What's Decidable About Sequences?

Carlo A. Furia

January 2010

Abstract

We present a first-order theory of sequences with integer elements, Presburger arithmetic, and regular constraints, which can model significant properties of data structures such as arrays and lists. We give a decision procedure for the quantifier-free fragment, based on an encoding into the first-order theory of concatenation; the procedure has PSPACE complexity. The quantifier-free fragment of the theory of sequences can express properties such as sortedness and injectivity, as well as Boolean combinations of periodic and arithmetic facts relating the elements of the sequence and their positions (e.g., "for all even i's, the element at position i has value $i + 3$ or $2i$"). The resulting expressive power is orthogonal to that of the most expressive decidable logics for arrays. Some examples demonstrate that the fragment is also suitable to reason about sequence-manipulating programs within the standard framework of axiomatic semantics.

Contents

1 Introduction

Verification is undecidable already for simple programs, but modern programming languages support a variety of sophisticated features that make it all the more complicated. These advanced features — such as arrays, pointers, dynamic allocation of resources, and object-oriented abstract data types — are needed because they raise the level of abstraction thus making programmers more productive and programs less buggy. Verification techniques have also progressed rapidly over the years, in an attempt to keep the pace with the development of programming languages.

Automated verification requires expressive program logics and powerful decision procedures. In response to the evolution of modern programming languages, new decidable program logic fragments and combination techniques for different fragments have mushroomed especially in recent years. Many of the most successful contributions have focused on verifying relatively restricted aspects of a program's behavior, for example by decoupling pointer structure and functional properties in the formal analysis of a dynamic data structure. This narrowing choice, partly deliberate and partly required by the formidable difficulty of the various problems, is effective because different aspects are often sufficiently decoupled so that each of them can be analyzed in isolation with the most appropriate, specific technique.

This paper contributes to the growing repertory of special program logics by exploring the decidability of properties of *sequences of elements* of homogeneous type. These can abstract fundamental features of several data structures: arrays imprimis, but also the sequence of values stored in a dynamically allocated list, or the content of a stack or a queue.

We take a new angle on reasoning about sequences, based on the *theory of concatenation*: a first-order theory where variables are interpreted as words (or sequences) over a finite alphabet and can be composed by concatenating them. Makanin's algorithm for solving word equations [32] implies the decidability of the quantifier-free fragment of the theory of concatenation. Based on this, we introduce a first-order *theory of sequences* $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ whose elements are integers. Section 3.2 presents a decision procedure for the quantifier-free fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$, which encodes the validity problem into the quantifier-free theory of concatenation. The decision procedure is in PSPACE; it is known, however, that Makanin's algorithm is reasonably efficient in practice [1].

The theory of sequences $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ allows concatenating sequences to build new ones, and it includes Presburger arithmetic over elements of a sequence. On the other hand, it forbids explicit indexed access to elements, which differentiates it from the theory of arrays and extensions thereof (see Section 5). The resulting quantifier-free fragment has significant expressiveness, in spite of its limitations in representing subsequences of variable length. In particular, we show some interesting properties that are inexpressible in powerful decidable array logics (such as those in [8, 16, 22, 21]) but are expressible in our theory of sequences. Conversely, there exist decidable properties of extensions of the theory of arrays that are inexpressible in $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$. These results support our claim that the theory of sequences provides a fresh angle on reasoning about sequences, orthogonal to most approaches that model sequences as arrays.

In order to better assess the limits of our theory of sequences, we also prove that several natural extensions of the quantifier-free fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ are

undecidable. Finally, we demonstrate reasoning about sequence-manipulating programs with annotations written in the quantifier-free fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$. A couple of examples in Section 4 illustrate the usage of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ formulas with the standard machinery of axiomatic semantics and backward reasoning.

Paper outline. Section 2 presents the theory of concatenation and summarizes a few decidability and undecidability results about it. Section 3 introduces our theory of integer sequences $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$, demonstrates its expressiveness, provides a decision procedure for its quantifier-free fragment, and shows undecidable extensions of the theory. Section 4 illustrates how to use the theory $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ to reason about programs in the standard axiomatic semantics framework. Finally, Section 5 reviews related work and Section 6 concludes by outlining future work.

2 The Theory of Concatenation

This section introduces some basic notation (Section 2.1) and summarizes some results about the first-order theory of concatenation (Section 2.2) that we will use in the remainder of the paper.

In the rest of the paper, we assume familiarity with the standard syntax and terminology of first-order theories (e.g., [7]); in particular, we assume the standard abbreviations and symbols of first-order theories with the following operator precedence $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \forall$ and $\exists$.

$FV(\phi)$ denotes the set of free variables of a formula ϕ. With standard terminology, a formula ϕ is a *sentence* iff it is *closed* iff $FV(\phi) = \emptyset$. Given a regular expression $\mathcal{Q}$ over $\{\exists, \forall\}$, the $\mathcal{Q}$-fragment of a first-order theory is the set of all formulas of the theory in the form $\mathcal{Q} \bullet \psi$, where ψ is quantifier-free. The universal and existential fragments are synonyms for the $\forall^*$- and $\exists^*$-fragment respectively. A fragment is decidable iff the validity problem is decidable for its sentences. It is customary to define the validity and satisfiability problems for a quantifier-free formula ψ as follows: ψ is valid iff the universal closure of ψ is valid, and ψ is satisfiable iff the existential closure of ψ is valid. As a consequence of this definition, the decidability of a quantifier-free fragment whose formulas are closed under negation is tantamount to the decidability of the universal or existential fragments. Correspondingly, in the paper we will allow some freedom in picking the terminology that is most appropriate to the context.

2.1 Sequences and Concatenation

$\mathbb{Z}$ denotes the set of integer numbers and $\mathbb{N}$ denotes the set of nonnegative integers.

Given a set $A = \{a, b, c, \ldots\}$ of constants, a *sequence* over A is any word $v = v(1)v(2)\cdots v(n)$ for some $n \in \mathbb{N}$ where $v(i) \in A$ for all $1 \leq i \leq n$. The symbol ϵ denotes the *empty sequence*, for which $n = 0$. $|v| = n$ denotes the *length* of v. A^* denotes the set of all finite sequences over A including $\epsilon \notin A$. It is also convenient to introduce the shorthand $v(k_1, k_2)$ with $k_1, k_2 \in \mathbb{Z}$ to

describe *subsequences* of a given sequence v; it is defined as follows.

$$v(k_1,k_2) \triangleq \begin{cases} v(k_1)v(k_1+1)\cdots v(k_2) & 1 \le k_1 \le k_2 \le |v| \\ v(k_1, |v|+k_2) & k_1 - |v| \le k_2 < 1 \le k_1 \\ v(|v|+k_1, |v|+k_2) & 1-|v| \le k_1 \le k_2 < 1 \\ \epsilon & \text{otherwise} \end{cases}$$

For two sequences $v_1, v_2 \in A^*$, $v_1 \star v_2$ denotes their *concatenation*: the sequence $v_1(1)\cdots v_1(|v_1|)v_2(1)\cdots v_2(|v_2|)$. We will drop the concatenation symbol whenever unambiguous.

The structure $\langle A^*, \star, \epsilon\rangle$ is also referred to as the *free monoid* with generators in A and neutral element ϵ. The size $|A|$ is called *rank* of the free monoid and it can be finite or infinite.

2.2 Decidability in the Theory of Concatenation

2.2.1 Syntax and Semantics

The theory of concatenation is the first-order theory $\mathcal{T}_{\mathsf{cat}}$ with signature

$$\Sigma_{\mathsf{cat}} \quad \triangleq \quad \{\doteq, \circ, \mathcal{R}\}$$

where $\doteq$ is the equality predicate,[1] $\circ$ is the binary concatenation function and $\mathcal{R} \triangleq \{\mathrm{R}_1, \mathrm{R}_2, \ldots\}$ is a set of unary (monadic) predicate symbols called *regularity constraints*. We sometimes write $R_i(x)$ as $x \in R_i$ and $\alpha \not\doteq \beta$ abbreviates $\neg(\alpha \doteq \beta)$.

An interpretation of a formula in the theory of concatenation is a structure $\langle A^*, \star, \epsilon, \underline{\mathcal{R}}, ev\rangle$ where $\langle A^*, \star, \epsilon\rangle$ is a free monoid, $\underline{\mathcal{R}} = \{\underline{\mathrm{R}}_1, \underline{\mathrm{R}}_2, \ldots\}$ is a collection of regular subsets of A^*, and ev is a mapping from variables to values in A^*. The satisfaction relation $\langle A^*, \star, \epsilon, \underline{\mathcal{R}}, ev\rangle \models \phi$ for formulas in $\mathcal{T}_{\mathsf{cat}}$ is defined in a standard fashion with the following assumptions.

- any variable x takes the value $ev(x) \in A^*$;
- the concatenation $x \circ y$ of two variables x, y takes the value $ev(x) \star ev(y)$;
- for each $\mathrm{R}_i \in \mathcal{R}$, the corresponding $\underline{\mathrm{R}}_i \in \underline{\mathcal{R}}$ defines the set of sequences $x \in \underline{\mathrm{R}}_i$ for which $\mathrm{R}_i(x)$ holds (this also subsumes the usage of constants).

2.2.2 (Un)Decidable Fragments

The following propositions summarize some decidability results about fragments of the theory of concatenation; they all are known results, or corollaries of them. The standard presentation of these results focuses on solving equations over sequences with free variables and, correspondingly, on existential fragments of the equational theory. On the contrary, in this paper we will mostly focus on the universal fragment, given its aptness for annotating sequence-manipulating programs (see Section 4). It is straightforward, however, to rephrase the results in terms of the dual existential fragments, given the availability of negation in the language.

[1] We use the symbol $\doteq$ to distinguish it from the standard arithmetic equality symbol $=$ used later in the paper.

Proposition 1 (Decidability [32, 13, 38])**.** *The universal and existential fragments of the theory of concatenation over free monoids with finite rank are decidable in PSPACE.*

Proof. Decidability is a consequence of Makanin's seminal result on word equations [32] and its extensions to the full existential (and universal) fragments [10, 13]. PSPACE complexity is a consequence of Plandowski's recent results [38, 39] and the fact that transforming first-order formulas into a single word equation introduces only a polynomial blow-up.

The only catch is that the standard presentation assumes formulas in the canonical form $\forall x_1 \in \mathrm{R}_1, x_2 \in \mathrm{R}_2, \ldots, x_v \in \mathrm{R}_v \bullet \rho$ where regularity constraints do not appear in ρ. This is, however, without loss of generality as we can put any universal formula $\forall \overline{x} \bullet \psi$ in canonical form: first rewrite ψ into

$$\bigwedge_{\substack{1 \leq i \leq |\overline{x}| \\ 1 \leq j \leq |\mathcal{R}|}} (x_i \doteq h_j^+ \vee x_i \doteq h_j^-) \Rightarrow \psi$$

for fresh $h_j^+ \in \mathrm{R}_j$ and $h_j^- \in A^* \setminus \mathrm{R}_j$. Then, put ψ in negated normal form and eliminate occurrences of regularity predicates by applying exhaustively the rules:

$$\frac{\psi[\mathrm{R}_m(x_n)]}{\psi[x_n \doteq h_m^+]} \qquad \frac{\psi[\neg \mathrm{R}_m(x_n)]}{\psi[x_n \doteq h_m^-]}$$

It is not difficult to see that this transformation preserves satisfiability and introduces a blow-up which is quadratic at most. □

Proposition 2 (Undecidability)**.**

- *[14] The $\forall^*\exists^*$ and $\exists^*\forall^*$ fragments of the theory of concatenation are undecidable; in particular the $\forall\exists^3$-fragment is undecidable already for negation-free formulas.*
- *[11] The existential and universal fragments of the extension of the theory of concatenation over the free monoid $\{a, b\}^*$ with: (1) two length functions $|x|_a \triangleq \{y \in a^* \mid y \text{ has the same number of } a\text{'s as } x\}$ and $|x|_b \triangleq \{y \in b^* \mid y \text{ has the same number of } b\text{'s as } x\}$; or (2) the function $Sp(x) \triangleq |x|_a \star |x|_b$ are undecidable.*

A set of sequences $S \subseteq A^*$ is *universally (resp. existentially) definable from concatenation* iff there is a universal (resp. existential) formula $\varphi[x]$ with $FV(\varphi[x]) = \{x\}$ such that $S = \{y \in A^* \mid \varphi[y]\}$.

Proposition 3 (Definability [11])**.**

- *The set $S^= \triangleq \{a^n b^n \mid n \in \mathbb{N}\}$ is neither universally nor existentially definable from concatenation.*
- *The equal length predicate $Elg(x, y) \triangleq |x| = |y|$ is not definable in the existential and universal fragments of concatenation.*

Proof. Büchi and Senger prove in [11, Corollary 3] that $S^=$ is not existentially definable. A very similar argument shows that $S^< \triangleq \{a^m b^n \mid 0 \leq m < n\}$ is also not existentially definable (using the terminology of [11, Corollary 3], the spacers of $a^{i-1}b^i$ relative to the atom a are $\langle a, ab^i \rangle$ and there are only $i - 1$ words in $S^<$ with these spacers). The existential non-definability of $S^<$ entails the existential non-definability of the set $S^\neq \triangleq \{a^n b^m \mid 0 \leq n \neq m\}$

by contradiction as follows. Assume that $S^{\neq}$ were existentially definable; then $x \in S^{<}$ could be defined as $x \in S^{\neq} \wedge \exists u, v, p(u \in a^* \wedge v \in b^* \wedge p \in a^+ \wedge upv \notin S^{\neq})$ (that is, $|u|+|p| = |v|$), a contradiction. Finally, $S^{=}$ is universally definable from concatenation iff $S^{\neq}$ is existentially definable from concatenation. In fact, the complement set $\{a, b\} \setminus S^{=}$ is $S^{\sim} \cup S^{\neq}$ with $S^{\sim} \triangleq \{a, b\}^* b \{a, b\}^* a \{a, b\}^*$ clearly existentially and universally definable from concatenation. This concludes the proof of the first part of the proposition.

The second part is proved in [11, Theorem 1] for the existential fragment and it is straightforward to adapt that proof to universal definability. □

It is currently unknown whether the extension of the existential or universal fragment of concatenation with *Elg* is decidable, while allowing membership constraints over deterministic context-free language gives an undecidable theory [13].

3 A Theory of Sequences

This section introduces a first-order theory of sequences (Section 3.1) with arithmetic, gives a decision procedure for its universal fragment (Section 3.2), and shows that "natural" larger fragments are undecidable (Section 3.3).

3.1 A Theory of Integer Sequences

We present an arithmetic theory of sequences whose elements are integers. It would be possible to make the theory parametric with respect to the element type. Focusing on integers, however, makes the presentation clearer and more concrete, with minimal loss of generality as one can introduce any theory definable in the integer arithmetic fragment.

3.1.1 Syntax and Semantics

Syntax. Properties of integers are expressed in Presburger arithmetic whose signature is:

$$\Sigma_{\mathbb{Z}} \quad \triangleq \quad \{0, 1, +, -, =, <\}$$

Then, our theory $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ of sequences with integer values has signature

$$\Sigma_{\mathsf{seq}(\mathbb{Z})} \quad \triangleq \quad \Sigma_{\mathsf{cat}} \cup \Sigma_{\mathbb{Z}}$$

Operator precedence is: $\circ; +$ and $-; \doteq, =$ and $<$ followed by logic connectives and quantifiers with the previously defined precedence.

We will generally consider formulas in prenex normal form

$$\mathcal{Q} \bullet \psi$$

where $\mathcal{Q}$ is a quantifier prefix and ψ is quantifier-free written in the grammar:

$$\begin{array}{rcl} seq & ::= & var \mid int \mid seq \circ seq \\ int & ::= & 0 \mid 1 \mid seq \mid int + int \mid int - int \\ fmla & ::= & seq \doteq seq \mid R(seq) \mid int = int \mid int < int \\ & & \mid \neg fmla \mid fmla \vee fmla \mid fmla \wedge fmla \mid fmla \Rightarrow fmla \end{array}$$

with *var* ranging over variable names.

Semantics. An interpretation of a sentence of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ is a structure $\langle \mathbb{Z}^*, \star, \epsilon, \underline{\mathcal{R}}, ev \rangle$ with the following assumptions.[2]

- $\langle \mathbb{Z}^*, \star, \epsilon, \underline{\mathcal{R}}, ev \rangle$ have the same meaning as in the theory of concatenation.
- As far as arithmetic is concerned:
  - The interpretation of a sequence $v_1 v_2 \cdots \in \mathbb{Z}^*$ of integers is the first integer in the sequence v_1, with the convention that the interpretation of the empty sequence is 0.
  - Conversely, the interpretation of an integer value $v \in \mathbb{Z}$ is the singleton sequence v.
  - Addition, subtraction, equality, and less than are interpreted accordingly.

The satisfaction relation is then defined in a standard fashion.

Shorthands. We introduce several shorthands to simplify the writing of complex formulas.

- A symbol for every constant $k \in \mathbb{Z}$, defined as obvious.
- $\alpha \neq \beta$, $\alpha \leq \beta$, $\alpha \geq \beta$, and $\alpha > \beta$ defined respectively as $\neg(\alpha = \beta)$, $\alpha < \beta \vee \alpha = \beta$, $\neg(\alpha < \beta)$, and $\alpha \geq \beta \wedge \alpha \neq \beta$.
- Shorthands such as $\alpha \leq \beta < \gamma$ or $\beta \in [\alpha, \gamma)$ for $\alpha \leq \beta \wedge \beta < \gamma$.
- Bounded length predicates such as $|x| < k$ for a variable x and a constant $k \in \mathbb{Z}$ abbreviating $\widehat{\mathrm{R}}^{<k}(x)$ with $\widehat{\mathrm{R}}^{<k}$ a regular constraint interpreted as $\{\epsilon\} \cup \bigcup_{0<i<k} \mathbb{Z}^i$. The definition of derived expressions such as $k_1 \leq |x| < k_2$ is also as obvious.
- Subsequence functions such as $x(k_1, k_2)$ for a variable x and two constants $k_1, k_2 \in \mathbb{Z}$ with the intended semantics (see Section 2.1). We define these functions in the theory $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ by the following rewriting rules, defined on formulas in prenex normal form with quantifier prefix $\mathcal{Q}$:

$$\frac{\mathcal{Q} \bullet \psi[x(k_1, k_2)]}{\mathcal{Q} \forall u, v, w \bullet \left(\begin{array}{ll} & \kappa_1 \wedge x \doteq uvw \wedge |u| = k_1 - 1 \wedge |v| = k_2 - k_1 + 1 \\ \vee & \kappa_2 \wedge x \doteq uvw \wedge |u| = k_1 - 1 \wedge |w| = -k_2 \\ \vee & \kappa_3 \wedge x \doteq uvw \wedge |v| = -k_1 + k_2 + 1 \wedge |w| = -k_2 \\ \vee & \neg(\kappa_1 \vee \kappa_2 \vee \kappa_3) \wedge u \doteq v \doteq w \doteq \epsilon \end{array} \right) \Rightarrow \psi[v]}$$

  where:

$$\begin{aligned} \kappa_1 &\triangleq 1 \leq k_1 \leq k_2 \leq |x| \\ \kappa_2 &\triangleq k_1 - |x| \leq k_2 < 1 \leq k_1 \\ \kappa_3 &\triangleq 1 - |x| \leq k_1 \leq k_2 < 1 \end{aligned}$$

- $\mathsf{fst}(x)$ and $\mathsf{lst}(x)$ for the first $x(1,1)$ and last element $x(0,0)$ of x, respectively.

[2] The presentation of the semantics of the theory is informal and implicit for brevity.

3.1.2 Examples

A few examples demonstrate the expressiveness of the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ to specify properties of sequences.

1. **Equality**: sequences u and v are equal.

$$\mathsf{u} \doteq \mathsf{v} \tag{1}$$

2. **Bounded equality**: sequences u and v are equal in the *constant* interval $[l, u]$ for $l, u \in \mathbb{Z}$.

$$\mathsf{u}(l, u) \doteq \mathsf{v}(l, u) \tag{2}$$

3. **Boundedness**: no element in sequence u is greater than value v.

$$\forall h, t \bullet \mathsf{u} \doteq ht \Rightarrow t \leq \mathsf{v} \tag{3}$$

4. **Sortedness**: sequence u is sorted (strictly increasing).

$$\forall h, m, t \bullet \mathsf{u} \doteq hmt \wedge |m| = 1 \wedge |t| > 0 \Rightarrow m < t \tag{4}$$

5. **Injectivity**: u has no repeated elements.

$$\forall h, v_1, m, v_2, t \bullet \mathsf{u} \doteq hv_1mv_2t \wedge |v_1| = 1 \wedge |v_2| = 1 \Rightarrow v_1 \neq v_2 \tag{5}$$

6. **Partitioning**: sequence u is partitioned at *constant* position $k > 0$.

$$\forall h_1, t_1, h_2, t_2 \bullet \left(\begin{array}{l} \mathsf{u}(1, k) \doteq h_1t_1 \\ \wedge\ \mathsf{u}(k+1, 0) \doteq h_2t_2 \\ \wedge\ |t_1| > 0 \wedge |t_2| > 0 \end{array} \right) \Rightarrow t_1 < t_2 \tag{6}$$

7. **Membership**: *constant* element $k \in \mathbb{Z}$ occurs in sequence u.

$$\mathsf{u} \in (\mathbb{Z}^* k \mathbb{Z}^*) \tag{7}$$

8. **Non-membership**: no element in sequence u has value v.

$$\forall h, t \bullet \mathsf{u} \doteq ht \wedge |t| > 0 \Rightarrow t \neq \mathsf{v} \tag{8}$$

9. **Periodicity**: in non-empty sequence u, elements on even positions have value 0 and elements on odd positions have value 1 (notice that $\mathsf{lst}(h) = 0$ if h is empty).

$$\forall h, t \bullet \mathsf{u} \doteq ht \wedge |t| > 0 \Rightarrow \left(\begin{array}{c} \mathsf{lst}(h) = 1 \\ \Rightarrow t = 0 \end{array} \right) \wedge \left(\begin{array}{c} \mathsf{lst}(h) = 0 \\ \Rightarrow t = 1 \end{array} \right) \tag{9}$$

10. **Comparison** between indices and values: for every index i, element at position i in the non-empty sequence u has value $i + 3$.

$$\mathsf{u} = 1{+}3 \wedge \forall h, t, v \bullet \mathsf{u} \doteq ht \wedge |h| > 0 \wedge |t| > 0 \wedge \mathsf{lst}(h) = v \Rightarrow t = v{+}1 \tag{10}$$

11. **Disjunction of value constraints**: for every pair of positions $i < j$ in the sequence u, either $\mathsf{u}(i, i) \leq \mathsf{u}(j, j)$ or $\mathsf{u}(i, i) \geq 2\mathsf{u}(j, j)$.

$$\forall h, v_1, m, v_2, t \bullet \mathsf{u} \doteq hv_1mv_2t \wedge |v_1| > 0 \wedge |v_2| > 0 \Rightarrow v_1 \leq v_2 \vee v_1 \geq v_2 + v_2 \tag{11}$$

Comparison with theories of arrays. Properties such as strict sortedness (4), periodicity (9), and comparisons between indices and values (10) are inexpressible in the array logic of Bradley et al. [8]. The latter is inexpressible also in the logic of Ghilardi et al. [16] because Presburger arithmetic is restricted to indices. Properties such as (11) are inexpressible both in the SIL array logic of [21] — because quantification on multiple array indices is disallowed — and in the related LIA logic of [22] — because disjunctions of comparisons of array elements are disallowed. Extensions of each of these logics to accommodate the required features would be undecidable.

Conversely, properties such as *permutation*, bounded equality *for an interval specified by indices*, length constraints *for a variable value*, membership *for a variable value*, and the *subsequence* relation, are inexpressible in the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$. Notice that membership and the subsequence relation are expressible in the dual existential fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$, while the other properties seem to entail undecidability of the corresponding $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ fragment (see Section 3.3). Bounded equality, length constraints, and membership, on the other hand, are expressible in all the logics of [8, 16, 21, 22], and [16] outlines a decidable extension which supports the subsequence relation (see Section 5).

3.2 Deciding Properties of Integer Sequences

This section presents a decision procedure $\mathcal{D}_{\mathsf{seq}(\mathbb{Z})}$ for the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$. The procedure transforms any universal $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ formula into an equisatisfiable universal formula in the theory of concatenation over the free monoid $\{a, b, c, d\}^*$. The basic idea is to encode integers as sequences over the four symbols $\{a, b, c, d\}$: the sequence $acb^{k_1}a$ encodes a nonnegative integer k_1, while the sequence $adb^{-k_2}a$ encodes a negative integer k_2. Suitable rewrite rules encode all quantifier-free Presburger arithmetic in accordance with this convention. The next subsection 3.2.1 outlines the decision procedure $\mathcal{D}_{\mathsf{seq}(\mathbb{Z})}$, while subsection 3.2.2 illustrates its correctness and discusses its complexity.

3.2.1 $\mathcal{D}_{\mathsf{seq}(\mathbb{Z})}$: A Decision Procedure for $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$

Consider a universal formula of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ in prenex normal form:

$$\forall x_1, \ldots, x_v \bullet \psi \tag{12}$$

where ψ is quantifier-free. Modify (12) by application of the following steps.

1. Introduce fresh variables to normalize formulas into the following form:

$$\begin{aligned} fmla \quad ::= \quad & var \doteq var \mid var \doteq var \circ var \mid R(var) \mid var = 0 \mid var = 1 \\ & \mid var = var \mid var = var + var \mid var = var - var \mid var < var \\ & \mid \neg fmla \mid fmla \vee fmla \mid fmla \wedge fmla \mid fmla \Rightarrow fmla \end{aligned}$$

   Clearly, we can achieve this by applying exhaustively rewrite rules that operate on ψ such as:

$$\frac{\psi[x \circ y]}{e \doteq x \circ y \Rightarrow \psi[e]} \qquad \frac{\psi[x + y]}{f = x + y \Rightarrow \psi[f]}$$

   for fresh variables e, f.

2. For each variable $x_i \in FV(\psi) = \{x_1, \ldots, x_v\}$, introduce the fresh variables h_i, t_i, s_i, m_i (for head, tail, sign, modulus) and rewrite ψ as:

$$\bigwedge_{1 \leq i \leq v} \left(\left(\begin{array}{l} x_i \doteq h_i t_i \\ \wedge\ h_i \doteq a s_i m_i a \\ \wedge\ s_i \in \{c, d\} \\ \wedge\ m_i \in b^* \\ \wedge\ t_i \in (acb^*a \cup adb^+a)^* \end{array} \right) \vee \left(\begin{array}{l} x_i \doteq \epsilon \\ \wedge\ h_i \doteq a s_i m_i a \\ \wedge\ s_i \doteq c \\ \wedge\ m_i \doteq \epsilon \\ \wedge\ t_i \doteq \epsilon \end{array} \right) \right) \Rightarrow \psi$$

3. Apply the following rule exhaustively to remove arithmetic equalities:

$$\frac{\psi[x_i = x_j]}{\psi[h_i \doteq h_j]} \qquad \frac{\psi[x_i = 0]}{\psi[h_i \in 0]} \qquad \frac{\psi[x_i = 1]}{\psi[h_i \in 1]}$$

4. Apply the following rule exhaustively to remove differences:

$$\frac{\psi[x_k = x_i - x_j]}{\psi[x_i = x_k + x_j]}$$

5. Apply the following rule exhaustively to remove comparisons:

$$\frac{\psi[x_i < x_j]}{\left(\begin{array}{l} m_i \doteq m_j \\ \vee\ m_i \doteq m_j p \\ \vee\ m_j \doteq m_i p \end{array} \right) \Rightarrow \psi \left[\begin{array}{c} s_i \doteq d \wedge s_j \doteq c \\ \vee \\ s_i \doteq s_j \doteq c \wedge m_j \doteq m_i p \\ \vee \\ s_i \doteq s_j \doteq d \wedge m_i \doteq m_j p \end{array} \right]}$$

for fresh $p \in b^+$.

6. Apply the following rule exhaustively to remove sums:

$$\frac{\psi[x_k = x_i + x_j]}{\left(\begin{array}{l} m_i \doteq m_j \\ \vee\ m_i \doteq m_j p \\ \vee\ m_j \doteq m_i p \end{array} \right) \Rightarrow \psi \left[\begin{array}{c} s_i \doteq s_j \wedge x_k \doteq a s_i m_i m_j a \\ \vee \\ s_i \not\doteq s_j \wedge m_i \doteq m_j \wedge x_k \doteq aca \\ \vee \\ s_i \not\doteq s_j \wedge m_i \doteq m_j p \wedge x_k \doteq a s_i p a \\ \vee \\ s_i \not\doteq s_j \wedge m_j \doteq m_i p \wedge x_k \doteq a s_j p a \end{array} \right]}$$

for fresh $p \in b^+$.

7. Modify the meaning of regularity constraints as follows: let $\underline{\mathrm{R}}_i$ be defined by a regular expression with constants in $\mathbb{Z}$. Substitute every occurrence of a nonnegative constant $k \in \mathbb{Z}$ by $acb^k a$; every occurrence of a negative constant $k \in \mathbb{Z}$ by $adb^{-k}a$; every occurrence of set $\mathbb{Z}$ by $acb^*a \cup adb^+a$.

The resulting formula is again in form (12) where ψ is now a quantifier-free formula in the theory of concatenation over $\{a, b, c, d\}^*$; its validity is decidable by Proposition 1.

3.2.2 Correctness and Complexity

Let us sketch the correctness argument for the decision procedure $\mathcal{D}_{\mathsf{seq}(\mathbb{Z})}$, which shows that the transformed formula is equi-satisfiable with the original one.

The justification for step 1 is straightforward. After applying it a series of substitutions eliminates arithmetic by reducing it to equations over the theory of concatenation with the unary encoding of integers defined above.

Step 2 requires that any variable x_i is a sequence of the form $(acb^*a \cup adb^+a)^*$ and introduces fresh variables to denote significant parts of the sequence: h_i aliases the first element of the sequence which is further split into its sign s_i (c for nonnegative and d for negative) and its absolute value m_i encoded as a unary string in b^*. The second term of the disjunction deals with the case of x_i being ϵ, which has the same encoding as 0.

The following steps replace elements of the signature of Presburger arithmetic by rewriting them as equations over sequences with the given encoding. Step 3 reduces the arithmetic equality of two sequences of integers to equivalence of the sequences encoding their first elements. Step 4 rewrites equations involving differences with equations involving sums.

Step 5 reduces arithmetic comparisons of two sequences of integers to a case discussion over the sequences h_i, h_j encoding their first elements. Let p be a sequence in b^+ encoding the difference between the absolute values corresponding to h_i and h_j; obviously such a p always exists unless the absolute values are equal. Then, h_i encodes an integer strictly less than h_j iff one of the following holds: (1) h_i is a negative value and h_j is a nonnegative one; (2) both h_i and h_j are a nonnegative value and the sequence of b's in h_j is longer than the sequence of b's in h_i; or (3) both h_i and h_j are a negative value and the sequence of b's in h_i is longer than the sequence of b's in h_j.

Step 6 reduces the comparison between the value of a sum of two variables and a third variable to an analysis of the three sequences h_i, h_j, h_k encoding the first elements of the three variables. As in step 6, the unary sequence p encodes the difference between the absolute values corresponding to h_i and h_j. Then, h_k encodes the sum of the values encoded by h_i and h_j iff one of the following holds: (1) h_i and h_j have the same sign and h_k contains a sequence of b's which adds up the sequences of b's of h_i and h_j, still with the same sign; (2) h_i and h_j have opposite sign but same absolute value, so h_k must encode 0; (3) h_i and h_j have opposite sign and the absolute value of h_i is greater than the absolute value of h_j, so h_k has the same sign as h_i and the difference of absolute values as its absolute value; or (4) h_i and h_j have opposite sign and the absolute value of h_j is greater than the absolute value of h_i, so h_k has the same sign as h_j and the difference of absolute values as its absolute value.

Finally, step 7 details how to translate the interpretation of the regular constraints over $\mathbb{Z}$ into the corresponding regularity constraints over $\{a, b, c, d\}$ with the given integer encoding.

It is not difficult to see that all rewriting steps in the decision procedure $\mathcal{D}_{\mathsf{seq}(\mathbb{Z})}$ increase the size of ψ at most quadratically (this accounts for fresh variables as well). Hence, the PSPACE complexity of the universal fragment of the theory of concatenation (Proposition 1) carries over to $\mathcal{D}_{\mathsf{seq}(\mathbb{Z})}$.

Theorem 4. *The universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ is decidable in PSPACE with the decision procedure $\mathcal{D}_{\mathsf{seq}(\mathbb{Z})}$.*

3.3 Undecidable Extensions

Theorem 5. *The following extensions of the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ are undecidable.*

1. *The $\forall^*\exists^*$ and $\exists^*\forall^*$ fragments.*
2. *For any pair of integer constants k_1, k_2, the extension with the two length functions $|x|_{k_1}, |x|_{k_2}$ counting the number of occurrences of k_1 and k_2 in x.*
3. *The extension with an equal length predicate $Elg(x, y) \triangleq |x| = |y|$.*
4. *The extension with a sum function $\sigma(x) \triangleq \sum_{i=1}^{|x|} x(i, i)$.*

Proof. 1. Sentences with one quantifier alternation are undecidable already for the theory of concatenation without arithmetic and over a monoid of finite rank (Proposition 2). Notice that the set of sentences that are expressible both in the $\forall^*\exists^*$ and in the $\exists^*\forall^*$ fragment is decidable [40, Th. 4.4]; however, this set lacks a simple syntactic characterization.

2. Corollary of Proposition 2.

3. We encode the universal theory of $\Pi = \langle \mathbb{N}, 0, 1, +, \pi \rangle$ — where $\pi(x, y) \triangleq x2^y$ — in the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ extended by the Elg predicate; undecidability follows from the undecidability of the existential and universal theories of Π [11, Corollary 5]. All we have to do is showing that $\pi(x, y) = p$ is universally definable in $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ with Elg. To this end, first define l_y as a sequence that begins with value y, ends with value 1, and where every element is the successor of the element that follows.

   $$\forall h, t \bullet \mathsf{fst}(l_y) = y \wedge \mathsf{lst}(l_y) = 1 \wedge l_y \doteq ht \wedge |h| > 0 \wedge |t| > 0 \Rightarrow \mathsf{lst}(h) = t + 1$$

   As a result l_y is in the form $y, y-1, \ldots, 1$ and hence has length y.[3] Then, $\pi(x, y)$ is universally definable as the sequence p with the same length as l_y, whose last element is x, and where every element is obtained by doubling the value of the element that follows:

   $$\forall g, u \bullet Elg(p, l_y) \wedge \mathsf{lst}(p) = x \wedge p \doteq gu \wedge |g| > 0 \wedge |u| > 0 \Rightarrow \mathsf{lst}(g) = u + u$$

   Hence p has the form $2^y x, 2^{y-1} x, \ldots, 2^2 x, 2x, x$ which encodes the desired value $x2^y$ in $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$. (Notice that the two universal definitions of l_y and p can be combined into a single universal definition by conjoining the definition of p to the consequent in the definition of l_y).

4. For any sequence x over $\{0, 1\}$ define $Sp(x) = y$ as $y \in 0^*1^* \wedge \sigma(y) = \sigma(x)$. Then, Proposition 2 implies undecidability because this extension of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ can define universal sentences over the free monoid $\{a, b\}^*$ with the function Sp. □

[3] This technique would allow the definition of the length function $|x|$ and full index arithmetic as well.

```
1 merge_sort (a: ARRAY): ARRAY
2 local l, r: ARRAY
3 do
4    if |a| ≤ 1 then
5       { sorted(a) }
6       Result := a
7    else
8       l , r := a[1:|a|/2] , a[|a|/2+1: |a|]
9       { l ⋆ r = a }
10      l , r := merge_sort (l) , merge_sort (r)
11      { sorted(l) ∧ sorted(r) }
12      from Result := ϵ
13      { invariant sorted(Result) ∧ sorted(l) ∧ sorted(r) ∧
14                  lst(Result) ≤ fst(l) ∧ lst(Result) ≤ fst(r) }
15      until |l| = 0 ∨ |r| = 0
16      loop
17         if l.first > r.first then
18           Result := Result ⋆ r.first ; r := r.rest
19         else
20           Result := Result ⋆ l.first ; l := l.rest
21         end
22      end
23      if |l| > 0 then
24         { |r| = 0 } Result := Result ⋆ l
25      else
26         { |l| = 0 } Result := Result ⋆ r
27      end
28 { ensure sorted(Result) }
```

```
1 reverse (a: LIST): LIST
2 local v: INTEGER ; s: STACK
3 do
4    from s := ϵ
5    { invariant s ∘ a = old a }
6    until a = ϵ
7    loop
8       s.push (a.first)
9       a := a.rest
10   end
11   from Result := ϵ
12   { invariant
13        s ∘ Result^R = old a }
14   until s = ϵ
15   loop
16      v := s.top
17      s.pop ; Result.extend (v)
18   end
19 { ensure Result^R = old a}
```

Table 1: Annotated Mergesort (left) and Array Reversal (right).

The decidability of the following is instead currently unknown: the extension of the universal fragment with a function $x \oplus 1$ defined as the sequence $x(1)+1, x(2)+1, \ldots, x(|x|)+1$. The fragment allows the definition of the set $S^{=}\{0^n1^n \mid n \in \mathbb{N}\}$ as the sequences x such that $x \in 0^*1^* \wedge \forall u, v \bullet x \doteq uv \wedge u \in 0^* \wedge v \in 1^* \Rightarrow u \oplus 1 \doteq v$. This is inexpressible in the universal fragment of the theory of concatenation, but the decidability of the resulting fragment is currently unknown (see Proposition 3).

4 Verifying Sequence-Manipulating Programs

This section outlines a couple of examples that demonstrate using formulas in the theory $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ to reason about sequence-manipulating programs. An implementation of the decision procedure $\mathcal{D}_{\mathsf{seq}(\mathbb{Z})}$ is needed to tackle more extensive examples; it is currently underway. The examples are in Eiffel-like pseudo-code [36]; it is not difficult to detail an axiomatic semantics and a backward substitution calculus, using the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$, for the portions of this language used in the examples.

Reversal. In Table 1 (right), a program reverses a sequence of integers, given as a list a, using a stack s. The query "first" returns the first element in a list, and the command "extend" adds an element to the right of a list; the query "top" and the commands "pop" and "push" for a stack have the usual semantics. In the annotations, s is modeled by a sequence whose first element is the bottom of the stack, whereas the expression **old** a denotes the value of a upon entering the routine.

The superscript $^{\mathsf{R}}$ denotes the reversal of a sequence. We do not know if the extension of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ by a reversal function is decidable. However, the following two simple update axioms are sufficient to handle any program which builds the reverse u^{R} of a sequence u starting from an empty sequence and adding one element at a time:

$$\mathsf{u}^{\mathsf{R}} = \epsilon \;\Leftrightarrow\; \mathsf{u} = \epsilon \qquad\qquad |x| = 1 \;\Rightarrow\; (\mathsf{u}x)^{\mathsf{R}} = x\mathsf{u}^{\mathsf{R}}$$

Consider, for instance, the verification condition that checks if the invariant of the second loop (lines 11–18) is indeed inductive:

$$s \circ \mathbf{Result}^{\mathsf{R}} = \mathbf{old}\ a \wedge s \neq \epsilon \quad\Rightarrow\quad s(1,-1) \circ (\mathbf{Result} \circ s(0,0))^{\mathsf{R}} = \mathbf{old}\ a$$

After rewriting $(\mathbf{Result} \circ s(0,0))^{\mathsf{R}}$ into $s(0,0) \circ \mathbf{Result}^{\mathsf{R}}$ the implication is straightforward to check for validity. The rest of the program is also simple to check with standard backward reasoning techniques.

Mergesort. Consider a straightforward recursive implementation of the Mergesort algorithm; Table 1 (left) shows an annotated version, where $\star$ denotes the concatenation operator in the programming language (whose semantics is captured by the corresponding logic operator $\circ$). The annotations specify that the routine produces a sorted array, where predicate sorted(u) is defined as:

$$\mathrm{sorted}(\mathsf{u}) \quad\triangleq\quad \forall h, m, t \;\bullet\; \mathsf{u} \doteq hmt \wedge |m| > 1 \wedge |t| > 0 \Rightarrow m \leq t$$

It is impossible to express in $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ another component of the full functional specification: the output is a permutation of the input. This condition is inexpressible in most of the expressive decidable extensions of the theory of arrays that are currently known, such as [8, 21] (see also Section 5). Complementary automated verification techniques — using different abstractions such as the multiset [37] — can, however, verify this orthogonal aspect.

We must also abstract away the precise splitting of array a into two halves in line 8. The way in which a is partitioned into l and r is however irrelevant as far as correctness if concerned (it only influences the complexity of the algorithm), hence we can simply over-approximate the instruction on line 8 by a nonderministic splitting in two continuous non-empty parts.

From the annotated program, we can generate verification conditions by standard backward reasoning. Universal sentences of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ can express the verification conditions, hence the verification process can be automated. Let us see an example on the non-trivial part of the process, namely checking that the formula on lines 13–14 is indeed an inductive invariant. Consider the "then" branch on line 18. Backward substitution of the invariant yields:

$$\begin{array}{c} \mathrm{sorted}(\mathbf{Result} * \mathsf{fst}(r)) \;\wedge\; \mathrm{sorted}(l) \;\wedge\; \mathrm{sorted}(r(2,0)) \;\wedge \\ \mathsf{lst}(\mathbf{Result} * \mathsf{fst}(r)) \leq \mathsf{fst}(l) \;\wedge\; \mathsf{lst}(\mathbf{Result} * \mathsf{fst}(r)) \leq \mathsf{fst}(r(2,0)) \end{array} \tag{13}$$

This condition must be discharged by the corresponding loop invariant hypothesis:

$$\begin{array}{c} \mathsf{fst}(l) > \mathsf{fst}(r) \;\wedge\; \mathrm{sorted}(\mathbf{Result}) \;\wedge\; \mathrm{sorted}(l) \;\wedge\; \mathrm{sorted}(r) \;\wedge \\ \mathsf{lst}(\mathbf{Result}) \leq \mathsf{fst}(l) \;\wedge\; \mathsf{lst}(\mathbf{Result}) \leq \mathsf{fst}(r) \;\wedge\; |l| \neq 0 \;\wedge\; |r| \neq 0 \end{array} \tag{14}$$

Checking that (14) entails (13) discharges the corresponding verification condition. Elements of this condition can be encoded in the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ and proven using the decision procedure of Section 3.2; for instance, the fact that $\mathsf{lst}(\mathbf{Result}) \leq \mathsf{fst}(l)$, $|l| \neq 0$, $|r| \neq 0$, and $\mathsf{fst}(l) > \mathsf{fst}(r)$ imply $\mathsf{lst}(\mathbf{Result} * \mathsf{fst}(r)) \leq \mathsf{fst}(l)$ corresponds to the validity of (all free variables are implicitly universally quantified):

$$\left(\begin{array}{l} r \doteq h_r m_r t_r \ \wedge\ |h_r| = 1 \ \wedge\ |r| \neq 0 \\ \wedge\ l \doteq h_l m_l t_l \ \wedge\ |h_l| = 1 \ \wedge\ |l| \neq 0 \\ \wedge\ \mathbf{Result} \circ h_r \doteq hmt \wedge |t| = 1 \\ \wedge\ h_l > h_r \end{array}\right) \Rightarrow t \leq h_l$$

5 Related Work

Pioneering efforts on automated program verification focused on very simple data types — in most cases just scalar variables — as the inherent difficulties were already egregious. As verification techniques progressed and matured, more complex data types were considered, such as lists (usually *à la* Lisp), arrays, maps, and pointers, up to complex dynamic data structures. Arrays in particular received a lot of attention, both for historical reasons (programming languages have been offering them natively for decades), and because they often serve as the basis for implementing more complex data structures. More generally, a renewed interest in developing decision procedures for new theories and in integrating existing ones has blossomed over the last few years. A review of this staggering amount of work is beyond the scope of this paper; for a partial account and further references we refer the reader to e.g., [43, 30] (and [24, 28] for applications). In this section, we review approaches that are most similar to ours and in particular which yield decidable logics that can be compared directly to our theory of sequences (see Section 3.1.2). This is the case with several of the works on the theory of arrays and extensions thereof.

The theory of arrays. McCarthy initiated the research on formal reasoning about arrays [34]. His theory of arrays defines the axiomatization of the basic access operations of *read* and *write* for quantifier-free formulas and without arithmetic or extensionality (i.e., the property that if all elements of two arrays are equal then the arrays themselves are equal). McCarthy's work has usually been the kernel of every theory of arrays: most works on (automated) reasoning about arrays extend McCarthy's theory with more complex (decidable) properties or efficiently automate reasoning within an existing theory.

Thus, a series of work extended the theory of arrays with arithmetic [42, 25] and with sorting predicates on array segments [33]; automated reasoning within these theories is possible only for restricted classes of programs. Extensionality is another very significant extension to the theory of arrays [41], which has now become standard as it is decidable.

The fast technological advances in automated theorem proving over the last years have paved the way for efficient implementations of the theory of arrays (usually with extensionality). These implementations use a variety of techniques such as SMT solving [3, 15, 5, 17, 18], saturation theorem proving [31, 2], and abstraction [9, 26, 27]. Automated invariant inference is an important appli-

cation of these decision procedures, which originated a specialized line of work [4, 35, 29].

Decidable extensions of the theory of arrays. The last few years have seen an acceleration in the development of decidable extensions of the extensional theory of arrays with more expressive predicates and functions.

Bradley et al. [8] develop the *array property fragment*, a decidable subset of the $\exists^*\forall^*$ fragment of the theory of arrays. An *array property* is a formula of the form $\exists^*\forall^* \bullet \iota \Rightarrow \nu$, where the universal quantification is restricted to index variables, ι is a guard on index variables with arithmetic (restricted to existentially quantified variables), and ν is a constraint on array values without arithmetic or nested reads, and where no universally quantified index variable is used to select an element that is written to. The array property fragment is decidable with a decision procedure that eliminates universal quantification on index variables by reducing it to conjunctions on a suitable finite set of index values. Extensions of the array property fragment that relax any of the restrictions on the form of array properties are undecidable. Bradley et al. also show how to adapt their theory of arrays to reason about maps.

Ghilardi et al. [16] develop "semantic" techniques to integrate decision procedures into a decidable extension of the theory of arrays. Their $\mathcal{ADP}$ theory merges the quantifier-free extensional theory of arrays with dimension and Presburger arithmetic over indices into a decidable logic. Two extensions of the $\mathcal{ADP}$ theory are still decidable: one with a unary predicate that determines if an array is *injective* (i.e., it has no repeated elements); and one with a function that returns the *domain* of an array (i.e., the set of indices that correspond to definite values). Ghilardi et al. suggest that these extensions might be the basis for automated reasoning on Separation Logic models. The framework of [16] also supports other decidable extensions, such as the *prefix*, and *sorting* predicates, as well as the *map* combinator also discussed in [12].

De Moura and Bjørner [12] introduce *combinatory array logic*, a decidable extension of the quantifier-free extensional theory of arrays with the *map* and *constant-value* combinators (i.e., array functors). The *constant-value* combinator defines an array with all values equal to a constant; the *map* combinator applies a k-ary function to the elements at position i in k arrays $a_1, \ldots, a_k$. De Moura and Bjørner define a decision procedure for their combinatory array logic, which is implemented in the Z3 SMT solver.

Habermehl et al. introduce powerful logics to reason about arrays with integer values [22, 21, 6]; unlike most related work, the decidability of their logic relies on automata-theoretic techniques for a special class of counter automata. More precisely, [22] defines the *Logic of Integer Arrays* **LIA**, whose formulas are in the $\exists^*\forall^*$ fragment and allow Presburger arithmetic on existentially quantified variables, difference and modulo constraints on index variables, and difference constraints on array values. Forbidding disjunctions of difference constraints on array values is necessary to ensure decidability. The resulting fragment is quite expressive, and in particular it includes practically useful formulas that are inexpressible in other decidable expressive fragments such as [8]. The companion work [21] introduces the *Single Index Logic* **SIL**, consisting of existentially quantified Boolean combinations of formulas of the form $\forall^* \bullet \iota \Rightarrow \nu$, where the universal quantification is restricted to index variables, ι is a positive Boolean

combination of bound and modulo constraints on index variables, and ν is a conjunction of difference constraints on array values. Again, the restrictions on quantifier alternations and Boolean combinations are tight in that relaxing one of them leads to undecidability. The expressiveness of **SIL** is very close to that of **LIA**, and the two logics can be considered two variants of the same basic kernel. The other work [6] shows how to use **SIL** to annotate and reason automatically about array-manipulating programs; the tight correspondence between **SIL** and a class of counter automata allows the automatic generation of loop invariants and hence the automation of the full verification process.

Other approaches. Static analysis and abstract interpretation techniques have also been successfully applied to the analysis of array operations, especially with the goal of inferring invariants automatically (e.g., [19, 20, 23]).

6 Future Work

Future work will investigate the decidability of the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ extended with "weak" predicates or functions that slightly increase its expressiveness (such as that outlined at the end of Section 3.3). We will study to what extent the decision procedure for the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$ can be integrated with other decidable logic fragments (and possibly with the dual existential fragment). We will investigate how to automate the generation of inductive invariants for sequence-manipulating programs by leveraging the decidability of the universal fragment of $\mathcal{T}_{\mathsf{seq}(\mathbb{Z})}$. Finally, we will implement the decision procedure, integrate it within a verification environment, and assess its empirical effectiveness on real programs.

References

[1] Habib Abdulrab and Jean-Pierre Pécuchet. Solving word equations. *Journal of Symbolic Computation*, 8(5):499–521, 1989.

[2] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Transactions on Compututational Logic*, 10(1), 2009.

[3] Alessandro Armando, Silvio Ranise, and Michaël Rusinowitch. Uniform derivation of decision procedures by superposition. In *Proceedings of the 15th International Workshop on Computer Science Logic (CSL'01)*, volume 2142 of *Lecture Notes in Computer Science*, pages 513–527. Springer, 2001.

[4] Dirk Beyer, Thomas A. Henzinger, Rupak Majumdar, and Andrey Rybalchenko. Invariant synthesis for combined theories. In *Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'07)*, volume 4349 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2007.

[5] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. A write-based solver for SAT modulo the

theory of arrays. In *Proceedings of 8th Conference on Formal Methods in Computer Aided Design (FMCAD'08)*, pages 1–8. IEEE, 2008.

[6] Marius Bozga, Peter Habermehl, Radu Iosif, Filip Konecný, and Tomás Vojnar. Automatic verification of integer array programs. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 157–172. Springer, 2009.

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

[8] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In *Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, volume 3855 of *Lecture Notes in Computer Science*, pages 427–442. Springer, 2006.

[9] Robert Brummayer and Armin Biere. Lemmas on demand for the extensional theory of arrays. In *Proceedings of the 6th International Workshop on Satisfiability Modulo Theories (SMT'08)*, 2008.

[10] J. Richard Büchi and Steven Senger. Coding in the existential theory of concatenation. *Archive for Mathematical Logic*, 26(1):101–106, 1987.

[11] J. Richard Büchi and Steven Senger. Definability in the existential theory of concatenation and undecidable extensions of this theory. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 34:337–342, 1988.

[12] Leonardo de Moura and Nikolaj Bjørner. Generalized, efficient array decision procedures. In *Proceedings of 9th Conference on Formal Methods in Computer Aided Design (FMCAD'09)*, pages 45–52, 2009.

[13] Volker Diekert. Makanin's algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words.* Cambridge University Press, 2002.

[14] Valery G. Durnev. Unsolvability of the positive $\forall\exists^3$-theory of a free semigroup. *Sibirskiĭ Matematicheskiĭ Zhurnal*, 36(5):1067–1080, 1995.

[15] Vijay Ganesh and David L. Dill. A decision procedure for bit-vectors and arrays. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 519–531. Springer, 2007.

[16] Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, and Daniele Zucchelli. Decision procedures for extensions of the theory of arrays. *Annals of Mathematics and Artificial Intelligence*, 50(3-4):231–254, 2007.

[17] Silvio Ghilardi, Enrica Nicolini, Silvio Ranise, and Daniele Zucchelli. Towards SMT model checking of array-based systems. In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR'08)*, volume 5195 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2008.

[18] Amit Goel, Sava Krstic, and Alexander Fuchs. Deciding array formula with frugal axiom instantiation. In *Proceedings of the 6th International Workshop on Satisfiability Modulo Theories (SMT'08)*, 2008.

[19] Denis Gopan, Thomas W. Reps, and Shmuel Sagiv. A framework for numeric analysis of array operations. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'05)*, pages 338–350. ACM, 2005.

[20] Sumit Gulwani, Bill McCloskey, and Ashish Tiwari. Lifting abstract interpreters to quantified logical domains. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*, pages 235–246. ACM, 2008.

[21] Peter Habermehl, Radu Iosif, and Tomás Vojnar. A logic of singly indexed arrays. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, volume 5330 of *Lecture Notes in Computer Science*, pages 558–573. Springer, 2008.

[22] Peter Habermehl, Radu Iosif, and Tomás Vojnar. What else is decidable about integer arrays? In *Proceedings of the 11th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 474–489. Springer, 2008.

[23] Nicolas Halbwachs and Mathias Péron. Discovering properties about arrays in simple programs. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI'08)*, pages 339–348. ACM, 2008.

[24] Pieter Hooimeijer and Westley Weimer. A decision procedure for subset constraints over regular languages. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'09)*, pages 188–198. ACM, 2009.

[25] Joxan Jaffar. Presburger arithmetic with array segments. *Information Processing Letters*, 12(2):79–82, 1981.

[26] Ranjit Jhala and Kenneth L. McMillan. Array abstractions from proofs. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 193–206. Springer, 2007.

[27] Deepak Kapur, Rupak Majumdar, and Calogero G. Zarba. Interpolation for data structures. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'05)*, pages 105–116, 2006.

[28] Adam Kiezun, Vijay Ganesh, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. HAMPI: a solver for string constraints. In *Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA'09)*, pages 105–116. ACM, 2009.

[29] Laura Kovács and Andrei Voronkov. Finding loop invariants for programs over arrays using a theorem prover. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering (FASE'09)*, volume 5503 of *Lecture Notes in Computer Science*, pages 470–485. Springer, 2009.

[30] Viktor Kuncak, Ruzica Piskac, Philippe Suter, and Thomas Wies. Building a calculus of data structures. In *Proceedings of the 11th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'10)*, Lecture Notes in Computer Science. Springer, 2010.

[31] Christopher Lynch and Barbara Morawska. Automatic decidability. In *Proceedings of the 17th IEEE Symposium on Logic in Computer Science (LICS'02)*, page 7. IEEE Computer Society, 2002.

[32] G. S. Makanin. The problem of solvability of equations in a free semigroup. *Rossiĭskaya Akademiya Nauk. Matematicheskiĭ Sbornik (Translated in Sbornik Mathematics)*, 103(2):147–236, 1977.

[33] Prabhaker Mateti. A decision procedure for the correctness of a class of programs. *Journal of the ACM*, 28(2):215–232, 1981.

[34] John McCarthy. Towards a mathematical science of computation. In *IFIP Congress*, pages 21–28, 1962.

[35] Kenneth L. McMillan. Quantified invariant generation using an interpolating saturation prover. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2008.

[36] Bertrand Meyer. *Object-oriented software construction*. Prentice Hall, 2nd edition, 1997.

[37] Ruzica Piskac and Viktor Kuncak. Decision procedures for multisets with cardinality constraints. In *Proceedings of the 9th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'08)*, volume 4905 of *Lecture Notes in Computer Science*, pages 218–232. Springer, 2008.

[38] Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *Journal of the ACM*, 51(3):483–496, 2004.

[39] Wojciech Plandowski. An efficient algorithm for solving word equations. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 467–476. ACM, 2006.

[40] Sebastian Seibert. Quantifier hierarchies over word relations. In *Proceedings of the 5th Workshop on Computer Science Logic (CSL'91)*, volume 626 of *Lecture Notes in Computer Science*, pages 329–352. Springer, 1992.

[41] Aaron Stump, Clark W. Barrett, David L. Dill, and Jeremy R. Levitt. A decision procedure for an extensional theory of arrays. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, pages 29–37. IEEE Computer Society, 2001.

[42] Norihisa Suzuki and David Jefferson. Verification decidability of presburger array programs. *Journal of the ACM*, 27(1):191–205, 1980.

[43] Karen Zee, Viktor Kuncak, and Martin C. Rinard. Full functional verification of linked data structures. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI'08)*, pages 349–361. ACM, 2008.