A Logic of Singly Indexed Arrays

Peter Habermehl[1], Radu Iosif[2], and Tomáš Vojnar[3]

[1] LSV, ENS Cachan, CNRS, INRIA; 61 av. du Président Wilson, F-94230 Cachan, France
and LIAFA, University Paris 7, Case 7014, 75205 Paris Cedex 13, haberm@liafa.jussieu.fr
[2] VERIMAG, CNRS, 2 av. de Vignate, F-38610 Gières, France, e-mail:iosif@imag.fr
[3] FIT BUT, Božetěchova 2, CZ-61266, Brno, Czech Republic, e-mail: vojnar@fit.vutbr.cz

Abstract. We present a logic interpreted over integer arrays, which allows difference bound comparisons between array elements situated within a constant sized window. It is shown that the satisfiability problem for the logic is undecidable for the class of formulae with quantifier prefix $\{\exists,\forall\}^*\forall^*\exists^*\forall^*$. For formulae with quantifier prefixes in the $\exists^*\forall^*$ fragment, decidability is established by an automata-theoretic argument. For each formula in the $\exists^*\forall^*$ fragment we can build a flat counter automaton with difference bound transition rules (FCADBM) whose traces correspond to the models of the formula. The construction is modular, following the syntax of the formula. Decidability of the $\exists^*\forall^*$ fragment of the logic is a consequence of the fact that reachability of a control state is decidable for FCADBM.

1 Introduction

Arrays are commonplace data structures in most programming languages. Reasoning about programs with arrays calls for expressive logics capable of encoding pre- and post-conditions as well as loop invariants. Moreover, in order to automate program verification, one needs tractable logics whose satisfiability problems can be answered by efficient algorithms.

In this paper, we present a logic of integer arrays based on universally quantified comparisons between array elements situated within a constant sized window, i.e., quantified boolean combinations of basic formulae of the form $\forall i \,.\, \gamma(i) \rightarrow a_1[i+k_1] - a_2[i+k_2] \leq m$ where γ is a positive boolean combination of bound and modulo constraints on the index variable i, a_1 and a_2 are array symbols, and $k_1, k_2, m \in \mathbb{Z}$ are integer constants. Hence the name of Single Index Logic (**SIL**).

The main idea in defining the logic is that only one universally quantified index may be used on the right hand side of the implication within a basic formula. According to [9], this restriction is not a real limitation of the expressive power of the logic since a formula using two or more universally quantified variables in a difference bound constraint on array values can be equivalently written in the form above, by introducing fresh array symbols. This technique has been detailed in [9].

Working directly with singly-indexed formulae allows to devise a simple and efficient decision procedure for the satisfiability problem of the $\exists^*\forall^*$ fragment of **SIL**, based on a modular translation of formulae into deterministic flat counter automata with difference bound transition rules (FCADBM). This is possible due to the fact that deterministic FCADBM are closed under union, intersection and complement, when considering their sets of traces.

The satisfiability problem for $\exists^*\forall^*$-**SIL** is thus reduced to checking reachability of a control state in an FCADBM. The latter problem has been shown to be decidable first in [6], by reduction to the satisfiability problem of Presburger arithmetic. Later on, the method described in [4] reduced this problem to checking satisfiability of a linear Diophantine system, leading to the implementation of the FLATA toolset [7].

Universally quantified formulae of the form $\forall i \,.\, \gamma(i) \rightarrow \upsilon(i)$ are a natural choice when reasoning about arrays as one usually tend to describe facts that must hold for all array elements (array invariants). A natural question is whether a more complex quantification scheme is possible, while preserving decidability. In this paper, we show that the satisfiability problem for the class of formulae with quantifier prefixes of the form $\forall^*\exists^*\forall^*$ is already undecidable, providing thus a formal reason for the choice of working with existentially quantified boolean combinations of universal basic formulae. The contribution of this paper is hence three-fold:

- we show that the satisfiability problem for the class of formulae with quantifier prefixes of the form $\forall^*\exists^*\forall^*$ is undecidable,
- we define a class of counter automata that is closed under union, intersection and complement of their sets of traces,
- we provide a decision procedure for the satisfiability problem within the fragment of formulae with alternation depth of at most one, based on a modular, simple, and efficient translation of formulae into counter automata.

The practical usefulness of the **SIL** logic is shown by giving a number of examples of properties that are recurrent in programs handling array data structures.

Related Work. The saga of papers on logical theories of arrays starts with the seminal paper [14], in which the read and write functions from/to arrays and their logical axioms were introduced. A decision procedure for the quantifier-free fragment of the theory of arrays was presented in [11]. Since then, various quantifier-free decidable logics on arrays have been considered—e.g., [16, 12, 10, 15, 1, 8].

In [5], an interesting logic, within the $\exists^*\forall^*$ quantifier fragment, is developed. Unlike our decision procedure based on automata theory, the decision procedure of [5] is based on a model-theoretic observation, allowing to replace universal quantification by a finite conjunction. The decidability of their theory depends on the decidability of the base theory of array values. However, compared to our results, [5] does not allow modulo constraints (allowing to speak about periodicity in the array values) nor reasoning about array entries at a fixed distance (i.e., reasoning about $a[i]$ and $a[i+k]$ for a constant k and a universally quantified index i). The authors of [5] give also interesting undecidability results for extensions of their logic. For example, they show that relating adjacent array values ($a[i]$ and $a[i+1]$), or having nested reads, leads to undecidability.

A restricted form of universal quantification within $\exists^*\forall^*$ formulae is also allowed in [2], where decidability is obtained based on a small model property. Unlike [5] and our work, [2] allows a hierarchy-restricted form of array nesting. However, similar to the restrictions presented above, neither modulo constraints on indices, nor reasoning about array entries at a fixed distance are allowed. A similar restriction not allowing to express properties of consecutive elements of arrays appears also in [3], where a quite general $\exists^*\forall^*$ logic on multisets of elements with associated data values is considered.

The closest in spirit to the present paper is our previous work in [9]. There, we established decidability of formulae in the $\exists^*\forall^*$ quantifier prefix class when references to adjacent array values (e.g., $a[i]$ and $a[i+1]$) are not used in disjunctive terms. However, there are two essential differences between this work and the one reported in [9].

On one hand, the basic propositions from [9], allowing multiple universally quantified indices could not be translated directly into counter automata. This led to a complex elimination procedure based on introducing new array symbols, which produces singly-indexed formulae. However, the automata resulting from this procedure are not closed under complement. Therefore, negation had to be eliminated prior to reducing the formula to the singly-indexed form, causing further complexity. In the present work, we start directly with singly-indexed formulae, convert them into automata, and compose the automata directly using boolean operators (union, intersection, complement).

On the other hand, using universally quantified array property formulae as building blocks for the formulae, although intuitive, is not formally justified in [9]. Here we provide the argument that, alternating quantifiers to a depth more than two leads to undecidability.

Roadmap. The paper is organised as follows. Section 2 introduces the necessary notions on counter automata and defines the class of FCADBM. Section 3 defines the logic **SIL**. Next, Section 4 gives the undecidability result for the entire logic, while Section 5 proves decidability of the satisfiability for the $\exists^*\forall^*$ fragment, by translation to deterministic FCADBM. Finally, Section 6 presents some concluding remarks.

2 Counter Automata

Given a formula φ, we denote by $FV(\varphi)$ the set of its free variables. If we denote a formula as $\varphi(x_1,\ldots,x_n)$, we assume $FV(\varphi) \subseteq \{x_1,\ldots,x_n\}$. For $\varphi(x)$, we denote by $\varphi[t/x]$ the formula in which each free occurrence of x is replaced by a term t. Given a formula φ, we denote by $\models \varphi$ the fact that φ is logically valid, i.e., it holds in every structure corresponding to its signature.

A *difference bound matrix* (DBM) formula is a conjunction of inequalities of the forms (1) $x - y \leq c$, (2) $x \leq c$, or (3) $x \geq c$, where $c \in \mathbb{Z}$ is a constant. We denote by $\top$ (true) the empty DBM. It is well-known that the negation of a DBM formula is equivalent to a finite disjunction of *pairwise disjoint* DBM formulae since, e.g., $\neg(x - y \leq c) \iff y - x \leq -c-1$ and $\neg(x \leq c) \iff x \geq c+1$. In particular, the negation of $\top$ is the empty disjunction, denoted as $\bot$ (false).

A *counter automaton* (CA) is a tuple $A = \langle \mathbf{x}, Q, I, \rightarrow, F \rangle$ where:

- $\mathbf{x}$ is a finite set of counters ranging over $\mathbb{Z}$,
- Q is a finite set of control states,
- $I \subseteq Q$ is a set of initial states,

- $\rightarrow$ is a transition relation given by a set of rules $q \xrightarrow{\varphi(\mathbf{x},\mathbf{x}')} q'$ where φ is an arithmetic formula relating current values of counters $\mathbf{x}$ to their future values $\mathbf{x}' = \{x' \mid x \in \mathbf{x}\}$,
- $F \subseteq Q$ is a set of final states.

A *configuration* of a counter automaton A is a pair (q, ν) where $q \in Q$ is a control state, and $\nu : \mathbf{x} \rightarrow \mathbb{Z}$ is a valuation of the counters in $\mathbf{x}$. For a configuration $c = (q, \nu)$, we designate by $val(c) = \nu$ the valuation of the counters in c. A configuration (q', ν') is an *immediate successor* of (q, ν) if and only if A has a transition rule $q \xrightarrow{\varphi(\mathbf{x},\mathbf{x}')} q'$ such that $\models \varphi(\nu(\mathbf{x}), \nu'(\mathbf{x}'))$. A configuration c is a *successor* of another configuration c' if and only if there exists a finite sequence of configurations $c = c_1 c_2 \ldots c_n = c'$ such that, for all $1 \leq i < n$, c_{i+1} is an immediate successor of c_i. Given two control states $q, q' \in Q$, a run of A from q to q' is a finite sequence of configurations $c_1 c_2 \ldots c_n$ with $c_1 = (q, \nu)$, $c_n = (q', \nu')$ for some valuations $\nu, \nu' : \mathbf{x} \rightarrow \mathbb{Z}$, and c_{i+1} is an immediate successor of c_i, for all $1 \leq i < n$. Let $\mathcal{R}(A)$ denote the set of runs of A from some initial state $q_0 \in I$ to some final state $q_f \in F$, and $Tr(A) = \{val(c_1)val(c_2)\ldots val(c_n) \mid c_1 c_2 \ldots c_n \in \mathcal{R}(A)\}$ be its set of *valuation traces*. If $\mathbf{z} \subseteq \mathbf{x}$ is a subset of the counters of A and $\nu : \mathbf{x} \rightarrow \mathbb{Z}$ is a valuation of its counters, let $\nu\downarrow_{\mathbf{z}}$ be the restriction of ν to the counters in $\mathbf{z}$. If $c = (q, \nu)$ is a configuration of A, we denote $c\downarrow_{\mathbf{z}} = (q, \nu\downarrow_{\mathbf{z}})$ and $Tr(A)\downarrow_{\mathbf{z}} = \{val(c_1)\downarrow_{\mathbf{z}} val(c_2)\downarrow_{\mathbf{z}} \ldots val(c_n)\downarrow_{\mathbf{z}} \mid c_1 c_2 \ldots c_n \in \mathcal{R}(A)\}$.

A counter $z \in \mathbf{x}$ is called a *parameter* of A if and only if, for each $\sigma = \nu_1 \ldots \nu_n \in Tr(A)$, we have $\nu_1(z) = \ldots = \nu_n(z)$, in other words the value of the counter does not change during any run of A.

A *control path* in a counter automaton A is a finite sequence $q_1 q_2 \ldots q_n$ of control states such that, for all $1 \leq i < n$, there exists a transition rule $q_i \xrightarrow{\varphi_i} q_{i+1}$. A *cycle* is a control path starting and ending in the same control state. An *elementary cycle* is a cycle in which each state appears only once, except for the first one, which appears both at the beginning and at the end. A counter automaton is said to be *flat* iff each control state belongs to at most one elementary cycle.

A counter automaton A is said to be *deterministic* if and only if (1) it has exactly one initial state, and (2) for each pair of transition rules with the same source state $q \xrightarrow{\varphi} q'$ and $q \xrightarrow{\psi} q''$, we have $\models \neg(\varphi \wedge \psi)$. It is easy to prove that, given a deterministic counter automaton A, for each sequence of valuations $\nu_1 \nu_2 \ldots \nu_n \in Tr(A)$ there exists exactly one control path $q_1 q_2 \ldots q_n$ such that $(q_0, \nu_1)(q_1, \nu_2) \ldots (q_{n-1}, \nu_n) \in \mathcal{R}(A)$.

2.1 Flat Counter Automata with DBM Transition Rules

In the rest of the paper, we use the class of *flat counter automata with DBM transition rules* (FCADBM). They are defined to be flat counter automata where each transition in a cycle is labelled by a DBM formula and each transition not in a cycle is labelled by a conjunction of a DBM formula with a (possibly empty) conjunction of modulo constraints on parameters of the form $z \equiv_s t$ where $0 \leq t < s$.

An extension of this class has been studied in [9]. Using results of [6, 4], [9] shows that, given a CA $A = \langle \mathbf{x}, Q, I, \rightarrow, F \rangle$ in the class it considers and a pair of control states

$q, q' \in Q$, the set $V_{q,q'} = \{(\nu, \nu') \in (\mathbf{x} \mapsto \mathbb{Z})^2 \mid A$ has a run from (q,ν) to $(q',\nu')\}$ is Presburger-definable. As an immediate consequence, the emptiness problem for A, i.e., $Tr(A) \stackrel{?}{=} \emptyset$, is decidable.

Theorem 1. *The emptiness problem for FCADBM is decidable.*

In this section, we show that *deterministic* FCADBM are closed under union, intersection, and complement of their sets of traces. Let $A_i = \langle \mathbf{x}, Q_i, \{q_{0i}\}, \rightarrow_i, F_i \rangle$, $i = 1,2$, be two deterministic FCADBM with the same set of counters. Note that this is not a restriction since one can add unrestricted counters without changing the behaviour of a counter automaton. We first show closure under intersection by defining the counter automaton $A_1 \otimes A_2 = \langle \mathbf{x}, Q_1 \times Q_2, \{(q_{01}, q_{02})\}, \rightarrow, F_1 \times F_2 \rangle$ where:

$$(q_1,q_2) \xrightarrow{\varphi} (q_1',q_2') \iff q_1 \xrightarrow{\psi_1}_1 q_1',\ q_2 \xrightarrow{\psi_2}_2 q_2', \text{ and } \models \varphi \leftrightarrow \psi_1 \wedge \psi_1$$

The next lemma proves the correctness of our construction.

Lemma 1. *For any two deterministic FCADBM $A_i = \langle \mathbf{x}, Q_i, \{q_{i0}\}, \rightarrow_i, F_i \rangle$, $i = 1,2$, $A_1 \otimes A_2$ is a deterministic FCADBM, and $Tr(A_1 \otimes A_2) = Tr(A_1) \cap Tr(A_2)$.*

Proof. The fact that $A_1 \otimes A_2$ is flat is proven as a part of Lemma 3 in [9], and it is immediate to see that its transitions are labelled by a conjunction of a DBM and modulo constraints where the latter do not appear in cycles. To show that $A_1 \otimes A_2$ is deterministic, let $(q_1,q_2) \xrightarrow{\varphi} (q_1',q_2')$ and $(q_1,q_2) \xrightarrow{\psi} (q_1'',q_2'')$ be two transitions of $A_1 \otimes A_2$ originating in the same control state. By definition, there are transitions $q_1 \xrightarrow{\varphi_1} q_1'$, $q_1 \xrightarrow{\psi_1} q_1''$ of A_1 and transitions $q_2 \xrightarrow{\varphi_2} q_2'$, $q_2 \xrightarrow{\psi_2} q_2''$ of A_2 such that $\models \varphi \leftrightarrow \varphi_1 \wedge \varphi_2$ and $\models \psi \leftrightarrow \psi_1 \wedge \psi_2$. Since A_1 and A_2 are deterministic, we have $\models \neg(\varphi_1 \wedge \psi_1)$ and $\models \neg(\varphi_2 \wedge \psi_2)$. But then, $\varphi \wedge \psi \leftrightarrow (\varphi_1 \wedge \varphi_2) \wedge (\psi_1 \wedge \psi_2) \leftrightarrow (\varphi_1 \wedge \psi_1) \wedge (\varphi_2 \wedge \psi_2)$, which is unsatisfiable since both $\varphi_1 \wedge \psi_1$ and $\varphi_2 \wedge \psi_2$ are.

Let $\nu_1\nu_2 \ldots \nu_n \in Tr(A_1 \otimes A_2)$. Then there exists a run $c_1c_2 \ldots c_n$ of $A_1 \otimes A_2$ such that $c_{i+1} = \langle (q_i^1, q_i^2), \nu_{i+1} \rangle$ for all $0 \le i < n$ where q_i^1 and q_i^2 are control states of A_1 and A_2, respectively. Moreover, $q_n^i \in F_i$, $i = 1,2$. By the definition of $A_1 \otimes A_2$, $\langle q_0^i, \nu_1 \rangle, \langle q_1^i, \nu_2 \rangle \ldots \langle q_{n-1}^i, \nu_n \rangle$ is a run of A_i, $i = 1,2$, hence $\nu_1\nu_2 \ldots \nu_n \in Tr(A_1) \cap Tr(A_2)$, i.e., $Tr(A_1 \otimes A_2) \subseteq Tr(A_1) \cap Tr(A_2)$. The opposite direction uses a similar argument. $\square$

Let $A = \langle \mathbf{x}, Q, I, \rightarrow, F \rangle$ be a deterministic FCADBM. Then we define $\overline{A} = \langle \mathbf{x}, Q \cup \{q_s\}, I, \rightarrow', (Q \setminus F) \cup \{q_s\} \rangle$ where $q_s \notin Q$ is a fresh sink state. The transition relation $\rightarrow'$ is defined as follows. For a control state $q \in Q$, let $O_A(q) = \bigvee_{q \xrightarrow{\varphi} q'} \varphi$.[4] Then, we have:

- $q_s \xrightarrow{\top}{}' q_s$, $q \xrightarrow{\varphi}{}' q'$ for each $q \xrightarrow{\varphi} q'$, and
- $q \xrightarrow{\psi_i}{}' q_s$, for all $1 \le i \le k$, where ψ_i are (unique) conjunctions of DBMs and modulo constraints [5] such that $\models \neg O_A(q) \leftrightarrow \bigvee_{i=1}^k \psi_i$, and $\models \neg(\psi_i \wedge \psi_j)$ for $i \ne j$, $1 \le i, j \le k$.

[4] If q has no immediate successors, then $O_A(q)$ is false by default.
[5] The negation of $z \equiv_s t$ with $t < s$ is equivalent to $\bigvee_{t' \in \{0,\ldots,s-1\} \setminus \{t\}} z \equiv_s t'$.

Flatness of $\overline{A}$ is a consequence of the fact that the only cycle of $\overline{A}$, which did not exist in A is the self-loop around q_s. That is, the newly added transitions do not create new cycles. It is immediate to see that $\overline{A}$ is deterministic whenever A is. The following lemma formalises correctness of the complement construction, proving thus that deterministic FCADBM are effectively closed under union[6], intersection, and complement of their sets of traces.

Lemma 2. *Given a deterministic FCADBM $A = \langle \mathbf{x}, Q, \{q_0\}, \rightarrow, F\rangle$, for any finite sequence of valuations $\sigma \in (\mathbf{x} \mapsto \mathbb{Z})^*$, we have $\sigma \in Tr(A)$ if and only if $\sigma \notin Tr(\overline{A})$.*

Proof. "$\Rightarrow$" Let $\sigma = \nu_1\nu_2\ldots\nu_n \in Tr(A)$. Then there exists a run $\langle q_0, \nu_1\rangle, \langle q_1, \nu_2\rangle, \ldots, \langle q_{n-1}, \nu_n\rangle$ of A such that $q_{n-1} \in F$. By the definition of $\overline{A}$, the same run is possible in A. Since $\overline{A}$ is deterministic, this is the only run of $\overline{A}$ corresponding to σ. However, $\sigma \notin Tr(\overline{A})$ since $q_{n-1} \notin (Q \setminus F) \cup \{q_s\}$.

"$\Leftarrow$" Let $\sigma = \nu_1\nu_2\ldots\nu_n \in Tr(\overline{A})$. Then there exists a run $\langle q_0, \nu_1\rangle, \langle q_1, \nu_2\rangle, \ldots, \langle q_{n-1}, \nu_n\rangle$ of $\overline{A}$ such that $q_{n-1} \in (Q \setminus F) \cup \{q_s\}$. If $q_{n-1} \in Q \setminus F$, the same run is possible in A. Since A is deterministic, this is the only run of A corresponding to σ. But since $q_{n-1} \notin F$, $\sigma \notin Tr(A)$. In the other case, if $q_{n-1} = q_s$, by the definition of $\overline{A}$, q_s was entered for the first time within some configuration $\langle q_{j-1}, \nu_j\rangle$, $1 < j \le n$, of the run (and was not left any more). Since A is deterministic, $\langle q_0, \nu_1\rangle, \ldots, \langle q_{j-1}, \nu_j\rangle$ is the only possible run of A corresponding to the trace $\nu_1 \ldots \nu_j$. Moreover, by the definition of $\overline{A}$, $\nu_{j-1}, \nu_j \not\models Out_A(q_{j-2})$, hence the run does not correspond to any run of A, i.e., $\sigma \notin Tr(A)$. $\square$

3 A Logic of Integer Arrays

3.1 Syntax

We consider three types of variables. The *array-bound variables* (k, l) appear within the bounds that define the intervals in which some property is required to hold. Let *BVar* denote the set of *bound variables*. The *index* (i, j) and *array* (a, b) *variables* are used in array terms. Let *IVar* denote the set of *integer variables* and *AVar* denote the set of *array variables*. All variable sets are supposed to be finite and of known cardinality.

Fig. 1 shows the syntax of the Single Index Logic **SIL**. The term $|a|$ denotes the length of an array variable a. We use the symbol $\top$ to denote the boolean value *true*. In the following, we will write $f \le i \le g$ instead of $f \le i \wedge i \le g$, $i < f$ instead of $i \le f - 1$, $i = f$ instead of $f \le i \le f$, $\varphi_1 \vee \varphi_2$ instead of $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$, and $\forall i \,.\, \upsilon(i)$ instead of $\forall i \,.\, \top \rightarrow \upsilon(i)$. If $\ell_1(k_1), \ldots, \ell_n(k_n)$ are array-bound terms with free variables $k_1, \ldots, k_n \in BVar$, respectively, we write any DBM formula φ on terms $a_1[\ell_1], \ldots, a_n[\ell_n]$, as a shorthand for $(\bigwedge_{k=1}^{n} \forall j \,.\, j = \ell_k \rightarrow a_k[j] = l_k) \wedge \varphi[l_1/a_1[\ell_1], \ldots, l_n/a_n[\ell_n]]$, where $l_1, \ldots, l_n$ are fresh array-bound variables.

For reasons that will be made clear later on, we allow only one index variable to occur within the right hand side of the implication in an array property formula $\forall i \,.\, \gamma \rightarrow \upsilon$,

[6] The FCADBM whose set of traces is the union of the sets of traces of two given FCADBM A_1, A_2 can be obtained simply as $\overline{\overline{A_1} \otimes \overline{A_2}}$.

$n, m, p \ldots$	$\in$	$\mathbb{Z}$	constants
$k, l, \ldots$	$\in$	$BVar$	array-bound variables
$i, j, \ldots$	$\in$	$IVar$	index variables
$a, b, \ldots$	$\in$	$AVar$	array variables
$\sim$	$\in$	$\{\leq, \geq\}$	
B	$:=$	$n \mid k+n \mid \lvert a \rvert + n$	array-bound terms
G	$:=$	$\top \mid i - j \leq n \mid i \leq B \mid B \leq i \mid i \equiv_s t \mid G \wedge G \mid G \vee G$	guard expressions ($0 \leq t < s$)
V	$:=$	$a[i+n] \sim B \mid a[i+n] - b[i+m] \sim p \mid$ $i - a[i+n] \sim m \mid V \wedge V$	value expressions
C	$:=$	$B \sim n \mid B - B \leq n \mid B \equiv_s t$	array-bound constraints ($0 \leq t < s$)
P	$:=$	$\forall i \,.\, G \rightarrow V$	array properties
F	$:=$	$P \mid C \mid \neg F \mid F \wedge F \mid F \vee F \mid \exists i \,.\, F$	formulae

Fig. 1. Syntax of the Single Index Logic

i.e., we require $FV(\upsilon) \cap IVar = \{i\}$. Hence the name Single Index Logic (**SIL**). Note that this does not restrict the expressive power w.r.t. the logic considered in [9]. One can always circumvent this restriction by using the method from [9] based on adding new array symbols together with a transitive (increasing, decreasing, or constant) constraint on their adjacent values. This way a relation between arbitrarily distant entries $a[i]$ and $b[j]$ is decomposed into a sequence of relations between neighbouring entries of a, b, and entries of the auxiliary arrays. However this transformation would greatly complicate the decision procedure, hence we prefer to avoid it here.

Notice also that one can compare an array value with an array-bound variable, or with another array value on the right hand side of an implication in an array property formula $\forall i \,.\, \gamma \rightarrow \upsilon$, but one cannot relate two or more array values with array-bound parameters in the same expression. Allowing more complex comparisons between array values would impact upon the decidability result reported in Section 5. For the same reason, already explained in [9], disjunctive terms are not allowed on the right hand side of implications in array properties.

Let υ be a value expression written in the syntax of Fig. 1 (starting with the V non-terminal). Let $\mathcal{B}(\upsilon)$ be the formula defined inductively on the structure of υ as follows:

- $\mathcal{B}(a[i+n] \leq B) = \mathcal{B}(B \leq a[i+n]) = 0 \leq i+n < |a|$
- $\mathcal{B}(i - a[i+n] \leq m) = \mathcal{B}(a[i+n] - i \leq m) = 0 \leq i+n < |a|$
- $\mathcal{B}(a[i+n] - b[i+m] \leq p) = 0 \leq i+n < |a| \wedge 0 \leq i+m < |b|$
- $\mathcal{B}(\upsilon_1 \wedge \upsilon_2) = \mathcal{B}(\upsilon_1) \wedge \mathcal{B}(\upsilon_2)$

Intuitively, $\mathcal{B}(\upsilon)$ is the conjunction of all sanity conditions needed in order for the array accesses in υ to occur within proper bounds.

3.2 Semantics

Let us fix $AVar = \{a_1, a_2, \ldots, a_k\}$ as the set of array variables for the rest of this section. A *valuation* is a pair of partial functions $\langle \iota, \mu \rangle$ where $\iota : BVar \cup IVar \rightarrow \mathbb{Z}_\perp$ associates an integer value with every free integer variable, and $\mu : AVar \to \mathbb{Z}^*$ associates a *finite sequence* of integers with every array symbol $a \in AVar$. If $\sigma \in \mathbb{Z}^*$ is such a sequence, we denote by $|\sigma|$ its length and by σ_i its i-th element.

By $I_{\iota,\mu}(t)$, we denote the value of the term t under the valuation $\langle \iota, \mu \rangle$. The semantics of a formula φ is defined in terms of the forcing relation $\models$ as follows:

$$
\begin{aligned}
I_{\iota,\mu}(|a|) &= |\mu(a)| \\
I_{\iota,\mu}(a[i+n]) &= \mu(a)_{\iota(i)+n}
\end{aligned}
$$

$$
\begin{array}{rcc}
\langle \iota, \mu \rangle \models a[i+n] \leq B & \iff & I_{\iota,\mu}(a[i+n]) \leq \iota(B) \\
\langle \iota, \mu \rangle \models A_1 - A_2 \leq n & \iff & I_{\iota,\mu}(A_1) - I_{\iota,\mu}(A_2) \leq n \\
\langle \iota, \mu \rangle \models \forall i \,.\, G \to V & \iff & \forall\, n \in \mathbb{Z} \,.\, \langle \iota[i \leftarrow n], \mu \rangle \models G \wedge \mathcal{B}(V) \to V \\
\langle \iota, \mu \rangle \models \exists i \,.\, F & \iff & \langle \iota[i \leftarrow n], \mu \rangle \models F \text{ for some } n \in \mathbb{N}
\end{array}
$$

Notice that the semantics of an array property formula $\forall i \,.\, G \to V$ ignores all values of i for which the array accesses of V are undefined since we consider only the values of i from $\mathbb{Z}$ that satisfy the safety assumption $\mathcal{B}(V)$. For space reasons, we do not give here a full definition of the semantics. However, the missing rules are standard in first-order arithmetic. A *model* of a **SIL** formula $\varphi(\mathbf{k}, \mathbf{a})$ is a valuation $\langle \iota, \mu \rangle$ such that the formula obtained by interpreting each variable $k \in \mathbf{k}$ as $\iota(k)$ and each array variable $a \in \mathbf{a}$ as $\mu(a)$ is logically valid: $\langle \iota, \mu \rangle \models \varphi$. We define $[\![\varphi]\!] = \{\langle \iota, \mu \rangle \mid \langle \iota, \mu \rangle \models \varphi\}$. A formula is said to be:

- *satisfiable* if and only if $[\![\varphi]\!] \neq \emptyset$, and
- *valid* if and only if $[\![\varphi]\!] = (BVar \cup IVar \to \mathbb{Z}_\perp) \times (AVar \to \mathbb{Z}^*)$

With these definitions, the *satisfiability problem* asks, given a formula φ if it has at least one model. Without losing generality, for the satisfiability problem, we can assume that the quantifier prefix of φ (in prenex normal form) does not start with $\exists$. Dually, the *validity problem* asks whether a given formula holds on every possible model. Symmetrically, for the validity problem, one can assume w.l.o.g. that the quantifier prefix of the given formula does not start with $\forall$.

3.3 Examples

We now illustrate the syntax, semantics, and use of the logic **SIL** on a number of examples. For instance, the formula $\forall i \,.\, a[i] = 0$ is satisfied by all functions μ mapping a to a finite sequence of 0's, i.e., $\mu(a) \in 0^*$. It is semantically equivalent to $\forall i \,.\, 0 \leq i < |a| \to a[i] = 0$, in which the range of i has been made explicit.

The formula $\forall i \,.\, 0 \leq i < k \to a[i] = 0$ is satisfied by all pairs $\langle \iota, \mu \rangle$ where μ maps a to a sequence whose first $\iota(k)$ elements (if they exist) are 0, i.e., $\mu(a) \in \{0^n \mid 1 \leq n < \iota(k)\} \cup 0^{\iota(k)}\mathbb{Z}^*$. It is semantically equivalent to $\forall i \,.\, 0 \leq i < \min(|a|, k) \to a[i] = 0$.

The capability of **SIL** to relate array entries at fixed distances (missing in many decidable logics such as those considered in [2, 5, 3]) is illustrated on a bigger example below. The modulo constraints on the index variables can then be used to state periodic facts. For instance, the formula $\forall i \,.\, i \equiv_2 0 \rightarrow a[i] = 0 \wedge \forall i \,.\, i \equiv_2 1 \rightarrow a[i] = 1$ describes the set of arrays a in which the elements on even positions have the value 0, and the elements on odd positions have the value 1.

The logic **SIL** also allows direct comparisons between indices and values. For instance, the formula $\forall i \,.\, a[i] = i+1$ is satisfied by all arrays a which are of the form $1234\ldots$. Alternatively, this can be specified as $a[0] = 1 \;\wedge\; \forall i \,.\, a[i+1] = a[i]+1$ where $a[0] = 1$ is a shorthand for $\forall i \,.\, i = 0 \rightarrow a[i] = 1$. Further, the set of arrays in which the value at position n is between zero and n can be specified by writing $\forall i \,.\, 0 \leq a[i] < i$, which cannot be described without an explicit comparison between indices and values (unless a comparison with an additional array describing the sequence $1234\ldots$ is used).

Checking verification conditions for array manipulating programs. The decision procedure for checking satisfiability of **SIL** formulae, described later on, can be used for discharging verification conditions of various interesting array-manipulating procedures. As a concrete example, let us consider the procedure for an in-situ left rotation of arrays, given below. We annotate the procedure (using double braces) with a pre-condition, post-condition, and a loop invariant. We distinguish below logical variables from program variables (typeset in print). The variable a_0 is a logical variable that relates the initial values of the array `a` with the values after the rotation.

$\{\{\ |\mathtt{a}| = |a_0| \wedge \forall j.\mathtt{a}[j] = a_0[j]\ \}\}$
`x=a[0];`
for `(i=0; i < |a|-1; i++)`
  $\{\{\ \mathtt{x} = a_0[0] \wedge \forall j.0 \leq j < \mathtt{i} \rightarrow \mathtt{a}[j] = a_0[j+1] \wedge \forall j.i \leq j < |\mathtt{a}| \rightarrow \mathtt{a}[j] = a_0[j]\ \}\}$
    `a[i]=a[i+1];`
`a[|a|-1]=x;`
$\{\{\ \mathtt{a}[|\mathtt{a}|-1] = a_0[0] \wedge \forall j.0 \leq j < |\mathtt{a}|-1 \rightarrow \mathtt{a}[j] = a_0[j+1])\ \}\}$

To check (partial) correctness of the procedure, one needs to check three verification conditions out of which we discuss one here (the others are similar). Namely, we consider checking the loop invariant, which requires checking validity of the formula:

$$\begin{gathered}
\mathtt{x} = a_0[0] \wedge \forall j.0 \leq j < \mathtt{i} \rightarrow \mathtt{a}[j] = a_0[j+1] \;\wedge\; \forall j.i \leq j < |\mathtt{a}| \rightarrow \mathtt{a}[j] = a_0[j] \;\wedge \\
\mathtt{i} < |\mathtt{a}|-1 \wedge |\mathtt{a}'| = |\mathtt{a}| \wedge \mathtt{i}' = \mathtt{i}+1 \wedge \mathtt{x}' = \mathtt{x} \;\wedge\; \mathtt{a}'[\mathtt{i}] = \mathtt{a}[\mathtt{i}+1] \wedge \forall j.j \neq \mathtt{i} \rightarrow \mathtt{a}'[j] = \mathtt{a}[j] \\
\longrightarrow \\
\mathtt{x}' = a_0[0] \wedge \forall j.0 \leq j < \mathtt{i}' \rightarrow \mathtt{a}'[j] = a_0[j+1] \;\wedge\; \forall j.i' \leq j < |\mathtt{a}'| \rightarrow \mathtt{a}'[j] = a_0[j]
\end{gathered}$$

Primed variables denote the values of program variables after one iteration of the loop. Checking validity of this formula amounts to checking that its negation is unsatisfiable. The latter condition is expressible in the decidable fragment of **SIL**. Note that the conditions used above refer to adjacent array positions, which could not be expressed in the logics defined in [2, 5, 3].

4 Undecidability of the Logic SIL

In this section, we show that the satisfiability problem for the $\forall^*\exists^*\forall^*$ fragment of **SIL** is undecidable, by reducing from the Hilbert's Tenth Problem [13]. In the following, Section 5 proves the decidability of the satisfiability problem for the fragment of boolean combinations of universally quantified array property formulae—the satisfiability of the $\forall^*$ fragment is proven. Since the leading existential prefix is irrelevant when one speaks about satisfiability, referring either to $\forall^*\exists^*\forall^*$ or to $\exists^*\forall^*\exists^*\forall^*$ makes no difference in this case. However, the question concerning the *validity* problem for the $\exists^*\forall^*$ fragment of **SIL** is still open.

First, we show that multiplication and addition of strictly positive integers can be encoded using formulae of $\forall^*\exists^*\forall^*$-**SIL**. Let $x, y, z \in \mathbb{N}$, with $z > 0$. We define:

$$\varphi_1(j) : a_2[j] > 0 \wedge a_3[j] > 0 \wedge a_1[j+1] = a_1[j]+1 \wedge a_2[j+1] = a_2[j]-1 \wedge a_3[j+1] = a_3[j]$$
$$\varphi_2(j) : a_2[j] = 0 \wedge a_3[j] > 0 \wedge a_1[j+1] = a_1[j] \wedge a_2[j+1] = y \wedge a_3[j+1] = a_3[j]-1$$

$$\begin{aligned}\varphi_{x=yz}(a_1,a_2,a_3,n_1,n_2) : {} & n_1 < n_2 \wedge a_1[n_1] = 0 \wedge a_2[n_1] = y \wedge a_3[n_1] = z \wedge a_1[n_2] = x \wedge a_3[n_2] = 0 \ \wedge \\ & \forall i.(n_1 \leq i < n_2 \to \exists j.i \leq j < n_2 \wedge \varphi_2(j) \wedge \forall k.(i \leq k < j \to \varphi_1(k)))\end{aligned}$$

Notice that $\varphi_{x=yz}$ is in the $\forall^*\exists^*\forall^*$ quantifier fragment of **SIL**.

Lemma 3. *$\varphi_{x=yz}(a_1,a_2,a_3,n_1,n_2)$ is satisfiable if and only if $x = yz$.*

Proof. We first suppose that $x = yz$ and give a model of $\varphi_{x=yz}(a_1,a_2,a_3,n_1,n_2)$. We choose $n_1 = 0$ and $n_2 = (y+1)z$. Then, we choose $a_1[n_2] = x$, $a_2[n_2] = y$ and $a_3[n_2] = 0$. Furthermore, for all j such that $0 \leq j < z$ and for all i such that $0 \leq i \leq y$, we choose $a_1[i + j(y+1)] = i + jy$, $a_2[i + j(y+1)] = y - i$ and $a_3[i + j(y+1)] = z - j$. Then, it is easy to check that this is a model of $\varphi_{x=yz}(a_1,a_2,a_3,n_1,n_2)$.

Let us consider now a model of $\varphi_{x=yz}(a_1,a_2,a_3,n_1,n_2)$. We show that this implies $x = yz$. A model of $n_1 < n_2 \wedge a_1[n_1] = 0 \wedge a_2[n_1] = y \wedge a_3[n_1] = z \wedge a_1[n_2] = x \wedge a_3[n_2] = 0 \wedge \forall i.(n_1 \leq i < n_2 \to \exists j.i \leq j < n_2 \wedge \varphi_2(j) \wedge \forall k.(i \leq k < j \to \varphi_1(k)))$ assigns values to n_1 and n_2 and defines array values for a_1, a_2, and a_3 between bounds n_1 and n_2. Clearly, $a_1[n_1] = 0$, $a_2[n_1] = y$, $a_3[n_1] = z$, $a_1[n_2] = x$, and $a_3[n_2] = 0$. Due to their definition, $\varphi_1(j)$ and $\varphi_2(j)$ cannot be true at the same point j since $\models \varphi_1(j) \to a_2[j] > 0$ and $\models \varphi_2(j) \to a_2[j] = 0$.

Since the subformula $\forall i.(n_1 \leq i < n_2 \to \exists j.i \leq j < n_2 \wedge \varphi_2(j) \wedge \forall k.(i \leq k < j \to \varphi_1(k)))$ holds, it is then clear that there exists points $j_1, \ldots, j_l$ with $l > 0$ and $n_1 \leq j_1 < j_2 < \cdots < j_l = n_2 - 1$ such that $\varphi_2(j)$ holds at all of these points. Furthermore, at all intermediary points k not equal to one of the j_i's, $\varphi_1(k)$ has to be true. This implies that l must be equal to z (since $\varphi_1(k)$ imposes $a_3[k+1] = a_3[k]$ whereas $\varphi_2(j)$ imposes $a_3[j+1] = a_3[j] - 1$).

Let us examine the intermediary points between n_1 and j_1. Due to $a_1[n_1] = 0$, $a_2[n_1] = y$, $a_3[n_1] = z$ and $\varphi_1(k)$ being true for all k such that $n_1 \leq k < j_1$ as well as $\varphi_2(j_1)$ being true, we must have $j_1 = y + n_1$, and, for all k such that $n_1 < k \leq j_1$, we have $a_1[k] = k - n_1$, $a_2[k] = y - k + n_1$, and $a_3[k] = z$. Furthermore, since $\varphi_2(j_1)$ is true, we have $a_1[j_1+1] = y$, $a_2[j_1+1] = y$, and $a_3[j_1+1] = z - 1$. We can continue this reasoning with the intermediary points between j_1 and j_2 and so on up to j_l. At the end we get $a_3[j_l+1] = 0$ and $a_1[j_l+1] = a_1[n_2] = yl$. Since $l = z$ and $a_1[n_2] = x$, this implies $x = yz$. □

Next, we define :

$$\varphi_3(j) : a_2[j] > 0 \wedge a_1[j+1] = a_1[j]+1 \wedge a_2[j+1] = a_2[j]-1$$

$$\varphi_{x=y+z}(a_1,a_2,n_1,n_2) : n_1 < n_2 \wedge a_1[n_1] = y \wedge a_2[n_1] = z \wedge a_1[n_2] = x \wedge a_2[n_2] = 0 \wedge \\ \forall k.n_1 \le k < n_2 \rightarrow \varphi_3(k)$$

Lemma 4. *$\varphi_{x=y+z}(a_1,a_2,n_1,n_2)$ is satisfiable if and only if $x = y+z$.*

Proof. Similar to Lemma 3. □

We are now ready to reduce from the Hilbert's Tenth Problem [13]. Given a Diophantine system S, we construct a **SIL** formula Ψ_S which is satisfiable if and only if the system has a solution. Without loss of generality, we can suppose that all variables in S range over strictly positive integers. Then S can be equivalently written as a system of equations of the form $x = yz$ and $x = y+z$ by introducing fresh variables. Let $\{x_1,\ldots,x_k\}$ be the variables of these equations. We enumerate separately all equations of the form $x = yz$ and those of the form $x = y+z$. Let n_m be the number of equations of the form $x = yz$ and n_a the number of equations of the form $x = y+z$.

Let Ψ_S be the following **SIL** formula with three array symbols (a_1, a_2 and a_3):

$$\exists x_1 \ldots \exists x_k \exists m_1^1 \ldots \exists m_{n_m+n_a}^1 \exists m_1^2 \ldots \exists m_{n_m+n_a}^2 \bigwedge_{i=1}^{n_m+n_a-1} m_i^2 < m_{i+1}^1 \wedge \bigwedge_{i=1}^{n_m} \varphi_i \wedge \bigwedge_{i=1}^{n_a} \varphi_i'$$

where the formulae φ_i and φ_i' are defined as follows: Let $x_{i_1} = x_{i_2}x_{i_3}$ be the i-th multiplicative equation. Then, $\varphi_i = \varphi_{x_{i_1}=x_{i_2}x_{i_3}}(a_1,a_2,a_3,m_i^1,m_i^2)$. Let $x_{i_1} = x_{i_2}+x_{i_3}$ be the i-th additive equation. Then, $\varphi_i' = \varphi_{x_{i_1}=x_{i_2}+x_{i_3}}(a_1,a_2,m_{n_m+i}^1,m_{n_m+i}^2)$.

Lemma 5. *A Diophantine system S has a solution if and only if the corresponding formula Ψ_S is satisfiable.*

Proof. The Diophantine system S is equivalently written as a conjunction of equations of the form $x = yz$ and $x = y+z$ using variables $\{x_1,\ldots,x_k\}$. Then, the Diophantine system has a solution if and only if all equations of the form $x = yz$ and $x = y+z$ have a common solution. Since all pairs m_i^1 and m_i^2 denote disjoint intervals and using Lemmas 3 and 4, we have that all equations of the form $x = yz$ and $x = y+z$ have a common solution if and only if Ψ_S is satisfiable. □

5 Decidability of the Satisfiability Problem for $\exists^*\forall^*$-SIL

We show that the set of models of a boolean combination φ of universally quantified array property formulae of **SIL** corresponds to the set of runs of an FCADBM A_φ, defined inductively on the structure of the formula. More precisely, each array variable in φ has a corresponding counter in A_φ, and given any model of φ that associates integer values to all array entries, A_φ has a run in which the values of the counters at different points of the run match the values of the array entries at corresponding positions in the model. Since the emptiness problem is decidable for FCADBM, this leads to decidability of the satisfiability problem for $\exists^*\forall^*$-**SIL** (or equivalently, for $\forall^*$-**SIL**).

5.1 Normalisation

Before describing the translation of $\exists^*\forall^*$-**SIL** formulae into counter automata, we need to perform a simple normalisation step. Let $\varphi(\mathbf{k},\mathbf{a})$ be a **SIL** formula in the $\exists^*\forall^*$ fragment i.e., an existentially quantified boolean combination of (1) DBM conditions or modulo constraints on array-bound variables $\mathbf{k}$ and array length terms $|a|$, $a \in \mathbf{a}$, and (2) array properties of the form $\forall i\ .\ \gamma(i,\mathbf{k},|\mathbf{a}|) \rightarrow \upsilon(i,\mathbf{k},\mathbf{a})$[7]. Without losing generality, we assume that the sanity condition $\mathcal{B}(\upsilon)$ is explicitly conjoined to the guard of every array property i.e., each array property is of the form $\forall i\ .\ \gamma \wedge \mathcal{B}(\upsilon) \rightarrow \upsilon$.

A *guard expression* is a conjunction of array-bound expressions $i \sim \ell$, $\sim \in \{\leq, \geq\}$, or modulo constraints $i \equiv_s t$ where ℓ is a an array bound term, and $s,t \in \mathbb{N}$ such that $0 \leq t < s$. For a guard γ and an integer constant $c \in \mathbb{Z}$, we denote by $\gamma + c$ the guard obtained by replacing each array-bound expression $i \sim b$ by $i \sim b + c$ and each modulo constraint $i \equiv_s t$ by $i \equiv_s t'$ where $0 \leq t' < s$ and $t' \equiv_s t + c$.

The normalisation consists in performing the following steps in succession:

1. Replace each array property subformula $\forall i\ .\ \bigvee_j \gamma_j \rightarrow \bigwedge_k \upsilon_k$ by the equivalent conjunction $\bigwedge_{j,k} \forall i\ .\ \gamma_j \rightarrow \upsilon_k$ where γ_j are guard expressions and υ_k are either $a[i+n] \sim \ell$, $a[i+n] - b[i+m] \sim p$, or $i - a[i+n] \sim m$, where $m,n,p \in \mathbb{Z}$, $\sim \in \{\leq, \geq\}$ and ℓ is an array bound term.
2. Simplify each newly obtained array property subformula as follows:

$$\begin{aligned}
\forall i\ .\ \gamma \rightarrow a[i+n] \sim \ell &\ \rightsquigarrow\ \forall i\ .\ \gamma + n \rightarrow a[i] \sim \ell \\
\forall i\ .\ \gamma \rightarrow i - a[i+n] \sim m &\ \rightsquigarrow\ \forall i\ .\ \gamma + n \rightarrow i - a[i] \sim m + n \\
\forall i\ .\ \gamma \rightarrow a[i+n] - b[i+m] \sim p &\ \rightsquigarrow\ \forall i\ .\ \gamma + n \rightarrow a[i] - b[i+m-n] \sim p \text{ if } m \geq n \\
\forall i\ .\ \gamma \rightarrow a[i+n] - b[i+m] \sim p &\ \rightsquigarrow\ \forall i\ .\ \gamma + m \rightarrow b[i] - a[i+n-m] \overline{\sim} -p \text{ if } m < n
\end{aligned}$$

   where:
   - $\sim \in \{\leq, \geq\}$ and $\overline{\sim}$ is $\geq$ ($\leq$) if $\sim$ is $\leq$ ($\geq$), respectively, and
   - ℓ is an array-bound term, and $m,n,p \in \mathbb{Z}$.
3. For each array property $\psi : \forall i\ .\ \gamma(i) \rightarrow \upsilon(i)$, let $B_\psi = \{b_1, \ldots, b_n\}$ be the set of array-bound terms occurring in γ. Then replace ψ by the disjunction $\bigvee_{1 \leq i,j \leq n} \bigwedge_{1 \leq k \leq n} b_i \leq b_k \leq b_j \wedge \psi$ (one considers all possible cases of minimal and maximal values for array-bound terms), and simplify all subformulae of the form $\bigwedge_j i \leq b_j$ ($\bigwedge_j i \geq b_j$) from γ to exactly one upper (lower) bound, according to the current conjunctive clause. If the lower and upper bound that appear in γ are inconsistent with the chosen minimal and maximal value added by the transformation to ψ (i.e., the lower bound is assumed to be bigger than the upper one), we replace ψ in the concerned conjunctive clause by $\top$ as it is trivially satisfied.

[7] An array property formula with more than one universally quantified index variable occurring in the guard can be equivalently written as an array property formula whose guard has exactly one universally quantified index variable. Indeed, a formula of the form $\forall i_1, \ldots, i_n\ .\ \gamma(i_1, \ldots, i_n, \mathbf{k}, |\mathbf{a}|) \rightarrow \upsilon(i_1, \mathbf{k}, \mathbf{a})$ is equivalent to $\forall i_1\ .\ ((\exists i_2, \ldots, i_n\ .\ \gamma(i_1, \ldots, i_n, \mathbf{k}, |\mathbf{a}|)) \rightarrow \upsilon(i_1, \mathbf{k}, \mathbf{a}))$ and then the existential quantifiers in $(\exists i_2, \ldots, i_n\ .\ \gamma(i_1, \ldots, i_n, \mathbf{k}, |\mathbf{a}|))$ can be eliminated possibly adding modulo constraints on $\mathbf{k}$, $|\mathbf{a}|$ and i_1.

4. Rewrite each conjunction $\bigwedge_j i \equiv_{s_j} t_j$ occurring within the guards of array property formulae into $\bigwedge_j i \equiv_S \frac{S \cdot t_j}{s_j}$ where S is the least common multiple of s_j, and simplify the conjunction either to false (in which case the array property subformula is vacuously true), or to a formula $i \equiv_S t$. In case there is no modulo constraint within a guard, for uniformity reasons, conjoin the guard with the constraint $i \equiv_1 0$.
5. Transform each array property subformula of the form

$$\forall i \, . \, f \leq i \leq g \wedge i \equiv_s t \longrightarrow a[i] - b[i+m] \sim n$$

where $m > 1$, $n \in \mathbb{Z}$, and $0 \leq t < s$ into the following conjunction:

$$\begin{array}{c} \forall i \, . \, f \leq i \leq g \wedge i \equiv_s t \longrightarrow a[i] - \tau_1[i+1] \sim 0 \wedge \\ \bigwedge_{j=1}^{m-2} \forall i . f + j \leq i \leq g + j \wedge i \equiv_s (t+j) \bmod s \longrightarrow \tau_j[i] - \tau_{j+1}[i+1] \sim 0 \wedge \\ \forall i \, . \, f + m - 1 \leq i \leq g + m - 1 \wedge i \equiv_s (t + m - 1) \bmod s \longrightarrow \tau_{m-1}[i] - b[i+1] \sim n \end{array}$$

where $\tau_1, \tau_2, \ldots, \tau_{m-1}$ are fresh array variables. Figure 2 depicts this transformation for $\sim = \leq$ – the case $\sim = \geq$ is similar.

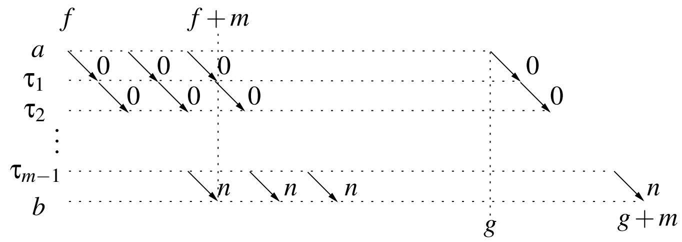

Fig. 2. Adding fresh array variables to array property formulae $\forall i \, . \, f \leq i \wedge i \leq g \wedge i \equiv_s t \rightarrow a[i] - b[i+m] \leq n$

The result of the normalisation step is a boolean combination of (1) DBM conditions or modulo constraints on array-bound variables $\mathbf{k}$ and array length terms $|a|$, $a \in \mathbf{a}$ and (2) array properties of the following form:

$$\forall i \, . \, f \leq i \leq g \; \wedge \; i \equiv_s t \rightarrow \upsilon$$

where f and g are array-bound terms, $s, t \in \mathbb{N}$, $0 \leq s < t$, and υ is one of the following:

$$(1) \; a[i] \sim \ell, \quad (2) \; i - a[i] \sim n, \quad (3) \; a[i] - b[i+1] \sim n$$

where $\sim \in \{\leq, \geq\}$, $n \in \mathbb{Z}$, and ℓ is an array-bound term.

We need the following definition to state the normal form lemma. If $X \subseteq AVar$ is a set of array variables, then $\mu \downarrow_X$ represents the restriction of $\mu : AVar \rightarrow \mathbb{Z}^*$ to the variables in X. For a formula φ of **SIL**, we denote by $[\![\varphi]\!] \downarrow_X$ the set $\{\langle \iota, \mu \downarrow_X \rangle \mid \langle \iota, \mu \rangle \models \varphi\}$.

Lemma 6. *Let $\varphi(\mathbf{k},\mathbf{a})$ be a formula of $\exists^*\forall^*$-**SIL** and $\phi(\mathbf{k},\mathbf{a},\mathbf{t})$ be the formula obtained from φ by normalisation where $\mathbf{t}$ is the set of fresh array variables added during normalisation. Then we have $[\![\varphi]\!] = [\![\phi]\!]\downarrow_{\mathbf{a}}$.*

Proof. Let $\varphi(\mathbf{k},\mathbf{a})$ be a formula of $\exists^*\forall^*$-**SIL**. Let $\varphi'(\mathbf{k},\mathbf{a})$ be the formula obtained from $\varphi(\mathbf{k},\mathbf{a})$ after Steps 1-4 of the normalisation procedure. It is clear that $\varphi'(\mathbf{k},\mathbf{a})$ is equivalent to $\varphi(\mathbf{k},\mathbf{a})$ since all transformation steps produce equivalent formulae. Furthermore, $\varphi'(\mathbf{k},\mathbf{a})$ is in the normal form except for the fact that in array properties of the form

$$\forall i \,.\, f \leq i \leq g \wedge i \equiv_s t \;\longrightarrow\; a[i] - b[i+m] \sim n$$

where $n \in \mathbb{Z}$, and $0 \leq t < s$, it may be the case that $m > 1$.

Let $\phi(\mathbf{k},\mathbf{a},\mathbf{t})$ be the formula obtained from $\varphi'(\mathbf{k},\mathbf{a})$ after Step 5 where $\mathbf{t}$ are all the additional array variables added in Step 5. It is clear that $\phi(\mathbf{k},\mathbf{a},\mathbf{t})$ is in the normal form. Furthermore, due to the role of the extra array variables, it is easy to see that a model of $\phi(\mathbf{k},\mathbf{a},\mathbf{t})$ restricted to $\mathbf{k}$ and $\mathbf{a}$ is a model of $\varphi'(\mathbf{k},\mathbf{a})$ (and of $\varphi(\mathbf{k},\mathbf{a})$ as well) and that, for a model of $\varphi'(\mathbf{k},\mathbf{a})$ (as well as of $\varphi(\mathbf{k},\mathbf{a})$), one can give a corresponding model of $\phi(\mathbf{k},\mathbf{a},\mathbf{t})$. $\square$

5.2 Translating Normalised Formulae into FCADBM.

Let $\varphi(\mathbf{k},\mathbf{a})$ be an $\exists^*\forall^*$-**SIL** formula that is already normalised as in the previous. The automaton encoding the models of φ is in fact a product $\mathcal{A}_\varphi = A_\varphi \otimes A_{tick}$, where A_φ is defined inductively on the structure of φ, and A_{tick} is a generic FCADBM, defined next. Both A_φ and A_{tick} (and, implicitly $\mathcal{A}_\varphi$) work with the set of counters $\mathbf{x} = \{x_k \mid k \in \mathbf{k}\} \cup \{x_{|a|} \mid a \in \mathbf{a}\} \cup \{x_a \mid a \in \mathbf{a}\} \cup \{x_{tick}\}$, where:

- x_k and $x_{|a|}$ are parameters corresponding to array-bound variables, i.e., their values do not change during the runs of $\mathcal{A}_\varphi$,
- x_a are counters corresponding to the array symbols, and
- x_{tick} is a special counter that is initialised to zero and incremented by each transition.

The main intuition behind the automata construction is that, for each model $\langle \iota, \mu \rangle$ of φ, there exists a run of $\mathcal{A}_\varphi$ such that, for each array symbol $a \in \mathbf{a}$, the value $\mu(a)_n$ equals the value of x_a when x_{tick} equals n, for all $0 \leq n < |a|$. The reason behind defining $\mathcal{A}_\varphi$ as the product of A_φ and A_{tick} is that the use of negation within φ, which involves complementation on the automata level, may not affect the flow of ticks, just the way they are dealt with within the guards. For this reason, A_φ can only read x_τ, while A_{tick} is the one updating it.

Formally, let $A_{tick} = \langle \mathbf{x}, \{q_0, q_{tick}\}, \{q_0\}, \rightarrow_{tick}, \{q_{tick}\} \rangle$, where

$$q_0 \xrightarrow{x_{tick}=0 \,\wedge\, x'_{tick}=x_{tick}+1 \,\wedge\, \bigwedge_{k\in\mathbf{k}} x'_k=x_k \,\wedge\, \bigwedge_{a\in\mathbf{a}} x'_{|a|}=x_{|a|}} q_{tick}$$

and

$$q_{tick} \xrightarrow{x'_{tick}=x_{tick}+1 \,\wedge\, \bigwedge_{k\in\mathbf{k}} x'_k=x_k \,\wedge\, \bigwedge_{a\in\mathbf{a}} x'_{|a|}=x_{|a|}} q_{tick}$$

are the only transitions rules. The construction of A_φ is recursive on the structure of φ:

– if φ is a DBM constraint or modulo constraint θ on array-bound terms, let $A_\varphi = \langle \mathbf{x}, \{q_0, q_1\}, \{q_0\}, \rightarrow, \{q_1\}\rangle$ where the transitions rules are $q_1 \xrightarrow{\top} q_1$ and $q_0 \xrightarrow{\overline{\theta}} q_1$, and $\overline{\theta}$ is obtained from the constraint θ by replacing all occurrences of $k \in \mathbf{k}$ by x_k, and all occurrences of $|a|$, $a \in \mathbf{a}$, by $x_{|a|}$.
– if $\varphi = \neg\psi$, let $A_\varphi = \overline{A_\psi}$,
– if $\varphi = \psi_1 \wedge \psi_2$, let $A_\varphi = A_{\psi_1} \otimes A_{\psi_2}$,
– if $\varphi = \psi_1 \vee \psi_2$, let $A_\varphi = \overline{\overline{A_{\psi_1}} \otimes \overline{A_{\psi_2}}}$.
– if φ is an array property, A_φ is defined below, according to the type of the value expression occurring on the right hand side of the implication.

Let $\varphi : \forall i \,.\, f \leq i \leq g \;\wedge\; i \equiv_s t \rightarrow \upsilon$ be an array property subformula after normalisation. Figure 3 gives the counter automaton A_φ for such a subformula. The formal definition of $A_\varphi = \langle \mathbf{x}, Q, I, \rightarrow, F\rangle$ follows:

– $Q = \{q_i \mid 0 \leq i < s\} \,\cup\, \{r_i \mid 0 \leq i < s\} \,\cup\, \{q_f\}$, $I = \{q_0\}$, and $F = \{q_f\}$.
– the transition rules of A_φ are as follows, for all $0 \leq i < s$:

$$q_i \xrightarrow{x_{tick} < \overline{f}-1} q_{(i+1) \bmod s} \qquad\qquad q_i \xrightarrow{x_{tick} = \overline{f}-1} r_{(i+1) \bmod s}$$

$$r_i \xrightarrow{\overline{f} \leq x_{tick} \leq \overline{g}} r_{(i+1) \bmod s} \text{ if } i \neq t \qquad\qquad r_t \xrightarrow{\overline{\upsilon} \,\wedge\, \overline{f} \leq x_{tick} \leq \overline{g}} r_{(t+1) \bmod s}$$

$$r_i \xrightarrow{x_{tick} > \overline{g}} q_f \qquad\qquad q_f \xrightarrow{\top} q_f$$

$$q_0 \xrightarrow{x_{tick}=0 \,\wedge\, \overline{\upsilon} \,\wedge\, \overline{f} \leq x_{tick} \leq \overline{g}} r_{1 \bmod s} \text{ if } t = 0 \qquad q_0 \xrightarrow{x_{tick}=0 \,\wedge\, \overline{f} \leq x_{tick} \leq \overline{g}} r_{1 \bmod s} \text{ if } t \neq 0$$

Here $\overline{\upsilon}$ is defined by:

- $\overline{\upsilon} \triangleq x_a \sim \overline{\ell}$ if υ is $a[i] \sim \ell$ where $\overline{\ell}$ is obtained from ℓ by replacing each occurrence of $k \in \mathbf{k}$ by x_k and each occurrence of $|a|$ by $x_{|a|}$, $a \in \mathbf{a}$,
- $\overline{\upsilon} \triangleq x_{tick} - x_a \sim n$ if υ is $i - a[i] \sim n$, and
- $\overline{\upsilon} \triangleq x_a - x'_b \sim n$ if υ is $a[i] - b[i+1] \sim n$.

Further, $\overline{f}$ ($\overline{g}$) are obtained from f (g) by replacing each $k \in \mathbf{k}$ by x_k and each $|a|$, $a \in \mathbf{a}$, by $x_{|a|}$, respectively.

Notice that A_φ is always deterministic. This is because the automata for array property formulae are deterministic in the use of the x_{tick} counter, complementation preserves determinism, and composition of two deterministic FCADBM results in a deterministic FCADBM.

Let $\varphi(\mathbf{k}, \mathbf{a})$ be a normalised $\exists^*\forall^*$-**SIL** formula, and $\mathcal{A}_\varphi = A_\varphi \otimes A_{tick}$ be the deterministic FCADBM whose construction was given in the previous. We define the following relation between valuations $\langle \iota, \mu\rangle \in [\![\varphi]\!]$ and traces $\sigma \in Tr(\mathcal{A}_\varphi)$, denoted $\langle \iota, \mu\rangle \equiv \sigma$, if and only if:

1. for all $k \in \mathbf{k}$, $\iota(k) = \sigma_0(x_k)$,
2. for all $a \in \mathbf{a}$, $\iota(|a|) = \sigma_0(x_{|a|}) = |\mu(a)| \leq |\sigma|$ and $\mu(a)_i = \sigma_i(x_a)$, $0 \leq i < |\mu(a)|$.

The following lemma establishes correctness of our construction:

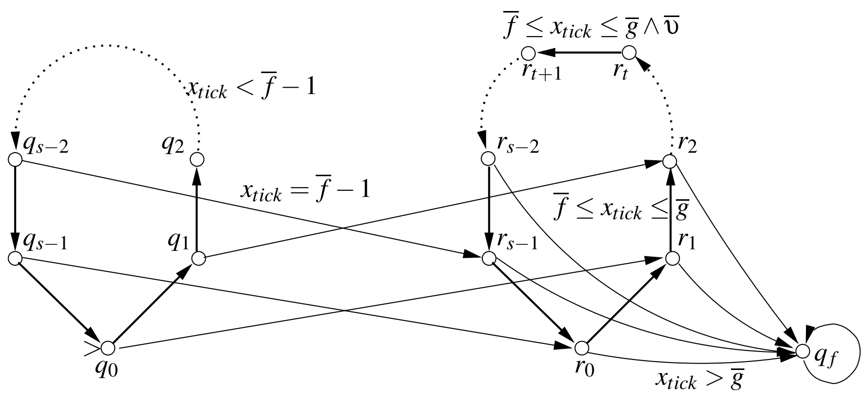

Fig. 3. The FCADBM for the formula $\forall i \,.\, f \leq i \leq g \;\wedge\; i \equiv_s t \rightarrow \upsilon$

Lemma 7. *Let $\varphi(\mathbf{k},\mathbf{a})$ be a normalised $\exists^*\forall^*$-**SIL** formula, and $\mathcal{A}_\varphi$ be its corresponding FCADBM. Then for each valuation $\langle \iota,\mu \rangle \in [\![\varphi]\!]$ there exist a trace $\sigma \in Tr(\mathcal{A}_\varphi)$ such that $\langle \iota,\mu \rangle \equiv \sigma$. Dually, for each trace $\sigma \in Tr(\mathcal{A}_\varphi)$ there exists a valuation $\langle \iota,\mu \rangle \in [\![\varphi]\!]$ such that $\langle \iota,\mu \rangle \equiv \sigma$.*

Proof. We first prove the following claim.

Claim. For each $\sigma \in Tr(A_{tick})$ and each $\langle \iota,\mu \rangle \in (BVar \rightarrow \mathbb{Z}) \times (AVar \rightarrow \mathbb{Z}^*)$ such that $\langle \iota,\mu \rangle \equiv \sigma$, we have $\langle \iota,\mu \rangle \in [\![\varphi]\!]$ if and only if $\sigma \in Tr(A_\varphi)$.

Proof. By induction on the structure of φ.

- φ is a DBM constraint or modulo constraint on array-bound terms. Let $\langle \iota,\mu \rangle \in [\![\varphi]\!]$. Since $\langle \iota,\mu \rangle \equiv \sigma$, we have $\sigma_0(x_k) = \iota(k)$ for all $k \in \mathbf{k}$, and $\sigma_0(x_{|a|}) = \iota(|a|)$ for all $a \in \mathbf{a}$. Then $\overline{\varphi}$ holds if we substitute each x_k with $\sigma_0(x_k)$ and each $x_{|a|}$ with $\sigma_0(x_{|a|})$, hence the transition $q_0 \xrightarrow{\overline{\varphi}} q_1$ is executable in A_φ. Moreover, once q_1 is reached, the control stays in q_1 forever. Therefore, we can deduce that $\sigma \in Tr(A_\varphi)$. Dually, if $\sigma \in Tr(A_\varphi)$, we have that $\overline{\varphi}$ holds if we substitute each x_k with $\sigma_0(x_k)$ and each $x_{|a|}$ with $\sigma_0(x_{|a|})$. Since $\langle \iota,\mu \rangle \equiv \sigma$, we obtain $\langle \iota,\mu \rangle \in [\![\varphi]\!]$.
- φ is a property formula of the form $\forall i \,.\, f \leq i \leq g \;\wedge\; i \equiv_s t \rightarrow \upsilon$. Let $I_\sigma(\overline{f})$ denote the value of the term $\overline{f}$, when replacing x_k by $\sigma_0(x_k)$, for all $k \in \mathbf{k}$, and $x_{|a|}$ by $\sigma_0(x_{|a|})$, for all $a \in \mathbf{a}$. Since $\sigma \in Tr(A_{tick})$, $I_\sigma(\overline{f})$ is independent of the position in σ. Moreover, we have $\sigma_i(x_{tick}) = i$, for all $0 \leq i \leq |\sigma|$.

  "$\Rightarrow$" Let $\langle \iota,\mu \rangle \in [\![\varphi]\!]$. Since $\langle \iota,\mu \rangle \equiv \sigma$, we have for all $a \in \mathbf{a}$, for all $0 \leq i < |\mu(a)|$, $\mu(a)_i = \sigma_i(x_a)$ and $|\mu(a)| \leq |\sigma|$. Since in the normalisation step we assumed the sanity conditions to be conjoined to the guards, we have $0 \leq I_{\iota,\mu}(f) \leq I_{\iota,\mu}(g) \leq |\mu(a)|$. We consider two cases:
  1. $I_{\iota,\mu}(f) = 0$. Since $\langle \iota,\mu \rangle \equiv \sigma$, $I_\sigma(\overline{f}) = I_{\iota,\mu}(f) = \sigma_0(x_{tick}) = 0$. Hence, if $t \neq 0$, the transition $q_0 \rightarrow r_1$ is clearly executable. Moreover, for the case of $t = 0$,

since $\langle \iota, \mu \rangle \equiv \sigma$, we have $\sigma_0(x_a) = \mu(a)_0$. From $\langle \iota, \mu \rangle \in [\![\varphi]\!]$, we see that υ must hold at index 0, and therefore the guard $\overline{\upsilon}$ holds in the initial configuration, and so the transition $q_0 \rightarrow r_1$ is executable in this case as well. Next, it is easy to see that control stays at $r_{0\ldots s-1}$ for all $1 \leq i \leq I_\sigma(\overline{g}) = I_{\iota,\mu}(g)$, due to the guard $\overline{f} \leq x_{tick} \leq \overline{g}$ on all transitions $r_i \rightarrow r_{(i+1) \mod s}$. Moreover, we have $\sigma_i(x_{tick}) \equiv_s k$ whenever the control is at r_k for all $1 \leq i \leq I_\sigma(\overline{g})$ and $0 \leq k < s$. Since $\langle \iota, \mu \rangle \equiv \sigma$, we have $\sigma_i(x_a) = \mu(a)_i$ for all $1 \leq i \leq I_\sigma(\overline{g})$. From $\langle \iota, \mu \rangle \in [\![\varphi]\!]$, we deduce that υ must hold for all values of i such that $i \equiv_s t$, therefore the guard $\overline{\upsilon}$ holds on the transitions $q_0 \rightarrow r_1$ or $r_t \rightarrow r_{t+1}$, depending on whether $t = 0$ or not. Finally, $\sigma \in Tr(A_\varphi)$ because for $i > I_\sigma(g)$, the control reaches q_f, and it stays in q_f until the end of the trace.

2. $I_{\iota,\mu}(f) > 0$. In this case, the control goes through $q_{0\ldots s-1}$ until $\sigma_i(x_{tick}) = I_{\iota,\mu}(f) - 1$, before moving to $r_{0\ldots s-1}$ by a transition $q_i \rightarrow r_{(i+1) \mod s}$. The argument for this case is similar to the above.

"$\Leftarrow$" Let $\sigma \in Tr(A_\varphi)$. Then A_φ has a run from q_0 to q_f such that σ is the sequence of counter valuations along the run. We consider two cases:

1. $I_\sigma(\overline{f}) = 0$. Since $\sigma \in Tr(A_{tick})$, we have that $\sigma_i(x_{tick}) = i$ for all $0 \leq i \leq |\sigma|$. In this case, the first transition of the run is $q_0 \rightarrow r_1$. Therefore the run visits the control states $r_{0\ldots s-1}$ between positions 1 and $I_\sigma(\overline{g})$. It can easily be seen that $\sigma_i(x_{tick}) \equiv_s k$ whenever the control is at r_k for all $1 \leq i \leq I_\sigma(\overline{g})$ and $0 \leq k < s$. Since $\langle \iota, \mu \rangle \equiv \sigma$, we have $\sigma_i(x_a) = \mu(a)_i$ for all $0 \leq i \leq I_\sigma(\overline{g})$. Since $\overline{\upsilon}$ holds on the transitions $q_0 \rightarrow r_1$ or $r_t \rightarrow r_{t+1}$, depending on whether $t = 0$ or not, then υ must hold on $\langle \iota, \mu \rangle$ at all positions $i \equiv_s t$. Therefore $\langle \iota, \mu \rangle \in [\![\varphi]\!]$.
2. $I_\sigma(\overline{f}) > 0$. In this case, the first transition of the run is $q_0 \rightarrow q_1$, and the run visits $q_{0,\ldots,s-1}$ between positions 0 and $I_\sigma(\overline{f})$, before moving to $r_{0,\ldots,s-1}$, from $I_\sigma(\overline{f})$ to $I_\sigma(\overline{g})$. The proof that $\langle \iota, \mu \rangle \in [\![\varphi]\!]$ follows an argument similar to the above.

- $\varphi = \varphi_1 \wedge \varphi_2$. Let $\langle \iota, \mu \rangle \in [\![\varphi_1 \wedge \varphi_2]\!] = [\![\varphi_1]\!] \cap [\![\varphi_2]\!]$. By the induction hypothesis and by Lemma 1, we obtain $\sigma \in Tr(A_{\varphi_1}) \cap Tr(A_{\varphi_2}) = Tr(A_{\varphi_1} \otimes A_{\varphi_2})$. By the definition of the automaton, $A_{\varphi_1} \otimes A_{\varphi_2} = A_{\varphi_1 \wedge \varphi_2}$. Dually, $\sigma \in Tr(A_{\varphi_1 \wedge \varphi_2})$ implies $\sigma \in Tr(A_{\varphi_1}) \cap Tr(A_{\varphi_2})$. By the induction hypothesis, we have $\langle \iota, \mu \rangle \in [\![\varphi_1]\!] \cap [\![\varphi_2]\!] = [\![\varphi_1 \wedge \varphi_2]\!]$.
- $\varphi = \neg\psi$. Let $\langle \iota, \mu \rangle \in [\![\neg\psi]\!]$, i.e., $\langle \iota, \mu \rangle \notin [\![\psi]\!]$. By the induction hypothesis, we have $\sigma \notin Tr(A_\psi)$, and by Lemma 2, $\sigma \in Tr(\overline{A_\psi})$. By the definition of the automaton, $\overline{A_\psi} = A_{\neg\psi}$. Dually, $\sigma \in Tr(A_{\neg\psi})$ implies $\sigma \notin Tr(A_\psi)$. By the induction hypothesis, $\langle \iota, \mu \rangle \notin [\![\psi]\!]$, hence $\langle \iota, \mu \rangle \in [\![\neg\psi]\!]$.
- $\varphi = \varphi_1 \vee \varphi_2$. Correctness of constructing $A_{\varphi_1 \vee \varphi_2}$ as $\overline{\overline{A_{\varphi_1}} \otimes \overline{A_{\varphi_2}}}$ is an immediate consequence of the two points above. $\square$

For the first part, we take any valuation $\langle \iota, \mu \rangle \in [\![\varphi]\!]$ and build trivially a trace $\sigma \in Tr(A_{tick})$ such that $\langle \iota, \mu \rangle \equiv \sigma$. Since $\langle \iota, \mu \rangle \in [\![\varphi]\!]$, we obtain $\sigma \in Tr(A_\varphi)$ by the claim, hence $\sigma \in Tr(\mathcal{A}_\varphi)$. Dually, for each trace $\sigma \in Tr(\mathcal{A}_\varphi) = Tr(A_\varphi) \cap Tr(A_{tick})$, we define a valuation $\langle \iota, \mu \rangle$ such that $\langle \iota, \mu \rangle \equiv \sigma$. The claim then ensures that $\langle \iota, \mu \rangle \in [\![\varphi]\!]$. $\square$

Theorem 2. *The satisfiability problem is decidable for the $\exists^*\forall^*$ fragment of* **SIL**.

Proof. Let $\varphi(\mathbf{k}, \mathbf{a})$ be a formula of $\exists^*\forall^*$-**SIL**. By normalisation, we obtain a formula $\phi(\mathbf{k}, \mathbf{a}, \mathbf{t})$ where $\mathbf{t}$ is the set of fresh array variables added during normalisation. Then, by Lemma 6, we have $[\![\varphi]\!] = [\![\phi]\!] \downarrow_{\mathbf{a}}$. To check satisfiability of φ, it is therefore enough to check satisfiability of ϕ. By Lemma 7, ϕ is satisfiable if and only if the language of the corresponding automaton $\mathcal{A}_\phi$ is not empty. This is decidable by Theorem 1. $\square$

6 Conclusion

In this paper we have introduced a logic over integer arrays based on universally quantified difference bound constraints on array elements situated within a constant sized window. We have shown that the logic is undecidable for formulae with quantifier prefix in the language $\forall^*\exists^*\forall^*$, and that the $\exists^*\forall^*$ fragment is decidable. This is shown with an automata-theoretic argument by constructing, for a given formula, a corresponding equivalent counter automaton whose emptiness problem is decidable. The translation of formulae into counter automata takes advantage of the fact that only one index is used in the difference bound constraints on array values, making the decision procedure for the logic simple and efficient. Future work involves automatic invariant generation for programs handling arrays, as well as implementation and experimental evaluation of the method.

References

1. A. Armando, S. Ranise, and M. Rusinowitch. Uniform Derivation of Decision Procedures by Superposition. In *Proc of CSL'01*, volume 2142 of *LNCS*, 2001.
2. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L.Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *CAV'01*, volume 2102 of *LNCS*. Springer, 2001.
3. A. Bouajjani, Y. Jurski, and M. Sighireanu. A Generic Framework for Reasoning About Dynamic Networks of Infinite-State Processes. In *TACAS'07*, volume 4424 of *LNCS*. Springer, 2007.
4. M. Bozga, R. Iosif, and Y. Lakhnech. Flat Parametric Counter Automata. In *Proc. of ICALP'06*, volume 4052 of *LNCS*. Springer, 2006.
5. A.R. Bradley, Z. Manna, and H.B. Sipma. What's Decidable About Arrays? In *Proc. of VMCAI'06*, volume 3855 of *LNCS*. Springer, 2006.
6. H. Comon and Y. Jurski. Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In *Proc. of CAV'98*, volume 1427 of *LNCS*. Springer, 1998.
7. The FLATA Toolset. http://www-verimag.imag.fr/˜async/FLATA/flata.html.
8. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decision Procedures for Extensions of the Theory of Arrays. *Annals of Mathematics and Artificial Intelligence*, 50, 2007.
9. P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In *Proc. of FOSSACS'08*, volume 4962 of *LNCS*. Springer, 2008.
10. J. Jaffar. Presburger Arithmetic with Array Segments. *Information Processing Letters*, 12, 1981.
11. J. King. *A Program Verifier*. PhD thesis, Carnegie Mellon University, 1969.
12. P. Mateti. A Decision Procedure for the Correctness of a Class of Programs. *Journal of the ACM*, 28(2), 1980.

13. Yuri Matiyasevich. Enumerable sets are diophantine. *Journal of Sovietic Mathematics*, 11:354 – 358, 1970.
14. J. McCarthy. Towards a Mathematical Science of Computation. In *IFIP Congress*, 1962.
15. A. Stump, C.W. Barrett, D.L. Dill, and J.R. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *Proc. of LICS'01*, 2001.
16. N. Suzuki and D. Jefferson. Verification Decidability of Presburger Array Programs. *Journal of the ACM*, 27(1), 1980.