What's Decidable About Arrays?*

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma

Computer Science Department
Stanford University
Stanford, CA 94305-9045
{arbrad,zm,sipma}@theory.stanford.edu

Abstract. Motivated by applications to program verification, we study a decision procedure for satisfiability in an expressive fragment of a theory of arrays, which is parameterized by the theories of the array elements. The decision procedure reduces satisfiability of a formula of the fragment to satisfiability of an equisatisfiable quantifier-free formula in the combined theory of equality with uninterpreted functions (EUF), Presburger arithmetic, and the element theories. This fragment allows a constrained use of universal quantification, so that one quantifier alternation is allowed, with some syntactic restrictions. It allows expressing, for example, that an assertion holds for all elements in a given index range, that two arrays are equal in a given range, or that an array is sorted. We demonstrate its expressiveness through applications to verification of sorting algorithms and parameterized systems. We also prove that satisfiability is undecidable for several natural extensions to the fragment. Finally, we describe our implementation in the πVC verification system.

1 Introduction

Software verification — whether via the classical Floyd-Hoare-style proof method with some automatic invariant generation, or through automatic methods like predicate abstraction — relies on the fundamental technology of decision procedures. Therefore, the properties of software that can be proved automatically are to a large extent limited by the expressiveness of the underlying fragments of theories for which satisfiability is decidable and can be checked efficiently.

Arrays are a basic data structure of imperative programming languages. Theories for reasoning about the manipulation of arrays in programs have been studied intermittently for about as long as computer science has been a recognized field [5]. Nonetheless, the strongest predicate about arrays that appears in a decidable fragment is equality between two unbounded arrays [7]. For software verification, unbounded equality is not enough: for example, assertions such as that two subarrays are equal or that all elements of a subarray satisfy a certain

* This research was supported in part by NSF grants CCR-01-21403, CCR-02-20134, CCR-02-09237, CNS-0411363, and CCF-0430102, by ARO grant DAAD19-01-1-0723, and by NAVY/ONR contract N00014-03-1-0939. The first author was additionally supported by a Sang Samuel Wang Stanford Graduate Fellowship.

property are not uncommon in normal programming tasks. We study a fragment of a theory of arrays that allows expressing such properties and many others, and for which satisfiability is decidable.

Various theories of arrays have been addressed in past work. Research in satisfiability decision procedures has focused on the quantifier-free fragments of array theories, as the full theories are undecidable (see Section 5). In our discussion, we use the sorts array, elem, and index for arrays, elements, and indices, respectively. The syntax $a[i]$ represents an array read, while $a\{i \leftarrow e\}$ represents the array with position i modified to e, for $\mathsf{array}\ a$, $\mathsf{elem}\ e$, and $\mathsf{index}\ i$. McCarthy proposed the main axiom of arrays, $\mathsf{read\text{-}over\text{-}write}$ [5]:

$$\begin{array}{l}(\forall\ \mathsf{array}\ a)(\forall\ \mathsf{elem}\ e)(\forall\ \mathsf{index}\ i, j)\\ \left[\begin{array}{lcl} \quad i = j & \rightarrow & a\{i \leftarrow e\}[j] = e \\ \wedge\ \ i \neq j & \rightarrow & a\{i \leftarrow e\}[j] = a[j] \end{array}\right]\end{array}$$

An *extensional* theory of arrays has been studied formally, most recently in [7] and [1]. The extensional theory relates equations between arrays and equations between their elements:

$$(\forall\ \mathsf{array}\ a, b)[(\forall\ \mathsf{index}\ i)\ a[i] = b[i]\ \rightarrow\ a = b]$$

In [8], a decidable quantifier-free fragment of an array theory that allows a restricted use of a *permutation* predicate is studied. Their motivation, as with our work, is that verification of software requires decision procedures for expressive assertion languages. They use their decision procedure to prove that various sorting algorithms return a permutation of their input. In the conclusion of [8], they suggest that a predicate expressing the sortedness of arrays would be useful.

The main theory of arrays that we study in this paper is motivated by practical requirements in software verification. We use Presburger arithmetic for our theory of indices, so the abstract sort index is concrete for us. Additionally, the theory is *parameterized* by the element theories used to describe the contents of arrays. Typical element theories include the theory of integers, the theory of reals, and the theory of equality.

Our satisfiability decision procedure is for a fragment, which we call the *array property* fragment, that allows a constrained use of universal quantification. We characterize the fragment in Section 2, but for now we note that the decidable fragment is capable of expressing *array equality*, the usual equality in an extensional theory of arrays; *bounded equality*, equality between two subarrays; and various properties, like sortedness, of (sub)arrays.

The satisfiability procedure reduces satisfiability of a formula of the array property fragment to satisfiability of a quantifier-free formula in the combined theory of equality with uninterpreted functions (EUF), Presburger arithmetic, and the element theories. The original formula is equisatisfiable to the reduced formula. For satisfiability, handling existential quantification is immediate. Universally quantified assertions are converted to finite conjunctions by instantiating the quantified index variables over a finite set of index terms. The main insight of the satisfiability decision procedure, then, is that for a given formula in the fragment, there is a finite set of index terms such that instantiating universally quantified index variables from only this set is sufficient for completeness (and, trivially, soundness).

After presenting and analyzing this decision procedure, we study a theory of *maps*, which are like arrays except that indices are uninterpreted. Therefore, the decidable fragment of the theory is less powerful for reasoning about arrays; however, it is more expressive than, for example, the quantifier-free fragment of the extensional theory presented in [7]. In particular, it is expressive enough to reason about hashtables.

The paper is organized as follows. Section 2 defines the theory and the fragment that we study. Section 3 describes the decision procedure for satisfiability of the fragment. In Section 4, we prove that the procedure is sound and complete. We also prove that when satisfiability for quantifier-free formulae of the combined theory of EUF, Presburger arithmetic, and array elements is in NP, then satisfiability for *bounded* fragments is NP-complete. In Section 5, we prove that several natural extensions to the fragment result in fragments for which satisfiability is undecidable; we identify one slightly larger fragment for which decidability remains open. Section 6 presents and analyzes a parametric theory of maps. Section 7 motivates the theories with several applications in software verification. We implemented the procedure in our verifying compiler $\pi\mathsf{VC}$; we describe our experience and results in Section 7.4.

2 An Array Theory and Fragment

We introduce the theory of arrays and the *array property* fragment for which satisfiability is decidable.

Definition 1 (Theories) The theory of arrays uses Presburger arithmetic, $T_{\mathbb{Z}}$, for array indices, and the parameter element theories $T^1_{\mathsf{elem}}, \ldots, T^m_{\mathsf{elem}}$, for $m > 0$, for its elements. The many-sorted array theory for the given element theories is called $T_{\mathsf{A}}^{\{\mathsf{elem}_k\}_k}$. We usually drop the superscript.

Recall that the signature of Presburger arithmetic is

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

Assume each T^k_{elem} has signature Σ^k_{elem}. T_{A} then has signature

$$\Sigma_{\mathsf{A}} = \Sigma_{\mathbb{Z}} \cup \bigcup_k \Sigma^k_{\mathsf{elem}} \cup \{\cdot[\cdot], \cdot\{\cdot \leftarrow \cdot\}\}$$

where the two new functions are *read* and *write*, respectively. The read $a[i]$ returns the value stored at position i of a, while the write $a\{i \leftarrow e\}$ is the array a modified so that position i has value e. For multidimensional arrays, we abbreviate $a[i]\cdots[j]$ with $a[i, \ldots, j]$.

The theory of equality with uninterpreted functions (EUF), T_{EUF}, is used in the decision procedure.

Definition 2 (Terms and Sorts) Index variables and terms have sort $\mathbb{Z}$ and are Presburger arithmetic terms. Element variables and terms have sort elem_k, for some element theory T^k_{elem}. Array variables and terms have functional sorts constructed from the $\mathbb{Z}$ and elem_k sorts:

– *One-dimensional sort*: $\mathbb{Z} \to \mathsf{elem}_k$, for some element theory T^k_{elem}
– *Multidimensional sort*: $\mathbb{Z} \to \cdots \to \mathsf{elem}_k$, for some element theory T^k_{elem}; *e.g.*, a two-dimensional array has sort $\mathbb{Z} \to \mathbb{Z} \to \mathsf{elem}_k$

For element term e, both a and $a\{i \leftarrow e\}$ are array terms; the latter term is a with position i modified to e. For array term a and index term i, $a[i]$ is either an element term if a has sort $\mathbb{Z} \to \mathsf{elem}_k$, or an array term if a has a multidimensional sort; *e.g.*, if a has sort $\mathbb{Z} \to \mathbb{Z} \to \mathsf{elem}_k$, then $a[i, j]$ is an element term of sort elem_k, while $a[i]$ is an array term of sort $\mathbb{Z} \to \mathsf{elem}_k$.

Definition 3 (Literal and Formula) A T_{A}-literal is either a $T_{\mathbb{Z}}$-literal or a T^k_{elem}-literal; literals can contain array subterms. A *formula* ψ in T_{A} is a quantified Boolean combination of T_{A}-literals.

Notationally, we say $\psi[t]$ is the formula that contains subterm t. $t \in \psi$ is true iff ψ contains subterm t.

We study satisfiability for a fragment of T_{A} that is a subset of the $\exists^*\forall^*_{\mathbb{Z}}$-fragment of T_{A}, where the subscript on $\forall$ indicates that the quantifier is only over index variables. We call this fragment the *array property* fragment.

Definition 4 (Array Property) An *array property* is a formula of the form

$$(\forall \bar{i})(\varphi_I(\bar{i}) \to \varphi_V(\bar{i}))$$

where $\bar{i}$ is a vector of index variables, and $\varphi_I(\bar{i})$ and $\varphi_V(\bar{i})$ are the *index guard* and the *value constraint*, respectively. The *height* of the property is the number of quantified index variables in the formula.

The form of an *index guard* $\varphi_I(\bar{i})$ is constrained according to the grammar

$$\begin{array}{rcl}
\mathsf{iguard} & \to & \mathsf{iguard} \wedge \mathsf{iguard} \mid \mathsf{iguard} \vee \mathsf{iguard} \mid \mathsf{atom} \\
\mathsf{atom} & \to & \mathsf{expr} \leq \mathsf{expr} \mid \mathsf{expr} = \mathsf{expr} \\
\mathsf{expr} & \to & \mathit{uvar} \mid \mathsf{pexpr} \\
\mathsf{pexpr} & \to & \mathbb{Z} \mid \mathbb{Z} \cdot \mathit{evar} \mid \mathsf{pexpr} + \mathsf{pexpr}
\end{array}$$

where *uvar* is any universally quantified variable, and *evar* is any existentially quantified integer variable.

The form of a *value constraint* $\varphi_V(\bar{i})$ is also constrained. Any occurrence of a quantified index variable $i \in \bar{i}$ in $\varphi_V(\bar{i})$ must be as a read into an array, $a[i]$, for array term a. Array reads may not be nested; *e.g.*, $a_1[a_2[i]]$ is not allowed.

Definition 5 (Array Property Fragment) The *array property fragment* of T_{A} consists of all existentially-closed Boolean combinations of array property formulae and quantifier-free T_{A}-formulae. The *height* of a formula in the fragment is the maximum height of an array property subformula.

Example 1 (Equality Predicates). Extensionality can be encoded in the array property fragment. We present $=$ and *bounded equality* as defined predicates. In the satisfiability decision procedure, instances of defined predicates are expanded to their definitions in the first step.

Equality $a = b$: Arrays a and b are equal.

$$(\forall i)(a[i] = b[i])$$

Bounded Equality $\mathsf{beq}(\ell, u, a, b)$: Arrays a and b are equal in the interval $[\ell, u]$.

$$(\forall i)(\ell \leq i \leq u \ \rightarrow \ a[i] = b[i])$$

Example 2 (Sorting Predicates). More specialized predicates can also be defined in the array property fragment. Consider the following predicates for specifying properties useful for reasoning about sortedness of integer arrays in the array property fragment of $T_{\mathsf{A}}^{\{\mathbb{Z}\}}$.

Sorted $\mathsf{sorted}(\ell, u, a)$: Integer array a is sorted (nondecreasing) between elements ℓ and u.

$$(\forall i, j)(\ell \leq i \leq j \leq u \ \rightarrow \ a[i] \leq a[j])$$

Partitioned $\mathsf{partitioned}(\ell_1, u_1, \ell_2, u_2, a)$: Integer array a is partitioned such that all elements in $[\ell_1, u_1]$ are less than or equal to every element in $[\ell_2, u_2]$.

$$(\forall i, j)(\ell_1 \leq i \leq u_1 < \ell_2 \leq j \leq u_2 \ \rightarrow \ a[i] \leq a[j])$$

The literal $u_1 < \ell_2$ can be expressed as $u_1 \leq \ell_2 - 1$ so that the syntactic restrictions are met.

Example 3 (Array Property Formula). The following formula is in the array property fragment of $T_{\mathsf{A}}^{\{\mathbb{Z}\}}$:

$$\begin{array}{l}(\exists \ \mathsf{array}\ a)(\exists w, x, y, z, k, \ell, n \in \mathbb{Z})\\ \left[\begin{array}{c} w < x < y < z \ \wedge \ 0 < k < \ell < n \ \wedge \ \ell - k > 1 \\ \wedge \ \mathsf{sorted}(0, n-1, a\{k \leftarrow w\}\{\ell \leftarrow x\}) \ \wedge \ \mathsf{sorted}(0, n-1, a\{k \leftarrow y\}\{\ell \leftarrow z\}) \end{array}\right]\end{array}$$

3 Decision Procedure $\mathsf{SAT_A}$

We now define the decision procedure $\mathsf{SAT_A}$ for satisfiability of formulae from the array property fragment. After removing array writes and skolemizing existentially quantified variables, $\mathsf{SAT_A}$ rewrites universally quantified subterms to finite conjunctions by instantiating the quantified variables over a set of index terms. The next definitions construct the set of index terms that is sufficient for making this procedure complete.

Definition 6 (Read Set) The *read set* for formula ψ is the set

$$\mathcal{R} \stackrel{\text{def}}{=} \{t \ : \ \cdot[t] \in \psi\}$$

for t representing index terms that are not universally quantified index variables.

Definition 7 (Bounds Set) The *bounds set* $\mathcal{B}$ for formula ψ is the set of Presburger arithmetic terms that arise as root pexprs (*i.e.*, pexpr terms whose parent is an expr) during the parsing of all index guards in ψ, according to the grammar of Def. 4.

The read set $\mathcal{R}$ is the set of index terms at which some array is read, while the bounds set $\mathcal{B}$ is the set of index terms that define boundaries on some array for an array property (*e.g.*, the boundaries of an interval in which array elements are sorted).

Definition 8 (Index Set) For a formula ψ, define

$$\mathcal{I}_\psi \overset{\text{def}}{=} \begin{cases} \{0\} & \text{if } \mathcal{R} = \mathcal{B} = \emptyset \\ \mathcal{R} \cup \mathcal{B} & \text{otherwise} \end{cases}$$

The procedure reduces the satisfiability of array property formula ψ to the satisfiability of a quantifier-free $(T_{\mathsf{EUF}} \cup T_{\mathbb{Z}} \cup \bigcup_k T^k_{\mathsf{elem}})$-formula.

Definition 9 ($\mathsf{SAT_A}$)

1. Replace instances of defined predicates with their definitions, and convert to negation normal form.
2. Apply the following rule exhaustively to remove writes:

$$\frac{\psi[a\{i \leftarrow e\}]}{\psi[b] \;\wedge\; b[i] = e \;\wedge\; (\forall j)(j \neq i \;\rightarrow\; a[j] = b[j])} \text{ for fresh } b \qquad (\mathsf{write})$$

   To meet the syntactic requirements on an index guard, we rewrite the third conjunct as

$$(\forall j)(j \leq i - 1 \;\vee\; i + 1 \leq j \;\rightarrow\; a[j] = b[j]) \;.$$

3. Apply the following rule exhaustively:

$$\frac{\psi[(\exists \bar{i})(\varphi_I(\bar{i}) \wedge \neg\varphi_V(\bar{i}))}{\psi[\varphi_I(\bar{j}) \wedge \neg\varphi_V(\bar{j})]} \text{ for fresh } \bar{j} \qquad (\mathsf{exists})$$

4. Apply the following rule exhaustively, where $\mathcal{I}_{\psi_3}$ is determined by the formula constructed in Step 3.

$$\frac{\psi[(\forall \bar{i})(\varphi_I(\bar{i}) \rightarrow \varphi_V(\bar{i}))]}{\psi\left[\displaystyle\bigwedge_{\bar{i} \in \mathcal{I}^n_{\psi_3}} (\varphi_I(\bar{i}) \rightarrow \varphi_V(\bar{i}))\right]} \qquad (\mathsf{forall})$$

5. Associate with each n-dimensional array variable a a fresh n-ary uninterpreted function f_a, and replace each array read $a[i, \ldots, j]$ by $f_a(i, \ldots, j)$. Decide this formula's satisfiability using a procedure for quantifier-free formulae of $T_{\mathsf{EUF}} \cup T_{\mathbb{Z}} \cup \bigcup_k T^k_{\mathsf{elem}}$.

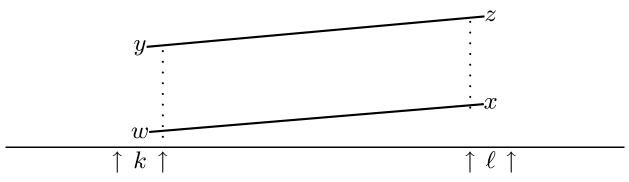

Fig. 1. Unsorted arrays

Step 2 introduces new index terms ($i-1$ and $i+1$, above).

Example 4 (New Indices). Consider again the array property formula

$$\begin{array}{l} w < x < y < z \ \wedge \ 0 < k < \ell < n \ \wedge \ \ell - k > 1 \\ \wedge \ \mathsf{sorted}(0, n-1, a\{k \leftarrow w\}\{\ell \leftarrow x\}) \ \wedge \ \mathsf{sorted}(0, n-1, a\{k \leftarrow y\}\{\ell \leftarrow z\}) \end{array}$$

(which is existentially closed). The first step of $\mathsf{SAT_A}$ replaces the sorted literals with definitions; the second applies write to remove array writes. For readability, we write the index guards resulting from write using disequalities:

$$\begin{array}{l} w < x < y < z \ \wedge \ 0 < k < \ell < n \ \wedge \ \ell - k > 1 \\ \wedge \ (\forall i, j)(0 \leq i \leq j \leq n-1 \ \rightarrow \ c[i] \leq c[j]) \\ \wedge \ (\forall i, j)(0 \leq i \leq j \leq n-1 \ \rightarrow \ e[i] \leq e[j]) \\ \wedge \ (\forall i)(i \neq \ell \ \rightarrow \ b[i] = c[i]) \ \wedge \ c[\ell] = x \\ \wedge \ (\forall i)(i \neq k \ \rightarrow \ a[i] = b[i]) \ \wedge \ b[k] = w \\ \wedge \ (\forall i)(i \neq \ell \ \rightarrow \ d[i] = e[i]) \ \wedge \ e[\ell] = z \\ \wedge \ (\forall i)(i \neq k \ \rightarrow \ a[i] = d[i]) \ \wedge \ d[k] = y \end{array}$$

Then $\mathcal{R} = \{k, \ell\}$, $\mathcal{B} = \{0, n-1, \ell-1, \ell+1, k-1, k+1\}$, and $\mathcal{I}_\psi = \{0, n-1, k-1, k, k+1, \ell-1, \ell, \ell+1\}$. Note that $\mathcal{R}$ and $\mathcal{B}$ do not include i or j, which are universally quantified, while $\mathcal{B}$ contains the terms produced by converting disequalities to disjunctions of inequalities. Applying forall to each array property subformula converts universal quantification to finite conjunction over $\mathcal{I}_\psi$. We have in particular that

$$c[k+1] \leq c[\ell] = x < y = d[k] \leq d[k+1] \ ,$$

yet $c[k+1] = b[k+1] = a[k+1] = d[k+1]$, a contradiction. Thus, the original formula is $T_{\mathsf{A}}^{\{\mathbb{Z}\}}$-unsatisfiable. The index term $k+1$ is essential for this proof.

We visualize this situation in Figure 1. Arrows indicate positions represented by the new indices introduced in Step 2. Pictorially, for both modified versions of a to be sorted requires that the two parallel lines in Figure 1 be one line. To prove that sortedness is impossible requires considering elements in the interval between k and ℓ, not just elements at positions k and ℓ.

4 Correctness

We prove the soundness and completeness of $\mathsf{SAT_A}$. Additionally, we show that if satisfiability of quantifier-free $(T_{\mathsf{EUF}} \cup T_{\mathbb{Z}} \cup \bigcup_k T_{\mathsf{elem}}^k)$-formulae is in NP, then satisfiability for each *bounded* fragment, in which all array properties have maximum height N, is NP-complete.

We refer to the formula constructed in Step n of $\mathsf{SAT_A}$ by ψ_n; *e.g.*, ψ_5 is the final quantifier-free formula constructed in Step 5.

Lemma 1 (Complete). *If ψ_5 is satisfiable, then ψ is satisfiable.*

Proof. Suppose that I is an interpretation such that $I \models \psi_5$; we construct an interpretation J such that $J \models \psi$. To this end, we define under I a *projection* operation, $\mathsf{proj} : \mathbb{Z} \to \mathcal{I}^I_{\psi_3}$: $\mathsf{proj}(z) = t^I$ such that $t \in \mathcal{I}_{\psi_3}$; and either $t^I \le z$ and $(\forall s \in \mathcal{I}_{\psi_3})(s^I \le t^I \vee s^I > z)$, or $t^I > z$ and $(\forall s \in \mathcal{I}_{\psi_3})(s^I \ge t^I)$. That is, $\mathsf{proj}(z)$ is the nearest neighbor to z in t^I, with preference for left neighbors. Extend proj to tuples of integers in the natural way: $\mathsf{proj}(z_1, \dots, z_k) = (\mathsf{proj}(z_1), \dots, \mathsf{proj}(z_k))$.

Equate all non-array variables in J and I; note that proj is now defined the same under I and J. For each k-dimensional array a of ψ, set $a^J[\overline{z}] = f^J_a(\mathsf{proj}(\overline{z}))$. We now prove that $J \models \psi$.

The manipulations in Steps 1, 3, and 5 are trivial. Step 2 implements the definition of array write, so that the resulting formula is equivalent to the original formula. Thus, we focus on Step 4. We prove that if $J \models \psi_4$, then $J \models \psi_3$.

Suppose that rule forall is applied to convert ψ_b to ψ_a and that $J \models \psi_a$. Application of this rule is the main focus of the proof: we prove that $J \models \psi_b$. That is, we assume that

$$J \models \underbrace{\psi' \left[\bigwedge_{\overline{i} \in \mathcal{I}^n_\psi} (\varphi_I(\overline{i}) \to \varphi_V(\overline{i})) \right]}_{\psi_a} \tag{1}$$

and prove that

$$J \models \underbrace{\psi' \left[(\forall \overline{i})(\varphi_I(\overline{i}) \to \varphi_V(\overline{i})) \right]}_{\psi_b} \; . \tag{2}$$

Below, we prove that

$$J \models \left[\bigwedge_{\overline{i} \in \mathcal{I}^n_\psi} (\varphi_I(\overline{i}) \to \varphi_V(\overline{i})) \;\to\; (\forall \overline{i})(\varphi_I(\overline{i}) \to \varphi_V(\overline{i})) \right] \; ,$$

which implies (2) since ψ' is in negation normal form. Our proof takes the form

$$\begin{tikzcd}
 & \varphi_I(\mathsf{proj}(\overline{z})) \arrow[r] & \varphi_V(\mathsf{proj}(\overline{z})) \arrow[d, dashed, leftrightarrow, "(B)"] \\
J \models & \varphi_I(\overline{z}) \arrow[u, dashed, "(A)"] \arrow[r, "?"] & \varphi_V(\overline{z})
\end{tikzcd}$$

where, for arbitrary $\overline{z} \in \mathbb{Z}^n$, we prove the implication labeled "?" by proving (A) and (B). The top implication follows from (1) and the definition of proj.

For (A), consider the atoms of the index guard under J. If $\ell^J \leq m^J$, then the definition of proj implies that $\mathsf{proj}(\ell^J) \leq \mathsf{proj}(m^J)$. At worst, it may be that $\ell^J < m^J$, while $\mathsf{proj}(\ell^J) = \mathsf{proj}(m^J)$. For an equation, $\ell^J = m^J$ iff $\mathsf{proj}(\ell^J) = \mathsf{proj}(m^J)$. Then (A) follows by structural induction over the index guard, noting that the index guard is a positive Boolean combination of atoms.

For (B), recall that arrays in J are constructed using proj. In particular, for any $\overline{z}$, $a^J[\overline{z}] = a^J[\mathsf{proj}(\overline{z})]$, so that (B) follows.

Therefore, $J \models \psi_b$, and $\mathsf{SAT_A}$ is complete.

Lemma 2 (Sound). *If ψ is satisfiable, then ψ_5 is satisfiable.*

Proof. An interpretation I satisfying ψ can be altered to J satisfying ψ_5 by assigning $f_a^J(\overline{i}^I) = a^I[\overline{i}^I]$ for each array variable a and equating all else. Universal quantification is replaced by conjunction over a finite subset of all indices, thus weakening each (positive) literal.

Theorem 1. *If satisfiability of quantifier-free $(T_{\mathsf{EUF}} \cup T_{\mathbb{Z}} \cup \bigcup_k T^k_{\mathsf{elem}})$-formulae is decidable, then $\mathsf{SAT_A}$ is a decision procedure for satisfiability in the array property fragment of $T_{\mathsf{A}}^{\{\mathsf{elem}_k\}_k}$.*

Theorem 2 (NP-Complete). *If satisfiability of quantifier-free $(T_{\mathsf{EUF}} \cup T_{\mathbb{Z}} \cup \bigcup_k T^k_{\mathsf{elem}})$-formulae is in NP, then for the subfragment of the array property fragment of $T_{\mathsf{A}}^{\{\mathsf{elem}_k\}_k}$ in which all array property formulae have height at most N, satisfiability is NP-complete.*

Proof. NP-hardness, even when ψ is a conjunction of literals, follows by NP-hardness of satisfiability of $T_{\mathbb{Z}}$ [6]. Steps 1-3 increase the size of the formula by an amount linear in the size of ψ. The rule forall increases the size of formulae by an amount polynomial in the size of ψ and exponential in the maximum height N. For fixed N, the increase is thus polynomial in ψ. The proof requires only a polynomial number (in the size of ψ) of applications of rules, so that the size of the quantifier-free $(T_{\mathsf{EUF}} \cup T_{\mathbb{Z}} \cup \bigcup_k T^k_{\mathsf{elem}})$-formula is at most polynomially larger than ψ. Inclusion in NP follows from the assumption of the theorem.

5 Undecidable Problems

Theorem 1 states that for certain sets of element theories $\{\mathsf{elem}_k\}_k$, $\mathsf{SAT_A}$ is a satisfiability decision procedure for the array property fragment of $T_{\mathsf{A}}^{\{\mathsf{elem}_k\}_k}$. The theory of reals, $T_{\mathbb{R}}$, in which variables range over $\mathbb{R}$ and with signature $\Sigma_{\mathbb{R}} = \{0, 1, +, -, =, <\}$, and the theory of integers, $T_{\mathbb{Z}}$, are such element theories. We now show that several natural extensions of the array property fragment result in a fragment of $T_{\mathsf{A}}^{\{\mathbb{R}\}}$ or $T_{\mathsf{A}}^{\{\mathbb{Z}\}}$ for which satisfiability is undecidable. We identify one extension for which decidability remains open.

Theorem 3. *Satisfiability of the $\exists^*\forall_{\mathbb{Z}}\exists_{\mathbb{Z}}$-fragment of both $T_{\mathsf{A}}^{\{\mathbb{R}\}}$ and $T_{\mathsf{A}}^{\{\mathbb{Z}\}}$ is undecidable, even with syntactic restrictions like in the array property fragment.*

Proof. In [3], we prove that termination of loops of this form is undecidable:

```
**real** $x_1, \dots, x_n$
$\theta : \bigwedge_{i \in I \subseteq \{1,\dots,n\}} x_i = c_i$
**while** $x_1 \geq 0$ **do**
  **choose** $\tau_i : \mathbf{x} := A_i \mathbf{x}$
**done**
```

c_i are constant integers, $c_i \in \mathbb{Z}$, while each A_i is an $n \times n$ constant array of integers, $A_i \in \mathbb{Z}^{n \times n}$. θ is the initial condition of the loop. Variables $x_1, \dots, x_n$ range over the reals, $\mathbb{R}$. There are $m > 0$ transitions, $\{\tau_1, \dots, \tau_m\}$; on each iteration, one is selected nondeterministically to be taken. $\mathbf{x}$ is an $\mathbb{R}^n$-vector representing the n variables $\{x_1, \dots, x_n\}$; each transition thus updates all variables simultaneously by a linear transformation. We call loops of this form linear loops. Termination for similar loops in which all variables are declared as integers is also undecidable.

We now prove by reduction from termination of linear loops that satisfiability of the $\exists^* \forall_{\mathbb{Z}} \exists_{\mathbb{Z}}$-fragment is undecidable. That is, given linear loop L, we construct formula φ such that φ is unsatisfiable iff L always terminates. In other words, a model of φ encodes a nonterminating computation of L.

For each loop variable x_i, we introduce array variable x_i. Let $\rho_\tau(s,t)$, for index terms s and t, encode transition $\tau : \mathbf{x} := A\mathbf{x}$ as follows:

$$\rho_\tau(s,t) \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} x_i[t] = A_{i,1} \cdot x_1[s] + \cdots + A_{i,n} \cdot x_n[s] \ .$$

Let $g(s)$, for index term s, encode the guard $x_1 \geq 0$:

$$g(s) \stackrel{\text{def}}{=} x_1[s] \geq 0 \ .$$

Let $\theta(s)$, for index term s, encode the initial condition:

$$\theta(s) \stackrel{\text{def}}{=} \bigwedge_{i \in I \subseteq \{1,\dots,n\}} x_i[s] = c_i \ .$$

Then form φ:

$$\varphi : \ (\exists x_1, \dots, x_n, z)(\forall i)(\exists j) \left[\theta(z) \ \wedge \ g(z) \ \wedge \ \bigvee_k \rho_{\tau_k}(i,j) \ \wedge \ g(j) \right] \ .$$

Suppose φ is satisfiable. Then construct a nonterminating computation $s_0 s_1 s_2 \dots$ as follows. Let each variable x_k of state s_0 take on the value $x_k[z]$ of the satisfying model. For s_1, choose the j that corresponds to $i = z$ and assign x_k according to $x_k[j]$. Continue forming the computation sequentially. Each state is guaranteed to satisfy the guard, so the computation is nonterminating.

Suppose $s_0 s_1 s_2 \ldots$ is a nonterminating computation of L. Then construct the following model for φ. Let $z = 0$; for each index $i \geq 0$, set $x_k[-i] = x_k[i] = x_k$ of state s_i.

Therefore, φ is unsatisfiable iff L always terminates, and thus satisfiability of the $\exists^* \forall_{\mathbb{Z}} \exists_{\mathbb{Z}}$-fragment of T_{A} is undecidable. Note that φ meets the syntactic restrictions of the array property fragment, except for the extra quantifier alternation.

Theorem 4. *Extending the array property fragment with any of*

- *nested reads (e.g., $a_1[a_2[i]]$, where i is universally quantified);*
- *array reads by a universally quantified variable in the index guard;*
- *general Presburger arithmetic expressions over universally quantified index variables (even just addition of 1, e.g., $i + 1$) in the index guard or in the value constraint*

results in a fragment of $T_{\mathsf{A}}^{\{\mathbb{Z}\}}$ for which satisfiability is undecidable.

Proof. In $T_{\mathsf{A}}^{\{\mathbb{Z}\}}$, the presence of nested reads allows skolemizing j in φ of the proof of Theorem 3:

$$(\exists x_1, \ldots, x_n, z, a_j)(\forall i) \left[\theta(z) \ \wedge \ g(z) \ \wedge \ \bigvee_k \rho_{\tau_k}(i, a_j[i]) \ \wedge \ g(a_j[i]) \right] .$$

Allowing array reads in the index guard enables flattening of nested reads through introduction of another universally quantified variable:

$$\psi[\varphi_I \ \rightarrow \ \varphi_V[a[a[i]]]] \quad \Rightarrow \quad \psi[\varphi_I \wedge j = a[i] \ \rightarrow \ \varphi_V[a[j]]] .$$

Allowing addition of 1 in the value constraint allows an encoding of termination similar to that in the proof of Theorem 3:

$$(\exists x_1, \ldots, x_n, z)(\forall i \geq z) \left[\theta(z) \ \wedge \ g(z) \ \wedge \ \bigvee_k \rho_{\tau_k}(i, i+1) \ \wedge \ g(i+1) \right] .$$

Finally, addition of 1 in the index guard can encode addition of 1 in the value constraint through introduction of another universally quantified variable:

$$\psi[\varphi_I \ \rightarrow \ \varphi_V[a[i+1]]] \quad \Rightarrow \quad \psi[\varphi_I \wedge j = i + 1 \ \rightarrow \ \varphi_V[a[j]]] .$$

Theorem 3 implies that a negated array property cannot be embedded in the consequent of another array property. Theorem 4 states that loosening most syntactic restrictions results in a fragment for which satisfiability is undecidable. One extension remains for which decidability of satisfiability is an open problem: the fragment in which index guards can contain strict inequalities, $<$ (equivalently, in which index guards can contain negations). In this fragment, one could express that an array has unique elements:

$$(\forall i, j)(i < j \ \rightarrow \ a[i] \neq a[j]) .$$

6 Maps

We consider an array theory in which indices are uninterpreted. For clarity, we call indices *keys* in this theory, and call the arrays *maps*.

Definition 10 (Map Theory) The parameterized map theory $T_{\mathsf{M}}^{\{\mathsf{elem}_\ell\}_\ell}$ has signature

$$\Sigma_{\mathsf{M}} = \Sigma_{\mathsf{EUF}} \cup \bigcup_{\ell} \Sigma_{\mathsf{elem}}^{\ell} \cup \{\cdot[\cdot], \cdot\{\cdot \leftarrow \cdot\}\} \ .$$

Key variables and terms are uninterpreted, with sort EUF. Element variables and terms have some sort elem_ℓ. Map variables and terms have functional sorts constructed from the EUF and elem_ℓ sorts; *e.g.*, $\mathsf{EUF} \to \mathsf{elem}_\ell$.

Definition 11 (Map Property Fragment) A *map property* is a formula of the form $(\forall \overline{k})(\varphi_K(\overline{k}) \to \varphi_V(\overline{k}))$, where $\overline{k}$ is a vector of key variables, and $\varphi_K(\overline{k})$ and $\varphi_V(\overline{k})$ are the *key guard* and the *value constraint*, respectively. The *height* of the property is the number of quantified variables in the formula.

The form of a *key guard* $\varphi_K(\overline{k})$ is constrained according to the grammar

$$\begin{array}{rcl} \mathsf{kguard} & \to & \mathsf{kguard} \wedge \mathsf{kguard} \mid \mathsf{kguard} \vee \mathsf{kguard} \mid \mathsf{atom} \\ \mathsf{atom} & \to & \mathsf{var} = \mathsf{var} \mid \mathit{evar} \neq \mathsf{var} \mid \mathsf{var} \neq \mathit{evar} \\ \mathsf{var} & \to & \mathit{evar} \mid \mathit{uvar} \end{array}$$

where *uvar* is any universally quantified key variable, and *evar* is any existentially quantified variable.

The form of a *value constraint* $\varphi_V(\overline{k})$ is also constrained. Any occurrence of a quantified key variable $k \in \overline{k}$ in $\varphi_V(\overline{k})$ must be as a read into a map, $h[k]$, for map term h. Map reads may not be nested; *e.g.*, $h_1[h_2[k]]$ is not allowed.

The *map property fragment* of T_{M} consists of all existentially-closed Boolean combinations of map property formulae and quantifier-free T_{M}-formulae.

Definition 12 (Key Set) For a formula ψ, define $\mathcal{R} = \{t \ : \ \cdot[t] \in \psi\}$; $\mathcal{B}$ as the set of variables that arise as *evar*s in the parsing of all key guards according to the grammar of Def. 11; and $\mathcal{K} = \mathcal{R} \cup \mathcal{B} \cup \{\kappa\}$, where κ is a fresh variable.

Definition 13 (SAT_M)

1. Step 1 of SAT_A.
2. Apply the following rule exhaustively to remove writes:

$$\frac{\psi[h\{k \leftarrow e\}]}{\psi[h'] \ \wedge \ h'[k] = e \ \wedge \ (\forall j)(j \neq k \ \to \ h[j] = h'[j])} \text{ for fresh } h' \qquad (\mathsf{write})$$

3. Step 3 of SAT_A.

4. Apply the following rule exhaustively, where $\mathcal{K}_{\psi_3}$ is determined by the formula constructed in Step 3.

$$\frac{\psi[(\forall \overline{k})(\varphi_K(\overline{k}) \rightarrow \varphi_V(\overline{k}))]}{\psi\left[\bigwedge_{\overline{k}\in\mathcal{K}^n_{\psi_3}} (\varphi_K(\overline{k}) \rightarrow \varphi_V(\overline{k}))\right]} \quad (\mathsf{forall})$$

5. Construct

$$\psi_4 \ \wedge \bigwedge_{k\in\mathcal{K}\setminus\{\kappa\}} k \neq \kappa$$

6. Step 5 of $\mathsf{SAT_A}$, except that the resulting formula is decided using a procedure for $T_{\mathsf{EUF}} \cup \bigcup_\ell T^\ell_{\mathsf{elem}}$.

Theorem 5. *If satisfiability of quantifier-free* $(T_{\mathsf{EUF}} \cup \bigcup_\ell T^\ell_{\mathsf{elem}})$*-formulae is decidable, then* $\mathsf{SAT_M}$ *is a decision procedure for satisfiability in the map property fragment of* $T_{\mathsf{M}}^{\{\mathsf{elem}_\ell\}_\ell}$.

The main idea of the proof, as in the proof of Theorem 1, is to define a projection operation, $\mathsf{proj} : \mathsf{EUF} \rightarrow \mathcal{K}^I_{\psi_3}$, for interpretation I. For object o of I, if $o = t^I$ for some $t \in \mathcal{K}_{\psi_3}$, then $\mathsf{proj}(o) = o \ (= t^I)$; otherwise, $\mathsf{proj}(o) = \kappa^I$. If proj is used to define J, as in the proof of Theorem 1, then proj preserves equations and disequalities in key guards and values of map reads in value constraints.

The relevant undecidability results from Section 5 carry over to maps, with the appropriate modifications.

7 Applications, Implementation, and Results

7.1 Verification of Sorting Algorithms

Figure 2 presents an annotated version of InsertionSort in an imperative language, where the annotations specify that InsertionSort returns a sorted array. @pre, @post, and @ label preconditions, postconditions, and (loop) assertions, respectively. For variable x, x_0 refers to its value upon entering a function; $|arr|$ maps array arr to its length; rv is the value returned by a function. Each verification condition is expressible in the array property fragment of $T_{\mathsf{A}}^{\{\mathbb{Z}\}}$ and is unsatisfiable, proving that InsertionSort returns a sorted array.

7.2 Verification of Parameterized Programs

The parallel composition of an arbitrary number of copies of a process is often represented as a *parameterized program*. Variables for which one copy appears in each process are modeled as arrays. Thus, it is natural to specify and prove properties of parameterized programs with a language of arrays.

```
@pre ⊤
@post sorted(0, |rv| − 1, rv)
int[] InsertionSort(int[] arr) {
  int i, j, t;
  for (i := 1; i < |arr|; i := i + 1)
      @(1 ≤ i ∧ sorted(0, i − 1, arr) ∧ |arr| = |arr_0|)
  {
    t := arr[i];
    for (j := i − 1; j ≥ 0 ∧ arr[j] > t; j := j − 1)
        @[    1 ≤ i < |arr| ∧ −1 ≤ j ≤ i − 1
          ∧ sorted(0, i − 1, arr) ∧ |arr| = |arr_0|
          ∧ (j < i − 1 → (arr[i − 1] ≤ arr[i] ∧ (∀k ∈ [j + 1, i]) arr[k] > t))]
      arr[j + 1] := arr[j];
    arr[j + 1] := t;
  }
  return arr;
}
```

Fig. 2. InsertionSort

```
                    int[] y := int[0..M − 1];
                    θ: y[0] = 1 ∧ (∀j ∈ [1, M − 1]) y[j] = 0

     ||         [ request(y, i);
 i∈[0,M−1]        while (true) @((∀j ∈ [0, M − 1]) y[j] = 0 ∧ i = i_0 ∧ |y| = |y_0|) {
                    critical;
                    release(y, i ⊕_M 1);
                    noncritical;
                    request(y, i);
                  }                                                                 ]
```

Fig. 3. Sem-N

Figure 3 presents a simple semaphore-based algorithm for mutual exclusion among M processes [4]. The semantics of **request** and **release** are the usual ones:

$$\begin{array}{rl}\textbf{request}(y,i): & y[i] > 0 \ \land \ y' = y\{i \leftarrow y[i]-1\} \\ \textbf{release}(y,i): & y' = y\{i \leftarrow y[i]+1\}\end{array}$$

Mutual exclusion at the **critical** section is implied by the invariant $(\forall j \in [0, M-1])\ y[j] = 0$, which appears as part of the loop invariant. The mutual exclusion property is verified using the array decision procedure.

7.3 A Decision Procedure for Hashtables

We show how to encode an assertion language for hashtables, with parameter theories $T^1_{\mathsf{elem}}, \ldots, T^m_{\mathsf{elem}}$ for values, into $T_{\mathsf{M}}^{\{\mathsf{elem}_\ell\}_\ell}$. Hashtables have the following operations: $\mathsf{put}(h, k, v)$ returns the hashtable that is equal to h except that key

k maps to value v; $\mathsf{remove}(h, k)$ returns the hashtable that is equal to h except that key k does not map to a value; and $\mathsf{get}(h, k)$ returns the value mapped by k, which is undetermined if h does not map k to any value. $\mathsf{init}(h)$ is true iff h does not map any key. For reasoning about keys, $k \in \mathsf{keys}(h)$ is true iff h maps k; key sets $\mathsf{keys}(h)$ can be unioned, intersected, and complemented. For the encoding onto the map property fragment of T_{M}, universal quantification is restricted to quantification over key variables; such variables may only be used in membership checking, $k \in K$, and gets, $\mathsf{get}(h, k)$. Finally, an init in the scope of a universal quantifier must appear positively. The encoding then works as follows:

1. Construct $\psi \ \wedge \ \top \neq \bot$, for fresh constants $\top$ and $\bot$.
2. Rewrite

$$\begin{array}{rcl} \psi[\mathsf{put}(h, k, v)] & \Rightarrow & \psi[h'] \ \wedge \ h' = h\{k \leftarrow v\} \ \wedge \ \mathsf{keys}_{h'} = \mathsf{keys}_h\{k \leftarrow \top\} \\ \psi[\mathsf{remove}(h, k)] & \Rightarrow & \psi[h'] \ \wedge \ \mathsf{keys}_{h'} = \mathsf{keys}_h\{k \leftarrow \bot\} \end{array}$$

   for fresh variable h'.
3. Rewrite

$$\begin{array}{rcl} \psi[\mathsf{get}(h, k)] & \Rightarrow & \psi[h[k]] \\ \psi[\mathsf{init}(h)] & \Rightarrow & \psi[(\forall k)(h[k] = \bot)] \\ \psi[k \in \mathsf{keys}(h)] & \Rightarrow & \psi[\mathsf{keys}_h[k] \neq \bot] \\ \psi[k \in K_1 \cup K_2] & \Rightarrow & \psi[k \in K_1 \ \vee \ k \in K_2] \\ \psi[k \in K_1 \cap K_2] & \Rightarrow & \psi[k \in K_1 \ \wedge \ k \in K_2] \\ \psi[k \in \overline{K}] & \Rightarrow & \psi[\neg(k \in K)] \end{array}$$

   where K, K_1, and K_2 are constructed from union, disjunction, and complementation of membership atoms.

Note that we rely on the defined predicate of equality between maps, $h_1 = h_2$, which is defined by $(\forall k)(h_1[k] = h_2[k])$. Subset checking between key sets, $K_1 \subset K_2$, and other useful operations can also be defined in this language.

An example specification might assert that $(\forall k \in \mathsf{keys}(h))(\mathsf{get}(h, k) \geq 0)$. Suppose that a function modifies h; then a verification condition could be

$$(\forall h, s, v, h') \left[\begin{array}{l} (\forall k \in \mathsf{keys}(h)) \ \mathsf{get}(h, k) \geq 0 \ \wedge \ v \geq 0 \ \wedge \ h' = \mathsf{put}(h, s, v) \\ \quad \rightarrow \ (\forall k \in \mathsf{keys}(h')) \ \mathsf{get}(h', k) \geq 0 \end{array} \right] .$$

The key sets provide a mechanism for reasoning about modifying hashtables.

7.4 Implementation and Results

We implemented $\mathsf{SAT_A}$ in our verifying compiler, $\pi\mathsf{VC}$, which verifies programs written in the pi (for *Prove It*) programming language. The syntax of the language is similar to that of Figure 2. We used CVC Lite [2] as the underlying decision procedure. We found that there is usually no need to instantiate quantifiers with all terms in $\mathcal{I}$; instead, the implementation makes several attempts to prove a formula unsatisfiable. It first tries using the set $\mathcal{R}$, then $\mathcal{R} \cup \mathcal{B}$, and finally $\mathcal{I}$. Moreover, common sense rules restrict the instantiation in the early attempts. If any attempt results in an unsatisfiable formula, then the original formula is unsatisfiable; and if the formula of the final attempt is satisfiable, then the original formula is satisfiable.

Frame conditions are ubiquitous in verification conditions. Thus, the implementation performs a simple form of resolution to simplify the original formula. After rewriting based on equations in the antecedent, conjuncts in the consequent that are syntactically equal to conjuncts in the antecedent are replaced with true. In practice, the resulting index sets are smaller. The combination of the phased instantiation and simplification makes the decision procedure quite responsive in practice.

We implemented annotated versions of MergeSort, BubbleSort, InsertionSort, QuickSort, Sem-N, and BinarySearch for integer arrays in our programming language pi. Verifying that the sorting algorithms return sorted arrays required less than 20 seconds each (1 second for each of BubbleSort and InsertionSort). Verifying mutual exclusion in Sem-N required a second. Verifying the membership property of BinarySearch required a second. All tests were performed on a 3 GHz x86; memory was not an issue.

8 Future Work

Future work will focus on the decidability of the extension identified in Section 5; on the complexity of deciding satisfiability for the full array property fragment for particular element theories; and, most importantly, on generating inductive invariants in the array property fragment automatically.

Acknowledgments. We thank the reviewers, Tom Henzinger, and members of the STeP group for their insightful comments and suggestions on this work.

References

1. Armando, A., Ranise, S., and Rusinowitch, M. Uniform derivation of decision procedures by superposition. In *International Workshop on Computer Science Logic (CSL)* (2001), Springer-Verlag.
2. Barrett, C., and Berezin, S. CVC Lite: A new implementation of the cooperating validity checker. In *Computer Aided Verification (CAV)* (2004), Springer-Verlag.
3. Bradley, A. R., Manna, Z., and Sipma, H. B. Polyranking for polynomial loops. In submission; available at `http://theory.stanford.edu/~arbrad`.
4. Manna, Z., and Pnueli, A. *Temporal Verification of Reactive Systems: Safety.* Springer, 1995.
5. McCarthy, J. Towards a mathematical science of computation. In *IFIP Congress 62* (1962).
6. Schrijver, A. *Theory of Linear and Integer Programming.* Wiley, 1986.
7. Stump, A., Barrett, C. W., Dill, D. L., and Levitt, J. R. A decision procedure for an extensional theory of arrays. In *Logic in Computer Science (LICS)* (2001).
8. Suzuki, N., and Jefferson, D. Verification decidability of Presburger array programs. *J. ACM 27*, 1 (1980).