A Logic of Singly Indexed Arrays
Peter Habermehl, Radu Iosif, Tomáš Vojnar

To cite this version:
Peter Habermehl, Radu Iosif, Tomáš Vojnar. A Logic of Singly Indexed Arrays. Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Nov 2008, Doha, Qatar. 10.1007/978-3-540-89439-1_39. hal-01418915

HAL Id: hal-01418915
https://hal.archives-ouvertes.fr/hal-01418915
Submitted on 17 Dec 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Logic of Singly Indexed Arrays*

Peter Habermehl¹, Radu Iosif², and Tomáš Vojnar³

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. We show that the satisfiability problem for the logic is undecidable for formulae with a quantifier prefix {∃, ∀} ∗ ∀ ∗ ∃ ∀ ∗. For formulae with quantifier prefixes in the ∃ ∀ ∗ fragment, decidability is established by an automata-theoretic argument. For each formula in the ∃ ∀ ∗ 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 ∃ ∀ ∗ 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 ∀ i . γ(i) → a₁[i + k₁] − a₂[i + k₂] ≤ m where γ is a positive boolean combination of bound and modulo constraints on the index variable i, a₁ and a₂ are array symbols, and k₁, k₂, m ∈ ℤ are integer constants. Hence the name of Single Index Logic (SIL). Note that SIL can also be viewed as a fragment of Presburger arithmetic extended with uninterpreted functions mapping naturals to integers.

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 [10], 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 [10].

Working directly with singly-indexed formulae allows to devise a simple and efficient decision procedure for the satisfiability problem of the ∃ ∀ ∗ fragment of SIL,

* The work was supported by the French Ministry of Research (RNTL project A VERILES), the Czech Grant Agency (projects 102/07/0322, 102/05/H050), the Czech-French Barrande project MEB 0208-40, and the Czech Ministry of Education by project MSM 0021630528.
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 $\exists^*\forall^*-\text{SIL}$ 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 $\forall i . \gamma(i) \rightarrow \nu(i)$ 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 [15], 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 [12]. Since then, various quantifier-free decidable logics on arrays have been considered—e.g., [17, 13, 11, 16, 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 $\exists^n \forall^m$ logic on multisets of elements with associated data values is considered.

The closest in spirit to the present paper is our previous work in [10]. There, we established decidability of formulae in the $\exists^n \forall^m$ 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 [10].

On one hand, the basic propositions from [10], 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 [10]. Here, we prove 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 \textbf{SIL}. Next, Section 4 gives the undecidability result for the entire logic, while Section 5 proves decidability of the satisfiability for the $\exists^n \forall^m$ fragment, by translation to deterministic FCADBM. Finally, Section 6 presents some concluding remarks. For space reasons, most of the proofs are deferred to [9].

2 Counter Automata

Given a formula $\varphi$, 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 $\varphi$, we denote by $\models \varphi$ the fact that $\varphi$ 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) $x − y ≤ c$, (2) $x ≤ c$, or (3) $x ≥ c$, where $c ∈ \mathbb{Z}$ 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., $\neg(x − y ≤ c) \iff y − x ≤ −c + 1$ and $\neg(x ≤ c) \iff x ≥ c + 1$. In particular, the negation of $\top$ is the empty disjunction, denoted as $\bot$ (false).

A counter automaton (CA) is a tuple $A = (x, Q, I, →, F)$ where:

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

A configuration of a counter automaton $A$ is a pair $(q, v)$ where $q ∈ Q$ is a control state, and $v : x → \mathbb{Z}$ is a valuation of the counters in $x$. For a configuration $c = (q, v)$, we
designate by \( \text{val}(c) = v \) the valuation of the counters in \( c \). A configuration \((q', v')\) is an immediate successor of \((q, v)\) if and only if \( A \) has a transition rule \( q \xrightarrow{\psi(q', x')} q' \) such that \( \models \psi(v(x), v'(x')) \). A configuration \( c \) is a successor of another configuration \( c' \) if and only if there exists a finite sequence of configurations \( c = c_1c_2...c_n = c' \) 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 \( c_1c_2...c_n \) with \( c_1 = (q, v) \), \( c_n = (q', v') \) for some valuations \( v, v' : x \rightarrow \mathbb{Z} \), and \( c_{i+1} \) is an immediate successor of \( c_i \), for all \( 1 \leq i < n \). Let \( R(A) \) denote the set of runs of \( A \) from some initial state \( q_0 \in I \) to some final state \( q_f \in F \), and \( Tr(A) = \{ \text{val}(c_1)\text{val}(c_2)\ldots\text{val}(c_n) \mid c_1c_2...c_n \in R(A) \} \) be its set of valuation traces. If \( z \subseteq x \) is a subset of the counters of \( A \) and \( v : x \rightarrow \mathbb{Z} \) is a valuation of its counters, let \( v \downharpoonright_z \) be the restriction of \( v \) to the counters in \( z \). If \( c = (q, v) \) is a configuration of \( A \), we denote \( c \downharpoonright_z = (q, v \downharpoonright_z) \) and \( Tr(A) \downharpoonright_z = \{ \text{val}(c_1)\downharpoonright_z \text{val}(c_2)\downharpoonright_z \ldots\text{val}(c_n)\downharpoonright_z \mid c_1c_2...c_n \in R(A) \} \).

A counter \( z \in x \) is called a parameter of \( A \) if and only if, for each \( \sigma = v_1...v_n \in Tr(A) \), we have \( v_1(z) = \ldots = v_n(z) \), 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 \( q_1q_2...q_n \) of control states such that, for all \( 1 \leq i < n \), there exists a transition rule \( q_i \xrightarrow{\psi} q_{i+1} \). 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 \( q \xrightarrow{\psi_1} q' \) and \( q \xrightarrow{\psi_2} q'' \), we have \( \models \neg(\psi_1 \land \psi_2) \). It is easy to prove that, given a deterministic counter automaton \( A \), for each sequence of valuations \( v_1v_2...v_n \in Tr(A) \) there exists exactly one control path \( q_1q_2...q_n \) such that \((q_0, v_1)(q_1, v_2)...(q_{n-1}, v_n) \in R(A) \).

### 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 \( z \equiv t \mod i \) where \( 0 \leq t < s \).

An extension of this class has been studied in [10]. Using results of [6, 4], [10] shows that, given a CA \( A = \langle x, Q, I, \rightarrow, F \rangle \) in the class it considers and a pair of control states \( q, q' \in Q \), the set \( V_{q,q'} = \{ (v, v') \in (x \rightarrow \mathbb{Z})^2 \mid A \text{ has a run from } (q,v) \text{ to } (q',v') \} \) is Presburger-definable. As an immediate consequence, the emptiness problem for \( A \), i.e., \( Tr(A) \not\equiv 0 \), 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 \( A_i = \langle x, Q_i, \{ q_0 \}, \rightarrow_i, F_i \rangle, i = 1, 2, \ldots \).
be two deterministic FCADBM with the same set of counters. Note that this is not a restriction as one can add unrestricted counters without changing the behaviour of a CA. We first show closure under intersection by defining the CA $A_1 \otimes A_2 = (x, Q_1 \times Q_2, \{(q_0, q_{02})\}, \rightarrow, F_1 \times F_2)$ where $(q_1, q_2) \xRightarrow{\psi} (q'_1, q'_2) \iff q_1 \xRightarrow{\psi_1} q'_1, q_2 \xRightarrow{\psi_2} q'_2$, and $q \models \psi \iff \psi_1 \land \psi_1$. The next lemma proves the correctness of our construction.

**Lemma 1.** For any two deterministic FCADBM $A_i = (x, Q_i, \{q_0\}, \rightarrow_i, F_i)$, $i = 1, 2$, $A_1 \otimes A_2$ is a deterministic FCADBM, and $Tr(A_1 \otimes A_2) = Tr(A_1) \cap Tr(A_2)$.

Let $A = (x, Q, I, \rightarrow, F)$ be a deterministic FCADBM. Then we define $\overline{A} = (x, Q \cup \{q_s\}, I, \rightarrow', (Q \setminus F) \cup \{q_s\})$ where $q_s \not\in Q$ is a fresh state. The transition relation $\rightarrow'$ is defined as follows. For a control state $q \in Q$, let $O_A(q) = \bigvee q^a_q q'$.\(^4\) Then, we have:

- $q \xRightarrow{\psi} q_s, q \xRightarrow{\psi} q'$ for each $q \xRightarrow{\psi} q'$, and
- $q \xRightarrow{\psi_i} q_s$, for all $1 \leq i \leq k$, where $\psi_i$ are (unique) conjunctions of DBMs and modulo constraints\(^5\) such that $\models \neg O_A(q) \rightarrow \bigvee_{i=1}^k \psi_i$ and $\models \neg (\psi_i \land \psi_j)$ for $i \neq j, 1 \leq i, j \leq k$.

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 $A = (x, Q, \{q_0\}, \rightarrow, F)$, for any finite sequence of valuations $\sigma \in (x \mapsto \mathbb{Z})^*$, we have $\sigma \in Tr(A)$ if and only if $\sigma \not\in Tr(\overline{A})$.

### 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 $true$.\(^7\)

In the following, we will write $f \leq i \leq g$ instead of $f \leq i \land i \leq g$, $i < f$ instead of $i \leq f - 1$, $i = f$ instead of $f \leq i \leq f$, $q_1 \lor q_2$ instead of $\neg (\neg q_1 \land \neg q_2)$, and $\forall i \cdot u(i)$ instead of $\forall i \cdot u(i)$. 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 $q$ on terms $a_1[\ell_1], \ldots, a_n[\ell_n]$, as a shorthand for $(\bigwedge_{k=1}^n \forall j, j = \ell_k \rightarrow a_k[j] = \ell_k) \land \psi[l_1/a_1(\ell_1), \ldots, l_n/a_n(\ell_n)]$, where $l_1, \ldots, l_n$ are fresh array-bound variables.

---

\(^4\) 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 $\bigvee_{r \in \{0, \ldots, s-1\} \setminus \{t\}} z \equiv_s r'$.

\(^6\) 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 $A_1 \otimes A_2$. 

5
Let us consider the Single Index Logic (SIL)

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 \( \forall i . \gamma \rightarrow v \), i.e., we require \( FV(v) \cap IVar = \{ i \} \). Hence the name Single Index Logic (SIL). Note that this does not restrict the expressive power w.r.t. the logic considered in [10]. One can always circumvent this restriction by using the method from [10] 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 \( \forall i . \gamma \rightarrow v \), 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, disjunctive terms are not allowed on the right hand side of implications in array properties: Intuitively, allowing disjunctions in value expressions would allow one to code 2-counter machines with possibly nested loops (as shown already in [10]).

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

- \( B(a[i + n] \leq B) = B(b \leq a[i + n]) = 0 \leq i + n < |a| \)
- \( B(i - a[i + n] \leq m) = B(a[i + n] - i \leq m) = 0 \leq i + n < |a| \)
- \( B(a[i + n] - b[i + m] \leq p) = 0 \leq i + n < |a| \land 0 \leq i + m < |b| \)
- \( B(v_1 \land v_2) = B(v_1) \land B(v_2) \)

Intuitively, \( B(v) \) is the conjunction of all sanity conditions needed in order for the array accesses in \( v \) 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 \( (\iota, \mu) \) where \( \iota : BVar \cup IVar \rightarrow \mathbb{Z} \) 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 $\sigma_i$ its $i$-th element.

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

$$I_\mu([a]) = |\mu(a)|$$

$$I_\mu(a[i+n]) = \mu(a)_{\langle i(n+i) + n \rangle}$$

$\langle t, \mu \rangle \models a[i+n] \leq B$ $\iff$ $I_\mu(a[i+n]) \leq I_\mu(B)$

$\langle t, \mu \rangle \models A_1 - A_2 \leq n$ $\iff$ $I_\mu(A_1) - I_\mu(A_2) \leq n$

$\langle t, \mu \rangle \models \forall i.G$ $\iff$ $\forall n \in \mathbb{Z} : \langle t[i+n], \mu \rangle \models G \land B(V) \rightarrow V$

$\langle t, \mu \rangle \models \exists i.F$ $\iff$ $\langle t[i-n], \mu \rangle \models F$ for some $n \in \mathbb{N}$

Notice that the semantics of an array property formula $\forall i. G \rightarrow V$ 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 $B(V)$. 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 $\varphi(k,a)$ is a valuation $\langle t, \mu \rangle$ such that the formula obtained by interpreting each variable $k \in k$ as $t(k)$ and each array variable $a \in a$ as $\mu(a)$ is logically valid: $\langle t, \mu \rangle \models \varphi$. We define $[\varphi] = \{ \langle t, \mu \rangle \mid \langle t, \mu \rangle \models \varphi \}$. A formula is said to be:

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

With these definitions, the satisfiability problem asks, given a formula $\varphi$ if it has at least one model. Without losing generality, for the satisfiability problem, we can assume that the quantifier prefix of $\varphi$ (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 $\mu$ mapping $a$ to a finite sequence of 0’s, i.e., $\mu(a) \in 0^*$. It is semantically equivalent to $\forall i. 0 \leq i < |a| \rightarrow a[i] = 0$, in which the range of $i$ has been made explicit.

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

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 $\forall i. i \equiv_2 0 \rightarrow a[i] = 0 \land \forall i. i \equiv_2 1 \rightarrow a[i] = 1$ 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
Alternatively, this can be specified as \( a[0] = 1 \land \forall i . \ a[i + 1] = a[i] + 1 \) 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... is used).

Checking verification conditions for array manipulating programs. The decision procedure for checking satisfiability of \( \text{SIL} \) 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.

\[
\begin{align*}
\{ & \{ a = a_0 \land \forall j . a[j] = a_0[j] \} \\
\text{x=a(0);} & \\
\text{for (i=0; i<|a|; i++)} & \\
\{ & \{ x = a_0[0] \land \forall j . 0 \leq j < i \rightarrow a[j] = a_0[j + 1] \land \forall j . i \leq j < |a| \rightarrow a[j] = a_0[j] \} \\
& \land a[i]=a[i+1]; \\
& a[|a|-1]=x; \\
& \{ a[|a| - 1] = a_0[0] \land \forall j . 0 \leq j < |a| - 1 \rightarrow a[j] = a_0[j + 1] \} \}
\end{align*}
\]

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:

\[
\begin{align*}
x &= a_0[0] \land \forall j . 0 \leq j < i \rightarrow a[j] = a_0[j + 1] \land \forall j . i \leq j < |a| \rightarrow a[j] = a_0[j] \land
i < |a| - 1 \land |a'| = |a| \land i' = i + 1 \land x' = x \land a'[i] = a[i + 1] \land \forall j . j \neq i \rightarrow a'[j] = a[j] \\
\rightarrow x' &= a_0[0] \land \forall j . 0 \leq j < i' \rightarrow a'[j] = a_0[j + 1] \land \forall j . i' \leq j < |a'| \rightarrow a'[j] = a_0[j]
\end{align*}
\]

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 \( \text{SIL} \). 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 \( \text{SIL} \) is undecidable, by reducing from the Hilbert’s Tenth Problem [14]. 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 \( \text{SIL} \) is still open.
First, we show that multiplication and addition of strictly positive integers can be encoded using formulae of $\forall^*\exists^*\forall^*\text{-SIL}$. Let $x, y, z \in \mathbb{N}$, with $z > 0$. We define:

$q_1(j) : a_2[j] > 0 \land a_3[j] > 0 \land a_1[j] + 1 = a_1[j] + 1 \land a_2[j] = a_1[j] + a_2[j + 1] + a_2[j] - 1$ \land a_3[j + 1] = a_3[j]$

$q_2(j) : a_2[j] = 0 \land a_3[j] > 0 \land a_1[j + 1] = a_1[j] \land a_2[j + 1] = y \land a_3[j + 1] = a_3[j] - 1$

$q_{x=y}(a_1, a_2, a_3, n_1, n_2) : n_1 < n_2 \land a_1[n_1] = 0 \land a_2[n_1] = y \land a_3[n_1] = z \land a_1[n_2] = x \land a_3[n_2] = 0 \land \forall i. (n_1 \leq i \land n_2 \rightarrow \exists j. i \leq j < n_2 \land q_2(j) \land \forall k. (i \leq k < j \rightarrow q_1(k)))$

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

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

**Proof.** We first suppose that $x = yz$ and give a model of $q_{x=y}(a_1, a_2, a_3, n_1, n_2)$. 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 $q_{x=y}(a_1, a_2, a_3, n_1, n_2)$.

Let us consider now a model of $q_{x=y}(a_1, a_2, a_3, n_1, n_2)$. We show that this implies $x = yz$. A model of $n_1 < n_2 \land a_1[n_1] = 0 \land a_2[n_1] = y \land a_3[n_1] = z \land a_1[n_2] = x \land a_3[n_2] = 0 \land \forall i. (n_1 \leq i < n_2 \rightarrow \exists j. i \leq j < n_2 \land q_2(j) \land \forall k. (i \leq k < j \rightarrow q_1(k)))$ 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, $q_1(j)$ and $q_2(j)$ cannot be true at the same point $j$ since $\models q_1(j) \rightarrow a_2[j] > 0$ and $\models q_2(j) \rightarrow a_2[j] = 0$.

Since the subformula $\forall i. (n_1 \leq i < n_2 \rightarrow \exists j. i \leq j < n_2 \land q_2(j) \land \forall k. (i \leq k < j \rightarrow q_1(k)))$ 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 $q_2(j)$ holds at all of these points. Furthermore, at all intermediary points $k$ not equal to one of the $j_i$’s, $q_1(k)$ has to be true. This implies that $l$ must be equal to $z$ (since $q_1(k)$ imposes $a_3[k + 1] = a_3[k]$ whereas $q_2(j)$ 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 $q_1(k)$ being true for all $k$ such that $n_1 \leq k < j_1$ as well as $q_2(j)$ 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 $q_2(j)$ 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:

$q_3(j) : a_2[j] > 0 \land a_1[j + 1] = a_1[j] + 1 \land a_2[j + 1] = a_2[j] - 1$

$q_{x=y+z}(a_1, a_2, n_1, n_2) : n_1 < n_2 \land a_1[n_1] = y \land a_2[n_1] = z \land a_1[n_2] = x \land a_2[n_2] = 0 \land \forall k.n_1 \leq k < n_2 \rightarrow q_3(k)$

9
Lemma 4. \(q_{x=y+z}(a_1, a_2, n_1, n_2)\) 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 [14]. Given a Diophantine system \(S\), we construct a \(\text{SIL} \) formula \(\Psi_S\) 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 \(\Psi_S\) be the following \(\text{SIL} \) formula with three array symbols \((a_1, a_2, a_3)\):

\[
\exists x_{1} \ldots \exists x_{k} \exists m_{1}^{1} \ldots \exists m_{n_{m}+n_{a}}^{1} \exists m_{2}^{1} \ldots \exists m_{n_{m}+n_{a}}^{2} \bigwedge_{i=1}^{n_{m}+n_{a}-1} m_{i}^{2} < m_{i+1}^{1} \bigwedge_{i=1}^{n_{a}} q_{i} \bigwedge_{i=1}^{n_{a}} q_{i}'
\]

where the formulae \(q_{i}\) and \(q_{i}'\) are defined as follows: Let \(x_{i_1} = x_{i_2}x_{i_3}\) be the \(i\)-th multiplicative equation. Then, \(q_{i} = q_{x_{i_1} = x_{i_2}x_{i_3}}(a_1, a_2, a_3, m_{1}^{1}, m_{2}^{1})\). Let \(x_{i_1} = x_{i_2} + x_{i_3}\) be the \(i\)-th additive equation. Then, \(q_{i}' = q_{x_{i_1} = x_{i_2} + x_{i_3}}(a_1, a_2, m_{n_{m}+i}^{1}, m_{n_{a}+i}^{2})\).

**Lemma 5.** A Diophantine system \(S\) has a solution if and only if the corresponding formula \(\Psi_S\) 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 \(m_1^{1}\) and \(m_2^{1}\) 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 \(\Psi_S\) is satisfiable. □

## 5 Decidability of the Satisfiability Problem for \(\exists^*\forall^*-\text{SIL}\)

We show that the set of models of a boolean combination \(q\) of universally quantified array property formulae of \(\text{SIL} \) corresponds to the set of runs of an FCADBM \(A_q\), defined inductively on the structure of the formula. More precisely, each array variable in \(q\) has a corresponding counter in \(A_q\), and given any model of \(q\) that associates integer values to all array entries, \(A_q\) 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 \(\exists^*\forall^*-\text{SIL} \) (or equivalently, for \(\forall^*-\text{SIL} \)).

### 5.1 Normalisation

Before describing the translation of \(\exists^*\forall^*-\text{SIL} \) formulae into counter automata, we need to perform a simple normalisation step. Let \(q(k, a)\) be a \(\text{SIL} \) formula in the \(\exists^*\forall^* \) fragment i.e., an existentially quantified boolean combination of (1) DBM conditions or
modulo constraints on array-bound variables $k$ and array length terms $|a|$, $a \in a$, and (2) array properties of the form $\forall i . \gamma(i, k, |a|) \rightarrow v(i, k, a)$.

Without losing generality, we assume that the sanity condition $B(v)$ is explicitly conjoined to the guard of every array property i.e., each array property is of the form $\forall i . \gamma \land B(v) \rightarrow v$.

A guard expression is a conjunction of array-bound expressions $i \sim \ell$, $\sim \in \{\leq, \geq\}$, or modulo constraints $i \equiv \pm t$ where $\ell$ is an array bound term, and $s, t \in \mathbb{N}$ such that $0 \leq t < s$. For a guard $\gamma$ 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 $i \equiv t$ by $i \equiv t'$ where $0 \leq t' < s$ and $t' \equiv t + c$.

The normalisation consists in performing the following steps in succession:

1. Replace each array property subformula $\forall i . \bigvee_j \gamma_j \rightarrow \bigwedge_k v_k$ by the equivalent conjunction $\bigwedge_{j,k} \forall i . \gamma_j \rightarrow v_k$ where $\gamma_j$ are guard expressions and $v_k$ 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 $\ell$ is an array bound term.

2. Simplify each newly obtained array property subformula as follows:

$$
\forall i . \gamma \rightarrow a[i+n] \sim \ell \quad \forall i . \gamma + n \rightarrow a[i] \sim \ell
$$

$$
\forall i . \gamma \rightarrow i - a[i+n] \sim m \quad \forall i . \gamma + n \rightarrow i - a[i] \sim m + n
$$

$$
\forall i . \gamma \rightarrow a[i+n] - b[i+m] \sim p \quad \forall i . \gamma + n \rightarrow a[i] - b[i+m-n] \sim p \quad \text{if } m \geq n
$$

$$
\forall i . \gamma \rightarrow a[i+n] - b[i+m] \sim p \quad \forall i . \gamma + m \rightarrow b[i] - a[i+n+m] \sim p \quad \text{if } m < n
$$

where:

- $\sim \in \{\leq, \geq\}$ and $\equiv$ is $\leq (\geq)$ if $\sim$ is $\leq (\geq)$, respectively, and
- $\ell$ is an array-bound term, and $m, n, p \in \mathbb{Z}$.

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

4. Rewrite each conjunction $\bigwedge_{j} i \equiv_{s_j} t_j$ occurring within the guards of array property formulae into $\bigwedge_{j} i \equiv_{s_j} \frac{s_j}{t_j}$ 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 $i \equiv t$. In case there is no modulo constraint within a guard, for uniformity reasons, conjoin the guard with the constraint $i \equiv 0$.

---

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 $\forall i_1, \ldots, i_n . \gamma(i_1, \ldots, i_n, k, |a|) \rightarrow v(i_1, k, a)$ is equivalent to $\forall i_1 . ((\exists i_2, \ldots, i_n . \gamma(i_1, \ldots, i_n, k, |a|)) \rightarrow v(i_1, k, a))$ and then the existential quantifiers in $(\exists i_2, \ldots, i_n . \gamma(i_1, \ldots, i_n, k, |a|))$ can be eliminated possibly adding modulo constraints on $k$, $|a|$ and $i_1$. 

11
5. Transform each array property subformula of the form

\[
\forall i. f \leq i \leq g \land i \equiv_s t \rightarrow a[i] - b[i + m] \sim n
\]

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

\[
\forall i. f \leq i \leq g \land i \equiv_s t \rightarrow a[i] - \tau_1[i + 1] \sim 0 \land \bigwedge_{j=1}^{m-2} \forall i. f + j \leq i \leq g + j \land i \equiv_s (t + j) \mod s \rightarrow \tau_j[i] - \tau_{j+1}[i + 1] \sim 0 \land \forall i. f + m - 1 \leq i \leq g + m - 1 \land i \equiv_s (t + m - 1) \mod s \rightarrow \tau_{m-1}[i] - b[i + 1] \sim n
\]

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.

![Figure 2. Adding fresh array variables to array property formulae](image)

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

\[
\forall i. f \leq i \leq g \land i \equiv_s t \rightarrow v
\]

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

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

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

We need the following definition to state the normal form lemma. If \( X \subseteq \text{AVar} \) is a set of array variables, then \( \mu \downarrow X \) represents the restriction of \( \mu : \text{AVar} \rightarrow \mathbb{Z}^* \) to the variables in \( X. \) For a formula \( \varphi \) of \( \text{SIL}, \) we denote by \([\varphi]_X\) the set \( \{\langle i, \mu \downarrow X \rangle \mid \langle i, \mu \rangle = \varphi \}. \)

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

### 5.2 Translating Normalised Formulae into FCADBm.

Let \( \varphi(k, a) \) be an \( \exists \forall^*\text{-SIL} \) formula that is already normalised as in the previous. The automaton encoding the models of \( \varphi \) is in fact a product \( A_{\varphi} = A_{\varphi} \otimes A_{\text{tick}}, \) where \( A_{\varphi} \) is defined inductively on the structure of \( \varphi, \) and \( A_{\text{tick}} \) is a generic FCADBm, defined next. Both \( A_{\varphi} \) and \( A_{\text{tick}} \) (and, implicitly \( A_{\varphi} \)) work with the set of counters \( x = \{x_k \mid k \in k\} \cup \{x_{\mu} \mid a \in a\} \cup \{x_a \mid a \in a\} \cup \{x_{\text{tick}}\}, \) where:
– \( x_k \) and \( x_{|a|} \) are parameters corresponding to array-bound variables, i.e., their values do not change during the runs of \( A_q \),
– \( x_i \) are counters corresponding to the array symbols, and
– \( \text{tick} \) 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 \((\tau, \mu)\) of \( q \), there exists a run of \( A_q \) such that, for each array symbol \( a \in \mathfrak{A} \), the value \( \mu(a) \) equals the value of \( x_i \) when \( x_{\text{tick}} \) equals \( n \), for all \( 0 \leq n < |a| \). The reason behind defining \( A_q \) as the product of \( A_q \) and \( A_{\text{tick}} \) is that the use of negation within \( q \), 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, \( A_q \) can only read \( x_i \), while \( A_{\text{tick}} \) is the one updating it.

Formally, let \( A_{\text{tick}} = (x, \{q_0, q_{\text{tick}}\}, \{q_0\}, \neg, \{q_{\text{tick}}\}) \), where
\[
q_0 \rightarrow \frac{x_{\text{tick}} = 0 \land x_{\text{tick}} = x_{\text{tick}} + 1 \land \bigwedge_{s \in \mathfrak{I} \cup \{\text{tick}\}} x_{s} = x_{s} \land \forall a \in \mathfrak{A}, x_{|a|} = x_{|a|}}{q_{\text{tick}}}
\]
and
\[
q_{\text{tick}} \rightarrow \frac{x_{\text{tick}} = x_{\text{tick}} + 1 \land \bigwedge_{s \in \mathfrak{I} \cup \{\text{tick}\}} x_{s} = x_{s} \land \forall a \in \mathfrak{A}, x_{|a|} = x_{|a|}}{q_{\text{tick}}}
\]
are the only transitions rules. The construction of \( A_q \) is recursive on the structure of \( q \):

– if \( q \) is a DBM constraint or modulo constraint \( \theta \) on array-bound terms, let \( A_q = (x, \{q_0, q_1\}, \{q_0\}, \rightarrow, \{q_1\}) \) where the transitions rules are \( q_1 \rightarrow q_1 \) and \( q_0 \rightarrow q_1 \), and \( \overline{\theta} \) is obtained from the constraint \( \theta \) by replacing all occurrences of \( k \in \mathfrak{K} \) by \( x_k \), and all occurrences of \( |a| \), \( a \in \mathfrak{A} \), by \( x_{|a|} \).
– if \( q = \neg \psi \), let \( A_q = A_{\overline{\psi}} \).
– if \( q = \psi_1 \land \psi_2 \), let \( A_q = A_{\psi_1} \oplus A_{\psi_2} \).
– if \( q = \psi_1 \lor \psi_2 \), let \( A_q = A_{\psi_1} \otimes A_{\psi_2} \).
– if \( q \) is an array property, \( A_q \) is defined below, according to the type of the value expression occurring on the right hand side of the implication.

Let \( q : \forall i \cdot f \leq i \leq g \land i \equiv_{s} t \rightarrow v \) be an array property subformula after normalisation. Figure 3 gives the counter automaton \( A_q \) for such a subformula. The formal definition of \( A_q = (x, Q, I, \rightarrow, F) \) 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 \( A_q \) are as follows, for all \( 0 \leq i < s \):

\[
\begin{align*}
q_i & \rightarrow \frac{s_{\text{tick}} \leq r_{i+1} \land i \equiv_{s} t}{q_{i+1} \mod s} \\
r_i & \rightarrow \frac{x_{\text{tick}} \leq s_{\text{tick}} \land r_{i+1} \mod s}{q_{i+1} \mod s} \text{ if } i \neq t \\
r_i & \rightarrow \frac{x_{\text{tick}} > s_{\text{tick}}}{q_{i+1} \mod s} \\
q_f & \rightarrow \frac{q_f}{q_f} \\
q_{0 \mod s} & \rightarrow \frac{s_{\text{tick}} = 0 \land \forall i \cdot f \leq s_{\text{tick}} \leq \overline{r}}{r_{i+1} \mod s} \text{ if } t = 0 \\
q_0 & \rightarrow \frac{s_{\text{tick}} = 0 \land \forall i \cdot f \leq s_{\text{tick}} \leq \overline{r}}{r_{i+1} \mod s} \text{ if } t \neq 0
\end{align*}
\]

Here \( \overline{\psi} \) is defined by:
The following lemma establishes correctness of our construction:

Let \( q(k, a) \) be a normalised \( \exists^* \forall^* -\text{SIL} \) formula, and \( A_q = A_q \otimes A_{\text{tick}} \) be the deterministic FCADBM whose construction was given in the previous. We define the following relation between valuations \( \langle t, \mu \rangle \in \llbracket q \rrbracket \) and traces \( \sigma \in Tr(A_q) \), denoted \( \langle t, \mu \rangle \equiv \sigma \), iff:

1. for all \( k \in k \), \( t(k) = \sigma_0(x_k) \),
2. for all \( a \in a \), \( t(|a|) = \sigma_0(x_{|a|}) = |\mu(a)| \leq |\sigma| \) and \( \mu(a) = \sigma_1(x_a), 0 \leq i < |\mu(a)| \).

The following lemma establishes correctness of our construction:

**Lemma 7.** Let \( q(k, a) \) be a normalised \( \exists^* \forall^* -\text{SIL} \) formula, and \( A_q \) be its corresponding FCADBM. Then for each valuation \( \langle t, \mu \rangle \in \llbracket q \rrbracket \) there exist a trace \( \sigma \in Tr(A_q) \) such that \( \langle t, \mu \rangle \equiv \sigma \). Dually, for each trace \( \sigma \in Tr(A_q) \) there exists a valuation \( \langle t, \mu \rangle \in \llbracket q \rrbracket \) such that \( \langle t, \mu \rangle \equiv \sigma \).

**Theorem 2.** The satisifiability problem is decidable for the \( \exists^* \forall^* \) fragment of SIL.

**Proof.** Let \( q(k, a) \) be a formula of \( \exists^* \forall^* -\text{SIL} \). By normalisation, we obtain a formula \( \phi(k, a, t) \) where \( t \) is the set of fresh array variables added during normalisation. Then, by Lemma 6, we have \( \llbracket q \rrbracket = \llbracket \phi \rrbracket \upharpoonright a \). To check satisfiability of \( q \), it is therefore enough to check satisfiability of \( \phi \). By Lemma 7, \( \phi \) is satisfiable if and only if the language of the corresponding automaton \( A_q \) 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