Counting Constraints in Flat Array Fragments

Francesco Alberti[1], Silvio Ghilardi[2], and Elena Pagani[2]

[1] Fondazione S. Raffaele, Milano, Italy
[2]Università degli Studi di Milano, Milano, Italy

May 28, 2016

Abstract

We identify a fragment of Presburger arithmetic enriched with free function symbols and cardinality constraints for interpreted sets, which is amenable to automated analysis. We establish decidability and complexity results for such a fragment and we implement our algorithms. The experiments run in discharging proof obligations coming from invariant checking and bounded model-checking benchmarks show the practical feasibility of our decision procedure.

1 Introduction

Enriching logic formalisms with counting capabilities is an important task in view of the needs of many application areas, ranging from database theory to formal verification. Such enrichments have been designed both in the description logics area and in the area of Satisfiability Modulo Theories (SMT), where some of the most important recent achievements were decidability and complexity bounds for BAPA [13] - the enrichment of Presburger arithmetic with the ability of talking about finite sets and their cardinalities. As pointed out in [14], BAPA constraints can be used for program analysis and verification by expressing data structure invariants, simulations between program fragments or termination conditions. The analysis of BAPA constraints was successfully extended also to formalisms encompassing multisets [17] as well as direct/inverse images along relations and functions [22].

A limitation of BAPA and its extensions lies in the fact that only uninterpreted symbols (for sets, relations, functions, etc.) are allowed. On the other hand, it is well-known that a different logical formalism, namely unary counting quantifiers, can be used in order to reason about the cardinality of definable (i.e. of interpreted) sets. Unary counting quantifiers can be added to Presburger arithmetic without compromising decidability, see [18],

however they might be quite problematic if combined in an unlimited way with free function symbols. In this paper, we investigate the extension of Presburger arithmetic including both counting quantifiers and uninterpreted function symbols, and we isolate fragments where we can achieve decidability and in some cases also relatively good complexity bounds. The key ingredient to isolate such fragments is the notion of flatness: roughly, in a flat formula, subterms of the kind $a(t)$ (where a is a free function symbol) can occur only if t is a variable. By itself, this naif flatness requirement is useless (any formula can match it to the price of introducing extra quantified variables); in order to make it effective, further syntactic restrictions should be incorporated in it, as witnessed in [1]. This is what we are going to do in this paper, where suitable notions of 'flat' and 'simple flat' formulæ are introduced in the rich context of Presburger arithmetic enriched with free function symbols and with unary counting quantifiers (we use free function symbols to model arrays, see below).

The fragments we design are all obviously more expressive than BAPA, but they do not come from pure logic motivations, on the contrary they are suggested by an emerging application area, namely the area of verification of fault-tolerant distributed systems. Such systems (see [7] for a good account) are modeled as partially synchronous systems, where a finite number of identical processes operate in lock-step (in each *round* they send messages, receive messages, and update their local state depending on the local state at the beginning of the round and the received messages). Messages can be lost, processes may omit to perform some tasks or also behave in a malicious way; for these reasons, the fact that some actions are enabled or not, and the correctness of the algorithms themselves, are subject to threshold conditions saying for instance that some qualified majority of processes are in a certain status or behave in a non-faulty way. Verifications tasks thus have to handle cardinality constraints of the kind studied in this paper (the reader interested in full formalization examples can directly go to Section 5).

The paper is organized as follows: we first present basic syntax (Section 2), then decidability (Section 3) and complexity (Section 4) results; experiments with our prototypical implementation are supplied in Section 5, and Section 6 concludes the work.

2 Preliminaries

We work within Presburger arithmetic enriched with free function symbols and cardinality constraints. This is a rather expressive logic, whose syntax is summarized in Figure 1. Terms and formulæ are interpreted in the natural way over the domain of integers $\mathbb{Z}$; as a consequence, satisfiability of a formula ϕ means that it is possible to assign values to parameters, free variables and array-ids so as to make ϕ true in $\mathbb{Z}$ (validity of ϕ means that

$0, 1, \ldots$		$\in \mathbb{Z}$	numerals (numeric constants)
$x, y, z, \ldots$		$\in Var$	individual variables
$M, N, \ldots$		$\in Par$	parameters (free constants)
$a, b, \ldots$		$\in Arr$	array ids (free unary function symbols)
$t, u, \ldots$	$::=$	$n \mid M \mid x \mid t + t \mid -t \mid a(t) \mid \sharp\{x \mid \phi\}$	terms
$A, B, \ldots$	$::=$	$t < t \mid t = t \mid t \equiv_n t$	atoms
$\phi, \psi, \ldots$	$::=$	$A \mid \phi \wedge \phi \mid \neg\phi \mid \exists x\, \phi$	formulae

Figure 1: Syntax

$\neg\phi$ is not satisfiable, equivalence of ϕ and ψ means that $\phi \leftrightarrow \psi$ is valid, etc.). We nevertheless implicitly assume few constraints (to be explained below) about our intended semantics.

To denote integer numbers, we have (besides variables and numerals) also parameters: the latter denote unspecified integers. Among parameters, we always include a specific parameter (named N) identifying the dimension of the system - alias the length of our arrays: in other words, it is assumed that for all array identifiers $a \in Arr$, the value $a(x)$ is conventional (say, zero) outside the interval $[0, N) = \{n \in \mathbb{Z} \mid 0 \leq n < N\}$. Although binary free function symbols are quite useful in some applications, in this paper we prefer not to deal with them. The operator $\sharp\{x \mid \phi\}$ *indicates the cardinality of the finite set formed by the* $x \in [0, N)$ *such that* $\phi(x)$ *holds.*

Notice that the cardinality constraint operator $\sharp\{x \mid -\}$, as well the quantifier $\exists x$, binds the variable x; below, we indicate with $\psi(\underline{x})$ (resp. $t(\underline{x})$) the fact that the formula ψ (the term t) has free individual variables included in the list $\underline{x}$. When we speak of a substitution, we always mean 'substitution without capture', meaning that, when we replace the free occurrences of a variable x with a term u in a formula ϕ or in a term t, the term u should not contain free variables that might be located inside the scope of a binder for them once the substitution is performed; the result of the substitution is denoted with $\phi(u/x)$ and $t(u/x)$.

The logic of Figure 1 is far from being tractable, because even the combination of free function symbols and Presburger arithmetic lands in a highly undecidable class [9]. We are looking for a mild fragment, nevertheless sufficiently expressive for our intended applications. These applications mostly come from verification tasks, like bounded model checking or invariant checking. Our aim is to design a decidable fragment (so as to be able not only to produce certifications, but also to find bugs) with some minimal closure properties; from this point of view, notice that for bounded model checking closure under conjunctions is sufficient, but for invariant checking we need also closure under negations in order to discharge entailments.

2.1 Flat formulæ

We now introduce some useful subclasses of the formulæ built up according to the grammar of Figure 1.

- *Arithmetic formulæ*: these are built up from the grammar of Figure 1 without using neither array-ids nor cardinality constraint operators; we use letter $\alpha, \beta, \ldots$ for arithmetic formulæ. Recall that, according to the well-known quantifier elimination result, arithmetic formulæ are equivalent to *quantifier-free* arithmetic formulæ.

- *Constraint formulæ*: these are built up from the grammar of Figure 1 without using array-ids.

- *Basic formulæ*: these are obtained from an arithmetic formula by simultaneously replacing some free variables by terms of the kind $a(y)$, *where y is a variable and a an array-id.* When we need to display full information, we may use the notation $\alpha(\underline{y}, \mathbf{a}(\underline{y}))$ to indicate basic formulæ. By this notation, we mean that $\underline{y} = y_1, \ldots, y_n$ are variables, $\mathbf{a} = a_1, \ldots, a_s$ are array-ids and that $\alpha(\underline{y}, \mathbf{a}(\underline{y}))$ is obtained from an arithmetic formula $\alpha(\underline{y}, \underline{z})$ (where $\underline{z} = z_{11}, \ldots, z_{sn}$) by replacing z_{ij} with $a_i(y_j)$ ($i = 1, \ldots, s$ and $j = 1, \ldots, n$).

- *Flat formulæ*: these are recursively defined as follows (i) basic formulæ are flat formulæ; (ii) if ϕ is a flat formula, β is a basic formula, z and x are variables, then $\phi(\sharp\{x \,|\, \beta\} \,/\, z)$ is a flat formula.[1]

Notice that all the above classes are closed under Boolean operations (in particular, under negations). The following result is proved in [18] (see also Appendix A):

Theorem 1 *For every constraint formula one can compute an arithmetic formula equivalent to it.*

3 Satisfiability for flat formulæ

We shall show that flat formulæ are decidable for satisfiability. In fact, we shall show decidability of the slightly larger class covered by the following

Definition 1 *Extended flat formulæ (briefly, E-flat formulæ) are formulæ of the kind*

$$\exists \underline{z}.\ \alpha \ \wedge\ \sharp\{x \mid \beta_1\} = z_1 \ \wedge \cdots \wedge\ \sharp\{x \mid \beta_K\} = z_K \tag{1}$$

[1] If we want to emphasize the way the basic formula β is built up, following the above conventions, we may write it as $\beta(x, \underline{y}, \mathbf{a}(x), \mathbf{a}(\underline{y}))$; here, supposing that $\mathbf{a}$ is $a_1, \ldots, a_s$, since x is a singleton, the tuple $\mathbf{a}(x)$ is $a_1(x), \ldots, a_s(x)$.

where $\underline{z} = z_1, \ldots, z_K$ *and* $\alpha, \beta_1, \ldots, \beta_K$ *are basic formulæ and* x *does not occur in* α*.*

Notice that α and the β_j in (1) above may contain further free variables $\underline{y}$ (besides $\underline{z}$) as well as the terms $\mathbf{a}(\underline{y})$ and $\mathbf{a}(\underline{z})$; the β_j may contain occurrences of x and of $\mathbf{a}(x)$.

That flat formulæ are also E-flat can be seen as follows: due to the fact that our substitutions avoid captures, we can use equivalences like $\phi(t/z) \leftrightarrow \exists z\,(t = z \wedge \phi)$ in order to abstract out the terms $t := \sharp\{x \mid \alpha\}$ occurring in the recursive construction of a flat formula ϕ. By repeating this linear time transformation, we end up in a formula of the kind (1). However, not all E-flat formulæ are flat because the dependency graph associated to (1) might not be acyclic (the graph we are talking about has the z_j as nodes and has an arc $z_j \to z_i$ when z_i occurs in β_j). The above conversion of a flat formula into a formula of the form (1) on the other hand produces an E-flat formula whose associated graph is acyclic.

We prove a technical lemma showing how we can manipulate E-flat formulæ without loss of generality. Formulae $\varphi_1, \ldots, \varphi_K$ are said to be a partition iff the formulæ $\bigvee_{l=1}^{K} \varphi_l$ and $\neg(\varphi_l \wedge \varphi_h)$ (for $h \neq l$) are valid. Recall that the existential closure of a formula is the sentence obtained by prefixing it with a string of existential quantifiers binding all variables having a free occurrence in it.

Lemma 1 *The existential closure of an E-flat formula is equivalent to a sentence of the kind*

$$\exists \underline{z}\ \exists \underline{y}.\ \alpha(\underline{y}, \underline{z}) \wedge \sharp\{x \mid \beta_1(x, \mathbf{a}(x), \underline{y}, \underline{z})\} = z_1 \wedge \cdots \wedge \sharp\{x \mid \beta_K(x, \mathbf{a}(x), \underline{y}, \underline{z})\} = z_K \tag{2}$$

where $\underline{y}$ *and* $\underline{z} := z_1, \ldots, z_K$ *are variables,* α *is arithmetical, and the formulæ* $\beta_1, \ldots, \beta_K$ *are basic and form a partition.*

Proof. The differences between (the matrices of) (2) and (1) are twofold: first in (2), the β_l form a partition and, second, in (1) the terms $a_s(y_i)$ and $a_s(z_h)$ (for $a_s \in \mathbf{a}$ and $y_i \in \underline{y}$, $z_h \in \underline{z}$) may occur in α and in the β_l.

We may disregard the $a_s(z_h)$ without loss of generality, because we can include them in the $a_s(y_i)$: to this aim, it is sufficient to take a fresh y, to add the conjunct $y = z_h$ to α and to replace everywhere $a_s(z_h)$ by $a_s(y)$. In order to eliminate also a term like $a_s(y_i)$, we make a guess and distinguish the case where $y_i \geq N$ and the case where $y_i < N$ (formally, 'making a guess' means to replace (1) with a disjunction - the two disjuncts being obtained by adding to α the case description). According to the semantics conventions we made in Section 2, the first case is trivial because we can just replace $a_s(y_i)$ by 0. In the other case, we first take a fresh variable u and apply the equivalence $\gamma(\ldots a_s(y_j) \ldots) \leftrightarrow \exists u\,(a_s(y_j) = u \wedge \gamma(\ldots u \ldots))$ (here γ is the whole (1)); then we replace $a_s(y_j) = u$ by the equivalent formula

$\sharp\{x \mid x = y_j \wedge a_s[x] = u\} = 1$ and finally the latter by $\exists u' \, (u' = 1 \wedge \sharp\{x \mid x = y_j \wedge a_s[x] = u\} = u')$ (the result has the desired shape once we move the new existential quantifiers in front).

After this, we still need to modify the β_l so that they form a partition (this further step produces an exponential blow-up). Let $\psi(\underline{y})$ be the matrix of a formula of the kind (2), where the β_l are not a partition. Let us put $\underline{K} := \{1, \ldots, K\}$ and let us consider further variables $\underline{u} = \langle u_\sigma \rangle_\sigma$, for $\sigma \in 2^{\underline{K}}$. Then it is clear that the existential closure of ψ is equivalent to the formula obtained by prefixing the existential quantifiers $\exists \underline{u} \, \exists \underline{z}$ to the formula

$$\left(\alpha \ \wedge \ \bigwedge_{l=1}^{K} z_l = \sum_{\sigma \in 2^{\underline{K}}, \, \sigma(l)=1} u_\sigma \right) \wedge \bigwedge_{\sigma \in 2^{\underline{K}}} \sharp\{x \mid \beta_\sigma\} = u_\sigma \tag{3}$$

where $\beta_\sigma; = \bigwedge_{l=1}^{K} \epsilon_{\sigma(l)} \beta_l$ (here $\epsilon_{\sigma(l)}$ is '$\neg$' if $\sigma(l) = 0$, it is a blank space otherwise). $\dashv$

Theorem 2 *Satisfiability of E-flat formulæ is decidable.*

Proof. We reduce satisfiability of (2) to satisfiability of constraint formulæ which is decidable by Theorem 1; in detail, we show that (2) is equisatisfiable with the constraint formula below (containing extra free variables $z_S, z_{l,S}$):

$$\begin{aligned} &\alpha \wedge \bigwedge_{S \in \wp(\underline{K})} \left(z_S = \sharp\{x \mid \bigwedge_{l \in S} \exists \underline{u} \, \beta_l(x, \underline{u}, \underline{y}, \underline{z}) \wedge \bigwedge_{l \notin S} \forall \underline{u} \neg \beta_l(x, \underline{u}, \underline{y}, \underline{z})\} \right) \wedge \\ &\wedge \bigwedge_{S \in \wp(\underline{K})} \left(z_S = \sum_{l \in S} z_{l,S} \right) \wedge \bigwedge_{l=1}^{K} \left(z_l = \sum_{S \in \wp(\underline{K}), l \in S} z_{l,S} \right) \wedge \bigwedge_{l \in S \in \wp(\underline{K})} z_{l,S} \geq 0 \end{aligned} \tag{4}$$

(according to our notations, the basic formulæ $\beta_l(x, \mathbf{a}(x), \underline{y}, \underline{z})$ from (2) were supposed to be built up from the arithmetic formulæ $\beta_l(x, \underline{u}, \underline{y}, \underline{z})$ by replacing the variables $\underline{u} = u_1, \ldots, u_s$ with the terms $\mathbf{a}(x) = a_1 x), \ldots, a_s(x))$.

Suppose that (4) *is satisfiable.* Then there is an assignment V to the free variables occurring in it so that (4) is true in the standard structure of the integers (for simplicity, we use the same name for a free variable and for the integer assigned to it by V). If $\mathbf{a} = a_1, \ldots, a_s$, we need to define $a_s(i)$ for all s and for all $i \in [0, N)$. For every $l = 1, \ldots, K$ this must be done in such a way that there are exactly z_l integer numbers taken from $[0, N)$ satisfying $\beta_l(x, \mathbf{a}(x), \underline{y}, \underline{z})$. The interval $[0, N)$ can be partioned by associating with each $i \in [0, N)$ the set $i_S = \{l \in \underline{K} \mid \exists \underline{u} \, \beta_l(i, \underline{u}, \underline{y}, \underline{z}) \text{ holds under } V\}$. For every $S \in \wp(\underline{K})$ the number of the i such that $i_S = S$ is z_S; for every $l \in S$, pick $z_{l,S}$ among them and, for these selected i, let the s-tuple $\mathbf{a}(i)$ be equal to an s-tuple $\underline{u}$ such that $\beta_l(i, \underline{u}, \underline{y}, \underline{z})$ holds (for this tuple $\underline{u}$, since the β_l are a partition, $\beta_h(i, \underline{u}, \underline{y}, \underline{z})$ does not hold, if $h \neq l$). Since $z_S = \sum_{l \in S} z_{l,S}$

and since $\sum_S z_S$ is equal to the length of the interval $[0, N)$, the definition of the $\mathbf{a}$ is complete. The formula (2) is true by construction.

On the other hand *suppose that* (2) *is satisfiable* under an assignment V; we need to find $V(z_S)$, $V(z_{l,S})$ (we again indicate them simply as $z_S, z_{l,S}$) so that (4) is true. For z_S there is no choice, since $z_S = \sharp\{i \mid \bigwedge_{l\in S} \exists\underline{u}\,\beta_l(i,\underline{u},\underline{y},\underline{z}) \wedge \bigwedge_{l\notin S} \forall\underline{u}\neg\beta_l(i,\underline{u},\underline{y},\underline{z})\}$ must be true; for $z_{l,S}$, we take it to be the cardinality of the set of the i such that $\beta_l(i,\mathbf{a}(i),\underline{y},\underline{z})$ holds under V and $S = \{h \in \underline{K} \mid \exists\underline{u}\ \beta_h(i,\underline{u},\underline{y},\underline{z})$ holds under $V\}$. In this way, for every S, the equality $z_S = \sum_{l\in S} z_{l,S}$ holds and for every l, the equality $z_l = \sum_{S\in\wp(\underline{K}),l\in S} z_{l,S}$ holds too. Thus the formula (4) becomes true under our extended V. ⊣

4 A more tractable subcase

Thus satisfiability of flat formulæ is decidable; since flat formulæ are closed under Boolean combinations, validity of implications of flat sentences is decidable too. This makes our result a *complete* algorithm for checking invariants in verification applications. However, the complexity of the decision procedure is very high: Lemma 1 introduces an exponential blow-up and other exponential blow-ups are introduced by Theorem 2 and by the decision procedure (via quantifier elimination) from [18]. Of course, all this might be subject to dramatic optimizations (to be investigated by future reseach); in this paper we show that there is a much milder (and still practically useful) fragment.

Definition 2 *Simple flat formulæ are recursively defined as follows: (i) basic formulæ are simple flat formulæ; (ii) if ϕ is a simple flat formula, $\beta(\mathbf{a}(x),\mathbf{a}(\underline{y}),\underline{y})$ is a basic formula and x, z are variables, then $\phi(\sharp\{x \,|\, \beta\} \,/\, z)$ is a simple flat formula.*

As an example of a simple flat formula consider the following one

$$a'(y) = z \wedge \sharp\{x \mid a'(x) = a(x)\} \geq N\text{–}1 \wedge (\sharp\{x \mid a'(x) = a(x)\} < N \rightarrow a(y) \neq z)$$

expressing that $a' = write(a, y, z)$ (i.e. that the array a' is obtained from a by over-writing z in the entry y).

Definition 3 *Simple E-flat formulæ are formulæ of the kind*

$$\begin{aligned}\exists\underline{z}.\ \alpha(\mathbf{a}(\underline{y}),\mathbf{a}(\underline{z}),\underline{y},\underline{z})\ \wedge\ \sharp\{x \mid \beta_1(\mathbf{a}(x),\mathbf{a}(\underline{y}),\mathbf{a}(\underline{z}),\underline{y},\underline{z})\} = z_1\ \wedge\cdots\\ \cdots\wedge\ \sharp\{x \mid \beta_K(\mathbf{a}(x),\mathbf{a}(\underline{y}),\mathbf{a}(\underline{z}),\underline{y},\underline{z}))\} = z_K\end{aligned} \tag{5}$$

where α and the β_i are basic.

It is easily seen that (once again) simple flat formulæ are closed under Boolean combinations and that simple flat formulæ are simple E-flat formulæ (the converse is not true, for ciclicity of the dependence graph of the z_i's in (5)).

The difference between simple and non simple flat/E-flat formulæ is that in simple formulæ *the abstraction variable cannot occur outside the read of an array symbol* (in other words, the β, β_i from the above definition are of the kind $\beta_i(\mathbf{a}(x), \mathbf{a}(\underline{y}), \mathbf{a}(\underline{z}), \underline{y}, \underline{z})$ and not of the kind $\beta_i(\mathbf{a}(x), \mathbf{a}(\underline{y}), \mathbf{a}(\underline{z}), x, \underline{y}, \underline{z})$). This restriction has an important semantic effect, namely that formulæ (5) are equi-satisfiable to formulæ which are *permutation-invariant*, in the following sense. The truth value of an arithmetical formula or of a formula like $z = \sharp\{x \mid \alpha(\mathbf{a}(x), \underline{y})\}$ is not affected by a permutation of the values of the $\mathbf{a}(x)$ for $x \in [0, N)$, because x does not occur free in α (permuting the values of the $\mathbf{a}(x)$ may on the contrary change the value of a flat non simple sentence like $z = \sharp\{x \mid a(x) \leq x\}$). This 'permutation invariance' will be exploited in the argument proving the correctness of decision procedure of Theorem 3 below. Formulae (5) themselves are not permutation-invariant because of subterms $\mathbf{a}(\underline{z}), \mathbf{a}(\underline{y})$, so we first show how to eliminate them up to satisfiability:

Lemma 2 *Simple E-flat formulæ are equi-satisfiable to disjunctions of permutation-invariant formulæ of the kind*

$$\exists \underline{z}.\ \alpha(\underline{y}, \underline{z}) \wedge \sharp\{x \mid \beta_1(\mathbf{a}(x), \underline{y}, \underline{z})\} = z_1 \wedge \cdots \wedge \sharp\{x \mid \beta_K(\mathbf{a}(x), \underline{y}, \underline{z}))\} = z_K \tag{6}$$

Proof. Let us take a formula like (5): we convert it to an equi-satisfiabe disjunction of formulæ of the kind (6). The task is to eliminate terms $\mathbf{a}(\underline{z})$, $\mathbf{a}(\underline{y})$ by a series of guessings (each guessing will form the content of a disjunct). Notice that we can apply the procedure of Lemma 1 to eliminate the $\mathbf{a}(\underline{z})$, but for the $\mathbf{a}(\underline{y})$ we must operate differently (the method used in Lemma 1 introduced non simple abstraction terms).

Let us suppose that $\underline{y} := y_1, \ldots, y_m$ and that, after a first guess, α contains the conjunct $y_j < N$ for each $j = 1, \ldots, m$ (if it contains $y_j \geq N$, we replace $a_s(y_j)$ by 0); after a second series of guesses, we can suppose also that α contains the conjuncts $y_{j_1} \neq y_{j_2}$ for $j_1 \neq j_2$ (if it contains $y_{j_1} = y_{j_2}$, we replace y_{j_1} by y_{j_2} everywhere, making y_{j_1} to disappear from the whole formula). In the next step, (i) we introduce for every $a \in \mathbf{a}$ and for every $j = 1, \ldots, m$ a fresh variable u_{aj}, (ii) we replace everywhere $a(y_j)$ by u_{aj} and (iii) we conjoin to α the equalities $a(y_j) = u_{aj}$. In this way we get a formula of the following kind

$$\exists \underline{z}.\ \bigwedge_{a \in \mathbf{a}, y_j \in \underline{y}} a(y_j) = u_{aj} \wedge \alpha(\underline{y}, \underline{u}, \underline{z}) \ \wedge \bigwedge_{l=1}^{K} \sharp\{x \mid \beta_l(\mathbf{a}(x), \underline{y}, \underline{u}, \underline{z})\} = z_l \tag{7}$$

where $\underline{u}$ is the tuple formed by the u_{aj} (varying a and j). We now make another series of guesses and conjoin to α either $u_{aj} = u_{a'j'}$ or $u_{aj} \neq u_{a'j'}$ for $(a, j) \neq (a', j')$. Whenever $u_{aj} = u_{a'j'}$ is conjoined, u_{aj} is replaced by $u_{a'j'}$ everywhere, so that u_{aj} disappears completely. The resulting formula still has the form (7), but now the map $(a, j) \mapsto u_{aj}$ is not injective anymore (otherwise said, u_{aj} now indicates the element from the tuple $\underline{u}$ associated with the pair (a, j) and we might have that the same u_{aj} is associated with different pairs (a, j)).

Starting from (7) so modified, let us define now the equivalence relation among the y_j that holds between y_j and $y_{j'}$ whenever for all $a \in \mathbf{a}$ there is $u_a \in \underline{u}$ such that α contains the equalities $a(y_j) = u_a$ and $a(y'_j) = u_a$. Each equivalence class E is uniquely identified by the corresponding function f_E from $\mathbf{a}$ into $\underline{u}$ (it is the function that for each $y_j \in E$ maps $a \in \mathbf{a}$ to the $u_a \in \underline{u}$ such that α contains as a conjunct the equality $a(y_j) = u_a$). Let $E_1, \ldots, E_r$ be the equivalence classes and let $n_1, \ldots, n_r$ be their cardinalities. We claim that (7) is equisatisfiable to

$$\begin{aligned} \exists \underline{z}.\ \alpha(\underline{y}, \underline{u}, \underline{z}) \ \wedge \bigwedge_{q=1}^{r} \sharp\{x \mid \bigwedge_{a \in \mathbf{a}} a(x) = f_{E_q}(a)\} \geq n_q \ \wedge \\ \wedge \bigwedge_{l=1}^{K} \sharp\{x \mid \beta_l(\mathbf{a}(x), \underline{y}, \underline{u}, \underline{z})\} = z_l \end{aligned} \tag{8}$$

In fact, satisfiability of (7) trivially implies the satisfiability of the formula (8); vice versa, since (8) is permutation-invariant, if it is satisfiable we can modify any assignment satisfying it via a simultaneous permutation of the values of the $a \in \mathbf{a}$ so as to produce an assignment satisfying (7).

We now need just the trivial observation that the inequalities $\sharp\{x \mid \bigwedge_{a \in \mathbf{a}} a(x) = f_{E_q}(a)\} \geq n_q$ can be replaced by the formulæ $\sharp\{x \mid \bigwedge_{a \in \mathbf{a}} a(x) = f_{E_q}(a)\} = z'_q \ \wedge \ z'_q \geq n_q$ (for fresh z'_q) in order to match the syntactic shape of (6). ⊣

We can freely assume that quantifiers do not occur in simple flat formulæ: this is without loss of generality because such formulæ are built up from arithmetic and basic formulæ.[2]

Theorem 3 *Satisfiability of simple flat formulæ can be decided in NP (and thus it is an NP-complete problem).*

Proof. First, by applying the procedure of the previous Lemma we can reduce to the problem of checking the satisfiability of formulæ of the kind

$$\alpha(\underline{y}, \underline{z}) \ \wedge \ \sharp\{x \mid \beta_1(\mathbf{a}(x), \underline{y}, \underline{z})\} = z_1 \ \wedge \cdots \wedge \ \sharp\{x \mid \beta_K(\mathbf{a}(x), \underline{y}, \underline{z})\} = z_K \tag{9}$$

[2] By the quantifier-elimination result for Presburger arithmetic, it is well-known that arithmetic formulæ are equivalent to quantifier-free ones. The same is true for basic formulæ because they are obtained from arithmetic formulae by substitutions without capture.

where $\alpha, \beta_1, \ldots, \beta_K$ are basic (notice also that each formula in the output of the procedure of the previous Lemma comes from a polynomial guess).

Suppose that $A_1(\mathbf{a}(x), \underline{y}, \underline{z}), \ldots, A_L(\mathbf{a}(x), \underline{y}, \underline{z})$ are the atoms occurring in $\beta_1, \ldots, \beta_K$. For a Boolean assignment σ to these atoms, we indicate with $[\![\beta_j]\!]^\sigma$ the Boolean value (0 or 1) the formula β_l has under such assignment. We first claim that (9) is satisfiable iff there exists *a set of assignments* Σ such that the formula

$$\alpha(\underline{y}, \underline{z}) \ \wedge \bigwedge_{\sigma \in \Sigma} \exists \underline{u} \left(\bigwedge_{j=1}^{L} \epsilon_{\sigma(A_j)} A_j(\underline{u}, \underline{y}, \underline{z}) \right) \wedge \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_K \end{bmatrix} = \sum_{\sigma \in \Sigma} v_\sigma \begin{bmatrix} [\![\beta_1]\!]^\sigma \\ [\![\beta_2]\!]^\sigma \\ \vdots \\ [\![\beta_K]\!]^\sigma \end{bmatrix} \wedge$$
$$\wedge \sum_{\sigma \in \Sigma} v_\sigma = N \wedge \bigwedge_{\sigma \in \Sigma} v_\sigma > 0 \tag{10}$$

is satisfiable (we introduced extra fresh variables v_σ, for $\sigma \in \Sigma$; notation $\epsilon_{\sigma(A_j)}$ is the same as in the proof of Lemma 1). In fact, on one side, if (9) is satisfiable under V, we can take as Σ the set of assigments for which $\bigwedge_{j=1}^{L} \epsilon_{\sigma(A_j)} A_j(\mathbf{a}(i), \underline{y}, \underline{z})$ is true under V for some $i \in [0, N)$ and for v_σ the cardinality of the set of the $i \in [0, N)$ for which $\bigwedge_{j=1}^{L} \epsilon_{\sigma(A_j)} A_j(\mathbf{a}(i), \underline{y}, \underline{z})$ holds. This choice makes (10) true. Vice versa, if (10) is true under V, in order to define the value of the tuple $\mathbf{a}(i)$ (for $i \in [0, N)$), pick for every $\sigma \in \Sigma$ some $\underline{u}_\sigma$ such that $\bigwedge_{j=1}^{L} \epsilon_{\sigma(A_j)} A_j(\underline{u}_\sigma, \underline{y}, \underline{z})$ holds; then, supposing $\Sigma = \{\sigma_1, \ldots, \sigma_h\}$, let $\mathbf{a}(i)$ be equal to $\underline{u}_{\sigma_1}$ for $i \in [0, v_{\sigma_1})$, to $\underline{u}_{\sigma_2}$ for $i \in [v_{\sigma_1}, v_{\sigma_2})$, etc. Since we have that $\sum_{\sigma \in \Sigma} v_\sigma = N$, the definition of the interpretation of the $\mathbf{a}$ is complete (any other permutation of the values $\mathbf{a}(x)$ inside $[0, N)$ would fit as well). In this way, formula (9) turns out to be true.

We so established that our original formula is satisfiable iff there is some Σ such that (10) is satisfiable; the only problem we still have to face is that Σ might be exponentially large. To reduce to a polynomial Σ, we use the same technique as in [15]. In fact, if (10) is satisfiable, then the column vector $(z_1, \ldots, z_K)^T$ is a linear combination with positive integer coefficients of the 0/1-vectors $([\![\beta_1]\!]^\sigma, \cdots, [\![\beta_K]\!]^\sigma)^T$ and it is known from [8] that, if this is the case, the same result can be achieved by assuming that at most $2K \log_2(4K)$ of the v_σ are nonzero. Thus polynomially many Σ are sufficient and for such Σ, a satisfying polynomial assignment for the existential Presburger formula (10) is a polynomial certificate. $\dashv$

4.1 Some heuristics

We discuss here some useful heuristics for the satisfiability algorithm for simple flat formulæ (these heuristics have been implemented in our prototype).

1.- The satisfiability test involves all formulæ (10) for each set of assignments Σ having cardinality *at most* $M = \lceil 2K \log_2(4K) \rceil$ (actually, one can improve this bound, see [15]). If we replace in (10), for every σ, the conjunct $v_\sigma > 0$ by $v_\sigma \geq 0$ and the conjunct $\exists \underline{u} \, (\bigwedge_{j=1}^{L} \epsilon_{\sigma(A_j)} A_j(\underline{u}, \underline{y}, \underline{z}))$ by $v_\sigma > 0 \rightarrow \exists \underline{u} \, (\bigwedge_{j=1}^{L} \epsilon_{\sigma(A_j)} A_j(\underline{u}, \underline{y}, \underline{z}))$, we can limit ourselves to the Σ having cardinality *equal to* M. This trick is useful if, for some reason, we prefer to go through any sufficient set of assignments (like the set of all assignments supplied by some Boolean propagation, see below).

2.- There is no need to consider assignments σ over the set of the atoms A_j occurring in the $\beta_1, \ldots, \beta_K$; any set of formulæ generating the $\beta_1, \ldots, \beta_K$ by Boolean combinations fits our purposes. As a consequence, the choice of these 'atoms' is subject to case-by-case evaluations.

3.- Universally quantified formulæ of the kind $\forall x \, (0 \leq x \wedge x < N \rightarrow \beta)$ can be turned into flat formulæ by rewriting them as $N = \sharp \{x \mid \beta\}$ (and in fact such universally quantified formulæ often occur in our benchmarks suite). These formulæ contribute to (9) via the conjuncts of the kind $z_i = N \wedge \sharp \{x \mid \beta_i(\mathbf{a}(x), \underline{y}, \underline{z})\} = z_i$. It is quite useful to consider the $\{\beta_{i_1}, \ldots, \beta_{i_L}\}$ arising in this way as atoms (in the sense of point 2 above) and restrict to the assignments σ such that $\sigma(\beta_{i_1}) = \cdots = \sigma(\beta_{i_L}) = 1$.

4.- Boolean propagation is a quite effective strategy to prune useless assignments; in our context, as soon as a partial assignment σ is produced inside the assignments enumeration subroutine, an SMT solver is invoked to test the satisfiability of $\alpha(\underline{y}, \underline{z}) \wedge \bigwedge_{j \in dom(\sigma)} \epsilon_{\sigma(A_j)} A_j(\underline{u}, \underline{y}, \underline{z})$; since this is a (skolemized) conjunct of (10), if the test is negative the current partial assignment is discarded and next partial assignment (obtained by complementing the value of the last assigned literal) is taken instead.

5 Examples and experiments

We implemented a prototype ArCa-Sat[3] producing out of simple E-flat formulæ (9) the proof obbligations (10) (written in SMT-LIB2 format), exploiting the heuristics explained in Section 4.1. To experiment the feasibility of our approach for concrete verification problems, we also implemented a (beta) version of a tool called ArCa producing out of the specification of a parametric distributed system and of a safety-like problem, some E-flat simple formulæ whose unsatisfiability formalizes invariant-checking and bounded-model checking problems. A script executing in sequence ArCa, ArCa-Sat and z3 can then solve such problems by reporting a 'sat/unsat' answer.

A system is specified via a pair of flat (simple) formulæ $\iota(\underline{p})$ and $\tau(\underline{p}, \underline{p}')$ and a safety problem via a further formula $\upsilon(\underline{p})$ (here the $\underline{p}$ are parameters

[3] ArCa stands for *Array with Cardinalities*.

Algorithm 1 Pseudo-code for the send-receive broadcast primitive.

Initialization:
To broadcast a *(session s)* message, a correct process sends *(init, session s)* to all.
End Initialization
for each correct process:
1. **if** received *(init, session s)* from at least $f + 1$ distinct processes **or**
2. received *(echo, session s)* from any process **then**
3. accept *(session s)*;
4. send *(echo, session s)* to all;
5. **endif**

end for

and array-ids, the $\underline{p}'$ are renamed copies of the $\underline{p}$). A bounded model checking problem is the problem of checking whether the formula

$$\iota(\underline{p}_0) \wedge \tau(\underline{p}_0, \underline{p}_1) \wedge \cdots \wedge \tau(\underline{p}_n, \underline{p}_{n+1}) \wedge \upsilon(\underline{p}_{n+1})$$

is satisfiable for a fixed n. An invariant-cheking problem, given also a formula $\phi(\underline{p})$, is the problem of checking whether the three formulæ

$$\iota(\underline{p}) \wedge \neg\phi(\underline{p}), \quad \phi(\underline{p}) \wedge \tau(\underline{p}, \underline{p}') \wedge \neg\phi(\underline{p}'), \quad \phi(\underline{p}) \wedge \upsilon(\underline{p})$$

are unsatisfiable. Notice that since all our algorthms terminate and are sound and complete, the above problems are always solved by the above tool combination (if enough computation resources are available). Thus, our technique is able *both to make safety certifications and to find bugs*.

To validate our technique, in the following we describe in detail the formalization of the send-receive broadcast primitive (SRBP) in [20]. SRBP is used as a basis to synchronize clocks in systems where processes may fail in sending and/or receiving messages. Periodically, processes broadcast the virtual time to be adopted by all, as a *(session s)* message. Processes that accept this message set s as their current time. SRBP aims at guaranteeing the following properties:

Correctness: if at least $f + 1$ correct processes broadcast the message *(session s)*, all correct processes accept the message.

Unforgeability: if no correct process broadcasts *(session s)*, no correct process accepts the message.

Relay: if a correct process accepts *(session s)*, all correct processes accept it.

where $f < N/2$ is the number of processes failing during an algorithm run, with N the number of processes in the system. Algorithm 1 shows the pseudo-code.

We model SRBP as follows: $IT(x)$ is the initial state of a process x; it is s when x broadcasts a *(init, session s)* message, and 0 otherwise. $SE(x) = s$ indicates that x has broadcast its own echo. $AC(x) = s$ indicates that x has accepted *(session s)*. Let pc be the program counter, r the round number, and G a flag indicating whether one round has been executed. We indicate with $F(x) = 1$ the fact that x is faulty, and $F(x) = 0$ otherwise. Finally, $CI(x)$ and $CE(x)$ are the number of respectively inits and echoes received. In the following, $\forall x$ means $\forall x \in [0, N)$. Some sentences are conjoined to all our proof obligations, namely: $\#\{x|F(x) = 0\} + \#\{x|F(x) = 1\} = N \wedge \#\{x|F(x) = 1\} < N/2$. For the Correctness property, we write ι_c as follows:

$$
\begin{aligned}
\iota_c \quad := \quad & pc = 1 \wedge r = 0 \wedge G = 0 \wedge s \neq 0 \wedge && (11)\\
& \#\{x|IT(x) = 0\} + \#\{x|IT(x) = s\} = N \wedge && (12)\\
& \#\{x|F(x) = 0 \wedge IT(x) = s\} \geq (\#\{x|F(x) = 1\} + 1) \wedge && (13)\\
& \forall x. SE(x) = 0 \wedge AC(x) = 0 \wedge CI(x) = 0 \wedge CE(x) = 0 && (14)
\end{aligned}
$$

where we impose that the number of correct processes broadcasting the init message is at least the number of faulty processes, f, plus 1. It is worth to notice that – from the above definition – our tool produces a specification that is checked for any $N \in \mathbb{N}$ number of processes. The constraints on IT allow to verify all admissible assignments of 0 or s to the variables. Similarly for $F(x)$.

The algorithm safety is verified by checking that the bad properties cannot be reached from the initial state. For Correctness, we set $\upsilon_c := pc = 1 \wedge G = 1 \wedge \#\{x|F(x) = 0 \wedge AC(x) = 0\} > 0$, that is, Correctness is not satisfied if – after one round – some correct process exists that has yet to accept. The algorithm evolution is described by two transitions: τ_1 and τ_2. The former allows to choose the number of both inits and echoes received by each process. The latter describes the actions in Algorithm 1.

$$
\begin{aligned}
\tau_1 \quad := \quad & pc = 1 \wedge pc' = 2 \wedge r' = r \wedge G' = G \wedge s' = s \wedge \exists K1, K2, K3, K4.\\
& K1 = \#\{x|F(x) = 0 \wedge IT(x) = s\} \wedge K2 = \#\{x|F(x) = 0 \wedge SE(x) = s\} \wedge\\
& K3 = \#\{x|F(x) = 1 \wedge IT(x) = s\} \wedge K4 = \#\{x|F(x) = 1 \wedge SE(x) = s\} \wedge\\
& \forall x. F(x) = 0 \Rightarrow (CI'(x) \geq K1 \wedge CI'(x) \leq (K1 + K3) \wedge CE'(x) \geq K2 \wedge\\
& CE'(x) \leq (K2 + K4)) \wedge\\
& \forall x. F(x) = 1 \Rightarrow (CI'(x) \geq 0 \wedge CI'(x) \leq (K1 + K3) \wedge CE'(x) \geq 0 \wedge\\
& CE'(x) \leq (K2 + K4)) \wedge\\
& \forall x. IT'(x) = IT(x) \wedge SE'(x) = SE(x) \wedge AC'(x) = AC(x)\\
\tau_2 \quad := \quad & pc = 2 \wedge pc' = 1 \wedge r' = (r + 1) \wedge s' = s \wedge G' = 1 \wedge\\
& \forall x. (CI(x) \geq \#\{x|F(x) = 1\} + 1 \Rightarrow SE'(x) = s \wedge AC'(x) = s) \wedge\\
& \forall x. (CI(x) < \#\{x|F(x) = 1\} + 1 \wedge CE(x) \geq 1 \Rightarrow SE'(x) = s \wedge AC'(x) = s) \wedge\\
& \forall x. (CI(x) < \#\{x|F(x) = 1\} + 1 \wedge CE(x) < 1 \Rightarrow SE'(x) = 0 \wedge AC'(x) = 0) \wedge\\
& \forall x. IT'(x) = IT(x) \wedge CI'(x) = CI(x) \wedge CE'(x) = CE(x)
\end{aligned}
$$

The same two transitions are used to verify both the Unforgeability and the Relay properties, for which however we have to change the initial and

Algorithm	Property	Condition	Problem	Outcome	Time (s.)
SRBP [20]	Correctness	$\geq (f+1)$ init's	BMC	safe	0.82
SRBP [20]	Correctness	$\leq f$ init's	BMC	unsafe	2.21
SRBP [20]	Unforgeability	$\geq (f+1)$ init's	BMC	safe	0.85
SRBP [20]	Relay	$\geq (f+1)$ init's	BMC	safe	1.93
BBP [21]	Correctness	$N > 3f$	BMC	safe	6.17
BBP [21]	Unforgeability	$N > 3f$	BMC	safe	0.25
BBP [21]	Unforgeability	$N \geq 3f$	BMC	unsafe	0.25
BBP [21]	Relay	$N > 3f$	BMC	safe	1.01
OT [3]	Agreement	threshold $> 2N/3$	IC	safe	4.20
OT [3]	Agreement	threshold $> 2N/3$	BMC	safe	278.95
OT [3]	Agreement	threshold $\leq 2N/3$	BMC	unsafe	17.75
OT [3]	Irrevocability	threshold $> 2N/3$	BMC	safe	8.72
OT [3]	Irrevocability	threshold $\leq 2N/3$	BMC	unsafe	9.51
OT [3]	Weak Validity	threshold $> 2N/3$	BMC	safe	0.45
OT [3]	Weak Validity	threshold $\leq 2N/3$	BMC	unsafe	0.59
UV [4]	Agreement	$\mathcal{P}_{nosplit}$ violated	BMC	unsafe	4.18
UV [4]	Irrevocability	$\mathcal{P}_{nosplit}$ violated	BMC	unsafe	2.04
UV [4]	Integrity	-	BMC	safe	1.02
$\mathrm{U}_{T,E,\alpha}$ [2]	Integrity	$\alpha = 0 \wedge \mathcal{P}_{safe}$	BMC	safe	1.16
$\mathrm{U}_{T,E,\alpha}$ [2]	Integrity	$\alpha = 0 \wedge \neg\mathcal{P}_{safe}$	BMC	unsafe	0.83
$\mathrm{U}_{T,E,\alpha}$ [2]	Integrity	$\alpha = 1 \wedge \mathcal{P}_{safe}$	BMC	safe	5.20
$\mathrm{U}_{T,E,\alpha}$ [2]	Integrity	$\alpha = 1 \wedge \neg\mathcal{P}_{safe}$	BMC	unsafe	4.93
$\mathrm{U}_{T,E,\alpha}$ [2]	Agreement	$\alpha = 0 \wedge \mathcal{P}_{safe}$	BMC	safe	59.80
$\mathrm{U}_{T,E,\alpha}$ [2]	Agreement	$\alpha = 0 \wedge \neg\mathcal{P}_{safe}$	BMC	unsafe	7.78
$\mathrm{U}_{T,E,\alpha}$ [2]	Agreement	$\alpha = 1 \wedge \mathcal{P}_{safe}$	BMC	safe	179.67
$\mathrm{U}_{T,E,\alpha}$ [2]	Agreement	$\alpha = 1 \wedge \neg\mathcal{P}_{safe}$	BMC	unsafe	31.94
MESI [16]	cache coherence	-	IC	safe	0.11
MOESI [19]	cache coherence	-	IC	safe	0.08
Dekker [5]	mutual exclusion	-	IC	safe	2.05

Table 1: Evaluated algorithms and experimental results.

final formula. For Unforgeability, (13) in ι changes as $... \wedge \#\{x|F(x) = 0 \wedge IT(x) = 0\} = \#\{x|F(x) = 0\} \wedge ...$; while $\upsilon_u := pc = 1 \wedge G = 1 \wedge \#\{x|F(x) = 0 \wedge AC(x) = s\} > 0$. In ι_u we say that all non-faulty processes have $IT(x) = 0$. Unforgeability is not satisfied if some correct process accepts. For Relay, we use:

$$\begin{aligned}\iota_r \quad := \quad & pc = 1 \wedge r = 0 \wedge s \neq 0 \wedge G = 0 \wedge \\ & \#\{x|F(x) = 0 \wedge AC(x) = s \wedge SE(x) = s\} = 1 \wedge \\ & \#\{x|AC(x) = 0 \wedge SE(x) = 0\} = (N-1) \wedge \#\{x|AC(x) = s \wedge SE(x) = s\} = 1 \wedge \\ & \forall x. IT(x) = 0 \wedge CI(x) = 0 \wedge CE(x) = 0\end{aligned}$$

while $\upsilon_r = \upsilon_c$. In this case, we start the system in the worst condition: by the hypothesis, we just know that one correct process has accepted. Upon acceptance, by the pseudo-code, it must have sent an echo. All the other processes are initialized in an idle state. We also produce an unsafe model of Correctness: we modify ι_c by imposing that just f correct processes broadcast the init message.

In Table 1, we report the results of validating these and other models with our tool. In the first column, the considered algorithm is indicated. The second column indicates the property to be verified; the third column reports the conditions of verification. In the fourth column, we indicate whether we

consider either a bounded model checking (BMC) or an invariant-checking (IC) problem. The fifth column supplies the obtained results. The sixth column shows the time jointly spent by ArCa, ArCa-Sat and z3 for the verification, considering for BMC the sum of the times spent for every traces of length up to 10. We used a PC equipped with Intel Core i7 processor and operating system Linux Ubuntu 14.04 64 bits. We focused on BMC problems as they produce longer formulas thus stressing more the tools. Specifically, following the example above, we modeled:

- the byzantine broadcast primitive (BBP) [21] used to simulate authenticated broadcast in the presence of malicious failures of the processes,
- the one-third algorithm (OT) [3] for consensus in the presence of benign transmission failures,
- the Uniform Voting (UV) algorithm [4] for consensus in the presence of benign transmission failures,
- the $\mathrm{U}_{T,E,\alpha}$ algorithm [2] for consensus in the presence of malicious transmission failures,
- the MESI [16] and MOESI [19] algorithms for cache coherence,
- the Dekker's algorithm [5] for mutual exclusion.

All the models, together with our tools to verify them, are available at `http://users.mat.unimi.it/users/ghilardi/arca`.

As far as the processing times are concerned, we observed that on average z3 accounts for around 68% of the processing time, while ArCa and ArCa-Sat together account for the remaining 32%. Indeed, the SMT tests performed by ArCa-Sat are lightweight – as they only prune assignments – yet effective, as they succeed in reducing the number of assignments of at least one order of magnitude.

6 Conclusions, related and further work

We identified two fragments of the rich syntax of Figure 1 and we showed their decidability (for the second fragment we showed also a tight complexity bound). Since our fragments are closed under Boolean connectives, it is possible to use them not only in bounded model checking (where they can both give certifications and find bugs), but also in order to decide whether an invariant holds or not. We implemented our algorithm for the weaker fragment and used it in some experiments. As far as we know, this is the first implementation of a *complete* algorithm for a fragment of arithmetic with arrays and counting capabilities for interpreted sets.

Since one of the major intended applications concerns fault-tolerant distributed systems, we briefly review and compare here some recent work in the area. Papers [12], [11], [10] represent a very interesting and effective research line, where cardinality constraints are not directly handled but abstracted away using interval abstract domains and counters. As a result, a remarkable amount of algorithms are certified, although the method might suffer of some lack of expressiveness for more complex examples.

On the contrary, paper [3] directly handles cardinality constraints for interpreted sets; nontrivial invariant properties are synthesized and checked, based on Horn constraint solving technology. At the level of decision procedures, some incomplete inference schemata are employed (completeness is nevertheless showed for array updates against difference bounds constraints).

Paper [6] introduces a very expressive logic, specifically tailored to handle consensus problems (whence the name 'consensus logic' CL). Such logic employs arrays with values into power set types, hence it is situated in a higher order logic context. Despite this, our flat fragment is not fully included into CL, because we allow arithmetic constraints on the sort of indexes and also mixed constraints between indexes and data: in fact, we have a unique sort for indexes and data, leading to the possibility of writing typically non permutation-invariant formulæ like $\sharp\{x \mid a(x) + x = N\} = z$. As pointed out in [1], this mono-sorted approach is useful in the analysis of programs, when pointers to the memory (modeled as an array) are stored into array variables. From the point of view of deduction, the paper [6] uses an incomplete algorithm in order to certify invariants. A smaller decidable fragment (identified via several syntactic restrictions) is introduced in the final part of the paper; the sketch of the decidability proof supplied for this smaller fragment uses bounds for minimal solutions of Presburger formulæ as well as Venn regions decompositions in order to build models where all nodes in the same Venn region share the same value for their function symbols.

In future, we plan to extend both our tool ArCa and our results in order to deal with more complex verification problems. Although it won't be easy to find richer fragments inheriting all the nice properties we discovered in this paper, we are confident that concrete applications will suggest viable effective extensions.

References

[1] F. Alberti, S. Ghilardi, and N. Sharygina. Decision procedures for flat array properties. In *TACAS*, pages 15–30, 2014.

[2] M. Biely, B. Charron-Bost, A. Gaillard, M. Hutle, A. Schiper, and J. Widder. Tolerating corrupted communication. In *Proc. PODC*, pages 244–253, 2007.

[3] N. Bjørner, K. von Gleissenthall, and A. Rybalchenko. Synthesizing cardinality invariants for parameterized systems. Available at `https://www7.in.tum.de/~gleissen/papers/sharpie.pdf`, 2015.

[4] B. Charron-Bost and A. Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, pages 49–71, 2009.

[5] E.W. Dijkstra. Cooperating Sequential Processes. In *Programming Languages*, Academic Press, 1968.

[6] C. Dragoi, T. Henzinger, H. Veith, J. Widder, and D. Zufferey. A logic-based framework for verifying consensus algorithms. In *Proc. of VMCAI*, 2014.

[7] C. Dragoi, T.A. Henzinger, and D. Zufferey. The need for language support for fault-tolerant distributed systems. In *Proc. of SNAPL*, 2015.

[8] F. Eisenbrand and G. Shmonin. Carathéodory bounds for integer cones. *Oper. Res. Lett.*, 34(5):564–568, 2006.

[9] J.Y. Halpern. Presburger arithmetic with unary predicates is Π_1^1 complete. *J. Symbolic Logic*, 56(2):637–642, 1991.

[10] A. John, I. Konnov, U. Schmid, H. Veith, and J. Widder. Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In *Proc. of FMCAD*, pages 201–209, Aug. 2013.

[11] I. Konnov, H. Veith, and J. Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In *Proc. of CONCUR*, LNCS, page 125140, 2014.

[12] I. Konnov, H. Veith, and J. Widder. SMT and POR beat Counter Abstraction: Parameterized Model Checking of Threshold-Based Distributed Algorithms. In *Proc. of CAV*, LNCS, 2015.

[13] V. Kuncak, H.H. Nguyen, and M. Rinard. An algorithm for deciding BAPA: Boolean Algebra with Presburger Arithmetic. In *Proc. of CADE-20*, volume 3632 of *LNCS*, July 2005.

[14] Viktor Kuncak, Huu Hai Nguyen, and Martin Rinard. Deciding Boolean Algebra with Presburger Arithmetic. *Journal of Automated Reasoning*, 36(3), 2006.

[15] V. Kunkak and M. Rinard. Towards efficient satisfiability checking for Boolean Algebras with Presburger arithmetic. In *CADE 21*, pages 215–230, 2007.

[16] M.S. Papamarcos and J.H. Patel. A low-overhead coherence solution for multiprocessors with private cache memories. In *Proc. ISCA*, page 348, 1984.

[17] R. Piskac and V. Kuncak. Decision procedures for multisets with cardinality constraints. In *Proc. of VMCAI*, LNCS, 2008.

[18] N. Schweikhart. Arithmetic, first-order logic, and counting quantifiers. *ACM TOCL*, pages 1–35, 2004.

[19] Y. Solihin. *Fundamentals of Parallel Computer Architecture Multichip and Multicore Systems*. Solihin Publishing & Consulting LLC, 2008.

[20] T.K. Srikanth and S. Toueg. Optimal clock synchronization. *Journal of the ACM*, 34(3):626–645, 1987.

[21] T.K. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[22] K. Yessenov, R. Piskac, and V. Kuncak. Collections, cardinalities, and relations. In *Proc. of VMCAI*, 2010.

A Counting constraints in Presburger arithmetic

We report here a proof of Theorem 1. This is not an original result and we will not try to optimize it, rather we just rewrite proofs inside our notations, trying at the same time to supply the reader some intuitive evidence about the reasons why the theorem holds.

Take a constraint formula ϕ (this is a formula built up from the grammar of Figure 1 without using array-ids). For every atom A occurring in it (i.e. for every subformula of the kind $t_1 < t_2, t_1 = t_2$ ot $t_1 \equiv_n t_2$) and for every outermost occurrence of a subterm of the kind $\sharp\{x \mid \psi\}$ in A, pick a fresh variable z and replace A in ϕ with $\exists z\,(z = \sharp\{x \mid \psi\} \wedge A')$, where A' is obtained from A replacing the occurrence of the subterm $\sharp\{x \mid \psi\}$ by z. If we call ϕ' the resulting formula, it is clear that ϕ and ϕ' are equivalent.

By repeating this procedure, we can transform any constraint formula (up to equivalence) into a constraint formula built up according to the following more restricted instructions:

(i) *arithmetic terms* are built up from numerals $0, 1, \ldots$, individual variables $x, y, z, \ldots$ and parameters $M, N, \ldots$ using $+$ and $-$;

(ii) *arithmetic atoms* are expressions of the kind $t_1 < t_2, t_1 = t_2, t_1 \equiv_n t_2$, where t_1, t_2 are arithmetic terms;

(iii) *arithmetic formulæ* are built up from arithmetic atoms using $\wedge, \neg, \exists$ (actually, $\exists$ is redundant, given that quantifier-elimination holds);

(iv) *constraint atoms* are either arithmetic atoms or expressions of the form $y = \sharp\{x \mid \alpha\}$, where α is an arithmetic formula;

(v) *constraint formulæ* are built up from constraint atoms using $\wedge, \neg, \exists$ and $y = \sharp\{x \mid -\}$ (that is, recursively: constraint atoms are constraint formulæ, if ϕ, ψ are constraint formulæ so are $\phi \wedge \psi$ and $\neg\phi$, if ϕ is a constraint formula and x, y variables, $\exists x\,\phi$ and $y = \sharp\{x \mid \phi\}$ are constraint formulæ).

Recall that we interpret $\sharp\{x \mid \alpha\}$ as the cardinality of the set formed by the x such that $0 \leq x < N$ and $\alpha(x)$ is true. Thus, if we want to translate our constraint atoms into the terminology of [18], we must translate $y = \sharp\{x \mid \alpha\}$ as $\exists^{=y} x\,(0 \leq x \wedge x < N \wedge \alpha)$ (in this sense, our formalism apparently looks slightly less expressive and the procedure below has few less cases than [18]).

It is then evident that Theorem 1 is proved once we show the following

Theorem 4 *Every constraint atom is equivalent to an arithmetic formula.*

Proof. The following *special case* of Proposition 4 is easy: if x does not occur in the arithmetic terms t_1, t_2, t_3, then the constraint atom

$$y = \sharp\{x \mid t_1 \leq x \wedge x < t_2 \wedge x \equiv_n t_3\} \tag{15}$$

is equivalent to the formula

$$\exists z \left(\begin{array}{l} \tilde{t}_1 \leq z \ \wedge \ z < \tilde{t}_2 \ \wedge \ z \equiv_n t_3 \wedge \\ \forall z' (\tilde{t}_1 \leq z \ \wedge \ z < \tilde{t}_2 \ \wedge \ z \equiv_n t_3 \ \rightarrow \ z \leq z') \wedge \\ y = \lceil \dfrac{\tilde{t}_2 - z}{n} \rceil \end{array} \right) \vee \qquad (16)$$
$$\vee \left(\begin{array}{l} \neg \exists z \, (\tilde{t}_1 \leq z \ \wedge \ z < \tilde{t}_2 \ \wedge z \equiv_n t_3) \wedge \\ y = 0 \end{array} \right)$$

wher $\tilde{t}_1$ stands for $max(0, t_1)$ and $\tilde{t}_2$ stands for $min(t_2, N)$ (notice that max and min functions are definable in Presburger arithmetic, so they can be safely used). What formula (16) says is that either there is no $z \in [\tilde{t}_1, \tilde{t}_2)$ such that $z \equiv_n t_3$ (and then $y = 0$) or there is such a z (and then, taking the minimum such z, we have that $y = \lceil \frac{\tilde{t}_2 - z}{n} \rceil$). Notice that the condition $y = \lceil \frac{\tilde{t}_2 - z}{n} \rceil$ can be expressed in Presburger arithmetic via $\bigvee_{l=0}^{n-1} (ny = l + \tilde{t}_2 - z)$.[4]

We now show how to reduce to the above special case, using the series of Lemmas of Subsection A.1 below.

Consider in fact a constraint atom $y = \sharp\{x \mid \alpha\}$; we can suppose that α is quantifier-free because Presburger arithmetic enjoys quantifier elimination. We can also eliminate negations in favour of disjunctions using the equivalences $t \neq u \leftrightarrow (t < u \vee u < t)$, and $t \not< u \leftrightarrow u \leq t$,[5] and $t \not\equiv_n u \leftrightarrow \bigvee_{l=1}^{n-1} (t \equiv_n u + l)$. Using Lemma 3 and disjunctive normal forms arising from Venn's regions analysis, we can freely assume that α is a conjunction of arithmetic atoms; atoms in which x does not occur can be eliminated using Lemma 4. By normalizing terms as linear polynomials, we can further limit to atoms of the kinds

$$kx = t, \quad t < kx, \quad kx < t, \quad kx \equiv_n t,$$

where $k, n \geq 1$ and where t is an arithmetic term in which x does not occur. By Lemma 5, we can solve the case where there are atoms of the kind $kx = t$. If there are no atoms like that, using Lemmas 11,10,6,7, we can freely assume that $k = 1$.[6]

To sum up, we are left with a constraint atom $y = \sharp\{x \mid \alpha\}$ where α is of the kind

$$\bigwedge_{i=1}^{q} t_i \leq x \ \wedge \ \bigwedge_{j=1}^{r} x < u_j \ \wedge \ \bigwedge_{h=1}^{s} x \equiv_{n_s} v_h \quad .$$

(we used obvious equivalences like $t < x \leftrightarrow t + 1 \leq x$). We can now reduce to $q = 1$ and $r = 1$ by making a disjunctive guess for determining the biggest

[4] We use obvious abbreviations like $ny = y + \cdots + y$ (n-times).

[5] Here $u \leq t$ stands for $u = t \vee u < t$.

[6] In case an inconsistent condition arises according to Lemma 7(i), the constraint atom is replaced by $y = 0$.

t_i and the lowest u_j (Lemma 4 is then used to eliminate atoms where x does not occur).[7] By Lemma 8 and Lemmas 3,4, we can also freely assume that $s = 1$. Thus we finally end up in the special case above. $\dashv$

A.1 Ingredient Lemmas

We collect here the facts we used in the above proof (they are all almost obvious).

Lemma 3 *If the formulae α_i are pairwise inconsistent, then $y = \sharp\{x \mid \bigvee_{i=1}^{n} \alpha_i\}$ is equivalent to*

$$\exists z_1 \cdots \exists z_n \, (\bigwedge_{i=1}^{n} z_i = \sharp\{x \mid \alpha_i\} \wedge y = \sum_{i=1}^{n} z_i) \quad .$$

Lemma 4 *If x does not occur in β, then $y = \sharp\{x \mid \alpha \ \wedge \ \beta\}$ is equivalent to*

$$(\neg\beta \wedge y = 0) \ \vee \ (\beta \ \wedge \ y = \sharp\{x \mid \alpha\}) \quad .$$

Lemma 5 *If x does not occur in t, then $y = \sharp\{x \mid \alpha \wedge kx = t\}$ is equivalent to*

$$(y = 1 \wedge \exists x \ (0 \leq x < N \wedge \alpha \wedge kx = t)) \vee (y = 0 \wedge \neg \exists x \ (0 \leq x < N \wedge \alpha \wedge kx = t)) \quad .$$

Lemma 6 *Let t be an arithmetic term where x does not occur; then the constraint atom $y = \sharp\{x \mid \alpha \ \wedge \ t \equiv_n kx\}$ is equivalent to*

$$\bigvee_{l=0}^{n-1} (t \equiv_n l \wedge y = \sharp\{x \mid \alpha \ \wedge \ l \equiv_n kx\})$$

Next two lemmas just report basic arithmetic facts:

Lemma 7 *For $n, l \geq 1$ and $k \geq 0$, let $g := gcd(l, n)$; consider the linear congruence $lx \equiv_n k$; we have that*

(i) *if $g \mid k$ does not hold, then $lx \equiv_n k$ is inconsistent (i.e. it does not have a solution);*

(ii) *if $g \mid k$ holds, then one can compute n', k' such that $lx \equiv_n k$ is equivalent to $x \equiv_{n'} k'$.*

Proof. Item (i) is obvious, because, if $lx \equiv_n k$ has a solution, then we have $lx - qn = k$ for some q. Suppose now that $g \mid k$ holds and let $l' := l/g$, $n' := n/g$, $\tilde{k} := k/g$. Since $gcd(n', l') = 1$ and since gcd's can be expressed as linear combinations, there exists l'' such that $l'l'' \equiv_{n'} 1$. But then $lx \equiv_n k$ is the same as $l'gx \equiv_{gn'} \tilde{k}g$ which is equivalent to $l'x \equiv_{n'} \tilde{k}$, i.e. to $x \equiv_{n'} k'$, for $k' := l''\tilde{k}$. $\dashv$

[7] We assume that $r, q \geq 1$ because $0 \leq x$ and $x < N$ must be included among the conjuncts of α.

Lemma 8 *Let $k_1, \ldots, k_m \in \mathbb{Z}$, $n_1, \ldots, n_m \geq 1$ and $l := lcm(k_1, \ldots, k_m)$; then $x \equiv_{n_1} k_1 \ \wedge \cdots \wedge \ x \equiv_{n_m} k_m$ is equivalent to*

$$\bigvee_{r=0}^{l-1} (x \equiv_l r \ \wedge \ r \equiv_{n_1} k_1 \ \wedge \cdots \wedge \ r \equiv_{n_m} k_m) \ .$$

Lemma 9 *Let t be an arithmetic term, $n \geq 1, q \in \mathbb{Z}$ and $l \in \{0, \ldots, n-1\}$; the following implications are valid*

$$\begin{aligned} t - 1 = nq + l \ &\rightarrow \ \forall z \, (nz < t \leftrightarrow z < q+1) \\ t + 1 = nq + l \ &\rightarrow \ \forall z \, (t < nz \leftrightarrow q-1 < z) \end{aligned}$$

Proof. We prove the validity of the first implication (the second is shown in an analogous way). Assume $t - 1 = nq + l$; then $nz < t$ is equivalent to $n(z - q) \leq l$. This is the same as $z - q \leq 0$ (i.e. to $z < q + 1$, as wanted), because otherwise we have $z - q \geq 1$ which implies $l \geq n(z - q) \geq n$, absurd. $\dashv$

Lemma 10 *Let t be an arithmetic term where x does not occur; then the constraint atom $y = \sharp\{x \mid \alpha \ \wedge \ nx < t\}$ is equivalent to*

$$\bigvee_{l=0}^{n-1} \exists q \, (t - 1 = nq + l \ \wedge \ y = \sharp\{x \mid \alpha \ \wedge \ x < q + 1\}) \ .$$

Proof. By the existence of quotients and remainders, $y = \sharp\{x \mid \alpha \ \wedge \ nx < t\}$ is equivalent to $\bigvee_{l=0}^{n-1} \exists q \, (t - 1 = nq + l) \ \wedge \ y = \sharp\{x \mid \alpha \ \wedge \ nx < t\}$, i.e. to

$$\bigvee_{l=0}^{n-1} \exists q \, (t - 1 = nq + l \ \wedge \ y = \sharp\{x \mid \alpha \ \wedge \ nx < t\}) \ .$$

Now it is sufficient to apply the previous lemma. $\dashv$

Lemma 11 *Let t be an arithmetic term where x does not occur; then the constraint atom $y = \sharp\{x \mid \alpha \ \wedge \ t < nx\}$ is equivalent to*

$$\bigvee_{l=0}^{n-1} \exists q \, (t + 1 = nq + l \ \wedge \ y = \sharp\{x \mid \alpha \ \wedge \ q - 1 < x\}) \ .$$

Proof. The same as for the previous lemma. $\dashv$