What Else Is Decidable about Integer Arrays?

Peter Habermehl, Radu Iosif, Tomáš Vojnar

▶ **To cite this version:**

Peter Habermehl, Radu Iosif, Tomáš Vojnar. What Else Is Decidable about Integer Arrays?. Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Mar 2008, Budapest, Hungary. 10.1007/978-3-540-78499-9_33 . hal-01418914

HAL Id: hal-01418914
https://hal.science/hal-01418914v1

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.

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, e-mail: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 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 the 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. We show interesting program properties expressible in our logic, and give an example of invariant verification for programs that handle integer arrays.

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 expressed 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 $\mathbf{i}$ (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 array a are equal to 0.

In the absence of specific syntactic restrictions, a logic with such an expressive power can be easily shown to be undecidable, as one can encode the histories 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. The logic **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, $\mathbf{a}$ (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 Appendix B, we give an example program showing the usefulness of this logic to express verification conditions.

In this paper, we prove the 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 A_φ is not empty. Decidability of the logic follows from the 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 the 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 an 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 the 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 Appendix D.

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 $\varphi(x)$, we denote by $\varphi[t/x]$ the formula in which each occurrence of x is replaced by a term t. Given a formula φ, we denote by $\models \varphi$ the fact that φ is logically valid, i.e. it holds in every structure corresponding to its signature. 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 (DBM) and Presburger Arithmetic (PA).

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, by quantifier elimination [15]. Clearly, every DBM formula is also in PA.

A *counter automaton* 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 $\to$ the 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 counter automaton 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' if and only if there exists a sequence of configurations $c = c_0 c_1 \ldots c_n = c'$ 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 $\beta:\mathbb{Z}\to S$.[4] We denote by ${}^{\omega}S^{\omega}$ the set of all bi-infinite sequences over S. A *bi-infinite Büchi counter automaton* 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 bi-infinite Büchi automaton 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 by $val(r)=\ldots val(r(-1))val(r(0))val(r(1))\ldots$ the bi-infinite sequence of valuations in r, and $\mathcal{V}(A)=\{val(r)\mid r\in\mathcal{R}(A)\}$.

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

A *control path* in a counter automaton 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, except the first one, appears only once. A counter automaton 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, and whose elementary cycles are labelled with parametric DBM formulae. We call this class FBCA in the following. We prove that the emptiness problem for FBCA is decidable, using results of [4], and their extensions, that can be found in Appendix A.

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

The FBCA class is also effectively closed under the operations of union and intersection. However, before proceeding, we need to elucidate the meaning of these operations for counter automata. 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 $A_i=\langle\mathbf{x_i},Q_i,L_i,R_i,\to_i\rangle$,

[4] 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 reasons, here we formally distinguish the bi-infinite sequences β, $\beta\circ\sigma^n$, and $\beta\circ\sigma^{-n}$.

$i = 1,2$, 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 the intervals of the 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 $\top$ symbol 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 $f \leq i$, $i \leq f$, $i - j \leq n$, $i \equiv_s t$, 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.

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 a up to a certain bound is defined as $\exists k \; \forall i \,.\, 0 \leq i < k \rightarrow a[i] - a[i+1] \leq -1$. The fact that the first k elements of array a are below the first l elements of array b at distance 5 is defined as $\exists k,l \; \forall i,j \,.\, 0 \leq i < k \wedge 0 \leq j < l \rightarrow a[i] - b[j] \leq -5$. Equality of two arrays up to a certain bound can be expressed as $\exists n \forall i \,.\, 0 \leq i < n \rightarrow a[i] = b[i]$. The use of modulo constraints as guards for indices allows one to express periodic facts, e.g. $\forall i,j \,.\, i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] \leq a[j]$, meaning that any value at some even position is

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

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

less than or equal to any value at some odd position in a. Appendix B shows 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 to conveniently deal with out-of-bound reference situations quite 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 formula of **LIA**. A *valuation* is a pair of partial functions[5] $\langle \iota, \mu \rangle$, with $\iota : \mathit{BVar} \cup \mathit{IVar} \to \mathbb{Z}_\perp$, associating an integer value with every free integer variable, and $\mu : \mathit{AVar} \to {}^{\omega}\mathbb{Z}^{\omega}_{\perp}$, 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:

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

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 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 $\forall \mathbf{i}.G \to V_1 \vee \ldots \vee V_n$, 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.*

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

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 automata from **LIA** formulae, we first normalize them into existentially quantified positive boolean combinations 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 normalized formulae, with the $\uplus$ and $\otimes$ operators being the counterparts for the $\vee$ and $\wedge$ connectives, respectively.

4.1 Normalization 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.

$$\exists \mathbf{k} \exists \mathbf{a} \,.\, \bigvee_p \Big(\bigwedge_q \phi_{pq}(\mathbf{a}, \mathbf{k}) \Big) \wedge \theta_p(\mathbf{k}) \tag{NF}$$

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

- θ_p 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_{pq} is a formula of the following forms, for some $m \in \mathbb{N}$, $0 \leq t < s$, $0 \leq v < u$, and $p \in \mathbb{Z}$, $q \in \mathbb{Z}^\infty$:

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

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

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

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.

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

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 into 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 $\iota : \mathbf{k} \to \mathbb{Z}$ 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 be thus 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 $(a,0) \xrightarrow{5} (t_\varphi,-3) \xrightarrow{0} \ldots \xrightarrow{0} (t_\varphi,1) \xrightarrow{0} (b,1)$ for the constraint $a[0] - b[1] \leq 5$, another path $(a,0) \xrightarrow{5} (t_\varphi,-3) \xrightarrow{0} \ldots \xrightarrow{0} (t_\varphi,3) \xrightarrow{0} (b,3)$ for the constraint $a[0] - b[3] \leq 5$, etc.

Formally, the constraint graph of φ is $G_{\iota,\varphi} = \langle V, E \rangle$ with the set of vertices $V = (\mathcal{A} \cup \mathcal{T} \cup \{\zeta\}) \times \mathbb{Z}$, where $\mathcal{A} = \{a,b\}$ are the array symbols in φ, $\mathcal{T} = \{t_\varphi\}$ are the auxiliary symbols (tracks), and ζ is a special symbol (zero track). The set of edges E is defined based on the type of φ, i.e. (F1)-(F3). 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 Appendix C. 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 $0 \leq s < t$, $0 \leq u < v$, $p \in \mathbb{Z}^\infty$, and $q \in \mathbb{Z}$:

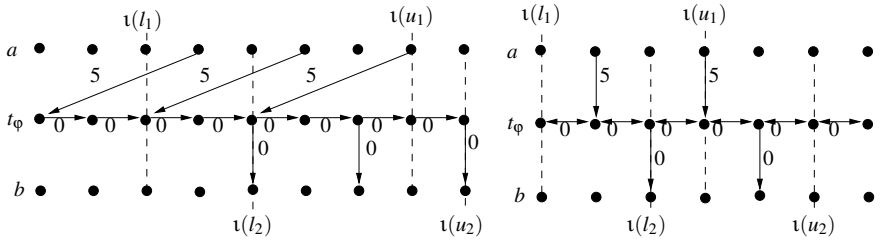
$$\forall i, j . \underbrace{\bigwedge_{k=1}^{K_1} f_k^1 \leq i \wedge \bigwedge_{l=1}^{L_1} i \leq g_l^1 \wedge i \equiv_s t}_{\phi^1} \wedge \underbrace{\bigwedge_{k=1}^{K_2} f_k^2 \leq j \wedge \bigwedge_{l=1}^{L_2} j \leq g_l^2 \wedge j \equiv_u v}_{\phi^2} \wedge i - j \leq p \rightarrow a[i] -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

Let $\phi^1(i,\mathbf{k})$ and $\phi^2(j,\mathbf{k})$ be the subformulae defining the ranges of i and j, respectively, and $\mathcal{P}_\iota^1 = \{n \in \mathbb{Z} \mid \models \phi^1{}_\iota[n/i]\}$, $\mathcal{P}_\iota^2 = \{n \in \mathbb{Z} \mid \models \phi^2{}_\iota[n/j]\}$, be these ranges under

the valuation ι. Let $T_{\leq} = \{(t_\varphi, k) \xrightarrow{0} (t_\varphi, k+1) \mid \exists n \in \mathcal{P}^1_\iota \ \exists m \in \mathcal{P}^2_\iota \ . \ n - m \leq p\}$ and $T_{\leq} = \{(t_\varphi, k) \xrightarrow{0} (t_\varphi, k-1) \mid \exists n \in \mathcal{P}^1_\iota \ \exists m \in \mathcal{P}^2_\iota \ . \ n - m \geq p\}$. Note that $T_{\leq}$ and $T_{\geq}$ are empty is 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 $a[i] - b[j] \leq q$, we have (Fig. 2(a)):
   $E \supset \{(a,k) \xrightarrow{q} (t_\varphi, k-p) \mid k \in \mathcal{P}^1_\iota\} \cup \{(t_\varphi, k) \xrightarrow{0} (b,k) \mid k \in \mathcal{P}^2_\iota\} \cup T_{\leq}$
   (b) for $a[i] - b[j] \geq q$, we have:
   $E \supset \{(b,k) \xrightarrow{-q} (t_\varphi, k+p) \mid k \in \mathcal{P}^2_\iota\} \cup \{(t_\varphi, k) \xrightarrow{0} (a,k) \mid k \in \mathcal{P}^1_\iota\} \cup T_{\geq}$
2. If $p = \infty$, we consider again two cases, based on the direction of $a[i] - b[j] \sim q$:
   (a) for $a[i] - b[j] \leq q$, we have (Fig. 2(b)):
   $E \supset \{(a,k) \xrightarrow{q} (t_\varphi, k) \mid k \in \mathcal{P}^1_\iota\} \cup \{(t_\varphi, k) \xrightarrow{0} (b,k) \mid k \in \mathcal{P}^2_\iota\} \cup T_{\leq} \cup T_{\geq}$
   (b) for $a[i] - b[j] \geq q$, we have:
   $E \supset \{(b,k) \xrightarrow{-q} (t_\varphi, k) \mid k \in \mathcal{P}^2_\iota\} \cup \{(t_\varphi, k) \xrightarrow{0} (a,k) \mid k \in \mathcal{P}^1_\iota\} \cup T_{\leq} \cup T_{\geq}$

Nothing else is in E.



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

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

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

Relating constraint graphs and models of formulae Let us point out the correspondence between constraint graphs and models of formulae of the forms (F1)-(F3), i.e. 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 proves the 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 $Lab(v_1) - Lab(v_2) \leq k$ 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.[6] 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 synchronism 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. The $\mathbf{x_k}$ parameters are used within guards of the form $x_\tau \sim f(\mathbf{x_k})$, where $\sim \in \{\leq, \geq\}$ and f is a linear combination of $\mathbf{x_k}$, in order to mark the position of the array boundaries, during the run of the automata.

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

If, for a given valuation of the parameters $\mathbf{x_k}$, the formula ϕ holds, 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 for the given valuation of $\mathbf{x_k}$, ϕ does not hold, 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 respectively upper bounds, where l_i and u_j are linear combinations of array-bound variables, and 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 $\mathbb{L} \cup \mathbb{U} \neq \emptyset$ – 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 diagonal edges that span over more than one position, into series of diagonal edges connecting only adjacent positions.[7]
- $Q = \{q_L,q_R\} \cup \{q_i \mid 0 \leq i < s\} \cup \{q_i^j \mid 0 \leq j < s,\ j+1 \leq i < j+p\}$. 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 $dir = \texttt{right}$, $\xi_i = \bigwedge_{k\in K_i} x_k - x'_{k+1} \leq \alpha_k$, for $K_i = \{k \mid 0 \leq k < p,\ i \equiv_s k+t\}$, $\alpha_0 = q$ and $\alpha_k = 0$, $k > 0$,

[7] 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 neighboring 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).

– if $dir = \texttt{left}$, $\xi_i = \bigwedge_{k \in K_i} x'_{k-1} - x_k \leq \alpha_k$, $K_i = \{k \mid 1 \leq k \leq p,\ k+i \equiv_s t\}$, $\alpha_1 = q$ 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 $D(a,b,p,q,s,t,\{0\},\emptyset,dir)$.

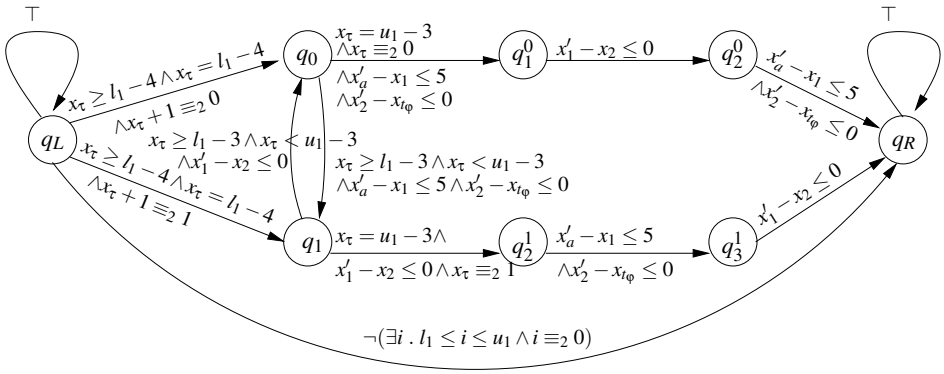


Fig. 3. The FBCA for the diagonal edges in the formula $\varphi : \forall i, j . l_1 \leq i \leq u_1 \wedge l_2 \leq j \leq u_2 \wedge i - j \leq 3 \wedge i \equiv_2 0 \wedge j \equiv_2 1 \rightarrow a[i] - b[j] \leq 5$ from Fig. 2(a) obtained as $D(a, t_\varphi, 3, 5, 2, 0-3, \{l_1 - 3\}, \{u_1 - 3\}, \texttt{left})$. 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.

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 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 $q_i \rightarrow q^i_{i+1}$ (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^j_i 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, $q \in \mathbb{Z}$, $p,s \in \mathbb{N}^+$, 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 respectively upper bounds, 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 $\mathbb{L} \cup \mathbb{U} \neq \emptyset$ – 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 $q_L \xrightarrow{\neg(\exists i \,.\, \bigwedge_{l \in \mathbb{L}} i \geq l(\mathbf{x_k}) \wedge \bigwedge_{u \in \mathbb{U}} i \leq u(\mathbf{x_k}) \wedge i \equiv_s t)} q_R$.
- $q_L \xrightarrow{\bigwedge_{l \in \mathbb{L}} x_\tau \geq l(\mathbf{x_k}) - 1 \wedge \bigvee_{l \in \mathbb{L}} x_\tau + 1 = l(\mathbf{x_k}) \wedge x_\tau + 1 \equiv_s i} q_i$, $0 \leq i < s$.
- $q_i \xrightarrow{\bigwedge_{l \in \mathbb{L}} x_\tau \geq l(\mathbf{x_k}) \wedge \bigwedge_{u \in \mathbb{U}} x_\tau < u(\mathbf{x_k}) \wedge x_a - x_b \leq q} q_{(i+1) \mod s}$, $0 \leq i < s$ 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 $i \not\equiv_s t$.
- $q_i \xrightarrow{\bigvee_{u \in \mathbb{U}} x_\tau = u(\mathbf{x_k}) \wedge x_\tau \equiv_s i \wedge x_a - x_b \leq q} q_R$, $0 \leq i < s$ 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, $l(\mathbf{x_k})$ and $u(\mathbf{x_k})$ denote the expressions l and u in which each occurrence of an array-bound variable k is replaced by the 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. 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 Appendix C. Let φ be the (F3)-type formula:

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

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.

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

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 $\{s+p \mid s \in S\}$

4.5 From Formulae to Counter Automata

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 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 correspond to r. One can see now that M_φ defines a function $M_\varphi : \mathcal{V}(A_\varphi) \to (\mathbf{k} \mapsto \mathbb{Z}) \times (\mathbf{a} \mapsto {}^{\omega}\mathbb{Z}^{\omega})$.[8]

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.*

[8] 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 $\mu_{r_1} = \mu_{r_2}$.

The proof of Corollary 1 uses the normalization 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 present a new decidable logic for reasoning about properties of programs handling integer arrays. This logic allows to relate adjacent array values, as well as to express periodic facts relating all values situated at equidistant positions. We establish decidability of this logic following the automata-theoretic approach. To this end, we define a new class of Büchi automata with counters, for which emptiness is decidable, and translate each formula into a corresponding 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. A. Armando, S. Ranise, and M. Rusinowitch. Uniform Derivation of Decision Procedures by Superposition. In *Proc of CSL'01*, LNCS 2142, 2001.
2. T. Arons, A. Pnueli, S. Ruah, J. Xu, and L.Zuck. Parameterized Verification with Automatically Computed Inductive Assertions. In *CAV'01*, LNCS 2102. Springer, 2001.
3. A. Bouajjani, Y. Jurski, and M. Sighireanu. A Generic Framework for Reasoning About Dynamic Networks of Infinite-State Processes. In *TACAS'07*, LNCS 4424. Springer, 2007.
4. M. Bozga, R. Iosif, and Y. Lakhnech. Flat Parametric Counter Automata. In *Proc. of ICALP'06*, LNCS 4052. Springer, 2006.
5. A.R. Bradley, Z. Manna, and H.B. Sipma. What 's Decidable About Arrays? In *Proc. of VMCAI'06*, LNCS 3855. Springer, 2006.
6. H. Comon and Y. Jurski. Multiple Counters Automata, Safety Analysis and Presburger Arithmetic. In *Proc. of CAV'98*, LNCS 1427. Springer, 1998.
7. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decision Procedures for Extensions of the Theory of Arrays. *Annals of Mathematics and Artificial Intelligence*, 50, 2007.
8. P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? Technical Report TR-2007-8, Verimag, 2007.
9. J. Jaffar. Presburger Arithmetic with Array Segments. *Inform. Proc. Letters*, 12, 1981.
10. J. King. *A Program Verifier*. PhD thesis, Carnegie Mellon University, 1969.
11. P. Mateti. A Decision Procedure for the Correctness of a Class of Programs. *Journal of the ACM*, 28(2), 1980.
12. J. McCarthy. Towards a Mathematical Science of Computation. In *IFIP Congress*, 1962.
13. M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.
14. M. Nivat and D. Perrin. Ensembles reconnaissables de mots biinfinis. *Canad. J. Math.*, 38:513–537, 1986.
15. M. Presburger. 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*, pages 92–101, Warsaw, Poland, 1929.
16. A. Stump, C.W. Barrett, D.L. Dill, and J.R. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *Proc. of LICS'01*, 2001.
17. N. Suzuki and D. Jefferson. Verification Decidability of Presburger Array Programs. *Journal of the ACM*, 27(1), 1980.
18. W. Thomas. Automata on Infinite Objects. In *Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics*. Elsevier, 1990.

A Extensions of Flat Counter Automata

In [6, 4] it is shown that given a flat counter automaton $A = \langle \mathbf{x}, Q, \rightarrow \rangle$, and a loop γ on a control state $q \in Q$, labelled with DBM formulae only, one can effectively build a PA formula $\Psi_{q,\gamma}(y, \mathbf{x}, \mathbf{x}')$ which is satisfied by all triples $\langle n, \mathbf{v}, \mathbf{v}' \rangle$, where there exists an execution corresponding to n loop iterations, in which the initial values of the counters are $\mathbf{v}$ and the final values are $\mathbf{v}'$. Based on this, we prove two important lemmas whose proofs can be found in [8].

Lemma 7. *If any control loop of A is labelled by a parametric DBM formula, then for any two control states $q, q' \in Q$, one can effectively build a PA formula $R_{q,q'}(\mathbf{x}, \mathbf{x}')$ such that, for any two configurations (q, ν) and (q', ν'), (q', ν') is a successor of (q, ν) if and only if $\models R_{q,q'}(\nu(\mathbf{x}), \nu'(\mathbf{x}'))$.*

Lemma 8. *Given a control loop γ labelled by a parametric DBM formula, and a control state q on γ, one can effectively build a PA formula $I_{q,\gamma}(\mathbf{x})$, such that, for any configuration (q, ν), there exists an infinite computation along γ, starting with (q, ν) if and only if $\models I_{q,\gamma}(\nu(\mathbf{x}))$.*

B Verification Conditions for an Array Merging Example

Consider the following program that takes two arrays a and b, and merges their first n elements by alternating elements from a with elements from b. Suppose, moreover, that the first n elements of a are less than or equal to the first n elements of b. The resulting array will have all its first n elements on even positions less than or equal to the first n elements on odd positions.

$\{\{\ \mathbf{n} > \mathbf{0} \wedge \forall \mathbf{i}, \mathbf{j}\ .\ \mathbf{0} \leq \mathbf{i}, \mathbf{j} < \mathbf{n} \rightarrow \mathbf{a}[\mathbf{i}] \leq \mathbf{b}[\mathbf{j}]\ \}\}$
for (k=0, l=0; k < n; k++, l+=2)
  $\{\{\ \mathbf{n} > \mathbf{0} \wedge \mathbf{k} \leq \mathbf{n} \wedge \mathbf{l} = \mathbf{2k} \wedge$
    $\forall \mathbf{i}, \mathbf{j}\ .\ \mathbf{0} \leq \mathbf{i}, \mathbf{j} < \mathbf{2k} \wedge \mathbf{i} \equiv_2 \mathbf{0} \wedge \mathbf{j} \equiv_2 \mathbf{1} \rightarrow \mathbf{c}[\mathbf{i}] \leq \mathbf{c}[\mathbf{j}] \wedge$
    $\forall \mathbf{i}, \mathbf{j}\ .\ \mathbf{0} \leq \mathbf{i}, \mathbf{j} < \mathbf{n} \rightarrow \mathbf{a}[\mathbf{i}] \leq \mathbf{b}[\mathbf{j}]\ \}\}$
{ c[l] = a[k];
  c[l + 1] = b[k]; }
$\{\{\ \mathbf{n} > \mathbf{0} \wedge \forall \mathbf{i}, \mathbf{j}\ .\ \mathbf{0} \leq \mathbf{i}, \mathbf{j} < \mathbf{2n} \wedge \mathbf{i} \equiv_2 \mathbf{0} \wedge \mathbf{j} \equiv_2 \mathbf{1} \rightarrow \mathbf{c}[\mathbf{i}] \leq \mathbf{c}[\mathbf{j}]\ \}\}$

The pre-, post-condition, and loop invariant needed for the proof of this program are annotated directly into the program text using double curly braces. We show in the following that the verification conditions to be checked to prove the correctness of the program fall into our logic, and so they are decidable.

We need to check three verification conditions corresponding to the initialisation of the loop, the loop body, and the finalisation of the loop.

The *initialisation* consists of the two unconditional assignment statements k=0 and l=0. We need to check that the following formula is logically valid (we use primed names of variables to distinguish the current and future values of the variables):

$$
\begin{array}{l}
\forall\, a,a',b,b',c,c',n,n',k,k',l,l'.\\
\quad n>0 \;\wedge\; (\forall i,j\;.\; 0\le i,j<n \rightarrow a[i]\le b[j]) \;\wedge\; k'=0 \;\wedge\; l'=0 \;\wedge\; n'=n \;\wedge\\
\quad (\forall i\;.\; a'[i]=a[i]) \;\wedge\; (\forall i\;.\; b'[i]=b[i]) \;\wedge\; (\forall i\;.\; c'[i]=c[i])\\
\quad \longrightarrow\\
\quad n'>0 \;\wedge\; k'\le n' \;\wedge\; l'=2k' \;\wedge\\
\quad (\forall i,j\;.\; 0\le i,j<2k' \;\wedge\; i\equiv_2 0 \;\wedge\; j\equiv_2 1 \rightarrow c'[i]\le c'[j]) \;\wedge\\
\quad (\forall i,j\;.\; 0\le i,j<n \rightarrow a'[i]\le b'[j])
\end{array}
$$

However, checking the validity of the above formula is equal to checking that its negation, which clearly fits our logic, is unsatisfiable:

$$
\begin{array}{l}
\exists\, a,a',b,b',c,c',n,n',k,k',l,l'.\\
\quad n>0 \;\wedge\; (\forall i,j\;.\; 0\le i,j<n \rightarrow a[i]\le b[j]) \;\wedge\; k'=0 \;\wedge\; l'=0 \;\wedge\; n'=n \;\wedge\\
\quad (\forall i\;.\; a'[i]=a[i]) \;\wedge\; (\forall i\;.\; b'[i]=b[i]) \;\wedge\; (\forall i\;.\; c'[i]=c[i])\\
\quad \wedge\\
\quad (n'\le 0 \;\vee\; k'>n' \;\vee\; l'<2k' \;\vee\; l'>2k' \;\vee\\
\quad\; (\exists i,j\;.\; 0\le i,j<2k' \;\wedge\; i\equiv_2 0 \;\wedge\; j\equiv_2 1 \;\wedge\; c'[i]>c'[j]) \;\vee\\
\quad\; (\exists i,j\;.\; 0\le i,j<n \;\wedge\; a'[i]>b'[j]))
\end{array}
$$

To see this, note that the existentially quantified index variables in the last two lines of the above formula can be given unique names and the appropriate quantifiers moved to the prefix of the formula.

To check the effect of the *loop body*, i.e. the assignments c[l] = a[k], c[l+1] = b[k], k++, and l+=2 which are executed provided that k<n, we have to prove that the following holds:

$$
\begin{array}{l}
\forall\, a,a',b,b',c,c',n,n',k,k',l,l'.\\
\quad n>0 \;\wedge\; k\le n \;\wedge\; l=2k \;\wedge\\
\quad (\forall i,j\;.\; 0\le i,j<2k \;\wedge\; i\equiv_2 0 \;\wedge\; j\equiv_2 1 \rightarrow c[i]\le c[j]) \;\wedge\\
\quad (\forall i,j\;.\; 0\le i,j<n \rightarrow a[i]\le b[j]) \;\wedge\\
\quad k<n \;\wedge\; k'=k+1 \;\wedge\; l'=l+2 \;\wedge\; n'=n \;\wedge\\
\quad (\forall i\;.\; a'[i]=a[i]) \;\wedge\; (\forall i\;.\; b'[i]=b[i]) \;\wedge\\
\quad (\forall i\;.\; i<l \rightarrow c'[i]=c[i]) \;\wedge\; (\forall i\;.\; i>l+1 \rightarrow c'[i]=c[i]) \;\wedge\\
\quad c'[l]=a[k] \;\wedge\; c'[l+1]=b[k]\\
\quad \longrightarrow\\
\quad n'>0 \;\wedge\; k'\le n' \;\wedge\; l'=2k' \;\wedge\\
\quad (\forall i,j\;.\; 0\le i,j<2k' \;\wedge\; i\equiv_2 0 \;\wedge\; j\equiv_2 1 \rightarrow c'[i]\le c'[j]) \;\wedge\\
\quad (\forall i,j\;.\; 0\le i,j<n \rightarrow a'[i]\le b'[j])
\end{array}
$$

Again, checking the validity of the above formula is equal to checking that its negation, is unsatisfiable:

$$
\begin{array}{l}
\exists\, a,a',b,b',c,c',n,n',k,k',l,l'.\\
\quad n>0 \;\wedge\; k\le n \;\wedge\; l=2k \;\wedge\\
\quad (\forall i,j\;.\; 0\le i,j<2k \;\wedge\; i\equiv_2 0 \;\wedge\; j\equiv_2 1 \rightarrow c[i]\le c[j]) \;\wedge\\
\quad (\forall i,j\;.\; 0\le i,j<n \rightarrow a[i]\le b[j]) \;\wedge\\
\quad k<n \;\wedge\; k'=k+1 \;\wedge\; l'=l+2 \;\wedge\; n'=n \;\wedge\\
\quad (\forall i\;.\; a'[i]=a[i]) \;\wedge\; (\forall i\;.\; b'[i]=b[i]) \;\wedge\\
\quad (\forall i\;.\; i<l \rightarrow c'[i]=c[i]) \;\wedge\; (\forall i\;.\; i>l+1 \rightarrow c'[i]=c[i]) \;\wedge\\
\quad c'[l]=a[k] \;\wedge\; c'[l+1]=b[k]\\
\quad \wedge
\end{array}
$$

$$
\begin{array}{l}
(n' \leq 0 \ \vee \ k' > n' \ \vee \ l' < 2k' \ \vee \ l' > 2k' \ \vee \\
\ (\exists i,j \ . \ 0 \leq i,j < 2k' \ \wedge \ i \equiv_2 0 \ \wedge \ j \equiv_2 1 \ \wedge \ c'[i] > c'[j]) \ \vee \\
\ (\exists i,j \ . \ 0 \leq i,j < n \ \wedge \ a'[i] > b'[j]))
\end{array}
$$

Finally, in order to check the *finalisation of the loop* (i.e. the exit of the loop when k $\geq$ n), one has to check the validity of the following formula:

$$
\begin{array}{l}
\forall \ a,a',b,b',c,c',n,n',k,k',l,l'. \\
\quad n > 0 \ \wedge \ k \leq n \ \wedge \ l = 2k \ \wedge \\
\quad (\forall i,j \ . \ 0 \leq i,j < 2k \ \wedge \ i \equiv_2 0 \ \wedge \ j \equiv_2 1 \rightarrow c[i] \leq c[j]) \ \wedge \\
\quad (\forall i,j \ . \ 0 \leq i,j < n \rightarrow a[i] \leq b[j]) \ \wedge \\
\quad k \geq n \ \wedge \ k' = k \ \wedge \ l' = l \ \wedge \ n' = n \ \wedge \\
\quad (\forall i \ . \ a'[i] = a[i]) \ \wedge \ (\forall i \ . \ b'[i] = b[i]) \ \wedge \ (\forall i \ . \ c'[i] = c[i]) \ \wedge \\
\quad \longrightarrow \\
\quad n' > 0 \ \wedge \ (\forall i,j \ . \ 0 \leq i,j < 2n' \ \wedge \ i \equiv_2 0 \ \wedge \ j \equiv_2 1 \rightarrow c'[i] \leq c'[j])
\end{array}
$$

Like in the previous cases, checking the validity of the above formula is equal to checking that its negation is unsatisfiable:

$$
\begin{array}{l}
\exists \ a,a',b,b',c,c',n,n',k,k',l,l'. \\
\quad n > 0 \ \wedge \ k \leq n \ \wedge \ l = 2k \ \wedge \\
\quad (\forall i,j \ . \ 0 \leq i,j < 2k \ \wedge \ i \equiv_2 0 \ \wedge \ j \equiv_2 1 \rightarrow c[i] \leq c[j]) \ \wedge \\
\quad (\forall i,j \ . \ 0 \leq i,j < n \rightarrow a[i] \leq b[j]) \ \wedge \\
\quad k \geq n \ \wedge \ k' = k \ \wedge \ l' = l \ \wedge \ n' = n \ \wedge \\
\quad (\forall i \ . \ a'[i] = a[i]) \ \wedge \ (\forall i \ . \ b'[i] = b[i]) \ \wedge \ (\forall i \ . \ c'[i] = c[i]) \ \wedge \\
\quad \wedge \\
\quad (n' \leq 0 \ \vee \ \ (\exists i,j \ . \ 0 \leq i,j < 2n' \ \wedge \ i \equiv_2 0 \ \wedge \ j \equiv_2 1 \ \wedge \ c'[i] > c'[j]))
\end{array}
$$

C Constraint Graphs and Counter Automata for (F1) and (F2)

Here we define the constraint graphs for formulae of type (F1) and (F2) and the corresponding counter automata as well as the automaton for Presburger bound constraint formulae.

C.1 Constraint graphs

Formulae of type (F1) Let φ be the formula

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

where $0 \leq t < s$. Let $\mathcal{P}_\iota = \{n \in \mathbb{Z} \mid \ \models \phi_\iota[n/i]\}$.

The set of edges E is defined by the following case split:

1. If the right hand side of the implication is $a[i] \leq h(\mathbf{k})$, we have (cf. Figure 4):
   $E \supset \{(a,k) \xrightarrow{h(\mathbf{k})} (\zeta,k) \mid k \in \mathcal{P}_\iota\}$
2. Otherwise, if the right hand side of the implication is $a[i] \geq h(\mathbf{k})$, we have:
   $E \supset \{(\zeta,k) \xrightarrow{-h(\mathbf{k})} (a,k) \mid k \in \mathcal{P}_\iota\}$

Nothing else is in E.

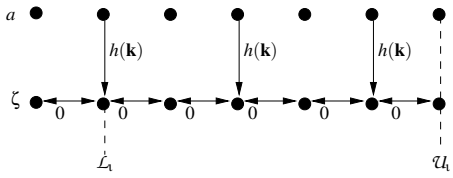


Fig. 4. Constraint graph for $\forall i \,.\, l \leq i \leq u \,\wedge\, i \equiv_2 0 \rightarrow a[i] \leq h(\mathbf{k})$

Formulae of type (F2) Let φ be the formula:

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

where $0 \leq s < t$, $p \in \mathbb{N}$, and $q \in \mathbb{Z}$. Let $\mathcal{P}_\iota = \{n \in \mathbb{Z} \mid \models \phi_\iota[n/i]\}$.

The set of edges E is defined by the following case split:

1. If the right hand side of the implication is $a[i] - b[i+p] \leq q$, we have (cf. Fig. 5): $E \supset \{(a,k) \xrightarrow{q} (b,k+p) \mid k \in \mathcal{P}_\iota\}$

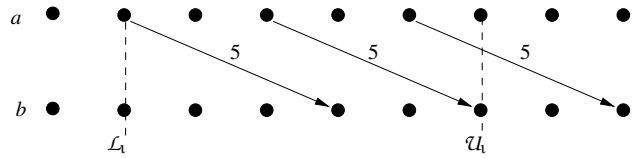


Fig. 5. Constraint graph for $\forall i . l \leq i \leq u \wedge i \equiv_2 0 \rightarrow a[i] - b[i+3] \leq 5$

2. If the right hand side of the implication is $a[i] - b[i+p] \geq q$, then $E \supset \{(b,k+p) \xrightarrow{-q} (a,k) \mid k \in \mathcal{P}_\iota\}$

Nothing else is in E.

C.2 Counter Automata

Formulae of type (F1) Let φ be

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

where $0 \leq t < s$. Let $\mathbb{L} = \{f_1, \ldots, f_K\}$ and $\mathbb{U} = \{g_1, \ldots, g_L\}$. Then we define $A_\varphi = A_1 \otimes A_2$, where A_1 and A_2 are instantiated according to Table 2(a).

(a)

$\sim$	A_1	A_2
$\leq$	$V(a,\zeta,h,s,t,\mathbb{L},\mathbb{U})$	$H(\zeta,bi,\top)$
$\geq$	$V(a,\zeta,-h,s,t,\mathbb{L},\mathbb{U})$	$H(\zeta,bi,\top)$

(b)

p	$\sim$	A_φ
0	$\leq$	$V(a,b,q,s,t,\mathbb{L},\mathbb{U})$
0	$\geq$	$V(b,a,-q,s,t,\mathbb{L},\mathbb{U})$
>0	$\leq$	$D(a,b,p,q,s,t,\mathbb{L},\mathbb{U},\texttt{right})$
>0	$\geq$	$D(b,a,p,-q,s,t,\mathbb{L},\mathbb{U},\texttt{left})$
<0	$\leq$	$D(a,b,-p,q,s,t+p,\mathbb{L}+p,\mathbb{U}+p,\texttt{left})$
<0	$\geq$	$D(b,a,-p,-q,s,t+p,\mathbb{L}+p,\mathbb{U}+p,\texttt{right})$

Table 2. The instantiation table for (F1) and (F2) formulae

Formulae of type (F2) Let φ be the formula :

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

where $0 \leq s < t$. As previously, we denote $\mathbb{L} = \{f_1,\ldots,f_K\}$ and $\mathbb{U} = \{g_1,\ldots,g_L\}$. The instantiation of A_φ is done according to the value of p and $\sim$ as described in Table 2(b).[9] Given a set of integers S and an integer p we use the notation $S+p$ for $\{s+p \mid s \in S\}$.

Counter Automata for Array-Bound Constraints. The FBCA A_θ for a Presburger constraint θ on array-bound variables is $A_\theta = \langle \mathbf{x_k}, Q, L, R, \rightarrow\rangle$, where $\mathbf{x_k}$ is the set $\{x_k \mid k \in FV(\theta)\}$, $Q = \{q_L, q_R\}$, $L = \{q_L\}$, $R = \{q_R\}$, and $\rightarrow = \{q_L \xrightarrow{\top} q_L, q_L \xrightarrow{\theta(\mathbf{x_k})} q_R, q_R \xrightarrow{\top} q_R\}$, and $\theta(\mathbf{x_k})$ denotes the formula θ in which each occurrence of an array-bound variable $k \in FV(\theta)$ is replaced by its corresponding parameter x_k.

D Proofs

Proof of Lemma 1 Let $A = \langle \mathbf{x}, Q, L, R, \rightarrow\rangle$. "$\Rightarrow$" r is left-accepting iff there exists an infinite decreasing sequence $\ldots i_3 < i_2 < i_1 < 0$ of positions in r, visiting a control state from L. This implies that $i_3 - 1 < i_2 - 1 < i_1 - 1 < 0$ visits the same control state, hence $r \circ s$ is left-accepting. r is right-accepting iff there exists an infinite increasing sequence $0 < j_1 < j_2 < j_3 < \ldots$ of positions in r, which visits a control state from R. But this implies that $0 < j_2 - 1 < j_3 - 1 < \ldots$ visits the same state from R, hence $r \circ s$ is right-accepting. "$\Leftarrow$" This direction follows a similar argument. □

[9] Note that in the last two lines of Table 2(b), 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.

Proof of Lemma 2 The proof uses the results of Appendix A, namely Lemmas 7 and 8. Let $A = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ be a FBCA. W.l.o.g. we can assume that any control state $q \in L \cup R$ belongs to exactly one elementary cycle. For, if q does not belong to a cycle, it cannot occur infinitely often on a run. Moreover, if q belongs to two or more elementary cycles, then A is not flat, in contradiction with the definition of FBCA. Let $l \in L$ and $r \in R$ be fixed for the rest of this proof. We construct a Presburger formula $\Phi_{l,r}$ which is satisfiable if and only if there exists a bi-infinite run that visits l infinitely often on the left and r infinitely often on the right.

Let γ be the elementary cycle to which l belongs, and $\overleftarrow{\gamma}$ be the cycle obtained by reversing each transition $q \xrightarrow{\varphi} q'$ into $q' \xrightarrow{\varphi'} q$, where φ' is obtained from φ by interchanging the occurrences of the counters in $\mathbf{x}$ with $\mathbf{x}'$, and vice versa. Let $I_{l,\overleftarrow{\gamma}}(\mathbf{x})$ be the Presburger formula defining the set of valuations ν for which there exists an infinite computation along $\overleftarrow{\gamma}$ starting in (l, ν).

Let δ be the elementary cycle to which r belongs, and $I_{r,\delta}(\mathbf{x})$ be the Presburger formula defining the set of valuations ν for which there exists an infinite computation along δ starting in (r, ν). The formula encoding the existence of a bi-infinite run that visits l infinitely often on the left and r infinitely often on the right, is the following:

$$\Phi_{l,r} : \exists \mathbf{x} \exists \mathbf{x}' \, . \, I_{l,\overleftarrow{\gamma}}(\mathbf{x}) \wedge R_{l,r}(\mathbf{x}, \mathbf{x}') \wedge I_{r,\delta}(\mathbf{x}')$$

The proof that $\Phi_{l,r}$ is satisfiable if and only if $\mathcal{R}(A) \neq \emptyset$ comes as an immediate consequence of the meaning of the $I_{l,\overleftarrow{\gamma}}$, $R_{l,r}$ and $I_{r,\delta}$ formulae. □

Proof of Proposition 1 The direction $\mathcal{R}(A) \subseteq \mathcal{R}(A^c)$ is trivial, since $L \subseteq L^c$ and $R \subseteq R^c$. To prove the fact that $\mathcal{R}(A) \supseteq \mathcal{R}(A^c)$, let r be an accepting run of A^c. Then there exists a state $q \in L^c$ that repeats infinitely often on the left in r. There are two situations: either $q \in L$, in which case r is directly left-accepting for A, or there exists a state $q' \in L$ which belongs to the same elementary cycle as q in A. By the flatness of A, this means that q' will be visited infinitely often on the left as well. Analogously, one proves that r is a right-accepting run of A. □

Proof of Lemma 3 The proof for closure under union is trivial. We will give the proof for closure under intersection in the following.

Let $A_i = \langle \mathbf{x_i}, Q_i, L_i, R_i, \rightarrow_i \rangle$, $i = 1, 2$ be two FBCA, and $A = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$ be their product, i.e. $A = A_1 \otimes A_2$.

1. We first prove that A belongs to the class FBCA. For this we need to show that each control state of A belongs to at most one elementary cycle. For an arbitrary state $(q, q') \in Q_1 \times Q_2$, let $pr_1((q, q')) = q$, $pr_2((q, q')) = q'$ and for an arbitrary cycle γ in A, let $pr_i(\gamma)$ denote the corresponding cycles in A_i, obtained by projection of the i-th control state, $i = 1, 2$. Suppose that there is a control state $(q, q') \in Q_1 \times Q_2$ that belongs to (at least) two different elementary cycles, γ and δ. Then q belongs to $pr_1(\gamma)$ and $pr_1(\delta)$ in A_1, and q' belongs to $pr_2(\gamma)$ and $pr_2(\delta)$ in A_2. Since, by the hypothesis A_1

and A_2 are flat, then $pr_i(\gamma)$ and $pr_i(\delta)$ must be (possibly trivial) unfoldings of the same elementary cycle ε_i in A_i, for $i = 1,2$, respectively. In other words, $pr_i(\gamma) = k_i \cdot \varepsilon_i$ and $pr_i(\delta) = l_i \cdot \varepsilon_i$, for $i = 1,2$ and $k_i, l_i \in \mathbb{N}$.

Let m be the least common multiple of $|\varepsilon_1|$ and $|\varepsilon_2|$, and $n_i = \frac{m}{|\varepsilon_i|}$, for $i = 1,2$. Let α be the cycle in A obtained by the composition of the two cycles obtained by iterating ε_1 n_1 times, and ε_2 n_2 times, respectively, i.e. $pr_i(\alpha) = n_i \cdot \varepsilon_i$, $i = 1,2$. Since ε_i are elementary cycles of A_i, it follows that α is the smallest cycle of A with the property that $pr_i(\alpha)$ is an unfolding of ε_i. Hence γ and δ, must both be either α or unfoldings of α, contradicting the assumption that they were different elementary cycles of A.

To prove that the elementary cycles of A are labelled with (parametric) DBM formulae only, notice that any cycle of A is a composition of two (unfoldings of) cycles in A_1 and A_2. Since both component cycles are labelled with DBM formulae, and the label of the transitions of A is the conjunction of the labels of the transitions in A_1, A_2, it follows that the resulting cycle is labelled with DBM formulae as well.

2. Second, we prove that $\mathcal{V}(A) = \mathcal{V}(A_1)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cap \mathcal{V}(A_2)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$.

Let $s \in \mathcal{V}(A_1)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cap \mathcal{V}(A_2)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$ be a bi-infinite sequence of counter valuations. From the definition of $\mathcal{V}(.)$, there exist $r_1 \in \mathcal{R}(A_1)$ and $r_2 \in \mathcal{R}(A_2)$ such that $val(r_1) = s\downarrow_{\mathbf{x_1}}$ and $val(r_2) = s\downarrow_{\mathbf{x_2}}$.

Let $i \in \mathbb{Z}$ be an arbitrary position, and $r_1(i) = (q_1, \nu_1)$, $r_1(i+1) = (q_1', \nu_1')$, $r_2(i) = (q_2, \nu_2)$, $r_2'(i+1) = (q_2', \nu_2')$ be successive configurations of r_1 and r_2, respectively, where $\nu_1, \nu_1' : \mathbf{x_1} \to \mathbb{Z}$ and $\nu_2, \nu_2' : \mathbf{x_2} \to \mathbb{Z}$ are valuations of $\mathbf{x_1}$, $\mathbf{x_2}$. Then there exist transition rules $q_1 \xrightarrow{\varphi_1(\mathbf{x_1}, \mathbf{x_1'})} q_1'$ in A_1, and $q_2 \xrightarrow{\varphi_2(\mathbf{x_2}, \mathbf{x_2'})} q_2'$ in A_2, such that $\varphi_1(\nu_1(\mathbf{x_1}), \nu_1'(\mathbf{x_1'}))$ and $\varphi_2(\nu_2(\mathbf{x_2}), \nu_2'(\mathbf{x_2'}))$ are both valid. Hence, by construction of A, there exists a transition rule $(q_1, q_2) \xrightarrow{\varphi_1 \wedge \varphi_2} (q_1', q_2')$, such that $\varphi_1 \wedge \varphi_2$ is satisfied by $(\nu_1 \cup \nu_1')\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cap (\nu_2 \cup \nu_2')\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$. In this way, one can build a bi-infinite run r of A, such that $val(r) = s$. It remains to be proven that this run is an accepting run of A.

Since r_i is an accepting run of A_i, then by Proposition 1, it is also an accepting run of A_i^c, for $i = 1,2$. By the flatness of A_1, and, implicitly of A_1^c, there exist a sequence σ_1 of states from L_1^c that repeats infinitely often to the left of r_1, i.e. there exists a position $k_1 \in \mathbb{Z}$, such that the restriction of r_1 to $(-\infty, k_1]$ is of the form $\ldots\sigma_1\sigma_1$. Analogously, there exists a sequence σ_2 of states from L_2^c, and a position $k_2 \in \mathbb{Z}$ such that the restriction of r_2 to $(-\infty, k_2]$ is of the form $\ldots\sigma_2\sigma_2$. Then, the restriction of r to $(-\infty, \min(k_1, k_2)]$ is of the form $\ldots\sigma\sigma$, where σ is a sequence of pairs $(q, q') \in L_1^c \times L_2^c$. Hence there exists such a pair repeating infinitely often to the left in r, i.e. r is left-accepting. Analogously, one proves that r is right-accepting. We have proved that $\mathcal{V}(A) \supseteq \mathcal{V}(A_1)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cap \mathcal{V}(A_2)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$. The direction $\mathcal{V}(A) \subseteq \mathcal{V}(A_1)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cap \mathcal{V}(A_2)\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$ is proved using a similar argument. □

Proof of Lemma 4 This can be proven by a reduction from the halting problem for 2-counter automata [13]. A 2-counter machine with non-negative counters c_1, c_2 is a sequential program:

$$0 : \texttt{ins}_0; 1 : \texttt{ins}_1; \cdots; n : \texttt{ins}_n;$$

where $\texttt{ins}_n$ is a `halt` instruction and $\texttt{ins}_i$ with $i = 0, 1, \cdots, n$ are instructions of the following two types, for $0 \leq k, k_1, k_2 \leq n$, and $1 \leq j \leq 2$:

1. $c_j = c_j + 1; \texttt{goto } k$;
2. $\texttt{if } c_j = 0 \text{ then goto } k_1 \text{ else } (c_j = c_j - 1; \text{ goto } k_2)$;

We give a formula φ such that the machine halts iff the formula is satisfiable. φ uses three arrays a_1, a_2 and a_3. a_1 (resp. a_2) contains values of counter 1 (resp. 2) and a_3 contains the control location. Each instruction $k : ins_k$ is translated into a formula $\varphi_k(i)$ having a parameter i. We give the translation for instructions concerning counter c_1. Instructions concerning counter c_2 are encoded in a similar way. Instructions of the form $k : c_1 = c_1 + 1; \texttt{goto } k'$ are translated into:

$$\varphi_k(i) : a_3[i] = k \wedge a_1[i+1] = a_1[i] + 1 \wedge a_2[i+1] = a_2[i] \wedge a_3[i+1] = k'$$

Instructions of the form $k : \texttt{if } c_j = 0 \text{ then goto } k_1 \text{ else } (c_j = c_j - 1; \text{ goto } k_2)$ are translated into:

$$\begin{aligned}\varphi_k(i) : (a_3[i] = k \wedge a_1[i] = 0 \wedge a_1[i+1] = a_1[i] \wedge a_2[i+1] = a_2[i] \wedge a_3[i+1] = k_1)\\ \vee (a_3[i] = k \wedge a_1[i] > 0 \wedge a_1[i+1] = a_1[i] - 1 \wedge a_2[i+1] = a_2[i] \wedge a_3[i+1] = k_2)\end{aligned}$$

Now the formula φ is given as

$$\exists a_1, a_2, a_3 \exists m. \forall i. ((0 \leq i \leq m-1) \rightarrow (a_3[0] = 0 \wedge \bigvee_{j=0}^{n-1} \varphi_j(i) \wedge a_3[m] = n))$$

The models of the formula are exactly the halting runs of the counter machine in m steps. $a_1[i]$ (resp. $a_2[i]$) is the value of counter c_1 (resp. c_2) after i steps and $a_3[i]$ is the corresponding control location. $a_3[0] = 0$ and $a_3[m] = n$ make sure that the machine starts at the initial control location 0 and goes to the halting location n and $\bigvee_{j=0}^{n-1} \varphi_j(i)$ insures that counter values and control locations stored in two consecutive positions (i and $i+1$) in the arrays a_1, a_2 and a_3 correspond to values in a run of the machine. Then it is clear, that the machine halts iff φ is satisfiable.

Note that one can easily give a formula using just one array. This is done by interleaving the three arrays and using the modulo constraints to access the counter values and the control locations. □

Proof of Lemma 5 We show how a formula, written in the syntax of Figure 1, can be transformed into an equivalent formula of the form (NF), by applying the steps below:

1. Put the left-hand sides of the subformulae $\forall \mathbf{i} \,.\, \varphi(\mathbf{i}) \rightarrow \psi(\mathbf{i})$ into disjunctive normal form, and then split both the left-hand and right-hand sides by applying exhaustively the following equivalence preserving transformations:

$$\begin{aligned}\forall i \,.\, \varphi_1 \vee \varphi_2 \rightarrow \psi &\iff \forall i \,.\, \varphi_1 \rightarrow \psi \;\wedge\; \forall i \,.\, \varphi_2 \rightarrow \psi\\ \forall i \,.\, \varphi \rightarrow \psi_1 \wedge \psi_2 &\iff \forall i \,.\, \varphi \rightarrow \psi_1 \;\wedge\; \forall i \,.\, \varphi \rightarrow \psi_2\end{aligned}$$

   The resulting formula will have only conjunctions of atomic formulae on the left-hand side of the implications and only atomic formulae on the right hand side of the implications.

2. Put the entire formula into disjunctive normal form, treating the implications $\forall \mathbf{i} \,.\, \varphi(\mathbf{i}) \rightarrow \psi$ as atomic propositions, and distribute the existential prefix to each disjunctive clause.
3. Eliminate negated implications using the equivalence $\neg\big(\forall \mathbf{i} \,.\, \varphi \rightarrow \psi\big) \iff \exists \mathbf{k} \,.\, \varphi \wedge (\top \rightarrow \neg\psi[\mathbf{k}/\mathbf{i}])$. Notice that, because of the previous step, ψ is an atomic DBM formula involving array terms, hence $\neg\psi$ can be written equivalently without negation. We move the existential quantifier to the prefix of existential quantifiers of the formula, renaming the index variables $\mathbf{i}$ by some fresh array-bound variables $\mathbf{k}$. We make φ a part of θ_p. The newly introduced implication is not preceded by a universal quantifier (as expected by the normal form we use), but this will be taken care of by the next step.
4. For each implication of the form $\forall \mathbf{i} \,.\, \varphi(\mathbf{k},\mathbf{i}) \rightarrow \psi(\mathbf{a},\mathbf{k},\mathbf{i})$, such that ψ contains an array term $a[f(\mathbf{k})]$ where $f(\mathbf{k})$ is a linear combination of array-bound variables, introduce a fresh universally quantified index variable j, and rewrite the whole implication as $\forall \mathbf{i} \cup \{j\} \,.\, \varphi \wedge j = f(\mathbf{k}) \rightarrow \psi[j/f(\mathbf{k})]$. This step ensures that array terms are indexed only by universally quantified index variables.
5. Normalise all DBM subformulae of the premises φ of the array subformulae $\forall \mathbf{i} \,.\, \varphi \rightarrow \psi$. This step computes also the transitive closure of the DBMs, making explicit all dependencies between indices. For each pair of constraints $i - j \leq n$ and $j - i \leq -m$ occurring in a conjunction within the premise of an implication of the form $\forall \mathbf{i} \,.\, \varphi \rightarrow \psi$, either it is the case that $n - m < 0$, in which case replace the whole implication by *true*, or else $n - m \geq 0$, in which case replace both constraints by $\bigvee_{l \in [m,n]} i - j = l$[10] and eliminate i from the implication subformula, by replacing each occurrence of i by $j + l$. This step ensures that no constraints of the form $m \leq i - j \leq n$ are left within the formula.
6. Rename the universally quantified index variables such that each array constraint of the form (i) $a[i+n] \sim g(\mathbf{k})$, (ii) $a[i+n] - b[i+m] \sim p$, or (iii) $a[i+n] - b[j+m] \sim p$, $n, m, p \in \mathbb{Z}$, uses index variables that are distinct from the other. In the following, we distinguish three cases:
   (i) For subformulae of the form $a[i+n] \sim g(\mathbf{k})$, replace i with $i - n$ throughout the formula. In particular, the array terms $a[i+n]$ are substituted with $a[i]$.
   (ii) For subformulae of the form $a[i+n] - b[i+m] \sim p$, suppose that $n \leq m$, the other case being symmetric. We replace i with $i - n$ throughout the formula. In particular, the array terms $a[i+n]$ are substituted with $a[i]$, and $a[i+m]$ with $a[i+m-n]$, respectively.
   (iii) For subformulae of the form $a[i+n] - b[j+m] \sim p$, replace i with $i - n$ and j with $j - m$. In particular, the array terms $a[i+n]$ and $b[j+m]$ are substituted with $a[i]$ and $b[j]$, respectively.

   This step ensures that the only constraints involving array terms are of the form $a[i] \sim g$, $a[i] - b[i+n] \sim m$ and $a[i] - b[j] \sim m$, where g is a linear polynomial in bound variables, $\sim \in \{\leq, \geq\}$, $n \in \mathbb{N}$ and $m \in \mathbb{Z}$.
7. Normalise the atomic propositions in all the premises of the implications $\forall \mathbf{i} \,.\, \varphi \rightarrow \psi$ by applying the following substitutions:
   (a) $f \sim i + n$ with $f - n \sim i$ for $\sim \in \{\leq, \geq\}$,

[10] By $\bigvee_{l \in [a,b]} \phi(l)$ we denote the disjunction $\varphi(a) \vee \varphi(a+1) \vee \ldots \vee \varphi(b)$.

(b) $i-j+n \leq p$ with $i-j \leq p-n$,
(c) $i+n \equiv_s t$ with $i \equiv_s t'$, where $0 \leq t' < t$ and $t' \equiv_s t+n$.

It can be easily checked that the formula obtained after applying the normalisation steps is in the form (NF), and that is equivalent to the initial formula, since every transformation preserves logical equivalence.

Proof of Lemma 6 We carry out the proof separately for φ being of type (F1)-(F3).

(F1) $\varphi : \forall i \,.\, \bigwedge_{k=1}^{K} f_k \leq i \wedge \bigwedge_{l=1}^{L} i \leq g_l \wedge i \equiv_s t \rightarrow a[i] \sim h(\mathbf{k})$ where $0 \leq t < s$

"$\Rightarrow$" By the construction of $G_{\iota,\varphi} = (V,E)$, we have $V = \{a,\zeta\} \times \mathbb{Z}$. Define $Lab : V \rightarrow \mathbb{Z}$ as $Lab((a,n)) = \mu(a,n)$ and $Lab((\zeta,n)) = 0$ for all $n \in \mathbb{Z}$. To show that Lab is consistent, let $\sim$ be $\leq$, the other case being symmetric. Let us consider any edge from E. For edges linking nodes from $\zeta \times \mathbb{Z}$, we have trivially $Lab((\zeta,n)) - Lab((\zeta,n+1)) \leq 0$ and $Lab((\zeta,n+1)) - Lab((\zeta,n)) \leq 0$. The only other edges in $G_{\iota,\varphi}$ are of the form $(a,n) \xrightarrow{h(\mathbf{k})} (\zeta,n)$ with $n \in \mathcal{P}_\iota$ where $\mathcal{P}_\iota$ is the set given in the construction of $G_{\iota,\varphi}$. Any $n \in \mathcal{P}_\iota$ satisfies the precondition of φ. Since (ι,μ) is a model of φ, we have that $\mu(a,n) - 0 \leq h(\mathbf{k})$, which implies $Lab((a,n)) - Lab((\zeta,n)) \leq h(\mathbf{k})$.

"$\Leftarrow$" This direction follows from a similar argument.

(F2) $\varphi : \forall i \,.\, \bigwedge_{k=1}^{K} f_k \leq i \wedge \bigwedge_{l=1}^{L} i \leq g_l \wedge i \equiv_s t \rightarrow a[i] - b[i+p] \sim q$ where $0 \leq s < t$, $p \in \mathbb{N}$, $q \in \mathbb{Z}$.

"$\Rightarrow$" By the construction of $G_{\iota,\varphi} = (V,E)$, we have $V = \{a,b,\zeta\} \times \mathbb{Z}$. Define $Lab : V \rightarrow \mathbb{Z}$ as $Lab((a,n)) = \mu(a,n)$, $Lab((b,n)) = \mu(b,n)$, and $Lab((\zeta,n)) = 0$ for all $n \in \mathbb{Z}$. To show that Lab is consistent, let $\sim$ be $\leq$, the other case being symmetric. Let us consider any edge from E. For edges linking nodes from $\zeta \times \mathbb{Z}$, we have trivially $Lab((\zeta,n)) - Lab((\zeta,n+1)) \leq 0$ and $Lab((\zeta,n+1)) - Lab((\zeta,n)) \leq 0$. The only other edges in $G_{\iota,\varphi}$ are of the form $(a,n) \xrightarrow{q} (b,n+p)$ with $n \in \mathcal{P}_\iota$ where $\mathcal{P}_\iota$ is the set given in the construction of $G_{\iota,\varphi}$. Since (ι,μ) is a model of φ, then for all $n \in \mathcal{P}_\iota$, we have $\mu(a,n) - \mu(b,n+p) \leq q$, which implies $Lab((a,n)) - Lab((b,n+p)) \leq q$.

"$\Leftarrow$" This direction follows from a similar argument.

(F3) $\varphi : \forall i,j \,.\, \bigwedge_{k=1}^{K_1} f_k^1 \leq i \wedge \bigwedge_{l=1}^{L_1} i \leq g_l^1 \wedge \bigwedge_{k=1}^{K_2} f_k^2 \leq j \wedge \bigwedge_{l=1}^{L_2} j \leq g_l^2 \wedge i - j \leq p \wedge i \equiv_s t \wedge j \equiv_u v \rightarrow a[i] - b[j] \sim q$ where $0 \leq s < t$, $0 \leq u < v$, $p \in \mathbb{Z}^\infty$, and $q \in \mathbb{Z}$.

Let us assume first that $p < \infty$ and that $\sim$ is $\leq$, the other cases being very similar. Let the sets $\mathcal{P}_\iota^1$ and $\mathcal{P}_\iota^2$ be defined as in the construction of the constraint graph $G_{\iota,\varphi}$.

"$\Rightarrow$" By the construction of $G_{\iota,\varphi} = (V,E)$, we have $V = \{a,b,\zeta,t_\varphi\} \times \mathbb{Z}$. First of all, we define $Lab : V \rightarrow \mathbb{Z}$ as $Lab((a,n)) = \mu(a,n)$, $Lab((b,n)) = \mu(b,n)$, and $Lab((\zeta,n)) = 0$ for all $n \in \mathbb{Z}$. It remains to define Lab for $t_\varphi \times \mathbb{Z}$.

Let us consider first the case where $T_\leq = \emptyset$. Then, there do not exist $k \in \mathcal{P}_\iota^1$ and $l \in \mathcal{P}_\iota^2$ such that $k - l \leq p$. This allows us to define $Lab((t_\varphi,n)) = \mu(a,n+p) - q$ for

$n+p \in \mathcal{P}^1_\iota$, $Lab((t_\varphi,n)) = \mu(b,n)$ for $n \in \mathcal{P}^2_\iota$ and $Lab((t_\varphi,n)) = 0$ for all other $n \in \mathbb{Z}$. As there is no n such that $n+p \in \mathcal{P}^1_\iota$ and $n \in \mathcal{P}^2_\iota$ and as there are no arcs linking nodes of $t_\varphi \times \mathbb{Z}$, it can be easily checked that the labelling is consistent.

Second, we consider the case where $T_\leq \neq \emptyset$. In such a case, there exist $k' \in \mathcal{P}^1_\iota$ and $l \in \mathcal{P}^2_\iota$ such that $k'-l \leq p$. Thus, $\mathcal{P}^2_\iota$ is not empty, and by definition it is finite, hence it has a maximum element.

Then, we define $Lab((t_\varphi,n))$ as follows: For $n \leq max(\mathcal{P}^2_\iota)$, $Lab((t_\varphi,n)) = min\{\mu(b,i) \mid i \in \mathcal{P}^2_\iota \text{ and } i \geq n\}$. For $n > max(\mathcal{P}^2_\iota)$, we define the labelling inductively as follows: If $n+p \in \mathcal{P}^1_\iota$, then $Lab((t_\varphi,n)) = max(Lab((t_\varphi,n-1)), \mu(a,n+p)-q)$, otherwise $Lab((t_\varphi,n)) = Lab((t_\varphi,n-1))$. It remains to show that $min\{\mu(b,i) \mid i \in \mathcal{P}^2_\iota \text{ and } i \geq n\}$ exists and that Lab is consistent.

Since μ is a model, we have $\mu(a,k') - \mu(b,j) \leq q$ for all $j \in \mathcal{P}^2_\iota$ with $k'-j \leq p$. This implies that the set $\{\mu(b,i) \mid i \in \mathcal{P}^2_\iota \text{ and } i \geq n\}$ is bounded from below. Therefore $min\{\mu(b,i) \mid i \in \mathcal{P}^2_\iota \text{ and } i \geq n\}$ exists.

To show that Lab is consistent, we consider all edges of $G_{\iota,\varphi}$.

For edges linking nodes from $\zeta \times \mathbb{Z}$, we have trivially $Lab((\zeta,n)) - Lab((\zeta,n+1)) \leq 0$ and $Lab((\zeta,n+1)) - Lab((\zeta,n)) \leq 0$.

For edges of $T_\leq$, we have by definition of the labelling of $t_\varphi \times \mathbb{Z}$ that $Lab((t_\varphi,n)) \leq Lab((t_\varphi,n+1))$ for all $n \in \mathbb{Z}$. Indeed, for $n \leq max(\mathcal{P}^2_\iota)$, we set $Lab(t_\varphi,n) = \min\{\mu(b,i) \mid i \in \mathcal{P}^2_\iota \text{ and } i \geq n\}$. Notice that, for $n_1 \leq n_2$ we have $Lab(t_\varphi,n_1) \leq Lab(t_\varphi,n_2)$. For $n > max(\mathcal{P}^2_\iota)$, we set $Lab(t_\varphi,n) = Lab(t_\varphi,n-1)$.

For edges in $\{(a,k) \xrightarrow{q} (t_\varphi,k-p) \mid k \in \mathcal{P}^1_\iota\}$, we consider two cases. If $k-p > max(\mathcal{P}^2_\iota)$, then by definition of the labelling, we have that $Lab((a,k)) = \mu(a,k)$ and $Lab((t_\varphi,k-p)) = max(Lab((t_\varphi,n-1)), \mu(a,k)-q)$. Therefore, $Lab((a,k)) - Lab((t_\varphi,k-p)) \leq q$. If $k-p \leq max(\mathcal{P}^2_\iota)$, then by definition of the labelling, we have that $Lab((a,k)) = \mu(a,k)$ and $Lab((t_\varphi,k-p)) = min\{\mu(b,i) \mid i \in \mathcal{P}^2_\iota \text{ and } i \geq k-p\}$. Let $m \in \mathcal{P}^2_\iota$ be such that $\mu(b,m) = Lab((t_\varphi,k-p))$. Since μ is a model, we have $\mu(a,k) - \mu(b,m) \leq q$. This implies $Lab((a,k)) - Lab((t_\varphi,k-p)) \leq q$.

Finally, for edges in $\{(t_\varphi,k) \xrightarrow{0} (b,k) \mid k \in \mathcal{P}^2_\iota\}$, we have by definition of the labelling that $Lab((t_\varphi,k)) - Lab((b,k)) \leq 0$.

"$\Leftarrow$" Let Lab be a consistent labelling of $G_{\iota,\varphi}$ and μ a valuation such that $\mu(a,i) = Lab((a,i))$ for all $a \in \mathbf{a}$ and $i \in \mathbb{Z}$. Let i,j such that $\bigwedge_{k=1}^{K_1} f^1_k \leq i \wedge \bigwedge_{l=1}^{L_1} i \leq g^1_l \wedge \bigwedge_{k=1}^{K_2} f^2_k \leq j \wedge \bigwedge_{l=1}^{L_2} j \leq g^2_l \wedge i-j \leq p \wedge i \equiv_s t \wedge j \equiv_u v$. By the construction of $G_{\iota,\varphi}$, there are edges $(a,i) \xrightarrow{q} (t_\varphi,i-p)$, $(t_\varphi,i) \xrightarrow{0} (t_\varphi,i+1)$,..., $(t_\varphi,j-1) \xrightarrow{0} (t_\varphi,j)$ and $(t_\varphi,j) \xrightarrow{0} (b,j)$. By the fact that Lab is consistent, we have $Lab((a,i)) - Lab((b,j)) \leq q$ which implies that $\mu(a,i) - \mu(b,j) \leq q$. □

D.1 Proof of Theorem 1

To proof the main theorem we give first several lemmas. The following two lemmas relate the control states visited by an accepting run of A_φ, with its positions.

Lemma 9. *Let $A = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$, where $Q = \{q_L, q_R\} \cup \{q_i \mid 0 \leq i < s\} \cup \{q_i^j \mid 0 \leq j < s,\ j+1 \leq i < j+p\}$ be an instance of the diagonal template $D(a,b,p,q,s,t,\mathbb{L},\mathbb{U},dir)$, and r_0 be any normalised accepting run of A. Supposing that $\mathbb{L} \cup \mathbb{U} \neq \emptyset$, for all $k \in \mathbb{Z}$, we have that, if either $r_0(k) = (q_i^j, \nu)$ or $r_0(k) = (q_i, \nu)$ for some valuation ν of the counters in A, then $\nu(k) \equiv_s i$.*

Proof. Follows easily from (1) the fact that to enter and to leave the states $\{q_i \mid 0 \leq i < s\}$ a guard checking the modulo constraint has to be satisfied and (2) the fact that if $\mathbb{L} \cup \mathbb{U} \neq \emptyset$, then an accepting run has to either enter or leave the states $\{q_i \mid 0 \leq i < s\}$ due to the presence of guards in the transitions. □

Lemma 10. *Let $A = \langle \mathbf{x}, Q, L, R, \rightarrow \rangle$, where $Q = \{q_L, q_R\} \cup \{q_i \mid 0 \leq i < s\}$ be an instance of the vertical template $V(a,b,p,q,s,t,\mathbb{L},\mathbb{U})$, and r_0 be any normalised accepting run of A. Supposing that $\mathbb{L} \cup \mathbb{U} \neq \emptyset$ for all $k \in \mathbb{Z}$, we have that, if $r_0(k) = (q_i, \nu)$ for some valuation ν, then $\nu(k) \equiv_s i$.*

Proof. Like the proof of Lemma 9. □

The following lemma is the basis of theorem 1.

Lemma 11. *Let $\varphi(\mathbf{k}, \mathbf{a})$ be a formula of the form (F1)-(F3) and A_φ the corresponding automaton, as defined in Section 4.4. Then, $M_\varphi(\mathcal{V}(A_\varphi)) = [\![\varphi]\!]$.*

Proof. We only give the proof for the most difficult case, i.e. formulae of the form (F3). For the other formulae, it is similar. Let us have a formula $\varphi : \forall i, j\ .\ \bigwedge_{k=1}^{K_1} f_k^1 \leq i \wedge \bigwedge_{l=1}^{L_1} i \leq g_l^1 \wedge \bigwedge_{k=1}^{K_2} f_k^2 \leq j \wedge \bigwedge_{l=1}^{L_2} j \leq g_l^2 \wedge i - j \leq p \wedge i \equiv_s t \wedge j \equiv_u v \rightarrow a[i] - b[j] \sim q$ 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. Let $\phi = \exists i, j\ .\ \bigwedge_{k=1}^{K_1} f_k^1 \leq i \wedge \bigwedge_{l=1}^{L_1} i \leq g_l^1 \wedge \bigwedge_{k=1}^{K_2} f_k^2 \leq j \wedge \bigwedge_{l=1}^{L_2} j \leq g_l^2 \wedge i - j \leq p \wedge i \equiv_s t \wedge j \equiv_u v$. We give the proof for $p > 0$ and $\sim = \leq$. The other cases are very similar. Let A_φ be the automaton corresponding to φ. We have $A_\varphi = A_1 \otimes A_2 \otimes A_3$ where A_1 is an instance of $D(a, t_\varphi, p, q, s, t-p, \mathbb{L}_1 - p, \mathbb{U}_1 - p, \texttt{left})$, A_2 is an instance of $H(t_\varphi, \texttt{right}, \phi)$, and A_3 is an instance of $V(t_\varphi, b, 0, u, v, \mathbb{L}_2, \mathbb{U}_2)$. We suppose that $\mathbb{L}_1 \cup \mathbb{U}_1 \neq \emptyset$ and $\mathbb{L}_2 \cup \mathbb{U}_2 \neq \emptyset$. The other cases are treated in a similar way.

"$\subseteq$" We first show that $M_\varphi(\mathcal{V}(A_\varphi)) \subseteq [\![\varphi]\!]$. Let r be an accepting run of A_φ and r_0 be the normalised run corresponding to r. Let $\iota_r : \mathbf{k} \rightarrow \mathbb{Z}$ and $\mu_r : \{a, b\} \times \mathbb{Z} \rightarrow \mathbb{Z}$ be the valuations of the free variables of φ corresponding to the run r_0. Let $G_{\iota_r, \varphi} = (V, E)$ be the constraint graph corresponding to φ for the valuation of the bound variables ι_r. We show below that starting from the run r_0, we can define a consistent labelling *Lab* of $G_{\iota_r, \varphi}$. Thanks to Lemma 6 which implies that the labelling *Lab* corresponds to a model, this is enough to prove that $M_\varphi(\mathcal{V}(A_\varphi)) \subseteq [\![\varphi]\!]$.

By construction, the run r_0 of the automaton $A_\varphi = A_1 \otimes A_2 \otimes A_3$ corresponds to runs r_0^i in the automata A_i ($i \in \{1, 2, 3\}$). We have $val(r_0)(x_a) = val(r_0^1)(x_a)$ and $val(r_0)(x_b) = val(r_0^3)(x_b)$ as well as $val(r_0)(x_{t_\varphi}) = val(r_0^1)(x_{t_\varphi}) = val(r_0^2)(x_{t_\varphi}) = val(r_0^3)(x_{t_\varphi})$.

The labelling *Lab* is defined as follows: $Lab((a,i)) = \mu_r(a,i)$, $Lab((b,i)) = \mu_r(b,i)$, and $Lab((t_\varphi, i)) = val(r_0(i))(x_{t_\varphi})$ for all $i \in \mathbb{Z}$. We show in the following that *Lab* is consistent. Let $\mathcal{L}_{i,\iota_r} = \max\{\iota_r(f_k^i) \mid 1 \leq k \leq K_i\}$ and $\mathcal{U}_{i,\iota_r} = \min\{\iota_r(g_l^i) \mid 1 \leq l \leq L_i\}$

for $i = 1,2$. Let $\mathcal{P}_{1,\mathfrak{l}_r} = \{k \mid \mathcal{L}_{1,\mathfrak{l}_r} \leq k \leq \mathcal{U}_{1,\mathfrak{l}_r} \ \wedge \ k \equiv_s t\}$ and $\mathcal{P}_{2,\mathfrak{l}_r} = \{k \mid \mathcal{L}_{2,\mathfrak{l}_r} \leq k \leq \mathcal{U}_{2,\mathfrak{l}_r} \ \wedge \ k \equiv_u v\}$. We have to consider several cases depending on the left and right-accepting states visited by the runs r_0^i. Let $A_1 = \langle \mathbf{x}_1, Q, L, R, \rightarrow \rangle$, $A_2 = \langle \mathbf{x}_2, Q', L', R', \rightarrow' \rangle$, and $A_3 = \langle \mathbf{x}_3, Q'', L'', R'', \rightarrow'' \rangle$. We have $Q = \{q_L, q_R\} \cup \{q_i \mid 0 \leq i < s\} \cup \{q_i^j \mid 0 \leq j < s, \ j+1 \leq i < j+p\}$, $L = \{q_L\} \cup \{q_i \mid 0 \leq i < s\}$ and $R = \{q_R\} \cup \{q_i \mid 0 \leq i < s\}$, $Q' = \{q'_L, q'_R, p'_L, p'_R\}$ and $Q'' = \{q''_L, q''_R\} \cup \{q''_i \mid 0 \leq i < s\}$.

1. The run r_0^1 is left-accepting using q_L and right-accepting using q_R and goes through the states $\{q_i \mid 0 \leq i < s\}$. We now show that Lab is consistent.
   (a) Let us consider an edge $(a,k) \xrightarrow{q} (t_\varphi, k-p)$ for some $k \in \mathcal{P}_{1,\mathfrak{l}_r}$. We have to show that $Lab((a,k)) - Lab((t_\varphi, k-p)) \leq q$. Due to the structure of the automaton A_1, we have for each $k \in \mathcal{P}_{1,\mathfrak{l}_r}$ that $r_0^1(k-p) = (q_i, \nu)$ for some i. Furthermore, $i \equiv_s t - p$ due to Lemma 9. Then, the construction of the automaton insures that in any run, the p transitions following q_i are such that $val(r_0^1(k))(x_a) - val(r_0^1(k-p))(x_{t_\varphi}) \leq q$. This holds due to the roles of the additional counters $\{x_i \mid 1 \leq i < p\}$ from the definition of D. This implies directly $Lab((a,k)) - Lab((t_\varphi, k-p)) \leq q$.
   (b) Let us consider an edge $(t_\varphi, k) \xrightarrow{0} (b,k)$ for some $k \in \mathcal{P}_{2,\mathfrak{l}_r}$. We have to show that $Lab((t_\varphi,k)) - Lab((b,k)) \leq 0$. If $\mathcal{P}_{2,\mathfrak{l}_r}$ is not empty, then the run r_0^3 must go through the states $\{q''_i \mid 0 \leq i < s\}$ but it cannot stay there all the time. We have for each $k \in \mathcal{P}_{2,\mathfrak{l}_r}$ that $r_0^3(k) = (q_i, \nu)$ with $i \equiv_u v$ due to Lemma 10. Then, the transition following q_i of the automaton ensures that $val(r_0^3(k))(x_{t_\varphi}) - val(r_0^3(k))(x_b) \leq 0$. This implies directly $Lab((t_\varphi,k)) - Lab((b,k)) \leq 0$.
   (c) Let us consider the edges $T_\leq$. The accepting run r_0^2 either goes through q'_L and q'_R or p'_L and p'_R. In the latter case, this means that the guard ϕ is not satisfied. Therefore, by definition, $T_\leq$ is empty. In the former case, we have $x_{t_\varphi} - x'_{t_\varphi} \leq 0$ for each transition of A_2. This gives $val(r_0^2(k))(x_{t_\varphi}) - val(r_0^2(k+1))(x_{t_\varphi}) \leq 0$ for all $k \in \mathbb{Z}$, which implies $Lab((t_\varphi,k)) - Lab((t_\varphi,k+1)) \leq 0$ for all $k \in \mathbb{Z}$.
2. The run r_0^1 is left-accepting using the state q_L and right-accepting using the state q_R and does not go through $\{q_i \mid 0 \leq i < s\}$. In this case, the run goes through the transition $q_L \xrightarrow{\neg(\exists i \,.\, \bigwedge_{l \in \mathbb{L}} i \geq l(\mathbf{x_k}) \wedge \bigwedge_{u \in \mathbb{U}} i \leq u(\mathbf{x_k}) \wedge i \equiv_s t)} q_R$. This means that $\mathcal{P}_{1,\mathfrak{l}_r}$ is empty and diagonal edges are trivially consistent. The other edges are shown to be consistent as in the cases 1(b) and 1(c).
3. The run r_0^1 is left-accepting using a state in $\{q_i \mid 0 \leq i < s\}$ and right-accepting using q_R. In this case, Lemmas 9 and 10 can still be applied in a similar way to the first case to show that the labelling is consistent.
4. The run r_0^1 is left-accepting using the state q_L and right-accepting using $\{q_i \mid 0 \leq i < s\}$. Symmetric to the previous case.
5. The run r_0^1 is left-accepting using a state in $\{q_i \mid 0 \leq i < s\}$ and right-accepting using $\{q_i \mid 0 \leq i < s\}$. This is impossible because $\mathbb{L}_1 \cup \mathbb{U}_1 \neq \emptyset$ implies that an accepting run must either enter or leave the states $\{q_i \mid 0 \leq i < s\}$ due to the presence of guards in the transitions.

"$\supseteq$" Now, we show that $[\![\varphi]\!] \subseteq M_\varphi(\mathcal{V}(A_\varphi))$. This is, given a model of φ, we have to show that the counter automaton A_φ has a corresponding accepting run. Let $\langle \iota, \mu \rangle$ be a model of φ. Because of Lemma 6, there exists a consistent labelling *Lab* of the constraint graph $G_{\iota,\varphi}$ with $\mu(a,i) = Lab((a,i))$ and $\mu(b,i) = Lab((b,i))$ for all $i \in \mathbb{Z}$. It remains to show that A_φ has a run corresponding to the labelling *Lab*. It is enough to show that the three automata A_1, A_2, and A_3 have runs corresponding to the same labelling *Lab*. That is, there are runs r_0^1 of A_1, r_0^2 of A_2, r_0^3 of A_3 such that $val(r_0^1(i))(x_a) = Lab((a,i))$ and $val(r_0^3(i))(x_b) = Lab((b,i))$ for all $i \in \mathbb{Z}$ as well as $val(r_0^1(i))(x_{t_\varphi}) = val(r_0^2(i))(x_{t_\varphi}) = val(r_0^3(i))(x_{t_\varphi}) = Lab((t_\varphi, i))$ for all $i \in \mathbb{Z}$.

We define a bi-infinite sequence $\nu : \mathbb{Z} \to (\{x_a, x_b, x_{t_\varphi}\} \cup \{x_k | k \in \mathbf{k}\} \cup \{x_j | j \in \{1, ..., p-1\}\} \to \mathbb{Z})$ of valuations of the counters of A_φ such that :

- $\forall k \in \mathbf{k} \forall i \in \mathbb{Z}.\nu(i)(x_k) = \iota(k)$
- $\forall i \in \mathbb{Z}.\nu(i)(x_a) = L((a,i))$ and $\nu(i)(x_b) = L((b,i))$
- $\forall i \in \mathbb{Z}.\nu(i)(x_{t_\varphi}) = L((t_\varphi, i))$
- $\forall j \in \{1, ..., p-1\} \forall i \in \mathbb{Z}.\nu(i)(x_j) = L((a, i+j)) - q$

Now, as ν corresponds in the needed way to *Lab*, it remains to show that each automaton A_1, A_2, A_3 has runs corresponding to ν (taking into account the relevant counters only). Let $\mathcal{L}_{i,\iota} = \max\{\iota(f_k^i) \mid 1 \leq k \leq K_i\}$ and $\mathcal{U}_{i,\iota} = \min\{\iota(g_l^i) \mid 1 \leq l \leq L_i\}$ for $i = 1, 2$. Let $\mathcal{P}_\iota^1 = \{k \mid \mathcal{L}_{1,\iota} \leq k \leq \mathcal{U}_{1,\iota} \wedge k \equiv_s t\}$ and $\mathcal{P}_\iota^2 = \{k \mid \mathcal{L}_{2,\iota} \leq k \leq \mathcal{U}_{2,\iota} \wedge k \equiv_u v\}$.

- The run r_0^1 of the automaton A_1 is composed of three parts. The "left-accepting part", the "middle part", and the "right-accepting" part. There are two cases to consider depending on the emptiness or non-emptiness of the set $\mathcal{P}_\iota^1$.
  - If $\mathcal{P}_\iota^1$ is empty, then the run is constructed in the following way: The left-accepting part goes through the transition $q_L \xrightarrow{\top} q_L$, then the middle part is the transition $q_L \xrightarrow{\neg(\exists i \,.\, \bigwedge_{l \in \mathbb{L}} i \geq l(\mathbf{x_k}) \wedge \bigwedge_{u \in \mathbb{U}} i \leq u(\mathbf{x_k}) \wedge i \equiv_s t)} q_R$ taken at an arbitrary point. The right-accepting part goes through the transition $q_R \xrightarrow{\top} q_R$. Since there are no constraints (up to choosing the values of the parameters $\mathbf{k}$), the run can be trivially chosen to correspond to ν.
  - If $\mathcal{P}_\iota^1$ is not empty, then the left-accepting part goes through the transition $q_L \xrightarrow{\top} q_L$ until one of the guards of the outgoing transitions is satisfied, which happens when x_τ reaches the value $\mathcal{L}_{1,\iota} - 1 - p$. The run then continues to one of the q_i states, namely the one for which $x_\tau + 1 \equiv_s i$. The middle part of the run then goes through the states $\{q_i \mid 0 \leq i < s\}$ till x_τ reaches the value $\mathcal{U}_{1,\iota} - p - 1$. Subsequently, the run continues through the states q_i^j to q_R where it loops forever. Within the run, the constraints that are to be satisfied when taking a transition from a q_i state include:
    1. $x'_a - x_1 \leq q$, which can be satisfied as in the sequence ν of valuations that the run needs to follow, the value of x_1 equals $x'_a - q$,
    2. $x'_{k-1} - x_k \leq 0$ for $1 < k < p$, which can be satisfied as in the sequence of valuations ν to be followed, all x'_{k-1} and x_k have the same value, and

3. $x'_{p-1} - x_{t_\varphi} \leq 0$. This last kind of constraints is tested at the moments when the value of x_τ corresponds to an index l when a diagonal arc arrives to t_φ. At that moment, in the sequence ν of valuations that we try to follow, x'_{p-1} has the value of $L((a,l+p)) - q$, and from the fact that the labelling is consistent (and hence $L((a,l+p)) - L((t_\varphi,l)) \leq q$), it is clear that the last kind of constraints can be satisfied too.

Hence, there is an accepting run corresponding to the bi-infinite sequence ν of valuations. A similar reasoning applies when passing through the states q_i^j.

– The runs r_0^2 of A_2 and r_0^3 of A_3 are constructed in a similar way.

□

Before proceeding with the proof of Theorem 1, let us first introduce some notation. Let $\mathbf{x}$ be an arbitrary set of variables, interpreted over some domain D, and $I \subseteq \mathbf{x} \mapsto D$ be a set of valuations. For some superset $\mathbf{y} \supset \mathbf{x}$ of the set of variables, we define $I\uparrow_{\mathbf{y}} = \{\iota : \mathbf{y} \to D \mid \iota\downarrow_{\mathbf{x}} \in I\}$. If $\mathbf{x_1}, \mathbf{y_1}$ and $\mathbf{x_2}, \mathbf{y_2}$ are sets of variables interpreted over domains D_1 and D_2, respectively, $\mathbf{x_1} \subseteq \mathbf{y_1}$, $\mathbf{x_2} \subseteq \mathbf{y_2}$, and $I_{12} \subseteq \mathbf{x_1} \mapsto D_1 \times \mathbf{x_2} \mapsto D_2$ is a set of pairs of valuations, let $I_{12}\uparrow_{\mathbf{y_1},\mathbf{y_2}} = \{\langle \iota_1, \iota_2\rangle \mid \iota_1 : \mathbf{y_1} \to D_1,\ \iota_2 : \mathbf{y_2} \to D_2,\ \langle \iota_1\downarrow_{\mathbf{x_1}}, \iota_2\downarrow_{\mathbf{x_2}}\rangle \in I_{12}\}$. The proof is by induction on the structure of φ. Lemma 11 takes care about the cases of φ being of type (F1)-(F3). If φ is a PA constraint on $\mathbf{k}$, the proof is immediate.

For the inductive case $\varphi = \psi_1 \wedge \psi_2$, let $\mathbf{k_i}$ and $\mathbf{a_i}$, be the sets of array-bound and array variables of ψ_i, for $i = 1,2$, respectively. We have by Lemma 3, that:

$$\mathcal{V}(A_{\psi_1} \otimes A_{\psi_2}) = \mathcal{V}(A_{\psi_1})\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}} \cap \mathcal{V}(A_{\psi_2})\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}$$

where $\mathbf{x_1}$ are the counters of A_{ψ_1} and $\mathbf{x_2}$ are the counters of A_{ψ_2}. Applying M_φ to this equality, we obtain:

$$M_\varphi(\mathcal{V}(A_{\psi_1} \otimes A_{\psi_2})) = M_\varphi(\mathcal{V}(A_{\psi_1})\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}) \cap M_\varphi(\mathcal{V}(A_{\psi_2})\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}})$$

since M_φ is defined point wise on sets of runs. By the induction hypothesis, we have $M_{\psi_i}(\mathcal{V}(A_{\psi_i})) = [\![\psi_i]\!]$, for $i = 1,2$. It is easy to see that, $M_\varphi(\mathcal{V}(A_{\psi_i})\uparrow_{\mathbf{x_1}\cup\mathbf{x_2}}) = [\![\psi_i]\!]\uparrow_{\mathbf{k_1}\cup\mathbf{k_2},\mathbf{a_1}\cup\mathbf{a_2}}$, for $i = 1,2$. Hence, we have:

$$M_\varphi(\mathcal{V}(A_{\psi_1} \otimes A_{\psi_2})) = [\![\psi_1]\!]\uparrow_{\mathbf{k_1}\cup\mathbf{k_2},\mathbf{a_1}\cup\mathbf{a_2}} \cap [\![\psi_2]\!]\uparrow_{\mathbf{k_1}\cup\mathbf{k_2},\mathbf{a_1}\cup\mathbf{a_2}} = [\![\psi_1 \wedge \psi_2]\!]$$

The proof for the case $\varphi = \psi_1 \vee \psi_2$ follows a similar argument. □