Decision Procedures for Flat Array Properties[*]

Francesco Alberti[1], Silvio Ghilardi[2], Natasha Sharygina[1]

[1] University of Lugano, Lugano, Switzerland
[2] Università degli Studi di Milano, Milan, Italy

Abstract. We present new decidability results for quantified fragments of theories of arrays. Our decision procedures are fully declarative, parametric in the theories of indexes and elements and orthogonal with respect to known results. We also discuss applications to the analysis of programs handling arrays.

1 Introduction

Decision procedures constitute, nowadays, one of the fundamental components of tools and algorithms developed for the formal analysis of systems. Results about the decidability of fragments of (first-order) theories representing the semantics of real system operations deeply influenced, in the last decade, many research areas, from verification to synthesis. In particular, the demand for procedures dealing with quantified fragments of such theories fast increased. Quantified formulas arise from several static analysis and verification tasks, like modeling properties of the heap, asserting frame axioms, checking user-defined assertions in the code and reasoning about parameterized systems.

In this paper we are interested in studying the decidability of quantified fragments of theories of arrays. Quantification is required over the indexes of the arrays in order to express significant properties like "the array has been initialized to 0" or "there exist two different positions of the array containing an element c", for example. From a logical point of view, array variables are interpreted as functions. However, adding free function symbols to a theory T (with the goal of modeling array variables) may yield to undecidable extensions of widely used theories like Presburger arithmetic [17]. It is, therefore, mandatory to identify fragments of the quantified theory of arrays which are on one side still decidable and on the other side sufficiently expressive. In this paper, we show that by combining restrictions on quantifier prefixes with 'flatness' limitations on dereferencing (only positions named by variables are allowed in dereferencing), one can restore decidability. We call the fragments so obtained *Flat Array Properties*; such fragments are orthogonal to the fragments already proven decidable in the literature [8, 15, 16] (we shall defer the technical comparison with these contributions to Section 5). Here we explain the *modularity* character of our

[*]The work of the first author was supported by Swiss National Science Foundation under grant no. P1TIP2_152261.

results and their *applications* to concrete decision problems for array programs annotated with assertions or postconditions.

We examine Flat Array Properties in two different settings. In one case, we consider Flat Array Properties over the theory of arrays generated by adding free function symbols to a given theory T modeling both indexes and elements of the arrays. In the other one, we take into account Flat Array Properties over a theory of arrays built by connecting two theories T_I and T_E describing the structure of indexes and elements. Our decidability results are fully declarative and parametric in the theories T, T_I, T_E. For both settings, we provide sufficient conditions on T and T_I, T_E for achieving the decidability of Flat Array Properties. Such hypotheses are widely met by theories of interest in practice, like Presburger arithmetic. We also provide suitable decision procedures for Flat Array Properties of both settings. Such procedures reduce the decidability of Flat Array Properties to the decidability of T-formulæ in one case and T_I- and T_E-formulæ in the other case.

We further show, as an application of our decidability results, that the safety of an interesting class of programs handling arrays or strings of unknown length is decidable. We call this class of programs $\mathsf{simple}^0_{\mathcal{A}}$*-programs*: this class covers non-recursive programs implementing for instance searching, copying, comparing, initializing, replacing and testing functions. The method we use for showing these safety results is similar to a classical method adopted in the model-checking literature for programs manipulating integer variables (see for instance [7,9,12]): we first assume flatness conditions on the control flow graph of the program and then we assume that transitions labeling cycles are "acceleratable". However, since we are dealing with array manipulating programs, acceleration requires specific results that we borrow from [3]. The key point is that the shape of most accelerated transitions from [3] matches the definition of our Flat Array Properties (in fact, Flat Array Properties were designed precisely in order to encompass such accelerated transitions for arrays).

From the practical point of view, we tested the effectiveness of state of the art SMT-solvers in checking the satisfiability of some Flat Array Properties arising from the verification of $\mathsf{simple}^0_{\mathcal{A}}$-programs. Results show that such tools fail or timeout on some Flat Array Properties. The implementation of our decision procedures, once instantiated with the theories of interests for practical applications, will likely lead, therefore, to further improvements in the areas of practical solutions for the rigorous analysis of software and hardware systems.

Plan of the paper The paper starts by recalling in Section 2 required background notions. Section 3 is dedicated to the definition of Flat Array Properties. Section 3.1 introduces a decision procedure for Flat Array Properties in the case of a mono-sorted theory $\mathtt{ARR}^1(T)$ generated by adding free function symbols to a theory T. Section 3.2 discusses a decision procedure for Flat Array Properties in the case of the multi-sorted array theory $\mathtt{ARR}^2(T_I, T_E)$ built over two theories T_I and T_E for the indexes and elements (we supply also full lower and upper complexity bounds for the case in which T_I and T_E are both Presburger arithmetic). In Section 4 we recall and adapt required notions from [3], define the class of

flat^0-programs and establish the requirements for achieving the decidability of reachability analysis on some flat^0-programs. Such requirements are instantiated in Section 4.1 in the case of $\mathsf{simple}^0_{\mathcal{A}}$-programs, array programs with flat control-flow graph admitting definable accelerations for every loop. In Section 4.2 we position the fragment of Flat Array Properties with respect to the actual practical capabilities of state-of-the-art SMT-solvers. Section 5 compares our results with the state of the art, in particular with the approaches of [8, 15].

2 Background

We use lower-case latin letters $x, i, c, d, e, \ldots$ for variables; for tuples of variables we use bold face letters like $\mathbf{x}, \mathbf{i}, \mathbf{c}, \mathbf{d}, \mathbf{e} \ldots$. The n-th component of a tuple $\mathbf{c}$ is indicated with c_n and $| - |$ may indicate tuples length (so that we have $\mathbf{c} = c_1, \ldots, c_{|\mathbf{c}|}$). Occasionally, we may use free variables and free constants interchangeably. For terms, we use letters $t, u, \ldots$, with the same conventions as above; $\mathbf{t}, \mathbf{u}$ are used for tuples of terms (however, tuples of variables are assumed to be distinct, whereas the same is not assumed for tuples of terms - this is useful for substitutions notation, see below). When we use $\mathbf{u} = \mathbf{v}$, we assume that two tuples have equal length, say n (i.e. $n := |\mathbf{u}| = |\mathbf{v}|$) and that $\mathbf{u} = \mathbf{v}$ abbreviates the formula $\bigwedge_{i=1}^{n} u_i = v_i$.

With $E(\mathbf{x})$ we denote that the syntactic expression (term, formula, tuple of terms or of formulæ) E contains at most the free variables *taken from* the tuple $\mathbf{x}$. We use lower-case Greek letters $\phi, \varphi, \psi, \ldots$ for **quantifier-free** formulæ and $\alpha, \beta, \ldots$ for arbitrary formulæ. The notation $\phi(\mathbf{t})$ identifies a quantifier-free formula ϕ obtained from $\phi(\mathbf{x})$ by substituting the tuple of variables $\mathbf{x}$ with the tuple of terms $\mathbf{t}$.

A *prenex formula* is a formula of the form $Q_1 x_1 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$, where $Q_i \in \{\exists, \forall\}$ and $x_1, \ldots, x_n$ are pairwise different variables. $Q_1 x_1 \cdots Q_n x_n$ is the *prefix* of the formula. Let R be a regular expression over the alphabet $\{\exists, \forall\}$. The R-class of formulæ comprises all and only those prenex formulæ whose prefix generates a string $Q_1 \cdots Q_n$ matched by R.

According to the SMT-LIB standard [22], a theory T is a pair $(\Sigma, \mathcal{C})$, where Σ is a signature and $\mathcal{C}$ is a class of Σ-structures; the structures in $\mathcal{C}$ are called the models of T. Given a Σ-structure $\mathcal{M}$, we denote by $S^{\mathcal{M}}, f^{\mathcal{M}}, P^{\mathcal{M}}, \ldots$ the interpretation in $\mathcal{M}$ of the sort S, the function symbol f, the predicate symbol P, etc. A Σ-formula α is T-satisfiable if there exists a Σ-structure $\mathcal{M}$ in $\mathcal{C}$ such that α is true in $\mathcal{M}$ under a suitable assignment to the free variables of α (in symbols, $\mathcal{M} \models \alpha$); it is T-valid (in symbols, $T \models \alpha$) if its negation is T-unsatisfiable. Two formulæ α_1 and α_2 are T-equivalent if $\alpha_1 \leftrightarrow \alpha_2$ is T-valid; α_1 T-entails α_2 (in symbols, $\alpha_1 \models_T \alpha_2$) iff $\alpha_1 \to \alpha_2$ is T-valid. The satisfiability modulo the theory T ($SMT(T)$) problem amounts to establishing the T-satisfiability of quantifier-free Σ-formulæ. **All theories T we consider in this paper have decidable $SMT(T)$-problem** (we recall that this property is preserved when adding free function symbols, see [13, 26]).

A theory $T = (\Sigma, \mathcal{C})$ admits *quantifier elimination* iff for any arbitrary Σ-formula $\alpha(\mathbf{x})$ it is always possible to compute a quantifier-free formula $\varphi(\mathbf{x})$ such that $T \models \forall \mathbf{x}.(\alpha(\mathbf{x}) \leftrightarrow \varphi(\mathbf{x}))$. Thus, in view of the above assumption on decidability of $SMT(T)$-problem, a theory having quantifier elimination is decidable (i.e. T-satisfiability of *every* formula is decidable). Our favorite example of a theory with quantifier elimination is *Presburger Arithmetic*, hereafter denoted with $\mathbb{P}$; this is the theory in the signature $\{0, 1, +, -, =, <\}$ augmented with infinitely many unary predicates D_k (for each integer k greater than 1). Semantically, the intended class of models for $\mathbb{P}$ contains just the structure whose support is the set of the natural numbers, where $\{0, 1, +, -, =, <\}$ have the natural interpretation and D_k is interpreted as the sets of natural numbers divisible by k (these extra predicates are needed to get quantifier elimination [21]).

3 Monic-flat array property fragments

Although $\mathbb{P}$ represents the fragment of arithmetic mostly used in formal approaches for the static analysis of systems, we underline that there are many other fragments that have quantifier elimination and can be quite useful; these fragments can be both weaker (like Integer Difference Logic [20]) and stronger (like the exponentiation extension of Semënov theorem [24]) than $\mathbb{P}$. Thus, the *modular* approach proposed in this Section to model arrays is not motivated just by generalization purposes, but can have practical impact.

There exist two ways of introducing arrays in a declarative setting, the mono-sorted and the multi-sorted ways. The former is more expressive because (roughly speaking) it allows to consider indexes also as elements[3], but might be computationally more difficult to handle. We discuss decidability results for both cases, starting from the mono-sorted case.

3.1 The mono-sorted case

Let $T = (\Sigma, \mathcal{C})$ be a theory; the theory $\mathtt{ARR}^1(T)$ *of arrays over* T is obtained from T by adding to it infinitely many (fresh) free unary function symbols. This means that the signature of $\mathtt{ARR}^1(T)$ is obtained from Σ by adding to it unary function symbols (we use the letters $a, a_1, a_2, \ldots$ for them) and that a structure $\mathcal{M}$ is a model of $\mathtt{ARR}^1(T)$ iff (once the interpretations of the extra function symbols are disregarded) it is a structure belonging to the original class $\mathcal{C}$.

For array theories it is useful to introduce the following notation. We use $\mathbf{a}$ for a tuple $\mathbf{a} = a_1, \ldots, a_{|\mathbf{a}|}$ of distinct 'array constants' (i.e. free function symbols); if $\mathbf{t} = t_1, \ldots, t_{|\mathbf{t}|}$ is a tuple of terms, the notation $\mathbf{a}(\mathbf{t})$ represents the tuple (of length $|\mathbf{a}| \cdot |\mathbf{t}|$) of terms $a_1(t_1), \ldots, a_1(t_{|\mathbf{t}|}), \ldots, a_{|\mathbf{a}|}(t_1), \ldots, a_{|\mathbf{a}|}(t_{|\mathbf{t}|})$.

$\mathtt{ARR}^1(T)$ may be highly undecidable, even when T itself is decidable (see [17]), thus it is mandatory to limit the shape of the formulæ we want to try to decide.

[3]This is useful in the analysis of programs, when pointers to the memory (modeled as an array) are stored into array variables.

A prenex formula or a term in the signature of $\mathtt{ARR}^1(T)$ are said to be *flat* iff for every term of the kind $a(t)$ occurring in them (here a is any array constant), the sub-term t is always a variable. Notice that every formula is logically equivalent to a flat one; however the flattening transformations are based on rewriting as

$$\phi(a(t),...) \rightsquigarrow \exists x(x = t \wedge \phi(a(x),...) \quad \text{or} \quad \phi(a(t),...) \rightsquigarrow \forall x(x = t \rightarrow \phi(a(x),...)$$

and consequently they may alter the quantifiers prefix of a formula. Thus it must be kept in mind (when understanding the results below), that flattening transformation cannot be operated on any occurrence of a term without exiting from the class that is claimed to be decidable. When we indicate a flat quantifier-free formula with the notation $\psi(\mathbf{x}, \mathbf{a}(\mathbf{x}))$, we mean that such a formula is obtained from a Σ-formula of the kind $\psi(\mathbf{x}, \mathbf{z})$ (i.e. from a quantifier-free Σ-formula where at most the free variables $\mathbf{x}, \mathbf{z}$ can occur) by replacing $\mathbf{z}$ by $\mathbf{a}(\mathbf{x})$.

Theorem 1. *If the T-satisfiability of $\exists^*\forall\exists^*$ sentences is decidable, then the $\mathtt{ARR}^1(T)$-satisfiability of $\exists^*\forall$-flat sentences is decidable.*

Proof. We present an algorithm, $\mathsf{SAT}_{\mathsf{MONO}}$, for deciding the satisfiability of the $\exists^*\forall$-flat fragment of $\mathtt{ARR}^1(T)$ (we let T be $(\Sigma, \mathcal{C})$). Subsequently, we show that $\mathsf{SAT}_{\mathsf{MONO}}$ is sound and complete. From the complexity viewpoint, notice that $\mathsf{SAT}_{\mathsf{MONO}}$ produces a quadratic instance of a $\exists^*\forall\exists^*$-satisfiability problem.

The decision procedure $\mathsf{SAT}_{\mathsf{MONO}}$.

STEP I. Let

$$F := \exists \mathbf{c}\, \forall i.\psi(i, \mathbf{a}(i), \mathbf{c}, \mathbf{a}(\mathbf{c}))$$

be a $\exists^*\forall$-flat $\mathtt{ARR}^1(T)$-sentence, where ψ is a quantifier-free Σ-formula. Suppose that s is the length of $\mathbf{a}$ and t is the length of $\mathbf{c}$ (that is, $\mathbf{a} = a_1, \ldots, a_s$ and $\mathbf{c} = c_1, \ldots, c_t$). Let $\mathbf{e} = \langle e_{l,m} \rangle$ $(1 \le l \le s,\ 1 \le m \le t)$ be a tuple of length $s \cdot t$ of fresh variables and consider the $\mathtt{ARR}^1(T)$-formula:

$$F_1 := \exists \mathbf{c}\, \exists \mathbf{e}\, \forall i.\psi(i, \mathbf{a}(i), \mathbf{c}, \mathbf{e}) \wedge \bigwedge_{1 \le l \le t} \bigwedge_{1 \le m \le s} a_m(c_l) = e_{l,m}$$

STEP II. From F_1 build the formula

$$F_2 := \exists \mathbf{c}\, \exists \mathbf{e}\, \forall i. \left[\psi(i, \mathbf{a}(i), \mathbf{c}, \mathbf{e}) \wedge \bigwedge_{1 \le l \le t} (i = c_l \rightarrow \bigwedge_{1 \le m \le s} a_m(i) = e_{l,m}) \right]$$

STEP III. Let $\mathbf{d}$ be a fresh tuple of variables of length s; check the T-satisfiability of

$$F_3 := \exists \mathbf{c}\, \exists \mathbf{e}\, \forall i\, \exists \mathbf{d}. \left[\psi(i, \mathbf{d}, \mathbf{c}, \mathbf{e}) \wedge \bigwedge_{1 \le l \le t} (i = c_l \rightarrow \bigwedge_{1 \le m \le s} d_m = e_{l,m}) \right]$$

Correctness and completeness of $\mathsf{SAT}_{\mathsf{MONO}}$. $\mathsf{SAT}_{\mathsf{MONO}}$ transforms an $\mathtt{ARR}^1(T)$-formula F into an equisatisfiable T-formula F_3 belonging to the $\exists^*\forall\exists^*$ fragment. More precisely, it holds that F, F_1 and F_2 are equivalent formulæ, because

$$\bigwedge_{1\le l\le t} \forall i.(i = c_l \rightarrow \bigwedge_{1\le m\le s} a_m(i) = e_{l,m}) \equiv \bigwedge_{1\le l\le t}\bigwedge_{1\le m\le s} a_m(c_l) = e_{l,m}$$

From F_2 to F_3 and back, satisfiability is preserved because F_2 is the Skolemization of F_3, where the existentially quantified variables $\mathbf{d} = d_1, \ldots, d_s$ are substituted with the free unary function symbols $\mathbf{a} = a_1, \ldots a_s$. $\dashv$

Since Presburger Arithmetic is decidable (via quantifier elimination), we get in particular that

Corollary 1. *The* $\mathtt{ARR}^1(\mathbb{P})$*-satisfiability of* $\exists^*\forall$*-flat sentences is decidable.*

As another example matching the hypothesis of Theorem 1 (i.e. as an example of a T such that T-satisfiability of $\exists^*\forall\exists^*$-sentences is decidable) consider pure first order logic with equality in a signature with predicate symbols of any arity but with only unary function symbols [6].

3.2 The multi-sorted case

We are now considering a theory of arrays parametric in the theories specifying constraints over indexes and elements of the arrays. Formally, we need two ingredient theories, $T_I = (\Sigma_I, \mathcal{C}_I)$ and $T_E = (\Sigma_E, \mathcal{C}_E)$. We can freely assume that Σ_I and Σ_E are disjoint (otherwise we can rename some symbols); for simplicity, we let both signatures be mono-sorted (but extending our results to many-sorted T_E is quite straightforward): let us call `INDEX` the unique sort of T_I and `ELEM` the unique sort of T_E.

The theory $\mathtt{ARR}^2(T_I, T_E)$ *of arrays over* T_I *and* T_E is obtained from the union of $\Sigma_I \cup \Sigma_E$ by adding to it infinitely many (fresh) free unary function symbols (these new function symbols will have domain sort `INDEX` and codomain sort `ELEM`). The models of $\mathtt{ARR}^2(T_I, T_E)$ are the structures whose reducts to the symbols of sorts `INDEX` and `ELEM` are models of T_I and T_E, respectively.

Consider now an atomic formula $P(t_1, \ldots, t_n)$ in the language of $\mathtt{ARR}^2(T_I, T_E)$ (in the typical situation, P is the equality predicate). Since the predicate symbols of $\mathtt{ARR}^2(T_I, T_E)$ are from $\Sigma_I \cup \Sigma_E$ and $\Sigma_I \cap \Sigma_E = \emptyset$, P belongs either to Σ_I or to Σ_E; in the latter case, all terms t_i have sort `ELEM` and in the former case all terms t_i are Σ_I-terms. We say that $P(t_1, \ldots, t_n)$ is an `INDEX`-*atom* in the former case and that it is an `ELEM`-*atom* in the latter case.

When dealing with $\mathtt{ARR}^2(T_I, T_E)$, *we shall limit ourselves to quantified variables of sort* `INDEX`: this limitation is justified by the benchmarks arising in applications (see Section 4).[4] A sentence in the language of $\mathtt{ARR}^2(T_I, T_E)$ is said to be *monic* iff it is in prenex form and every `INDEX` atom occurring in it contains at most one variable falling within the scope of a *universal* quantifier.

[4] Topmost existentially quantified variables of sort `ELEM` can be modeled by enriching T_E with free constants.

Example 1. Consider the following sentences:

$(I)\ \forall i.\, a(i) = i;$ $(II)\ \forall i_1 \forall i_2.\, (i_1 \leq i_2 \to a(i_1) \leq a(i_2));$

$(III)\ \exists i_1 \exists i_2.\, (i_1 \leq i_2 \wedge a(i_1) \not\leq a(i_2));$ $(IV)\ \forall i_1 \forall i_2.\, a(i_1) = a(i_2);$

$(V)\ \forall i.\, (D_2(i) \to a(i) = 0);$ $(VI)\ \exists i\, \forall j.\, (a_1(j) < a_2(3i)).$

The flat formula (I) is not well-typed, hence it is not allowed in $\mathtt{ARR}^2(\mathbb{P}, \mathbb{P})$; however, it is allowed in $\mathtt{ARR}^1(\mathbb{P})$. Formula (II) expresses the fact that the array a is sorted: it is flat but not monic (because of the atom $i_1 \leq i_2$). On the contrary, its negation (III) is flat and monic (because i_1, i_2 are now existentially quantified). Formula (IV) expresses that the array a is constant; it is flat and monic (notice that the universally quantified variables i_1, i_2 both occur in $a(i_1) = a(i_2)$ but the latter is an ELEM atom). Formula (V) expresses that a is initialized so to have all even positions equal to 0: it is monic and flat. Formula (VI) is monic but not flat because of the term $a_2(3i)$ occurring in it; however, in $3i$ no universally quantified variable occurs, so it is possible to produce by flattening the following sentence

$$\exists i\, \exists i'\, \forall j\ (i' = 3i \wedge a_1(j) < a_2(i'))$$

which is logically equivalent to (VI), it is flat and still lies in the $\exists^*\forall$-class. Finally, as a more complicated example, notice that the following sentence

$$\exists k\, \forall i.\ (D_2(k) \wedge a(k) = \text{`}\backslash\mathtt{0}\text{`} \wedge (D_2(i) \wedge i < k \to a(i) = \text{`}\mathtt{b}\text{`}) \wedge (\neg D_2(i) \wedge i < k \to a(i) = \text{`}\mathtt{c}\text{`}))$$

is monic and flat: it says that a represents a string of the kind $(\mathtt{bc})^*$.

Theorem 2. *If T_I-satisfiability of $\exists^*\forall$-sentences is decidable, then $\mathtt{ARR}^2(T_I, T_E)$-satisfiability of $\exists^*\forall^*$-monic-flat sentences is decidable.*

Proof. As we did for $\mathsf{SAT}_{\mathsf{MONO}}$, we give a decision procedure, $\mathsf{SAT}_{\mathsf{MULTI}}$, for the $\exists^*\forall^*$-monic-flat fragment of $\mathtt{ARR}^2(T_I, T_E)$; for space reasons, we give here just some informal justifications, the reader is referred to [2] for proofs. First (STEP I), the procedure *guesses* the sets (called 'types') of relevant INDEX atoms satisfied in a model to be built. Subsequently (STEP II) it introduces a representative variable for each type together with the constraint that guessed types are exhaustive. Finally (STEP III, IV and V) the procedure applies combination techniques for purification. ⊣

The decision procedure $\mathsf{SAT}_{\mathsf{MULTI}}$. The algorithm is non-deterministic: the input formula is satisfiable iff we can guess suitable data $\mathcal{T}, \mathcal{B}$ so that the formulæ F_I, F_E below are satisfiable.

STEP I. Let F be a $\exists^*\forall^*$-monic-flat formula; let it be

$$F := \exists \mathbf{c}\, \forall \mathbf{i}.\psi(\mathbf{i}, \mathbf{a}(\mathbf{i}), \mathbf{c}, \mathbf{a}(\mathbf{c})),$$

(where as usual ψ is a $T_I \cup T_E$-quantifier-free formula). Suppose $\mathbf{a} = a_1, \ldots, a_s$, $\mathbf{i} = i_1, \ldots, i_n$ and $\mathbf{c} = c_1, \ldots, c_t$. Consider the set (notice that all atoms in K are Σ_I-atoms and have just one free variable because F is monic)

$$K = \{A(x, \mathbf{c}) \mid A(i_k, \mathbf{c}) \text{ is an } \mathtt{INDEX} \text{ atom of } F\}_{1 \leq k \leq n} \cup \{x = c_l\}_{1 \leq l \leq t}$$

Let us call *type* a set of literals M such that: (i) each literal of M is an atom in K or its negation; (ii) for all $A(x, \mathbf{c}) \in K$, either $A(x, \mathbf{c}) \in M$ or $\neg A(x, \mathbf{c}) \in M$. Guess a set $\mathcal{T} = \{M_1, \ldots, M_q\}$ of types.

STEP II. Let $\mathbf{b} = b_1, \ldots, b_q$ be a tuple of new variables of sort `INDEX` and let

$$F_1 := \exists \mathbf{b}\, \exists \mathbf{c} \left[\begin{array}{l} \forall x. \left(\bigvee_{j=1}^{q} \bigwedge_{L \in M_j} L(x, \mathbf{c}) \right) \wedge \\ \bigwedge_{j=1}^{q} \bigwedge_{L \in M_j} L(b_j, \mathbf{c}) \wedge \\ \bigwedge_{\sigma : \mathbf{i} \to \mathbf{b}} \psi(\mathbf{i}\sigma, \mathbf{a}(\mathbf{i}\sigma), \mathbf{c}, \mathbf{a}(\mathbf{c})) \end{array} \right]$$

where $\mathbf{i}\sigma$ is the tuple of terms $\sigma(i_1), \ldots, \sigma(i_n)$.

STEP III. Let $\mathbf{e} = \langle e_{l,m} \rangle$ $(1 \le l \le s, 1 \le m \le t+q)$ be a tuple of length $s \cdot (t+q)$ of free constants of sort `ELEM`. Consider the formula

$$F_2 := \exists \mathbf{b}\, \exists \mathbf{c} \left[\begin{array}{l} \forall x. \left(\bigvee_{j=1}^{q} \bigwedge_{L \in M_j} L(x, \mathbf{c}) \right) \wedge \\ \bigwedge_{j=1}^{q} \bigwedge_{L \in M_j} L(b_j, \mathbf{c}) \wedge \\ \bar{\psi}(\mathbf{b}, \mathbf{c}, \mathbf{e}) \wedge \\ \bigwedge_{d_m, d_n \in \mathbf{b} * \mathbf{c}} \bigwedge_{l=1}^{s} (d_m = d_n \to e_{l,m} = e_{l,n}) \end{array} \right]$$

where $\mathbf{b} * \mathbf{c} := d_1, \ldots, d_{q+t}$ is the concatenation of the tuples $\mathbf{b}$ and $\mathbf{c}$ and $\bar{\psi}(\mathbf{b}, \mathbf{c}, \mathbf{e})$ is obtained from

$$\bigwedge_{\sigma : \mathbf{i} \to \mathbf{b}} \psi(\mathbf{i}\sigma, \mathbf{a}(\mathbf{i}\sigma), \mathbf{c}, \mathbf{a}(\mathbf{c}))$$

by substituting each term in the tuple $\mathbf{a}(\mathbf{b}) * \mathbf{a}(\mathbf{c})$ with the constant occupying the corresponding position in the tuple $\mathbf{e}$.

STEP IV. Let $\mathcal{B}$ a full Boolean satisfying assignment for the atoms of the formula

$$F_3 := \bar{\psi}(\mathbf{b}, \mathbf{c}, \mathbf{e}) \;\wedge \bigwedge_{d_m, d_n \in \mathbf{b} * \mathbf{c}} \bigwedge_{l=1}^{s} (d_m = d_n \to e_{l,m} = e_{l,n})$$

and let $\bar{\psi}_I(\mathbf{b}, \mathbf{c}), \bar{\psi}_E(\mathbf{e})$ be the (conjunction of the) sets of literals of sort `INDEX` and `ELEM`, respectively, induced by $\mathcal{B}$.

STEP V. Check the T_I-satisfiability of

$$F_I := \exists \mathbf{b}\, \exists \mathbf{c}. \left[\forall x. \left(\bigvee_{j=1}^{q} \bigwedge_{L \in M_j} L(x, \mathbf{c}) \right) \wedge \bigwedge_{j=1}^{q} \bigwedge_{L \in M_j} L(b_j, \mathbf{c}) \wedge \bar{\psi}_I(\mathbf{b}, \mathbf{c}) \right]$$

and the T_E-satisfiability of

$$F_E := \bar{\psi}_E(\mathbf{e})$$

Notice that F_I is an $\exists^*\forall$-sentence; F_E is ground and the T_E-satisfiability of F_E (considering the $\mathbf{e}$ as variables instead of as free constants) is decidable because we assumed that all the theories we consider (hence our T_E too) have quantifier-free fragments decidable for satisfiability.

Theorem 2 applies to $\mathtt{ARR}^2(\mathbb{P}, \mathbb{P})$ because $\mathbb{P}$ admits quantifier elimination. For this theory, we can determine complexity upper and lower bounds:

Theorem 3. $\mathtt{ARR}^2(\mathbb{P}, \mathbb{P})$*-satisfiability of* $\exists^*\forall^*$*-monic-flat sentences is* NExpTime*-complete.*

Proof. We use exponentially bounded domino systems for reduction [6,19], see [2] for details. $\dashv$

4 A decidability result for the reachability analysis of flat array programs

Based on the decidability results described in the previous section, we can now achieve important decidability results in the context of reachability analysis for programs handling arrays of unbounded length. As a reference theory, we shall use $\mathtt{ARR}^1(\mathbb{P}^+)$ or $\mathtt{ARR}^2(\mathbb{P}^+, \mathbb{P}^+)$, where $\mathbb{P}^+$ is $\mathbb{P}$ enriched with free constant symbols and with *definable* predicate and function symbols. We do not enter into more details concerning what a definable symbol is (see, e.g., [25]), we just underline that definable symbols are nothing but useful macros that can be used to formalize case-defined functions and SMT-LIB commands like if-then-else. The addition of definable symbols does not compromise quantifier elimination, hence decidability of $\mathbb{P}^+$. Below, we let $\mathcal{T}$ be $\mathtt{ARR}^1(\mathbb{P}^+)$ or $\mathtt{ARR}^2(\mathbb{P}^+, \mathbb{P}^+)$.

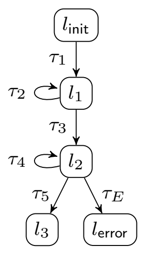
Henceforth $\mathbf{v}$ will denote, in the following, the variables of the programs we will analyze. Formally, $\mathbf{v} = \mathbf{a}, \mathbf{c}$ where, according to our conventions, $\mathbf{a}$ is a tuple of array variables (modeled as free unary function symbols of $\mathcal{T}$ in our framework) and $\mathbf{c}$ a tuple of scalar variables; the latter can be modeled as variables in the logical sense - in $\mathtt{ARR}^2(\mathbb{P}^+, \mathbb{P}^+)$ we can model them either as variables of sort `INDEX` or as free constants of sort `ELEM`.

A *state-formula* is a formula $\alpha(\mathbf{v})$ of $\mathcal{T}$ representing a (possibly infinite) set of configurations of the program under analysis. A *transition formula* is a formula of $\mathcal{T}$ of the kind $\tau(\mathbf{v}, \mathbf{v}')$ where $\mathbf{v}'$ is obtained from copying the variables in $\mathbf{v}$ and adding a prime to each of them. For the purpose of this work, programs will be represented by their control-flow automaton.

Definition 1 (Programs). *Given a set of variables* $\mathbf{v}$*, a* program *is a triple* $\mathcal{P} = (L, \Lambda, E)$*, where (i)* $L = \{l_1, \ldots, l_n\}$ *is a set of* program locations *among which we distinguish an initial location* l_{init} *and an error location* l_{error}*; (ii)* Λ *is a finite set of transition formulæ* $\{\tau_1(\mathbf{v}, \mathbf{v}'), \ldots, \tau_r(\mathbf{v}, \mathbf{v}')\}$ *and (iii)* $E \subseteq L \times \Lambda \times L$ *is a set of* actions*.*

```
     procedure initEven ( a[N] , v ) :
l1      for (i = 0; i < N; i = i + 2)  a[i] = v;
l2      for (i = 0; i < N; i = i + 2)  assert(a[i] = v);
```

(a)



(b)

Fig. 1. The initEven procedure (a) and its control-flow graph (b).

We indicate by $src, \mathcal{L}, trg$ the three projection functions on E; that is, for $e = (l_i, \tau_j, l_k) \in E$, we have $src(e) = l_i$ (this is called the 'source' location of e), $\mathcal{L}(e) = \tau_j$ (this is called the 'label' of e) and $trg(e) = l_k$ (this is called the 'target' location of e).

Example 2. Consider the procedure initEven in Fig. 1. For this procedure, $\mathbf{a} = a$, $\mathbf{c} = i, v$. N is a constant of the background theory. Λ is the set of formulæ (we omit identical updates):

$$\begin{aligned}
&\tau_1 := i' = 0\\
&\tau_2 := i < N \wedge a' = \lambda j.\mathsf{if}\ (j = i)\ \mathsf{then}\ v\ \mathsf{else}\ a(j) \wedge i' = i + 2\\
&\tau_3 := i \geq N \wedge i' = 0\\
&\tau_4 := i < N \wedge a(i) = v \wedge i' = i + 2\\
&\tau_5 := i \geq N\\
&\tau_E := i < N \wedge a(i) \neq v
\end{aligned}$$

The procedure initEven can be formalized as the control-flow graph depicted in Fig. 1(b), where $L = \{l_{\mathsf{init}}, l_1, l_2, l_3, l_{\mathsf{error}}\}$.

Definition 2 (Program paths). *A* program path *(in short,* path*) of $\mathcal{P} = (L, \Lambda, E)$ is a sequence $\rho \in E^n$, i.e., $\rho = e_1, e_2, \ldots, e_n$, such that for every e_i, e_{i+1}, $trg(e_i) = src(e_{i+1})$. We denote with $|\rho|$ the length of the path. An* error path *is a path ρ with $src(e_1) = l_{\mathsf{init}}$ and $trg(e_{|\rho|}) = l_{\mathsf{error}}$. A path ρ is a* feasible path *if $\bigwedge_{j=1}^{|\rho|} \mathcal{L}(e_j)^{(j)}$ is $\mathcal{T}$-satisfiable, where $\mathcal{L}(e_j)^{(j)}$ represents $\tau_{i_j}(\mathbf{v}^{(j-1)}, \mathbf{v}^{(j)})$, with $\mathcal{L}(e_j) = \tau_{i_j}$.*

The *(unbounded) reachability problem* for a program $\mathcal{P}$ is to detect if $\mathcal{P}$ admits a feasible error path. Proving the safety of $\mathcal{P}$, therefore, means solving the reachability problem for $\mathcal{P}$. This problem, given well known limiting results, is not decidable for an arbitrary program $\mathcal{P}$. The consequence is that, in general, reachability analysis is sound, but not complete, and its incompleteness manifests itself in (possible) divergence of the verification algorithm (see, e.g., [1]).

To gain decidability, we must first impose restrictions on the shape of the transition formulæ, for instance we can constrain the analysis to formulæ falling within decidable classes like those we analyzed in the previous section. This is not sufficient however, due to the presence of loops in the control flow. Hence we assume flatness conditions on such control flow and "accelerability" of the transitions labeling self-loops. This is similar to what is done in [7, 9, 12] for integer variable programs, but since we handle array variables we need specific restrictions for acceleration. Our result for the decidability of the safety of annotated array programs builds upon the results presented in Section 3 and the acceleration procedure presented in [3].

We first give the definition of flat^0-program, i.e., programs with only self-loops for which each location belongs to at most one loop. Subsequently we will identify sufficient conditions for achieving the full decidability of the reachability problem for flat^0-programs.

Definition 3 (flat^0-program). *A program $\mathcal{P}$ is a* flat^0-program *if for every path $\rho = e_1, \ldots, e_n$ of $\mathcal{P}$ it holds that for every $j < k$ ($j, k \in \{1, \ldots, n\}$), if $src(e_j) = trg(e_k)$ then $e_j = e_{j+1} = \cdots = e_k$.*

We now turn our attention to transition formulæ. Acceleration is a well-known formalism in the area of model-checking. It has been integrated in several frameworks and constitutes a fundamental technology for the scalability and efficiency of modern model checkers (e.g., [5]). Given a loop, represented as a transition relation τ, the accelerated transition τ^+ allows to compute *in one shot* the *precise* set of states reachable after n unwindings of that loop, for any n. This prevents divergence of the reachability analysis along τ, caused by its unwinding. What prevents the applicability of acceleration in the domain we are targeting is that accelerations are not always definable. By definition, the acceleration of a transition $\tau(\mathbf{v}, \mathbf{v}')$ is the union of the n-th compositions of τ with itself, i.e. it is $\tau^+ := \bigvee_{n>0} \tau^n$, where

$$\tau^1(\mathbf{v}, \mathbf{v}') := \tau(\mathbf{v}, \mathbf{v}'), \qquad \tau^{n+1}(\mathbf{v}, \mathbf{v}') := \exists \mathbf{v}''.(\tau(\mathbf{v}, \mathbf{v}'') \wedge \tau^n(\mathbf{v}'', \mathbf{v}')) \ .$$

τ^+ can be practically exploited only if there exists a formula $\varphi(\mathbf{v}, \mathbf{v}')$ equivalent, modulo the considered background theory, to $\bigvee_{n>0} \tau^n$. Based on this observation on definability of accelerations, we are now ready to state a general result about the decidability of the reachability problem for programs with arrays. The theorem we give is, as we did for results in Section 3, modular and general. We will show an instance of this result in the following section. Notationally, let us extend the projection function $\mathcal{L}$ by denoting $\mathcal{L}^+(e) := \mathcal{L}(e)^+$ if $src(e) = trg(e)$ and $\mathcal{L}^+(e) := \mathcal{L}(e)$ otherwise, where $\mathcal{L}(e)^+$ denotes the acceleration of the transition labeling the edge e.

Theorem 4. *Let $\mathcal{F}$ be a class of formulæ decidable for $\mathcal{T}$-satisfiability. The unbounded reachability problem for a flat^0-program $\mathcal{P}$ is decidable if (i) $\mathcal{F}$ is closed under conjunctions and (ii) for each $e \in E$ one can compute $\alpha(\mathbf{v}, \mathbf{v}') \in \mathcal{F}$ such that $\mathcal{T} \models \mathcal{L}^+(e) \leftrightarrow \alpha(\mathbf{v}, \mathbf{v}')$,*

Proof. Let $\rho = e_1, \ldots, e_n$ be an error path of $\mathcal{P}$; when testing its feasibility, according to Definition 3, we can limit ourselves to the case in which $e_1, \ldots, e_n$

are all distinct, provided we replace the labels $\mathcal{L}(e_k)^{(k)}$ with $\mathcal{L}^+(e_k)^{(k)}$ in the formula $\bigwedge_{j=1}^n \mathcal{L}(e_j)^{(j)}$ from Definition 2.[5] Thus $\mathcal{P}$ is unsafe iff, for some path $e_1, \ldots, e_n$ whose edges are all distinct, the formula

$$\mathcal{L}^+(e_1)^{(1)} \wedge \cdots \wedge \mathcal{L}^+(e_n)^{(n)} \tag{1}$$

is $\mathcal{T}$-satisfiable. Since the involved paths are finitely many and $\mathcal{T}$-satisfiability of formulæ like (1) is decidable, the safety of $\mathcal{P}$ can be decided. $\dashv$

4.1 A class of array programs with decidable reachability problem

We now produce a class of programs with arrays – we call it $\mathsf{simple}^0_{\mathcal{A}}$-programs– for which requirements of Theorem 4 are met. The class of $\mathsf{simple}^0_{\mathcal{A}}$-programs contains non recursive programs implementing searching, copying, comparing, initializing, replacing and testing procedures. As an example, the $\mathsf{initEven}$ program reported in Fig. 1 is a $\mathsf{simple}^0_{\mathcal{A}}$-program. Formally, a $\mathsf{simple}^0_{\mathcal{A}}$*-program* $\mathcal{P} = (L, \Lambda, E)$ is a flat^0-program such that (i) every $\tau \in \Lambda$ is a formula belonging to one of the decidable classes covered by Corollary 1 or Theorem 3; (ii) if $e \in E$ is a self-loop, then $\mathcal{L}(e)$ is a simple_k-assignment.

Simple_k-assignments are transitions (defined below) for which the acceleration is first-order definable and is a Flat Array Property. For a natural number k, we denote by $\bar{k}$ the term $1 + \cdots + 1$ (k-times) and by $\bar{k} \cdot t$ the term $t + \cdots + t$ (k-times).

Definition 4 (simple_k-assignment). *Let $k \geq 0$; a* simple_k-assignment *is a transition $\tau(\mathbf{v}, \mathbf{v}')$ of the kind*

$$\phi_L(\mathbf{c}, \mathbf{a}[d]) \wedge d' = d + \bar{k} \wedge \mathbf{d}' = \mathbf{d} \wedge \mathbf{a}' = \lambda j.\mathsf{if}\ (j = d)\ \mathsf{then}\ \mathbf{t}(\mathbf{c}, \mathbf{a}(d))\ \mathsf{else}\ \mathbf{a}(j)$$

where (i) $\mathbf{c} = d, \mathbf{d}$ and (ii) the formula $\phi_L(\mathbf{c}, \mathbf{a}[d])$ and the terms $\mathbf{t}(\mathbf{c}, \mathbf{a}[d])$ are flat.

The following Lemma (which is an instance of a more general result from [3]) gives the template for the accelerated counterpart of a simple_k-assignment.

Lemma 1. *Let $\tau(\mathbf{v}, \mathbf{v}')$ be a simple_k-assignment. Then $\tau^+(\mathbf{v}, \mathbf{v}')$ is $\mathcal{T}$-equivalent to the formula*

$$\exists y > 0 \begin{pmatrix} \forall z. \left(\left(d \leq z < d + \bar{k} \cdot y \wedge D_{\bar{k}}(z - d) \right) \to \ \phi_L(z, \mathbf{d}, \mathbf{a}(d)) \right) \ \wedge \\ \mathbf{a}' = \lambda j.\mathbf{U}(j, y, \mathbf{v}) \ \wedge \ d' = d + \bar{k} \cdot y \ \wedge \ \mathbf{d}' = \mathbf{d} \end{pmatrix}$$

where the definable functions $U_h(j, y, \mathbf{v})$, $1 \leq h \leq s$ of the tuple $\mathbf{U}$ are

$$\mathsf{if}\ (d \leq j < d + \bar{k} \cdot y \ \wedge D_{\bar{k}}(j - d))\ \mathsf{then}\ t_h(j, \mathbf{d}, \mathbf{a}(j))\ \mathsf{else}\ a_h(j)\ .$$

[5] Notice that by these replacements we can represent in one shot infinitely many paths, namely those executing self-loops any given number of times.

Example 3. Consider transition τ_2 from the formalization of our running example of Fig. 1. The acceleration τ_2^+ of such formula is (we omit identical updates)

$$\exists y > 0. \begin{pmatrix} \forall z.(i \leq z < i + 2y \land D_2(z - i) \rightarrow z < N) \land i' = i + 2y \land \\ a' = \lambda j.\,(\mathsf{if}\ \ (i \leq j < 2y + i \land D_2(j - i))\ \ \mathsf{then}\ v\ \mathsf{else}\ a[j]) \end{pmatrix}$$

We can now formally show that the reachability problem for $\mathsf{simple}^0_{\mathcal{A}}$-programs is decidable, by instantiating Theorem 4 with the results obtained so far.

Theorem 5. *The unbounded reachability problem for* $\mathsf{simple}^0_{\mathcal{A}}$*-programs is decidable.*

Proof. By prenex transformations, distributions of universal quantifiers over conjunctions, etc., it is easy to see that the decidable classes covered by Corollary 1 or Theorem 3 are closed under conjunctions. Since the acceleration of a simple_k-assignment fits inside these classes (just eliminate definitions via λ-abstractions by using universal quantifiers), Theorem 4 applies. ⊣

4.2 Experimental observations

We evaluated the capabilities of available SMT-Solvers on checking the satisfiability of Flat Array Properties and for that we selected some $\mathsf{simple}^0_{\mathcal{A}}$-programs, both safe and unsafe. Following the procedure identified in the proof of Theorem 4 we generated 200 SMT-LIB2-compliant files with Flat Array Properties[6]. The $\mathsf{simple}^0_{\mathcal{A}}$-programs we selected perform some simple manipulations on arrays of unknown length, like searching for a given element, initializing the array, swapping the arrays, copying one array into another, etc. We tested CVC4 [4] (version 1.2) and Z3 [10] (version 4.3.1) on the generated SMT-LIB2 files. Experimentation has been performed on a machine equipped with a 2.66 GHz CPU and 4GB of RAM running Mac OSX 10.8.5. From our evaluation, both tools timeout on some proof-obligations[7]. These results suggest that the fragment of Flat Array Properties definitely identifies fragments of theories which are decidable, but their satisfiability is still not entirely covered by modern and highly engineered tools.

5 Conclusions and related work

In this paper we identified a class of Flat Array Properties, a quantified fragment of theories of arrays, admitting decision procedures. Our results are parameterized in the theories used to model indexes and elements of the array; in this sense, there is some similarity with [18], although (contrary to [18]) we consider purely syntactically specified classes of formulæ. We provided a complexity analysis of our decision procedures. We also showed that the decidability of Flat Array

[6]Such files have been generated automatically with our prototype tool which we make available at `www.inf.usi.ch/phd/alberti/prj/booster`.

[7]See the discussion in [2] for more information on the experiments.

Properties, combined with acceleration results, allows to depict a sound and complete procedure for checking the safety of a class of programs with arrays.

The modular nature of our solution makes our contributions orthogonal with respect to the state of the art: we can enrich $\mathbb{P}$ with various definable or even not definable symbols [24] and get from our Theorems 1,2 decidable classes which are far from the scope of existing results. Still, it is interesting to notice that also the special cases of the decidable classes covered by Corollary 1 and Theorem 3 are orthogonal to the results from the literature. To this aim, we make a closer comparison with [8,15]. The two fragments considered in [8,15] are characterized by rather restrictive syntactic constraints. In [15] it is considered a subclass of the $\exists^*\forall$-fragment of $\mathtt{ARR}^1(T)$ called SIL, *Single Index Logic.* In this class, formulæ are built according to a grammar allowing (i) as atoms only difference logic constraints and some equations modulo a fixed integer and (ii) as universally quantified subformulæ only formulæ of the kind $\forall \mathbf{i}.\phi(\mathbf{i}) \to \psi(\mathbf{i}, \mathbf{a}(\mathbf{i} + \bar{\mathbf{k}}))$ (here $\mathbf{k}$ is a tuple of integers) where ϕ, ψ are conjunctions of atoms (in particular, no disjunction is allowed in ψ). On the other side, SIL includes some non-flat formulæ, due to the presence of constant increment terms $\mathbf{i} + \bar{\mathbf{k}}$ in the consequents of the above universally quantified implications. Similar restrictions are in [16]. The Array Property Fragment described in [8] is basically a subclass of the $\exists^*\forall^*$-fragment of $\mathtt{ARR}^2(\mathbb{P}, \mathbb{P})$; however universally quantified subformulæ are constrained to be of the kind $\forall \mathbf{i}.\phi(\mathbf{i}) \to \psi(\mathbf{a}(\mathbf{i}))$, where in addition the $\mathtt{INDEX}$ part $\phi(\mathbf{i})$ must be a conjunction of atoms of the kind $i \leq j, i \leq t, t \leq i$ (with $i, j \in \mathbf{i}$ and where t does not contain occurrences of the universally quantified variables $\mathbf{i}$). These formulæ are flat but not monic because of the atoms $i \leq j$.

From a computational point of view, a complexity bound for $\mathsf{SAT}_{\mathsf{MONO}}$ has been shown in the proof of Theorem 1, while the complexity of the decision procedure proposed in [15] is unknown. On the other side, both $\mathsf{SAT}_{\mathsf{MULTI}}$ and the decision procedure described in [8] run in NExpTime (the decision procedure in [8] is in NP only if the number of universally quantified index variables is bounded by a constant N). Our decision procedures for quantified formulæ are also partially different, in spirit, from those presented so far in the SMT community. While the vast majority of SMT-Solvers address the problem of checking the satisfiability of quantified formulæ via instantiation (see, e.g., [8,11,14,23]), our procedures – in particular $\mathsf{SAT}_{\mathsf{MULTI}}$ – are still based on instantiation, but the instantiation refers to a set of terms enlarged with the free constants witnessing the guessed set of realized types.

From the point of view of the applications, providing a full decidability result for the unbounded reachability analysis of a class of array programs is what differentiates our work with other contributions like [1,3].

References

1. F. Alberti, R. Bruttomesso, S. Ghilardi, S. Ranise, and N. Sharygina. Lazy abstraction with interpolants for arrays. In *LPAR*, pages 46–61, 2012.
2. F. Alberti, S. Ghilardi, and N. Sharygina. Decision procedures for flat array properties. Technical Report 2013/04, University of Lugano, oct 2013. Available at `http://www.inf.usi.ch/research_publication.htm?id=77`.

3. F. Alberti, S. Ghilardi, and N. Sharygina. Definability of accelerated relations in a theory of arrays and its applications. In *FroCoS*, pages 23–39, 2013.
4. C. Barrett, C.L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In *CAV*, pages 171–177, 2011.
5. G. Behrmann, J. Bengtsson, A. David, K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL implementation secrets. In *FTRTFT*, pages 3–22, 2002.
6. E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997.
7. M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, (91):275–303, 2009.
8. A.R. Bradley, Z. Manna, and H.B. Sipma. What's decidable about arrays? In *VMCAI*, pages 427–442, 2006.
9. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV*, volume 1427 of *LNCS*, pages 268–279. Springer, 1998.
10. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.
11. D.L. Detlefs, G. Nelson, and J.B. Saxe. Simplify: a theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003.
12. A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *FSTTCS*, pages 145–156, 2002.
13. H. Ganzinger. Shostak light. In *Automated deduction—CADE-18*, volume 2392 of *Lecture Notes in Comput. Sci.*, pages 332–346. Springer, Berlin, 2002.
14. Y. Ge and L. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *CAV*, pages 306–320, 2009.
15. P. Habermehl, R. Iosif, and T. Vojnar. A logic of singly indexed arrays. In *LPAR*, pages 558–573, 2008.
16. P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In *FOSSACS*, 2008.
17. J.Y. Halpern. Presburger arithmetic with unary predicates is Π^1_1 complete. *J. Symbolic Logic*, 56(2):637–642, 1991.
18. C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In *TACAS*, pages 265–281. Springer, 2008.
19. H.B. Lewis. Complexity of solvable cases of the decision problem for the predicate calculus. In *19th Ann. Symp. on Found. of Comp. Sci.*, pages 35–47. IEEE, 1978.
20. R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic. In *CAV'05*, pages 321–334, 2005.
21. D.C. Oppen. A superexponential upper bound on the complexity of Presburger arithmetic. *J. Comput. System Sci.*, 16(3):323–332, 1978.
22. S. Ranise and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2006.
23. A. Reynolds, C. Tinelli, A. Goel, S. Krstic, M. Deters, and C. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In *CADE*, pages 377–391, 2013.
24. A.L. Semënov. Logical theories of one-place functions on the set of natural numbers. *Izvestiya: Mathematics*, 22:587–618, 1984.
25. J.R. Shoenfield. *Mathematical logic.* Association for Symbolic Logic, Urbana, IL, 2001. Reprint of the 1973 second printing.
26. C. Tinelli and C.G. Zarba. Combining nonstably infinite theories. *J. Automat. Reason.*, 34(3):209–238, 2005.