Ann Math Artif Intell (2007) 50:231–254
DOI 10.1007/s10472-007-9078-x

Decision procedures for extensions of the theory of arrays

Silvio Ghilardi · Enrica Nicolini · Silvio Ranise · Daniele Zucchelli

Published online: 27 July 2007

Abstract The theory of arrays, introduced by McCarthy in his seminal paper "Towards a mathematical science of computation," is central to Computer Science. Unfortunately, the theory alone is not sufficient for many important verification applications such as program analysis. Motivated by this observation, we study extensions of the theory of arrays whose satisfiability problem (i.e., checking the satisfiability of conjunctions of ground literals) is decidable. In particular, we consider extensions where the indexes of arrays have the algebraic structure of Presburger arithmetic and the theory of arrays is augmented with axioms characterizing additional symbols such as dimension, sortedness, or the domain of definition of arrays. We provide methods for integrating available decision procedures for the theory of arrays and Presburger arithmetic with automatic instantiation strategies which allow us to reduce the satisfiability problem for the extension of the theory of arrays to that of the theories decided by the available procedures. Our approach aims to reuse as much as possible existing techniques so as to ease the implementation of the proposed methods. To this end, we show how to use model-theoretic, rewriting-based

S. Ghilardi · D. Zucchelli (✉)
Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Milan, Italy
e-mail: zucchelli@dsi.unimi.it

S. Ghilardi
e-mail: ghilardi@dsi.unimi.it

E. Nicolini · S. Ranise · D. Zucchelli
LORIA & INRIA-Lorraine, Nancy Cedex, France

E. Nicolini
e-mail: enrica.nicolini@loria.fr

S. Ranise
e-mail: silvio.ranise@loria.fr

theorem proving (i.e., superposition), and techniques developed in the Satisfiability Modulo Theories communities to implement the decision procedures for the various extensions.

Keywords Constraint satisfiability problems · Decision procedures · Combination methods · Instantiation strategies · Theory of arrays with extensionality · Presburger arithmetic

Mathematics Subject Classifications (2000) 68T27 · 03B70 · 68T15 · 03B25

1 Introduction

Since its introduction by McCarthy in [17], the theory of arrays ($\mathcal{A}$) has played a very important role in Computer Science. Hence, it is not surprising that many papers [2, 6, 9, 15, 16, 21, 25, 26] have been devoted to its study in the context of verification and many reasoning techniques, both automatic (see, e.g., [2]) and manual (see, e.g., [21]), have been developed to reason in such a theory.

Unfortunately, as many previous works [6, 15, 16, 26] have already observed, $\mathcal{A}$ alone or even extended with extensional equality between arrays (as in [2, 25]) is not sufficient for many applications of verification. For example, the works in [15, 16, 26] tried to extend the theory to reason about sorted arrays. More recently, Bradley et al [6] have shown the decidability of the satisfiability problem for a restricted class of (possibly quantified) first-order formulae that allows one to express many important properties about arrays.

In this paper, we consider the theory of arrays with extensionality whose indexes have the algebraic structure of Presburger arithmetic ($\mathcal{P}$), and extend it with additional (function or predicate) symbols expressing important features of arrays (e.g., the dimension of an array or an array being sorted). The *main contribution* of the paper is a method to integrate two decision procedures for the constraint satisfiability problem, one for $\mathcal{A}$ and one for $\mathcal{P}$, with instantiation strategies that allow us to reduce the constraint satisfiability problem of the extension of $\mathcal{A} \cup \mathcal{P}$ to the problem decided by the two available procedures.

Our approach to show the correctness of a non-deterministic version of the decision procedure for the constraint satisfiability problem for the theory of arrays with dimension is inspired by model-theoretic methods for combinations of satisfiability problems [13]. The key technical tools in the proof are two: standard models and closures of sets of literals w.r.t. some of the axioms of the theories. The former can be suitably augmented to cope with various extensions of the theory of arrays with dimension which are of interest for program verification (e.g., sorted arrays). The latter allows us to design a uniform three-step methodology for the proofs of the correctness of the various decision procedures: (a) define instantiation strategies to identify (ground) instances of the axioms used to extend the base theory $\mathcal{A} \cup \mathcal{P}$ (e.g., those defining the dimension of an array) and define the notion of closing a set of (ground) literals under such strategies, (b) show that the computation of closed sets of literals always terminates, and (c) prove that the result returned by (a combination

of) the available decision procedures for $\mathcal{A}$ and $\mathcal{P}$ is correct for the extension of the theory of arrays we are dealing with, when considering closed sets of literals. It is important to notice that instantiation strategies give sufficient conditions for computing closed sets of literals. As a consequence, any (possibly more efficient) refinement of a strategy satisfying the closure properties can be used while preserving the correctness of the decision procedure.

While non-deterministic procedures are useful for showing correctness, they are not suited for implementation. We address implementation issues in two ways. First, for certain extensions of the base theory, it is possible to significantly reduce the non-determinism by using rewriting-based methods to build decision procedures (see, e.g., [1, 2]). Since rewriting-based methods are sensitive to the axiomatization of the theories and they are not applicable to all extensions considered in this work, we adapt ideas developed in the Satisfiability Modulo Theories (SMT) community to design practical decision procedures for all extensions of the theory of arrays with dimension. In particular, we exploit the insight in [5] of using a Boolean solver to efficiently implement the guessing phase required by the non-deterministic procedures. This paves the way to re-use the optimizations for efficiency already available in SMT solvers and is the second (and main) way to solve non-determinism.

Related work The work most closely related to ours is [6] by Bradley et al., where a syntactic characterization of a class of full first-order formulae is considered, which turns out to be expressive enough to specify many properties of interest about arrays. The main difference with our work is that we have a semantic approach to extending $\mathcal{A}$ by considering a well-chosen class of first-order structures. This allows us to get a more refined characterization of some properties of arrays, yielding, e.g., the decidability of the constraint satisfiability problem for the extension of $\mathcal{A}$ with the injectivity axiom (see Section 5.1). The decidability of a similar problem is left open by Bradley et al., since their class of models (associated to a set of axioms) is larger than the one considered in this work.

Our instantiation strategy based on Superposition Calculus (see Section 5.2) has a similar spirit of the work in [12], where equational reasoning is integrated in instantiation-based theorem proving. The main difference with [12] is that we solve the state-explosion problem, due to the recombination of formulae caused by the use of standard superposition rules (see, e.g., [20]), by deriving a new termination result for an extension of $\mathcal{A}$ as recommended by the rewriting approach to satisfiability procedures of [2]. This allows us to re-use efficient state-of-the-art theorem provers without the need to implement a new inference system as required by [12].

Plan of the paper Section 2 introduces some formal notions necessary to the development of the results in this paper. Section 3 gives the intuition underlying the models of the theory of arrays with dimension and formally defines this theory. Section 4 describes a non-deterministic decision procedure for the constraint satisfiability problem of such a theory and proves its correctness. Section 5 considers several extensions of the base theory introduced in Section 3.1 and describes how to extend the procedure of Section 4 to decide such extensions. Section 6 discusses two techniques to implement the abstract decision procedures described in Sections 4 and 5. Finally, Section 7 presents some conclusions.

2 Formal preliminaries

We work in *many-sorted first-order logic with equality* and we assume the basic syntactic and semantic concepts as in, e.g., [11].

A *signature* Σ is a non-empty set of sort symbols together with a set of function symbols and a set of predicate symbols (both equipped with suitable lists of sort symbols as arity). The set of predicate symbols contains a symbol $=_S$ for equality for every sort S (we usually omit its subscript). If Σ is a signature, a *simple* expansion of Σ is a signature Σ' obtained from Σ by adding a set $\underline{a} := \{a_1, ..., a_n\}$ of "fresh" constants (each of them again equipped with a sort), i.e., $\Sigma' := \Sigma \cup \underline{a}$, where $\underline{a}$ is such that Σ and $\underline{a}$ are disjoint. Below, we write $\Sigma^{\underline{a}}$ as the simple expansion of Σ with a set $\underline{a}$ of fresh constant symbols.

First-order terms and formulae over a signature Σ are defined in the usual way, i.e., they must respect the arities of function and predicate symbols and the variables occurring in them must also be equipped with sorts (well-sortedness). A Σ-*atom* is a predicate symbol applied to (well-sorted) terms. A Σ-*literal* is a Σ-atom or its negation. A *ground* literal is a literal not containing variables. A *constraint* is a finite conjunction $\ell_1 \wedge \cdots \wedge \ell_n$ of literals, which can also be seen as a finite set $\{\ell_1, \ldots, \ell_n\}$. A Σ-*sentence* is a first-order formula over Σ without free variables.

A Σ-*structure* $\mathcal{M}$ consists of non-empty and pairwise disjoint domains $S^{\mathcal{M}}$ for every sort S, and interprets each function symbol f and predicate symbol P as functions $f^{\mathcal{M}}$ and relations $P^{\mathcal{M}}$, respectively, according to their arities. If t is a ground term, we also use $t^{\mathcal{M}}$ for the element denoted by t in the structure $\mathcal{M}$. If $\Sigma_0 \subseteq \Sigma$ is a sub-signature of Σ and if $\mathcal{M}$ is a Σ-structure, the Σ_0-*reduct* of $\mathcal{M}$ is the Σ_0-structure $\mathcal{M}_{|\Sigma_0}$ obtained from $\mathcal{M}$ by forgetting the interpretation of sorts, function and predicate symbols from $\Sigma \setminus \Sigma_0$. Validity of a formula φ in a Σ-structure $\mathcal{M}$ (in symbols, $\mathcal{M} \models \varphi$), satisfiability, and logical consequence are defined in the usual way. A Σ-*theory* T is a (possibly infinite) set of Σ-sentences. The Σ-structure $\mathcal{M}$ is a *model* of the Σ-theory T if and only if all the sentences of T are valid in $\mathcal{M}$. Let T be a theory; we refer to the signature of T as Σ_T. If there exists a set $Ax(T)$ of sentences in T such that every formula φ of T is a logical consequence of $Ax(T)$, then we say that $Ax(T)$ is a set of axioms of T. A theory T is *complete* if and only if, given a sentence φ, we have that either φ or $\neg\varphi$ is a logical consequence of T.

In this paper, we are concerned with the *(constraint) satisfiability problem* for a theory T, also called the T-satisfiability problem, which is the problem of deciding whether a Σ_T-constraint is satisfiable in a model of T. Notice that a constraint may contain variables: since these variables may be equivalently replaced by free constants, we can reformulate the constraint satisfiability problem as the problem of deciding whether a finite conjunction of ground literals in a simply expanded signature $\Sigma_T^{\underline{a}}$ is true in a $\Sigma_T^{\underline{a}}$-structure whose Σ_T-reduct is a model of T; from now on we shall adopt the latter formulation. We say that a Σ_T-constraint is T-*satisfiable* if and only if there exists a model of T satisfying it. Two Σ_T-constraints φ and ψ are T-equisatisfiable whenever the following condition holds: there exists a structure $\mathcal{M}_1$ such that $\mathcal{M}_1 \models T \wedge \varphi$ if and only if there exists a structure $\mathcal{M}_2$ such that $\mathcal{M}_2 \models T \wedge \psi$.

Without loss of generality, when considering a set L of ground literals to be checked for satisfiability, we may assume that each literal ℓ in L is *flat*, i.e., ℓ is required to be either of the form $a = f(a_1, \ldots, a_n)$, $P(a_1, \ldots, a_n)$, or $\neg P(a_1, \ldots, a_n)$,

where $a, a_1, \ldots, a_n$ are (sort-preserving) constants, f is a function symbol, and P is a predicate symbol (possibly also equality).

3 Arrays with dimension

An array is a data structure that consists of a group of elements having a single name. Elements in the array are usually numbered and individual elements are accessed by their index (i.e., numeric position). We consider two main types of arrays which are natively supported by imperative languages (such as C): *fixed-size* and *dynamically-allocated* arrays. A fixed-size array occupies a contiguous area of storage that never changes during run-time and whose fixed dimension is known at compile-time. In contrast, the size of the memory reserved to dynamically-allocated arrays can be unknown at compile-time and may change at runtime, even though this may be an expensive operation involving the copy of the entire content of an array (consider, e.g., the C's function `realloc` applied to a `malloc`'ed array). To be precise, there exists a third type of arrays called *dynamic*, which are supported by interpreted (e.g., Perl) and object-oriented programming languages (e.g., the C++'s `std::vector` or the `ArrayList` classes of Java API and the .NET Framework) in which memory handling is usually hidden. A detailed discussion of such a data structure is beyond the scope of this paper. Here, it is sufficient to observe that dynamic arrays can be efficiently implemented by imposing an appropriate memory allocation policy on dynamically-allocated arrays (see, e.g., [7]). For all types of arrays, their elements have usually the same type.

After the declaration, the content of an array is in general not initialized, both in the case of fixed-size or dynamically-allocated arrays (recall, e.g., the difference between the C's functions `malloc` and `calloc`). To formalize this, we introduce a distinguished element $\perp$ (for undefined), which is distinct from every other element in arrays, and assume that any array contains $\perp$ at every position except one, after creation. This distinguished position is the capacity of an array a (minus 1, since 0 is used to identify the first element of a), i.e., how many elements a will be able to store. Under this assumption, the situation where a predefined element is used to fill the array after declaration can be simulated by using an appropriate sequence of assignments. In our formal model, we abstract from memory and efficiency issues and assume the capability of storing an element e at an arbitrary index i of an array a, by allocating (only) the necessary extra space when i is bigger than the actual size of a; the resulting array is denoted with $\mathsf{store}(a, i, e)$. In this way, we can formalize the capacity of an array as the function dim returning the smallest index, after which no more elements of the array exist. For simplicity, we will talk about the 'dimension' of an array instead of its capacity.

To summarize, we have chosen to formalize dynamically-allocated arrays while abstracting away any considerations about memory handling. The reader may wonder why we have taken such a decision. The answer is twofold. First, dynamically-allocated arrays are at the core of many algorithms and abstract data types (such as heaps, queues, and hash tables). So, the availability of a procedure (see Section 4) to reason about such a type of arrays would greatly help the task of verifying many programs. The second reason is that dynamically-allocated arrays more accurately model heaps, i.e., the areas of memory where pointer-based data structures are

dynamically allocated. For example, as observed in [18], the absence of aliasing in linked lists can be specified by using an axiom for injectivity of the function modelling the heap. It is possible to extend dynamic arrays with a recognizer for "injective arrays," where $\perp$ models the null-pointer, and obtain a decision procedure also for this theory (see Section 5.1). As another example, consider Separation Logic as introduced by Reynolds in [21]. The key feature of this logic is its capability to support "local reasoning" by formalizing heaps as partial function from addresses to values and introducing new logical connectives, such as the separating conjunction $P \star Q$ that asserts that P and Q hold for *disjoint* portions of a certain heap. Indeed, the partial function modelling heaps can be turned into total functions by using the standard trick of returning an undefined value whenever they are undefined. In this sense, heaps can naturally be seen as dynamic arrays, which can be extended with a "domain" function, returning the set of non-$\perp$ elements. We will see that also this extension of the theory of arrays with dimension is decidable (see Section 5.2); this can also be seen as a first step in the direction of providing automatic support for Separation logic by decision procedures developed in first-order logic.

We are now in the position to discuss the simple mathematical model underlying dynamic arrays. Given a set A, by $Arr(A)$ we denote the set of finite arrays with natural numbers as indexes and whose elements are from A. An element of $Arr(A)$ is a sequence $a : \mathbb{N} \longrightarrow A \cup \{\perp\}$ eventually equal to $\perp$ (here $\perp$ is an element not in A denoting an "undefined" value). In this way, for every array $a \in Arr(A)$ there is a smallest index $n \geq 0$, called the *dimension* of a, such that the value of a at index j is equal to $\perp$ for $j \geq n$. We do not require any value of a at $k < n$ to be distinct from $\perp$: this is also the reason to use the word 'dimension' rather than 'length'. There is just one array whose dimension is zero which we indicate by ε and call it the *empty array*. Since many applications of verification require arithmetic expressions on indexes of arrays, we introduce Presburger arithmetic $\mathcal{P}$ over indexes: any other decidable fragment of Arithmetic would be a good alternative. Thus the relevant operations on our arrays include *addition over indexes, read, write, and dimension*. Below, we will consider a theory, denoted by $\mathcal{ADP}$, capable of formally expressing the properties described above.

3.1 Arrays with dimension as a combined theory

Formally, the theory $\mathcal{ADP}$ can be seen as a combination of two well-known theories: $\mathcal{P}$ and the theory $\mathcal{A}_e$ of arrays with extensionality (see, e.g., [2]), extended with a function for the dimension which takes an array and returns a natural number. Because of the function for dimension, the combination is *non-disjoint* and cannot be handled by classical combination schemas such as Nelson–Oppen [19]. Nevertheless, following [13], it is convenient to see $\mathcal{ADP}$ as a combination of $\mathcal{P}$ with a theory of array with dimension $\mathcal{A}_{\text{dim}}$: $\mathcal{A}_{\text{dim}}$ extends $\mathcal{A}_e$ (both in the signature and in the axioms), but is contained in $\mathcal{ADP}$, because in $\mathcal{A}_{\text{dim}}$ indexes are only endowed with a discrete linear poset structure. In this way, we have that $\mathcal{ADP} = \mathcal{A}_{\text{dim}} \cup \mathcal{P}$ and the theories $\mathcal{A}_{\text{dim}}$ and $\mathcal{P}$ share the well-known complete theory $\mathcal{T}_0$ of natural numbers endowed with zero and successor (see e.g., [10]): this theory admits quantifier elimination, so that the $\mathcal{T}_0$-compatibility hypothesis of [13] needed for the non-disjoint Nelson–Oppen combination is satisfied. Unfortunately, the combination result in [13] cannot be applied to $\mathcal{ADP}$ for mainly two reasons. First, $\mathcal{T}_0$ is not locally finite (see,

e.g., [13] for details). Secondly, $\mathcal{A}_{\text{dim}}$ is a proper extension of the theory $\mathcal{A}_e$, hence the decision procedures for the $\mathcal{A}_e$-satisfiability problem (such as, e.g., the one in [2]) must be extended. In the rest of the paper, we will show that it is sufficient to use decision procedures for the $\mathcal{P}$- and $\mathcal{A}_e$-satisfiability problem to solve the $\mathcal{ADP}$-satisfiability problem, provided that a suitable pre-processing of the input set of literals is performed.

We now introduce the basic theories of interests for this paper.

$\boxed{\mathcal{T}_0}$ has just one sort symbol INDEX, the following function and predicate symbols: $0 :$ INDEX, $\mathsf{s} :$ INDEX $\rightarrow$ INDEX, and $<:$ INDEX $\times$ INDEX. It is axiomatized by the following formulae:[1]

$$y \neq 0 \rightarrow \exists z(y = \mathsf{s}(z)) \tag{1}$$

$$x < \mathsf{s}(y) \leftrightarrow (x < y \vee x = y) \tag{2}$$

$$\neg(x < 0) \tag{3}$$

$$x < y \vee x = y \vee y < x \tag{4}$$

$$x < y \rightarrow \neg(y < x) \tag{5}$$

$$x < y \rightarrow (y < z \rightarrow x < z) \tag{6}$$

where x, y and z are variables of sort INDEX. This theory admits elimination of quantifiers and it is complete, see [10] for details.

$\boxed{\mathcal{P}}$ is the well-known Presburger arithmetic (see, e.g., [10]) over indexes. The signature is that of $\mathcal{T}_0$ extended with the function symbol for addition $+ :$ INDEX $\times$ INDEX $\rightarrow$ INDEX, written infix. Since $\mathcal{P}$ is not finitely axiomatizable (see, again [10]), we assume as axioms all the sentences valid in the standard model of natural numbers. Notice that $\mathcal{T}_0 \subset \mathcal{P}$.

$\boxed{\mathcal{A}}$ is the theory of arrays (see, e.g., [2]) which has the following signature:

- sort symbols: INDEX, ELEM, ARRAY and
- function symbols: select : ARRAY $\times$ INDEX $\rightarrow$ ELEM and store : ARRAY $\times$ INDEX $\times$ ELEM $\rightarrow$ ARRAY

and it is axiomatized by the following formulae:

$$\mathsf{select}(\mathsf{store}(a, i, e), i) = e \tag{7}$$

$$i \neq j \rightarrow \mathsf{select}(\mathsf{store}(a, i, e), j) = \mathsf{select}(a, j) \tag{8}$$

$\boxed{\mathcal{A}_e}$ is the theory of arrays with extensionality (see, e.g., [2]) which has the same signature of $\mathcal{A}$ and it is axiomatized by (7, 8), and the axiom of extensionality:

$$\forall i(\mathsf{select}(a, i) = \mathsf{select}(b, i)) \rightarrow a = b \tag{9}$$

The converse implication is an obvious consequence of the congruence of equality; hence, there is no need to explicitly take it into account since we

[1] Here and in the following, we omit the outermost universal quantification for the sake of readability.

work in (many-sorted) first-order logic with equality (cf. first sentence of Section 2). Notice also that $\mathcal{A} \subset \mathcal{A}_e$.

$\mathcal{A}_{\text{dim}}$ is the simple theory of arrays with dimension whose signature is the union of the signatures of $\mathcal{T}_0$ and $\mathcal{A}_e$ extended with the following three symbols: $\perp$: ELEM, ε : ARRAY, and dim : ARRAY $\rightarrow$ INDEX. It is axiomatized by the axioms in $\mathcal{T}_0$, those in $\mathcal{A}_e$, and the following formulae:

$$\mathsf{dim}(a) \leq i \rightarrow \mathsf{select}(a, i) = \perp \tag{10}$$

$$\mathsf{dim}(a) = \mathsf{s}(i) \rightarrow \mathsf{select}(a, i) \neq \perp \tag{11}$$

$$\mathsf{dim}(\varepsilon) = 0 \tag{12}$$

Notice that $\mathcal{T}_0 \subset \mathcal{A}_{\text{dim}}$ and $\mathcal{A}_e \subset \mathcal{A}_{\text{dim}}$.

$\mathcal{ADP}$ is the theory of arrays with dimension whose signature is the union of the signatures of $\mathcal{A}_{\text{dim}}$ and $\mathcal{P}$ and is axiomatized by the axioms in $\mathcal{A}_{\text{dim}}$ and all valid sentences in $\mathcal{P}$.

The theories $\mathcal{T}_0$ and $\mathcal{P}$ are decidable (see [10]); moreover, the constraint satisfiability problem for the theories $\mathcal{A}$ and $\mathcal{A}_e$ is decidable (see [2]). These are important observations for the results of this paper, since the decision procedure for $\mathcal{ADP}$-satisfiability will assume the availability of two decision procedures for the constraint satisfiability problems of $\mathcal{P}$ and $\mathcal{A}$. The theories $\mathcal{A}_e$, $\mathcal{A}_{\text{dim}}$, and $\mathcal{ADP}$ admit a particular subclass of models, which we call the *standard* ones and are exactly those introduced above in order to motivate the definition of $\mathcal{ADP}$. Formally, a standard model is the model induced by a pair (A, κ), where A is a set of elements and κ is a distinguished element of A as explained in the following definition.

Definition 3.1 Let A be a set and κ be an element of A. The *standard model of $\mathcal{ADP}$ induced by the pair (A, κ)* is the $\Sigma_{\mathcal{ADP}}$-structure $\mathcal{M}$ such that

(1) The sort INDEX is interpreted in $\mathcal{M}$ as $\mathbb{N}$ and the symbols $0, <, s, +$ have their natural meaning;
(2) The sort ELEM is interpreted in $\mathcal{M}$ as A and the constant $\perp$ is interpreted as κ;
(3) The sort ARRAY is interpreted in $\mathcal{M}$ as the set of functions $a : \mathbb{N} \longrightarrow A$ such that there is some $n_a \in \mathbb{N}$ for which we have $a(m) = \kappa$ whenever $m \geq n_a$; moreover, the constant ε is interpreted as the constant function with value κ;
(4) $dim^{\mathcal{M}}(a)$ is the smallest $n \in \mathbb{N}$ such that $a(m) = \kappa$ holds for all $m \geq n$;
(5) We have $\mathsf{select}^{\mathcal{M}}(a, i) := a(i)$ and

$$\mathsf{store}^{\mathcal{M}}(a, i, e)(n) := \begin{cases} a(n) & \text{if } n \neq i, \\ e & \text{otherwise.} \end{cases}$$

The standard models of $\mathcal{A}_e$ and $\mathcal{A}_{\text{dim}}$ can be defined in a similar way by taking the $\Sigma_{\mathcal{A}_e}$- and $\Sigma_{\mathcal{A}_{\text{dim}}}$-reduct (respectively) of $\mathcal{ADP}$-standard models; notice that the dimension of the empty array is 0 and the dimension of a non-empty array is the successor of the index of the last element different from $\perp$. Of course, when investigating constraint satisfiability we are mainly interested in satisfiability of constraints in standard models and we shall in fact prove that a constraint is satisfiable in a model of $\mathcal{ADP}$ if and only if it is satisfiable in a standard model (see Lemma 4.3, below).

4 A decision procedure for arrays with dimension

In the rest of the paper, we assume the availability of two decision procedures solving the $\mathcal{A}$- and $\mathcal{P}$-satisfiability problems; we will see how to reduce to these latter the $\mathcal{ADP}$-satisfiability problem. In order to introduce the reader into the details of the procedure, we consider an example which illustrates some key ideas.

Example 4.1 Consider the problem of checking the $\mathcal{ADP}$-satisfiability of

$$\begin{aligned} \mathsf{dim}(a) = n \ \wedge \ \mathsf{dim}(b) = m \ \wedge \ b = \mathsf{store}(a, n, e) \ \wedge \\ e \neq \bot \ \wedge \ m > 0 \ \wedge \ n = m + 1 \end{aligned} \tag{13}$$

where a, b, m, n, e are free constants of appropriate sorts. To detect the unsatisfiability of (13), it is crucial to derive that $m < n$ in Presburger arithmetic. In fact, we can detect the $\mathcal{A}_e$-unsatisfiability of $b = \mathsf{store}(a, n, e) \wedge e \neq \bot$ in (13) and $\mathsf{select}(b, n) = \bot$, which is a logical consequence of (10) and $\mathsf{dim}(b) = m < n = \mathsf{dim}(a)$. If we were not able to derive facts in Presburger arithmetic, we would have failed to show the $\mathcal{ADP}$-unsatisfiability of (13).

The capability of deriving all facts entailed by a constraint can be problematic, since we only assume the availability of a decision procedure to solve the $\mathcal{P}$-satisfiability problem without further capabilities. To overcome this difficulty, we will transform the problem of checking a logical consequence into a satisfiability problem, i.e., if φ and ψ are two constraints in $\mathcal{P}$, then in order to check $\mathcal{P} \cup \{\varphi\} \models \psi$, we will check the $\mathcal{P}$-unsatisfiability of $\varphi \wedge \neg\psi$. Indeed, it will be necessary to *guess* the entailed constraint ψ. This is a standard technique in the field of combining decision procedures (see, e.g., [13]), which allows us to abstractly describe our decision procedure and more easily prove its correctness.

4.1 The architecture

The overall schema of the decision procedure for the $\mathcal{ADP}$-satisfiability problem is depicted in Fig. 1. The module Flatten pre-processes the literals in the input constraint so as to make them flat and easily recognizable as belonging to one

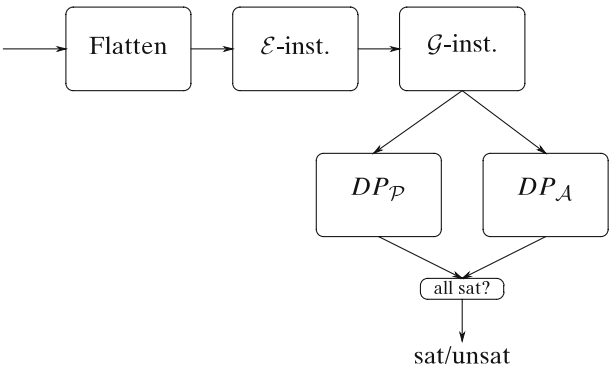

Fig. 1 The architecture of the decision procedure for $\mathcal{ADP}$

theory among those used to define $\mathcal{ADP}$ (see Section 3.1), i.e., $\mathcal{T}_0$, $\mathcal{P}$, $\mathcal{A}_e$, or $\mathcal{A}_{\text{dim}}$. The module $\mathcal{E}$-instantiation produces suitable instances of the extensionality axiom, i.e., (9), so that a simpler decision procedure for the $\mathcal{A}$-satisfiability problem (with respect to one for $\mathcal{A}_e$) is assumed available. The module $\mathcal{G}$-instantiation is non-deterministic and guesses sufficiently many facts which are potentially entailed by the constraints in $\mathcal{P}$. The modules $DP_{\mathcal{P}}$ and $DP_{\mathcal{A}}$ implement the decision procedures for Presburger arithmetic and for the constraint satisfiability problem for the theory of arrays (without extensionality). The module 'all sat?' returns *satisfiable* if both decision procedures for $\mathcal{P}$ and $\mathcal{A}$ returned *satisfiable*; otherwise returns *unsatisfiable*. Now, we are ready to describe the internal workings of each module in detail.

4.1.1 Flattening

It is well-known (see, e.g., [2]) that it is possible to transform a constraint φ into an equisatisfiable constraint φ' containing only flat literals in linear time by introducing sufficiently many fresh constant symbols to name subterms. In our case, we assume that the module Flatten in Fig. 1 transforms (in linear time) a set of arbitrary literals over the signature $\Sigma^{\underline{a}}_{\mathcal{ADP}}$ into an equisatisfiable set of flat literals on the signature $\Sigma^{\underline{c}}_{\mathcal{ADP}}$, for some set $\underline{c} \supseteq \underline{a}$ of constants (the constants in $\underline{c} \setminus \underline{a}$ are said to be fresh). Notice that a flattened set of literals L over a simple expansion of $\Sigma_{\mathcal{ADP}}$ can be represented as a set-theoretic union $L_{\mathcal{A}_{\text{dim}}} \cup L_{\mathcal{P}}$, where $L_{\mathcal{A}_{\text{dim}}}$ collects all the literals from L over a simple expansion of $\Sigma_{\mathcal{A}_{\text{dim}}}$ and $L_{\mathcal{P}}$ collects all the literals from L over a simple expansion of $\Sigma_{\mathcal{P}}$ (thus $L_{\mathcal{A}_{\text{dim}}} \cap L_{\mathcal{P}}$ contains precisely the literals from L over a simple expansion of $\Sigma_{\mathcal{T}_0}$).

4.1.2 $\mathcal{E}$-Instantiation closure

The module $\mathcal{E}$-instantiation finds enough instances of the axiom (9) for extensionality of arrays so that it can be eliminated without compromising the correctness of the decision procedure for $\mathcal{ADP}$.

Definition 4.1 ($\mathcal{E}$-instantiation closed set of literals) A set L of ground flat literals is $\mathcal{E}$-instantiation closed if and only if the following condition is satisfied:

1. If $a \neq b \in L$, with $a, b : \text{ARRAY}$, then $\{\mathsf{select}(a, i) = e_1, \mathsf{select}(b, i) = e_2, e_1 \neq e_2\} \subseteq L$ for some constants $i : \text{INDEX}, e_1, e_2 : \text{ELEM}$;

It is not difficult to see that, given a set of ground flat literals L, there exists an $\mathcal{ADP}$-equisatisfiable set $L^{\mathcal{E}} \supseteq L$ that contains the Skolemization of some logical consequences of $\mathcal{A}_e \cup L$ and is $\mathcal{E}$-instantiation closed.

Lemma 4.1 *There exists a linear time algorithm which takes a set L of flat literals over the signature $\Sigma^{\underline{a}}_{\mathcal{ADP}}$ and returns an $\mathcal{E}$-instantiation closed set $L^{\mathcal{E}}$ of flat literals over the signature $\Sigma^{\underline{c}}_{\mathcal{ADP}}$ such that* (1) *$L \subseteq L^{\mathcal{E}}$,* (2) *$L$ and $L^{\mathcal{E}}$ are $\mathcal{ADP}$-equisatisfiable, and* (3) *$\underline{a} \subseteq \underline{c}$.*

The signature $\Sigma^{\underline{c}}_{\mathcal{ADP}}$ of $L^{\mathcal{E}}$ is a proper simple expansion of the signature $\Sigma^{\underline{a}}_{\mathcal{ADP}}$ of L, because Skolem constants are fresh. It is straightforward to see that, if L contains n literals, at most $3n$ new literals are sufficient to obtain an $\mathcal{E}$-instantiation

closed set of literals containing L. Under the assumption that producing a new literal takes constant time, there exists a linear time algorithm to compute $\mathcal{E}$-instantiation closed sets.

4.1.3 $\mathcal{G}$-Instantiation closure

The module $\mathcal{G}$-instantiation is non-deterministic and it is responsible to produce suitable instances of the axioms about the dimension of arrays, i.e., (10) and (11), and to guess enough facts of $\mathcal{P}$ entailed by the input constraint so as to guarantee the correctness of the overall decision procedure for $\mathcal{ADP}$-satisfiability.

Definition 4.2 ($\mathcal{G}$-instantiation closed set of literals) A set L of ground flat literals is $\mathcal{G}$-instantiation closed if and only if the following conditions are satisfied:

1. If ε occurs in L, then $\mathsf{dim}(\varepsilon) = 0 \in L$;
2. If $\mathsf{dim}(a) = i \in L$, with $a : \textsc{array}$ and $i : \textsc{index}$, then $\{i = 0\} \subseteq L$ or $\{e \neq \bot, \mathsf{select}(a, j) = e, s(j) = i\} \subseteq L$ for some constants $j : \textsc{index}$ and $e : \textsc{elem}$;
3. If i, j occur in L, with $i, j : \textsc{index}$, then $i = j \in L$ or $i \neq j \in L$;
4. If i, j occur in L, with $i, j : \textsc{index}$ and $i \neq j \in L$, then $i < j \in L$ or $j < i \in L$;
5. If $\{\mathsf{dim}(a) = i, i \leq j\} \subseteq L$, with $a : \textsc{array}$ and $i, j : \textsc{index}$, then $\{\mathsf{select}(a, j) = \bot\} \subseteq L$ (here $i \leq j$ stands for $i < j$ or $i = j$).

Given a set of literals, it is always possible to compute an equisatisfiable $\mathcal{G}$-instantiation closed set in (non-deterministic) polynomial time.

Lemma 4.2 *There exists a non-deterministic polynomial time algorithm which takes as input a set L of ground flat literals over a signature $\Sigma^{\underline{a}}_{\mathcal{ADP}}$ and returns a $\mathcal{G}$-instantiation closed set $L^{\mathcal{G}}$ of flat literals over the signature $\Sigma^{\underline{c}}_{\mathcal{ADP}}$ such that* (1) $L \subseteq L^{\mathcal{G}}$, (2) *$L$ and $L^{\mathcal{G}}$ are $\mathcal{ADP}$-equisatisfiable, and* (3) $\underline{a} \subseteq \underline{c}$.

Proof Let m be the number of literals in L of the form $\mathsf{dim}(a_k) = d_{a_k}$ where d_{a_k} is a constant of sort $\textsc{index}$. Let us consider a set $\underline{b} = \{j_1, \ldots, j_m, e_1, \ldots, e_m\}$ of fresh constants, where $j_k : \textsc{index}$, $e_k : \textsc{elem}$, and $k \in \{1, \ldots, m\}$. A $\mathcal{G}$-instantiation $L^{\mathcal{G}}$ of L can be computed by sequentially executing the following three steps:

1. For each pair i, j of constants of sort $\textsc{index}$ in $\underline{a} \cup \underline{b} \cup \{0\}$, exactly one of the atoms $i = j$ and $i \neq j$ is added to $L^{\mathcal{G}}$, and in the latter case either $i < j$ or $j < i$ is also added;
2. For each literal $\mathsf{dim}(a_k) = d_{a_k} \in L^{\mathcal{G}}$, then:
   a. If $0 = d_{a_k} \in L^{\mathcal{G}}$ or $0 \equiv d_{a_k}$, then add $\{j_k = 0, e_k = \bot\}$ to $L^{\mathcal{G}}$;
   b. If $0 < d_{a_k} \in L^{\mathcal{G}}$, then add $\{\mathsf{s}(j_k) = d_{a_k}, \mathsf{select}(a_k, j_k) = e_k, e_k \neq \bot\}$ to $L^{\mathcal{G}}$.
3. If $\{\mathsf{dim}(a) = i, i \leq j\} \subseteq L^{\mathcal{G}}$, then add $\{\mathsf{select}(a, j) = \bot\}$ to $L^{\mathcal{G}}$.

There are two important observations. First, each new constant $j_k : \textsc{index}$ ($k \in \{1, \ldots, m\}$) denotes the predecessor of the dimension of a_k, when the latter is guessed to be different from 0 (if the dimension of a_k is guessed to be 0, then j_k is set to zero). Second, each new constant $e_k : \textsc{elem}$ ($k \in \{1, \ldots, m\}$) denotes the result of reading the content of array a_k at position j_k.

These two observations together with the fact that the process described above to build $L^{\mathcal{G}}$ closely follows Definition 4.2 should make it clear that L is $\mathcal{ADP}$-satisfiable if and only if there exist a set $L^{\mathcal{G}}$ which is $\mathcal{G}$-instantiation closed and $\mathcal{ADP}$-satisfiable. The non-deterministic polynomial time result is obtained by a straightforward inspection of the process described above. □

It is easy to check that one obtains a set of both $\mathcal{E}$- and $\mathcal{G}$-instantiation closed set of literals by invoking first the $\mathcal{E}$- and then the $\mathcal{G}$-instantiation module.

4.2 The algorithm

The algorithm in Fig. 2 gives a (non-deterministic) decision procedure to solve the $\mathcal{ADP}$-satisfiability problem. Without loss of generality (see Section 4.1.1), we assume that L contains only flat literals.

The function DP_T, for $T \in \{\mathcal{ADP}, \mathcal{A}, \mathcal{P}\}$, denotes a decision procedure to solve the T-satisfiability problem, i.e., DP_T takes a set L of literals over (a simple expansion of) the signature Σ_T and returns *satisfiable* when L is T-satisfiable; *unsatisfiable*, otherwise. If L is a set of flat literals, then

$$L_T := \{\ell | \ell \in L \text{ is a } \Sigma_T^a\text{-literal}\},$$

where $T \in \{\mathcal{A}, \mathcal{P}\}$. So, for example, $L_{\mathcal{P}}^{\mathcal{G}}$ is the subset of the literals in $L^{\mathcal{G}}$ over a simple expansion of the signature $\Sigma_{\mathcal{P}}$ (for the sake of readability, when it is clear from the context, the term "simple expansion" will be omitted). The set T in Fig. 2 contains the names of the theories for which a decision procedure for the T-satisfiability problem is assumed available.

Let L be a set of flat literals over the signature $\Sigma_{\mathcal{ADP}}$ to be checked for $\mathcal{ADP}$-satisfiability. The decision procedure $DP_{\mathcal{ADP}}$ first computes the $\mathcal{E}$-instantiation $L^{\mathcal{E}}$ of L (recall from Lemma 4.1 that this can be done in linear time). Then, it enumerates all possible $\mathcal{G}$-instantiations (cf. the *for each* loop in Fig. 2). If it is capable of finding a $\mathcal{G}$-instantiation $L^{\mathcal{G}}$ such that its literals in $L_{\mathcal{P}}^{\mathcal{G}}$ over the signature $\Sigma_{\mathcal{P}}$ are $\mathcal{P}$-satisfiable and its literals in $L_{\mathcal{A}}^{\mathcal{G}}$ over the signature $\Sigma_{\mathcal{A}}$ are $\mathcal{A}$-satisfiable, then $DP_{\mathcal{ADP}}$ returns the $\mathcal{ADP}$-satisfiability of the input set L of literals. Otherwise, if all possible $\mathcal{G}$-instantiations are enumerated and the test of the conditional in the body of the loop always fails, $DP_{\mathcal{ADP}}$ returns the $\mathcal{ADP}$-unsatisfiability of the input set L of literals.

Theorem 4.1 *The constraint satisfiability problem for $\mathcal{ADP}$ is NP-complete.*

Fig. 2 The (extensible) decision procedure for $\mathcal{ADP}$

```
T ⟵ {A, P}
function DP_ADP (L: set of flat literals)
    L^E ⟵ E-instantiation(L)
    for each L^G ⟵ G-instantiation(L^E) do begin
        for each T ∈ T do ρ_T ⟵ DP_T(L^G_T)
        if ⋀_{T∈T}(ρ_T = sat) then return sat
    end
    return unsat
end
```

The proof is based on the following considerations: (1) the constraint satisfiability problem for the theory of Presburger arithmetic reduces to the Integer Linear Programming problem; (2) both Integer Linear Programming and constraint satisfiability for the theory of array are known to be NP-complete problems (see, e.g., [24] and [25] respectively); (3) the size of the $\mathcal{E}$- and $\mathcal{G}$-instantiation closed set is polynomially bounded with respect to the size of the original constraint. From (1) and (2) it follows the NP-hardness of the problem, whereas from (3) and the correctness of $DP_{\mathcal{ADP}}$ (Theorem 4.2) it follows that the problem is in NP, hence the thesis.

4.3 Correctness of the procedure

The termination of $DP_{\mathcal{ADP}}$ is obvious, since the computation of $L^{\mathcal{E}}$ terminates (see Lemma 4.1) and there are only finitely many possible sets $L^{\mathcal{G}}$ to be considered in the *for each* loop of Fig. 2 (see Lemma 4.2).

The soundness and completeness of $DP_{\mathcal{ADP}}$ are consequences of the following combination lemma.

Lemma 4.3 (Combination) *Let L be an $\mathcal{E}$- and $\mathcal{G}$-instantiation closed set. Then the following conditions are equivalent:*

(1) *L is satisfiable in a standard model of $\mathcal{ADP}$;*
(2) *L is $\mathcal{ADP}$-satisfiable;*
(3) *$L_{\mathcal{A}}$ is $\mathcal{A}$-satisfiable and $L_{\mathcal{P}}$ is $\mathcal{P}$-satisfiable.*

Proof Since the implications (1) $\Rightarrow$ (2) $\Rightarrow$ (3) are trivial, it is sufficient to show that (3) $\Rightarrow$ (1) to conclude the proof.

Let $\mathcal{M}'$ be a structure such that $\mathcal{M}' \models \mathcal{A} \cup L_{\mathcal{A}}$ and $\mathcal{N}$ be a structure such that $\mathcal{N} \models \mathcal{P} \cup L_{\mathcal{P}}$. Since $\mathcal{P}$ is complete, we are entitled to assume that $\mathcal{N}$ is the standard structure of natural numbers $\mathbb{N}$. We are now ready to build a standard model $\mathcal{M}$ for $\mathcal{ADP} \cup L$ out of $\mathcal{M}'$ as follows. We take $\textsc{elem}^{\mathcal{M}}$ to be $\textsc{elem}^{\mathcal{M}'}$ and $\bot^{\mathcal{M}}$ to be $\bot^{\mathcal{M}'}$; the free constants occurring in L are interpreted as follows:

(a) For each constant $i : \textsc{index}$ occurring in $L_{\mathcal{P}}$, let $i^{\mathcal{M}} := i^{\mathcal{N}}$;
(b) For each constant $e : \textsc{elem}$ occurring in $L_{\mathcal{A}}$, let $e^{\mathcal{M}} := e^{\mathcal{M}'}$;
(c) For each constant $a : \textsc{array}$ occurring in $L_{\mathcal{A}}$, we define $a^{\mathcal{M}}$ to be the sequence $\{e_n\}$ such that

$$e_n := \begin{cases} \mathsf{select}(a, i)^{\mathcal{M}'} & \text{if } n = i^{\mathcal{M}} \text{ for some } i \text{ occurring in } L_{\mathcal{P}}, \\ \bot^{\mathcal{M}} & \text{otherwise.} \end{cases}$$

The construction in (c) is well-defined: indeed, if two constants i_1 and i_2 of sort $\textsc{index}$ occurring in $L_{\mathcal{P}}$ are interpreted into the same element in $\mathcal{M}$, then $i_1^{\mathcal{N}} = i_2^{\mathcal{N}}$; since L is $\mathcal{G}$-instantiation closed, the atom $i_1 = i_2$ is in $L_{\mathcal{P}}$ (and hence in $L_{\mathcal{A}}$) and so $\mathcal{M}' \models \mathsf{select}(a, i_1) = \mathsf{select}(a, i_2)$.

Now, we show that for each $\ell \in L$, we have $\mathcal{M} \models \ell$. This is obvious for $\ell \in L_{\mathcal{P}}$ and for ℓ of the form $e_1 = e_2$ or $e_1 \neq e_2$, with $e_1, e_2 : \textsc{elem}$. We are left to consider the following cases depending on the form of ℓ:

(1) $\mathsf{select}(a, i) = e$. $\mathcal{M} \models \ell$ because of (a), (b), (c);
(2) $a_1 = a_2$, with $a_1, a_2 : \textsc{array}$. $\mathcal{M} \models \ell$ because $a_1^{\mathcal{M}'} = a_2^{\mathcal{M}'}$, so $\mathsf{select}(a_1, i)^{\mathcal{M}'} = \mathsf{select}(a_2, i)^{\mathcal{M}'}$ for each constant $i : \textsc{index}$ occurring in $L_{\mathcal{P}}$. Hence, $a_1^{\mathcal{M}} = a_2^{\mathcal{M}}$ by (c);
(3) $\mathsf{store}(a_1, i, e) = a_2$. $\mathcal{M} \models \ell$ by considering an argument similar to that used for case (2);
(4) $a_1 \neq a_2$, with $a_1, a_2 : \textsc{array}$. $\mathcal{M} \models \ell$ since

$$\{\mathsf{select}(a_1, i) = e_1, \mathsf{select}(a_2, i) = e_2, e_1 \neq e_2\} \subseteq L_{\mathcal{A}}$$

by Definition 4.1 of $\mathcal{E}$-instantiation closed set of literals and $\mathcal{M}' \models L_{\mathcal{A}}$ and hence $\mathsf{select}(a_1, i)^{\mathcal{M}} \neq \mathsf{select}(a_2, i)^{\mathcal{M}}$ because of (1). As a consequence, we have $a_1^{\mathcal{M}} \neq a_2^{\mathcal{M}}$.
(5) $\mathsf{dim}(a) = i$. We consider two sub-cases, according to (2) of Definition 4.2:

– If $i = 0 \in L_{\mathcal{P}}$ or $i \equiv 0$, then it is sufficient to prove that for each integer n, e_n is equal to $\bot^{\mathcal{M}}$ where $\{e_n\} = a^{\mathcal{M}}$. If $n = j^{\mathcal{M}}$ for some constant $j : \textsc{index}$ such that

$$\{i < j\} \subseteq L_{\mathcal{P}} \quad \text{or} \quad \{i = j\} \subseteq L_{\mathcal{P}},$$

then, since L is $\mathcal{G}$-instantiation closed, $\mathsf{select}(a, j) = \bot \in L_{\mathcal{A}}$ hence $e_n = \bot^{\mathcal{M}}$ by (c); otherwise, $e_n = \bot^{\mathcal{M}}$ by (c).
– If $i \neq 0 \in L_{\mathcal{P}}$, then for each integer $n \geq i^{\mathcal{M}}$, $e_n = \bot^{\mathcal{M}}$ by a similar argument to the one used for the previous sub-case. In fact, we observe that since L is $\mathcal{G}$-instantiation closed, $\mathsf{s}(j) = i$ is in $L_{\mathcal{P}}$ for some constant $j : \textsc{index}$, and both $\mathsf{select}(a, j) = e$ and $e \neq \bot$ must also be in $L_{\mathcal{A}}$, therefore the thesis follows from (b), (c) and (1). □

Now, we are able to state and prove the correctness of $DP_{\mathcal{ADP}}$.

Theorem 4.2 *$DP_{\mathcal{ADP}}$ is a decision procedure for the $\mathcal{ADP}$-satisfiability problem, i.e., for any set L of flat literals, L is $\mathcal{ADP}$-satisfiable if and only if $DP_{\mathcal{ADP}}(L)$ returns satisfiable. Furthermore, $DP_{\mathcal{ADP}}$ decides the satisfiability problem in the standard models of $\mathcal{ADP}$.*

Proof If L is $\mathcal{ADP}$-satisfiable, then it is obvious that $DP_{\mathcal{ADP}}(L)$ returns *satisfiable*. We are left with the task of proving that the converse holds. We will prove that when $DP_{\mathcal{ADP}}(L)$ returns *satisfiable*, then L is satisfiable in a standard model of $\mathcal{ADP}$. If $DP_{\mathcal{ADP}}(L)$ returns *satisfiable*, then $DP_{\mathcal{ADP}}$ has found a $\mathcal{G}$-instantiation $L^{\mathcal{G}}$ of $L^{\mathcal{E}}$ at some iteration of the *for each* loop in Fig. 2. The set $L^{\mathcal{G}}$ is such that

$$L^{\mathcal{G}}_{\mathcal{A}} \text{ is } \mathcal{A}\text{-satisfiable and } L^{\mathcal{G}}_{\mathcal{P}} \text{ is } \mathcal{P}\text{-satisfiable.}$$

From these two facts, the existence of a standard $\mathcal{ADP}$-model of $L^{\mathcal{G}}$ immediately follows by using Lemma 4.3 above. □

5 Extensions of the theory of arrays with dimension

We now show the decidability of the constraint satisfiability problem for some interesting extensions of $\mathcal{ADP}$.

As observed in [18], certain properties of pointer-based data structures, such as *no-aliasing*, can be specified by using first-order axioms. The first extension of $\mathcal{ADP}$ is obtained by adding an axiom recognizing injective arrays (which, according to [18], may characterize memory configurations where pointers satisfy the no-aliasing property) and then showing how to extend the decision procedure for $\mathcal{ADP}$ by an instantiation strategy so as to consider enough (ground) instances of the injectivity axiom. We notice that the decidability of a similar problem in [6] is left open: we are capable of deriving a decidability result since we use a richer theory that identifies a more restricted class of models.

The second extension of $\mathcal{ADP}$ we consider is again motivated by applications in program verification. As already observed in [21], it is quite helpful to regard arrays as functions equipped with an operator to compute their domains. This is used, for example, to define the semantics of separating connectives (supporting local reasoning) in Separation Logic [22]. So, we extend $\mathcal{ADP}$ with a set of axioms characterizing a function which, given an array a, returns the domain D of a, i.e., D is a set of indexes such that $\mathsf{select}(a, i) \neq \bot$ for i in D. We regard this as a first step in the direction of providing automatic support for Separation logic by decision procedures developed in first-order logic.

The section concludes taking into account some other interesting extensions, which exemplify the flexibility of our approach and are all relevant for applications as discussed in, e.g., [6].

5.1 Injective arrays

We extend the (empty) set of predicate symbols in $\mathcal{ADP}$ by the unary predicate symbol $\mathsf{Inj} : \textsc{array}$ which, intuitively, recognizes injective arrays, i.e., arrays containing unique elements, with the exception of the undefined element $\bot$. To formalize the intuitive meaning of Inj, we extend the set of axioms of $\mathcal{ADP}$ by the following definition:

$$\mathsf{Inj}(a) \;\leftrightarrow\; \forall i, j(\mathsf{select}(a, i) = \mathsf{select}(a, j) \rightarrow i = j \vee \mathsf{select}(a, i) = \bot) \qquad (14)$$

where a is an implicitly universally quantified variable of sort $\textsc{array}$. Let $\mathcal{ADP}_{\mathsf{inj}}$ be the theory obtained by extending $\mathcal{ADP}$ with axiom (14). Notice that, since the new predicate Inj has an explicit definition in the theory $\mathcal{ADP}_{\mathsf{inj}}$, every model for $\mathcal{ADP}$ extends uniquely to a model for $\mathcal{ADP}_{\mathsf{inj}}$ (see, e.g., [27]). Furthermore, *standard models of* $\mathcal{ADP}_{\mathsf{inj}}$ will be those models of $\mathcal{ADP}_{\mathsf{inj}}$ whose reduct is a standard model of $\mathcal{ADP}$.

In order to obtain a decision procedure for $\mathcal{ADP}_{\mathsf{inj}}$, it is necessary to find suitable extensions of Definitions 4.1 and 4.2 so that enough instances of (14) are considered and the results of the available decision procedures for $\mathcal{A}$ and $\mathcal{P}$ are conclusive about the satisfiability of the original constraint in the extended theory. We formalize the meaning of "enough instances" for this extension of $\mathcal{ADP}$ in the following two definitions.

Definition 5.1 ($\mathcal{E}_{\mathsf{inj}}$-instantiation closed set of literals) A set L of ground flat literals is $\mathcal{E}_{\mathsf{inj}}$-instantiation closed if and only if L is $\mathcal{E}$-instantiation closed (cf. Definition 4.1) and the following condition is satisfied:

1. If $\neg\mathsf{Inj}(a) \in L$, then $\{\mathsf{select}(a, i) = e, \mathsf{select}(a, j) = e, i < j, e \neq \bot\} \subseteq L$ for some constants e : ELEM, i, j : INDEX.

Definition 5.2 ($\mathcal{G}_{\mathsf{inj}}$-instantiation closed set of literals) A set L of ground flat literals is $\mathcal{G}_{\mathsf{inj}}$-instantiation closed if and only if L is $\mathcal{G}$-instantiation closed (cf. Definition 4.2) and the following conditions are satisfied:

1. If $\mathsf{Inj}(a) \in L$ then, for each constant i of sort INDEX occurring in L, $\mathsf{select}(a, i) = \bot \in L$ or $\{\mathsf{select}(a, i) = e, e \neq \bot\} \subseteq L$ for some constant e : ELEM;
2. If $\{\mathsf{Inj}(a), i < j, \mathsf{select}(a, i) = e_1, \mathsf{select}(a, j) = e_2, e_1 \neq \bot, e_2 \neq \bot\} \subseteq L$, then $e_1 \neq e_2 \in L$.

Lemmas 4.1 and 4.2 can easily be adapted to the theory $\mathcal{ADP}_{\mathsf{inj}}$, taking into consideration the additional requirements of Definitions 5.1 and 5.2. A decision procedure $DP_{\mathcal{ADP}_{\mathsf{inj}}}$ for $\mathcal{ADP}_{\mathsf{inj}}$ can be obtained from $DP_{\mathcal{ADP}}$ by replacing the modules for $\mathcal{E}$- and $\mathcal{G}$-instantiation in Fig. 1 with those taking into account Definitions 5.1 and 5.2. We are now ready to state and prove the correctness of $DP_{\mathcal{ADP}_{\mathsf{inj}}}$.

Theorem 5.1 *$DP_{\mathcal{ADP}_{\mathsf{inj}}}$ is a decision procedure for the $\mathcal{ADP}_{\mathsf{inj}}$-satisfiability problem. Furthermore, $DP_{\mathcal{ADP}_{\mathsf{inj}}}$ decides the constraint satisfiability problem in the standard models of $\mathcal{ADP}_{\mathsf{inj}}$.*

Proof Soundness is trivial. Regarding the key point for completeness, suppose we are given an $\mathcal{E}_{\mathsf{inj}}$- and a $\mathcal{G}_{\mathsf{inj}}$-instantiation closed finite set of literals

$$L = L_{\mathcal{ADP}} \cup L_{\mathsf{inj}}$$

(here L_{inj} is the set of literals from L involving the predicate Inj) such that $L_{\mathcal{A}}$ is $\mathcal{A}$-consistent and $L_{\mathcal{P}}$ is $\mathcal{P}$-consistent. The construction of Lemma 4.3 yields a standard model $\mathcal{M}$ of $\mathcal{ADP}$ satisfying $L_{\mathcal{ADP}}$. We are left to prove that the expansion of $\mathcal{M}$ to a model of $\mathcal{ADP}_{\mathsf{inj}}$ is a model of L_{inj}. But this is easy by Definitions 5.1 and 5.2. □

5.2 Arrays with domain

We equip arrays with a function computing their domain, i.e., the set of indexes at which they store "defined" values, i.e., values distinct from $\bot$. To this end, we need to formalize a very simple theory of sets of indexes, which is a straightforward extension of that used in [2]. Let $\mathcal{S}^{\emptyset}$ be the theory whose sort symbols are BOOL and SET, whose function symbols are $\mathsf{true}, \mathsf{false}$: BOOL, $\emptyset$: SET, mem : INDEX $\times$ SET $\rightarrow$ BOOL, ins : INDEX $\times$ SET $\rightarrow$ SET, and whose axioms are the following:

$$\mathsf{mem}(i, \emptyset) = \mathsf{false} \tag{15}$$

$$\mathsf{mem}(i, \mathsf{ins}(i, s)) = \mathsf{true} \tag{16}$$

$$i_1 \neq i_2 \rightarrow \mathsf{mem}(i_1, \mathsf{ins}(i_2, s)) = \mathsf{mem}(i_1, s) \tag{17}$$

$$\mathsf{true} \neq \mathsf{false} \wedge \forall x : \text{BOOL}\ (x = \mathsf{true} \vee x = \mathsf{false}) \tag{18}$$

where i, i_1, i_2 are implicitly universally quantified variables of sort INDEX and s is an implicitly universally quantified variable of sort SET. Moreover, we will call $\mathcal{S}^{\emptyset}_{-}$ the theory given by the axioms (15), (16), and (17).

Intuitively, $\emptyset$ denotes the empty set, mem is the test for membership of an index to a set, ins adds an index to a set if it is not already in the set. The constants true and false allow us to encode the membership predicate with the Boolean valued function mem. It is possible to adapt the decidability result of [2] to $\mathcal{S}^{\emptyset}$ (see the technical report [14] for further details). Hence, from now on, we consider the availability of a decision procedure for the constraint satisfiability problem of $\mathcal{S}^{\emptyset}$ in addition to those for $\mathcal{A}$ and $\mathcal{P}$.

Since we want to be able to compare sets by using the membership predicate mem, we need to consider the theory $\mathcal{S}^{\emptyset}_{e}$ obtained from $\mathcal{S}^{\emptyset}$ by adding the following axiom of extensionality for sets:

$$\forall i(\mathsf{mem}(i, s_1) = \mathsf{mem}(i, s_2)) \rightarrow s_1 = s_2 \tag{19}$$

where s_1, s_2 are implicitly universally quantified variables of sort SET. The *standard models* of the theory $\mathcal{S}^{\emptyset}_{e}$ are the models in which the sort SET is interpreted as the set of (characteristic functions of) finite subsets of the interpretation of the sort INDEX.

We are now in the position to give a precise definition of the extension of $\mathcal{ADP}$ by the domain function for arrays. Let $\mathcal{ADP}_{\mathsf{dom}}$ be the theory obtained by extending the (disjoint) union of $\mathcal{ADP}$ with $\mathcal{S}^{\emptyset}_{e}$ by the function symbol dom : ARRAY $\rightarrow$ SET together with the following axiom:

$$\mathsf{select}(a, i) = \bot \leftrightarrow \mathsf{mem}(i, \mathsf{dom}(a)) = \mathsf{false} \tag{20}$$

where i and a are implicitly universally quantified variables of sort INDEX and ARRAY, respectively. Again, notice that a standard model of $\mathcal{ADP} \cup \mathcal{S}^{\emptyset}_{e}$ can be expanded in a unique way to a model (called *standard* as well) of $\mathcal{ADP}_{\mathsf{dom}}$.

In order to obtain a decision procedure for the constraint satisfiability problem of $\mathcal{ADP}_{\mathsf{dom}}$, it is necessary to find suitable extensions of Definitions 4.1 and 4.2 so that enough instances of axioms (19) and (20) are considered and the results of the available decision procedures for the constraint satisfiability problems of $\mathcal{A}$, $\mathcal{P}$, and $\mathcal{S}^{\emptyset}$ are conclusive about the satisfiability of the original constraint in the extended theory. We formalize the meaning of "enough instances" for axioms (19) and (20) in the following definitions.

Definition 5.3 ($\mathcal{E}_{\mathsf{set}}$-instantiation closed set of literals) A set L of ground flat literals is $\mathcal{E}_{\mathsf{set}}$-instantiation closed if and only if L is $\mathcal{E}$-instantiation closed (cf. Definition 4.1) and the following condition is satisfied:

1. If $s_1 \neq s_2 \in L$, with s_1, s_2 constants of sort SET, then $\{\mathsf{mem}(i, s_1) = b_1, \mathsf{mem}(i, s_2) = b_2, b_1 \neq b_2\} \subseteq L$ for some constants b_1, b_2 : BOOL, i : INDEX.

Definition 5.4 ($\mathcal{G}_{\mathsf{dom}}$-instantiation closed set of literals) A set L of ground flat literals is $\mathcal{G}_{\mathsf{dom}}$-instantiation closed if and only if L is $\mathcal{G}$-instantiation closed (cf. Definition 4.2) and the following conditions are satisfied:

1. If a literal of the kind $\mathsf{dom}(a) = s_a$ belongs to L, then for each constant i of sort INDEX occurring in L, $\mathsf{select}(a, i) = \bot \in L$ or $\{\mathsf{select}(a, i) = e, e \neq \bot\} \subseteq L$ for some constant e : ELEM;

2. If $\{\mathsf{select}(a, i) = \bot, \mathsf{dom}(a) = s_a\} \subseteq L$ then $\{\mathsf{mem}(i, s_a) = b\,, b \neq \mathsf{true}\} \subseteq L$ for some constant b : BOOL; otherwise, if $\{\mathsf{select}(a, i) = e, e \neq \bot, \mathsf{dom}(a) = s_a\} \subseteq L$ then $\mathsf{mem}(i, s_a) = \mathsf{true} \in L$;

Lemmas 4.1 and 4.2 can easily be adapted to the theory $\mathcal{ADP}_{\mathsf{dom}}$. The decision procedure $DP_{\mathcal{ADP}_{\mathsf{dom}}}$ for the theory $\mathcal{ADP}_{\mathsf{dom}}$ is obtained from $DP_{\mathcal{ADP}}$ by (1) replacing the modules for $\mathcal{E}$- and $\mathcal{G}$-instantiation in Fig. 1 with those taking into account Definitions 5.3 and 5.4 and by (2) adding the decision procedure for $\mathcal{S}^{\emptyset}$ to the set of decision procedures available to the schema in Fig. 2, i.e., by setting T to $\{\mathcal{A}, \mathcal{P}, \mathcal{S}^{\emptyset}\}$.

Theorem 5.2 *$DP_{\mathcal{ADP}_{\mathsf{dom}}}$ is a decision procedure for the $\mathcal{ADP}_{\mathsf{dom}}$-satisfiability problem. Furthermore, $DP_{\mathcal{ADP}_{\mathsf{dom}}}$ decides the satisfiability problem in the standard models of $\mathcal{ADP}_{\mathsf{dom}}$.*

Since the arguments used in the proof of the Theorem above are quite similar to the ones used in Theorem 4.2, we omit the proof.

5.3 Further extensions of $\mathcal{ADP}$

To show the flexibility of our approach, we consider here some further extensions of $\mathcal{ADP}$ whose satisfiability problem can be checked by augmenting the decision procedure of Section 4 with suitable instantiation strategies. The extensions considered below are all relevant for applications as discussed, e.g., in [6]. It is remarkable that the decision procedures for the constraint satisfiability problem for the various extensions considered below can simply be obtained by modifying the modules for $\mathcal{E}$-instantiation and $\mathcal{G}$-instantiation in Fig. 1.

Prefixes. We consider the new binary predicate symbol $\sqsubseteq$: ARRAY $\times$ ARRAY and we extend the set of axioms of $\mathcal{ADP}$ by adding the following sentence:

$$a \sqsubseteq b \quad \leftrightarrow \quad \forall i(i < \mathsf{dim}(a) \rightarrow \mathsf{select}(a, i) = \mathsf{select}(b\,, i)) \tag{21}$$

where i is a variable of sort INDEX, a and b are implicitly universally quantified variables of sort ARRAY. We denote the extended theory with $\mathcal{ADP}_{\mathsf{pfx}}$. Intuitively, a is a prefix of b whenever $a \sqsubseteq b$ holds.

Iterators. We consider two finite sets $\{\mathrm{mapf}_1, \ldots, \mathrm{mapf}_n\}$ and $\{f_1, \ldots, f_n\}$ of fresh unary function symbols such that mapf_k : ARRAY $\rightarrow$ ARRAY and f_k : ELEM $\rightarrow$ ELEM ($k \in \{1, \ldots, n\}$). We extend the set of axioms of $\mathcal{ADP}$ by adding a finite number of sentences of the following form:

$$\mathsf{select}(\mathrm{mapf}_k(a), i) = f_k(\mathsf{select}(a, i)) \tag{22}$$

$$f_k(\bot) = \bot, \tag{23}$$

where i and a are implicitly universally quantified variables of sort INDEX and ARRAY, respectively ($k \in \{1, \ldots, n\}$). We denote the extended theory with $\mathcal{ADP}_{\mathsf{map}}$. Intuitively, $\mathrm{mapf}_k(a)$ can be seen as an application of the higher-order function map, which is routinely used in many functional languages, such as ML or Haskell, i.e., $\mathrm{mapf}_k(a)$ is equivalent to $(\mathsf{map}\ \ f_k\ \ a)$.

Sorting. We consider the new binary predicate symbol $\preceq$: ELEM $\times$ ELEM and we extend the axioms of $\mathcal{ADP}$ by adding sentences stating that $\preceq$ is a total order over the sort ELEM. We also add the unary predicate symbol Sorted over the sort ARRAY, recognizing those arrays which are sorted in ascending order according to the total order $\preceq$ (with the exception of $\perp$ element). We also extend the set of axioms by adding the following sentence:

$$\mathsf{Sorted}(a) \leftrightarrow \forall i, j \left(i < j \rightarrow \begin{pmatrix} \mathsf{select}(a, i) \preceq \mathsf{select}(a, j) \vee \\ \mathsf{select}(a, i) = \perp \quad \vee \\ \mathsf{select}(a, j) = \perp \end{pmatrix} \right) \tag{24}$$

where a is an implicitly universally quantified variable of sort ARRAY. Notice that, because of the ordering over the sort ELEM, in order to obtain a decision procedure for the $\mathcal{ADP}_{\mathsf{ord}}$-satisfiability problem it is also needed to replace the decision procedure for $\mathcal{A}$-satisfiability with a decision procedure obtained by combining *à la* Nelson–Oppen [19] the decision procedure for $\mathcal{A}$ with one for the theory of total order (see, e.g., [3]).

More details on the decision procedures for these extensions can be found in [14]. All the extensions considered above can be combined together in order to obtain a decidable fragment which is very expressive and able to cope with many properties of interest for the field of software verification.

6 Implementation issues

The following section is devoted to address some of the problems arising in the implementation of the procedures presented above. The key issue is how to efficiently handle the non-determinism introduced by the various $\mathcal{G}$-instantiation modules considered above (cf. Definitions 4.2, 5.2, and 5.4). An ad hoc solution to this problem for the theory of arrays with domains will be given by using the rewriting-approach to build satisfiability procedures. Unfortunately, this solution is not general since, for example, the theory of arrays augmented with the injective axiom does not seem to be amenable to such an approach without resorting to suitable extensions of the calculus to handle cancellation axioms (see, e.g. [23]) which are not implemented in state-of-the-art provers. A more general solution, relying on the use of Satisfiability Modulo Theories solvers will then be described which is capable of coping with all the extensions considered above.

6.1 A rewriting-based procedure for $\mathcal{ADP}_{\mathsf{dom}}$

An alternative to the model-theoretic approach described in Section 5.2 is represented by the rewriting-approach to satisfiability procedures described in [2], which allows us to better handle the non-determinism introduced by the guessing. In fact, we can use the Superposition Calculus (from now on denoted by $\mathcal{SP}$, see [20]) to build a decision procedure for the satisfiability problem in the union of the theories $\mathcal{A}_e$ and $\mathcal{S}_e^{\emptyset}$ extended with axiom (20). Such a procedure is then combined with a decision procedure for the satisfiability problem in $\mathcal{P}$ to build a decision procedure for $\mathcal{ADP}_{\mathsf{dom}}$.

In [2], it is shown how to use $\mathcal{SP}$ to build decision procedures for the constraint satisfiability problem of theories axiomatized by a finite set of first-order clauses. The key observation is that, in order to show that $\mathcal{SP}$ is a decision procedure, it is sufficient to prove that $\mathcal{SP}$ terminates on the set of clauses obtained by the union of the axioms of the theory and an arbitrary set of ground and flat literals. According to [2], $\mathcal{SP}$ terminates also for some of the theories considered in this paper, e.g., $\mathcal{A}$ and $\mathcal{S}^{\emptyset}_{-}$ (when considered in isolation). Modularity results in [1] allow us to conclude that $\mathcal{SP}$ also terminates for the union $\mathcal{A} \cup \mathcal{S}^{\emptyset}_{-}$. Unfortunately, this is not enough since our goal is to build a decision procedure for the $\mathcal{ADP}_{\mathsf{dom}}$-satisfiability problem whose set of axioms also contains (18) and (20).

As a preliminary step to applying $\mathcal{SP}$, we need to partially instantiate axioms (18) and (20) with the constants of sort ARRAY and BOOL occurring in L. This is so because $\mathcal{SP}$ does not seem to terminate on theories axiomatizing enumerated data types such as the Booleans (see [4] for a discussion on this point).

Definition 6.1 Let L be a set of ground and flat $\Sigma_{\mathcal{A}_e \cup \mathcal{S}^{\emptyset}}$-literals; we define $\mathcal{I}_L$ to be the following set of (partial) instances of axioms (18) and (20):

$$\begin{aligned} \mathsf{select}(a, x) \neq \bot \vee \mathsf{mem}(x, \mathsf{dom}(a)) &\neq \mathsf{true} \\ \mathsf{select}(a, x) = \bot \vee \mathsf{mem}(x, \mathsf{dom}(a)) &= \mathsf{true} \\ b = \mathsf{true} \vee b &= \mathsf{false} \\ \mathsf{true} &\neq \mathsf{false} \end{aligned}$$

for each $\mathsf{dom}(a) = s$ in L and for each constant b : BOOL occurring in L.

Along the lines of [2], to build a decision procedure for the $\mathcal{ADP}_{\mathsf{dom}}$-satisfiability problem it is necessary to show that $\mathcal{SP}$ terminates on the class of clauses obtained by the union of ground flat literals and the axioms which have not been completely instantiated, namely those in $\mathcal{A}$, $\mathcal{S}^{\emptyset}_{-}$, and those in $\mathcal{I}_L$.

Lemma 6.1 *$\mathcal{SP}$ terminates on $\mathcal{A} \cup \mathcal{S}^{\emptyset}_{-} \cup \mathcal{I}_L \cup L$ for every finite set L of ground and flat $\Sigma_{\mathcal{A} \cup \mathcal{S}^{\emptyset}}$-literals.*

Let us call $\mathcal{ASD}$ the theory axiomatized by $\mathcal{A}_e \cup \mathcal{S}^{\emptyset}_{e} \cup \{(20)\}$. The following lemma is needed to prove the correctness of the decision procedure for $\mathcal{ADP}_{\mathsf{dom}}$.

Lemma 6.2 *Let L be an $\mathcal{E}_{\mathsf{set}}$-instantiation closed set of $\Sigma_{\mathcal{A} \cup \mathcal{S}^{\emptyset}}$-literals. Then, L is $\mathcal{ASD}$-satisfiable if and only if L is $(\mathcal{A} \cup \mathcal{S}^{\emptyset}_{-} \cup \mathcal{I}_L)$-satisfiable.*

For lack of space, we omit the proofs of both of the technical lemmas above (the interested reader can find them in [14]). Below, we denote with $DP_{\mathcal{SP}}$ the function taking a set L of ground $\Sigma_{\mathcal{A} \cup \mathcal{S}^{\emptyset}}$-literals, computing $\mathcal{I}_L$ and then invoking $\mathcal{SP}$ on the clauses $\mathcal{A} \cup \mathcal{S}^{\emptyset}_{-} \cup \mathcal{I}_L \cup L$. If the empty clause is derived by $\mathcal{SP}$, then $DP_{\mathcal{SP}}$ returns *unsatisfiable*; otherwise, it returns *satisfiable*. Hence, the new variant of the decision procedure $DP_{\mathcal{ADP}_{\mathsf{dom}}}$ for the theory $\mathcal{ADP}_{\mathsf{dom}}$ can be obtained from $DP_{\mathcal{ADP}}$ by replacing the module for $\mathcal{E}$-instantiation in Fig. 1 with a module for $\mathcal{E}_{\mathsf{set}}$-instantiation

(cf. Definition 5.3) and by invoking $DP_{\mathcal{SP}}$ and $DP_{\mathcal{P}}$ in Fig. 2, i.e., by setting T to $\{\mathcal{SP}, \mathcal{P}\}$.

Now, we can state and prove the correctness of the new version of $DP_{\mathcal{ADP}_{\mathsf{dom}}}$.

Theorem 6.1 *$DP_{\mathcal{ADP}_{\mathsf{dom}}}$ is a decision procedure for the $\mathcal{ADP}_{\mathsf{dom}}$-satisfiability problem.*

Proof According to the result in Theorem 5.2, an $\mathcal{E}_{\mathsf{set}}$- and a $\mathcal{G}_{\mathsf{dom}}$-instantiation closed finite set of literals

$$L = L_{\mathcal{ADP}} \cup L_{\mathcal{S}^\emptyset} \cup L_{\mathsf{dom}} \tag{25}$$

is $\mathcal{ADP}_{\mathsf{dom}}$-satisfiable whenever $L_{\mathcal{A}}$, $L_{\mathcal{P}}$ and $L_{\mathcal{S}^\emptyset}$ are $\mathcal{A}$-, $\mathcal{P}$- and $\mathcal{S}^\emptyset$-satisfiable, respectively (here L_{dom} is the set of literals from L involving the function dom). From now on, we assume that the set of literals (25) is only $\mathcal{E}_{\mathsf{set}}$- and a $\mathcal{G}$-instantiation closed. We still assume that $L_{\mathcal{P}}$ is $\mathcal{P}$-satisfiable and (this is the new fact due to Lemma 6.2) that $L_{\mathcal{A}} \cup L_{\mathcal{S}^\emptyset} \cup L_{\mathsf{dom}}$ is $\mathcal{ASD}$-satisfiable. Now, consider a model $\mathcal{M}$ of $\mathcal{ASD} \cup L_{\mathcal{A}} \cup L_{\mathcal{S}^\emptyset} \cup L_{\mathsf{dom}}$: looking at this model, we can add to $L_{\mathcal{A}} \cup L_{\mathcal{S}^\emptyset}$ more literals true in $\mathcal{M}$ (let them be $\tilde{L}_{\mathcal{A}} \cup \tilde{L}_{\mathcal{S}^\emptyset}$), in such a way that

$$\tilde{L} = (L_{\mathcal{ADP}} \cup \tilde{L}_{\mathcal{A}}) \cup (L_{\mathcal{S}^\emptyset} \cup \tilde{L}_{\mathcal{S}^\emptyset}) \cup L_{\mathsf{dom}}$$

is $\mathcal{E}_{\mathsf{set}}$- and $\mathcal{G}_{\mathsf{dom}}$-instantiation closed (notice in fact that the newly introduced literals do not contain new constants of sort INDEX); now, $\tilde{L} \supseteq L$ satisfies the requirements of Theorem 5.2 and is $\mathcal{ADP}_{\mathsf{dom}}$-satisfiable. □

6.2 An SMT-based algorithm

We present an algorithm which integrates our instantiation-based schema into an SMT solver by adapting the ideas described in [5], where a Boolean solver is used in order to efficiently handle the guessing phase of non-deterministic procedures.

The decision procedure described in Fig. 3 relies on two simple functions. The former is a propositional abstraction function, i.e., a bijective function *fol2prop* which maps a ground first-order formula φ into a Boolean formula φ^p as follows: *fol2prop* maps Boolean atoms into themselves, ground atoms into fresh Boolean atoms, and is homomorphic with respect to Boolean operators. The second function, *prop2fol* (called, the refinement) is the inverse of *fol2prop*. In the following, the procedure DP_T is a decision procedure for the constraint satisfiability problem for T, where T is $\mathcal{A}$ or $\mathcal{P}$. If a constraint L is T-satisfiable, DP_T returns $(sat, \emptyset)$, otherwise it returns $(unsat, \pi)$ where $\pi \subseteq L$ and π is a T-unsatisfiable set, called the (theory) conflict set. The associated (theory) conflict clause is $\neg\pi$.

The algorithm runs as follows. First of all, the function *flatten* transforms, by introducing sufficiently many fresh constants to name subterms, the input formula φ into an equisatisfiable formula of the kind $\varphi_u \wedge \varphi_s$, where φ_u is a constraint containing just positive flat equalities (including the literal $\mathsf{dim}(\varepsilon) = 0$) and φ_s is a Boolean combination of equalities between constants. Then, according to Definitions 4.1 and

Fig. 3 An SMT-solver for $\mathcal{ADP}$-satisfiability

function SMT (φ: *quantifier-free* $\Sigma^a_{\mathcal{ADP}}$*-formula*)
  $\varphi \longleftarrow$ *flatten*(φ)
  $\varphi \longleftarrow \varphi \wedge e\text{-}inst(\varphi) \wedge g_2\text{-}inst(\varphi)$
  $\varphi \longleftarrow \varphi \wedge g_{3,4}\text{-}inst(\varphi)$
  $\varphi^p \longleftarrow fol2prop(\varphi)$
  while Bool-*satisfiable*(φ^p) **do begin**
    $\beta^p \longleftarrow pick\_total\_assign(\varphi^p)$
    $L^{\mathcal{G}} \longleftarrow g_5\text{-}inst(prop2fol(\beta^p))$
    $(\rho_{\mathcal{A}}, \pi_{\mathcal{A}}) \longleftarrow DP_{\mathcal{A}}(L^{\mathcal{G}}_{\mathcal{A}})$
    $(\rho_{\mathcal{P}}, \pi_{\mathcal{P}}) \longleftarrow DP_{\mathcal{P}}(L^{\mathcal{G}}_{\mathcal{P}})$
    if ($\rho_{\mathcal{A}} = sat \wedge \rho_{\mathcal{P}} = sat$) **then return** *sat* **else**
      if ($\rho_{\mathcal{A}} = unsat$) **then** $\varphi^p \longleftarrow \varphi^p \wedge \neg fol2prop(\pi_{\mathcal{A}})$
      if ($\rho_{\mathcal{P}} = unsat$) **then** $\varphi^p \longleftarrow \varphi^p \wedge \neg fol2prop(\pi_{\mathcal{P}})$
  end while
  return *unsat*
end function

4.2, we add to the input formula φ some of its logical consequences with respect to $\mathcal{ADP}$. More in detail, we have

$$\varphi \longleftarrow \varphi \wedge e\text{-}inst(\varphi) \wedge g_2\text{-}inst(\varphi)$$
$$\varphi \longleftarrow \varphi \wedge g_{3,4}\text{-}inst(\varphi)$$

where the functions *e-inst*, g_2-*inst* and $g_{3,4}$-*inst* are such that

- *e-inst*(φ) is the conjunction of the formulae $a \neq b \rightarrow (\mathsf{select}(a, i) = e_1 \wedge \mathsf{select}(b, i) = e_2 \wedge e_1 \neq e_2)$ for each constants a, b : ARRAY such that $a = b$ occurs in φ (see Definition 4.1);
- g_2-*inst*(φ) is the conjunction of the formulae $i = 0 \vee (e \neq \perp \wedge \mathsf{select}(a, j) = e \wedge \mathsf{s}(j) = i)$ for each constants a : ARRAY, i : INDEX such that $\mathsf{dim}(a) = i$ occurs in φ [see (2) of Definition 4.2]; and
- $g_{3,4}$-*inst*(φ) is the conjunction of the clauses of the kind $i < j \vee i = j \vee j < i$ for each constant i, j : INDEX occurring in φ [see (3) and (4) of Definition 4.2].

At this point, φ contains almost all the atoms needed to eventually obtain $\mathcal{E}$- and $\mathcal{G}$-instantiation closed sets of literals; the only missing closure is w.r.t. condition (5) of Definition 4.2. This will be done by the function g_5-*inst* in the loop, as it will be clear in a moment.

The **while** loop is iterated until there exists a propositional assignment β^p which satisfies the propositional abstraction φ^p of φ. The propositional assignment β^p is refined, thus obtaining a constraint which is (deterministically) closed under condition (5) of Definition 4.2 by the function g_5-*inst*, and then passed to the decision procedures for Presburger arithmetic $DP_{\mathcal{P}}$ and for the constraint satisfiability problem for the theory of arrays $DP_{\mathcal{A}}$. If both procedures return $(sat, \emptyset)$, then the algorithm stops returning satisfiability; otherwise, as it is customary in lazy SMT solvers (see, e.g., [5]), the corresponding conflict clause is used to prune the search space in order to avoid enumerating useless guesses, i.e., all those sharing the same conflict set.

The correctness of the procedure can be obtained along the lines of the Delayed Theory Combination algorithm in [5]. The main differences lie in showing that the pre-processing steps preserve the $\mathcal{ADP}$-equisatisfiability and that $L^{\mathcal{G}}$ is an $\mathcal{E}$- and $\mathcal{G}$-instantiation closed set of literals so that Lemma 4.3 above can be re-used.

Finally, we notice that all the extensions considered in Section 5.3 can be easily integrated in the algorithm of Fig. 3 by adapting the g_*'s functions in order to mirror the extensions in the definition of $\mathcal{G}$-instantiation closed sets.

7 Conclusions and future work

We have considered extensions of the theory of arrays which are relevant for many important applications such as program verification. These extensions are such that the indexes of arrays have the algebraic structure of Presburger arithmetic and the theory of arrays is augmented with axioms characterizing additional symbols such as dimension, injectivity, or the domain of definition of arrays. We have obtained the decidability of the constraint satisfiability problem for all the considered extensions by a combination of decision procedures for the theories of arrays and Presburger arithmetic with various instantiation strategies based both on model-theoretic and rewriting-based methods. We have also described techniques for the efficient implementation of the non-deterministic decision procedures by adapting techniques developed in the SMT community.

There are two lines of future work. First, we plan to implement the SMT-based algorithm described in Section 6.2 in haRVey [8] and perform some experimental evaluations. In particular, this should allow us to implement the two variants of the decision procedure for $\mathcal{ADP}$ and to compare their relative benefits on some significant problems. The second line of future work is longer term and concerns the definition of a general framework to declaratively specify and prove correct procedures for theories obtained by augmenting a base theory (for which a decision procedure is available) with instantiation strategies for some selected class of axiomatic extension. Ideally, such a framework should allow us to develop general decidability results which can be instantiated to the theories of container data-structures such as those considered in this paper as well as sets (with all the usual set-theoretic operations of union, intersection, cardinality, and so on), lists, and multisets.

Acknowledgements We thank the anonymous reviewers for their comments and suggestions to improve the readability of the paper. We also thank Alessandro Armando for interesting discussions about the topic of the paper.

References

1. Armando, A., Bonacina, M.P., Ranise, S., Schulz, S.: On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal. In: Proceedings of the 5th International Workshop on Frontiers of Combining Systems (FroCoS'05). LNCS, vol. 3717, pp. 65–80. Springer (2005)
2. Armando, A., Ranise, S., Rusinowitch, M.: A rewriting approach to satisfiability procedures. Inf. Comput. **183**(2), 140–164 (2003)

3. Bjørner, N., Stickel, M.E., Uribe, T.E.: A practical integration of first-order reasoning and decision procedures. In: Proceedings of the 14th International Conference on Automated Deduction (CADE'97). LNCS, vol. 1249, pp. 101–115. Springer (1997)
4. Bonacina, M.P., Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decidability and undecidability results for Nelson-Oppen and rewrite-based decision procedures. In: Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06). LNCS, vol. 4130, pp. 513–527. Springer (2006)
5. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., Ranise, S., van Rossum, P., Sebastiani, R.: Efficient theory combination via boolean search. Inf. Comput. **204**(10), 1493–1525 (2006)
6. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'06). LNCS, vol. 3855, pp. 427–442. Springer (2006)
7. Brodnik, A., Carlsson, S., Demaine, E.D., Munro, J.I., Sedgewick, R.: Resizable arrays in optimal time and space. In: Proceedings of 6th International Workshop on Algorithms and Data Structures (WADS'99). LNCS, vol. 1663, pp. 37–48. Springer (1999)
8. Déharbe, D., Ranise, S.: Light-weight theorem proving for debugging and verifying units of code. In: Proceedings of the 1st International Conference on Software Engineering and Formal Methods (SEFM'03), pp. 220–228. IEEE Computer Society (2003)
9. Downey, P.J., Sethi, R.: Assignment commands with array references. J. ACM **25**(4), 652–666 (1978)
10. Enderton, H.B.: A Mathematical Introduction to Logic. Academic, New York (1972)
11. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Harper & Row (1986)
12. Ganzinger, H., Korovin, K.: Integrating equational reasoning in instantiation-based theorem proving. In: Proceedings of the 18th International Workshop on Computer Science in Logic (CSL'04). LNCS, vol. 3210, pp. 71–84. Springer (2004)
13. Ghilardi, S.: Model-theoretic methods in combined constraint satisfiability. J. Autom. Reason. **33**(3–4), 221–249 (2004)
14. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Deciding extension of the theory of arrays by integrating decision procedures and instantiation strategies. Rapporto Interno DSI 309-06, Università degli Studi di Milano, Milano (Italy) (2006) Available at http://homes.dsi.unimi.it/∼zucchell/publications/techreport/GhiNiRaZu-RI309-06.pdf
15. Jaffar, J.: Presburger arithmetic with array segments. Inf. Process. Lett. **12**(2), 79–82 (1981)
16. Mateti, P.: A decision procedure for the correctness of a class of programs. J. ACM **28**(2), 215–232 (1981)
17. McCarthy, J.: Towards a mathematical theory of computation. In: Proceedings of IFIP Congress, pp. 21–28 (1962)
18. McPeak, S., Necula, G.: Data structures specification via local equality axioms. In: Proceedings of the 17th International Conference on Computer Aided Verification (CAV'05). LNCS, vol. 3576, pp. 476–490. Springer (2005)
19. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. ACM Trans. Program. Lang. Syst. **1**(2), 245–257 (1979)
20. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, chapter 7, pp. 371–443. Elsevier Science (2001)
21. Reynolds, J.C.: Reasoning about arrays. Commun. ACM **22**(5), 290–299 (1979)
22. Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: Proceedings of the 17th IEEE Symposium on Logic in Computer Science (LICS'02), pp. 55–74. IEEE Computer Society (2002)
23. Rusinowitch, M.: Démonstration Automatique (Techniques de réécriture). InterEditions (1989)
24. Schrijver, A.: Theory of Linear and Integer Programming. Wiley (1986)
25. Stump, A., Barrett, C.W., Dill, D.L., Levitt, J.: A decision procedure for an extensional theory of arrays. In: Proceedings of the 16th IEEE Symposium on Logic in Computer Science (LICS'01). pp. 29–37. IEEE Computer Society (2001)
26. Suzuki, N., Jefferson, D.R.: Verification decidability of Presburger array programs. J. ACM **27**(1), 191–205 (1980)
27. van Dalen, D.: Logic and Structure, 2nd edn. Springer (1989)