A new Acceleration-based Combination Framework for Array Properties

Francesco Alberti[1], Silvio Ghilardi[2], Natasha Sharygina[3]

[1] Fondazione Centro San Raffaele, Milan, Italy
[2] Università degli Studi di Milano, Milan, Italy
[3] Università della Svizzera Italiana, Lugano, Switzerland

Abstract. This paper presents an acceleration-based combination framework for checking the satisfiability of classes of quantified formulæ of the theory of arrays. We identify sufficient conditions for which an 'acceleratability' result can be used as a black-box module inside such satisfiability procedures. Besides establishing new decidability results and relating them to results from recent literature, we discuss the application of our combination framework to the problem of checking the safety of imperative programs with arrays.

1 Introduction

The theory of arrays is one of the most relevant theories for software verification, this is the reason why current research in automated reasoning dedicated so much effort in establishing decision and complexity results for it. From a logical perspective, arrays can be modeled just by adding free function symbols to some fragment of arithmetic. As soon as quantified formulæ are concerned, however, satisfiability becomes intractable when free unary function symbols are added to mild fragments of arithmetic [15]. Nevertheless, since applications require the use of quantifiers, e.g. in order to express invariants of program loops, it becomes crucial to identify sufficiently expressive tractable quantified fragments of the theory of arrays.

Various decidability (and sometimes also complexity) results for such fragments are known from the literature. These results are often orthogonal to each other, rely on rather different techniques, and finding common generalizations is a hard task. Let us mention for instance two contributions from recent literature, namely the decidability results for SIL-fragments in [14] and those for flat mono-sorted fragments in [4]. The flat mono-sorted fragment of [4] is decidable via an SMT-based combination method involving an extra quantifier-elimination step; on the contrary the SIL-fragment of [14] is decided by a procedure that requires back and forth conversions between logic and automata. The SIL-decidable fragment of [14] has a heavy syntactic limitation on consequents of guards: such consequents must be difference bound constraints. On the other hand, the main limitation of the flat mono-sorted fragment in [4], inherited by an analogous limitation from [9], is the impossibility of applying dereference to terms which are not variables in the consequents of guards. This limitation typically prevents

applications to programs where terms like $a[i]$ and $a[i+1]$ are *both* used as, for instance, in array updates.

The technique used in [14] exploits previous acceleration results for difference bound constraints; acceleration (i.e. the definability of the transitive closure of special classes of relations [5–8, 10, 11]) plays an important role in several model-checking approaches which are, in a sense, orthogonal to the SMT-inspired model-checking methods. The curious fact is that acceleration results, previously established using counter automata, were transferred in [14] to array theories via another further conversion from array logic to counter automata formalism.

The contribution of this paper is the definition of a new framework for checking the satisfiability of quantified array formulæ. The principal feature of our framework is that it can exploit and combine acceleration results as *black box* modules. Indeed, the main questions we answer in this paper, in a combination spirit, are the following: *can we use acceleration results as they are, i.e., as black box modules inside decision procedures for array fragments*? What are the *formal conditions* that such 'acceleration modules' have to satisfy in order to be combined as black box modules? Formally, we will answer these two questions with the Definitions 1-2 and with Theorem 3. In particular, the algorithm of Theorem 3 supplies a simple 'guess-and-group' preprocessing step putting current SMT-solvers in the condition of importing acceleration modules. Decidability results like those in [4] and [14] follow as immediate consequences, and further decidable array property fragments can be designed by mere combination.

A remark about acceleration is in order. In our earlier work [2,4], we adopted acceleration techniques into SMT-based software model-checking; to this aim, we investigated what in this paper will be called 'vertical' acceleration, i.e., acceleratability of array relations expressed via specific (syntactically characterized) formulæ in the theory of arrays. In this paper the aim is different, because acceleratability in the underlying arithmetic is used as an ingredient for designing satisfiability procedures at the more complex level of array formulæ. We will refer to this acceleration as 'horizontal' acceleration.

A running example Consider the program of Fig. 1. We want to prove that the assertion in location 4 cannot be violated. The formal proof we produce is *precise*, it does not rely on any form of abstraction or of over-approximation. To do this we need two forms of acceleration: (i) we need *vertical* acceleration (i.e. acceleration at the level of the theory of arrays) to summarize the two loops; (ii) we need *horizontal* acceleration (i.e. acceleration at the level of the underlying arithmetic) in order to discharge the proof obligation coming from (i).

The program in Fig. 1 has two array variables (namely $\mathtt{a}_1, \mathtt{a}_2$) and an integer variable (namely $\mathtt{I}$). An error path violating the assertion should comprise the following steps: (1) an initialization step leading to initial values $\mathtt{a}_1^{(1)}, \mathtt{a}_2^{(1)}, \mathtt{I}^{(1)}$; (2) n executions of the loop in location 2 leading from $\mathtt{a}_1^{(1)}, \mathtt{a}_2^{(1)}, \mathtt{I}^{(1)}$ to updated values $\mathtt{a}_1^{(2)}, \mathtt{a}_2^{(2)}, \mathtt{I}^{(2)}$; (3) the exit step from the loop in location 2 and the execution of the instruction in location 3, leading from $\mathtt{a}_1^{(2)}, \mathtt{a}_2^{(2)}, \mathtt{I}^{(2)}$ to updated values $\mathtt{a}_1^{(3)}, \mathtt{a}_2^{(3)}, \mathtt{I}^{(3)}$; (4) m executions of the loop in location 4 leading from

$$
\begin{array}{l}
\mathsf{int}\ \mathtt{a}_1[N{+}1];\ \mathsf{int}\ \mathtt{a}_2[N{+}1];\ \mathsf{int}\ \mathtt{I};\\
1\ \ \mathtt{I} = 0;\ \mathtt{a}_1[\mathtt{I}] = 0;\ \mathtt{a}_2[\mathtt{I}] = 0;\\
2\ \ \mathsf{while}\ (\mathtt{I} < N)\ \left\{\begin{array}{l}\mathtt{a}_1[\mathtt{I}+1] = \mathtt{a}_1[\mathtt{I}] + 1;\\ \mathtt{a}_2[\mathtt{I}+1] = \mathtt{I} + 1;\\ \mathtt{I}{+}{+};\end{array}\right\}\\
3\ \ \mathtt{I} = 0;\\
4\ \ \mathsf{while}\ (\mathtt{I} < N)\ \left\{\begin{array}{l}\mathsf{assert}(\mathtt{a}_1[\mathtt{I}{+}1] = \mathtt{a}_2[\mathtt{I}{+}1]);\\ \mathtt{I}{+}{+};\end{array}\right\}
\end{array}
$$

Fig. 1: Running example.

$$
\begin{array}{ll}
(\alpha_1) & \mathtt{I}^{(1)} = 0 \ \wedge\ \mathtt{a}_1^{(1)}[\mathtt{I}^{(1)}] = 0 \ \wedge\ \mathtt{a}_2^{(1)}[\mathtt{I}^{(1)}] = 0\\
(\alpha_2) & \left(\begin{array}{l}
\forall i\ \left(\begin{array}{c}\mathtt{I}^{(1)} \le i < \mathtt{I}^{(1)} + n \ \rightarrow\ i < N \ \wedge\ \mathtt{a}_1^{(2)}[i{+}1] = \mathtt{a}_1^{(2)}[i] + 1 \ \wedge\\ \wedge\ \mathtt{a}_2^{(2)}[i{+}1] = i{+}1\end{array}\right) \ \wedge\\
\wedge\ \forall i\ \left(\mathtt{I}^{(1)}{+}n{+}1 \le i \le N \ \rightarrow\ \mathtt{a}_1^{(2)}[i] = \mathtt{a}_1^{(1)}[i] \ \wedge\ \mathtt{a}_2^{(2)}[i] = \mathtt{a}_2^{(1)}[i]\right) \wedge\\
\wedge\ \forall i\ \left(0 \le i \le \mathtt{I}^{(1)} \ \rightarrow\ \mathtt{a}_1^{(2)}[i] = \mathtt{a}_1^{(1)}[i] \ \wedge\ \mathtt{a}_2^{(2)}[i] = \mathtt{a}_2^{(1)}[i]\right) \ \wedge\\
\wedge\ \mathtt{I}^{(2)} = \mathtt{I}^{(1)}{+}n
\end{array}\right)\\
(\alpha_3) & \mathtt{I}^{(2)} \ge N \ \wedge\ \mathtt{I}^{(3)} = 0 \ \wedge\ \mathtt{a}_1^{(3)} = \mathtt{a}_1^{(2)} \ \wedge\ \mathtt{a}_2^{(3)} = \mathtt{a}_2^{(2)}\\
(\alpha_4) & \left(\begin{array}{l}
\forall i\ \left(\mathtt{I}^{(3)} \le i < \mathtt{I}^{(3)}{+}m \ \rightarrow\ i < N \ \wedge\ \mathtt{a}_1^{(3)}[i{+}1] = \mathtt{a}_2^{(3)}[i{+}1]\right) \ \wedge\\
\wedge\ \mathtt{a}_1^{(4)} = \mathtt{a}_1^{(3)} \ \wedge\ \mathtt{a}_2^{(4)} = \mathtt{a}_2^{(3)} \ \wedge\\
\wedge\ \mathtt{I}^{(4)} = \mathtt{I}^{(3)}{+}m
\end{array}\right)\\
(\alpha_5) & \mathtt{a}_1^{(4)}[\mathtt{I}^{(4)}{+}1] \ne \mathtt{a}_2^{(4)}[\mathtt{I}^{(4)}{+}1]
\end{array}
$$

Fig. 2: Proof obligation.

$\mathtt{a}_1^{(3)}, \mathtt{a}_2^{(3)}, \mathtt{I}^{(3)}$ to updated values $\mathtt{a}_1^{(4)}, \mathtt{a}_2^{(4)}, \mathtt{I}^{(4)}$; (5) the satisfiability of the error exit condition from the loop in location 4 by the final values $\mathtt{a}_1^{(4)}, \mathtt{a}_2^{(4)}, \mathtt{I}^{(4)}$. Thus, the error path is feasible iff the conjunction of the formulæ $(\alpha_1)-(\alpha_5)$ from Fig. 2 is satisfiable.

The formulæ (α_2) and (α_4) are computed following mechanical patterns for 'vertical' array acceleration (see Section 5 for more). The satisfiability test of the conjunction $(\alpha_1) \wedge \cdots \wedge (\alpha_5)$ is not handled by current SMT-solvers and it is not trivial indeed, because quantifiers and some form of induction are involved (to make first order logic conversion, read array equalities like $\mathtt{a}_1^{(3)} = \mathtt{a}_1^{(4)}$ as $\forall i\ (\mathtt{a}_1^{(3)}[i] = \mathtt{a}_1^{(4)}[i])$). In order to discharge the proof obligation consisting of the satisfiability test of the conjunction of the formulæ $(\alpha_1) - (\alpha_5)$, the method we propose in Theorems 3-4 below relies on acceleration results for

fragments of plain arithmetic. Roughly speaking, the idea is the following. Take for instance the consequent of the first guard of (α_2): such a consequent comes from the loop body of location 2 in Fig. 1. The instructions describing such loop can be converted into a logical formula relating the values of the program variables $\mathtt{I}, \mathtt{a}_1, \mathtt{a}_2$ before a single execution of the loop with the corresponding values $\overline{\mathtt{I}}, \overline{\mathtt{a}_1}, \overline{\mathtt{a}_2}$ after the single execution of the loop. If, in such a logical formula, we abstract out the terms $\mathtt{I}, \mathtt{a}_1[\mathtt{I}], \mathtt{a}_2[\mathtt{I}], \overline{\mathtt{a}_1}[\mathtt{I}], \overline{\mathtt{a}_2}[\mathtt{I}]$ with the fresh variables $i, e_1, e_2, \bar{e}_1, \bar{e}_2$ and the terms $\mathtt{I}{+}1, \mathtt{a}_1[\mathtt{I}{+}1], \mathtt{a}_2[\mathtt{I}{+}1], \overline{\mathtt{a}_1}[\mathtt{I}{+}1], \overline{\mathtt{a}_2}[\mathtt{I}{+}1]$ with the corresponding primed variables $i', e_1', e_2', \bar{e}_1', \bar{e}_2'$ we get a purely arithmetical relation $\phi(i, e_1, e_2, \bar{e}_1, \bar{e}_2, i', e_1', e_2', \bar{e}_1', \bar{e}_2')$ between 5-tuples of variables. If this relation is acceleratable, we can replace (up to satisfiability) the first guard of (α_2) with the arithmetical formula expressing the fact that the relation denoted by ϕ is composed with itself $(\mathtt{I}^{(1)} + n) - \mathtt{I}^{(1)} = n$ times. In this way, we get a reduction of the proof obligation of Fig. 2 to plain Presburger arithmetic. Of course, the above description of the reduction is very loose and quite incomplete, we shall turn to this Example in Section 5 and run it in full details. Here we just observe that accelerations in fragments of arithmetic are used as *black boxes* during the *horizontal* phase of the reduction (this phase is computing accelerations inside array intervals, whereas vertical acceleration computes acceleration between array variables as wholes).

Plan of the paper. Section 2 fixes notations; in Section 3 we introduce acceleratable fragments and in Section 4 we use them in decision procedures for quantified array formulæ. In Section 5 we show applications to reachability problems for array programs. Section 6 concludes. For space reasons, additional material (including some proofs) has been moved to Appendixes.

2 Notation

We work in a decidable fragment of arithmetic (typically Presburger arithmetic, but given the modularity of our approach, we can consider even more expressive fragments like [16]); we expand the related language with *free constants* and *free unary function symbols*. When we speak about truth or about validity, *we refer to the structures having as reduct the standard model of the integers* with the natural interpretation of the arithmetic symbols. In order to make our language more manageable, we may enrich it with definable function and predicate symbols (see any textbook in mathematical logic, like [17] for the notion of a definable predicate or function symbol). A *purely arithmetical* formula is a formula that does not contain free function symbols (notice however that such a formula may contain *parameters*, i.e. free constants).

It is convenient to partition the set of variables we use into two disjoint sets $\mathcal{V} = \{x, y, z, w, \ldots, i, j, \ldots\}$ and $\mathcal{V}' = \{x', y', z', w', \ldots, i', j', \ldots\}$, where $\mathcal{V}'$ contains precisely a 'primed' copy of each variable in $\mathcal{V}$. *Renaming substitutions* are bijections σ on variables respecting primed copies (i.e. we have $\sigma(x)' = \sigma(x')$ for all x). Free constants are indicated with letters $c, d, \ldots$, terms with letters $t, u, \ldots$ and formulæ with letters $\phi, \psi, \ldots$. Underlined or bold letters usually

denote tuples (of variables, constants, terms) of unspecified length. With $\underline{t} = \underline{u}$ we mean component-wise equality, i.e. $\bigwedge_i t_i = u_i$, where it is implicitly assumed that $\underline{t}, \underline{u}$ have the same length and that $\underline{t} = t_1, \ldots, t_n$ and $\underline{u} = u_1, \ldots, u_n$. The notation $\phi(\underline{x}), t(\underline{x})$ indicates that *at most* the variables $\underline{x}$ occur free in ϕ, t.

When we talk about arrays we assume that they are modeled as unary free function symbols to be denoted with letters $a, b, \ldots$. Read operation (i.e. function application) is denoted with $[-]$; $\mathbf{a}, \mathbf{b}$ stands for tuples of array variables. If $\mathbf{a} = a_1, \ldots, a_n$ and $\underline{t} = t_1, \ldots, t_n$, then $\mathbf{a}[\underline{t}]$ stands for $a_1[t_1], a_2[t_2], \ldots, a_n[t_n]$;[4] however, if $\underline{t}$ is a single term, we may use $\mathbf{a}[t]$ for $a_1[t], \ldots, a_n[t]$. Notations like $\phi(\underline{x}, \mathbf{a}[\underline{t}])$ mean that $\phi(\underline{x}, \underline{y})$ is purely arithmetical and that the tuple of distinct variables $\underline{y}$ has been component-wise replaced by the length-matching tuple of terms $\mathbf{a}[\underline{t}]$.

Arrays are equipped with length; for simplicity, we assume that length is the same for all arrays we consider; we represent it by a free constant N. In our intended models, for every array a, *we assume that* $a[x]$ *is equal to a conventional value (say, 0) for any* x *outside the interval* $[0, N]$. When we discuss satisfiability of array fragments, we are only interested in sub-intervals of $[0, N]$, hence we use notations like $t \in [u_1, u_2]$ to mean the conjunction $0 \leq u_1 \leq t \leq u_2 \leq N$; similarly, $t \in [u_1, u_2)$ means $t \in [u_1, u_2 - 1]$. We may also use standard notation for relativized quantifiers, e.g. $\forall i \in [u_1, u_2)\, \psi$ abbreviates $\forall i\, (i \in [u_1, u_2) \to \psi)$.

3 Acceleratable fragments

A formula $\phi(\underline{x}, \underline{x}')$ denotes a relation among tuples; $\phi^n(\underline{x}, \underline{x}')$ is the formula representing the composition of the relation denoted by ϕ with itself n-times. More precisely, we have

$$\phi^1(\underline{x}, \underline{x}') : \equiv \phi(\underline{x}, \underline{x}'); \qquad \phi^{n+1}(\underline{x}, \underline{x}') : \equiv \exists \underline{x}^\sharp\, (\phi^n(\underline{x}, \underline{x}^\sharp) \wedge \phi(\underline{x}^\sharp, \underline{x}')).$$

Definition 1. *A purely arithmetical formula* $\phi(\underline{x}, \underline{x}')$*, is said to be* acceleratable *iff there exists (and one can actually compute) a formula* $\phi^*(\underline{x}, \underline{x}', j)$ *such that for all* $n \in \mathbb{N}$

$$\models \phi^n(\underline{x}, \underline{x}') \leftrightarrow \phi^*(\underline{x}, \underline{x}', \bar{n}) \tag{1}$$

where $\bar{n}$ *is the* n*-th numeral, i.e. it is* $S(\cdots S(0) \cdots)$*, where* S *- the successor symbol - is applied* n*-times.*[5]

Definition 2. *A set of purely arithmetical formulæ* Γ *is said to be an* acceleratable fragment *iff every* $\phi \in \Gamma$ *is acceleratable and* Γ *is closed under conjunctions and renaming substitutions. An acceleratable fragment* Γ *is said to be* normal *iff it contains the formulæ* $x' = x + 1$ *and* $y' = x'$ *for every variables* x, y*.*

Closure under conjunctions of acceleratable fragments is a crucial condition: we need it in the 'grouping' step of the modular algorithm of Theorem 3; normality is required in the applications from Section 5. We supply below, using

[4] This is different from previous papers of ours.

[5] Sometimes we shall write n instead of $\bar{n}$ if confusion does not arise.

relevant results from the literature, some important examples of acceleratable fragments.

Theorem 1. [8,10] Difference Bounds Constraints, *i.e. conjunctions of formulae of the kind*

$$x - x' \bowtie \bar{n}, \quad x - y \bowtie \bar{n}, \quad x' - y' \bowtie \bar{n}$$

(where $n \in \mathbb{Z}$ and $\bowtie \in \{\leq, \geq\}$) are a normal acceleratable fragment.

Theorem 2. [6] Octagons, *i.e. conjunctions of formulae of the kind*

$$x \pm x' \bowtie \bar{n}, \quad x \pm y \bowtie \bar{n}, \quad x' \pm y' \bowtie \bar{n}, \quad 2x \bowtie \bar{n}, \quad 2x' \bowtie \bar{n}$$

(where $n \in \mathbb{Z}$ and $\bowtie \in \{\leq, \geq\}$) are a normal acceleratable fragment.

Another class is introduced in Proposition 1 below. This class (called the class of iteratable formulæ) is ubiquitous; in essence, the idea of an iteratable formula is based on the simple idea of combining (a generalized form of) variable increments with non-deterministic updates; the formal definition of an iteratable formula requires nevertheless some technicalities to match Definition 2.

We recall the notion of an iterator from [2]. Given a m-tuple of terms

$$\mathbf{u}(\underline{x}) \ := \ u_1(x_1, \ldots, x_m), \ldots, u_m(x_1, \ldots, x_m) \tag{2}$$

containing the m variables $\underline{x} = x_1, \ldots, x_m$, we indicate with $\mathbf{u}^n$ the term expressing the n-times composition of (the function denoted by) $\mathbf{u}$ with itself. Formally, we have $\mathbf{u}^0(\underline{x}) := \underline{x}$ and

$$\mathbf{u}^{n+1}(\underline{x}) \ := \ u_1(\mathbf{u}^n(\underline{x})), \ldots, u_m(\mathbf{u}^n(\underline{x})) \ .$$

Definition 3. *A tuple of terms $\mathbf{u}$ like* (2) *is said to be an* iterator *iff there is an m-tuple of $m+1$-ary terms $\mathbf{u}^*(\underline{x}, y) \ := \ u_1^*(x_1, \ldots, x_m, y), \ldots, u_m^*(x_1, \ldots, x_m, y)$ such that for any natural number $n \geq 0$ it happens that the formula*

$$\mathbf{u}^n(\underline{x}) = \mathbf{u}^*(\underline{x}, \bar{n}) \tag{3}$$

is valid.[6] *The ordered tuple of (distinct) variables $(x_1, \ldots, x_m)$ is said to be the* domain *of the iterator $\mathbf{u}$ and $\mathbf{u}$ is said to be an m-ary iterator.*

Example 1. The canonical example is when we have $m = 1$ and $\mathbf{u} := u_1(x_1) := x_1 + 1$; this is an iterator with $u_1^*(x_1, y) := x_1 + y$. □

Example 2. The previous example can be modified, by choosing $\mathbf{u}$ to be $x_1 + \bar{k}$, for some integer $k \neq 0$: then we have $u_1^*(x_1, y) := x_1 + k * y$ (where $k * y$ is the sum $y + \cdots + y$ of k copies of y). □

Example 3. Sometimes we need to use definable functions to build $\mathbf{u}^*$. Take $\mathbf{u}$ to be $\bar{n} - x_1$; then we have $u_1^*(x_1, y) :=$ (**if** $y \equiv 0$ $(mod\ 2)$ **then** x_1 **else** $\bar{n} - x_1$). □

[6] Recall that in this paper 'validity' means validity in the class of our intended structures - those having the standard model of arithmetic as reduct.

Example 4. Finite monoid affine transformations [11] supply another interesting example. Let v be a vector from $\mathbb{Z}^m$ and let M be a $m \times m$ integer matrix generating a finite monoid (i.e. we have that $M^{k+l} = M^k$ for some $k, l > 0$). Putting $\mathbf{u}(\underline{x}) := M\underline{x} + v$, we get an iterator (the definition of $\mathbf{u}^*(\underline{x}, y)$ requires the identification of a rather complex - but straightforward - definable function, see [11]). □

Definition 4. *A formula $\phi(\underline{x}, \underline{x}')$ is said to be* iteratable *iff there is a finite set I_ϕ of iterators such that:*

(0) *the variables $\underline{x}, \underline{x}'$ are partitioned as $\underline{z}, \underline{z}', \underline{w}'$ (notice that the unprimed variables $\underline{w}$ do not occur in ϕ);*
(i) *the domain of every $\mathbf{u} \in I_\phi$ is included in $\underline{z}$;*
(ii) *every $z \in \underline{z}$ belongs to the domain of at least one $\mathbf{u} \in I_\phi$;*
(ii) *if $(z_{i_1}, \ldots, z_{i_s}) \subseteq \underline{z}$ is the domain of $\mathbf{u} = (u_1, \ldots, u_s) \in I_\phi$, then*

$$\phi(\underline{x}, \underline{x}') \to z'_{i_j} = u_j(z_{i_1}, \ldots, z_{i_s}) \tag{4}$$

is valid for all $j = 1, \ldots, s$.

Thus, in iteratable formulae, the 'updates' z'_i are deterministic and expressed via an iterator; the reason why we allow I_ϕ to be a set (not just a singleton) is because we want to ensure closure under conjunctions of iteratable formulæ. The typical example of an iteratable formula is a formula of the kind

$$\underline{z}' = \mathbf{u}(\underline{z}) \wedge \psi(\underline{z}, \underline{z}', \underline{w}') \tag{5}$$

where ψ is arbitrary and $\mathbf{u}$ is an iterator with domain $\underline{z}$. Notice that, when taking iterated composition of the relation denoted by the above formula with itself, the variables $\underline{w}'$ are non-deterministically chosen at each step.

Proposition 1. *The set of iteratable formulæ form a normal acceleratable fragment.*

Proof. The full proof is deferred to Appendix B. Here we just give the explicit definition of the accelerated formula ϕ^* for an iteratable ϕ. Suppose that free variables occurring in ϕ are partitioned as $z_1, \ldots, z_n, z'_1, \ldots, z'_n, \underline{w}'$; to make our notation more compact, we let $\underline{z} := z_1, \ldots, z_n$ and $\underline{x} := \underline{z}, \underline{w}$. The formula $\phi^*(\underline{x}, \underline{x}', j)$ is given by

$$\underline{z}' = \mathbf{v}(\underline{z}, j) \wedge \forall k \in [0, j)\ \exists \underline{\tilde{w}} \left(\begin{array}{l} (k = j-1 \to \underline{\tilde{w}} = \underline{w}') \ \wedge \\ \quad \wedge\ \phi(\mathbf{v}(\underline{z}, k), \mathbf{v}(\underline{z}, k+1), \tilde{w}) \end{array} \right) \tag{6}$$

where the tuple of terms $\mathbf{v}(\underline{z}, k) := v_1(\underline{z}, k), \ldots, v_n(\underline{z}, k)$ is obtained as follows. For every $z_l \in \underline{z}$, choose some iterator $\mathbf{u}^l \in I_\phi$ such that z_l is in the domain of $\mathbf{u}^l$: if z_l occurs at the h-th place in such a domain, we let v_l be $(\mathbf{u}^l)^*_h(\underline{z}, k)$. In other words: we compute the iteration $(\mathbf{u}^l)^*$ of $\mathbf{u}^l$ according to (3), take its h-component, and apply it to $\underline{z}$ and k (actually, $(\mathbf{u}^l)^*$ will be applied to k and to the subset of $\underline{z}$ which is the domain of $\mathbf{u}^l$, however it is compatible with our notational conventions to display more variables than those actually occurring in a syntactic expression). □

Example 5. Consider the following formula ϕ (this example will be used for the proof obligation of Fig. 2 - variables are numbered so to make this application easier to recognize):

$$i' = i{+}1 \ \wedge \ i < N \ \wedge \ (x_1^{(2)})' = x_1^{(2)}{+}1 \ \wedge \ (x_2^{(2)})' = i{+}1 \ \wedge \ (x_1^{(3)})' = (x_1^{(2)})' \ \wedge$$
$$\wedge \ (x_2^{(3)})' = (x_2^{(2)})' \ \wedge \ (x_1^{(3)})' = (x_2^{(3)})' \ \wedge \ (x_1^{(4)})' = (x_1^{(3)})' \ \wedge \ (x_2^{(4)})' = (x_2^{(3)})'$$

To show that ϕ is iteratable, let us put

$$\underline{z} \ := \ i, x_1^{(2)}, \qquad \underline{w}' \ := \ (x_2^{(2)})', (x_1^{(3)})', (x_2^{(3)})', (x_1^{(4)})', (x_2^{(4)})' \ ;$$

an iterator $\mathbf{u}(\underline{z})$ such that $\phi \models \underline{z}' = \mathbf{u}(\underline{z})$ is given by

$$i' = i{+}1, \quad (x_1^{(2)})' = x_1^{(2)}{+}1 \ .$$

As a consequence, the formula $\phi^*(\underline{z}, \underline{z}', \underline{w}', j)$, according to (6), can be written as

$$i' = i{+}j \ \wedge \ (x_1^{(2)})' = x_1^{(2)}{+}j \ \wedge \ \forall k \ \in [0, j) \ \exists \underline{\tilde{w}} \ ((k = j - 1 \rightarrow \underline{\tilde{w}} = \underline{w}') \wedge \tilde{\phi}_k) \quad (7)$$

where $\tilde{\phi}_k$ (omitting the trivial literals $i + k + 1 = i + k + 1$ and $x_1^{(2)} + k + 1 = x_1^{(2)} + k + 1$) is the following formula

$$\begin{aligned} \tilde{\phi}_k \ \equiv \quad & i{+}k < N \ \wedge \ \tilde{x}_2^{(2)} = i{+}k{+}1 \ \wedge \ \tilde{x}_1^{(3)} = x_1^{(2)}{+}k{+}1 \ \wedge \\ & \wedge \ \tilde{x}_2^{(3)} = \tilde{x}_2^{(2)} \ \wedge \ \tilde{x}_1^{(3)} = \tilde{x}_2^{(3)} \ \wedge \ \tilde{x}_1^{(4)} = \tilde{x}_1^{(3)} \ \wedge \ \tilde{x}_2^{(4)} = \tilde{x}_2^{(3)} \end{aligned}$$

Notice that formula (6) introduces quantifiers; this does not matter because the underlying fragment of arithmetic we work with is assumed to be fully decidable. In practice, fragments used in verification - like difference logic and Presburger arithmetic - admit quantifier elimination and, if we eliminate quantifiers from (7), we can simplify it to

$$\begin{aligned} & i' = i{+}j \ \wedge \ (x_1^{(2)})' = x_1^{(2)}{+}j \ \wedge \ i + j < N \ \wedge \ (x_2^{(2)})' = i{+}j \ \wedge \ (x_1^{(3)})' = x_1^{(2)}{+}j \wedge \\ & \wedge \ (x_2^{(3)})' = i{+}j \ \wedge \ x_1^{(2)} = i \ \wedge \ (x_1^{(4)})' = x_1^{(2)}{+}j \ \wedge \ (x_2^{(4)})' = i{+}j \end{aligned} \quad (8)$$

Thus (8) represents the formula $\phi^*(\underline{z}, \underline{z}', \underline{w}', j)$, up to equivalence. □

4 Acceleration modules in satisfiability procedures

Let Γ be an acceleratable fragment; a Γ-*guard* is a formula of the kind

$$\forall i \ (\ i \in [t, u) \rightarrow \phi(i, \mathbf{a}[i], \mathbf{a}[i + 1]) \ ) \quad (9)$$

such that the formula $i' = i{+}1 \wedge \phi(i, \underline{y}, \underline{y}')$ belongs to Γ and t, u are ground terms (recall that we expanded the language with free constants, hence ground terms may contain them). Notice that, since Γ is closed under renaming substitutions, the choice of the tuple $i, \underline{y}, i', \underline{y}'$ is immaterial.

Theorem 3. *Let Γ be an acceleratable fragment; then, any Boolean combination of ground formulae and Γ-guards is decidable for satisfiability.*

Proof. Since the negation of a Γ-guard can be converted into a ground formula by Skolemization, it is sufficient to check the satisfiability of a conjunction

$$L_1 \wedge \cdots \wedge L_n \wedge G_1 \wedge \cdots \wedge G_m \tag{10}$$

of ground literals and Γ-guards. We design a satisfiability algorithm below.

Step I [Guess an ordering]. Let S be the set of *ground* terms occurring in (10);[7] guess a partition on S and an ordering $C_1 < \cdots < C_l$ of the equivalence classes. For each equivalence class C_i, introduce a fresh constant c_i; then add to the current formula the literals of the form $c_i = t$ (varying $t \in C_i$) and of the form $c_i < c_{i+1}$.

Step II [Cleaning] We call a constant c_h an *out-of-bound* constant in case h is bigger (resp. smaller) than the index of the constant corresponding to the equivalence class of N (resp. of 0); a term t is out-of-bound iff the free constant c_k representing the equivalence class of t is out-of-bound. Dereference terms $a_j[t]$, where t is out-of-bound, are replaced by the conventional value 0. Guards whose antecedent is of the kind $i \in [t, u)$, where t is out-of-bound or u is out-of-bound or u is in the same equivalence class as N, are removed (by our conventions from Section 2, these guards are tautological, having an inconsistent antecedent). Similarly, guards whose antecedent is of the kind $i \in [c_h, c_k)$ for $h \geq k$ are removed too.

Step III [Grouping the guards]. In this step, we rewrite guards. The new guards will be of the kind

$$\forall i\, (i \in [c_k, c_{k+1}) \rightarrow \psi_k(i, \mathbf{a}[i], \mathbf{a}[i+1]))$$

where $k+1$ is less or equal to the index of the constant corresponding to the equivalence class of N and k is bigger or equal to the index of the constant corresponding to the equivalence class of 0. The formula ψ_k is obtained by taking the conjunction of the relevant consequents of the guards from (10). In other words, if (10) contains the guard $\forall i\, (i \in [t, u) \rightarrow \phi(i, \mathbf{a}[i], \mathbf{a}[i+1]))$ and the equivalence class of c_k follows the equivalence class of t and the equivalence class of u follows the equivalence class of c_{k+1}, then ϕ is included among the conjuncts of ψ_k. Since acceleratable fragments are closed under conjunctions, the new guards we obtain are still Γ-guards.

Step IV [Reduction to Pure Arithmetic]. Let

$$G \wedge \bigwedge_{k=1}^{l} \forall i\, (i \in [c_k, c_{k+1}) \rightarrow \psi_k(i, \mathbf{a}[i], \mathbf{a}[i+1])) \tag{11}$$

[7] We must include $0, N$ among such terms; however, to economize the guessing step, besides $0, N$, we can limit ourselves to terms t occurring in sub-expression of the form $a[t], i \in [t, u), i \in [u, t)$.

be the formula we obtain after Step III. Here G is a conjunction of ground literals (including the literals added in Step I) and the quantified formulae are Γ-guards. By the definition of a Γ-guard, the formulæ

$$\phi_k(i, i', \underline{y}, \underline{y}') \ :\equiv \ \ i' = i + 1 \wedge \psi_k(i, \underline{y}, \underline{y}') \tag{12}$$

are in Γ. We now replace (11) by the formula

$$G(\underline{d}_1/\mathbf{a}[c_1], \ldots, \underline{d}_l/\mathbf{a}[c_l]) \wedge \bigwedge_{k=1}^{l} \phi_k^*(c_k, c_{k+1}, \underline{d}_k, \underline{d}_{k+1}, c_{k+1} - c_k) \tag{13}$$

where $\underline{d}_1, \ldots, \underline{d}_l$ are tuples of fresh constants and $G(\underline{d}_1/\mathbf{a}[c_1], \ldots, \underline{d}_l/\mathbf{a}[c_l])$ is obtained from G by replacing component-wise, for each $k = 1, \ldots, l$ and for each t lying in the same equivalence class as c_k, the tuple $\mathbf{a}[t]$ by the tuple $\underline{d}_k$.

We claim that the formulæ (13) are equi-satisfiable with the original formula (10). Clearly, it is sufficient to show that (13) is equi-satisfiable to (11). Suppose that (13) is satisfiable: this means that we can assign integer numbers to the free constants occurring in (13), so to make the statement (13) true. We use the same letters to denote a free constant, the number assigned to it and the corresponding numeral. From the fact that $\phi_k^*(c_k, c_{k+1}, \underline{d}_k, \underline{d}_{k+1}, c_{k+1} - c_k)$ is true (for the given choice of the $c_k, c_{k+1}, \underline{d}_k, \underline{d}_{k+1}$), we can infer that $\phi_k^{c_{k+1}-c_k}(c_k, c_{k+1}, \underline{d}_k, \underline{d}_{k+1})$ holds by Definition 1. By (12) and the definition of relation composition, we get tuples $\underline{d}_k := \underline{d}_{c_k}, \underline{d}_{c_k+1}, \underline{d}_{c_k+2} \ldots, \underline{d}_{c_k+(c_{k+1}-c_k)} := \underline{d}_{k+1}$ such that

$$\psi_k(c_k, \underline{d}_{c_k}, \underline{d}_{c_k+1}), \ \psi_k(c_k + 1, \underline{d}_{c_k+1}, \underline{d}_{c_k+2}), \ \ldots, \ \psi_k(c_{k+1} - 1, \underline{d}_{c_{k+1}-1}, \underline{d}_{c_{k+1}})$$

all hold. Thus, we define the interpretations of the unary integer functions $\mathbf{a}$, by letting $\mathbf{a}(n) := 0$ for $n > N$ and $n < 0$ and for $c_k \leq n \leq c_{k+1}$ by taking $\mathbf{a}(n)$ to be $\underline{d}_n$. Formula (11) holds by construction. Similar considerations, read in the opposite sense, show that the satisfiability of (11) implies the satisfiability of (13). □

Remark 1 (Complexity). Since this is a modular procedure, its complexity can only be evaluated relatively to the complexities of the acceleration module and of the underlying arithmetic solver. To this aim, notice that Steps I-II introduce a linear guessing followed by linear manipulations and that Step III produces a quadratically long formula (11). After these steps, the complexity relies entirely on the complexity of the acceleration module and on the complexity of the arithmetic solver: if we suppose that the former requires space $f_S(n)$ and time $f_T(n)$ to produce the accelerated formula (13) out of (11) and that the latter requires space $g_S(m)$ and time $g_T(m)$ for its satisfiability checks, the cost of the whole procedure requires space bounded by $g_S(f_S(O(n^2)))$ and time bounded by $2^{O(n^2)} \cdot g_T(f_T(O(n^2)))$ (we need exponential time to go through all possible linear orderings).

We underline that in examples coming from practical verification problems, the expensive guess of Step I is not needed, because the few consistent guessings are suggested by the problem itself, as witnessed by the example below.

Example 6. We consider the proof obligation of Fig. 2. We first need to rewrite all universally quantified guards in it in such a way that they match the pattern given by (9). Thus, sub-formulae like $\forall i\ (t \leq i \leq u \ \rightarrow\ \gamma(i, \mathbf{a}[i]))$ must be rewritten as $(t \leq u \rightarrow \gamma(t, \mathbf{a}[t])) \wedge \forall i\ (i \in [t, u) \ \rightarrow\ \gamma(i+1, \mathbf{a}[i+1]))$; similarly, array equations of the form $\mathbf{a} = \mathbf{b}$ are rewritten to $\mathbf{a}[0] = \mathbf{b}[0] \wedge \forall i\ (i \in [0, N) \rightarrow \mathbf{a}[i{+}1] = \mathbf{b}[i{+}1])$. After these rewritings, we can observe that all guards occurring in Fig. 2 are Γ-guards, where Γ is the acceleratable fragment of Proposition 1 (one may equivalently use the fragment of Theorem 1 instead). To see this, let us abstract out $a_1^{(k)}[i], a_1^{(k)}[i+1]$ with $x_1^{(k)}, (x_1^{(k)})'$ and $a_2^{(k)}[i], a_2^{(k)}[i+1]$ with $x_2^{(k)}, (x_2^{(k)})'$ $(k = 1, \ldots, 4)$. Then, let us consider for instance the first guard of (α_2): the formula to be checked to belong to the acceleratable fragment is

$$i' = i + 1 \wedge i < N \wedge (x_1^{(2)})' = x_1^{(2)} + 1 \wedge (x_2^{(2)})' = i + 1$$

and it is clear that this formula fits Proposition 1 (and Theorem 1 too). Thus, we can run the algorithm of Theorem 3 to check the unsatisfiability of the conjunction of the formulæ $(\alpha_1) - (\alpha_5)$. As for Step I, consider the partition

$$\{0, \mathtt{I}^{(1)}, \mathtt{I}^{(3)}\} \ < \ \{\mathtt{I}^{(3)}{+}m, \mathtt{I}^{(4)}\} \ < \ \{\mathtt{I}^{(4)}{+}1\} \ < \ \{N, \mathtt{I}^{(2)}, \mathtt{I}^{(1)}{+}n\} \ < \ \{\mathtt{I}^{(1)}{+}n{+}1\}$$

(other partitions are either analogous to this one or do not admit a consistent ordering). We call c_1, c_2, c_3, c_4, c_5, respectively, the fresh constants denoting a generic element of the above classes of the partition (notice that c_5 is out-of-bound). Step II eliminates the second guard from (α_2). Step III produces a formula which is the conjunction of the ground literals from Fig. 3 with three Γ-guards $\gamma_{12}, \gamma_{23}, \gamma_{34}$ (relative to the intervals $[c_1, c_2), [c_2, c_3), [c_3, c_4)$, respectively), also displayed in Fig. 3.

Going to Step IV, we now consider the formula (13); the acceleration formulæ replacing the Γ-guards $\gamma_{12}, \gamma_{23}, \gamma_{34}$ can be drawn from Example 5 (strictly speaking, Example 5 analyzes only γ_{12}, but the other two Γ-guards are analyzed in the same way). Thus formula (13) becomes equivalent to the conjunction of the literals from Fig. 3 together with the additional literals from Fig. 4. To improve readability, in Fig. 4 we do not replace terms $\mathbf{a}[t]$ with fresh constants depending on the equivalence class of t like in (13) (as a consequence, we shall need below congruence closure besides arithmetic to check inconsistency). To conclude the unsatisfiability test of the proof obligation from Fig. 2 it is then sufficient to observe that the following unsatisfiable subset can be extracted from the literals in Fig. 3-4:

$$\begin{array}{lll}
\mathtt{a}_1^{(4)}[\mathtt{I}^{(4)}{+}1] \neq \mathtt{a}_2^{(4)}[\mathtt{I}^{(4)}{+}1], & c_3 = \mathtt{I}^{(4)}{+}1, & c_1 = 0, \\
\mathtt{a}_1^{(4)}[c_3] = \mathtt{a}_1^{(2)}[c_2]{+}(c_3{-}c_2), & c_1 = \mathtt{I}^{(1)}, & \mathtt{a}_1^{(2)}[0] = \mathtt{a}_1^{(1)}[0], \\
\mathtt{a}_1^{(2)}[c_2] = \mathtt{a}_1^{(2)}[c_1]{+}(c_2{-}c_1), & \mathtt{a}_1^{(1)}[\mathtt{I}^{(1)}] = 0, & \mathtt{a}_2^{(4)}[c_3] = c_2{+}(c_3{-}c_2). \ \square
\end{array}$$

The decidable class covered by Theorem 3 includes some remarkable classes known to be decidable from the literature: in particular, it covers the SIL-fragments of [14] and the flat mono-sorted fragments of [4]. We point out, however, that some other known decidable classes are still orthogonal to the classes

Literals:

$$\mathtt{I}^{(1)} = 0,\ \mathtt{a}_1^{(1)}[\mathtt{I}^{(1)}] = 0,\ \mathtt{a}_2^{(1)}[\mathtt{I}^{(1)}] = 0,\ \mathtt{a}_1^{(2)}[0] = \mathtt{a}_1^{(1)}[0],\ \mathtt{a}_2^{(2)}[0] = \mathtt{a}_2^{(1)}[0],$$
$$\mathtt{I}^{(2)} = \mathtt{I}^{(1)} + n,\ \mathtt{I}^{(2)} \geq N,\ \mathtt{I}^{(3)} = 0,\ \mathtt{a}_1^{(3)}[0] = \mathtt{a}_1^{(2)}[0],\ \mathtt{a}_2^{(3)}[0] = \mathtt{a}_2^{(2)}[0],$$
$$\mathtt{I}^{(4)} = \mathtt{I}^{(3)} + m,\ \mathtt{a}_1^{(4)}[0] = \mathtt{a}_1^{(3)}[0],\ \mathtt{a}_2^{(4)}[0] = \mathtt{a}_2^{(3)}[0],\ \mathtt{a}_1^{(4)}[\mathtt{I}^{(4)}+1] \neq \mathtt{a}_2^{(4)}[\mathtt{I}^{(4)}+1],$$
$$c_1 = 0,\ c_1 = \mathtt{I}^{(1)},\ c_1 = \mathtt{I}^{(3)},\ c_1 < c_2,\ c_2 = \mathtt{I}^{(3)} + m,\ c_2 = \mathtt{I}^{(4)},\ c_2 < c_3,\ c_3 = \mathtt{I}^{(4)}+1,$$
$$c_3 < c_4,\ c_4 = N,\ c_4 = \mathtt{I}^{(2)},\ c_4 = \mathtt{I}^{(1)} + n,\ c_4 < c_5,\ c_5 = \mathtt{I}^{(1)} + n + 1\ .$$

Guards:

$$\gamma_{12} \equiv \quad \forall i \in [c_1, c_2) \begin{pmatrix} i < N \wedge\ \mathtt{a}_1^{(2)}[i{+}1] = \mathtt{a}_1^{(2)}[i] + 1\ \wedge\ \mathtt{a}_2^{(2)}[i{+}1] = i{+}1\ \wedge \\ \wedge\ \mathtt{a}_1^{(3)}[i{+}1] = \mathtt{a}_1^{(2)}[i{+}1]\ \wedge\ \mathtt{a}_2^{(3)}[i{+}1] = \mathtt{a}_2^{(2)}[i{+}1] \wedge \\ \wedge\ \mathtt{a}_1^{(3)}[i+1] = \mathtt{a}_2^{(3)}[i+1] \wedge \\ \wedge\ \mathtt{a}_1^{(4)}[i{+}1] = \mathtt{a}_1^{(3)}[i{+}1]\ \wedge\ \mathtt{a}_2^{(4)}[i{+}1] = \mathtt{a}_2^{(3)}[i{+}1] \end{pmatrix}$$

$$\gamma_{23} \equiv \quad \forall i \in [c_2, c_3) \begin{pmatrix} i < N \wedge\ \mathtt{a}_1^{(2)}[i{+}1] = \mathtt{a}_1^{(2)}[i] + 1\ \wedge\ \mathtt{a}_2^{(2)}[i{+}1] = i{+}1\ \wedge \\ \wedge\ \mathtt{a}_1^{(3)}[i{+}1] = \mathtt{a}_1^{(2)}[i{+}1]\ \wedge\ \mathtt{a}_2^{(3)}[i{+}1] = \mathtt{a}_2^{(2)}[i{+}1]\ \wedge \\ \wedge\ \mathtt{a}_1^{(4)}[i{+}1] = \mathtt{a}_1^{(3)}[i{+}1]\ \wedge\ \mathtt{a}_2^{(4)}[i{+}1] = \mathtt{a}_2^{(3)}[i{+}1] \end{pmatrix}$$

$$\gamma_{34} \equiv \quad \forall i \in [c_3, c_4) \begin{pmatrix} i < N \wedge\ \mathtt{a}_1^{(2)}[i{+}1] = \mathtt{a}_1^{(2)}[i] + 1\ \wedge\ \mathtt{a}_2^{(2)}[i{+}1] = i{+}1\ \wedge \\ \wedge\ \mathtt{a}_2^{(3)}[i{+}1] = \mathtt{a}_2^{(2)}[i{+}1]\ \wedge\ \mathtt{a}_1^{(3)}[i{+}1] = \mathtt{a}_1^{(2)}[i{+}1]\ \wedge \\ \wedge\ \mathtt{a}_1^{(4)}[i{+}1] = \mathtt{a}_1^{(3)}[i{+}1]\ \wedge\ \mathtt{a}_2^{(4)}[i{+}1] = \mathtt{a}_2^{(3)}[i{+}1] \end{pmatrix}$$

Fig. 3: Literals and Guards after Step III (see Example 6).

presented in this paper. Since a comprehensive comparison is rather technical and require more space, we defer it to Appendix A.

5 Applications to imperative programs

In this section we show how to use our results in order to establish decidability of safety problems for a class of imperative programs handling arrays. We will consider programs with *flat* control-flow graph with loops represented by acceleratable formulæ. This section provides just initial assesments: future more extensive work may comprise the exploitation of generalized notions like iterators/selectors [2] and the adoption of compiler-oriented optimization features [1, 13] which lie outside the scope of this work.

Henceforth $\mathbf{v}$ will denote the variables of the programs we analyze. Formally, $\mathbf{v} = \mathbf{a}, I$ where, according to our conventions, $\mathbf{a}$ is a tuple of array variables (modeled as free unary function symbols in our framework) and I is an integer variable to be used as a counter to scan arrays (we omit further integer variables for simplicity, but see Remark 2 below). As stated in Section 2, we work in a

Literals from γ_{12}:

$c_2 = c_1 + (c_2 - c_1)$, $\mathtt{a}_1^{(2)}[c_2] = \mathtt{a}_1^{(2)}[c_1] + (c_2 - c_1)$, $\mathtt{a}_2^{(2)}[c_2] = c_1 + (c_2 - c_1)$,
$\mathtt{a}_1^{(3)}[c_2] = \mathtt{a}_1^{(2)}[c_1] + (c_2 - c_1)$, $\mathtt{a}_1^{(4)}[c_2] = \mathtt{a}_1^{(2)}[c_1] + (c_2 - c_1)$, $c_1 + (c_2 - c_1) < N$,
$\mathtt{a}_2^{(3)}[c_2] = c_1 + (c_2 - c_1)$, $\mathtt{a}_2^{(4)}[c_2] = c_1 + (c_2 - c_1)$, $\mathtt{a}_1^{(2)}[c_1] = c_1$.

Literals from γ_{23}:

$c_3 = c_2 + (c_3 - c_2)$, $\mathtt{a}_1^{(2)}[c_3] = \mathtt{a}_1^{(2)}[c_2] + (c_3 - c_2)$, $\mathtt{a}_2^{(2)}[c_3] = c_2 + (c_3 - c_2)$,
$\mathtt{a}_1^{(3)}[c_3] = \mathtt{a}_1^{(2)}[c_2] + (c_3 - c_2)$, $\mathtt{a}_1^{(4)}[c_3] = \mathtt{a}_1^{(2)}[c_2] + (c_3 - c_2)$, $c_2 + (c_3 - c_2) < N$,
$\mathtt{a}_2^{(3)}[c_3] = c_2 + (c_3 - c_2)$, $\mathtt{a}_2^{(4)}[c_3] = c_2 + (c_3 - c_2)$.

Literals from γ_{34}:

$c_4 = c_3 + (c_4 - c_3)$, $\mathtt{a}_1^{(2)}[c_4] = \mathtt{a}_1^{(2)}[c_3] + (c_4 - c_3)$, $\mathtt{a}_2^{(2)}[c_4] = c_3 + (c_4 - c_3)$,
$\mathtt{a}_1^{(3)}[c_4] = \mathtt{a}_1^{(2)}[c_3] + (c_4 - c_3)$, $\mathtt{a}_1^{(4)}[c_4] = \mathtt{a}_1^{(2)}[c_3] + (c_4 - c_3)$, $c_3 + (c_4 - c_3) < N$,
$\mathtt{a}_2^{(3)}[c_4] = c_3 + (c_4 - c_3)$, $\mathtt{a}_2^{(4)}[c_4] = c_3 + (c_4 - c_3)$.

Fig. 4: (Step IV) Literals whose conjunction is the formula $\bigwedge_{k=1}^{3} \phi_k^*$ from (13) (see Example 6).

decidable fragment of arithmetic, extended with free constants and free unary function symbols. A *state-formula* is a formula $\alpha(\mathbf{v})$ representing a (possibly infinite) set of configurations of the program under analysis. A *transition formula* is a formula of the kind $\tau(\mathbf{v}, \overline{\mathbf{v}})$ where $\overline{\mathbf{v}}$ is a renaming of the tuple $\mathbf{v}$ (we prefer not to use here the standard model-checking notation $\mathbf{v}'$ for $\overline{\mathbf{v}}$, because we already used the primed notation in the previous sections in a different context).

Definition 5 (Programs). *Given a set of variables* $\mathbf{v}$*, a* program *is a triple* $\mathcal{P} = (L, \Lambda, E)$*, where (i)* $L = \{l_1, \ldots, l_n\}$ *is a set of* program locations *among which we distinguish an initial location* l_{init} *and an error location* l_{error}*; (ii)* Λ *is a finite set of transition formulæ* $\{\tau_1(\mathbf{v}, \overline{\mathbf{v}}), \ldots, \tau_r(\mathbf{v}, \overline{\mathbf{v}})\}$ *and (iii)* $E \subseteq L \times \Lambda \times L$ *is a set of* actions.

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

Definition 6 (Program paths). *A* program path *(in short,* path*) of* $\mathcal{P} = (L, \Lambda, E)$ *is a sequence* $\rho \in E^n$*, i.e.,* $\rho = e_1, e_2, \ldots, e_n$*, such that for every* e_i, e_{i+1}*, we have* $tgt(e_i) = src(e_{i+1})$*. We denote with* $|\rho|$ *the length of the path. An* error path *is a path* ρ *with* $src(e_1) = l_{\mathsf{init}}$ *and* $tgt(e_{|\rho|}) = l_{\mathsf{error}}$*. A path* ρ *is a* feasible path *if* $\bigwedge_{j=1}^{|\rho|} \mathcal{L}(e_j)^{(j)}$ *is satisfiable, where* $\mathcal{L}(e_j)^{(j)}$ *represents*

$\tau_{i_j}(\mathbf{v}^{(j-1)}, \mathbf{v}^{(j)})$, *with* $\mathcal{L}(e_j) = \tau_{i_j}$. *The* (unbounded) reachability problem *for a program* $\mathcal{P}$ *is to detect if* $\mathcal{P}$ *admits a feasible error path.*

One word about the notation $\tau_{i_j}(\mathbf{v}^{(j-1)}, \mathbf{v}^{(j)})$ used above: when we use tuples of variables like $\mathbf{v}^{(j)}$, we mean that we simultaneously employ many disjointed renamed copies (written $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{v}^{(3)}, \dots$) of the tuple $\mathbf{v}$. Obviously, $\tau_{i_j}(\mathbf{v}^{(j-1)}, \mathbf{v}^{(j)})$ indicates the formula obtained from $\tau_{i_j}(\mathbf{v}, \overline{\mathbf{v}})$ by replacing $\mathbf{v}$ by $\mathbf{v}^{(j-1)}$ and $\overline{\mathbf{v}}$ by $\mathbf{v}^{(j)}$.

We first give the definition of a flat^0-program, i.e. of a program with only self-loops for which each location belongs to at most one loop.

Definition 7 (flat^0-program). *A program* $\mathcal{P}$ *is a* flat^0-program *if for every path* $\rho = e_1, \dots, e_n$ *of* $\mathcal{P}$ *it holds that for every* $j < k$ *(*$j, k \in \{1, \dots, n\}$*), if* $src(e_j) = tgt(e_k)$ *then* $e_j = e_{j+1} = \cdots = e_k$.

We shall consider below only programs whose transitions are of two kinds:

(i) *quantifier-free* formulae $\tau(\mathbf{v}, \overline{\mathbf{v}})$: these formulæ can be used only as labels for actions which are not self-loops (i.e. whose source and target locations do not coincide);
(ii) transitions used as labels in *self-loops*: these transitions must be of the following kind

$$(\forall i \neq I{+}1\ \mathbf{a}[i] = \overline{\mathbf{a}}[i]) \ \wedge\ \gamma(I, \mathbf{a}[I], \mathbf{a}[I+1], \overline{\mathbf{a}}[I+1]) \ \wedge\ \overline{I} = I{+}1 \qquad (14)$$

where γ is quantifier-free and arithmetical over $\mathbf{a}[I], \mathbf{a}[I+1], \overline{\mathbf{a}}[I+1]$.

Formula (14) says that the loop modifies just the entry $I{+}1$ of each array $\mathbf{a}$. It is general enough to include instructions of the following kind

$$\texttt{while}(\delta(I, \mathbf{a}[I], \mathbf{a}[I+1]))\{\ \ \mathbf{a}[I+1] := \mathbf{t}(I, \mathbf{a}[I], \mathbf{a}[I+1]);\ \ I{+}{+};\ \ \}$$

where δ is a guard expressed via a quantifier-free formula arithmetical over $\mathbf{a}[I], \mathbf{a}[I+1]$ and where the terms $\mathbf{t}$ are also arithmetical over $\mathbf{a}[I], \mathbf{a}[I+1]$.

Remark 2. Additional integer variables can be modeled as arrays as follows. Suppose we have an integer variable C and that inside the loop we want to update it as $C := \mathbf{u}(I, C, \mathbf{a}[I], \mathbf{a}[I+1])$. Then we can introduce a fresh array variable c; this variable is (partially) initialized as $c[I] := C$ before the loop, it is returned as $C := c[I]$ after the loop and it is updated inside the loop as $c[I+1] := \mathbf{u}(I, c[I], \mathbf{a}[I], \mathbf{a}[I+1])$.

Assumption. *We assume from now on that our programs are* flat^0*-programs and that their transitions are subject to the above restrictions* (i) *and* (ii).

This assumption is not yet sufficient for decidability, though. To gain decidability, we put further conditions on guards and updates. Let us consider the list of variables $I, \mathbf{e}, \overline{\mathbf{e}}, I', \mathbf{e}', \overline{\mathbf{e}}'$ where the variables $\mathbf{e}, \overline{\mathbf{e}}$ are meant to abstract

out $\mathbf{a}[I], \overline{\mathbf{a}}[I]$ and the variables $\mathbf{e}', \overline{\mathbf{e}}'$ to abstract out $\mathbf{a}[I+1], \overline{\mathbf{a}}[I+1]$.We call *arithmetic projections* of $\mathcal{P}$ the formulæ

$$I' = I + 1 \ \wedge \ \gamma(I, \overline{\mathbf{e}}, \mathbf{e}', \overline{\mathbf{e}}') \tag{15}$$

extracted from the self-loops instructions (14) occurring in $\mathcal{P}$. We give some sufficient practical (relatively simple) conditions so that the simultaneous acceleration of the formulæ (14) occurring in a path of $\mathcal{P}$ meets the hypothesis of Theorem 3. One needs to pay attention to the fact that the update of $\mathbf{a}[I+1]$ is recursive; this is why the variables $\overline{\mathbf{e}}$ abstracting out $\overline{\mathbf{a}}[I]$ have been preferred to[8] the $\mathbf{e}$ (abstracting out $\mathbf{a}[I]$) when defining arithmetic projections.

Theorem 4. *The unbounded reachability problem for $\mathcal{P}$ is decidable if there is a normal acceleratable fragment containing all arithmetic projections of $\mathcal{P}$.*

Proof. (Sketch, see Appendix B for details). For a transition relation $\tau(\mathbf{v}, \overline{\mathbf{v}})$ given by (14), the transition $\tau^*(\mathbf{v}, \overline{\mathbf{v}}, \bar{n})$ expressing the n-times composition of τ with itself is given by:

$$\begin{pmatrix} \forall i \in [I, I+n) & \gamma(i, \overline{\mathbf{a}}[i], \mathbf{a}[i+1], \overline{\mathbf{a}}[i+1]) & \wedge \\ \forall i \in [0, I+1) & \overline{\mathbf{a}}[i] = \mathbf{a}[i] & \wedge \\ \forall i \in [I+n+1, N+1) & \overline{\mathbf{a}}[i] = \mathbf{a}[i] & \wedge \\ \overline{I} = I + n & & \end{pmatrix} \tag{16}$$

Let now Γ be a normal acceleratable fragment containing all arithmetic projections of $\mathcal{P}$: the key observation is that (16) (after little rewriting) *is a conjunction of ground literals and Γ-guards*. This allows to check the satisfiability of all formulæ expressing the feasibility of an error path. □

Example 7. We apply the procedures of Theorem 4 to the example of Fig. 1. The relevant error path comprises the execution of the instruction in location 1, n executions of the loop in location 2, the exit condition from this loop together with execution of the instruction in location 3, m executions of the loop in location 4 and the error exit condition from that loop. If we apply formulæ (16) for acceleration, we get the proof obligation of Fig. 2 (with little simplifications improving readability). Example 6 shows that the conjunction of the formulæ from Fig. 2 is inconsistent, hence the program of Fig. 1 is safe. □

6 Conclusions and future work

In this paper we presented a new framework for deciding the satisfiability of quantified formulæ with arrays. Such framework allows for the integration of acceleration results satisfying the conditions identified in Definitions 1-2, and exploits them as black-box modules, as described by the algorithm of Theorem 3.

[8] Notice that $\mathbf{a}[I] = \overline{\mathbf{a}}[I]$ is nevertheless a logical consequence of (14).

The framework can also be applied in a software model-checking scenario, where it can be proven that the safety of a new class of programs with arrays can be decided by integrating our new results with acceleration procedures.

On the practical side, in our experience [2–4], the tools get remarkable benefits from acceleration/decidability results, both whenever the results are used directly in decisions procedures like that of Theorem 4 and when they are used indirectly, via abstraction and instantiation, like in [2]. Implementing the results of this work is an interesting and substantial future project which we intent to pursue building upon our tools Booster [3] and mcmt [12].

References

1. A.V. Aho, M.S. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Educational Publishers, Incorporated, 2007.
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. F. Alberti, S. Ghilardi, and N. Sharygina. Booster : an acceleration-based verification framework for array programs. In *ATVA*, pages 18–23, 2014.
4. F. Alberti, S. Ghilardi, and N. Sharygina. Decision procedures for flat array properties. In *TACAS*, pages 15–30, 2014.
5. B. Boigelot. On iterating linear transformations over recognizable sets of integers. *Theor. Comput. Sci.*, 309(1):413–468, December 2003.
6. M. Bozga, C. Girlea, and R. Iosif. Iterating octagons. In *TACAS*, LNCS, pages 337–351, 2009.
7. M. Bozga, R. Iosif, and F. Konecny. Fast acceleration of ultimately periodic relations. In *CAV*, LNCS, 2010.
8. M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, (91):275–303, 2009.
9. A.R. Bradley, Z. Manna, and H.B. Sipma. What's decidable about arrays? In *VMCAI*, pages 427–442, 2006.
10. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV*, volume 1427 of *LNCS*, pages 268–279. Springer, 1998.
11. A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *FST TCS 02*, pages 145–156. Springer, 2002.
12. S. Ghilardi and S. Ranise. MCMT: A Model Checker Modulo Theories. In *IJCAR*, pages 22–29, 2010.
13. A. Gurfinkel, S. Chaki, and S. Sapra. Efficient predicate abstraction of program summaries. In *NASA Formal Methods - NFM*, pages 131–145, 2011.
14. P. Habermehl, R. Iosif, and T. Vojnar. A logic of singly indexed arrays. In *LPAR*, pages 558–573, 2008.
15. J.Y. Halpern. Presburger arithmetic with unary predicates is Π^1_1 complete. *J. Symbolic Logic*, 56(2):637–642, 1991.
16. A.L. Semënov. Logical theories of one-place functions on the set of natural numbers. *Izvestiya: Mathematics*, 22:587–618, 1984.
17. J.R. Shoenfield. *Mathematical logic*. Association for Symbolic Logic, Urbana, IL, 2001. Reprint of the 1973 second printing.

A Array Fragments: a Comparison

In this section we make a comparison between different quantified fragments known to be decidable from the literature (including the fragment considered in this paper). They all are $\exists^*\forall$-fragments. The comparison below is made in the settings of the present paper where arrays are interpreted on intervals $[0, N]$: in other words, arrays are modeled as free unary function symbols taking a conventional value, say 0, outside this interval (the condition $\forall i(i < 0 \vee i > N \rightarrow a[i] = 0)$ is assumed or can be expressed in all fragments below).

Fragment from [4]. In [4], two classes of array properties are shown to be decidable. The second class (the two-sorted monic-flat fragment) is orthogonal to the class covered by Theorem 3; the (lighter, but still useful) first class consists of the flat formulæ whose quantified prefix is $\exists^*\forall$ (a formula is *flat* iff for every term of the kind $a[t]$ occurring in it, the sub-term t is a variable). If we skolemize existential quantifiers with free constants, abstract out ground terms with fresh constants, put the matrix in normal form and distribute the universal quantifier, from flat $\exists^*\forall$-formulæ we get formulæ that are equivalent to conjunctions of ground literals and of guards of the kind

$$\forall i\, (i \in [0, N{+}1) \rightarrow \phi(i, \mathbf{a}[i])$$

where ϕ is purely arithmetical. If we rewrite this as

$$\phi(0, \mathbf{a}[0]) \ \wedge \ \forall i\, (i \in [0, N) \rightarrow \phi(i{+}1, \mathbf{a}[i{+}1])$$

we clearly get conjunctions of ground formulæ and Γ-guards, where Γ is the acceleratable fragment of Proposition 1. □

Fragment from [9]. In [9], the following class is shown to be decidable: the class consists on formulæ which are equivalent to disjunctions of formulæ of the kind $\exists \underline{k}\ (\psi_1(\underline{k}, \mathbf{a}) \wedge \cdots \wedge \psi_n(\underline{k}, \mathbf{a}))$, where each ψ's is either a literal or a BM-guard. In turn, a BM-guard is a formula of the kind

$$\forall \underline{i}\ (G \rightarrow \theta) \tag{17}$$

where (i) G is a conjunction of atoms of the kind $i_1 \leq i_2, i_1 \leq t(\underline{k}), t(\underline{k}) \leq i_1$ for $i_1, i_2 \in \underline{i}$ and $t(\underline{k})$ arithmetical; (ii) θ is obtained from a quantifier-free arithmetical formula $\alpha(\underline{k}, \underline{x})$ by replacing the variables $\underline{x}$ by terms of the kind $a[t(\underline{k})], a[i]$ for $i \in \underline{i}, t(\underline{k})$ arithmetical and $a \in \mathbf{a}$.

To make a comparison, we must concentrate on BM-guards. Notice that we can treat the $\underline{k}$ in (17) as free constants. These guards are orthogonal to the Γ-guards, where Γ is the acceleratable fragment of Proposition 1: on one side, in fact, the consequent of θ has limitations that do not occur in the definition of Γ-guards (in θ, the quantified variables $\underline{i}$ must occur only as a read of an array) and on the other side the guards (17) have more than one quantified variable.[9]

[9] In addition, [9] considers also many-dimensional arrays.

To understand why a conjunction ϕ of ground literals and of BM-guards is decidable for satisfiability, it is useful to make a preprocessing interval guessing partitions $[c_1, c_2), \ldots, [c_{l-1}, c_l)$ like in Step 1 of the algorithm from Theorem 3. Having this partition, it is then not difficult to see that if our ϕ it is satisfiable, then it is satisfiable in a model where the array variables $\mathbf{a}$ are interpreted on functions which are *constants on each interval*: this is because in the consequents θ of the BM-guards (17) each $i \in \underline{i}$ occurs only as $\mathbf{a}[i]$ (i.e. inside a read) and so we may assume that the $\mathbf{a}[i]$ are equal to the values $\mathbf{a}[c_j]$ where c_j is the left bound of the interval i belongs to. This fact is exploited in the completeness proof of [9] and justifies a decision procedure by instantation: to check satisfiability of ϕ it is sufficient to check satisfiability of the formula obtained from ϕ by instantiating all universal quantifiers on interval bounds in all possible ways.

Notice that the decision procedure drawn from the combination of Proposition 1 and Theorem 3 (and the decision procedure from from [4] too) is not based on instantiation, but on quantifier handling in pure arithmetic. For the lack of a better method, handling quantifiers in Presburger arithmetic requires a quantifier elimination step, which might be harmful for complexity. This quantifier elimination step is the price paid for allowing the quantified variable i to occur outside the read of arrays in the consequents of the guards (in particular, it is the price paid for allowing genuinely 'mono-sorted' atoms like $a[i] = i,\ a[i] < i,\ a[i] \neq i + 2$, etc. in such consequents). □

Fragment from [14]. In [14] another interesting decidable fragment is introduced. The peculiarity of this fragment (making it orthogonal to all previously known fragments) is that of allowing both sub-terms of the kind $a[i]$ and of the kind $a[i+1]$, where i is the (unique) universally quantified variable that can be used. This liberality is compensated by other restrictions, on the arithmetic expressions and on the use of disjunctions. The syntax of the so-called SIL-fragment introduced in [14] is rather elaborated, however there is a normalized equivalent formulation the authors themselves use after a preprocessing step (Lemma 6, in Section 5.1 of [14]).

The normalized formulation covers Boolean combinations of ground formulæ and of guards of the following two types

$$\forall i\ (i \in [t, u) \to \nu) \tag{18}$$

$$\forall i\ (i \in [t, u) \wedge i \equiv_n \overline{m} \to \nu) \tag{19}$$

where: (i) t, u are ground arithmetical terms; (ii) $m, n \in \mathbb{N}$; (iii) ν is a *conjunction* of literals of the kinds

$$a_1[i] \sim \ell, \quad a_1[i] - i \sim \bar{n}, \quad a_1[i] - a_2[i+1] \sim \bar{n}$$

for $\sim \in \{\leq, \geq\}$, ℓ ground arithmetical, a_1, a_2 array variables, $n \in \mathbb{Z}$.

Clearly, the guards of the form (18) are Γ-guards, where Γ is the accceleratable fragment of Theorem 1.[10] The guards of the form (19) can be transformed

[10] The atom involving the term ℓ can be replaced, up to satisfiability, by the literal $a_1[i] \sim a_\ell[i]$, where a_ℓ is an extra array variable subject to the constraints $a_\ell[0] =$

into guards of the form (18), by some straightforward (but tedious) procedure. We show the procedure by an example.

Suppose that in (19) we have $n = 3$ and $m = 0$; in addition, we need a preliminary guess about the equivalence classes of t, u modulo 3. We suppose $t \equiv_3 0$ and $u \equiv_3 0$. For simplicity, we consider a single array variable a. The idea is to triplicate a into a_0, a_1, a_2 where a_j records the values of $a[i]$ for $i \equiv_3 j$. Recall that ν is of the kind $\nu(i, a[i], a[i+1])$; adding a literal $t = 3c$, $u = 3d$ (with fresh c, d), we may rewrite (19) as

$$\forall i \ (i \in [c, d) \rightarrow \nu(i, a_0[i], a_1[i]))$$

preserving satisfiability. Clearly, the general case is much more involved (and presumably not optimal from the complexity viewpoint), because one has to transform all together many guards of the form (18)-(19): this requires the consideration of the l.c.m. of the congruences indexes involved, etc. Still, the above sketch should make evident that a reduction is in principle possible. □

The conclusion of the analysis of this section is that Theorem 3 encompasses many results from current literature concerning decidability of quantified fragments of the theory of arrays, although there are known orthogonal classes still not covered by it (notably, the two-sorted monic-flat fragment of [4] and the non-singly universally quantified class of [9]).

$\ell \wedge \forall i \ (a_\ell[i] = a_\ell[i+1])$ (this constraint is a conjunction of a ground literal and of a Γ-guard, for Γ as in Theorem 1).

B Missed Proofs

Proposition 1 *The set of iteratable formulæ form a normal acceleratable fragment.*

Proof. Closure under renamings is immediate. For closure under conjunctions, it is sufficient to take the union of the corresponding sets of iterators (notice that, in case of overlaps of iterators domains, it might well happen that taking conjunctions of iterator formulæ yield an inconsistent formula as a result - this is not a problem from a formal point of view). By Example 1 above, it is clear that $x' = x + 1$ is an iteratable formula; that $x' = y'$ is iteratable is clear from Definition 4 (x and y do not occur in it). Hence the fragment of definition 4 is normal.

We show that an iteratable formula ϕ is acceleratable (this is a straightforward, but a little annoying computation). Suppose that free variables occurring in ϕ are partitioned as $z_1, \ldots, z_n, z'_1, \ldots, z'_n, \underline{w}'$; to make our notation more compact, we let $\underline{z} := z_1, \ldots, z_n$ and $\underline{x} := \underline{z}, \underline{w}$. The accelerated formula $\phi^*(\underline{x}, \underline{x}', j)$ is given by (6), namely

$$\underline{z}' = \mathbf{v}(\underline{z}, j) \wedge \forall k \in [0, j)\ \exists \underline{\tilde{w}} \begin{pmatrix} (k = j-1 \rightarrow \underline{\tilde{w}} = \underline{w}')\ \wedge \\ \wedge\ \phi(\mathbf{v}(\underline{z}, k), \mathbf{v}(\underline{z}, k+1), \tilde{w}) \end{pmatrix}$$

where the tuple of terms $\mathbf{v}(\underline{z}, k) := v_1(\underline{z}, k), \ldots, v_n(\underline{z}, k)$ is obtained as follows. For every $z_l \in \underline{z}$, choose some iterator $\mathbf{u}^l \in I_\phi$ such that z_l is in the domain of $\mathbf{u}^l$: if z_l occurs at the h-th place in such a domain, we let v_l be $(\mathbf{u}^l)^*_h(\underline{z}, k)$. In other words: we compute the iteration $(\mathbf{u}^l)^*$ of $\mathbf{u}^l$ according to (3), take its h-component, and apply it to $\underline{z}$ and k.

By induction, using (4) and (3), we first check that

$$\phi^K(\underline{x}, \underline{x}') \rightarrow \underline{z}' = \mathbf{v}(\underline{z}, \overline{K}) \tag{20}$$

is valid for every $K \geq 1$: this follows from the (easily seen) fact that we have $\models \phi^K(\underline{x}, \underline{x}') \rightarrow \underline{z}'_0 = \mathbf{u}^*(\underline{z}_0, \overline{K})$, for every K and for every $\mathbf{u} \in I_\phi$ having domain $\underline{z}_0 \subseteq \underline{z}$.

We prove that (6) satisfies (1) by induction on j; the case $j = 1$ is immediate, because $\mathbf{v}(\underline{z}, 0) = \underline{z}$, and $\mathbf{v}(\underline{z}, 1) = \underline{z}'$ holds under assumption $\phi(\underline{x}, \underline{x}')$ by (4) and (3). Suppose now $j > 1$; we need to prove that $\phi(\underline{x}, \underline{x}')^j$ is equivalent to the formula (6) where we replace the variable j by the numeral $\bar{j}$. After this replacement, the relativized quantifier $\forall k \in [0, \bar{j})$ can be turned into a conjunction ($k \in [0, \bar{j})$ is equivalent to $k = 0 \vee \cdots \vee k = \overline{j-1}$), thus we may rewrite (6) as

$$\underline{z}' = \mathbf{v}(\underline{z}, j) \wedge \exists\, \underline{\tilde{w}}_0 \cdots \underline{\tilde{w}}_{j-1} \begin{pmatrix} \underline{\tilde{w}}_{j-1} = \underline{w}' \wedge \\ \wedge \displaystyle\bigwedge_{k=0}^{j-1} \phi(\mathbf{v}(\underline{z}, k), \mathbf{v}(\underline{z}, k+1), \tilde{w}_k) \end{pmatrix} \tag{21}$$

Now recall that $\phi^j(\underline{x}, \underline{x}')$ is $\exists \underline{x}^\sharp\, (\phi^{j-1}(\underline{x}, \underline{x}^\sharp) \wedge \phi(\underline{x}^\sharp, \underline{x}'))$, that is

$$\exists \underline{z}^\sharp, \underline{w}^\sharp\ (\phi^{j-1}(\underline{x}, \underline{z}^\sharp, \underline{w}^\sharp) \wedge \phi(\underline{z}^\sharp, \underline{x}'))\ .$$

The latter, moving all existential quantifiers in front and applying induction hypothesis, can be written as $\exists \underline{z}^\sharp$, $\underline{w}^\sharp$, $\underline{\tilde{w}}_0 \cdots \underline{\tilde{w}}_{j-2}\ \theta$, where the matrix θ is equivalent to

$$\begin{aligned} & \underline{z}^\sharp = \mathbf{v}(\underline{z}, j-1) \wedge\ \underline{\tilde{w}}_{j-2} = \underline{w}^\sharp \wedge \\ & \wedge \bigwedge_{k=0}^{j-2} \phi(\mathbf{v}(\underline{z}, k), \mathbf{v}(\underline{z}, k+1), \tilde{w}_k)\ \wedge\ \phi(\underline{z}^\sharp, \underline{z}', \underline{w}') \wedge \\ & \wedge\ \underline{z}' = \mathbf{v}(\underline{z}, j) \end{aligned} \tag{22}$$

(the last conjunct has been added taking into consideration (20) for $K := j$). We can now remove the redundant quantification over $\underline{w}^\sharp$ and introduce a new quantification $\exists \underline{\tilde{w}}_{j-1}(\underline{w}' = \underline{\tilde{w}}_{j-1} \wedge \cdots)$; thus we get an existential formula $\exists \underline{z}^\sharp\ \underline{\tilde{w}}_0 \cdots \underline{\tilde{w}}_{j-1}\ \theta'$, where θ' now is

$$\begin{aligned} & \underline{z}^\sharp = \mathbf{v}(\underline{z}, j-1)\ \wedge\ \underline{\tilde{w}}_{j-1} = \underline{w}' \wedge \\ & \wedge \bigwedge_{k=0}^{j-2} \phi(\mathbf{v}(\underline{z}, k), \mathbf{v}(\underline{z}, k+1), \tilde{w}_k)\ \wedge\ \phi(\underline{z}^\sharp, \underline{z}', \underline{\tilde{w}}_{j-1}) \wedge \\ & \wedge\ \underline{z}' = \mathbf{v}(\underline{z}, j) \end{aligned} \tag{23}$$

If we remove also the quantifier $\exists \underline{z}^\sharp$ by replacing $\underline{z}^\sharp$ with $\mathbf{v}(\underline{z}, j-1)$, the matrix of the resulting formula can be witten as the conjunction of

$$\underline{\tilde{w}}_{j-1} = \underline{w}'\ \wedge \bigwedge_{k=0}^{j-2} \phi(\mathbf{v}(\underline{z}, k), \mathbf{v}(\underline{z}, k+1), \tilde{w}_k) \tag{24}$$

with

$$\phi(\mathbf{v}(\underline{z}, j-1), \underline{z}', \underline{\tilde{w}}_{j-1})\ \wedge\ \underline{z}' = \mathbf{v}(\underline{z}, j)\ . \tag{25}$$

Now, (24) $\wedge$ (25) is the same as the matrix of (21), because (25) and

$$\phi(\mathbf{v}(\underline{z}, j-1), \mathbf{v}(\underline{z}, j), \underline{\tilde{w}}_{j-1})\ \wedge\ \underline{z}' = \mathbf{v}(\underline{z}, j)$$

are logically equivalent. □

Theorem 4 *The unbounded reachability problem for $\mathcal{P}$ is decidable if there is a normal acceleratable fragment containing all arithmetic projections of $\mathcal{P}$.*

Proof. For a transition relation $\tau(\mathbf{v}, \overline{\mathbf{v}})$ given by (14), the accelerated transition τ^+ allows to represent *in one shot* the *precise* set of states reachable after n unwindings of that loop, for any n. By definition, the acceleration of a transition $\tau(\mathbf{v}, \overline{\mathbf{v}})$ is the union of the n-th compositions of τ with itself, i.e. it is $\tau^+(\mathbf{v}, \overline{\mathbf{v}}) := \bigvee_{n>0} \tau^n(\mathbf{v}, \overline{\mathbf{v}})$, where

$$\tau^1(\mathbf{v}, \overline{\mathbf{v}}) := \tau(\mathbf{v}, \overline{\mathbf{v}}), \quad \tau^{n+1}(\mathbf{v}, \overline{\mathbf{v}}) := \exists \tilde{\mathbf{v}}\ (\tau^n(\mathbf{v}, \tilde{\mathbf{v}}) \wedge \tau(\tilde{\mathbf{v}}, \overline{\mathbf{v}}))\ .$$

Claim: we prove that $\tau^n(\mathbf{v}, \overline{\mathbf{v}})$ is equivalent to the formula $\tau^*(\mathbf{v}, \overline{\mathbf{v}}, \bar{n})$ given by (16), namely:

$$\begin{pmatrix} \forall i \in [I, I+n) & \gamma(i, \overline{\mathbf{a}}[i], \mathbf{a}[i+1], \overline{\mathbf{a}}[i+1]) & \wedge \\ \forall i \in [0, I+1) & \overline{\mathbf{a}}[i] = \mathbf{a}[i] & \wedge \\ \forall i \in [I+n+1, N+1) & \overline{\mathbf{a}}[i] = \mathbf{a}[i] & \wedge \\ \overline{I} = I+n & & \end{pmatrix}$$

We argue by induction on n. In fact, if $n = 1$ it is clear that (16) is equivalent to (14) (this is because $\forall i \neq I{+}1$ $(\mathbf{a}[i] = \overline{\mathbf{a}}[i])$ entails $\mathbf{a}[I] = \overline{\mathbf{a}}[I]$).

For the induction step, we have that the composition of the relation (14) with itself $n+1$-times is represented (using induction hypothesis) by the relation

$$\exists \tilde{I}\ \exists \tilde{\mathbf{a}}\ \begin{pmatrix} \tau^*(I, \mathbf{a}, \tilde{I}, \tilde{\mathbf{a}}, n) \wedge (\forall i \neq \tilde{I}{+}1\ \tilde{\mathbf{a}}[i] = \overline{\mathbf{a}}[i]) \wedge \\ \wedge\ \gamma(\tilde{I}, \tilde{\mathbf{a}}[\tilde{I}], \tilde{\mathbf{a}}[\tilde{I}+1], \overline{\mathbf{a}}[\tilde{I}{+}1])\ \wedge\ \overline{I} = \tilde{I}{+}1 \end{pmatrix} \tag{26}$$

This formula contains existential quantifiers ($\exists \tilde{\mathbf{a}}$ are second order quantifiers); however, after a direct inspection, we shall realize below that such quantifiers are redundant. In fact, (26) can be rewritten to

$$\exists \tilde{I}\ \exists \tilde{\mathbf{a}}\ \begin{pmatrix} \tilde{I} = I+n\ \wedge \tilde{\mathbf{a}} = \lambda j\ (\textbf{if}\ j = I{+}n{+}1\ \textbf{then}\ \mathbf{a}[I{+}n{+}1]\ \textbf{else}\ \overline{\mathbf{a}}[j]) \wedge \\ \wedge\ \tau^*(I, \mathbf{a}, \tilde{I}, \tilde{\mathbf{a}}, n) \wedge (\forall i \neq \tilde{I}{+}1\ \tilde{\mathbf{a}}[i] = \overline{\mathbf{a}}[i]) \wedge \\ \wedge\ \gamma(\tilde{I}, \tilde{\mathbf{a}}[\tilde{I}], \tilde{\mathbf{a}}[\tilde{I}{+}1], \overline{\mathbf{a}}[\tilde{I}{+}1])\ \wedge\ \overline{I} = \tilde{I}{+}1 \end{pmatrix}$$

Eliminating the existential quantifiers and applying β-conversion, we get:

$$\begin{pmatrix} \forall i \in [I, I+n)\ \gamma(i, \overline{\mathbf{a}}[i], \mathbf{a}[i+1], \overline{\mathbf{a}}[i+1]) & \wedge \\ \forall i \in [0, I+1)\ \overline{\mathbf{a}}[i] = \mathbf{a}[i] & \wedge \\ \forall i \in [I+n+2, N+1)\ \overline{\mathbf{a}}[i] = \mathbf{a}[i] & \wedge \\ \gamma(I{+}n, \overline{\mathbf{a}}[I{+}n], \mathbf{a}[I{+}n{+}1], \overline{\mathbf{a}}[I{+}n{+}1]) & \wedge \\ \overline{I} = I{+}n{+}1 & \end{pmatrix}$$

which is precisely $\tau^*(I, \mathbf{a}, \overline{I}, \overline{\mathbf{a}}, n{+}1)$. This ends of the proof of the claim.

Because of the above claim, in the class of our standard models we have that $\tau^+(\mathbf{v}, \overline{\mathbf{v}})$ is equivalent to $\exists x\ \tau^*(\mathbf{v}, \overline{\mathbf{v}}, x)$. We extend the projection function $\mathcal{L}$ by denoting $\mathcal{L}^+(e) := \mathcal{L}(e)^+$ if $src(e) = trt(e)$ and $\mathcal{L}^+(e) := \mathcal{L}(e)$ otherwise, where $\mathcal{L}(e)^+$ denotes the acceleration of the transition labeling the edge e.

Let now Γ be a normal acceleratable fragment containing all arithmetic projections of $\mathcal{P}$: *the key observation is that* (16) *is a conjunction of ground literals and Γ-guards*: to see it, recall that Γ is normal (in the sense of Definition 1) and

notice that we can slightly rewrite (16) as:[11]

$$\begin{pmatrix} \forall i \in [I, I+n) & \gamma(i, \overline{\mathbf{a}}[i], \mathbf{a}[i+1], \overline{\mathbf{a}}[i+1]) & \wedge \\ \forall i \in [0, I) & \overline{\mathbf{a}}[i{+}1] = \mathbf{a}[i{+}1] & \wedge \\ \forall i \in [I{+}n{+}1, N) & \overline{\mathbf{a}}[i{+}1] = \mathbf{a}[i{+}1] & \wedge \\ \overline{\mathbf{a}}[0] = \mathbf{a}[0] \ \wedge \ \overline{\mathbf{a}}[I{+}n{+}1] = \mathbf{a}[I{+}n{+}1] & & \wedge \\ \overline{I} = I + n & & \end{pmatrix} \tag{27}$$

Let $\rho = e_1, \ldots, e_m$ be an error path of $\mathcal{P}$; when testing its feasibility, according to Definition 7, we can limit ourselves to the case in which $e_1, \ldots, e_m$ are all distinct, provided we replace the labels $\mathcal{L}(e_k)^{(k)}$ with $\mathcal{L}^+(e_k)^{(k)}$ in the formula $\bigwedge_{j=1}^{m} \mathcal{L}(e_j)^{(j)}$ from Definition 6.[12] Thus $\mathcal{P}$ is unsafe iff, for some path $e_1, \ldots, e_m$ whose edges are all distinct, the formula

$$\mathcal{L}^+(e_1)^{(1)} \wedge \cdots \wedge \mathcal{L}^+(e_m)^{(m)} \tag{28}$$

is satisfiable. Since the involved paths are finitely many and the satisfiability of the formulæ (28) is decidable by Theorem 3 (the existential quantifiers $\exists x \ \tau^*(\mathbf{v}, \overline{\mathbf{v}}, x)$ occurring in (28) can be skolemized away via free constants), the safety of $\mathcal{P}$ can be decided. $\square$

[11] Normality is needed for the second and the third conjunct of (27) to be Γ-guards.

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