminating runs—and recurring runs in particular—by satisfiable formulas. However, in order to formalize a run that eventually reaches the `halt` instruction by a satisfiable sentence, it suffices to fix the length of the chunks representing a single configuration to a size that can accommodate all configurations that occur in the run, depending on the machine program and on the given input. In Ge and de Moura's fragment uninterpreted constant symbols are available that can be used for this purpose. In what follows, the uninterpreted constant d is used to determine the length of subchunks, as depicted below.

[6] Alternatively, the relation $<$ could be defined using equality testing on lists and existential quantification over list—in the same manner as we have defined $<$ in Presburger arithmetic in Section 2.1.



Moreover, we now start the encoding of the run at the very first bit of the bit string represented by P. We replace the formula φ_1 by the following, somewhat simpler formula φ_1'''. Let k be the result of the expression $\max(K+6, m+4, n+4)$, where K is the address of the last program line and m and n are the input values.

$$\begin{aligned}
\varphi_1''' := \;& d \geq k \;\wedge\; e \geq 0 \\
& \wedge\; \psi_{001011}(0) \;\wedge\; \psi_{0011}(d) \;\wedge\; \psi_{0011}(2d) \;\wedge\; \big(\forall x.\, x < 0 \;\longrightarrow\; \neg P(x)\big) \\
& \wedge\; \big(\forall x.\, \psi_{001011}(x) \;\wedge\; x < 3d \;\wedge\; x \neq 0 \;\longrightarrow\; \bot\big) \\
& \wedge\; \big(\forall x.\, \psi_{0011}(x) \;\wedge\; x < 3d \;\wedge\; x \neq d \;\wedge\; x \neq 2d \;\longrightarrow\; \bot\big) \\
& \wedge\; \big(\forall x.\, \psi_{001011}(x) \;\wedge\; x < e \;\longrightarrow\; \psi_{001011}(x+3d)\big) \\
& \wedge\; \big(\forall x.\, \psi_{001011}(x+3d) \;\wedge\; x \geq 0 \;\longrightarrow\; \psi_{001011}(x)\big) \\
& \wedge\; \big(\forall x.\, \psi_{0011}(x) \;\wedge\; x < e \;\longrightarrow\; \psi_{0011}(x+3d)\big) \\
& \wedge\; \big(\forall x.\, \psi_{0011}(x+3d) \;\wedge\; x \geq 0 \;\longrightarrow\; \psi_{0011}(x)\big)
\end{aligned}$$

The purpose of the uninterpreted constant e is to mark the end of the run, as we will see later.

The sentences φ_2 and φ_3 can be adapted in the same spirit:

$$\begin{aligned}
\varphi_2''' :=\;& \chi_0(0) \;\wedge\; \psi_{10}(d+3+m) \;\wedge\; \psi_{10}(2d+3+n) \\
\varphi_3''' :=\;& \forall xy.\, \psi_{001011}(x) \;\wedge\; \psi_{10}(y) \;\wedge\; x+5+K < y \;\wedge\; y \leq x+d \;\longrightarrow\; \bot\,.
\end{aligned}$$

The adapted encoding of an instruction `inc(`C_1`)` comprises the formulas

$$\begin{aligned}
& \forall xy.\, \psi_{001011}(x) \;\wedge\; x+d \leq y \;\wedge\; y \leq x+2d \;\wedge\; \psi_{10}(y) \;\wedge\; \chi_j(x) \\
& \qquad\qquad\qquad\qquad\qquad\qquad \longrightarrow\; \psi_{10}(y+3d+1) \;\wedge\; \chi_{j+1}(x+3d) \\
& \forall xz.\, \psi_{001011}(x) \;\wedge\; x+2d \leq z \;\wedge\; z \leq x+3d \;\wedge\; \psi_{10}(z) \;\wedge\; \chi_j(x) \;\longrightarrow\; \psi_{10}(z+3d)
\end{aligned}$$

The other instructions can be adapted analogously. The only exception is the `halt` instruction in the last program line which we shall not encode, as in the proof sketch for Lemma 16.

Finally, we also have to modify the condition that the two-counter machine halts at some point in time. We use another uninterpreted constant e for this purpose:

$$\varphi_4''' := \;\; \psi_{001011}(e) \;\wedge\; \chi_K(e)\,.$$

Consequently, using the fragment given in [20], we can encode the halting problem of a two-counter machine $\mathcal{M}$ on input $\langle m, n\rangle$ using only a single uninterpreted unary predicate symbol P (or a single function symbol) plus two uninterpreted constant symbols d, e. More precisely, if $\mathcal{M}$ halts on $\langle m, n\rangle$, then there is model $\mathcal{I}$ of the encoding sentence such that $P^{\mathcal{I}}$ is a finite set of integers.

The outlined formalization is sufficient for a halting run of a two-counter machine. However, we cannot formalize recurring counter machines in this way. Thus, we do not obtain hardness beyond recursive enumerability. Indeed, this is in line with [20], where a refutationally complete calculus is given for the described fragment.

The realm of recursive enumerability can be left easily. For instance, it is sufficient to allow scalar multiplication combined with addition for integer-sort variables, i.e. expressions of the form

$kx + y$ for integers k. Similarly, it would suffice to admit expressions $g(x) + y$, as we can define, e.g.,

$$\text{times}_k(0) = 0 \;\wedge\; \forall x.\, x \geq 0 \;\rightarrow\; \text{times}_k(x+1) = \text{times}_k(x) + k$$

for any positive integer k. With a syntax extended this way, one could realize the encoding from Section 3.2.

6 Conclusion

In this paper we have sharpened the known undecidability results for the language of Presburger arithmetic augmented with uninterpreted predicate or function symbols. We have shown that already the purely universal fragment of such extended languages yields an undecidable satisfiability problem. More precisely, we have shown Σ_1^0-completeness of the corresponding set of unsatisfiable sentences. In the case of extensions by uninterpreted function symbols, the fragment can even be restricted to Horn clauses while retaining an undecidable satisfiability problem. Moreover, we have strengthened Halpern's Σ_1^1-hardness result (Theorem 3.1 in [23]) in that we have shown that a single $\forall\exists$ quantifier alternation suffices for a proof. More precisely, the satisfiability problem for $\forall^*\exists^2$-sentences of this extended language is Σ_1^1-hard.

In addition, we have transfered the mentioned undecidability and hardness results to the realm of linear arithmetic over the ordered real numbers augmented with a single uninterpreted predicate symbol.

Concerning automated reasoning, we have mentioned in Section 3.6 that there are refutationally complete deductive calculi for the purely universal and purely existential fragments of Presburger arithmetic with uninterpreted predicate symbols. In the existential case even decision procedures exist. On the other hand, the hardness result presented in Section 4 entails that there cannot be sound and refutationally complete calculi that can handle $\forall\exists$ quantifier alternations, if the problem is not restricted any further. The same applies to fragments allowing unrestricted combinations of universal quantification and function symbols, as Skolem functions are at least as powerful as existential quantifiers in this context.

Apart from their theoretical value, the results presented in this paper are relevant for several areas of verification. In Section 5 we have elaborated on the implications for the Bernays–Schönfinkel fragment ($\exists^*\forall^*$-sentences) of separation logic, quantified theories of data structures, arrays in particular, and quantified combinations of the theory of equality over uninterpreted functions with strongly restricted fragments of Presburger arithmetic. Moreover, we have argued that in certain settings we cannot even hope for refutationally complete deductive calculi. In such cases we either have to content ourselves with heuristics instead of sound and complete methods or formulate restricted fragments that lead to less hard (un)satisfiability problems.

Acknowledgments

The authors thank Radu Iosif and Stanislav Speranski for inspiring and enlightening discussions.

References

[1] Ernst Althaus, Evgeny Kruglov, and Christoph Weidenbach. Superposition modulo linear arithmetic SUP(LA). In *Frontiers of Combining Systems (FroCoS'09)*, pages 84–99, 2009.

[2] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.

[3] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1), 2009.

[4] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Refutational theorem proving for hierarchic first-order theories. *Applicable Algebra in Engineering, Communication and Computing*, 5:193–212, 1994.

[5] Peter Baumgartner and Uwe Waldmann. Hierarchic superposition with weak abstraction. In *Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2013.

[6] Nikolaj Bjørner, Kenneth L. McMillan, and Andrey Rybalchenko. On solving universally quantified horn clauses. In *Static Analysis (SAS'13)*, pages 105–125, 2013.

[7] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

[8] Aaron R. Bradley. *Safety Analysis of Systems*. PhD thesis, 2007.

[9] Aaron R. Bradley and Zohar Manna. *The Calculus of Computation*. Springer, 2007.

[10] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In *Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, pages 427–442, 2006.

[11] Randal E. Bryant, Shuvendu K. Lahiri, and Sanjit A. Seshia. Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. In *Computer Aided Verification (CAV'02)*, pages 78–92, 2002.

[12] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[13] Peter J. Downey. Undecidability of presburger arithmetic with a single monadic predicate letter. Technical report, Center for Research in Computer Technology, Harvard University, 1972.

[14] Herbert B. Enderton. *A mathematical introduction to logic*. Harcourt/Academic Press, 2001.

[15] Michael Jo Fischer and Michael O. Rabin. Super-exponential complexity of presburger arithmetic. In *SIAM-AMS Symposium in Applied Mathematics*, pages 27–41, 1974.

[16] Cormac Flanagan, Rajeev Joshi, and James B. Saxe. An explicating theorem prover for quantified formulas. Technical Report HPL-2004-199, HP Laboratories Palo Alto, 2004.

[17] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): fast decision procedures. In *Computer Aided Verification (CAV'04)*, pages 175–188, 2004.

[18] Solomon Garfunkel and James H. Schmerl. The undecidability of theories of groupoids with an extra predicate. *Proceedings of the American Mathematical Society*, 42(1):286–289, 1974.

[19] Yeting Ge, Clark W. Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Ann. Math. Artif. Intell.*, 55(1-2):101–122, 2009.

[20] Yeting Ge and Leonardo Mendonça de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *Computer Aided Verification (CAV'09)*, pages 306–320, 2009.

[21] Yuri Gurevich. The decision problem for standard classes. *J. Symb. Log.*, 41(2):460–464, 1976.

[22] Peter Habermehl, Radu Iosif, and Tomás Vojnar. What else is decidable about integer arrays? In *Foundations of Software Science and Computational Structures (FOSSACS'08)*, pages 474–489, 2008.

[23] Joseph Y. Halpern. Presburger arithmetic with unary predicates is Π^1_1 complete. *Journal of Symbolic Logic*, 56(2):637–642, 1991.

[24] David Harel, Amir Pnueli, and Jonathan Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983.

[25] Daniel Kroening and Ofer Strichman. *Decision Procedures*. Texts in Theoretical Computer Science. An EATCS Series. Springer, second edition, 2016.

[26] Evgeny Kruglov and Christoph Weidenbach. Superposition decides the first-order logic fragment over ground theories. *Mathematics in Computer Science*, 6(4):427–456, 2012.

[27] V. A. Lifshits. Some reduction classes and undecidable theories. In *Studies in Constructive Mathematics and Mathematical Logic*, volume 4 of *Seminars in Mathematics*, pages 24–25. Steklov Mathematical Institute, 1969.

[28] Marvin Lee Minsky. *Computation: finite and infinite machines*. Prentice-Hall, 1967.

[29] Greg Nelson and Derek C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, October 1979.

[30] Piergiorgio Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1992.

[31] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Sprawozdanie z I Kongresu matematyków krajów słowiańskich, Warszawa*, pages 92–101, 1929. See [40] for an English translation.

[32] Hilary Putnam. Decidability and essential undecidability. *J. Symb. Log.*, 22(1):39–54, 1957.

[33] Andrew Reynolds, Radu Iosif, and Cristina Serban. Reasoning in the Bernays–Schönfinkel–Ramsey Fragment of Separation Logic. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'17)*, pages 462–482, 2017.

[34] Andrew Reynolds and Viktor Kuncak. Induction for SMT solvers. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'15)*, pages 80–98, 2015.

[35] Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura. Finding conflicting instances of quantified formulas in SMT. In *Formal Methods in Computer-Aided Design (FMCAD'14)*, pages 195–202, 2014.

[36] Hartley Rogers. *Theory of recursive functions and effective computability (Paperback reprint from 1967)*. MIT Press, 1987.

[37] Robert Irving Soare. *Recursively Enumerable Sets and Degrees*. Springer, 1987.

[38] Stanislav O. Speranski. Collapsing probabilistic hierarchies. I. *Algebra and Logic*, 52(2):159–171, 2013.

[39] Stanislav O. Speranski. A note on definability in fragments of arithmetic with free unary predicates. *Arch. Math. Log.*, 52(5-6):507–516, 2013.

[40] Ryan Stansifer. Presburger's article on integer arithmetic: Remarks and translation. Technical Report TR84-639, Cornell University, Computer Science Department, 1984.