Università della Svizzera italiana

Faculty of Informatics

USI Technical Report Series in Informatics

Decision Procedures for Flat Array Properties

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

[1] Faculty of Informatics, University of Lugano, Switzerland
[2] Università degli Studi, 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.

Report Info

Published
October 2013

Revised
January 2014

Number
USI-INF-TR-2013-04

Institution
Faculty of Informatics
University of Lugano
Lugano, Switzerland

Online Access
www.inf.usi.ch/techreports

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 [16]. 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 [7, 14, 15] (we shall defer the technical comparison with these contributions to Section 5). Here we explain the *modularity* character of our 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 in-

dexes 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 $\mathsf{simple}^0_{\mathscr{A}}$-*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 [6, 8, 11]): 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 [2]. The key point is that the shape of most accelerated transitions from [2] 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 $\mathsf{simple}^0_{\mathscr{A}}$-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 [2], 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 $\mathsf{simple}^0_{\mathscr{A}}$-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 [7, 14].

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 [21], a theory T is a pair $(\Sigma, \mathscr{C})$, where Σ is a signature and $\mathscr{C}$ is a class of Σ-structures; the structures in $\mathscr{C}$ are called the models of T. Given a Σ-structure $\mathscr{M}$, we denote by $S^{\mathscr{M}}, f^{\mathscr{M}}, P^{\mathscr{M}}, \ldots$ the interpretation in $\mathscr{M}$ 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 $\mathscr{M}$ in $\mathscr{C}$ such that α is true in $\mathscr{M}$ under a suitable assignment to the free variables of α (in symbols, $\mathscr{M} \models \alpha$); 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 $\alpha_1 \longleftrightarrow \alpha_2$ is T-valid; α_1 T-entails α_2 (in symbols, $\alpha_1 \models_T \alpha_2$) iff $\alpha_1 \rightarrow \alpha_2$ 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 [12, 25]).

A theory $T = (\Sigma, \mathscr{C})$ 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 $T \models \forall \mathbf{x}.(\alpha(\mathbf{x}) \longleftrightarrow \varphi(\mathbf{x}))$. 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 [20]).

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 [19]) and stronger (like the exponentiation extension of Semënov theorem [23]) 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[1], 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 $T = (\Sigma, \mathscr{C})$ 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 $\mathscr{M}$ 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 $\mathscr{C}$.

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 [16]), thus it is mandatory to limit the shape of the formulæ we want to try to decide. 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

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

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 3.1. *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.*

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

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 $(\Sigma, \mathscr{C})$). 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.

3.1.1 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 $\mathbf{e} = \langle e_{l,m} \rangle$ $(1 \leq l \leq s, 1 \leq m \leq t)$ be a tuple of length $s \cdot t$ of fresh variables and consider the $\mathtt{ARR}^1(T)$-formula:

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

STEP II. From F_1 build the formula

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

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

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

3.1.2 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

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

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 $\mathbf{a} = a_1, \ldots a_s$. ⊣ □

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

Corollary 3.1. *The* $\mathtt{ARR}^1(\mathbb{P})$*-satisfiability of* $\exists^*\forall$*-flat sentences is decidable.*

As another example matching the hypothesis of Theorem 3.1 (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 [5].

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, $T_I = (\Sigma_I, \mathscr{C}_I)$ and $T_E = (\Sigma_E, \mathscr{C}_E)$. 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).[2] 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.

Example 3.1. Consider the following sentences:

$$(I)\ \forall i.\, a(i)=i; \qquad (II)\ \forall i_1\forall i_2.(i_1\le i_2\rightarrow a(i_1)\le a(i_2));$$
$$(III)\ \exists i_1\exists i_2.(i_1\le i_2\wedge a(i_1)\not\le a(i_2)); \qquad (IV)\ \forall i_1\forall i_2.\, a(i_1)=a(i_2);$$
$$(V)\ \forall i.(D_2(i)\rightarrow a(i)=0); \qquad (VI)\ \exists i\,\forall j.(a_1(j)<a_2(3i)).$$

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 $i_1\le i_2$). 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

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

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

Theorem 3.2. *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 Appendix A 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. ⊣ □

3.2.1 The decision procedure $\mathsf{SAT}_{\mathsf{MULTI}}$.

The algorithm is non-deterministic: the input formula is satisfiable iff we can guess suitable data $\mathscr{T},\mathscr{B}$ 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)

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

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 $\mathscr{T}=\{M_1,\ldots,M_q\}$ of types.

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

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

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

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 $\mathscr{B}$ a full Boolean satisfying assignment for the atoms of the formula

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

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 $\mathscr{B}$.

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.

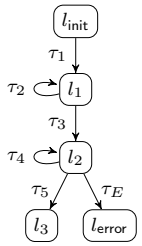
Theorem 3.2 applies to $\mathtt{ARR}^2(\mathbb{P}, \mathbb{P})$ because $\mathbb{P}$ admits quantifier elimination. For this theory, we can determine complexity upper and lower bounds:

Theorem 3.3. $\mathtt{ARR}^2(\mathbb{P}, \mathbb{P})$*-satisfiability of* $\exists^*\forall^*$*-monic-flat sentences is* NEXPTIME*-complete.*

Proof. We use exponentially bounded domino systems for reduction [5, 18], see Appendix A for details. ⊣ □

```
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)

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

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., [24]), 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 $\mathscr{T}$ 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 $\mathscr{T}$ 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 $\mathscr{T}$ representing a (possibly infinite) set of configurations of the program under analysis. A *transition formula* is a formula of $\mathscr{T}$ 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 4.1 (Programs)**.** Given a set of variables $\mathbf{v}$, a *program* is a triple $\mathscr{P}=(L,\Lambda,E)$, 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*.

We indicate by $src,\mathscr{L},trg$ 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), $\mathscr{L}(e)=\tau_j$ (this is called the 'label' of e) and $trg(e)=l_k$ (this is called the 'target' location of e).

Example 4.1. Consider the procedure initEven in Figure 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 Figure 1(b), where $L=\{l_{\text{init}},l_1,l_2,l_3,l_{\text{error}}\}$.

Definition 4.2 (Program paths)**.** A *program path* (in short, *path*) of $\mathscr{P}=(L,\Lambda,E)$ 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 $src(e_1)=l_{\text{init}}$ and $trg(e_{|\rho|})=l_{\text{error}}$. A path ρ is a *feasible path* if $\bigwedge_{j=1}^{|\rho|}\mathscr{L}(e_j)^{(j)}$ is $\mathscr{T}$-satisfiable, where $\mathscr{L}(e_j)^{(j)}$ represents $\tau_{i_j}(\mathbf{v}^{(j-1)},\mathbf{v}^{(j)})$, with $\mathscr{L}(e_j)=\tau_{i_j}$.

The *(unbounded) reachability problem* for a program $\mathscr{P}$ is to detect if $\mathscr{P}$ admits a feasible error path. Proving the safety of $\mathscr{P}$, therefore, means solving the reachability problem for $\mathscr{P}$. This problem, given well known

limiting results, is not decidable for an arbitrary program $\mathscr{P}$. 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 [6, 8, 11] 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 [2].

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 4.3 (flat^0-program)**.** A program $\mathscr{P}$ is a flat^0*-program* if for every path $\rho = e_1, \ldots, e_n$ of $\mathscr{P}$ 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., [4]). 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 $\mathscr{L}$ by denoting $\mathscr{L}^+(e) := \mathscr{L}(e)^+$ if $src(e) = trg(e)$ and $\mathscr{L}^+(e) := \mathscr{L}(e)$ otherwise, where $\mathscr{L}(e)^+$ denotes the acceleration of the transition labeling the edge e.

Theorem 4.1. *Let $\mathscr{F}$ be a class of formulæ decidable for $\mathscr{T}$-satisfiability. The unbounded reachability problem for a flat^0-program $\mathscr{P}$ is decidable if (i) $\mathscr{F}$ is closed under conjunctions and (ii) for each $e \in E$ one can compute $\alpha(\mathbf{v}, \mathbf{v}') \in \mathscr{F}$ such that $\mathscr{T} \models \mathscr{L}^+(e) \hookrightarrow \alpha(\mathbf{v}, \mathbf{v}')$,*

Proof. Let $\rho = e_1, \ldots, e_n$ be an error path of $\mathscr{P}$; when testing its feasibility, according to Definition 4.3, we can limit ourselves to the case in which $e_1, \ldots, e_n$ are all distinct, provided we replace the labels $\mathscr{L}(e_k)^{(k)}$ with $\mathscr{L}^+(e_k)^{(k)}$ in the formula $\bigwedge_{j=1}^n \mathscr{L}(e_j)^{(j)}$ from Definition 4.2.[3] Thus $\mathscr{P}$ is unsafe iff, for some path $e_1, \ldots, e_n$ whose edges are all distinct, the formula

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

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

4.1 A class of array programs with decidable reachability problem

We now produce a class of programs with arrays – we call it $\mathsf{simple}^0_{\mathscr{A}}$-programs– for which requirements of Theorem 4.1 are met. The class of $\mathsf{simple}^0_{\mathscr{A}}$-programs contains non recursive programs implementing searching, copying, comparing, initializing, replacing and testing procedures. As an example, the initEven program reported in Figure 1 is a $\mathsf{simple}^0_{\mathscr{A}}$-program. Formally, a $\mathsf{simple}^0_{\mathscr{A}}$*-program* $\mathscr{P} = (L, \Lambda, E)$ is a flat^0-program

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

such that (i) every $\tau \in \Lambda$ is a formula belonging to one of the decidable classes covered by Corollary 3.1 or Theorem 3.3; (ii) if $e \in E$ is a self-loop, then $\mathscr{L}(e)$ 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 4.4 (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 [2]) gives the template for the accelerated counterpart of a simple_k-assignment.

Lemma 4.1. *Let $\tau(\mathbf{v},\mathbf{v}')$ be a simple_k-assignment. Then $\tau^+(\mathbf{v},\mathbf{v}')$ is $\mathscr{T}$-equivalent to the formula*

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

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

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

Example 4.2. Consider transition τ_2 from the formalization of our running example of Figure 1. The acceleration τ_2^+ of such formula is (we omit identical updates)

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

We can now formally show that the reachability problem for $\mathsf{simple}^0_{\mathscr{A}}$-programs is decidable, by instantiating Theorem 4.1 with the results obtained so far.

Theorem 4.2. *The unbounded reachability problem for $\mathsf{simple}^0_{\mathscr{A}}$-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 3.1 or Theorem 3.3 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 4.1 applies. ⊣ □

4.2 Experimental observations

We evaluated the capabilities of available SMT-Solvers on checking the satisfiability of Flat Array Properties (for more information, see Appendix B) and for that we selected some $\mathsf{simple}^0_{\mathscr{A}}$-programs, both safe and unsafe. Following the procedure identified in the proof of Theorem 4.1 we generated 200 SMT-LIB2-compliant files with Flat Array Properties[4]. The $\mathsf{simple}^0_{\mathscr{A}}$-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 [3] (version 1.2) and Z3 [9] (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[5]. 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.

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

[5] See the discussion in Appendix B for more information on the experiments.

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 [17], although (contrary to [17]) 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 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 [23] and get from our Theorems 3.1,3.2 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 3.1 and Theorem 3.3 are orthogonal to the results from the literature. To this aim, we make a closer comparison with [7, 14]. The two fragments considered in [7, 14] are characterized by rather restrictive syntactic constraints. In [14] 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 $\forall\mathbf{i}.\phi(\mathbf{i}) \rightarrow \psi(\mathbf{i}, \mathbf{a}(\mathbf{i}+\bar{\mathbf{k}}))$ (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 [15]. The Array Property Fragment described in [7] 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 $\forall\mathbf{i}.\phi(\mathbf{i}) \rightarrow \psi(\mathbf{a}(\mathbf{i}))$, where in addition the `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 3.1, while the complexity of the decision procedure proposed in [14] is unknown. On the other side, both $\mathsf{SAT}_{\mathsf{MULTI}}$ and the decision procedure described in [7] run in NExpTime (the decision procedure in [7] 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., [7, 10, 13, 22]), 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, 2].

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. Definability of accelerated relations in a theory of arrays and its applications. In *FroCoS*, pages 23–39, 2013.

[3] 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.

[4] G. Behrmann, J. Bengtsson, A. David, K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL implementation secrets. In *FTRTFT*, pages 3–22, 2002.

[5] E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997.

[6] M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, (91):275–303, 2009.

[7] A.R. Bradley, Z. Manna, and H.B. Sipma. What's decidable about arrays? In *VMCAI*, pages 427–442, 2006.

[8] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *CAV*, volume 1427 of *LNCS*, pages 268–279. Springer, 1998.

[9] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, pages 337–340, 2008.

[10] D.L. Detlefs, G. Nelson, and J.B. Saxe. Simplify: a theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003.

[11] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *FSTTCS*, pages 145–156, 2002.

[12] H. Ganzinger. Shostak light. In *Automated deduction—CADE-18*, volume 2392 of *Lecture Notes in Comput. Sci.*, pages 332–346. Springer, Berlin, 2002.

[13] Y. Ge and L. de Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *CAV*, pages 306–320, 2009.

[14] P. Habermehl, R. Iosif, and T. Vojnar. A logic of singly indexed arrays. In *LPAR*, pages 558–573, 2008.

[15] P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In *FOSSACS*, 2008.

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

[17] C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In *TACAS*, pages 265–281. Springer, 2008.

[18] 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.

[19] R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic. In *CAV'05*, pages 321–334, 2005.

[20] D.C. Oppen. A superexponential upper bound on the complexity of Presburger arithmetic. *J. Comput. System Sci.*, 16(3):323–332, 1978.

[21] S. Ranise and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2006.

[22] 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.

[23] A.L. Semënov. Logical theories of one-place functions on the set of natural numbers. *Izvestiya: Mathematics*, 22:587–618, 1984.

[24] J.R. Shoenfield. *Mathematical logic*. Association for Symbolic Logic, Urbana, IL, 2001. Reprint of the 1973 second printing.

[25] C. Tinelli and C.G. Zarba. Combining nonstably infinite theories. *J. Automat. Reason.*, 34(3):209–238, 2005.

A Proofs

In this Appendix we report detailed proofs of the results which were omitted in the main body of the paper due to page limitation.

A.1 Correctness and completeness of $\mathsf{SAT}_{\mathsf{MULTI}}$ (proof of Theorem 3.2).

The proof of Theorem 3.2[6] is split into two lemmas, showing correctness and completeness of the algorithm $\mathsf{SAT}_{\mathsf{MULTI}}$.

Before stating the two Lemmas and proving them, we introduce useful notation. We use letters $\tilde{b}, \tilde{c}, \ldots$ for elements from the support of a model; notation $\tilde{\mathbf{b}}, \tilde{\mathbf{c}}, \ldots$ is used for tuples (possibly with repetitions) of such elements. For a formula $\varphi(\mathbf{c})$ containing the free variables $\mathbf{c} := c_1, \ldots, c_n$ and for a tuple of elements $\tilde{\mathbf{c}} := \tilde{c}_1, \ldots, \tilde{c}_n$ from the support of a model $\mathscr{M}$, the notation $\mathscr{M} \models \varphi(\tilde{\mathbf{c}})$ means that: (i) we expanded the language with free constants naming $\tilde{c}_1, \ldots, \tilde{c}_n$ (the constant naming $\tilde{c}_i$ is called $\tilde{c}_i$ again for simplicity); (ii) the constant naming $\tilde{c}_i$ is interpreted as $\tilde{c}_i$; (iii) in this expansion of $\mathscr{M}$, we have that $\varphi(\tilde{\mathbf{c}})$ turns out to be true (here, according to our general conventions, $\varphi(\tilde{\mathbf{c}})$ is obtained from $\varphi(\mathbf{c})$ by replacing the variables $\mathbf{c}$ with the names of the $\tilde{\mathbf{c}}$). The above formalism of language expansion is adopted in standard mathematical logic textbooks [24] and is a default machinery in all model-theoretic literature.[7]

Below, we assume that F is the $\exists^*\forall^*$-monic-flat formula

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

the formulæ F_1, F_2, F_3, F_I, F_E are as described in the decision procedure $\mathsf{SAT}_{\mathsf{MULTI}}$ of subsection 3.2.

Lemma A.1 (Completeness of $\mathsf{SAT}_{\mathsf{MULTI}}$)**.** *If F is $\mathtt{ARR}^2(T_I, T_E)$-satisfiable, then it is possible to choose the set $\mathscr{T}$ and the Boolean assignment $\mathscr{B}$ so that F_I is T_I-satisfiable and F_E is T_E-satisfiable.*

Proof. Let $\mathscr{M}$ be a model of F. We have $\mathscr{M} \models \forall \mathbf{i}. \psi(\mathbf{i}, \mathbf{a}(\mathbf{i}), \tilde{\mathbf{c}}, \mathbf{a}(\tilde{\mathbf{c}}))$ for suitable $\tilde{\mathbf{c}}$ from $\mathtt{INDEX}^{\mathscr{M}}$.

A type M is *realized* in $\mathscr{M}$ iff there is some $\tilde{b} \in \mathtt{INDEX}^{\mathscr{M}}$ such that $\mathscr{M} \models \bigwedge_{L \in M} L(\tilde{b}, \tilde{\mathbf{c}})$ (we say in this case that $\tilde{b}$ realizes M).[8] We take $\mathscr{T}$ to be the set of types realized in $\mathscr{M}$; if $\mathscr{T} = \{M_1, \ldots, M_q\}$, we pick a tuple $\tilde{\mathbf{b}} = \tilde{b}_1, \ldots, \tilde{b}_q$ from $\mathtt{INDEX}^{\mathscr{M}}$ realizing them. By assigning precisely this tuple to the variables $\mathbf{b}$ of F_1, we get

$$\begin{aligned} \mathscr{M} \models\ & \forall x. \left(\bigvee_{j=1}^{q} \bigwedge_{L \in M_j} L(x, \tilde{\mathbf{c}}) \right) \wedge \\ & \bigwedge_{j=1}^{q} \bigwedge_{L \in M_j} L(\tilde{b}_j, \tilde{\mathbf{c}}) \wedge \\ & \bigwedge_{\sigma : \mathbf{i} \to \tilde{\mathbf{b}}} \psi(\mathbf{i}\sigma, \mathbf{a}(\mathbf{i}\sigma), \tilde{\mathbf{c}}, \mathbf{a}(\tilde{\mathbf{c}})) \end{aligned}$$

(this formula is F_1 without the outermost existential quantifiers and with $\mathbf{c}, \mathbf{b}$ replaced by - the names of - $\tilde{\mathbf{c}}, \tilde{\mathbf{b}}$).

[6] The reader might have noticed that (by considering the special case of formulæ in which ELEM atoms do not occur), Theorem 3.2 has the following corollary concerning only T_I: "if T_I-satisfiability of the $\exists^*\forall$-sentences is decidable, then T_I-satisfiability of $\exists^*\forall^*$-monic-flat sentences is decidable". There is nothing wrong in this, because by help of (expensive indeed!) Boolean manipulations one can check directly that $\exists^*\forall^*$-monic-flat T_I-sentences are equivalent to disjunctions of $\exists^*\forall$ T_I-sentences. In other words, the notion of being monic becomes interesting only in presence of ELEM atoms.

[7] People preferring a formulation of Tarski semantics in terms of assignments may interpret $\mathscr{M} \models \varphi(\tilde{\mathbf{c}})$ as meaning that $\varphi(\mathbf{c})$ is true in $\mathscr{M}$ under the assignment mapping the $\mathbf{c}$ to the $\tilde{\mathbf{c}}$.

[8] Notice that this type realization notion is relative to the choice of the elements $\tilde{\mathbf{c}}$ assigned to the $\mathbf{c}$.

If we furthermore let the tuple $\tilde{\mathbf{e}}$ be the tuple of the elements denoted by the terms $\mathbf{a}[\tilde{\mathbf{c}}]*\mathbf{a}[\tilde{\mathbf{b}}]$, we get

$$\begin{aligned}\mathscr{M} \models \forall x.\left(\bigvee_{j=1}^{q}\bigwedge_{L\in M_j} L(x,\tilde{\mathbf{c}})\right) \wedge \\ \bigwedge_{j=1}^{q}\bigwedge_{L\in M_j} L(\tilde{b}_j,\tilde{\mathbf{c}}) \wedge \\ \bar{\psi}(\tilde{\mathbf{b}},\tilde{\mathbf{c}},\tilde{\mathbf{e}}) \wedge \\ \bigwedge_{\tilde{d}_m,\tilde{d}_n\in\tilde{\mathbf{b}}*\tilde{\mathbf{c}}}\bigwedge_{l=1}^{s}(\tilde{d}_m=\tilde{d}_n\rightarrow\tilde{e}_{l,m}=\tilde{e}_{l,n})\end{aligned}$$

as well. Now we can get our $\mathscr{B}$ just by collecting the truth-values of the relevant INDEX and ELEM atoms involved in the above formula; by construction, it is clear that F_I and F_E become both true. ⊣ □

Lemma A.2 (Soundness of $\mathsf{SAT}_{\mathsf{MULTI}}$)**.** *If there exist $\mathscr{T} := \{M_1,\ldots,M_q\}$ and $\mathscr{B}$ such that F_I is T_I-satisfiable and F_E is T_E-satisfiable, then F is $\mathtt{ARR}^2(T_I,T_E)$-satisfiable.*

Proof. Suppose we are given a set of types $T=\{M_1,\ldots,M_q\}$ and a Boolean assignment $\mathscr{B}$ such that there exists two models $\mathscr{M}_I,\mathscr{M}_E$ of T_I,T_E, respectively, such that $\mathscr{M}_I\models F_I$ and $\mathscr{M}_E\models F_E$. From the fact that F_I is true in $\mathscr{M}_I$, it follows that there are elements $\tilde{\mathbf{c}},\tilde{\mathbf{b}}$ from $\mathtt{INDEX}^{\mathscr{M}_I}$ such that

$$\mathscr{M}_I\models\forall x.\left(\bigvee_{j=1}^{q}\bigwedge_{L\in M_j} L(x,\tilde{\mathbf{c}})\right)\wedge\bigwedge_{j=1}^{q}\bigwedge_{L\in M_j} L(\tilde{b}_j,\tilde{\mathbf{c}})\wedge\tilde{\psi}_I(\tilde{\mathbf{b}},\tilde{\mathbf{c}})\,. \tag{2}$$

In particular,

$$\mathscr{M}_I\models\bigwedge_{L\in M_j} L(\tilde{b}_j,\tilde{\mathbf{c}})$$

holds for every $M_j\in\mathscr{T}$. Thus, each $M_j\in\mathscr{T}$ is associated with an element $\tilde{b}_j\in\mathtt{INDEX}^{\mathscr{M}_I}$ that realizes it, while

$$\mathscr{M}_I\models\forall x.\left(\bigvee_{j=1}^{q}\bigwedge_{L\in M_j} L(x,\tilde{\mathbf{c}})\right) \tag{3}$$

implies that every $\tilde{z}\in\mathtt{INDEX}^{\mathscr{M}_I}$ realizes some $M_j\in\mathscr{T}$ (see the proof of the previous Lemma for the definition of type realization). We introduce the following notation: given two elements $\tilde{z}_1,\tilde{z}_2\in\mathtt{INDEX}^{\mathscr{M}_I}$, $\tilde{z}_1\approx\tilde{z}_2$ holds iff $\tilde{z}_1$ and $\tilde{z}_2$ realize the same type. Thus, for every $\tilde{z}\in\mathtt{INDEX}^{\mathscr{M}_I}$ there is a (unique because types are mutually inconsistent) $\tilde{b}_j\in\tilde{\mathbf{b}}$ such that $\tilde{z}\approx\tilde{b}_j$. We call this b_j the *representative* of $\tilde{z}$.

Now, since $\mathscr{M}_E\models F_E$, there are elements $\tilde{\mathbf{e}}\in\mathtt{ELEM}^{\mathscr{M}_E}$ such that (once they are used to interpret the constants $\mathbf{e}$) we have

$$\mathscr{M}_E\models\bar{\psi}_E(\tilde{\mathbf{e}})\,. \tag{4}$$

To get a model $\mathscr{M}$ for $\mathtt{ARR}^2(T_I,T_E)$ we need only to interpret the function symbols $\mathbf{a}=a_1,\ldots,a_s$ as functions from $\mathtt{INDEX}^{\mathscr{M}_I}$ into $\mathtt{ELEM}^{\mathscr{M}_E}$. Before doing that, let us observe that, because of our choice of $\mathscr{B}$, we have that $\bar{\psi}_I(\mathbf{b},\mathbf{c})\wedge\bar{\psi}_E(\mathbf{e})\rightarrow F_3$ is a tautology; recalling the definition of F_3 from Step IV of the procedure $\mathsf{SAT}_{\mathsf{MULTI}}$, this means that (independently on how we define the interpretation of the symbols $\mathbf{a}$ not occurring in F_3) we shall have

$$\mathscr{M}\models\bar{\psi}(\tilde{\mathbf{b}},\tilde{\mathbf{c}},\tilde{\mathbf{e}})\wedge\bigwedge_{\tilde{d}_m,\tilde{d}_n\in\tilde{\mathbf{b}}*\tilde{\mathbf{c}}}\bigwedge_{l=1}^{s}(\tilde{d}_m=\tilde{d}_n\rightarrow\tilde{e}_{l,m}=\tilde{e}_{l,n})\,. \tag{5}$$

For every $l=1,\ldots,s$ and for every $\tilde{d}_m\in\tilde{\mathbf{b}}*\tilde{\mathbf{c}}$ we put

$$a_l^{\mathscr{M}}(\tilde{d}_m):=\tilde{e}_{l,m}\,. \tag{6}$$

By (5), this definition gives a partial function. To make it total, for any other $\tilde{z}$ (i.e. $\tilde{z}\notin\tilde{\mathbf{b}}*\tilde{\mathbf{c}}$) pick the representative $\tilde{b}_j$ of $\tilde{z}$, and define

$$a_l^{\mathscr{M}}(\tilde{z}):=a_l^{\mathscr{M}}(\tilde{b}_j)\,. \tag{7}$$

We *claim* that we have, for every $\tilde{z}_1, \tilde{z}_2 \in \mathtt{INDEX}^{\mathscr{M}_I}$

$$\tilde{z}_1 \approx \tilde{z}_2 \quad \Rightarrow \quad a_l^{\mathscr{M}}(\tilde{z}_1) = a_l^{\mathscr{M}}(\tilde{z}_2)\,. \tag{8}$$

To prove the claim, it is sufficient to show that, if $\tilde{b}_j$ is the representative of $\tilde{z}$, then $a_l^{\mathscr{M}}(\tilde{z}) = a_l^{\mathscr{M}}(\tilde{b}_j)$. This is obvious if $\tilde{z} \notin \tilde{\mathbf{b}} * \tilde{\mathbf{c}}$ and if $\tilde{z} \in \tilde{\mathbf{b}} * \tilde{\mathbf{c}}$, we only have to check the case in which $\tilde{z}$ is some $\tilde{c}_l \in \tilde{\mathbf{c}}$. However, since $x = c_l$ is among the atoms contributing to the definition of a type (see STEP I of the procedure $\mathsf{SAT}_{\mathsf{MULTI}}$), it follows that the representative $\tilde{b}_j$ of $\tilde{c}_l$ satisfies the formula $x = \tilde{c}_l$ (because the latter is trivially satisfied by $\tilde{c}_l$) and hence we have that $\tilde{b}_j = \tilde{c}_l$. By (5) and (6), it follows that $a_l^{\mathscr{M}}(\tilde{c}_j) = a_l^{\mathscr{M}}(\tilde{b}_j)$. This ends the proof of the claim.

It remains to prove that $\mathscr{M}$ is a model of F, i.e. that we have

$$\mathscr{M} \models \forall \mathbf{i}.\psi(\mathbf{i}, \mathbf{a}(\mathbf{i}), \tilde{\mathbf{c}}, \mathbf{a}(\tilde{\mathbf{c}}))\,. \tag{9}$$

First notice that, by (6),(5) and by the definition of $\tilde{\psi}(\mathbf{b}, \mathbf{c}, \mathbf{e})$ (see STEP IV of the procedure $\mathsf{SAT}_{\mathsf{MULTI}}$), we have[9]

$$\mathscr{M} \models \bigwedge_{\sigma:\mathbf{i}\to\tilde{\mathbf{b}}} \psi(\mathbf{i}\sigma, \mathbf{a}(\mathbf{i}\sigma), \tilde{\mathbf{c}}, \mathbf{a}(\tilde{\mathbf{c}}))\,. \tag{10}$$

Let τ be the map that associates with every $\tilde{z}$ its representative $\tilde{b}_j \in \tilde{\mathbf{b}}$; it is sufficient to show that for every $\tilde{\mathbf{z}} = \tilde{z}_1, \ldots, \tilde{z}_n$ from $\mathtt{INDEX}^{\mathscr{M}}$,[10] we have, for every atom $A(\mathbf{i}, \mathbf{a}(\mathbf{i}), \mathbf{c}, \mathbf{a}(\mathbf{c}))$ occurring in $\psi(\mathbf{i}, \mathbf{a}(\mathbf{i}), \mathbf{c}, \mathbf{a}(\mathbf{c}))$

$$\mathscr{M} \models A(\tilde{\mathbf{z}}, \mathbf{a}(\tilde{\mathbf{z}}), \tilde{\mathbf{c}}, \mathbf{a}(\tilde{\mathbf{c}})) \longleftrightarrow A(\tilde{\mathbf{z}}\tau, \mathbf{a}(\tilde{\mathbf{z}}\tau), \tilde{\mathbf{c}}, \mathbf{a}(\tilde{\mathbf{c}})) \tag{11}$$

(then (9) follows from (10) and (11) by induction on the number of Boolean connectives, taking for every assignment $\mathbf{i} \mapsto \tilde{\mathbf{z}}$ the conjunct σ corresponding to $\mathbf{i} \mapsto \tilde{\mathbf{z}} \mapsto \tilde{\mathbf{z}}\tau$). In turn, (11) is a special case of the following more general fact: if $\tilde{\mathbf{z}}$ and $\tilde{\mathbf{z}}'$ have length n and we have $\tilde{z}_i \approx \tilde{z}'_i$ (for every $i = 1, \ldots, n$), then

$$\mathscr{M} \models A(\tilde{\mathbf{z}}, \mathbf{a}(\tilde{\mathbf{z}}), \tilde{\mathbf{c}}, \mathbf{a}(\tilde{\mathbf{c}})) \longleftrightarrow A(\tilde{\mathbf{z}}', \mathbf{a}(\tilde{\mathbf{z}}'), \tilde{\mathbf{c}}, \mathbf{a}(\tilde{\mathbf{c}})) \tag{12}$$

for every atom A occurring in ψ. However, (12) holds for ELEM atoms thanks to (8) and for INDEX atoms due to the fact that $\tilde{z}_i, \tilde{z}'_i$ realize the same type and the input formula $F := \exists \mathbf{c} \forall \mathbf{i}.\psi(\mathbf{i}, \mathbf{a}(\mathbf{i}), \mathbf{c}, \mathbf{a}(\mathbf{c}))$ is monic. ⊣ □

A.2 Complexity analysis of $\mathsf{SAT}_{\mathsf{MULTI}}$ (proof of Theorem 3.3).

The proof of Theorem 3.3 can also be split into the two lemmas below, giving lower and upper bounds.

Lemma A.3 (Lower Bound)**.** $\mathtt{ARR}^2(\mathbb{P}, \mathbb{P})$*-satisfiability of* $\exists^*\forall^*$*-monic-flat sentences is* NEXPTIME*-hard.*

Proof. First, we introduce the bounded version of the domino problem used in the reduction. A *domino system* is a triple $\mathscr{D} = (D, H, V)$, where D is a finite set of *domino types* and $H, V \subseteq D \times D$ are the horizontal and vertical matching conditions. Let $\mathscr{D}$ be a domino system and $I = d_0, \ldots, d_{n-1} \in D^*$ an *initial condition*, i.e. a sequence of domino types of length $n > 0$. A mapping $\tau : \{0, \ldots, 2^{n+1} - 1\} \times \{0, \ldots, 2^{n+1} - 1\} \to D$ is a 2^{n+1}*-bounded solution of* $\mathscr{D}$ *respecting the initial condition* I iff, for all $x, y < 2^{n+1}$, the following holds:

- if $\tau(x, y) = d$ and $\tau(x \oplus_{2^{n+1}} 1, y) = d'$, then $(d, d') \in H$;
- if $\tau(x, y) = d$ and $\tau(x, y \oplus_{2^{n+1}} 1) = d'$, then $(d, d') \in V$;
- $\tau(i, 0) = d_i$ for $i < n$;

where $\oplus_{2^{n+1}}$ denotes addition modulo 2^{n+1}.

It is well-known [5, 18] that there is a domino system $\mathscr{D} = (D, H, V)$ such that the following problem is NEXPTIME-hard: given an initial condition $I = d_0, \ldots, d_{n-1} \in D^*$, does $\mathscr{D}$ have a 2^{n+1}-bounded solution respecting I or not?

We show that this problem can be reduced in polynomial time to satisfiability of $\exists^*\forall^*$-flat and simple sentences in $\mathtt{ARR}^2(\mathbb{P}, \mathbb{P})$.

[9] Since types are pairwise inconsistent, the elements $\tilde{\mathbf{b}}$ are in bijective correspondence to the variables $\mathbf{b}$, hence we can freely suppose that the maps σ indexing the big conjunct of (10) have codomain $\tilde{\mathbf{b}}$.

[10] Recall that n is the length of the tuple $\mathbf{i}$. Here $\tilde{\mathbf{z}}$ ranges over all possible tuples of elements that can be assigned to the tuple of variables $\mathbf{i}$.

Let us associate (in an injective way) with every element $d \in D$ a numeral (we call this numeral again d for simplicity);[11] we shall use just one array variable, to be called a.

Let $p_0, \ldots, p_n, q_0, \ldots, q_n$ be distinct pairwise coprime numbers (we underline that they can be computed in time polynomial in n and that polynomially many bits are needed to represent them and, as a consequence, also the divisibility predicates $D_{p_0}, \ldots, D_{p_n}, D_{q_0}, \ldots, D_{q_n}$).[12]

We say a natural number i *represents* the point of coordinates $(x, y) \in [0, 2^{n+1} - 1] \times [0, 2^{n+1} - 1]$ iff for all $k = 0, \ldots, n$, we have that

(i) $D_{p_k}(i)$ holds iff the k-th bit of the binary representation of x is 0;

(ii) $D_{q_k}(i)$ holds iff the k-th bit of the binary representation of y is 0.

Of course, the same (x, y) can be represented in many ways, but at least one representative number exists by the Chinese Reminder Theorem.

We now introduce the following abbreviations:

- $H_E(e, e')$ stands for $\bigvee_{(d,d') \in H}(e = d \wedge e' = d')$;
- $V_E(e, e')$ stands for $\bigvee_{(d,d') \in V}(e = d \wedge e' = d')$;
- $H_I(i, i')$ stands for the conjunction of $\bigwedge_{k=0}^{n}(D_{q_k}(i) \longleftrightarrow D_{q_k}(i'))$ with

$$(\bigwedge_{k=0}^{n}(\neg D_{p_k}(i) \wedge D_{p_k}(i'))) \vee \bigvee_{k=0}^{n}(\bigwedge_{l>k}(D_{p_l}(i) \longleftrightarrow D_{p_l}(i')) \wedge$$
$$\wedge D_{p_k}(i) \wedge \neg D_{p_k}(i') \wedge \bigwedge_{l<k}(\neg D_{p_l}(i) \wedge D_{p_l}(i')))$$

- $V_I(i, i')$ stands for the conjunction of $\bigwedge_{k=0}^{n}(D_{p_k}(i) \longleftrightarrow D_{p_k}(i'))$ with

$$(\bigwedge_{k=0}^{n}(\neg D_{q_k}(i) \wedge D_{q_k}(i'))) \vee \bigvee_{k=0}^{n}(\bigwedge_{l>k}(D_{q_l}(i) \longleftrightarrow D_{q_l}(i')) \wedge$$
$$\wedge D_{q_k}(i) \wedge \neg D_{q_k}(i') \wedge \bigwedge_{l<k}(\neg D_{q_l}(i) \wedge D_{q_l}(i')))$$

Thus, $H_I(i, i')$ holds iff i represents (x, y), i' represents (x', y') and we have $y = y'$ and $x' = x \oplus_{2^{n+1}} 1$. Similarly, $V_I(i, i')$ holds iff i represents (x, y), i' represents (x', y') and we have $x = x'$ and $y' = y \oplus_{2^{n+1}} 1$.

We introduce abbreviations $P_{0,0}(i), \ldots, P_{n-1,0}(i)$ to express the fact that i respectively represents the point of coordinates $(0, 0), \ldots, (n-1, 0)$ by using the formulae

$$P_{0,0}(i) := \bigwedge_{k=0}^{n} D_{q_k(i)} \wedge \bigwedge_{k=0}^{n} D_{p_k(i)}$$
$$P_{1,0}(i) := \bigwedge_{k=0}^{n} D_{q_k(i)} \wedge \neg D_{p_0}(i) \wedge \bigwedge_{k=1}^{n} D_{p_k(i)}$$
$$P_{2,0}(i) := \bigwedge_{k=0}^{n} D_{q_k(i)} \wedge D_{p_0}(i) \wedge \neg D_{p_1}(i) \wedge \bigwedge_{k=2}^{n} D_{p_k(i)}$$
$$\ldots$$

[11] A numeral is a ground term of the kind $1 + \cdots + 1$, i.e. a ground term canonically representing a number. The argument we use works also for weaker theories like $\mathtt{ARR}^2(\mathbb{P}, Eq)$, where Eq is the pure identity theory in a language containing infinitely many constants constrained to be distinct.

[12] One can use the elementary Euclid's argument to show this (much better bounds are known from number theory estimates for the n-th prime number function, see a textbook like E. Bach and J. Shallit, *Algorithmic number theory*, Vol. 1, Foundations of Computing Series, MIT Press, 1996). In fact, define $h(2) := 2$ and $h(n+1) := 1 + \prod_{m<n} h(m)$; it is clear that if $k_1 < k_2$, then $h(k_1)$ and $h(k_2)$ are coprime, because the reminder of the division of $h(k_2)$ by every factor of $h(k_1)$ is 1. Also, we easily get $h(n) \leq n!$ (so $h(n) \leq 2^{n \cdot \log_2 n}$) by induction: indeed, $h(2) \leq 2!$ and $h(n+1) \leq 1 + \prod_{m \leq n} h(m) \leq 1 + n \cdot n! \leq (n+1)!$.

The existence of a tiling is then expressed by the satisfiability of the formula below (the first conjunct takes care of the initialization, whereas the last two about tile matchings):

$$\begin{aligned}&\bigwedge_{k=0}^{n-1} \forall i\,(P_{k,0}(i) \rightarrow a[i] = d_k) \wedge \\ &\wedge\ \forall i_1 \forall i_2\,(H_I(i_1,i_2) \rightarrow H_E(a[i_1],a[i_2])) \wedge \\ &\wedge\ \forall i_1 \forall i_2\,(V_I(i_1,i_2) \rightarrow V_E(a[i_1],a[i_2]))\ .\end{aligned}$$

Notice that the above (polynomially long) formula is in the $\forall^*$-monic-flat fragment, as it can be seen by inspecting the definitions of the macros we used for for $P_{k,0}(i)$, $V_I(i_1,i_2)$, $H_I(i_1,i_2)$. ⊣ □

Lemma A.4 (Uppper Bound)**.** $\mathtt{ARR}^2(\mathbb{P},\mathbb{P})$*-satisfiability of* $\exists^*\forall^*$*-monic-flat sentences is in* NExpTime.

Proof. To show the matching upper bound, it is sufficient to inspect our decision algorithm $\mathsf{SAT}_{\mathsf{MULTI}}$. Clearly, Step I introduces an exponential guess; the formulæ F_1, F_2, F_3, F_I, F_E are all exponentially long (notice that there are exponentially many σ in F_1 and $\mathscr{B}$ can be seen as a set of exponentially many literals). It is well-known that $\mathbb{P}$-satisfiability of quantifier-free formulæ is in NP (see the historical references in [20] for the origins of this result), so that satisfiability of F_E also takes non deterministic exponential time. We only have to discuss $\mathbb{P}$-satisfiability of F_I in more detail. Now, F_I is not quantifier-free and in order to check its satisfiability we need to run quantifier-elimination procedure to the subformula

$$\neg\exists x \neg \left(\bigvee_{j=1}^{q} \bigwedge_{L \in M_j} L(x,\mathbf{c}) \right) \tag{13}$$

The point is that this formula is exponentially long and so we must carefully analyze the cost of the elimination of a single existential quantifier in Presburger arithmetic. We need the following lemma from [20] (Theorem 1, p.327):

Lemma A.5. *Suppose that Cooper's quantifier elimination algorithm, applied to a formula* $\exists x\,\phi$ *(with quantifier-free* ϕ*) yields the quantifier-free formula* ϕ'*. Let* c_0 *(resp.* c_1*) be the number of distinct positive integers appearing as indexes of divisibility predicates or as variable coefficients within* ϕ *(resp.* ϕ'*); let* s_0 *(resp.* s_1*) be the largest absolute values of integer constants (including coefficients) occurring in* ϕ *(resp.* ϕ'*); let* a_0 *(resp.* a_1*) be the number of atoms of* ϕ *(resp.* ϕ'*). Then the following relationship hold:*

$$c_1 \leq c_0^4, \qquad s_1 \leq s_0^{4c_0}, \qquad a_1 \leq a_0^4 s_0^{2c_0}.$$

Now notice that (13) is exponentially long, but integer constants, integer coefficients and indexes of divisibility predicates are the same as in the input formula. Thus, if N bounds the length of the input formula, we get a $2^{O(N^2)}$-bound for the above parameters c_1, s_1, a_1 for the formula ϕ' resulting from the eliminatioon of the universal quantifier from (13). Now (quoting from [20], p.329), "the space required to store [a formula] F_k is bounded by the product of the number of atoms a_k in F_k, the maximum number $m+1$ of constants per atom, the maximum amount of space s_k required to store each constant, and some constant q (included for the various arithmetic and logical operators, etc.)." This means that our ϕ' is exponentially long and, as a consequence, our satisfiability testing for F_I works in NExpTime, as it applies an NP algorithm to an exponential instance. ⊣ □

B Experiments with cvc4 and Z3

In order to test the capabilities of the available state-of-the-art SMT-Solvers on checking the satisfiability of Flat Array Properties, we generated, with the help of a Python prototype implementing the algorithm underlying the proof of Theorem 4.1, 200 SMT-LIB files with Flat Array Properties[13]. These files have been generated by feeding our prototype with some $\mathsf{simple}^0_{\mathscr{A}}$-programs, that is, programs performing simple operations on arrays, like copying one array into another one, initializing an array, finding an element in an array, etc. Table 1 reports the results of our experiments, performed on a computer equipped with a 2.66 GHz CPU and 4GB of RAM running Mac OSX 10.8.4. As stated at the beginning of Section 4, $\mathtt{ARR}^1(\mathbb{P}^+)$ or $\mathtt{ARR}^2(\mathbb{P}^+,\mathbb{P}^+)$ are the reference theories for this experimental part.

[13] More information available at www.inf.usi.ch/phd/alberti/prj/booster.

BENCHMARK	STATUS	CVC4	Z3
init	safe	2/4	4/4
init non constant	safe	2/4	4/4
init partial	safe	2/4	4/4
init partial buggy	unsafe	0/4	4/4
init even	safe	2/4	4/4
init even buggy	unsafe	2/4	3/4
copy	safe	2/4	4/4
copy partial	safe	2/4	4/4
copy odd	safe	4/4	4/4
copy odd buggy	safe	4/4	4/4
check swap	safe	28/32	32/32
check swap buggy	unsafe	28/32	32/32
double swap	safe	60/64	64/64
strcpy	safe	2/4	4/4
strlen	safe	2/2	2/2
strlen buggy	unsafe	1/2	2/2
find	safe	4/4	4/4
find first nonnull	safe	4/4	4/4
merge interleave	safe	0/8	8/8
merge interleave buggy	unsafe	0/8	6/8

Table 1: Success rate for CVC4 (version 1.2) and Z3 (version 4.3.1) on the verification of Flat Array Properties. In this table we do not distinguish timeouts from `unknown` messages or other failures of the tools.

Both tools fail on last program, *merge interleave buggy*. We discuss this program and one of the problematic Flat Array Properties arising from its verification in the following section.

B.1 The "merge interleave buggy" example

Consider the procedure mergeInterleave in Figure 2. For this procedure, $\mathbf{a} = a, b, r$, $\mathbf{c} = i, k$. N is a constant of the background theory. T is the set of formulæ (we omit identical updates):

$$\begin{aligned}
\tau_1 &:= pc = l_{\mathsf{init}} \wedge i' = 0 \wedge pc' = l_1 \\
\tau_2 &:= pc = l_1 \wedge i < N \wedge pc' = l_1 \wedge r' = \lambda j.\mathsf{if}\ (j = i)\ \mathsf{then}\ a(j)\ \mathsf{else}\ r(j) \wedge i' = i + 2 \\
\tau_3 &:= pc = l_1 \wedge i \geq N \wedge pc' = l_2 \wedge i' = 1 \\
\tau_4 &:= pc = l_2 \wedge i < N \wedge pc' = l_2 \wedge r' = \lambda j.\mathsf{if}\ (j = i)\ \mathsf{then}\ b(j)\ \mathsf{else}\ r(j) \wedge i' = i + 2 \\
\tau_5 &:= pc = l_2 \wedge i \geq N \wedge pc' = l_3 \\
\tau_{E_1} &:= pc = l_3 \wedge k \geq 0 \wedge k < N \wedge k \equiv_2 0 \wedge r[k] \neq b[k] \wedge pc' = l_{\mathsf{error}} \\
\tau_{E_2} &:= pc = l_3 \wedge k \geq 0 \wedge k < N \wedge k \equiv_2 1 \wedge r[k] \neq a[k] \wedge pc' = l_{\mathsf{error}}
\end{aligned}$$

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

B.1.1 Acceleration

Transitions τ_2 and τ_4 are simple_k-assignments. Their accelerations are (omitting identical updates):

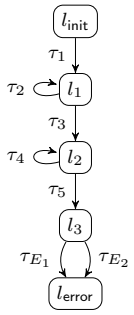
$$\tau_2^+ := \exists y. \begin{pmatrix} y > 0 \wedge pc = l_1 \wedge pc' = l_1 \wedge i' = i + 2y \wedge \\ \forall j.((i \leq j < i + 2y \wedge D_2(j - i)) \rightarrow j < N) \wedge \\ r' = \lambda j.\mathsf{if}\ (i \leq j < 2y + i \wedge D_2(j - i))\ \mathsf{then}\ a(j)\ \mathsf{else}\ r(j) \end{pmatrix}$$

and

$$\tau_4^+ := \exists y. \begin{pmatrix} y > 0 \wedge pc = l_2 \wedge pc' = l_2 \wedge i' = i + 2y \wedge \\ \forall j.((i \leq j < i + 2y \wedge D_2(j - i)) \rightarrow j < N) \wedge \\ r' = \lambda j.\mathsf{if}\ (i \leq j < 2y + i \wedge D_2(j - i))\ \mathsf{then}\ b(j)\ \mathsf{else}\ r(j) \end{pmatrix}$$

```
procedure mergeInterleave ( a[N] , b[N] , r[N] , k ):
l1  for (i = 0; i < N; i = i + 2) r[i] = a[i];
l2  for (i = 1; i < N; i = i + 2) r[i] = b[i];
l3  if(0 ≤ k ∧ k < N ∧ k ≡2 0) assert(r[k] = b[k]);
    if(0 ≤ k ∧ k < N ∧ k ≡2 1) assert(r[k] = a[k]);
```

(a)



(b)

Figure 2: The mergeInterleave procedure (a) and its control-flow graph (b).

B.1.2 Error trace

The procedure mergeInterleave is not safe: a possible execution run showing the unsafety is $\tau_1 \wedge \tau_2^+ \wedge \tau_3 \wedge \tau_4^+ \wedge \tau_5 \wedge \tau_{E_1}$, because r is initialized in the even positions with elements from a, not from b. The error trace is the formula:

$$
\begin{aligned}
&pc_0 = l_{\text{init}} \wedge pc_1 = l_1 \wedge i_1 = 0 \wedge \forall j.r_1(j) = r_0(j) \wedge \\
&\exists y_1. \left(\begin{array}{l} y_1 > 0 \wedge pc_1 = l_1 \wedge pc_2 = l_1 \wedge i_2 = i_1 + 2y_1 \wedge \\ \forall j.((i_1 \le j < i_1 + 2y_1 \wedge D_2(j - i_1)) \rightarrow j < N) \wedge \\ \forall j.(r_2(j) = \text{if } (i_1 \le j < 2y_1 + i_1 \wedge D_2(j - i_1)) \text{ then } a(j) \text{ else } r_1(j)) \end{array} \right) \wedge \\
&pc_2 = l_1 \wedge i_2 \ge N \wedge pc_3 = l_2 \wedge i_3 = 1 \wedge \forall j.(r_3(j) = r_2(j)) \wedge \\
&\exists y_3. \left(\begin{array}{l} y > 0 \wedge pc_3 = l_2 \wedge pc_4 = l_2 \wedge i_4 = i_3 + 2y_3 \wedge \\ \forall j.((i_3 \le j < i_3 + 2y_3 \wedge D_2(j - i_3)) \rightarrow j < N) \wedge \\ r_4 = \lambda j.\text{if } (i_3 \le j < 2y_3 + i_3 \wedge D_2(j - i_3)) \text{ then } b(j) \text{ else } r_3(j) \end{array} \right) \wedge \\
&pc_4 = l_2 \wedge i_4 \ge N \wedge pc_5 = l_3 \wedge i_5 = i_4 \wedge \forall j.(r_5(j) = r_4(j)) \wedge \\
&pc_5 = l_3 \wedge 0 \le k \wedge k < N \wedge D_2(k) \wedge r_5(k) \neq b(k) \wedge pc_6 = l_{\text{error}} \wedge i_6 = i_5 \wedge \forall j.(r_6(j) = r_5(j))
\end{aligned}
$$