What Else Is Decidable about Integer 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
iosif@imag.fr
[3] FIT BUT, Božetěchova 2, CZ-61266, Brno, Czech Republic
vojnar@fit.vutbr.cz

Abstract. We introduce a new decidable logic for reasoning about infinite arrays of integers. The logic is in the $\exists^*\forall^*$ first-order fragment and allows (1) Presburger constraints on existentially quantified variables, (2) difference constraints as well as periodicity constraints on universally quantified indices, and (3) difference constraints on values. In particular, using our logic, one can express constraints on consecutive elements of arrays (e.g., $\forall i \,.\, 0 \leq i < n \rightarrow a[i+1] = a[i] - 1$) as well as periodic facts (e.g., $\forall i \,.\, i \equiv_2 0 \rightarrow a[i] = 0$). The decision procedure follows the automata-theoretic approach: we translate formulae into a special class of Büchi counter automata such that any model of a formula corresponds to an accepting run of an automaton, and vice versa. The emptiness problem for this class of counter automata is shown to be decidable as a consequence of earlier results on counter automata with a flat control structure and transitions based on difference constraints.

1 Introduction

Arrays are a fundamental data structure in computer science. They are used in all modern imperative programming languages. To verify software which manipulates arrays, it is essential to have a sufficiently powerful logic, which can express meaningful program properties, arising as verification conditions within, e.g., inductive invariant checking, or verification of pre- and post-conditions. In order to have an automatic decision procedure for the program verification problems, one needs a decidable logic.

In this paper, we develop a logic of arrays indexed by integer numbers, and having integers as values. To be as general as possible, and also to avoid having to deal explicitly with expressions containing out-of-bounds array accesses, we interpret formulae over both-ways infinite arrays. Bounded arrays can then be conveniently expressed in the logic by restricting indices to be within given bounds.

Properties that are typically of interest about arrays in a program are (existentially quantified) boolean combinations of formulae of the form $\forall \mathbf{i}.G \rightarrow V$ where G is a *guard expression* containing constraints over the universally quantified index variables $\mathbf{i}$

[*] The work was supported by the French Ministry of Research (RNTL project AVERILES), the Czech Grant Agency (projects 102/07/0322, 102/05/H050), the Czech-French Barrande project 2-06-27, and the Czech Ministry of Education by project MSM 0021630528.

(which often range in between some existentially quantified bounds), and V is a *value expression* containing constraints over array values. Based on examples, we identified two types of array properties which seem to appear quite often in programs: (1) properties relating consecutive elements of an array, e.g., $\forall i \,.\, l_1 \leq i < l_2 \rightarrow a[i+1] = a[i] - 1$, which states the fact that each value of a between two bounds l_1 and l_2 is less than its predecessor by one, (2) properties stating periodic facts, e.g., $\forall i \,.\, i \equiv_2 0 \rightarrow a[i] = 0$, stating that all even elements of an array a are equal to 0.

Without specific syntactic restrictions, a logic with such an expressive power can be easily shown to be undecidable as one can encode histories of computations of a 2-counter machine [13] as models of a formula over arrays. From this reduction, one can derive two restrictions leading to decidability. The first restriction forbids references to $a[i]$ and $a[i+1]$ in the same formula, which is considered in the work of Bradley, Manna, and Sipma [5]. The second restriction, considered in this paper, allows only array formulae $\forall \mathbf{i}. G \rightarrow V$ in which V does not contain disjunctions. We have chosen the second option, mainly to retain the possibility of relating consecutive arrays elements, i.e., $a[i]$ and $a[i+1]$, which appears to be important for expressing properties of programs.

We introduce a new logic **LIA** (Logic on Integer Arrays) in the $\exists^*\forall^*$ first-order fragment. **LIA** is essentially the set of existentially quantified boolean combinations of (1) array formulae of the form $\forall \mathbf{i} \,.\, \varphi(\mathbf{k}, \mathbf{i}) \rightarrow \psi(\mathbf{k}, \mathbf{i}, \mathbf{a})$ where $\mathbf{i}$ is a set of index variables and $\mathbf{a}$ (resp. $\mathbf{k}$) is a set of existentially quantified array (resp. *array-bound*) variables, φ is a formula on index variables with difference as well as periodicity constraints on variables $\mathbf{i}$ wrt. the array-bounds $\mathbf{k}$, and ψ is a difference constraint on array terms, and (2) Presburger arithmetic formulae on array-bound variables. In [8], we give an example program showing the usefulness of this logic to express verification conditions.

We prove decidability of the logic **LIA** using the classical idea of the connection between logic and automata [18]: from a formula φ of the logic, we build an automaton A_φ such that φ is satisfiable if and only if the language of A_φ is not empty. Decidability of the logic then follows from decidability of the emptiness problem for the class of automata that is deployed. To this end, we define a new class of counter automata, called FBCA (bi-infinite Flat Büchi Counter Automata). These are counter automata running to infinity in both left and right directions, equipped with a Büchi acceptance condition. For an arbitrary formula φ of **LIA**, we give the construction of the FBCA A_φ whose runs correspond to models of φ: the value of the counter x_a at a given point i in an execution of A_φ corresponds to the value of $a[i]$ in a model of φ. We prove decidability of **LIA** by showing that the emptiness problem for FBCA is decidable by extending known results [6,4] on flat counter automata with difference bound constraints.

Related work. In the seminal paper [12], 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 [10]. Since then, various decidable logics on arrays have been considered—e.g., [17,11,9,16,1,7]. These logics include working with various predicates (reasoning about sortedness, permutations, etc.) and in terms of various arithmetic (usually Presburger) constraints on array indices and/or values of array entries. However, unlike our logic, most of these works consider quantifier free formulae. In these cases, nested array reads (like $a[a[i]]$) are allowed, which is not the case in our logic.

In [5], an interesting logic within the $\exists^*\forall^*$ fragment is developed. Unlike our decision procedure based on automata theory, the decision procedure of [5] is based on the fact that the universal quantification can be replaced by a finite conjunction. The result is parameterised in the sense of allowing an arbitrary decision procedure to be used for the data stored in arrays. However, compared to our results, [5] does not allow modulo constraints (allowing to speak about periodicity in the array values), general difference constraints on *universally* quantified indices (only $i - j \leq 0$ is allowed), 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 then appears also in [3] where a quite general $\exists^*\forall^*$ logic on multisets of elements with associated data values is considered.

Remark. For space reasons, all proofs are deferred to [8].

2 Counter Automata

Given a formula φ, we denote by $FV(\varphi)$ the set of its free variables. If we denote a formula as $\varphi(x_1,...,x_n)$, we assume $FV(\varphi) \subseteq \{x_1,...,x_n\}$. For $\varphi(x_1,...,x_n)$, we denote by $\varphi[t/x_i]$, $1 \leq i \leq n$, the formula in which each occurrence of x_i 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. By $\sigma : \mathbb{Z} \to \mathbb{Z}$, $\sigma(n) = n+1$, we denote the successor function on integers. In the following, we work with two sets of arithmetic formulae: difference bound matrices and Presburger arithmetic.

A *difference bound matrix* (DBM) formula is a conjunction of inequalities of the form $x - y \leq c$, $x \leq c$, or $x \geq c$ where $c \in \mathbb{Z}$ is a constant. If there is no constraint between x and y, we may explicitly write $x - y \leq \infty$. In the following, $\mathbb{Z}^\infty$ denotes $\mathbb{Z} \cup \{\infty\}$. Let $\mathbf{z} = \{z_1, \ldots, z_n\}$ be a designated set of variables, called *parameters*. A *parametric DBM* formula is a conjunction of a DBM formula with atomic propositions of the forms $x \leq f(\mathbf{z})$ or $x \geq f(\mathbf{z})$ where f is a linear combination of parameters, i.e., $f = a_0 + \sum_{i=1}^n a_i z_i$ for some $a_i \in \mathbb{Z}$, $0 \leq i \leq n$.

A *Presburger arithmetic* (PA) formula is a disjunction of conjunctions of either linear constraints of the form $\sum_{i=1}^n a_i x_i + b \geq 0$ or modulo constraints $\sum_{i=1}^n a_i x_i + b \equiv c \mod d$ where $a_i, b, c, d \in \mathbb{Z}$, $c \geq 0$ and $d > 0$, are constants. It is well-known that every formula of the arithmetic of integers with addition $\langle \mathbb{Z}, \geq, +, 0, 1 \rangle$ can be written in this form due to quantifier elimination [15]. Clearly, every DBM formula is also in PA.

A *counter automaton* (CA) is a tuple $A = \langle \mathbf{x}, Q, \to \rangle$ where $\mathbf{x}$ is a finite set of counters ranging over $\mathbb{Z}$, Q a finite set of control states, and $\to$ a transition relation given by 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 $\mathbf{x}'$. A *configuration* of a CA A is a pair (q,ν) where $q \in Q$ is a control state, and $\nu : \mathbf{x} \to \mathbb{Z}$ 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' iff there exists a sequence of configurations $c = c_0c_1\ldots c_n = c'$ such that, for all $0 \leq i < n$, 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 $c_0c_1\ldots c_n$ with $c_0 = (q,\nu)$, $c_n = (q',\nu')$ for some valuations $\nu,\nu' : \mathbf{x} \to \mathbb{Z}$, and c_{i+1} is an immediate successor of c_i for all $0 \leq i < n$.

Let S be a set. A *bi-infinite sequence* of S is a function $\beta : \mathbb{Z} \to S$.[1] We denote by ${}^{\omega}S^{\omega}$ the set of all bi-infinite sequences over S. A *bi-infinite Büchi counter automaton* (BCA) is a tuple $A = \langle \mathbf{x}, Q, L, R, \to \rangle$ where $\mathbf{x}$ is a finite set of counters, Q is a finite set of control states, $L,R \subseteq Q$ are the left-accepting and right-accepting states, and $\to$ is a transition relation defined in the same way as for counter automata.

A *run* of a BCA A is a bi-infinite sequence of configurations $\ldots c_{-2}c_{-1}c_0c_1c_2\ldots$ such that, for all $i \in \mathbb{Z}$, c_{i+1} is an immediate successor of c_i. A run r is *left-accepting* iff there exists a state $q \in L$ and an infinite decreasing sequence of integers $\ldots < i_2 < i_1 < 0$ such that, for all $j \in \mathbb{N}$, we have $r(i_j) = (q,\nu_j)$ for some valuations ν_j of the counters of A. Symmetrically, a run is *right-accepting* iff there exists a state $q \in R$ and an infinite increasing sequence of integers $0 < i_0 < i_1 < i_2 < \ldots$ such that, for all $j \in \mathbb{N}$, we have $r(i_j) = (q,\nu_j)$ for some valuations ν_j of the counters of A. A run is *accepting* iff it is both left- and right-accepting. The set of all accepting runs of A is denoted as $\mathcal{R}(A)$. If $r \in \mathcal{R}(A)$ is a run of A, we define as $val(r) = \ldots val(r(-1))val(r(0))val(r(1))\ldots$ the bi-infinite sequence of valuations in r, and we let $\mathcal{V}(A) = \{val(r) \mid r \in \mathcal{R}(A)\}$.

Lemma 1. *For any BCA A, we have $r \in \mathcal{R}(A)$ if and only if $r \circ \sigma \in \mathcal{R}(A)$.*

A *control path* in a CA (or BCA) A is a finite sequence $q_0q_1\ldots q_n$ of control states such that, for all $0 \leq i < n$, 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 the first one that appears twice. A CA (or BCA) is said to be *flat* iff each control state belongs to at most one elementary cycle.

Decidability and Closure Properties of FBCA. We consider in the following the class of bi-infinite Büchi counter automata which are flat, whose elementary cycles are labelled with parametric DBM formulae, and the remaining transitions are labelled with PA formulae. Moreover, each transition constraint enforces the values of parameters to remain constant. We call this class FBCA. We prove that the emptiness problem for FBCA is decidable using results of [6,4] and their extensions that can be found in [8].

Lemma 2. *The emptiness problem is decidable for the class of FBCA.*

[1] In the early literature [14], a bi-infinite sequence is defined as the equivalence class of all compositions $\beta \circ \sigma^n \circ \sigma^{-m}$ for arbitrary $n,m \in \mathbb{N}$. This is because a bi-infinite sequence remains the same if shifted left or right. For simplicity, we formally distinguish here the bi-infinite sequences β, $\beta \circ \sigma^n$, and $\beta \circ \sigma^{-n}$ for $n > 0$.

The FBCA class is also effectively closed under union and intersection. However, before proceeding, we need to elucidate the meaning of these operations for CA (BCA). For a valuation $\nu : \mathbf{x} \rightarrow \mathbb{Z}$, if $\mathbf{z} \subseteq \mathbf{x}$ is a subset of the counters in $\mathbf{x}$, let $\nu\downarrow_{\mathbf{z}}$ denote the restriction of ν to the domain $\mathbf{z}$. For some subset $\mathbf{z} \subset \mathbf{x}$ of the counters of A and $s \in \mathcal{V}(A)$, we define the restriction operator on sequences $s\downarrow_{\mathbf{z}} = \ldots val(s(-1))\downarrow_{\mathbf{z}} val(s(0))\downarrow_{\mathbf{z}} val(s(1))\downarrow_{\mathbf{z}} \ldots$, and $\mathcal{V}(A)\downarrow_{\mathbf{z}} = \{s\downarrow_{\mathbf{z}} \mid s \in \mathcal{V}(A)\}$. Symmetrically, for $\mathbf{z} \supset \mathbf{x}$, we define the extension operator on sequences $\mathcal{V}(A)\uparrow_{\mathbf{z}} = \{v \in {}^{\omega}(\mathbf{z} \mapsto \mathbb{Z})^{\omega} \mid v\downarrow_{\mathbf{x}} \in \mathcal{V}(A)\}$.

A class of counter automata is said to be *closed* under union and intersection if there exist operations $\uplus$ and $\otimes$ such that, for any two FBCA $A_i = \langle \mathbf{x_i}, Q_i, L_i, R_i, \rightarrow_i \rangle$, $i = 1,2$, we have that $\mathcal{V}(A_1 \uplus A_2) = \mathcal{V}(A_1)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cup \mathcal{V}(A_2)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$ and $\mathcal{V}(A_1 \otimes A_2) = \mathcal{V}(A_1)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cap \mathcal{V}(A_2)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$, respectively. The class is said to be *effectively* closed under union and intersection if these operators are effectively computable.

Proposition 1. *Let $A = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ be a FBCA. Let $A^c = \langle \mathbf{x}, Q, L^c, R^c, \rightarrow \rangle$ be the FBCA such that (1) for all $q \in L$ and $q' \in Q$, q' belongs to the same elementary cycle as q iff $q' \in L^c$, (2) for all $q \in R$ and $q' \in Q$, q' belongs to the same elementary cycle as q iff $q' \in R^c$. Then we have that $\mathcal{R}(A) = \mathcal{R}(A^c)$.*

Assuming w.l.o.g. that $Q_1 \cap Q_2 \neq \emptyset$, the union is defined as $A_1 \uplus A_2 = \langle \mathbf{x_1} \cup \mathbf{x_2}, Q_1 \cup Q_2, L_1 \cup L_2, R_1 \cup R_2, \rightarrow_1 \cup \rightarrow_2 \rangle$. The product is defined as $A_1 \otimes A_2 = \langle \mathbf{x_1} \cup \mathbf{x_2}, Q_1 \times Q_2, L_1^c \times L_2^c, R_1^c \times R_2^c, \rightarrow \rangle$ where $\rightarrow$ is as follows: $(q_1, q_1') \xrightarrow{\varphi_1 \wedge \varphi_2} (q_2, q_2')$ iff $q_1 \xrightarrow{\varphi_1} q_2$ is a transition rule of A_1 and $q_1' \xrightarrow{\varphi_2} q_2'$ is a transition rule of A_2. Here, L_i^c and R_i^c denote the extended left-accepting and right-accepting sets of A_i from Proposition 1 for $i = 1,2$.

Lemma 3. *The class of FBCA is effectively closed under union and intersection.*

3 A Logic for Integer Arrays

In this section we define the Logic of Integer Arrays (**LIA**) that we use to specify properties of programs handling arrays of integers.

Syntax. We consider three types of variables. The *array-bound variables* (k,l) appear within the so-called array-bound terms. These terms can be used to define intervals of indices and also as static references inside arrays. The *index* (i,j) and *array* (a,b) *variables* are used to build array terms. Fig. 1 shows the syntax of the logic **LIA**. We use the symbol $\top$ to denote the boolean value *true*. In the following, we will use $f \leq i \leq g$ instead of $f \leq i \wedge i \leq g$, $i < f$ instead of $i \leq f - 1$, and $i = f$ instead of $f \leq i \leq f$. Intuitively, our logic is the set of existentially quantified boolean combinations of:

1. Array formulae of the form $\forall \mathbf{i} \,.\, \varphi(\mathbf{k}, \mathbf{i}) \rightarrow \psi(\mathbf{k}, \mathbf{i}, \mathbf{a})$ where $\mathbf{k}$ is a set of array-bound variables, $\mathbf{i}$ is a set of index variables, $\mathbf{a}$ is a set of array variables, φ is an arithmetic formula on index variables, and ψ is an arithmetic formula on array terms. In particular, ψ is a DBM formula, and φ is composed of atomic propositions of the form either $f \leq i$, $i \leq f$, $i - j \leq n$, or $i \equiv_s t$ where f is a linear combination of array-bound variables, $n \in \mathbb{Z}$, and $0 \leq t < s$. Both $\mathbf{k}$ and $\mathbf{a}$ variables are free in the array formulae, but they can be existentially quantified at the top-most level.
2. PA formulae on array-bound variables.

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

Fig. 1. Syntax of the logic **LIA**

Examples. To accustom the reader with the logic, we consider several properties of interest that can be stated about arrays. For instance, a strictly increasing ordering of an array a up to a certain bound is defined as $\exists k\ \forall i\ .\ 0 \le i < k \rightarrow a[i] - a[i+1] \le -1$. The fact that the first k elements of an array a are below the first l elements of an array b at distance 5 is defined as $\exists k,l\ \forall i,j\ .\ 0 \le i < k\ \wedge\ 0 \le j < l \rightarrow a[i] - b[j] \le -5$. Equality of two arrays up to a certain bound can be expressed as $\exists n \forall i\ .\ 0 \le i < n \rightarrow a[i] = b[i]$. The use of modulo constraints as guards for indices allows one to express periodic facts, e.g., $\forall i,j\ .\ i \equiv_2 0\ \wedge\ j \equiv_2 1 \rightarrow a[i] \le a[j]$ meaning that any value at some even position is less than or equal to any value at some odd position in a. In [8], we show that to prove the correctness of an array merging program, such properties are needed.

Semantics. The logic **LIA** is interpreted on *both-ways infinite arrays*. This allows us to conveniently deal with out-of-bound reference situations common in programs handling arrays. One can prevent and/or check for out-of-bound references by introducing explicit existentially quantified array-bound variables for array variables. Let $\varphi(\mathbf{k},\mathbf{a})$ be any **LIA** formula. A *valuation* is a pair of partial functions[2] $\langle \iota, \mu \rangle$ with $\iota : \mathit{BVar} \cup \mathit{IVar} \rightarrow \mathbb{Z}_\perp$ associating an integer value with every free integer variable and $\mu : \mathit{AVar} \rightarrow {}^\omega\mathbb{Z}^\omega_\perp$ associating a bi-infinite sequence of integers with every array symbol $a \in \mathbf{a}$. The valuation ι is extended in the standard way to array-bound terms ($\iota(B)$) and index terms ($\iota(I)$). By $I_{\iota,\mu}(A)$, we denote the value of the array term A given by the valuation $\langle \iota, \mu \rangle$. The semantics of a formula φ is defined in terms of the forcing relation $\models$ as follows:

$$\begin{array}{rlcr}
 & \langle \iota,\mu \rangle \models A \le B & \iff & I_{\iota,\mu}(A) \le \iota(B) \\
I_{\iota,\mu}(a[I]) = \mu(a)(\iota(I)) & \langle \iota,\mu \rangle \models A_1 - A_2 \le n & \iff & I_{\iota,\mu}(A_1) - I_{\iota,\mu}(A_2) \le n \\
I_{\iota,\mu}(a[B]) = \mu(a)(\iota(B)) & \langle \iota,\mu \rangle \models \forall i\ .\ G \rightarrow V & \iff & \forall\, n \in \mathbb{Z}\ .\ \langle \iota[i \leftarrow n],\mu \rangle \models G \rightarrow V \\
 & \langle \iota,\mu \rangle \models \exists a\ .\ \psi & \iff & \exists\, \beta \in {}^\omega\mathbb{Z}^\omega\ .\ \langle \iota,\mu[a \leftarrow \beta] \rangle \models \psi
\end{array}$$

For space reasons, we do not give here a full definition. However, the missing rules are standard in first-order arithmetic. A *model* of $\varphi(\mathbf{k},\mathbf{a})$ is a valuation $\langle \iota, \mu \rangle$ such that

[2] The symbol $\perp$ is used to denote that a partial function is undefined at a given point.

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 *satisfiable* if and only if $[\![\varphi]\!] \neq \emptyset$.

An Undecidability Result. The reason behind the restriction that array terms may not occur within disjunctions in value expressions (cf. Fig. 1) is that, without it, the logic becomes undecidable. The essence of the proof is that an array formula $\forall \mathbf{i}.G \rightarrow V_1 \vee \ldots \vee V_n$, for $n > 1$, corresponds to n nested loops in a counter automaton. Undecidability is shown by reduction from the halting problem for 2-counter machines [13].

Lemma 4. *The logic obtained by extending* **LIA** *with disjunctions within the value expressions is undecidable.*

Note that having more than one nested loop is a necessary condition for undecidability of 2-counter machines since a flat 2-counter machine would trivially fall into the class of decidable counter machines from [6,4].

4 Decidability of the Satisfiability Problem

The idea behind our method for deciding the satisfiability problem for **LIA** is that, for any formula of **LIA**, there exists an FBCA A_φ such that φ has a model if and only if A_φ has an accepting run. 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 such that the values of the counters at different points of the run match the values of the array entries at corresponding indices in the model. Since, by Lemma 2, the emptiness problem is decidable for FBCA, this leads to decidability of **LIA**.

In order to build an automaton from a **LIA** formula, we first normalise it into an existentially quantified positive boolean combination of simple array property formulae (cf. Fig. 1). Second, each such array property formula is translated into an FBCA. The final automaton A_φ is defined recursively on the structure of the normalised formula with the $\uplus$ and $\otimes$ operators being the counterparts for the $\vee$ and $\wedge$ connectives, respectively.

4.1 Normalisation of Formulae

The goal of this step is to transform any formula written using the syntax of Figure 1 into a formula of the following normal form:

$$\exists \mathbf{k} \exists \mathbf{a} \,.\, \bigvee_c \Big(\bigwedge_d \phi_{cd}(\mathbf{a}, \mathbf{k}) \Big) \wedge \theta_c(\mathbf{k}) \tag{NF}$$

where $\mathbf{a}$ is a set of array variables, $\mathbf{k}$ is a set of integer variables, and

- θ_d is a conjunction of terms of the forms (i) $g(\mathbf{k}) \geq 0$ or (ii) $g(\mathbf{k}) \equiv_s t$ with g being a linear combination of the variables in $\mathbf{k}$ and $0 \leq t < s$,
- ϕ_{cd} is a formula of the following forms for $\sim \in \{\leq, \geq\}, m \in \mathbb{N}, 0 \leq t < s, 0 \leq v < u$, $q \in \mathbb{Z}$, and f_k, g_l, f_k^1, g_l^1, f_k^2, g_l^2 being linear combinations of array-bound variables:

$$\forall i \,.\, \bigwedge_{k=1}^{K} f_k \leq i \,\wedge\, \bigwedge_{l=1}^{L} i \leq g_l \,\wedge\, i \equiv_s t \rightarrow a[i] \sim h(\mathbf{k}) \tag{F1}$$

The (F1) formulae bind all values of a in some interval by some linear combination h of variables in $\mathbf{k}$.

$$\forall i \,.\, \bigwedge_{k=1}^{K} f_k \leq i \,\wedge\, \bigwedge_{l=1}^{L} i \leq g_l \,\wedge\, i \equiv_s t \rightarrow a[i] - b[i+p] \sim q \tag{F2}$$

Here, $p \in \mathbb{Z}$. The (F2) formulae relate all values of a and b in the same interval such that the distance between the indices of a and b, respectively, is constant.

$$\begin{array}{c}\forall i,j \,.\, \bigwedge_{k=1}^{K_1} f_k^1 \leq i \,\wedge\, \bigwedge_{l=1}^{L_1} i \leq g_l^1 \,\wedge\, \bigwedge_{k=1}^{K_2} f_k^2 \leq j \,\wedge\, \bigwedge_{l=1}^{L_2} j \leq g_l^2 \,\wedge \\ i - j \leq p \,\wedge\, i \equiv_s t \,\wedge\, j \equiv_u v \rightarrow a[i] - b[j] \sim q\end{array} \tag{F3}$$

Here, $p \in \mathbb{Z}^\infty$. The (F3) formulae relate all values of a with all values of b within two (possibly equal) intervals. The case when $p = \infty$ corresponds to the situation when no constraint $i - j \leq p$ with $p \in \mathbb{Z}$ is used.

Lemma 5. *A formula of* **LIA** *can be equivalently written in the form (NF).*

In the following, we refer to the *matrix* of φ as to the formula obtained by forgetting the existential quantifier prefix from the (NF) form of φ.

4.2 Formulae and Constraint Graphs

In [6,4], the set of runs of a flat counter automaton is represented by an unbounded constraint graph. Here, we view the models of a formula as a constraint graph both left- and right-infinite. These constraint graphs are then seen as executions of FBCA, relating in this way models of formulae to runs of automata.

Let $\varphi(\mathbf{k},\mathbf{a})$ be a formula of type (F1)-(F3), and $\iota : \mathbf{k} \rightarrow \mathbb{Z}$ a valuation of its array-bound variables $\mathbf{k}$. For the rest of this section, we fix the valuation ι, and we denote by φ_ι the formula obtained from φ by replacing each occurrence of $k \in \mathbf{k}$ by the value $\iota(k)$.

The formula φ_ι can thus be represented by a weighted directed graph $G_{\iota,\varphi}$ in which each node (a,n) represents the array entry $a[n]$ for some $a \in \mathbf{a}$ and $n \in \mathbb{Z}$, and there is a path of weight w between nodes (a,n) and (b,m) iff the constraint $a[n] - b[m] \leq w$ is implied by φ_ι. In the next section, we will show that these graphs are in a one-to-one correspondence with the accepting runs of an FBCA.

In order to build the constraint graph of a formula, one needs to pay attention to the following issue. Consider, e.g., the formula $\forall i,j.i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$. The constraint graph of this formula needs to have a path of weight 5 between, e.g., $a[0]$ and $b[1]$, $a[0]$ and $b[3]$, $a[0]$ and $b[5]$, etc. As one can easily notice, the span of such paths is potentially unbounded. Since we would like this graph to represent a computation of a flat counter automaton, it is essential to define it as a sequence composed of (a possibly unbounded number of) repetitions of a finite number of (finite) sub-graphs (see, e.g., Fig. 2(a) or Fig. 2(b)). To this end, we introduce intermediary nodes which are connected between themselves with 0 arcs such that, for each non-local constraint of the form $a[n] - b[m] \leq w$ where $|n - m|$ can be arbitrarily large, there exists exactly one path of weight w through these nodes. E.g., in Fig. 2(a), there is a path

$(a,0) \xrightarrow{5} (t_\varphi,-3) \xrightarrow{0} \dots \xrightarrow{0} (t_\varphi,1) \xrightarrow{0} (b,1)$ for the constraint $a[0]-b[1] \le 5$, another path $(a,0) \xrightarrow{5} (t_\varphi,-3) \xrightarrow{0} \dots \xrightarrow{0} (t_\varphi,3) \xrightarrow{0} (b,3)$ for the constraint $a[0]-b[3] \le 5$, etc.

Formally, the constraint graph $G_{\iota,\varphi} = \langle V,E \rangle$ of a formula φ of type (F1)-(F3) is defined as follows: The set of vertices is $V = (\mathcal{A} \cup \mathcal{T} \cup \{\zeta\}) \times \mathbb{Z}$. Here, $\mathcal{A} = \{a\}$ for (F1) formulae, and $\mathcal{A} = \{a,b\}$ for (F2)-(F3) formulae, with a or a,b being the arrays that appear in φ of type (F1) or (F2)-(F3), respectively. Next, $\mathcal{T} = \emptyset$ for (F1)-(F2) formulae, and $\mathcal{T} = \{t_\varphi\}$ for (F3) formulae where t_φ is a unique auxiliary symbol (track) associated with each formula φ of type (F3). Finally, ζ is a special shared symbol (zero track). The set of edges E is defined based on the type (F1)-(F3) of φ. For space reasons, we give here only the definitions for formulae of type (F3), which are the most interesting. Formulae (F1) and (F2) are treated in [8]. In general, for all types of formulae, we have:

$$E \supset \{(\zeta,k) \xrightarrow{0} (\zeta,k+1) \mid k \in \mathbb{Z}\} \cup \{(\zeta,k+1) \xrightarrow{0} (\zeta,k) \mid k \in \mathbb{Z}\}$$

i.e., the value of the zero track stays constant.

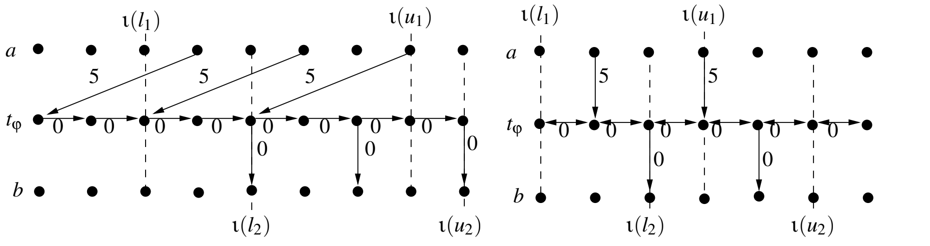
Constraint graphs for (F3) formulae. Let φ be the formula below where $0 \le s < t$, $0 \le u < v$, $p \in \mathbb{Z}^\infty$, $q \in \mathbb{Z}$, and f_k^1, g_l^1, f_k^2, g_l^2 are linear combinations of array-bound variables:

$$\forall i,j \,.\, \underbrace{\bigwedge_{k=1}^{K_1} f_k^1 \le i \wedge \bigwedge_{l=1}^{L_1} i \le g_l^1 \wedge i \equiv_s t}_{\phi^1} \wedge \underbrace{\bigwedge_{k=1}^{K_2} f_k^2 \le j \wedge \bigwedge_{l=1}^{L_2} j \le g_l^2 \wedge j \equiv_u v}_{\phi^2} \wedge i-j \le p \rightarrow a[i]-b[j] \sim q$$

Let $\phi^1(i,\mathbf{k})$ and $\phi^2(j,\mathbf{k})$ be the subformulae defining the ranges of i and j, respectively, and $\mathcal{P}_\iota^1 = \{n \in \mathbb{Z} \mid \models \phi_\iota^1[n/i]\}$ and $\mathcal{P}_\iota^2 = \{n \in \mathbb{Z} \mid \models \phi_\iota^2[n/j]\}$ be these ranges under the valuation ι. Let $T_\le = \{(t_\varphi,k) \xrightarrow{0} (t_\varphi,k+1) \mid k \in \mathbb{Z} \wedge \exists n \in \mathcal{P}_\iota^1 \; \exists m \in \mathcal{P}_\iota^2 \,.\, n-m \le p\}$ and $T_\ge = \{(t_\varphi,k) \xrightarrow{0} (t_\varphi,k-1) \mid k \in \mathbb{Z} \wedge \exists n \in \mathcal{P}_\iota^1 \; \exists m \in \mathcal{P}_\iota^2 \,.\, n-m \ge p\}$. Note that $T_\le$ and $T_\ge$ are empty if the precondition of φ is not satisfiable. The set of edges E is defined by the following case split:

1. If $p < \infty$, we consider two cases based on the direction of $a[i]-b[j] \sim q$:
   (a) for $a[i]-b[j] \le q$, we have (Fig. 2(a)):
   $E \supset \{(a,k) \xrightarrow{q} (t_\varphi,k-p) \mid k \in \mathcal{P}_\iota^1\} \cup \{(t_\varphi,k) \xrightarrow{0} (b,k) \mid k \in \mathcal{P}_\iota^2\} \cup T_\le$
   (b) for $a[i]-b[j] \ge q$, we have:
   $E \supset \{(b,k) \xrightarrow{-q} (t_\varphi,k+p) \mid k \in \mathcal{P}_\iota^2\} \cup \{(t_\varphi,k) \xrightarrow{0} (a,k) \mid k \in \mathcal{P}_\iota^1\} \cup T_\ge$
2. If $p = \infty$, we consider again two cases based on the direction of $a[i]-b[j] \sim q$:
   (a) for $a[i]-b[j] \le q$, we have (Fig. 2(b)):
   $E \supset \{(a,k) \xrightarrow{q} (t_\varphi,k) \mid k \in \mathcal{P}_\iota^1\} \cup \{(t_\varphi,k) \xrightarrow{0} (b,k) \mid k \in \mathcal{P}_\iota^2\} \cup T_\le \cup T_\ge$
   (b) for $a[i]-b[j] \ge q$, we have:
   $E \supset \{(b,k) \xrightarrow{-q} (t_\varphi,k) \mid k \in \mathcal{P}_\iota^2\} \cup \{(t_\varphi,k) \xrightarrow{0} (a,k) \mid k \in \mathcal{P}_\iota^1\} \cup T_\le \cup T_\ge$

Nothing else is in E.



(a) $\forall i,j.l_1 \le i \le u_1 \wedge l_2 \le j \le u_2 \wedge i-j \le 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i]-b[j] \le 5$ (b) $\forall i,j.l_1 \le i \le u_1 \wedge l_2 \le j \le u_2 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i]-b[j] \le 5$

Fig. 2. Examples of constraint graphs for (F3) formulae

Relating constraint graphs and models of formulae. We can now prove a correspondence between constraint graphs and models of formulae of the forms (F1)-(F3). Namely, it is the fact that if the vertices of a constraint graph for a formula φ can be labelled in a consistent way, then from the labelling, one can extract a model for φ, and vice versa. This formalises correctness of the construction for constraint graphs using the additional tracks.

Let $\varphi(\mathbf{k},\mathbf{a})$ be a formula of the forms (F1)-(F3), $\iota : \mathbf{k} \rightarrow \mathbb{Z}$ a valuation of the array-bound variables in φ, and $G_{\iota,\varphi} = (V,E)$ its corresponding constraint graph. A *labelling* $Lab : V \rightarrow \mathbb{Z}$ of $G_{\iota,\varphi}$ is called *consistent* if and only if (1) for all edges $v_1 \xrightarrow{k} v_2 \in E$, we have $Lab(v_1) - Lab(v_2) \le k$ and (2) $Lab((\zeta,n)) = 0$ for all $n \in \mathbb{Z}$.

Lemma 6. *Let $\varphi(\mathbf{k},\mathbf{a})$ be a formula of the form (F1)-(F3). Then, for all valuations $\iota : \mathbf{k} \rightarrow \mathbb{Z}$ and $\mu : \mathbf{a} \rightarrow {}^{\omega}\mathbb{Z}^{\omega}$, we have that $\langle \iota,\mu \rangle \models \varphi$ if and only if there exists a consistent labelling Lab of $G_{\iota,\varphi}$ such that $\mu(a)(i) = Lab((a,i))$ for all $a \in \mathbf{a}$ and $i \in \mathbb{Z}$.*

4.3 From Formulae to Counter Automata

In this section, we describe the construction of an FBCA A_φ corresponding to a formula φ such that (1) each run of A_φ corresponds to a model of φ, and (2) for each model of φ, A_φ has at least one corresponding run. In this way, we effectively reduce the satisfiability problem for **LIA** to the emptiness problem for FBCA.

The construction of FBCA is by induction on the structure of the formulae. For the rest of this section, let φ be a formula, $\mathbf{k}$ the set of array-bound variables in φ, and $\mathbf{a}$ the set of array variables in φ, i.e., $FV(\varphi) = \mathbf{k} \cup \mathbf{a}$. Suppose that φ is the matrix of a formula in the normal form (NF), i.e., $\varphi : \bigvee_{i \in I} \theta_i(\mathbf{k}) \wedge \bigwedge_{j \in J} \psi_{ij}(\mathbf{k},\mathbf{a})$ where θ_i are PA constraints and ψ_{ij} are formulae of types (F1)-(F3). The automaton A_φ is defined as $\biguplus_{i \in I} A_{\theta_i} \otimes \bigotimes_{j \in J} A_{\psi_{ij}}$ where $\uplus$ and $\otimes$ are the union and intersection operators on FBCA. The construction of counter automata $A_{\psi_{ij}}$ for the formulae ψ_{ij} of type (F1)-(F3) relies on the definition of the constraint graphs in Section 4.2. Namely, each accepting run of $A_{\psi_{ij}}$ gives a consistent valuation of the constraint graph of ψ_{ij}.

Counter Automata Templates. To simplify the definition of counter automata, we note that each constraint graph for the basic formulae of type (F1)-(F3) is composed of *horizontal*, *vertical*, and *diagonal* edges, which are defined in roughly the same way for all types of formulae (cf. Section 4.2). We take advantage of this fact and we start by defining three types of counter automata *templates*, which are subsequently used to define the counter automata for the basic formulae.[3] More precisely, the automata for (F1)-(F3) formulae will be defined as $\otimes$-products of particular instances of the automata templates for the horizontal, vertical, and diagonal edges of the appropriate constraint graphs. In the following definitions, we assume the existence of a special counter x_τ (tick) incremented by each transition rule, i.e., we suppose that the constraint $x'_\tau = x_\tau + 1$ is implicitly in conjunction with each formula labelling a transition rule. Intuitively, the role of the x_τ counter is to synchronise all automata composed by the $\otimes$-product on a common current position.

The template for the horizontal edges. Let a be an array symbol, $dir \in \{\texttt{left}, \texttt{right}, \texttt{bi}\}$ be a *direction* parameter, and ϕ be a formula on array-bound variables. Let $\mathbf{x_k}$ be the set $\{x_k \mid k \in FV(\phi)\}$. We define the template $H(a, dir, \phi) = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where:

- $\mathbf{x} = \{x_a\} \cup \mathbf{x_k}$. These counters will have the same names in all instances of H.
- $Q = \{q_L, q_R, p_L, p_R\}$. The control states are required to have fresh names in every instance of H. $L = \{q_L, p_L\}$ and $R = \{q_R, p_R\}$.
- $q_L \xrightarrow{\xi} q_L$, $q_R \xrightarrow{\xi} q_R$, $q_L \xrightarrow{\phi(\mathbf{x_k}) \wedge \xi} q_R$, $p_L \xrightarrow{\top} p_L$, $p_R \xrightarrow{\top} p_R$, and $p_L \xrightarrow{\neg\phi(\mathbf{x_k})} p_R$.

In the above, $\phi(\mathbf{x_k})$ is the formula obtained by replacing each occurrence of an array-bound variable $k \in FV(\phi)$ by its corresponding counter x_k. The formula $\xi(x_a, x'_a)$ is $x_a - x'_a \leq 0$ if $dir = \texttt{right}$, $x'_a - x_a \leq 0$ if $dir = \texttt{left}$, and $x'_a = x_a$ if $dir = \texttt{bi}$. Moreover, for each transition rule, we assume the conjunction $\bigwedge_{k \in FV(\phi)} x'_k = x_k$ to be added implicitly to the labelling formula, i.e., the value of an x_k counter stays constant throughout a run.

If the formula ϕ holds for a given valuation of the parameters $\mathbf{x_k}$, then any accepting run of (any instance of) H visits q_L infinitely often on the left and q_R infinitely often on the right. Otherwise, if ϕ does not hold for the given valuation of $\mathbf{x_k}$, the instance automata have a run that goes infinitely often through p_L on the left and through p_R on the right. In this case, the automata do not impose any constraints on x_a.

The template for the diagonal edges. Let a, b be array symbols, $q \in \mathbb{Z}$, $p, s \in \mathbb{N}^+$, $t \in [0, s-1]$, and $dir \in \{\texttt{left}, \texttt{right}\}$ be a direction parameter. In the following, we refer to the sets $\mathbb{L} = \{l_1, \ldots, l_K\}$ and $\mathbb{U} = \{u_1, \ldots, u_L\}$ of lower and upper bounds, respectively, where l_i and u_j are linear combinations of array-bound variables. Let $\mathbf{x_k} = \{x_k \mid k \in \bigcup_{i=1}^{K} FV(l_i) \cup \bigcup_{j=1}^{L} FV(u_j)\}$. Further, we assume that $\mathbb{L} \cup \mathbb{U} \neq \emptyset$ and we deal with the case of $\mathbb{L} \cup \mathbb{U} = \emptyset$ later on. We define the template $D(a, b, p, q, s, t, \mathbb{L}, \mathbb{U}, dir) = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where:

- $\mathbf{x} = \{x_a, x_b\} \cup \mathbf{x_k} \cup \{x_i \mid 1 \leq i < p\}$. The counters x_a, x_b, and $\mathbf{x_k}$ will have the same names in all instances of D. On the other hand, the counters x_i, $1 \leq i < p$, will have fresh names in every instance of D. The x_i counters are used for splitting

[3] By a *template*, we mean a class of counter automata which all share the same structure.

diagonal edges that span over more than one position into series of diagonal edges connecting only adjacent positions.[4]

- $Q = \{q_L, q_R\} \cup \{q_i \mid 0 \leq i < s\} \cup \{q_i^j \mid 0 \leq j < s,\ j+1 \leq i < j+p\}$. The control states are required to have fresh names in every instance of D. Let $L = \{q_L\} \cup \{q_i \mid 0 \leq i < s\}$ and $R = \{q_R\} \cup \{q_i \mid 0 \leq i < s\}$.
- $q_L \xrightarrow{\top} q_L$, $q_R \xrightarrow{\top} q_R$, and $q_L \xrightarrow{\neg(\exists i\ .\ \bigwedge_{l \in \mathbb{L}} i \geq l(\mathbf{x_k})\ \wedge\ \bigwedge_{u \in \mathbb{U}} i \leq u(\mathbf{x_k})\ \wedge\ i \equiv_s t)} q_R$.
- $q_L \xrightarrow{\bigwedge_{l \in \mathbb{L}} x_\tau \geq l(\mathbf{x_k})-1\ \wedge\ (\bigvee_{l \in \mathbb{L}} x_\tau = l(\mathbf{x_k})-1)\ \wedge\ x_\tau+1 \equiv_s i} q_i$ for all $0 \leq i < s$.
- $q_i \xrightarrow{\bigwedge_{l \in \mathbb{L}} x_\tau \geq l(\mathbf{x_k})\ \wedge\ \bigwedge_{u \in \mathbb{U}} x_\tau < u(\mathbf{x_k})\ \wedge\ \xi_i[x_a/x_0, x_b/x_p]} q_{(i+1) \bmod s}$ for all $0 \leq i < s$.
- $q_i \xrightarrow{\bigvee_{u \in \mathbb{U}} x_\tau = u(\mathbf{x_k})\ \wedge\ x_\tau \equiv_s i\ \wedge\ \xi_i[x_a/x_0, x_b/x_p]} q_{i+1}^i$ for all $0 \leq i < s$.
- $q_i \xrightarrow{\bigvee_{u \in \mathbb{U}} x_\tau = u(\mathbf{x_k})\ \wedge\ x_\tau \equiv_s i\ \wedge\ \xi_i[x_a/x_0, x_b/x_p]} q_R$ for all $0 \leq i < s$ if $p = 1$.
- $q_i^j \xrightarrow{\xi_i[x_a/x_0, x_b/x_p]} q_{i+1}^j$ for all $0 \leq j < s$, $j < i < j+p-1$.
- $q_{j+p-1}^j \xrightarrow{\xi_i[x_a/x_0, x_b/x_p]} q_R$ for all $0 \leq j < s$ if $p > 1$.

In the above, $l(\mathbf{x_k})$ and $u(\mathbf{x_k})$ denote the expressions l and u in which each occurrence of an array-bound variable k is replaced by its corresponding parameter x_k. As before, for each transition rule, we assume the conjunction $\bigwedge_{k \in FV(\phi)} x_k' = x_k$ to be added implicitly to the labelling formula, i.e., we require that the value of an x_k counter stays constant throughout the run. The formulae ξ_i are defined as follows:

- if $dir =$ `right`, $\xi_i = \bigwedge_{k \in K_i} x_k - x_{k+1}' \leq \alpha_k$ for $K_i = \{k \mid 0 \leq k < p,\ i \equiv_s k+t\}$, $\alpha_0 = q$, and $\alpha_k = 0$, $k > 0$,
- if $dir =$ `left`, $\xi_i = \bigwedge_{k \in K_i} x_{k-1}' - x_k \leq \alpha_k$, $K_i = \{k \mid 1 \leq k \leq p,\ k+i \equiv_s t\}$, $\alpha_1 = q$, and $\alpha_k = 0$, $k > 1$.

Finally, for the case $\mathbb{L} = \mathbb{U} = \emptyset$, we define any instance of $D(a,b,p,q,s,t,\emptyset,\emptyset,dir)$ to be $A_1 \otimes A_2$ where A_1 is an instance of $D(a,b,p,q,s,t,\emptyset,\{0\},dir)$ and A_2 is an instance of $D(a,b,p,q,s,t,\{0\},\emptyset,dir)$.

The construction can be understood by considering an accepting run of (any instance of) D. Let us consider the case in which there exists a value i in between the bounds that satisfies also the modulo constraint. If this is not the case, there will be an accepting run that takes the transition $q_L \xrightarrow{\neg(\exists i\ .\ \bigwedge_{l \in \mathbb{L}} i \geq l(\mathbf{x_k})\ \wedge\ \bigwedge_{u \in \mathbb{U}} i \leq u(\mathbf{x_k})\ \wedge\ i \equiv_s t)} q_R$ exactly once.

Since the run is accepting, it must visit a state from L infinitely often on the left, and a state from R infinitely often on the right. There are three cases: (1) $\mathbb{L} \neq \emptyset$ and $\mathbb{U} \neq \emptyset$, (2) $\mathbb{L} = \emptyset$ and $\mathbb{U} \neq \emptyset$, and (3) $\mathbb{L} \neq \emptyset$ and $\mathbb{U} = \emptyset$. In the case (1), a bi-infinite run will visit q_L infinitely often on the left and q_R infinitely often on the right. Notice that the run

[4] For instance, the constraint $a[i] - b[i+3] \leq 5$ can be split to $a[i] - x_1[i+1] \leq 5$, $x_1[i+1] - x_2[i+2] \leq 0$, and $x_2[i+2] - b[i+3] \leq 0$. The constraints for array values of neighbouring indices can then be conveniently expressed by using the current and future values of the appropriate counters (e.g., for our example constraint, $x_a - x_1' \leq 5$, $x_1 - x_2' \leq 0$, and $x_2 - x_b' \leq 0$, which of course appear on subsequent transitions of the appropriate FBCA).

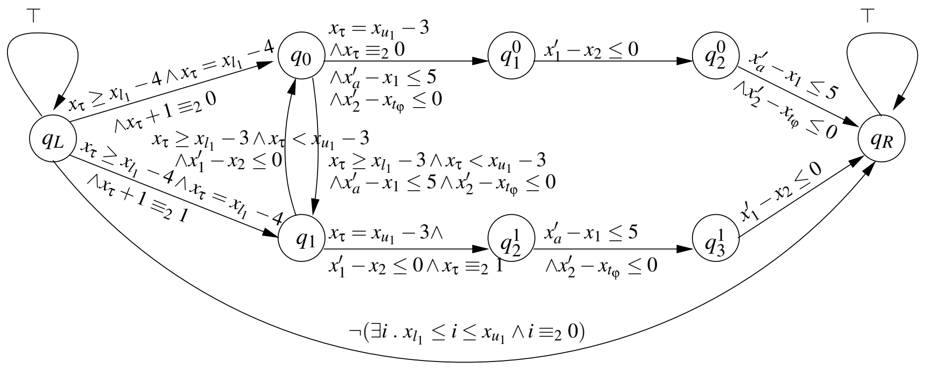

$\top$
$\top$
q_L
q_0
q_1
q_1^0
q_2^0
q_2^1
q_3^1
q_R
$x_\tau \geq x_{l_1} - 4 \wedge x_\tau = x_{l_1} - 4$
$\wedge x_\tau + 1 \equiv_2 0$
$x_\tau \geq x_{l_1} - 4 \wedge x_\tau = x_{l_1} - 4$
$\wedge x_\tau + 1 \equiv_2 1$
$x_\tau = x_{u_1} - 3$
$\wedge x_\tau \equiv_2 0$
$\wedge x_a' - x_1 \leq 5$
$\wedge x_2' - x_{t_\varphi} \leq 0$
$x_\tau \geq x_{l_1} - 3 \wedge x_\tau < x_{u_1} - 3$
$\wedge x_1' - x_2 \leq 0$
$x_\tau \geq x_{l_1} - 3 \wedge x_\tau < x_{u_1} - 3$
$\wedge x_a' - x_1 \leq 5 \wedge x_2' - x_{t_\varphi} \leq 0$
$x_1' - x_2 \leq 0$
$x_a' - x_1 \leq 5$
$\wedge x_2' - x_{t_\varphi} \leq 0$
$x_\tau = x_{u_1} - 3 \wedge$
$x_1' - x_2 \leq 0 \wedge x_\tau \equiv_2 1$
$x_a' - x_1 \leq 5$
$\wedge x_2' - x_{t_\varphi} \leq 0$
$x_1' - x_2 \leq 0$
$\neg(\exists i \,.\, x_{l_1} \leq i \leq x_{u_1} \wedge i \equiv_2 0)$

Fig. 3. The FBCA for the diagonal edges in the formula $\varphi : \forall i,j.l_1 \leq i \leq u_1 \wedge l_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$ from Fig. 2(a) obtained as $D(a, t_\varphi, 3, 5, 2, 0-3, \{l_1 - 3\}, \{u_1 - 3\}, \texttt{left})$. To understand the formula ξ_0 on the transition from q_0 to q_1, note that the constraint $i \equiv_s k + t$ in the definition of the set K_0 instantiates to $0 \equiv_2 k - 3$, and hence $K_0 = \{1, 3\}$. A similar reasoning applies for the other transitions.

cannot visit the loop $q_0 \rightarrow \ldots \rightarrow q_{s-1}$ infinitely often due to the presence of both lower and upper bounds on x_τ. In the case (2), the run cannot take any of the transitions $q_L \rightarrow q_i$, $0 \leq i < s$, due to the emptiness of $\mathbb{L}$, which makes the guard unsatisfiable. Hence, the only possibility for an accepting bi-infinite run is to visit the states $q_0 \rightarrow \ldots \rightarrow q_{s-1}$ infinitely often on the left. Due to the presence of the upper bound on x_τ, the run cannot stay forever inside this loop and must exit via one of the $q_i \rightarrow q_{i+1}^i$ (or $q_i \rightarrow q_R$ for $p = 1$) transitions, getting trapped into q_R on the right. Case (3) is symmetric to (2).

Note that, in all cases, due to the modulo tests on x_τ in the entry and exit of the main loop $q_0 \rightarrow \ldots \rightarrow q_{s-1}$ on any accepting run, whenever a state q_i, $0 \leq i < s$, is visited, the value of the x_τ counter must equal i modulo s. Note also that the role of the q_i^j states is to describe constraints corresponding to edges that start inside the given interval bounds and lead above its upper bound (or vice versa). The number of such edges is bounded. We do not use the same construction at the beginning of the interval as the templates are applied such that none of the edges represented goes below the lower bounds.

Template for the vertical edges. Let a, b be array symbols, q a linear combination of array-bound variables, $p, s \in \mathbb{N}^+$, and $t \in [0, s-1]$. We again refer to the sets $\mathbb{L} = \{l_1, \ldots, l_K\}$ and $\mathbb{U} = \{u_1, \ldots, u_L\}$ of lower and upper bounds, respectively, where l_i and u_j are linear combinations of array-bound variables. Also, let $\mathbf{x_k} = \{x_k \mid k \in \bigcup_{i=1}^K FV(l_i) \cup \bigcup_{j=1}^L FV(u_j)\}$. Further, we assume that $\mathbb{L} \cup \mathbb{U} \neq \emptyset$ and we deal with the case of $\mathbb{L} \cup \mathbb{U} = \emptyset$ later on. We define the template $V(a, b, p, q, s, t, \mathbb{L}, \mathbb{U}) = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ where:

– $\mathbf{x} = \{x_a, x_b\} \cup \mathbf{x_k}$. The counters x_a, x_b, $\mathbf{x_k}$ have the same names in all instances of V.
– $Q = \{q_L, q_R\} \cup \{q_i \mid 0 \leq i < s\}$. The control states are required to have fresh names in every instance of V. $L = \{q_L\} \cup \{q_i \mid 0 \leq i < s\}$ and $R = \{q_R\} \cup \{q_i \mid 0 \leq i \leq s\}$.
– $q_L \xrightarrow{\top} q_L$, $q_R \xrightarrow{\top} q_R$, and $q_L \xrightarrow{\neg(\exists i \,.\, \bigwedge_{l \in \mathbb{L}} i \geq l(\mathbf{x_k}) \,\wedge\, \bigwedge_{u \in \mathbb{U}} i \leq u(\mathbf{x_k}) \,\wedge\, i \equiv_s t)} q_R$.

- $q_L \xrightarrow{\bigwedge_{l\in\mathbb{L}} x_\tau \geq l(\mathbf{x_k})-1 \,\wedge\, \bigvee_{l\in\mathbb{L}} x_\tau+1=l(\mathbf{x_k}) \,\wedge\, x_\tau+1\equiv_s i} q_i$, $0 \leq i < s$.
- $q_i \xrightarrow{\bigwedge_{l\in\mathbb{L}} x_\tau \geq l(\mathbf{x_k}) \,\wedge\, \bigwedge_{u\in\mathbb{U}} x_\tau < u(\mathbf{x_k}) \,\wedge\, x_a - x_b \leq q(\mathbf{x_k})} q_{(i+1) \mod s}$, $0 \leq i < s$ and $i \equiv_s t$.
- $q_i \xrightarrow{\bigwedge_{l\in\mathbb{L}} x_\tau \geq l(\mathbf{x_k}) \,\wedge\, \bigwedge_{u\in\mathbb{U}} x_\tau < u(\mathbf{x_k})} q_{(i+1) \mod s}$, $0 \leq i < s$ and $i \not\equiv_s t$.
- $q_i \xrightarrow{\bigvee_{u\in\mathbb{U}} x_\tau = u(\mathbf{x_k}) \,\wedge\, x_\tau \equiv_s i \,\wedge\, x_a - x_b \leq q(\mathbf{x_k})} q_R$, $0 \leq i < s$ and $i \equiv_s t$.
- $q_i \xrightarrow{\bigvee_{u\in\mathbb{U}} x_\tau = u(\mathbf{x_k}) \,\wedge\, x_\tau \equiv_s i} q_R$, $0 \leq i < s$ and $i \not\equiv_s t$.

In the above, $l(\mathbf{x_k})$, $u(\mathbf{x_k})$, and $q(\mathbf{x_k})$ denote the expressions l, u, and q where each occurrence of an array-bound variable k is replaced by the parameter x_k. As before, we assume that for each transition rule the conjunction $\bigwedge_{k\in FV(\phi)} x'_k = x_k$ is added implicitly to the labelling formula, i.e., the value of an x_k counter stays constant throughout the run. Finally, if $\mathbb{L} = \mathbb{U} = \emptyset$, we define any instance of $V(a,b,p,q,s,t,\emptyset,\emptyset)$ as $A_1 \otimes A_2$ where A_1 is an instance of $V(a,b,p,q,s,t,\emptyset,\{0\})$ and A_2 is an instance of $V(a,b,p,q,s,t,\{0\},\emptyset)$. The intuition behind the construction of V is similar to the one of D.

4.4 Counter Automata for Basic Formulae

We are now ready to define the construction of FBCA for the basic formulae. This is done by composing instances of templates using the $\otimes$ operator for intersection (cf. Section 2). For space reasons, we only give here the construction of the FBCA for (F3) formulae. The formulae of type (F1), (F2), and PA constraints on array-bound variables are treated analogously in [8]. Let φ be an (F3)-type formula

$$\forall i,j\,.\, \underbrace{\bigwedge_{k=1}^{K_1} f_k^1 \leq i \wedge \bigwedge_{l=1}^{L_1} i \leq g_l^1 \wedge \bigwedge_{k=1}^{K_2} f_k^2 \leq j \wedge \bigwedge_{l=1}^{L_2} j \leq g_l^2 \wedge i-j \leq p \wedge i \equiv_s t \wedge j \equiv_u v}_{\phi} \rightarrow a[i]-b[j] \sim q$$

where $0 \leq s < t$ and $0 \leq u < v$. Let $\mathbb{L}_i = \{f_1^i, \ldots, f_{K_i}^i\}$ and $\mathbb{U}_i = \{g_1^i, \ldots, g_{L_i}^i\}$ for $i = 1,2$, respectively. By ϕ, we denote the precondition of φ. The automaton A_φ is defined as $A_\varphi = A_1 \otimes A_2 \otimes A_3$ where A_1, A_2, A_3 are instantiated according to Table 1.

4.5 Assembling Automata for Entire Normalised Formulae

Given a formula $\varphi(\mathbf{k},\mathbf{a})$ which is a positive boolean combination of formulae of types (F1)-(F3) and PA constraints on the array-bound variables $\mathbf{k}$, let A_φ be the automaton defined inductively on the structure of φ as follows:

- if φ is of type (F1)-(F3), or a PA constraint on $\mathbf{k}$, then A_φ is as in Section 4.4,
- if $\varphi = \psi_1 \wedge \psi_2$, then $A_\varphi = A_{\psi_1} \otimes A_{\psi_2}$,
- if $\varphi = \psi_1 \vee \psi_2$, then $A_\varphi = A_{\psi_1} \uplus A_{\psi_2}$.

Let $r \in \mathcal{R}(A_\varphi)$ be an accepting run of A_φ and $\delta(r) = val(r(0))(x_\tau)$ be the value of the x_τ (tick) counter at position 0 on r. We denote by $\eta(r) = r \circ \sigma^{-\delta(r)}$ the *centered run* obtained from r by shifting it such that the value of x_τ at position 0 is also 0. By Lemma 1, r is an accepting run of A_φ if and only if $\eta(r)$ is. Notice that r induces the

Table 1. The instantiation table for (F3) formulae. Note that in some lines, we shift the original bounds appearing in the formula in order to be able to re-use the prepared templates that do not explicitly deal with edges leaving from within the given bounds and going below the lower bound. Due to the way the templates are constructed, the shifting preserves the semantics of the formula—instead of edges going below the lower bound of a certain interval, we obtain the same edges just going above the upper bound of the shifted interval, which our templates are prepared for. Given a set of integers S and an integer p, we use the notation $S+p$ for $\{s+p \mid s \in S\}$.

p	$\sim$	A_1	A_2	A_3
∞	$\leq$	$V(a,t_\varphi,q,s,t,\mathbb{L}_1,\mathbb{U}_1)$	$H(t_\varphi,\mathtt{bi},\exists i,j.\phi)$	$V(t_\varphi,b,0,u,v,\mathbb{L}_2,\mathbb{U}_2)$
∞	$\geq$	$V(b,t_\varphi,-q,u,v,\mathbb{L}_2,\mathbb{U}_2)$	$H(t_\varphi,\mathtt{bi},\exists i,j.\phi)$	$V(t_\varphi,a,0,s,t,\mathbb{L}_1,\mathbb{U}_1)$
0	$\leq$	$V(a,t_\varphi,q,s,t,\mathbb{L}_1,\mathbb{U}_1)$	$H(t_\varphi,\mathtt{right},\exists i,j.\phi)$	$V(t_\varphi,b,0,u,v,\mathbb{L}_2,\mathbb{U}_2)$
0	$\geq$	$V(b,t_\varphi,-q,u,v,\mathbb{L}_2,\mathbb{U}_2)$	$H(t_\varphi,\mathtt{left},\exists i,j.\phi)$	$V(t_\varphi,a,0,s,t,\mathbb{L}_1,\mathbb{U}_1)$
>0	$\leq$	$D(a,t_\varphi,p,q,s,t-p,\mathbb{L}_1-p,\mathbb{U}_1-p,\mathtt{left})$	$H(t_\varphi,\mathtt{right},\exists i,j.\phi)$	$V(t_\varphi,b,0,u,v,\mathbb{L}_2,\mathbb{U}_2)$
>0	$\geq$	$D(b,t_\varphi,p,-q,u,v,\mathbb{L}_2,\mathbb{U}_2,\mathtt{right})$	$H(t_\varphi,\mathtt{left},\exists i,j.\phi)$	$V(t_\varphi,a,0,s,t,\mathbb{L}_1,\mathbb{U}_1)$
<0	$\leq$	$D(a,t_\varphi,-p,q,s,t,\mathbb{L}_1,\mathbb{U}_1,\mathtt{right})$	$H(t_\varphi,\mathtt{right},\exists i,j.\phi)$	$V(t_\varphi,b,0,u,v,\mathbb{L}_2,\mathbb{U}_2)$
<0	$\geq$	$D(b,t_\varphi,-p,-q,u,v+p,\mathbb{L}_2+p,\mathbb{U}_2+p,\mathtt{left})$	$H(t_\varphi,\mathtt{left},\exists i,j.\phi)$	$V(t_\varphi,a,0,s,t,\mathbb{L}_1,\mathbb{U}_1)$

following valuations on $\mathbf{k}$ and $\mathbf{a}$, respectively: $\iota_r(k) = val(\eta(r)(0))(x_k)$ for all $k \in \mathbf{k}$, and $\mu_r(a)(i) = val(\eta(r)(i))(x_a)$ for all $a \in \mathbf{a}$ and $i \in \mathbb{Z}$.

For an arbitrary valuation $\nu \in \mathcal{V}(A_\varphi)$, there exists $r \in \mathcal{R}(A_\varphi)$ such that $\nu = val(r)$. Let $M_\varphi(\nu) = \langle \iota_r, \mu_r \rangle$ be the valuation of the free variables in φ that corresponds to r. One can see now that M_φ defines a function $M_\varphi : \mathcal{V}(A_\varphi) \rightarrow (\mathbf{k} \mapsto \mathbb{Z}) \times (\mathbf{a} \mapsto {}^{\omega}\mathbb{Z}^{\omega})$.[5]

Theorem 1. *Let $\varphi(\mathbf{k},\mathbf{a})$ be a positive boolean combination of formulae of types (F1)-(F3) and PA constraints on the array-bound variables $\mathbf{k}$, and A_φ be the automaton defined in the previous. Then, $M_\varphi(\mathcal{V}(A_\varphi)) = [\![\varphi]\!]$.*

The proof is by induction on the structure of φ. For the base case, we use the correspondence between models and constraint graphs of formulae (F1)-(F3) (Lemma 6). The inductive step follows as a consequence of the fact that the class of FBCA is closed under union and intersection (Lemma 3). The main result of the paper is the following:

Corollary 1. *The logic* **LIA** *is decidable.*

The proof of Corollary 1 uses the normalisation step (cf. Lemma 5) to rewrite any formula of **LIA** into the form (NF) and applies Theorem 1 to the matrix of the formula (i.e., the formula obtained by skipping the existential quantifier prefix).

5 Conclusions and Future Work

We have presented a new decidable logic for reasoning about properties of programs with integer arrays. This logic allows one to relate adjacent array values as well as to

[5] By definition, for each $\nu \in \mathcal{V}(A_\varphi)$, there exist valuations ι_r and μ_r, so M_φ is defined for all $\nu \in \mathcal{V}(A_\varphi)$. Let $r_1, r_2 \in \mathcal{R}(A_\varphi)$ be two runs such that $val(r_1) = val(r_2) = \nu$. We have $\delta(r_1) = \delta(r_2)$, therefore $\eta(r_1) = \eta(r_2)$, which leads to $\iota_{r_1} = \iota_{r_2}$ and $\mu_{r_1} = \mu_{r_2}$.

express periodic facts relating all values situated at equidistant positions. We have established decidability of this logic by following the automata-theoretic approach. To this end, we have defined a new class of Büchi automata with counters, for which emptiness is decidable. We translate each formula into a corresponding automaton of this kind and transform deciding satisfiability of the formula to deciding emptiness of the automaton.

Future work will include the study of the complexity of our decision procedure and its implementation. We furthermore plan to develop invariant generation methods in order to give automatic correctness proofs for programs with integer arrays.

References

1. Armando, A., Ranise, S., Rusinowitch, M.: Uniform Derivation of Decision Procedures by Superposition. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, p. 2001. Springer, Heidelberg (2001)
2. Arons, T., Pnueli, A., Ruah, S., Xu, J., Zuck, L.: Parameterized Verification with Automatically Computed Inductive Assertions. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, Springer, Heidelberg (2001)
3. Bouajjani, A., Jurski, Y., Sighireanu, M.: A Generic Framework for Reasoning About Dynamic Networks of Infinite-State Processes. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, Springer, Heidelberg (2007)
4. Bozga, M., Iosif, R., Lakhnech, Y.: Flat Parametric Counter Automata. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, Springer, Heidelberg (2006)
5. Bradley, A.R., Manna, Z., Sipma, H.B.: What 's Decidable About Arrays? In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, Springer, Heidelberg (2005)
6. Comon, H., Jurski, Y.: Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, Springer, Heidelberg (1998)
7. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decision Procedures for Extensions of the Theory of Arrays. Annals of Mathematics and Artificial Intelligence 50 (2007)
8. Habermehl, P., Iosif, R., Vojnar, T.: What else is decidable about integer arrays? Technical Report TR-2007-8, Verimag (2007)
9. Jaffar, J.: Presburger Arithmetic with Array Segments. Inform. Proc. Letters 12 (1981)
10. King, J.: A Program Verifier. PhD thesis, Carnegie Mellon University (1969)
11. Mateti, P.: A Decision Procedure for the Correctness of a Class of Programs. Journal of the ACM 28(2) (1980)
12. McCarthy, J.: Towards a Mathematical Science of Computation. In: IFIP Congress (1962)
13. Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Inc., Englewood Cliffs (1967)
14. Nivat, M., Perrin, D.: Ensembles reconnaissables de mots biinfinis. Canad. J. Math. 38, 513–537 (1986)
15. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves, Warsaw, Poland, pp. 92–101 (1929)
16. Stump, A., Barrett, C.W., Dill, D.L., Levitt, J.R.: A Decision Procedure for an Extensional Theory of Arrays. In: Proc. of LICS 2001 (2001)
17. Suzuki, N., Jefferson, D.: Verification Decidability of Presburger Array Programs. Journal of the ACM 27(1) (1980)
18. Thomas, W.: Automata on Infinite Objects. In: Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, Elsevier, Amsterdam (1990)