Journal of Automated Reasoning
https://doi.org/10.1007/s10817-020-09578-5

Higher-Order Quantifier Elimination, Counter Simulations and Fault-Tolerant Systems

Silvio Ghilardi[1] · Elena Pagani[2]

Received: 3 January 2019 / Accepted: 5 August 2020

Abstract
We develop *quantifier elimination procedures* for fragments of higher order logic arising from the formalization of distributed systems (especially of fault-tolerant ones). Such procedures can be used in symbolic manipulations like the computation of pre/post images and of projections. We show in particular that our procedures are quite effective in producing *counter abstractions* that can be model-checked using standard SMT technology. In fact, very often in the current literature verification tasks for distributed systems are accomplished via counter abstractions. Such abstractions can sometimes be justified via simulations and bisimulations. In this work, we supply logical foundations to this practice, by our technique for second order quantifier elimination. We *implemented* our procedure for a simplified (but still expressive) subfragment and we showed that our method is able to successfully handle verification benchmarks from various sources with interesting performances.

Keywords 2nd order quantifier elimination · Satisfiability Modulo theories · Verification of parameterized distributed systems · Counter abstractions

1 Introduction

There is an increasing interest concerning algorithmic methods (and, in particular, quantifier elimination methods) applying to second order logic, as witnessed by recent dedicated workshops [40]. Quoting from the textbook [25] "In recent years there has been an increasing use of logical methods and significant new developments have been spawned in several areas of computer science, ranging from artificial intelligence and software engineering to agent-

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10817-020-09578-5) contains supplementary material, which is available to authorized users.

✉ Elena Pagani
elena.pagani@unimi.it

Silvio Ghilardi
silvio.ghilardi@unimi.it

1 Università degli Studi di Milano, Via C. Saldini 50, 20133 Milan, Italy

2 Università degli Studi di Milano, Via G. Celoria 18, 20133 Milan, Italy

Published online: 29 August 2020

based systems and the semantic web. In the investigation and application of logical methods there is a tension between: (i) the need for a representational language strong enough to express domain knowledge of a particular application, and the need for a logical formalism general enough to unify several reasoning facilities relevant to the application, on the one hand, and (ii) the need to enable computationally feasible reasoning facilities, on the other hand. Second-order logics are very expressive and allow us to represent domain knowledge with ease, but there is a high price to pay for the expressiveness. Most second-order logics are incomplete and highly undecidable. It is the quantifiers which bind relation symbols that make second-order logics computationally unfriendly. It is therefore desirable to eliminate these second-order quantifiers, when this is mathematically possible; and often it is."

However, most known applications of second-order quantifier elimination concern modal-like logics or knowledge representation area (see again [25]), with limited - if not negligible at all - impact on other areas of computer science, like formal methods. In this paper, we are partially filling this gap, by developing *specialized second order elimination techniques applying to the verification of distributed (especially fault-tolerant) algorithms*. In designing the fragments of second order logic to which our algorithms apply, we are strictly guided by our intended main applications, although we feel that our contribution could be interesting also in a general logical context.

1.1 The Challenge of Verification of Distributed Systems

The automated, formal verification of distributed algorithms is a crucial, although challenging, task. The processes executing these algorithms communicate with one another, their actions depend on the messages received, and their number is arbitrary. These characteristics are captured by so called reactive parameterized systems. The task of validating or refuting properties of these systems is daunting, due to the difficulty of limiting the possible evolutions, thus having to deal with genuinely infinite-state systems.

Building accurate *declarative* models of these systems requires powerful formalisms, involving arrays [28], [29] and, in the fault-tolerant case, also some fragment of higher-order logic [21], [4] (this is needed in order to have some form of comprehension to play with cardinalities of definable sets). On the other hand, for a long time, it has been observed that *counter systems* [18,19,22] can be sufficient to specify many problems (like cache coherence or broadcast protocols) in the distributed algorithms area. Recently, counter abstractions have been effectively used also in the verification of fault-tolerant distributed protocols [3,34,35,38]. It should be noticed that, unlike what happens in the old framework of [18,19,22], these new applications are often (although not always) based on abstractions that can only *simulate* the original algorithms and such simulation may sometimes be the result of an a-priori reasoning on the characteristics of the algorithm, embedded into the model. Despite this fact, all runs from the original specifications are represented in the simulations with counter systems (this is in fact the formal content of the notion of a 'simulation'), thus for instance safety certifications for the simulating model apply also to the original model. The advantage of this approach is that, as it is evident e.g. from the experiments in [3], *verification of counter systems is very well supported by the existing technology*. In fact, although basic problems about counter systems are themselves undecidable, the sophisticated machinery (predicate abstraction [24], IC3 [17,32], etc.) developed inside the SMT community leads to impressively performing tools like μZ [33], NUXMV [13], SEAHORN [31].... which are nowadays being used to solve many verification problems regarding counter systems.

Being conscious that building simulations requires in any case some human interaction, we tried to build in this paper a uniform framework. Our framework relies on recent powerful techniques for deciding cardinality and array constraints [4,6,43]; we shall exploit these techniques in order to obtain *quantifier elimination results in a higher order context*. Via these quantifier elimination results, we shall show how to *automatically build the best possible counter simulations* users can obtain once they fix (i) the specification of the system, (ii) possibly some helpful invariants and (iii) the counter variables involved in the projected simulation (such variables are cardinality counters for definable sets). We demonstrate the effectiveness of our approach by producing, for some common benchmarks, counter systems simulations which are effectively model-checked by current SMT-based tools.

1.2 A Toy Example

Let us begin by illustrating our methodology via a first example, the MESI protocol (the same simple technique applies to all examples from e.g. [2]). The MESI protocol is a cache coherence protocol; here we analyze the simplified version reported in the extended version of [2] (in Appendix A.3 in the supplementary material we shall make a detailed analysis of the original algorithm from [47]). We have a finite set `Proc` of `N` identical processes; each process i can take a local state $L(i)$ within the enumerated set

$$\mathtt{Data} = \{(\mathtt{m})\mathtt{odified}, (\mathtt{e})\mathtt{xclusive}, (\mathtt{s})\mathtt{hared}, (\mathtt{i})\mathtt{nvalid}\}\ .$$

Initially all processes are in state `i` and the system can evolve from L to L' according to one of the four nondeterministic rules:

$$\begin{aligned}
&(\tau_1)\ \ \exists i\ (L(i) = \mathtt{e} \wedge L'(i) = \mathtt{m} \wedge \forall j \neq i\ L'(j) = L(j))\\
&(\tau_2)\ \ \exists i\ (L(i) = \mathtt{i} \wedge L'(i) = \mathtt{s} \wedge\\
&\quad \forall j \neq i\ (L(j) = \mathtt{i} = L'(j) \vee (L(j) \neq \mathtt{i} \wedge L'(j) = \mathtt{s})))\\
&(\tau_3)\ \ \exists i\ (L(i) = \mathtt{s} \wedge L'(i) = \mathtt{e} \wedge \forall j \neq i\ L'(j) = \mathtt{i})\\
&(\tau_4)\ \ \exists i\ (L(i) = \mathtt{i} \wedge L'(i) = \mathtt{e} \wedge \forall j \neq i\ L'(j) = \mathtt{i})
\end{aligned}$$

This specification of the system involves a function variable $L : \mathtt{Proc} \longrightarrow \mathtt{Data}$ and we want to simulate it using only integer variables. To this aim, we introduce counters $z_\mathtt{i}, z_\mathtt{s}, z_\mathtt{e}, z_\mathtt{m}$ for the definable sets

$$\{i \mid L(i) = \mathtt{i}\},\ \{i \mid L(i) = \mathtt{s}\},\ \{i \mid L(i) = \mathtt{e}\},\ \{i \mid L(i) = \mathtt{m}\},$$

respectively. In other words, the formulæ $(\tau_1) - (\tau_4)$ are modified by adding to them the 8 equations below as further conjuncts

$$\begin{aligned}
z_\mathtt{i} = \sharp\{i \mid L(i) = \mathtt{i}\}, \quad z_\mathtt{s} = \sharp\{i \mid L(i) = \mathtt{s}\},\\
z_\mathtt{e} = \sharp\{i \mid L(i) = \mathtt{e}\}\ , \quad z_\mathtt{m} = \sharp\{i \mid L(i) = \mathtt{m}\},
\end{aligned} \tag{1}$$

$$\begin{aligned}
z'_\mathtt{i} = \sharp\{i \mid L'(i) = \mathtt{i}\}, \quad z'_\mathtt{s} = \sharp\{i \mid L'(i) = \mathtt{s}\},\\
z'_\mathtt{e} = \sharp\{i \mid L'(i) = \mathtt{e}\}, \quad z'_\mathtt{m} = \sharp\{i \mid L'(i) = \mathtt{m}\}
\end{aligned} \tag{2}$$

For $i = 1, \ldots, 4$, we let (τ_i^+) be the conjunction of (τ_i) with the 8 equalities (1)–(2) above. A similar transformation is done for the formula (ι), expressing the system initialization, namely $\forall i\ L(i) = \mathtt{i}$: this is modified to (ι^+) by conjoining to it the first 4 equations above (namely the equations (1)).

Now, observe that the safety property we are interested in, namely that processes in state $\mathtt{m}$ cannot coexist with processes in state $\mathtt{s}$, can be expressed in terms of arithmetic properties of our counters as $z_{\mathtt{m}} > 0 \to z_{\mathtt{s}} = 0$. Thus, from an observational point of view, only counter arithmetics is relevant.

How to approximate the whole specification of the system using just the above counters? The idea is to *existentially quantify and then eliminate the higher order variable* L from the formulæ (ι^+), $(\tau_1^+) - (\tau_4^+)$: we shall see that this is possible in this and in many cases. In our case, after applying the quantifier elimination procedure, we get

$$\begin{array}{ll}
(\tilde{\iota}) & z_{\mathtt{i}} = \mathtt{N} \land z_{\mathtt{s}} = 0 \land z_{\mathtt{e}} = 0 \land z_{\mathtt{m}} = 0 \\
(\tilde{\tau}_1) & z_{\mathtt{e}} > 0 \land z'_{\mathtt{e}} = z_{\mathtt{e}} - 1 \land z'_{\mathtt{m}} = z_{\mathtt{m}} + 1 \land z'_{\mathtt{i}} = z_{\mathtt{i}} \land z'_{\mathtt{s}} = z_{\mathtt{s}} \\
(\tilde{\tau}_2) & z_{\mathtt{i}} > 0 \land z'_{\mathtt{i}} = z_{\mathtt{i}} - 1 \land z'_{\mathtt{m}} = 0 \land z'_{\mathtt{e}} = 0 \land z'_{\mathtt{s}} = z_{\mathtt{s}} + z_{\mathtt{m}} + z_{\mathtt{e}} + 1 \\
(\tilde{\tau}_3) & z_{\mathtt{s}} > 0 \land z'_{\mathtt{s}} = 0 \land z'_{\mathtt{m}} = 0 \land z'_{\mathtt{i}} = z_{\mathtt{i}} + z_{\mathtt{s}} - 1 + z_{\mathtt{e}} + z_{\mathtt{m}} \land z'_{\mathtt{e}} = 1 \\
(\tilde{\tau}_4) & z_{\mathtt{i}} > 0 \land z'_{\mathtt{s}} = 0 \land z'_{\mathtt{m}} = 0 \land z'_{\mathtt{i}} = z_{\mathtt{i}} - 1 + z_{\mathtt{s}} + z_{\mathtt{e}} + z_{\mathtt{m}} \land z'_{\mathtt{e}} = 1
\end{array}$$

In this new system, the only variables are the arithmetic variables $z_{\mathtt{i}}, z_{\mathtt{s}}, z_{\mathtt{e}}, z_{\mathtt{m}}$; we will show that the new system *simulates the old system 'in the best possible way'* (in the sense formally explained in Sect. 4.1) using the variables $z_{\mathtt{i}}, z_{\mathtt{s}}, z_{\mathtt{e}}, z_{\mathtt{m}}$.[1] Since the property to be checked is a safety property, we can hope to check it for the new (simpler) system: if we succeed, we get the desired safety certification for the original system. This is in fact what happens: an SMT-based tool like μZ or NUXMV (among others) is able to solve the new safety problem instantaneously.

1.3 Our Four-Steps Plan

It should now be clear what is our general strategy:

(1) system specifications are formulated in higher order logic, i.e. using a declarative formalism which is sufficiently expressive and close to informal specifications;
(2) counters for definable sets are added by the user to the system specification, in such a way that the observationally relevant properties can be reformulated as arithmetic properties of these counters;
(3) higher order variables are eliminated, by applying an automatic procedure;
(4) the resulting system is finally model-checked by using an SMT-based tool for counter systems.

In this plan, only steps (1)–(2) require manual intervention (we shall better discuss these steps in Sect. 7); step (3) is effective every time the syntactic restrictions for our quantifier elimination procedures are matched; step (4) is subject to two risks, namely the fact that model-checkers may not terminate on such (often undecidable) arithmetical problems and the fact that simulations may introduce spurious traces. Non-termination, giving the actual state of the art (much progress has been made both at the theoretical and at the practical level) is less frequent than one can imagine. There are positive theoretical results: besides classical achievements [1,28] showing that backward search terminates for coverability problems whenever system states carry a wqo ordering, additional recent results [39] show that the system diameter may be (unexpectedly!) finite, thus reducing search to bounded model checking. Concerning the second risk, notice that if spurious traces arise, they can be recognized because SMT-tools supply concrete numerical values for counterexamples; then,

[1] Actually, this example is quite simple and the counters simulation is in fact a bisimulation.

one can try to go back to step (2) and to refine the abstraction by adding more counters for definable sets. In fact, the design of a good counter abstraction for a specific property to be verified requires some manual ingenuity, because it is true that our results guarantee the existence of a best counter abstraction once the integer variables are fixed, but one can get better and better counter abstractions by adding further integer variables expressing the cardinality of definable finite sets. One should not however exceed with the number of these variables: adding a full set of counters for e.g. n Boolean flags requires an exponential number of counters, which may be unfeasible and cause SMT-based model checkers to get in trouble in the final verification phase. In principle, it might also be the case that no invariant involving just arithmetic counters exists (in which case all attempts relying on counter abstractions are bound to fail), but both theoretical results and practical experiments show that very often counter abstractions are successful.

1.4 Structure of the Paper

The paper is structured as follows: in Sect. 2 we supply syntactic background and in Sect. 3 we state and prove our quantifier elimination results. In Sect. 4 we introduce our formalism for system specifications and show how quantifier elimination can be used to compute arithmetic projections (i.e. best counter simulations). Section 5 shows how a restricted (more tractable) format for system specifications can be handled inside our tool ArcaSim; Sect. 6 describes the set of benchmarks used in our experiments and report ArcaSim performances. Finally, in Sect. 7 we discuss related and future work. In the electronic Appendix A in the supplementary material we run three representative examples in full detail; the electronic Appendix is available on the publisher website. Preliminary versions of the material included in this paper were presented in the Workshops [26,27].

2 Higher Order Logic and Flat Constraints

In order to have enough expressive power, we use higher order logic, more specifically *Church's type theory* (see e.g. [7] for an introduction to the subject).[2] It should be noticed, however, that our primary aim is *to supply a framework for model-checking and not to build a deductive system*. Thus we shall introduce below only suitable languages (via higher order signatures) and a semantics for such languages - such semantics can be specified e.g. inside any classical foundational system for set theory. In addition, as typical for model-checking, we want to constrain our semantics so that certain sorts have a fixed meaning: the primitive sort $\mathbb{Z}$ has to be interpreted as the (standard) set of integers, the sort Ω has to be interpreted as the set of truth values $\{\mathtt{tt}, \mathtt{ff}\}$; moreover, some primitive sorted operations like $+$, 0, S (addition, zero, successor for natural numbers) and $\wedge$, $\vee$, $\rightarrow$, $\neg$ (Boolean operations for truth values) must have their natural interpretation. Some sorts might be *enumerated*, i.e. they must be interpreted as a specific finite 'set of values' $\{\mathtt{a}_0, \dots, \mathtt{a}_\mathtt{k}\}$, where the $\mathtt{a}_\mathtt{i}$'s are mentioned among the constants of the language and are assumed to be distinct. Finally, we may ask for a primitive sort to be interpreted as a *finite set* (by abuse, we shall call such sorts *finite*): for instance, we shall constrain in this way the sort $\mathtt{Proc}$ modeling the set of processes in a distributed system. In addition, if a sort is interpreted into a finite set, we may constrain some numerical parameter (typically, the parameter we choose for this is named $\mathtt{N}$) to indicate the

[2] Some notation we use might look slightly non-standard; it is similar to the notation of [44].

cardinality of such finite set. The notion of constrained signature below incorporates all the above requirements in a general framework.

A *constrained signature* Σ consists of a set of (primitive) sorts and of a set of (primitive) sorted function symbols,[3] together with a class $\mathcal{C}_\Sigma$ of Σ-structures, called the *models* of Σ. Using primitive sorts, *types* can be built up using exponentiation (= functions type); *terms* can be built up using variables, function symbols, as well as λ-abstraction and functional application.

Remark 1 In the standard model-checking literature, $\mathcal{C}_\Sigma$ is a singleton; here we must allow *many* structures in $\mathcal{C}_\Sigma$, because our model-checking problems are *parametric*: the sort modeling the set of processes of our system specifications must be interpreted onto a finite set whose cardinality is not a priori fixed. Our definition of a 'constrained signature' is analogous to the definition of a 'theory' in SMT literature; in fact, in SMT literature, a 'theory' is just a pair given by a signature and a class of structures. When transferred to a higher order context, such definition coincides with that of a 'constrained signature' above (thus our formal framework is very similar to e.g. that of [51]).

Our constrained signatures always include the sort Ω of truth-values; terms of type Ω are called *formulae* (we use greek letters $\alpha, \beta, \dots, \phi, \psi, \dots$ for them). For a type S, the type $S \to \Omega$ is indicated as $\wp(S)$ and called the *power set* of S; if S is constrained to be interpreted as a finite set, Σ might contain a cardinality operator $\sharp : \wp(S) \longrightarrow \mathbb{Z}$, whose interpretation is assumed to be the intended one ($\sharp s$ is the number of the elements of s - as such it is always a nonnegative number). If ϕ is a formula and S a type, we use $\{x^S \mid \phi\}$ or just $\{x \mid \phi\}$ for $\lambda x^S \phi$. We assume to have binary equality predicates for each type; universal and existential quantifiers for formulæ can be introduced by standard abbreviations (see e.g. [44]). We shall use the roman letters $x, y, \dots, i, j, \dots, v, w, \dots$ for variables (of course, each variable is suitably typed, but types are left implicit if confusion does not arise). Bold letters like $\mathbf{v}$ (or underlined letters like $\underline{x}$) are used for tuples of free variables; below, we indicate with $t(\mathbf{v})$ the fact that the term t has free variables included in the list $\mathbf{v}$ (whenever this happens, we say that t is a $\mathbf{v}$-*term*, or a $\mathbf{v}$-*formula* if it has type Ω). The result of a simultaneous substitution of the tuple of variables $\mathbf{v}$ by the tuple of (type matching) terms $\underline{u}$ in t is denoted by $t(\underline{u}/\mathbf{v})$ or directly as $t(\underline{u})$.

Given a tuple of variables $\mathbf{v}$, a Σ-*interpretation* of $\mathbf{v}$ in a model $\mathcal{M} \in \mathcal{C}_\Sigma$ is a function $\mathcal{I}$ mapping each variable onto an element of the correponding type (as interpreted in $\mathcal{M}$). The evaluation of a term $t(\mathbf{v})$ according to $\mathcal{I}$ is recursively defined in the standard way and is written as $t_{\mathcal{M},\mathcal{I}}$. A Σ-formula $\phi(\mathbf{v})$ is *true* under $\mathcal{M}, \mathcal{I}$ iff it evaluates to $\mathtt{tt}$ (in this case, we may also say that $\mathbf{v}_{\mathcal{M},\mathcal{I}}$ *satisfies* ϕ); ϕ is *valid* iff it is true for all models $\mathcal{M} \in \mathcal{C}_\Sigma$ and all interpretations $\mathcal{I}$ of $\mathbf{v}$ over $\mathcal{M}$. We write $\models_\Sigma \phi$ (or just $\models \phi$) to mean that ϕ is valid and $\phi \models_\Sigma \psi$ (or just $\phi \models \psi$) to mean that $\phi \to \psi$ is valid; we say that ϕ and ψ are Σ-*equivalent* (or just equivalent) iff $\phi \leftrightarrow \psi$ is valid.

2.1 Flat Cardinality Constraints

Let us fix a constrained signature Σ for the remaining part of the paper. Such Σ should be adequate for modeling parameterized systems, hence we assume that Σ consists of:

(i) the integer sort $\mathbb{Z}$, together with some parameters (i.e. free individual constants) as well as all operations and predicates of linear arithmetic (namely, $0, 1, +, -, =, <, \equiv_n$);

[3] These include 0-ary function symbols, called constants; constants of sort $\mathbb{Z}$ will be called (arithmetic) *parameters*.

(ii) the enumerated truth value sort Ω, with the constants `tt`, `ff` and the Boolean operations on them;

(iii) a finite sort `Proc`, whose cardinality is constrained to be equal to the arithmetic parameter `N` (in the applications, this sort is used to represent the processes acting in our distributed systems);

(iv) a further sort `Data`, with appropriate operations, modeling local data; we assume that (a) *first-order quantifier elimination* holds for `Data`, meaning that all first-order formulæ built up from `Data`-atoms (i.e. from variables of type `Data` using operations and predicates relative to the sort `Data`) are equivalent to quantifier-free ones; (b) *ground* (i.e. variable-free) `Data`-atoms are equivalent to $\bot$ or to $\top$.

In principle, we could consider having finitely many signatures for data instead of just one, but this generalization is only apparent because one can use product sorts and recover component sorts via suitable pairing and projection operations.

If `Data` is an enumerated sort, we call Σ *finitary*; the subsignature Σ_0 of Σ obtained by restricting to sorts and operations in (i)–(ii) is called the *arithmetic* subsignature of Σ.

In the syntactic definitions below, we freely take inspiration from [4], however the present framework is greatly simplified because we do not view `Proc` as a subsort of $\mathbb{Z}$, like in [4]; in addition, notice that Σ does not contain operations or relation symbols specific to the sort `Proc` (apart from equality) - this restriction reduces terms of sort `Proc` to just variables.

Below, besides *integer* variables (namely variables of sort $\mathbb{Z}$), *data* variables (namely variables of sort `Data`) and *index* variables (namely variables of sort `Proc`), we use two other kinds of variables, that we call *array-ids* and *matrix-ids*. An array-id is a variable of type `Proc` $\to$ `Data` or of type `Proc` $\to \mathbb{Z}$ and a matrix-id is a variable of type `Proc` $\to$ (`Proc` $\to$ `Data`) or of type `Proc` $\to$ (`Proc` $\to \mathbb{Z}$). Array-ids and matrix-ids of codomain sort $\mathbb{Z}$ are called *arithmetical* array-ids or matrix-ids; if `Data` is enumerated, array-ids and matrix-ids of codomain sort `Data` are called *finitary*. If M is a matrix-id and i, y are index variables, we may write $M_i(y)$ or $M(i, y)$ instead of $M(i)(y)$.

Let us now introduce some useful classes of formulæ.

- *Open formulæ*: these are built up from atomic formulæ containing arithmetic parameters and the above mentioned variables, using Boolean connectives only (no binders, i.e. no λ-abstractors and no quantifiers).
- *1-Flat formulæ*: these are formulæ of the kind $\phi(\sharp\{x \mid \psi_1\}/z_1, \ldots, \sharp\{x \mid \psi_n\}/z_n)$, where $\phi(z_1, \ldots, z_n)$, $\psi_1, \ldots, \psi_n$ are open and x is a variable of type `Proc`.
- Given an index variable i, a formula ϕ is said to be *i-uniform* with respect to a matrix-id M (resp. an array-id a) iff i is not used as a bounded variable in ϕ and the only terms occurring in ϕ containing an occurrence of M (resp. of a) are of the kind $M_i(y)$ (resp. $a(i)$) for a variable y.

Notice that, some quantified formulæ can be rewritten as 1-flat formulæ: for instance $\forall x\ (a(x) = c \to b(x) = d)$ is the same as $\sharp\{x \mid a(x) = c \to b(x) = d\} =$ `N`, and similarly $\exists x\ (a(x) = c)$ can be re-written as $\sharp\{x \mid a(x) = c\} > 0$. Such rewriting however is not possible for nested quantifiers: $\forall i\ \sharp\{x \mid a(x) = b(i)\} > 0$ (semantically equivalent to $\forall i\ \exists x\ a(x) = b(i)$) is not 1-flat because it cannot be obtained by replacing terms of the kind $\{x \mid \psi_j\}$ (with open ψ_j) for free arithmetic variables inside an open formula ϕ.

Remark 2 1-Flat formulæ of this paper are slightly different from the flat formulæ of [4,5] (they roughly correspond to the flat formulæ of degree 1 of [5]); the definition here is not recursive and is simplified by the fact that we do not have nonvariable terms of type `Proc`; on the other hand, we allow matrix-ids to occur in our formulæ, whereas the syntax of [4,5] is restricted to array-ids.

Remark 3 In the applications, we typically use matrix-ids $M(i, x)$ in the finitary case where `Data` is the set of Boolean truth values: $M(i, x)$ asserts for instance that x sent a message of a certain type to i. This allows to count the number of such messages received by i via the term $\sharp\{x \mid M(i, x)\}$ (according to our notational conventions, this is the term $\sharp\{x \mid M_i(x)\}$). The definition of a i-uniform formula is meant precisely to make such terms available: they are used in order to formalize standard benchmarks like the byzantine broadcast primitive protocol (see Appendix A.1 in the supplementary material).

3 Quantifier Elimination

In this technical section we state and prove the quantifier elimination results we need. Let us fix a constrained signature Σ like in Sect. 2.1. We first investigate in a closer way our open formulæ. Notice first that if an open formula is *pure* (i.e. it does not contain array-ids or matrix-ids), then it is a Boolean combination of arithmetic, index or data atoms, where:

- arithmetic atoms are built up from variables of sort $\mathbb{Z}$, parameters (i.e free constants of sort $\mathbb{Z}$), by using $=, <, \equiv_n$ as predicates and $+, -, 0, 1$ as function symbols;
- index atoms are of the kind $i = j$, where i, j are variables of sort `Proc` (we do not consider further operations and predicates for this sort - apart from equality - in this paper);
- data atoms are built up from variables of sort `Data` by applying some specific set of predicates and operations (predicates include equality, all arguments of such predicates and operations are of type `Data`).

By assumption (see Sect. 2.1), quantifier elimination holds for first-order `Data`-formulæ, but this result extends very easily to all pure first-order formulæ. We state this formally as a Lemma:

Lemma 1 *Any pure first-order formula is equivalent to an open pure first-order formula.*

Proof Using prenex formula transformations, it is sufficient to show how to eliminate a quantifier $\exists x\, \alpha$, where α is open and pure. Actually, using disjunctive normal forms, we can assume that α is a conjunction of literals. Pushing the existential quantifier inside, we can assume that such literals are all arithmetic, all index or all data literals, depending on the sort of x. The case of arithmetic literals is covered by Presburger quantifier elimination [49], whereas the case of data literals is covered by our assumption. It remains to consider the case of index literals; excluding trivial cases where the existential quantifier is redundant or eliminable by substitution, we are left with the case where α is $x \neq y_1 \wedge \cdots \wedge x \neq y_n$. By introducing a disjunction of cases (and by distributing the existential quantifier over such disjunction and removing redundant variables), we reduce to a disjunction of formulæ of the kind

$$\exists x\ (x \neq y_1 \wedge \cdots \wedge x \neq y_{n'} \wedge \bigwedge_{i \neq j} y_i \neq y_j)$$

The latter is equivalent to $\mathtt{N} > \bar{n}' \wedge \bigwedge_{i \neq j} y_i \neq y_j$, where $\bar{n}'$ is $1 + \cdots + 1$ (n'-times). □

In case array-ids and matrix-ids do not occur, 1-flat formulæ can also be trivialized:[4]

[4] If the sort `Proc` is identified with a definable finite subset of $\mathbb{Z}$, the result still holds but is much less trivial: to get it, one must apply results from Presburger arithmetic with counting quantifiers [52].

Lemma 2 *A 1-flat formula without array-ids and matrix-ids is equivalent to a pure formula.*

Proof Let us eliminate subterms t of the kind $\sharp\{x \mid \alpha\}$ (with pure α) inside a pure formula ϕ. We can first remove from α arithmetic and data atoms, as well as index atoms not containing x, by the following equivalence (let A be the atom to be removed):

$$\phi \leftrightarrow ([A \wedge \phi(\top/A)] \vee [\neg A \wedge \phi(\bot/A)]) \ .$$

By Venn regions decomposition, we can assume that α is a conjunction of literals: in fact, if $\{\alpha_1, \ldots, \alpha_k\}$ is a Venn regions decomposition of α, then $\sharp\{x \mid \alpha\}$ is sematically equal to $\sum_{j=1}^{k} \sharp\{x \mid \alpha_j\}$. In addition, if t is of the kind $\sharp\{x \mid x = i \wedge \alpha\}$, we can remove it using the equivalence:

$$\phi \leftrightarrow ([\alpha(i/x) \wedge \phi(1/t)] \vee [\neg\alpha(i/x) \wedge \phi(0/t)]) \ .$$

Thus we are left only with the case in which t is $\sharp\{x \mid \bigwedge_{s=1}^{n} x \neq i_s\}$; we can also assume that ϕ entails $\bigwedge_{s \neq s'} i_s \neq i_{s'}$ (otherwise we can force this by making ϕ a disjunction of case distinctions). Then we can remove t using

$$\phi \leftrightarrow (\mathtt{N} \geq \bar{n} \wedge \phi(\mathtt{N} - \bar{n}/t)) \vee (\mathtt{N} < \bar{n} \wedge \phi(0/t)) \ .$$

Once all t are removed (one by one), the statement is proved. □

It is now convenient to introduce a notation for open (not necessarily pure) formulæ (from now on we shall reserve the letters $\alpha, \beta, \ldots$ to first-order *pure* formulæ, to recognize them). Considering that there are no operation symbols of sort `Proc`, the only new terms that might arise in open non pure formulæ (wrt pure formulæ) are of the kind $a(i)$ or $M_i(j)$, where a is an array-id, M is a matrix-id and i, j are variables of sort `Proc`. Thus we may write an open formula ϕ as the formula obtained by replacing in a pure formula some arithmetic variables with terms of the kind $a(i)$ or $M_i(j)$. If our open ϕ does not contain matrix-ids, we can write it as

$$\alpha(\underline{z}, \underline{k}, \mathbf{a}(\underline{k})/\underline{e}, \underline{d}) \text{ or simply as } \alpha(\underline{z}, \underline{k}, \mathbf{a}(\underline{k}), \underline{d}) \tag{3}$$

where $\alpha(\underline{z}, \underline{k}, \underline{e}, \underline{d})$ is pure, $\underline{z}$ is a tuple of arithmetic variables, $\underline{k}$ is a tuple of index variables, $\underline{d}$ is a tuple of data variables, $\mathbf{a}$ is a tuple of array-ids (the $\underline{e}$ might be arithmetic or `Data`-variables depending on the types of the $\mathbf{a}$); if $\mathbf{a} = a_1, \ldots, a_n$ and $\underline{k} = k_1, \ldots, k_m$, then $\mathbf{a}(\underline{k})$ is the tuple

$$a_1(k_1), \ldots, a_1(k_m), \ldots, a_n(k_1), \ldots, a_n(k_m)$$

so that the matching tuple of arithmetic or data variables $\underline{e}$ can be indexed as $e_{11}, \ldots, e_{nm}$.

A 1-flat formula without matrix-ids is then written as

$$\alpha(\underline{z}, \underline{k}, \mathbf{a}(\underline{k}), \underline{d}, \sharp\{x \mid \beta_1(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}(\underline{k}), \underline{d})\}, \ldots, \sharp\{x \mid \beta_s(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}(\underline{k}), \underline{d})\})$$

or (with some abuse of notation) shortly as

$$\alpha(\underline{z}, \underline{k}, \mathbf{a}(\underline{k}), \underline{d}, \sharp\{x \mid \underline{\beta}(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}(\underline{k}), \underline{d})\}) \tag{4}$$

where $\underline{\beta}$ is a tuple of formulæ (we use the convention that $\sharp\{x \mid \underline{\beta}\}$ stands for the tuple of terms $\sharp\{x \mid \beta_1\}, \ldots, \sharp\{x \mid \beta_s\}$).

Displaying 1-flat formulæ with matrix-ids requires an even more complex notation, that we will not use though. These notations are apparently cumbersome but have the merit of displaying the essential information on how our formulæ are built up from pure formulæ.

Lemma 3 *If ϕ is an open formula and $\underline{d}'$ are arithmetic and data variables, then $\exists \underline{d}'\, \phi$ is equivalent to an open formula.*

Proof Let ϕ be $\alpha(\underline{z}, \underline{k}, \mathbf{a}(\underline{k}), \underline{d})$
as in (3) and let $\underline{d}'$ be data variables.[5] Suppose that $\underline{d} = \underline{d}'\underline{d}''$; then $\exists \underline{d}'\alpha(\underline{z}, \underline{k}, \underline{e}, \underline{d})$, by Lemma 1, is equivalent to a pure formula $\beta(\underline{z}, \underline{k}, \underline{e}, \underline{d}'')$, so that,
applying a substitution, $\exists \underline{d}'\alpha(\underline{z}, \underline{k}, \mathbf{a}(\underline{k})/\underline{e}, \underline{d})$ is equivalent to $\beta(\underline{z}, \underline{k}, \mathbf{a}(\underline{k})/\underline{e}, \underline{d}'')$, which is as desired. The case in which $\underline{d}$ are arithmetic variables is treated similarly. □

We now state a first quantifier elimination result (this is essentially Theorem 4 from [5], we nevertheless report the proof for the sake of completeness):

Theorem 1 *Suppose that ϕ is a 1-flat formula containing the array ids $\mathbf{a}, \mathbf{a}'$ (and not containing matrix-ids); then the formula $\exists \mathbf{a}'\, \phi$ is equivalent to a formula $\exists \underline{e}\, \psi$, where the $\underline{e}$ are arithmetic and data variables, ψ is 1-flat and contains only the array-ids $\mathbf{a}$.*

Proof We start with a formula having the form

$$\exists \mathbf{a}' \quad \alpha(\underline{z}, \underline{k}, \mathbf{a}(\underline{k}), \mathbf{a}'(\underline{k}), \underline{d}, \sharp\{x \mid \underline{\beta}(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}'(x), \mathbf{a}(\underline{k}), \mathbf{a}'(\underline{k}), \underline{d})\}) \tag{5}$$

where $\underline{k}$ are index variables, $\underline{z}$ arithmetic variables and $\underline{d}$ data variables. We can first get rid of the terms $\mathbf{a}(\underline{k}), \mathbf{a}'(\underline{k})$, as follows. Suppose that $\mathbf{a} = a_1, \ldots, a_n$, $\mathbf{a}' = a'_1, \ldots, a'_{n'}$ and $\underline{k} = k_1, \ldots, k_m$; we introduce new variables

$$\underline{e} = e_{11}, \ldots, e_{nm}, e'_{11}, \ldots, e'_{n'm}$$

and rewrite (5) as

$$\exists \underline{e}.\ \bigwedge_{i,j} a_i(k_j) = e_{ij} \wedge \exists \mathbf{a}' \left(\begin{array}{l} \bigwedge_{i,j} \sharp\{x \mid x = k_i \wedge a'_j(x) = e'_{ij}\} = 1 \wedge \\ \wedge\ \alpha(\underline{z}, \underline{k}, \underline{e}, \underline{d}, \sharp\{x \mid \underline{\beta}(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}'(x), \underline{e}, \underline{d})\}) \end{array} \right)$$

Thus it will be sufficient to eliminate the $\exists \mathbf{a}'$ from formulæ of the kind

$$\exists \mathbf{a}'\, \gamma_0(\underline{z}, \underline{k}, \underline{d}', \sharp\{x \mid \underline{\beta}(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}'(x), \underline{d}')\}) \tag{6}$$

(here the $\underline{d}'$ include the old $\underline{d}$ and the $\underline{e}$ - the latter are existentially quantified and will remain such in the final outcome).

If we suppose that $\underline{\beta}$ is $\beta_1, \ldots, \beta_t$, we can set $K := \wp(\{1, \ldots, t\})$ and introduce for every $r \in K$ a new existentially quantified arithmetic variable u_r, thus rewriting (6) as

$$\exists \underline{u}, \mathbf{a}'.\ \bigwedge_r u_r = \sharp\{x \mid \beta_r(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}'(x), \underline{d}')\} \wedge \gamma(\underline{z}, \underline{k}, \underline{d}', \underline{u}) \tag{7}$$

where $\underline{u}$ is the tuple formed by the u_r's (varying r) and β_r is the 'Venn region' $\bigwedge_{l \in r} \beta_l \wedge \bigwedge_{l \notin r} \neg\beta_l$; the formula γ is obtained from γ_0 by replacing, for all l, the term $\sharp\{x \mid \beta_l\}$ with $\sum_{l \in r} u_r$. Notice that at this point γ is pure and the new β_r's are a partition (i.e. they are mutually inconsistent and $\bigvee_r \beta_r$ is valid).

To continue, following the technique in [4], we need a further 'Venn region decomposition' δ_S. For every $S \in \wp(K)$ let $\delta_S(\underline{z}, x, \underline{k}, \mathbf{a}(x), \underline{d}')$ be the pure formula

$$\bigwedge_{r \in S} \exists y\, \beta_r(\underline{z}, x, \underline{k}, \mathbf{a}(x), y, \underline{d}') \wedge \bigwedge_{r \notin S} \neg\exists y\, \beta_r(\underline{z}, x, \underline{k}, \mathbf{a}(x), y, \underline{d}')$$

5 In (3), there are no matrix-ids; if there were also matrix-ids, then the argument would be the same (we do not insist, because we shall need the lemma only for formulæ without matrix-ids).

(here data quantifiers $\exists y$ can be eliminated using Lemma 3). We claim that the formula (7) is equivalent to the formula obtained by prefixing the existential quantifiers $\exists u_r$ (varying $r \in K$), $\exists u_S$ (varying $S \in \wp(K)$) and

$\exists u_{r,S}$ (varying $S \in \wp(K)$ and $r \in K$) to the formula

$$\begin{aligned}\bigwedge_{S \in \wp(K)} u_S = \sharp\{x \mid \delta_S(\underline{z}, x, \underline{k}, \mathbf{a}(x), \underline{d}')\} \wedge \bigwedge_{S \in \wp(K)} \left(u_S = \sum_{r \in S} u_{r,S} \right) \wedge \\ \wedge \bigwedge_{r \in K} \left(u_r = \sum_{S \in \wp(K), r \in S} u_{r,S} \right) \wedge \bigwedge_{r \in S \in \wp(K)} u_{r,S} \geq 0 \wedge \gamma(\underline{z}, \underline{k}, \underline{d}', \underline{u})\end{aligned} \tag{8}$$

Suppose that (8) *is satisfied under the assignment* $\mathcal{I}$ to the free variables occurring in it (for simplicity, we use the same name for a free variable and for the integer assigned to it by $\mathcal{I}$). Let us assume that `Proc` is interpreted (up to a finite sets bijection) as the interval $[0, \mathtt{N})$; we need to define for all $i \in [0, N)$ the tuple $\mathbf{a}'(i)$ - namely $a'_s(i)$, for all $s = 1, \dots, n'$. For every $r = 1, \dots, K$ this must be done in such a way that there are exactly u_r elements taken from $[0, N)$ satisfying $\beta_r(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}'(x), \underline{d}')$. The interval $[0, \mathtt{N})$ can be partioned by associating with each $i \in [0, \mathtt{N})$ the set $i_S = \{r \in K \mid \exists \underline{y}\ \beta_r(\underline{z}, i, \underline{k}, \mathbf{a}(i), \underline{y}, \underline{d}')$ holds under $\mathcal{I}\}$. From the fact that (8) is true, we know that for every $S \in \wp(K)$ the number of the i's such that $i_S = S$ is u_S; for every $r \in S$, pick $u_{r,S}$ among them and, for these selected i, let the s-tuple $\mathbf{a}'(i)$ be equal to an s-tuple $\underline{y}$ such that $\beta_r(\underline{z}, i, \underline{k}, \mathbf{a}(i), \underline{y}, \underline{d}')$ holds (for this tuple $\underline{y}$, since the β_r's are a partition, $\beta_h(\underline{z}, i, \underline{k}, \mathbf{a}(i), \underline{y}, \underline{d}')$ does not hold, if $h \neq r$). Since $u_S = \sum_{r \in S} u_{r,S}$ and since $\sum_S u_S$ is equal to $\mathtt{N}$ (because the formulæ $\bigwedge_{r \in S} \exists \underline{y}\ \beta_r \wedge \bigwedge_{r \notin S} \forall \underline{y} \neg \beta_r$ are a partition), the definition of the $\mathbf{a}'$ is complete. The formula (7) is true by construction.

On the other hand *suppose that the matrix of* (7) *is satisfiable* under an assignment $\mathcal{I}$; we need to find $\mathcal{I}(u_S)$, $\mathcal{I}(u_{r,S})$ (we again indicate them simply as $u_S, u_{r,S}$) so that (8) is true (the u_r are already given since (7) is true). For u_S there is no choice, since $u_S = \sharp\{x \mid \delta_S(\underline{z}, x, \underline{k}, \mathbf{a}(x), \underline{d}')\}$ must hold; for $u_{r,S}$, we take it to be the cardinality of the set of the i such that $\beta_r(\underline{z}, i, \underline{k}, \mathbf{a}(i), \mathbf{a}'(i), \underline{d}')$ holds under $\mathcal{I}$ and $S = \{h \in K \mid \exists \underline{y}\ \beta_h(\underline{z}, i, \underline{k}, \mathbf{a}(i), \underline{y}, \underline{d}')$ holds under $\mathcal{I}\}$. In this way, for every S, the equality $u_S = \sum_{r \in S} u_{r,S}$ holds and for every r, the equality $u_r = \sum_{S \in \wp(K), r \in S} u_{r,S}$ holds too. Thus the formula (8) becomes true under our extended $\mathcal{I}$. $\square$

The following Corollary follows from Theorem 1 and Lemmas 2,1:

Corollary 1 *Suppose that ϕ is a 1-flat formula containing the array ids $\mathbf{a}$ (and not containing matrix-ids); then the formula $\exists \mathbf{a}\, \phi$ is equivalent to an open pure formula.*

Notice that the above result (as it happens with all our quantifier elimination results) immediately implies that 1-flat formulæ not containing matrix-ids are decidable for satisfiability.

If the sort `Data` is enumerated and all array-ids are finitary, we can improve Corollary 1 above by including an extra quantified variable, as shown in the following Theorem (the Theorem is useful for some benchmarks, see Appendix A in the supplementary material for an example):

Theorem 2 *Let the sort* `Data` *be enumerated and let the 1-flat formula ϕ contain only the finitary array ids $\mathbf{a}$ (and no matrix-ids); then the formula*

$$\exists \mathbf{a}\, \forall i\, \exists \underline{y}\, \phi \tag{9}$$

(where i is an index variable and the $\underline{y}$ are arithmetic and data variables) is equivalent to an open pure formula.

Proof Let $\mathtt{Data}$ be enumerated as $\{\mathtt{a}_0, \dots, \mathtt{a}_\mathtt{k}\}$; let $\underline{z}$ be the arithmetic variables occurring freely in (9) and let $\underline{k} = k_1, \dots, k_n$ be the index variables occurring freely in (9) (thus i is not among the $\underline{k}$ and the $\underline{y}$ are not among the $\underline{z}$). We can assume that the $\underline{y}$ are arithmetic variables because, since $\mathtt{Data}$ is enumerated, existential data variables can be elimitated via disjunctions. For simplicity, we assume that (9) contains only one array-id, let it be a.[6]

Before working on the formula (9), it is better to make some preprocessing steps. Our final aim is to produce a formula logically equivalent to (9), which is a disjunction of existentially quantified formulæ whose matrices are pure open formulæ: in this way the extra existentially quantified variables we introduce can be eliminated in the very end using Lemma 1. We need also to introduce extra information to complete (9): this extra information is achieved by rewriting (9) as a disjunction (each disjunct formalizes a suitable guess) and by operating on each disjunct separately.

Concretely, we shall freely assume that $\forall i\, \exists \underline{y}\, \phi$ in (9) is of the kind

$$\begin{aligned} &\mathtt{Diff}(\underline{k}) \;\wedge\; \bigwedge_i (a(k_i) = \mathtt{a}_{l_i}) \;\wedge\; \bigwedge_j (u_j = \sharp\{x \mid a(x) = \mathtt{a}_j\}) \;\wedge \\ &\wedge\; \forall i\; \exists \underline{y}\; \phi'(\underline{z}, \underline{y}, i, \underline{k}, a(i), \sharp\{x \mid \underline{\beta}(\underline{z}, \underline{y}, x, i, \underline{k}, a(x), a(i))\}) \end{aligned} \tag{10}$$

where

- the formula $\mathtt{Diff}(\underline{k})$ says that the $\underline{k}$ are pairwise distinct (i.e. it is $\bigwedge_{i \neq j} k_i \neq k_j$): this can be assumed without loss of generality, because one can guess a partition (introducing a disjunction over all partitions) and make the appropriate replacements so as to keep only one representative for each equivalence class of variables;
- since $\mathtt{Data}$ is enumerated we can guess (via a disjunction) for each k_i the $\mathtt{a}_{l_i}$ which is the value of $a(k_i)$ (then, all occurrences of the term in the remaining part of the formula can be replaced by this $\mathtt{a}_{l_i}$);
- the u_j are fresh arithmetic variables indicating the cardinality of the set of indices whose a-value is $\mathtt{a}_j$ (these u_j are the extra existentially quantified variables to be eliminated in the very end by Lemma 1);
- $\underline{\beta}$ are open formulæ as displayed and ϕ' is a 1-flat formula as displayed (notice that the terms $a(k_i)$ do not occur anymore here, because we can assume that they have been replaced by the corresponding $\mathtt{a}_{l_i}$).

We now operate further transformations on the subformula $\forall i\, \exists \underline{y}\, \phi'$: we want to show that this formula is equivalent to a 1-flat formula (hence without the quantifier $\forall i$), so that the claim of the Theorem follows from an application of Corollary 1 and Lemma 1 - by these results in fact all quantified variables in (9) can be eliminated in favor of a pure open formula in which only the $\underline{k}$, $\underline{z}$ occur. When manipulating $\forall i\, \exists \underline{y}\, \phi'$ below, we assume all the information we have from (10), namely that the $\underline{k}$ are all distinct and that the values of the $a(k_i)$ are known.

As a first step, we can distinguish the case in which i is equal to some of the $\underline{k}$ from the case in which it is different from all of them; in the latter case, we can also guess the value of $a(i)$. This observation shows that $\forall i\, \phi'$ is equal to the conjunction of an open formula (expressing what happens if i is equal to any of the $\underline{k}$) with the conjunctions (varying $\mathtt{a}_\mathtt{j}$ in our enumerated data)

$$\forall i.\; \mathtt{Diff}(i, \underline{k}) \wedge a(i) = \mathtt{a}_\mathtt{j} \to \exists \underline{y}\, \phi''(\underline{z}, \underline{y}, i, \underline{k}, \sharp\{x \mid \underline{\beta}'(\underline{z}, \underline{y}, x, i, \underline{k}, a(x))\}) \tag{11}$$

[6] This is without loss of generality: since $\mathtt{Data}$ is enumerated and the $\mathbf{a}$ are finitary, one may take a product of $\mathtt{Data}$ and replace the tuple $\mathbf{a}$ with a single array with values in such a product.

where the ϕ'', $\underline{\beta}'$ are obtained from the ϕ', $\underline{\beta}$ by replacing $a(i)$ with $\mathtt{a}_\mathtt{j}$. Again, it will be sufficient to show that (11) is equivalent to an open formula.

First observe that ϕ'' is obtained from a pure formula by replacing arithmetic variables with the terms $\sharp\{x \mid \underline{\beta}'(\underline{z}, \underline{y}, x, i, \underline{k}, a(x))\}$; since equality is the only predicate of sort $\mathtt{Proc}$ (and there are no function symbols of sort $\mathtt{Proc}$), the only atoms of sort $\mathtt{Proc}$ that might occur in a pure formula are of the kind $i = k_s, k_s = k_{s'}$ for some $s \neq s'$, but these can all be replaced by $\bot$ because we have $\mathtt{Diff}(i, \underline{k})$ in the antecedent of the implication of (11). As a consequence ϕ'' *can be displayed as* $\phi''(\underline{z}, \underline{y}, \sharp\{x \mid \underline{\beta}'(\underline{z}, x, i, \underline{k}, a(x))\})$.

A similar observation applies also to the $\underline{\beta}'$, however here we must take into consideration also atoms of the kind $x = i, x = k_s$. Thus, the $\underline{\beta}'$ are built up using Boolean connectives from atoms of the kind $x = i, x = k_s$, from arithmetic atoms $A(\underline{z}, \underline{y})$ and from $\mathtt{Data}$-atoms that might contain the term $a(x)$. We can disregard arithmetic atoms, because for each such atoms $A(\underline{z}, \underline{y})$ we may rewrite ϕ'' as

$$[A(\underline{z}, \underline{y}) \wedge \phi''(\underline{z}, \underline{y}, \sharp\{x \mid \underline{\beta}'(\top/A)\})] \;\vee\; [\neg A(\underline{z}, \underline{y}) \wedge \phi''(\underline{z}, \underline{y}, \sharp\{x \mid \underline{\beta}'(\bot/A)\})] \; . \tag{12}$$

Thus the $\underline{\beta}'$ can be displayed as $\underline{\beta}'(x, i, \underline{k}, a(x))$.

When $x = i$ or $x = k_s$ (for some s) the $\underline{\beta}'$ can be simplified to $\top$ or $\bot$ because we know the values of $a(i), a(k_s)$ (and as a consequence the numbers $\sharp\{x \mid x = i \wedge \underline{\beta}'\}, \sharp\{x \mid x = k_s \wedge \underline{\beta}'\}$ are 0/1-tuples). In conclusion we have that, for some tuple of numbers $\underline{m}$[7] that can be computed, we have that (11) is equivalent to

$$\forall i.\ \mathtt{Diff}(i, \underline{k}) \wedge a(i) = \mathtt{a}_\mathtt{j} \to \exists \underline{y}\, \phi''(\underline{z}, \underline{y}, \bar{m} + \sharp\{x \mid \mathtt{Diff}(x, i, \underline{k}) \wedge \underline{\beta}''(a(x))\}) \tag{13}$$

where $\underline{\beta}''$ is obtained from $\underline{\beta}'$ by replacing the atoms $x = i, x = k_s$ with $\bot$. Fix now some β''_s from the tuple $\underline{\beta}''$; for every enumerated data $\mathtt{a}_k$, each of the formulæ $\beta''_s(\mathtt{a}_k)$ simplify to either $\top$ or $\bot$ and, since we know that $u_k = \sharp\{x \mid a(x) = \mathtt{a}_k\}$ from (10), we can deduce that $\sharp\{x \mid \mathtt{Diff}(x, i, \underline{k}) \wedge a(x) = \mathtt{a}_k \wedge \beta''_s(\mathtt{a}_k)\}$ is equal to either 0 (in case $\beta''_s(\mathtt{a}_k)$ simplifies to $\bot$) or to $u_k - n_k$, where n_k is the number of the $\underline{k}, i$ for which we know that $a(\underline{k}), a(i)$ is equal to $\mathtt{a}_k$. As a consequence $\sharp\{x \mid \mathtt{Diff}(x, i, \underline{k}) \wedge \underline{\beta}''(a(x))\}$ is equal to $\sum_k (u_k - n_k)$ (where the sum extends to all k such that $\beta''_s(\mathtt{a}_k)$ simplifies to $\top$).

All this can be summarized by saying that we can rewrite (13) as

$$\forall i.\ \mathtt{Diff}(i, \underline{k}) \wedge a(i) = \mathtt{a}_\mathtt{j} \to \exists \underline{y}\, \theta_j(\underline{y}, \underline{z}, \underline{u}) \tag{14}$$

where the formulæ θ_j are pure (the tuple $\underline{u}$ is the tuple of the u_j from (10)). By Presburger quantifier elimination, we can drop the $\exists \underline{y}$, thus getting

$$\forall i.\ \mathtt{Diff}(i, \underline{k}) \wedge a(i) = \mathtt{a}_\mathtt{j} \to \theta'_j(\underline{z}, \underline{u}) \tag{15}$$

Since now θ'_j does not contain occurrences of i, we can rewrite this as

$$\exists i\ (\mathtt{Diff}(i, \underline{k}) \wedge a(i) = \mathtt{a}_\mathtt{j}) \to \theta'_j(\underline{z}, \underline{u}) \tag{16}$$

and finally as

$$\sharp\{x \mid \mathtt{Diff}(x, \underline{k}) \wedge a(x) = \mathtt{a}_\mathtt{j}\} > 0 \to \theta'_j(\underline{z}, \underline{u}) \tag{17}$$

This is a 1-flat formula. To sum up, our original formula (9) is equivalent to a formula of the kind $\exists a\, \exists \underline{u}\, \vartheta$, where ϑ is 1-flat. Then (after swapping the quantifiers $\exists a\, \exists \underline{u}$) we can first use Theorem 1 to remove $\exists a$ and then Lemma 1 to produce an equivalent pure open formula (involving just the arithmetic variables $\underline{z}$ and the index variables $\underline{k}$). □

[7] This tuple depends on j, i.e. on the $\mathtt{a}_j$ used in the antecedent of (11) (we do not indicate this dependency for simplicity).

In case we have uniformity, we can further extend the above result to cover formulæ in which arithmetic array-ids and matrix-ids occur.

Remark 4 Let us continue the considerations we made in Remark 3. We saw there that, when building i-uniform (also 1-flat) formulae, we can employ terms counting the messages received by a process i. Suppose that in our protocol we want to say for instance that all processes i having received enough messages (e.g. messages from some qualified majority of the network) are allowed to change their status into some 'accepting' status: to express this, we need a formula of the kind $\forall i\ \phi$, where ϕ is i-uniform 1-flat (this is the case for instance of the examples of Appendix A in the supplementary material). Next Theorem guarantees that we can indeed eliminate the higher order quantifiers over array-ids and matrix-ids from (a slight generalized set of) such formulae.

Theorem 3 *Let the sort* `Data` *be enumerated and let i be an index variable; suppose that all matrix-ids* $\mathbf{M}$ *occurring in the 1-flat formula ϕ are i-uniform and that all array-ids* $\mathbf{a}$ *occurring in ϕ are either finitary or i-uniform. Then the formula*

$$\exists \mathbf{a}\, \exists \mathbf{M}\, \forall i\, \exists \underline{y}\, \phi \tag{18}$$

(where the $\underline{y}$ are arithmetic and data variables) is equivalent to an open pure formula.

Proof The first step is to remove $\exists M$ for each $M \in \mathbf{M}$, using uniformity. In fact, by uniformity, M occurs in ϕ only inside terms of the kind $M_i(y)$ (for some index variable y); thus, applying a reverse skolemization step, we can rewrite (18) as

$$\exists \mathbf{a}\, \forall i\, \exists \mathbf{b}\, \exists \underline{y}\, \phi(\cdots \mathbf{b}/\mathbf{M}_i \cdots) \tag{19}$$

(here the $\mathbf{b}$ are existentially quantified array-ids: formula (18) is the skolemization of (19)). Then we swap the existential quantifiers $\exists \mathbf{b} \exists \underline{y}$ and apply Theorem 1 to the subformula $\exists \mathbf{b}\, \phi(\cdots \mathbf{b}/\mathbf{M}_i \cdots)$, thus obtaining a formula of the kind $\exists \mathbf{a}\, \forall i\, \exists \underline{y}\, \exists \underline{e}\, \psi$ where the $\underline{e}$ are further arithmetic or data variables, ψ is 1-flat and contains only the array-id $\mathbf{a}$. Let us now split the $\mathbf{a}$ as $\mathbf{a}', \mathbf{a}''$, where the $\mathbf{a}''$ are i-uniform and the $\mathbf{a}'$ are finitary (notice that the syntactic transformations of Theorem 1 maintain the i-uniformity of the $\mathbf{a}''$). We can apply the same anti-skolemization argument to the $\mathbf{a}''$ and rewrite $\exists \mathbf{a}'\, \mathbf{a}''\, \forall i\, \exists \underline{y}\, \exists \underline{e}\, \psi$ as $\exists \mathbf{a}'\, \forall i\, \exists \underline{z}\, \exists \underline{y}\, \exists \underline{e}\, \psi(\underline{z}/\mathbf{a}''(i))$, where the $\underline{z}$ are fresh arithmetic variables replacing the terms $\mathbf{a}''(i)$ in ψ. Now Theorem 2 can be used to eliminate the $\mathbf{a}'$. $\square$

4 System Specifications and Simulations

We now turn to verification applications. The behavior of a system can be modeled through a *transition system*, which is a tuple

$$\mathcal{T} = (W, W_0, R, AP, V)$$

such that (i) W is the set of possible configurations, (ii) $W_0 \subseteq W$ is the set of initial configurations, (iii) AP is a set of 'atomic propositions', (iv) $V : W \longrightarrow AP$ is a function labeling each state with the set of propositions 'true in it', (v) $R \subseteq W \times W$ is the transition relation: $w_1 R w_2$ describes how the system can 'evolve in one step'.

Definition 1 We say that the transition system $\mathcal{T}' = (W', W_0', R', AP, V')$ *simulates* the transition system $\mathcal{T} = (W, W_0, R, AP, V)$ (notice that AP is the same in the two systems) iff there is a relation $\rho \subseteq W \times W'$ (called *simulation*) such that

(i) for all $w \in W$ there is $w' \in W'$ such that $w \rho w'$;
(ii) if $w \rho w'$ and $w \in W_0$, then $w' \in W_0'$;
(iii) if $w \rho w'$ and $w R v$, then there is $v' \in W'$ such that $w' R' v'$ and $v \rho v'$;
(iv) if $w \rho w'$, then $V(w) = V'(w')$;

If the converse ρ^{op} of ρ is also a simulation, then ρ is said to be a *bisimulation* and $\mathcal{T}'$ and $\mathcal{T}$ are said to be *bisimilar*.

Bisimilar systems are equivalent in the sense that the properties expressible in common temporal logic specifications (e.g. in CTL, LTL, CTL^*, etc.) are invariant under bisimulations; simulation is also useful as important properties (like safety properties, or more generally properties expressible in sublogics like $ACTL$) can be transferred from a system to the systems simulated by it (but not vice versa).

We write $\mathcal{T} \leq \mathcal{T}'$ iff $W \subseteq W'$ and the inclusion is a simulation. This relation is a partial order and notice that if $\mathcal{T}'$ simulates $\mathcal{T}$ and $\mathcal{T}' \leq \mathcal{T}''$, then $\mathcal{T}''$ also simulates T; in this case, the simulation supplied by $\mathcal{T}'$ is said to be *stronger* or *better* (in fact, one has more chances of establishing an $ACTL$-property of $\mathcal{T}$ by using $\mathcal{T}'$ than by using $\mathcal{T}''$).

The above formalism of transition systems is often too poor, because it cannot cover rich features arising in concrete applications. That is why we need higher order logic, namely constrained signatures as introduced in Sect. 2. Constrained signatures are used for our system specifications as follows:

Definition 2 A *system specification* $\mathcal{S}$ is a tuple

$$\mathcal{S} = (\Sigma, \mathbf{v}, \Phi, \iota, \tau, AP)$$

where (i) Σ is a constrained signature, (ii) $\mathbf{v}$ is a tuple of variables, (iii) Φ, ι are $\mathbf{v}$-formulæ and AP is a set of $\mathbf{v}$-formulæ, (iv) τ is a $(\mathbf{v}, \mathbf{v}')$-formula (here the $\mathbf{v}'$ are renamed copies of the $\mathbf{v}$) such that

$$\iota(\mathbf{v}) \models_\Sigma \Phi(\mathbf{v}), \qquad \Phi(\mathbf{v}) \wedge \tau(\mathbf{v}, \mathbf{v}') \models_\Sigma \Phi(\mathbf{v}') \quad . \tag{20}$$

In the above definition, the $\mathbf{v}$ are meant to be the variables specifying the system status, ι is meant to describe initial states, τ is meant to describe the transition relation and the AP are the ‘observable propositions’ we are interested in. The $\mathbf{v}$-formula Φ, as it is evident from (20), describes an invariant of the system (known to the user). Of course, since in our expressive type theory higher order quantifiers are available, it would be easy to write down the ‘best possible’ invariant describing in a precise way the set of reachable states; however, the $\mathbf{v}$-formula for such invariant might involve logical constructors lying outside the tractable fragments we plan to use. On the other hand, invariants are quite useful - and often essential - in concrete verification tasks, that is why we included them in Definition 2.

It is now clear how to associate a transition system with any system specification:

Definition 3 The transition system associated with the system specification $\mathcal{S} = (\Sigma, \mathbf{v}, \Phi, \iota, \tau, AP)$ is the transition system $\mathcal{T}^{\mathcal{S}}$ given by

$$(W^{\mathcal{S}}, W_0^{\mathcal{S}}, R^{\mathcal{S}}, AP^{\mathcal{S}}, V^{\mathcal{S}})$$

where: (i) the set of states $W^{\mathcal{S}}$ is the set of the tuples $\mathbf{v}_{\mathcal{M},\mathcal{I}}$ satisfying $\Phi(\mathbf{v})$, varying $\mathcal{M}, \mathcal{I}$ among the Σ-models and Σ-interpretations of $\mathbf{v}$; (ii) $W_0^{\mathcal{S}}$ is the set of states satisfying $\iota(\mathbf{v})$;

(iii) $R^{\mathcal{S}}$ contains the couples of states $\mathbf{v}_{\mathcal{M},\mathcal{I}}, \mathbf{v}'_{\mathcal{M},\mathcal{I}'}$[8] satisfying $\tau(\mathbf{v}, \mathbf{v}')$; (iv) $AP^{\mathcal{S}}$ is AP; (v) for $\phi(\mathbf{v}) \in AP^{\mathcal{S}}$, we have that $V(\phi)$ contains precisely the states satisfying $\phi(\mathbf{v})$.

4.1 Simulations

Model-checking a transition system like $\mathcal{T}^{\mathcal{S}}$ might be a terribly difficult task, that is why it might be useful to replace it with a (bi)similar, simpler system: in our applications, we shall try to replace $\mathcal{S}$ by some $\mathcal{S}'$ whose variables are all integer variables. To this aim, we 'project' $\mathcal{S}$ onto a subsystem $\mathcal{S}'$, i.e. onto a system comprising only some of the variables of $\mathcal{S}$.

In order to give a precise definition of what we have in mind, we must first consider subsignatures: here a *subsignature* Σ_0 of Σ is a signature obtained from Σ by dropping some symbols of Σ and taking as Σ_0-models the class $\mathcal{C}_{\Sigma_0}$ of the restrictions $\mathcal{M}_{|\Sigma_0}$ to the Σ_0-symbols of the structures $\mathcal{M} \in \mathcal{C}_{\Sigma}$.

Definition 4 Let $\mathcal{S} = (\Sigma, \mathbf{v}, \Phi, \iota, \tau, AP)$ be a system specification; a *sub-system specification* of it is a system specification $\mathcal{S}_0 = (\Sigma_0, \mathbf{v}_0, \Phi_0, \iota_0, \tau_0, AP_0)$ where Σ_0 is a subsignature of Σ, $\mathbf{v}_0 \subseteq \mathbf{v}$, $AP_0 = AP$ and we have

$$\Phi(\mathbf{v}) \models_{\Sigma} \Phi_0(\mathbf{v}_0), \quad \iota(\mathbf{v}) \models \iota_0(\mathbf{v}_0), \quad \Phi(\mathbf{v}) \wedge \tau(\mathbf{v}, \mathbf{v}') \models \tau_0(\mathbf{v}_0, \mathbf{v}'_0) \tag{21}$$

The following fact is immediate:

Proposition 1 *Let $\mathcal{S}_0$ be a sub-system specification of $\mathcal{S}$ like in Definition 4; then the map $\pi_{\mathcal{S}_0}$ associating*
$(\mathbf{v})_{\mathcal{M}_{|\Sigma_0},\mathcal{I}_{|\mathbf{v}_0}}$ to $\mathbf{v}_{\mathcal{M},\mathcal{I}}$ is a simulation of $\mathcal{T}^{\mathcal{S}}$ by $\mathcal{T}^{\mathcal{S}_0}$ (called a projection simulation *over $\Sigma_0, \mathbf{v}_0$).*

Projection simulations are ordered according to the ordering of the simulations of $\mathcal{S}$ they produce, i.e. we say that $\mathcal{S}_0$ is *stronger* or *better* than $\mathcal{S}'_0$ iff $\mathcal{T}^{\mathcal{S}_0} \leq \mathcal{T}^{\mathcal{S}'_0}$. Once $\Sigma_0, \mathbf{v}_0$ are fixed, one may wonder whether there exists the best projection simulation over $\Sigma_0, \mathbf{v}_0$. The following easy result supplies a (practically useful) sufficient condition:

Proposition 2 *Let $\mathcal{S} = (\Sigma, \mathbf{v}, \Phi, \iota, \tau, AP)$ be a system specification, let Σ_0 be a subsignature of Σ and let $\mathbf{v}_0 \subseteq \mathbf{v}$ be Σ_0-variables. Suppose that there exist Σ_0-formulæ $\Phi_0(\mathbf{v}_0), \iota_0(\mathbf{v}_0), \tau_0(\mathbf{v}_0, \mathbf{v}'_0)$ such that (let $\mathbf{v} := \mathbf{v}_0, \mathbf{v}_1$):*

(i) $\models_{\Sigma} \Phi_0(\mathbf{v}_0) \leftrightarrow \exists \mathbf{v}_1 \Phi(\mathbf{v}_0, \mathbf{v}_1)$;
(ii) $\models_{\Sigma} \iota_0(\mathbf{v}_0) \leftrightarrow \exists \mathbf{v}_1 \iota(\mathbf{v}_0, \mathbf{v}_1)$;
(iii) $\models_{\Sigma} \tau_0(\mathbf{v}_0, \mathbf{v}'_0) \leftrightarrow \exists \mathbf{v}_1 \exists \mathbf{v}'_1 (\Phi(\mathbf{v}_0, \mathbf{v}_1) \wedge \tau(\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}'_0, \mathbf{v}'_1))$.

If we let $\mathcal{S}_0$ be the subsystem specification $(\Sigma_0, \mathbf{v}_0, \Phi_0, \iota_0, \tau_0, AP)$, then the projection simulation $\pi_{\mathcal{S}_0}$ is the best projection simulation over $\Sigma_0, \mathbf{v}_0$.

Proof That $\mathcal{S}_0 = (\Sigma_0, \mathbf{v}_0, \Phi_0, \iota_0, \tau_0, AP)$ is a subsystem specification of $\mathcal{S}$ is clear; let us now pick another subsystem specification $\mathcal{S}' = (\Sigma_0, \mathbf{v}_0, \Phi', \iota', \tau', AP)$ of $\mathcal{S}$ inducing a projection simulation over the same subsignature Σ_0 and the same sub-tuple of variables $\mathbf{v}_0$. According to (21), we have

$$\Phi(\mathbf{v}) \models_{\Sigma} \Phi'(\mathbf{v}_0), \quad \iota(\mathbf{v}) \models \iota'(\mathbf{v}_0), \quad \Phi(\mathbf{v}) \wedge \tau(\mathbf{v}, \mathbf{v}') \models \tau'(\mathbf{v}_0, \mathbf{v}'_0)$$

[8] Notice that $\mathcal{M}$ is the same in $\mathbf{v}_{\mathcal{M},\mathcal{I}}$ and in $\mathbf{v}'_{\mathcal{M},\mathcal{I}'}$. In principle, $W^{\mathcal{S}}$ might be a proper class, but if one wants to avoid this, it is sufficient to ask for the set of models $\mathcal{C}_{\Sigma}$ of the constrained signature Σ to be a set, not a proper class.

that is

$$\Phi_0(\mathbf{v}_0) \models_\Sigma \Phi'(\mathbf{v}_0), \quad \iota_0(\mathbf{v}_0) \models \iota'(\mathbf{v}_0), \quad \tau_0(\mathbf{v}, \mathbf{v}') \models \tau'(\mathbf{v}_0, \mathbf{v}'_0)$$

which guarantees that $\mathcal{T}^{\mathcal{S}_0} \leq \mathcal{T}^{\mathcal{S}'_0}$. □

To understand the meaning of the above proposition, one should keep in mind that there is no reason why the Σ-formulæ $\exists \mathbf{v}_1 \Phi$, $\exists \mathbf{v}_1 \iota$ and $\exists \mathbf{v}_1 \exists \mathbf{v}'_1(\Phi \wedge \tau)$ should be equivalent to Σ_0-formulæ (in our applications, Σ_0 contains only the sort and the symbols of linear first-order arithmetic, so no higher-order variables are allowed in Σ_0-formulæ). Thus, the road map to apply Proposition 2 is to prove some *quantifier-elimination* results in order to find Σ_0-formulæ equivalent to $\exists \mathbf{v}_1 \Phi$, $\exists \mathbf{v}_1 \iota$, $\exists \mathbf{v}_1 \exists \mathbf{v}'_1(\Phi \wedge \tau)$.

Such quantifier elimination results will supply *the best possible projected simulation* onto the set of the selected variables, as informally mentioned in the toy example of Sect. 1.2.

4.2 Bisimulations

For the sake of completeness, we make a little *digression* on bisimulations (in fact, in some lucky cases, the best projection simulation is also a bisimulation). The content of this digression will not be used in the rest of the paper.

For considerations involving bisimulations, it is useful to consider special kinds of subsignatures, those whose models are quite close to the models of the original signature:

we say that Σ_0 is a *core* subsignature iff every Σ_0-model is the restriction of a *unique* (up to isomorphism) Σ-model. As a typical example (used in our applications) of this situation, consider a signature Σ containing the sorts $\mathbb{Z}$, Ω (with the usual operations), some enumerated sorts, and a finite sort `Proc` whose cardinality is constrained by a parameter `N` (we do not have operations on `Proc`, just the cardinality function $\sharp$ on $\wp$(`Proc`)). Suppose that now we consider the subsignature Σ_0 containing just $\mathbb{Z}$ and Ω (with inherited operations) *and* the parameter `N`: this is evidently a core subsignature, because the interpretation of enumerated sorts and of the sort `Proc` is uniquely determined—up to isomorphism—by the Σ_0-reduct.

Proposition 3 *Let $\mathcal{S}_0$ be a core sub-system specification of $\mathcal{S}$ ($\mathcal{S}$ and $\mathcal{S}_0$ are as displayed in Definition 4) and let $\mathbf{v} := \mathbf{v}_0, \mathbf{v}_1$. Then $\pi_{\mathcal{S}_0}$ is a bisimulation iff the following conditions*

(i) $\models_\Sigma \Phi_0(\mathbf{v}_0) \leftrightarrow \exists \mathbf{v}_1 \Phi(\mathbf{v}_0, \mathbf{v}_1)$;
(ii)$'$ $\iota_0(\mathbf{v}_0) \models_\Sigma \forall \mathbf{v}_1(\Phi(\mathbf{v}_0, \mathbf{v}_1) \to \iota(\mathbf{v}_0, \mathbf{v}_1))$;
(iii)$'$ $\Phi_0(\mathbf{v}_0) \wedge \tau_0(\mathbf{v}_0, \mathbf{v}'_0) \models_\Sigma \forall \tilde{\mathbf{v}}_1(\Phi(\mathbf{v}_0, \tilde{\mathbf{v}}_1) \to \exists \mathbf{v}'_1 \tau(\mathbf{v}_0, \tilde{\mathbf{v}}_1, \mathbf{v}'_0, \mathbf{v}'_1))$

are satisfied.

Proof We need to show that assuming the four conditions of Definition 1 for both $\pi_{\mathcal{S}_0}$ and its converse relation is the same as assuming the three conditions above. First notice that condition (iv) of Definition 1 is trivially satisfied (both for $\pi_{\mathcal{S}_0}$ and its converse relation) because the formulæ in AP are $\mathbf{v}_0$-formulæ in the signature Σ_0 according to the definition of a subsystem specification (Definition 4). Similarly, conditions (i)–(iii) of Definition 1 are also guaranteed for $\pi_{\mathcal{S}_0}$ by Definition 4, so that such conditions are relevant only for the converse of $\pi_{\mathcal{S}_0}$.

Since $\mathcal{S}_0$ is a core sub-system specification of $\mathcal{S}$, asking that "every $(\mathbf{v}_0)_{\mathcal{M}_0,\mathcal{I}_0}$ satisfying Φ_0 comes (by restriction to Σ_0, $\mathbf{v}_0$) from some $(\mathbf{v})_{\mathcal{M},\mathcal{I}}$ satisfying Φ"[9] is the same as asking

[9] This is the relevant content of condition (i) of Definition 1 in our case, where the simulation relation is the converse of the projection function $\pi_{\mathcal{S}_0}$.

condition (i) above: the only possible candidate for $\mathcal{M}$ is the unique extension of $\mathcal{M}_0$ to Σ, hence the above statement under quotation marks holds just in case $\Phi_0(\mathbf{v}_0) \leftrightarrow \exists \mathbf{v}_1 \Phi(\mathbf{v}_0, \mathbf{v}_1)$ is valid in all Σ-models.

Similarly, it is now evident that condition (ii) of Definition 1 for the converse of $\pi_{\mathcal{S}_0}$ is the same as condition (ii)$'$ above and condition (iii) of Definition 1 for the converse of $\pi_{\mathcal{S}_0}$ is the same as condition (iii)$'$ above. □

Notice that condition (i) is the same in Proposition 2 and in Proposition 3. Moreover, using such condition (i) (and the fact that $\iota_0(\mathbf{v}_0) \models_{\Sigma_0} \Phi_0(\mathbf{v}_0)$, see Definition 2) it is not difficult to see that condition (ii)$'$ of Proposition 3 is stronger than the corresponding condition (ii) of Proposition 2; the same observation applies also to conditions (iii)$'$ and (iii) in case we make the additional mild and obvious assumption that $\tau_0(\mathbf{v}_0, \mathbf{v}_0') \models_{\Sigma} \Phi_0(\mathbf{v}_0)$.

4.3 Arithmetic Projections

Let now $\mathcal{S} = (\Sigma, \mathbf{v}, \Phi, \iota, \tau, AP)$ be a system specification based on a signature Σ as discussed in Sect. 4.1. The variables $\mathbf{v}$ of $\mathcal{S}$ include some integer variables $\mathbf{v}_0$ and in addition variables for arrays and matrices. If a is an array variable, then $a(y)$ represents a local status of the process y; if M is a matrix variable, then $M_i(y)$ represents the content of a message received by i from y (or, in other words, sent by y to i).[10] Let us suppose that $\mathbf{v} = \mathbf{v}_0\mathbf{v}_1$, where $\mathbf{v}_1$ is the tuple of array and matrix variables and the $\mathbf{v}_0$ are all the integer variables of the system. We suppose also that the formulæ in AP—namely the formulæ expressing observable properties—are all open $\mathbf{v}_0$-formulæ (in particular, they are all Σ_0-formulæ, where Σ_0 is the arithmetic subsignature of Σ).

Let $\mathcal{S} = (\Sigma, \mathbf{v}_0\mathbf{v}_1, \Phi, \iota, \tau, AP)$ be as above; a subsystem specification of the kind $\mathcal{S}_0 = (\Sigma_0, \mathbf{v}_0, \Phi_0, \iota_0, \tau_0, AP)$ is called a *counter abstraction* of $\mathcal{S}$; as usual, counter abstractions are ordered according to the ordering of the simulations of $\mathcal{S}$ they produce, i.e. we say that $\mathcal{S}_0$ is stronger than $\mathcal{S}_0'$ iff $\mathcal{T}^{\mathcal{S}_0} \leq \mathcal{T}^{\mathcal{S}_0'}$. We are interested in sufficient conditions on Φ, ι, τ ensuring the existence of a strongest counter abstraction.

Theorem 4 *If Φ, ι, τ do not contain matrix-ids and are of the kind $\exists k_1 \cdots \exists k_n \phi$ for a 1-flat formula ϕ and for index variables $k_1, \dots, k_n$, then the system specification $\mathcal{S} = (\Sigma, \mathbf{v}_0\mathbf{v}_1, \Phi, \iota, \tau, AP)$ has a strongest (effectively computable) counter abstraction.*

Proof We apply Proposition 2. Let us rename the arithmetic variables $\mathbf{v}_0$ as $\underline{z}$; then $\mathbf{v}_1$ is the tuple of array-ids $\mathbf{a}$; we also abbreviate $k_1, \dots, k_n$ as $\underline{k}$. We need to show that a formula of the kind

$$\exists \mathbf{a}\, \exists \underline{k}\, \alpha(\underline{z}, \underline{k}, \mathbf{a}(\underline{k}), \sharp\{x \mid \underline{\beta}(\underline{z}, x, \underline{k}, \mathbf{a}(x), \mathbf{a}(\underline{k}))\}) \tag{22}$$

is equivalent to a pure arithmetic formula.[11] But this is indeed the case: just swap the existential quantifiers and apply Corollary 1 and Lemma 1. The result follows because there are no ground index atoms and all ground data atoms are equivalent to $\top$ or to $\bot$, according to our assumptions from Sect. 2.1. □

Next result concerns specifications using matrix-ids in a finitary signature.

[10] A conventional value may be employed to specify that no message has been sent at all. We do not model time passing and message scheduling in this paper.

[11] In view of condition (iii) of Proposition 2, we need also the observation that formulæ of the kind $\exists \underline{k}\, \phi$ (for 1-flat ϕ) are closed under conjunctions.

Theorem 5 *Let the sort* `Data` *be enumerated and let Φ, ι, τ be disjunctions of formulæ of the kind*

$$\exists \underline{k}\ \forall i\ \exists \underline{y}\ \phi \tag{23}$$

where ϕ is 1-flat, $\underline{k}$ are index variables, $\underline{y}$ are arithmetic and data variables and i is an index variable such that all matrix variables and all non-finitary array variables from $\mathbf{v}$ are i-uniform in ϕ; then $\mathcal{S} = (\Sigma, \mathbf{v}_0\mathbf{v}_1, \Phi, \iota, \tau, AP)$ has a strongest (effectively computable) counter abstraction.

Proof Similar to the proof of Theorem 4, using Theorem 3 instead of Corollary 1. $\dashv$ □

Strongest counter abstractions (typically computed via Theorems 4,5) will be called *arithmetic projections*.

In the next section, we describe the infrastructure we deployed, leveraging the presented formal results, so as to model distributed fault-tolerant algorithms and to verify their properties.

5 Implementation

The quantifier elimination procedures relying on Theorems 1, 2, 3 are very expensive: the procedure of Theorem 1 requires a double exponential blow-up and includes as subprocedures other expensive algorithms, like quantifier elimination in Presburger arithmetic. Although we feel that the above results are needed if one wants to build and then process some formal precise declarative models derived out of the original pseudo-code of fault-tolerant algorithms (see our detailed analysis in Appendix A in the supplementary material for an example of what we mean), from a practical point of view it is often possible to build coarser models requiring a less expressive language. For instance, one trick that is often used in the literature (see e.g. [46]) is to replace a Boolean-valued matrix-id $M(i, x)$ by an arithmetic array $a(i)$ representing the term $\sharp\{x \mid M(i, x)\}$ (the intended meaning is that, in this way, we only *count the number* of the M-messages received by process i, disregarding the source of these messages). Quite often, the loss of expressivity due to such abstractions does not prevent the possibility of formalizing the dynamics of the evolution of the whole system in a satisfactory way.

For the above reasons, we *designed a restricted input specification language* for our tool ArcaSim: we describe below such a language and we supply a milder quantifier elimination procedures applying to it. In the experimental Sect. 6 below, we shall see that such restricted language is sufficient to handle benchmarks from different sources.

5.1 The ArcaSim Tool

ArcaSim accepts system specifications matching the syntactic format explained below and produces as output a file in the Horn SMT_Lib format, ready to be model-checked e.g. by μZ [33], the fixpoint engine of the SMT solver Z3. In successful cases, μZ produces an invariant (entirely expressed in terms of our counters) which guarantees the safety of the original system.

A specification file for ArcaSim should first contain *declarations* for

– parameters,
– integer variables,
– arithmetic array-ids,
– enumerated array-ids.

Parameters include a symbol `N` denoting the (finite but unknown) number of processes acting in the system; moreover, with each enumerated array-id, a number m is associated, whose meaning is that of telling the tool that the values of such array-id are taken into the set $\{0, \dots, m-1\}$.

Then *counters definitions* are introduced: these must have the form of equalities $z_i = \sharp\{x \mid \psi_i(x)\}$, where ψ_i is a Boolean combination of data atoms (in the restricted ArcaSim language the constrained signature Σ is assumed to be finitary, i.e. that the sort `Data` is enumerated).

The system *transition* is given as a single (index variable) universally quantified disjunction of cases of the form

$$\forall i \bigvee_j (\phi_{j1} \wedge \phi_{j2}) \tag{24}$$

where: (i) $\phi_{j1}(\mathbf{a}(i)/\underline{y}, \underline{z})$ is obtained from a conjunction of arithmetic atoms $\phi_{j1}(\underline{y}, \underline{z})$ by replacing some arithmetic variables $\underline{y}$ with terms of the kind $\mathbf{a}(i)$ (notice that the above introduced counters for definable sets may occur here);

(ii) $\phi_{j2}(i)$ is a Boolean combination of data formulæ containing the free variable i.

Notice that primed arithmetic array-ids cannot occur in ϕ_{j1}, whereas both primed and unprimed enumerated array-ids can occur in ϕ_{j2}.

The *initial* formula follows the same syntax as the transition formula (but only one disjunct is allowed), whereas the formula expressing the (negation of the) *safety* property must be an arithmetic formula containing only counters, integer variables and parameters.

We underline that the above limitations to the formats of transitions, initial and safety formulae are due to an implementation choice: our theoretical results from Sects. 3 and 4are much richer, however we realized that most standard benchmarks can be formalized in the above restricted ArcaSim language.

In order to produce a file for μZ, ArcaSim applies the following simplified procedure for quantifier elimination.

(i) First, it eliminates (from the arithmetic part ϕ_{i1} of each transition case) the arithmetic array-ids by reverse skolemization and Presburger quantifier elimination: the formal justification for this is that, for an arithmetic array-id a, we have that $\exists a\, \forall i \bigvee_j (\phi_{j1}(a(i)) \wedge \phi_{j2})$ is equivalent to $\forall i\, \exists z \bigvee_j (\phi_{j1}(z) \wedge \phi_{j2})$ and finally to $\forall i \bigvee_j (\exists z\, \phi_{j1}(z) \wedge \phi_{j2})$ - in this last formula the $\exists \underline{z}$ can be eliminated via Presburger quantifier elimination.

(ii) Then, the whole transition is rewritten as a disjunction of formulæ of the kind

$$\bigwedge_k (z_k = \sharp\{x \mid \psi_k(x)\}) \wedge \alpha \wedge \forall x\, \theta(x) \tag{25}$$

where we have, besides the counter definitions $z_k = \sharp\{x \mid \psi_k(x)\}$, a Boolean assignment α (seen as a conjunction of literals) to the arithmetic atoms occurring in the problem, and a single-variable universally quantified formula $\forall x\, \theta(x)$ built up from data atoms only: the formal justification for this is that an arithmetic assignment can be guessed and that, in presence of it, arithmetic atoms can be everywhere replaced by $\top$ or $\bot$.

(iii) Auxiliary counters are now introduced: we have one counter z_f for each function f associating values to enumerated array-ids and their primed copies. In detail, z_f counts the cardinality of the set

$$\{x \mid \bigwedge_a a(x) = f_a \wedge \bigwedge_a a'(x) = f_{a'}\} . \tag{26}$$

The set of counters z_f is much richer than the set of the user-defined original counters. There are two reasons for that: first, the user might have decided to introduce only a relatively small set of counters, the set of counters he feels it could be sufficient to get an appropriate simulation. Second (more important!), the user can only introduce *static counters*, whereas the z_f are *dynamic counters*: they count how many processes make each possible change of values for their enumerated array-ids.[12] The original counters are expressed as linear combinations of these new dynamic counters; in addition, in each disjunct (25), the universally quantified formula $\forall x\, \theta(x)$ is replaced by the equation $\mathtt{N} = \sum \epsilon_f z_f$, where ϵ_f is 0 or 1 depending on whether the formula defining the set (26) counted by z_f is consistent or not with θ.

(iv) In the final steps, all arithmetic atoms involving old and new counters are collected for each disjunct (25); the new dynamic counters are eliminated by quantifier elimination and the resulting formulæ give the disjuncts of the transition of the desired arithmetic projection of the input system.

Contrary to what one might expect, the quantifier elimination steps in (i) and (iv) are not so problematic, because of the special shapes of the arithmetic formulæ arising from the benchmarks we analyzed. In fact, we did not even use a full Presburger quantifier elimination module in ArcaSim for the reasons we are going to explain. In our examples, the quantifier elimination problems in (i) involve just easy (‘difference bounds’-like) constraints and those in (iv) are usually solved by a substitution (in other words, the formula where a variable z needs to be eliminated from, always contains an equality like $z = t$).[13] Notice also that, in case a difficult integer quantifier elimination problem arises, shifting to the (better behaved from the complexity viewpoint) Fourier-Motzkin real arithmetic quantifier elimination procedure is a sound strategy: this is because, in the end, the tool needs to produce just a simulation (i.e. an abstraction). Although ArcaSim was prepared to make such a shifting to Fourier-Motzkin procedure, it never applied it during our experiments.

The step (ii) basically amounts to an “all sat” problem (i.e. to the problem of listing all Boolean assignments satisfying a formula), which is difficult but can be handled efficiently. The real bottleneck seems to be the need of introducing in (iii) a large amount of auxiliary ‘dynamic’ counters: future work should concentrate on improving heuristics here.

[12] If, for instance, there are two array-ids a_1, a_2 each of them taking values from the set $\{1, 2, 3\}$, then we have one dynamic counter for every 4-tuple from $\{1, 2, 3\}$: a 4-tuple is seen as a function f from $\{a_1, a_2, a'_1, a'_2\}$ into $\{1, 2, 3\}$ and the corresponding counter z_f counts the number of the processes x such that $a_1(x) = f_{a_1}, a_2(x) = f_{a_2}, a'_1(x) = f_{a'_1}, a'_2(x) = f_{a'_2}$.

[13] In case a maximum choice of static counters is made by the user, one can even formally prove that this is always the case.

6 Our Experiments

In this section we report our experiments; the source files for the benchmarks are available at https://homes.di.unimi.it/~pagae/ARCASIM/, while the ArcaSim executables are available at the link http://users.mat.unimi.it/users/ghilardi/arca_tools/.

6.1 Classes of Problems Considered

We analyzed several benchmarks representative of different classes of problems. In the following, we analyze the peculiarities and complexity of modeling the algorithms belonging to each class. In Appendix A in the supplementary material, three models are described in detail. Most of the verified algorithms are synchronous and round-based, i.e. they assume that within a round both each process performs the computation of the algorithm for that round, and messages sent in the round are delivered to their destinations by the end of the same round. The few exceptions are highlighted. Only *safety* properties of the algorithms are verified with the proposed tool; liveness properties are considered just when they can be rewritten as safety properties (one way to get such a safety reformulation is to strenghten a liveness property by asking that a desired event must happen within a specified number of rounds).

All experiments have been conducted on a PC equipped with Intel Core i7-7700 processor 3.60 GHz and operating system Linux Ubuntu 18.04 (64 bits).

6.1.1 Agreement Algorithms with Either Omission or Malicious Failures.

The algorithms of this class consider a set of N processes residing on different hosts and communicating through a data network. Processes must reach an agreement about some value, in spite of possibile failures of some of them. Faulty processes may either (i) omit to send or receive some of the messages considered by the algorithm (*benign* failures), or (ii) maliciously fail (*byzantine* failures) reporting fake information. In the latter case, malicious processes might also coalesce in order to fool honest processes. In the former case, a process may also fail crash, that is, from a certain point on no message is anymore sent or received.

In order to express our problems within the restricted language explained in Sect. 5 (i.e. in the ArcaSim format), all the models just consider the behavior of correct processes, while faulty ones are *abstracted* away. (This is a technique used also in [34,35,37–39]).

Both omission and byzantine failures are modeled by using a global variable f, whose value may be upper bounded by e.g. the algorithm resilience t. The number of correct processes is thus $N - f$, and every message sent by a correct process is received by all other correct processes. Using the counter abstraction, we disregard the identities of processes sending a certain message; we simply impose that—if cm is the number of correct processes sending a certain kind of message—then each correct process receives in between cm and $cm + f$ messages of that kind, where cm is the worst case of all faulty processes actually failing, and $cm + f$ is the best case of all faulty processes behaving correctly. As far as omission failures are considered, faulty processes may or may not send their messages. As far as byzantine failures are concerned, independently of their state faulty processes may send whatever message they want (or none at all), and even send different messages to different destinations.

The verification results for this set of algorithms are reported in Table 1, where we show the considered algorithm, the property to be verified and the conditions under which it is verified, the number of transitions produced by ArcaSim and its running time, the running time of Z3 to process the file produced by ArcaSim and the outcome of the verification. In order to properly understand the results in the Table, recall that when the μZ module of Z3 gives a `sat` answer, this means that *there exists a safety invariant* for the abstracted counter system (so that the original system is also *safe*); on the contrary, an `unsat` answer by Z3 means that the safety condition for the abstracted counter system is violated (which is likely - but not necessarily - implying that the original is not safe).[14]

The One-Third (OT) algorithm [11] solves the Consensus problem in case of benign failures. Formally, the problem is defined as follows: each process starts with its own initial value. By the end of the algorithm, processes must decide for one of those values so that the following properties are satisfied:

Agreement whenever two processes have reached a decision, the values they have decided on must be equal.

Integrity - Weak Validity if all processes propose the same initial value, they must decide on that value.

Irrevocability if a process has decided on a value it does not revoke its decision later.

No upper bound is needed on the number of faulty processes. As for all other Consensus algorithms considered in this work, we limited the set of possibile initial values to {0, 1} (this is often not a limitation, see the 0-1 theorems from [46]).
The Irrevocability property requires to check that *never in the future* a revocation occurs. For OT and the other Consensus algorithms mentioned in the sequel, we verified it by reformulating Irrevocability as a safety property: an integer global variable dec is used, which is initialized to 0, and whose value becomes 1 whenever a process having already decided for a value v takes a decision for a value $\overline{v} \neq v$. The unsafe condition is $dec > 0$ and—through backward search—we check whether a state with $dec = 1$ can be reached from the initial condition with all processes undecided and with no constraint on their initial values.

Formal verification highlighted something that was not evident in the original formulation of some problems, that is, it allowed to discover that some properties in the problems enunciations are indeed "trivial"; in fact, they cannot be violated for any number of faulty processes. This is the case for the Weak Validity property of OT: if all processes own the same initial value, there is no way to decide for a different value, considering that processes cannot lie.

The Uniform Voting algorithm (UV) [16] similarly solves the Consensus problem in the presence of benign failures. The solved problem is analogous to that defined for OT, reformulating the Integrity property as follows: "Any decision value is the initial value of some process" (which is also indicated as *Strong Validity*). In order to guarantee the Agreement and Irrevocability property, UV requires the system to satisfy a $\mathcal{P}_{nosplit}$ condition that imposes that communication failures must not partition the network, that is, there cannot exist two subsets of processes such that processes belonging to the same subset communicate amongst themselves, but processes belonging to different subsets not. We omitted to include the $\mathcal{P}_{nosplit}$ property in our models because—by abstracting away the processes identities, and just counting both the number of processes performing a certain action and the number of messages received whatever are their sources—we cannot represent the identities of communicating processes and thus there is no way to model the property within the restricted

[14] It may happen that the counter abstraction employed in our model is too coarse (more counters should be introduced) or even that there is no way of certifying the safety of the original system by using just counters projections. For some benchmarks, one can prove offline that the employed counter abstraction is not just a simulation but actually a bisimulation: in such cases, an `unsat` outcome by Z3 correponds to the effective unsafety of the original system.

Table 1 Agreement algorithms with either benign or malicious failures

Algorithm	Property	Conditions	ARCA_SIM		Z3	
			#Trans.	Time (s)	Time (s)	Answer
OT [11]	Agreement	$t \geq 2N/3$	11	0.58	0.50	`sat`
OT [11]	Agreement	$t < 2N/3$	14	0.80	0.01	`unsat`
OT [11]	Weak Validity	$t \geq 2N/3$	11	0.62	0.01	`sat`
OT [11]	Weak Validity	$t < 2N/3$	14	0.81	0.01	`sat`
OT [11]	Irrevocability	$t \geq 2N/3$	22	1.61	0.87	`sat`
OT [11]	Irrevocability	$t < 2N/3$	28	2.23	0.07	`unsat`
BBP [55]	Correctness	$t < N/3$	7	0.43	0.04	`sat`
BBP [55]	Correctness	$t \leq N/3$	7	0.41	0.03	`sat`
BBP [55]	Unforgeability	$t < N/3$	7	0.40	0.03	`sat`
BBP [55]	Unforgeability	$t \leq N/3$	7	0.41	0.02	`unsat`
BBP [55]	Relay (I)	$t < N/3$	7	0.42	0.01	`sat`
BBP [55]	Relay (I)	$t \leq N/3$	7	0.42	0.01	`sat`
BBP [55]	Relay (II)	$t < N/3$	6	0.38	0.03	`sat`
BBP [55]	Relay (II)	$t \leq N/3$	6	0.39	0.03	`sat`
UV [16]	Integrity	–	21	15.36	0.07	`sat`
UV [16]	Agreement	no $\mathcal{P}_{nosplit}$	21	19.85	0.03	`unsat`
UV [16]	Irrevocability	no $\mathcal{P}_{nosplit}$	36	28.62	0.16	`unsat`
CoUV [16]	Integrity	–	15	288.66	0.14	`sat`
CoUV [16]	Agreement	no $\mathcal{P}_{nosplit}$	15	333.29	0.10	`sat`
CoUV [16]	Irrevocability	no $\mathcal{P}_{nosplit}$	23	408.05	0.04	`unsat`

Table 1 continued

Algorithm	Property	Conditions	ARCA_SIM		Z3	
			#Trans.	Time (s)	Time (s)	Answer
SiCoUV [16]	Integrity	–	11	257.81	0.04	`sat`
SiCoUV [16]	Agreement	no $\mathcal{P}_{nosplit}$	11	284.13	0.06	`sat`
SiCoUV [16]	Irrevocability	no $\mathcal{P}_{nosplit}$	19	410.09	1.04	`sat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Integrity	$\alpha = 0 \wedge \neg\mathcal{P}_{safe}$	17	7.95	0.04	`unsat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Integrity	$\alpha = 0 \wedge \mathcal{P}_{safe}$	30	13.14	0.10	`sat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Integrity	$\alpha = 1 \wedge \neg\mathcal{P}_{safe}$	14	6.74	0.05	`unsat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Integrity	$1 \leq \alpha < N/2 \wedge \neg\mathcal{P}_{safe}$	14	6.75	0.06	`unsat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Integrity	$\alpha = 1 \wedge \mathcal{P}_{safe}$	26	11.52	0.13	`sat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Integrity	$1 \leq \alpha < N/2 \wedge \mathcal{P}_{safe}$	26	11.52	0.21	`sat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Agreement	$\alpha = 0 \wedge \neg\mathcal{P}_{safe}$	17	10.73	0.10	`unsat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Agreement	$\alpha = 0 \wedge \mathcal{P}_{safe}$	30	17.66	0.90	`sat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Agreement	$\alpha = 1 \wedge \neg\mathcal{P}_{safe}$	14	8.99	0.12	`unsat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Agreement	$1 \leq \alpha < N/2 \wedge \neg\mathcal{P}_{safe}$	14	8.86	0.06	`unsat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Agreement	$\alpha = 1 \wedge \mathcal{P}_{safe}$	26	15.25	0.56	`sat`
$\mathcal{U}_{T,E,\alpha}$ [10]	Agreement	$1 \leq \alpha < N/2 \wedge \mathcal{P}_{safe}$	26	15.29	0.85	`sat`
SRBP [54]	Correctness	$t < N/2 \wedge \geq (f+1)$ inits	9	1.25	0.02	`sat`
SRBP [54]	Correctness	$t < N/2 \wedge \geq f$ inits	9	1.25	0.01	`unsat`
SRBP [54]	Unforgeability	$t < N/2 \wedge \geq (f+1)$ inits	9	1.25	0.02	`sat`
SRBP [54]	Relay (I)	$t < N/2 \wedge$ no echo	9	1.26	0.02	`sat`
SRBP [54]	Relay (II)	$t < N/2 \wedge$ 1 echo	9	1.61	0.02	`sat`
Ben-Or [9]	Validity	$t < N/5$	13	0.56	0.02	`sat`
Ben-Or [9]	Validity	$t \leq N/3$	21	0.89	0.13	`unsat`
Ben-Or [9]	Agreement	$t < N/5$	13	0.83	0.08	`sat`
Ben-Or [9]	Agreement	$t \leq N/3$	21	1.30	0.23	`unsat`

language of ArcaSim. As a consequence, the models verifications expectedly result in an unsafe outcome.

The Coordinated UV algorithm (CoUV) [16] derives from UV and solves the same problem under the same conditions. Differently from UV—which adopts a distributed communication pattern where each process communicates with all the others—CoUV adopts a rotating coordinator paradigm such that at each round a process behaves as coordinator, to which other processes send their values and which tries to help them decide. The Simplified CoUV (SiCoUV) [16] shortens the algorithm execution with slight modifications of the process computation that allow to reduce the number of rounds needed to decide. In both cases, the rotating coordinators have been modeled by using a local variable $C[x]$ that assumes three values indicating whether x already has been, currently is, or never was so far a coordinator. At the beginning of each round, a coordinator is nondeterministically taken from the yet unelected processes.
These two algorithms show the impact of central coordination on correctness. Both satisfy Agreement also in the presence of partitions: processes in the same partition as the coordinator decide according to its indications, while partitioned processes do not decide. Yet, in CoUV, they may retain some value different from that chosen by the coordinator. If partitions change later and the new coordinator is a previously partitioned process with a value different from that disseminated by the previous coordinator, then a process may change its decision, thus violating Irrevocability. By contrast, in SiCoUV, the simplification—consisting in voting just for the value received by the coordinator of the current phase or for no value at all—prevents processes owning values from previous coordinators to vote for those values, thus possibly inducing inconsistent decisions for stale values.

$\mathcal{U}_{T,E,\alpha}$ [10] solves the Consensus problem without the Irrevocability property, in the presence of byzantine failures. It requires the system to fulfill two properties, namely that the number of malicious messages received by each process in each round is $\leq \alpha < N/2$ ($\mathcal{P}_\alpha$ property), and that the total number of received correct messages is $> N/2$ ($\mathcal{P}_{safe}$ property). Various combinations of both α and $\mathcal{P}_{safe}$ have been considered in our verifications.

In [9], an algorithm is proposed to solve Consensus in asynchronous systems with byzantine failures and a resilience $t < N/5$. Such an algorithm may also work in synchronous systems if the number of faulty processes is upper bounded by $O(\sqrt{N})$. In this case, we modeled an *asynchronous system* in that no time quantization is reproduced, but just a division in phases as in the original algorithm. In the first phase, a process waits till it receives at least $N - t$ messages; it decides the value to diffuse subsequently and switches to the next phase. In the second phase, a process waits till it receives at least $N - t$ messages and tries to decide. If this is not possible, the process goes back to the first phase. Leveraging the backward search, we started from the configurations in which the considered property is violated. The algorithm runs in which the sufficient number of messages is not received—which do not lead to any action—are unimportant; we verified whether the cases in which actions are undertaken by the processes can lead to an unsafe state. As a resilience value we used $t < N/5$ as in the original paper, which actually leads to a *safe* result. Moreover—since the article ignores the lower bound on the resilience for Byzantine failures ($t < N/3$ [48]) – we considered the case of violation of this lower bound which correctly gives *unsafe* results.

The Byzantine Broadcast Primitive (BBP) [55] aims at achieving agreement among the processes about the messages to deliver. This algorithm tolerates byzantine failures and requires that the number t of faulty processes is such that $N > 3t$. BBP is a round-based algorithm operating in synchronous systems. It fulfills the following properties:

Correctness If correct process p broadcasts (p, m, k) in round k, then every correct process accepts (p, m, k) in the same round.

Unforgeability If process p is correct and does not broadcast (p, m, k), then no correct process ever accepts (p, m, k).

Relay If a correct process accepts (p, m, k) in round $r > k$, then every other correct process accepts (p, m, k) in round $r + 1$ or earlier.

The Relay property asks to check all the states reachable from the configurations satisfying the hypothesis; as this is not possibile, we had to re-write this property as two separate safety properties sequentially verified. In Appendix A.1 in the supplementary material, we explain in detail this procedure.

Formal verification reveals that Correctness and Relay properties cannot be violated for any number of faulty processes. The Correctness property cannot be violated since the threshold for acceptance is equal to the minimum number of correct processes, and the initial broadcast is performed by a correct process and thus received by all correct processes; hence, there is no way for a correct process to not receive enough echo's. The Relay property cannot be violated because a correct accepting process must have received at least $N - 2t$ echo's from correct processes; those echo's are received by each correct process, all correct processes send their own echo, and as a consequence there are $N - t$ correct echo's around that allow each correct process to decide.

The Send Receive Broadcast Primitive (SRBP) algorithm from [54] is proposed as a basis for clock synchronization in systems affected by benign failures. It requires an upper bound t on the number f of faulty processes, i.e., $N > 2t$ and $t \geq f$. The algorithm satisfies the same properties as BBP, with the broadcast message consisting in a time signal ($round\ k$). The same two-steps verification of the Relay property is adopted as for BBP. In this case as well, formal verification shows that Unforgeability and Relay properties cannot be violated. The former trivially follows from the fact that no message is around and faulty processes cannot lie. The latter follows from considerations similar to the case of BBP.

6.1.2 Agreement and Reliable Multicast Algorithms with Crash Failures

The algorithms considered for this category have in common a failure model such that processes behave correctly until they possibly fail, but from the failure on no action is anymore taken, nor any message is sent or received (fail-stop model). Crash failures may *partially* disrupt the broadcast transmission of a message in the sense that the message may reach just a subset of its destinations.

Although counter-intuitive, these failures are harder to model than both omission and byzantine failures. We describe the state of each process with a local variable F that can assume three values, indicating if the process is (so far) correct, if it crashed in the past, or if it is crashing in the current algorithm step and it will send only part of the messages scheduled to be transmitted currently. In the last case, by the end of the step the process is moved to the crashed processes and it will do nothing in the future. At every step, the sum of both crashing and crashed processes must not exceed the algorithm resilience.

The results for these algorithms are reported in Table 2.

FloodSet [45] is a Consensus algorithm that satisfies the same properties as OT, with no resilience and terminating after $f + 1$ rounds with f the number of crashed processes in the current run. Processes circulate the values received in the previous rounds; in the last round, processes decide either for the unique value they received, or for a default value if more values have been observed.

Table 2 Agreement and Reliable multicast algorithms with crash failures

Algorithm	Property	Conditions	ARCA_SIM		Z3	
			#Trans.	Time (s)	Time (s)	Answer
FloodSet [45]	Weak Validity	–	11	4.43	0.02	`sat`
FloodSet [45]	Agreement	–	11	5.18	0.03	`sat`
FloodMin [45]	Weak Validity	$k = 1 \wedge \lvert V \rvert = 2$	9	1.37	0.01	`sat`
FloodMin [45]	Strong Validity	$k = 1 \wedge \lvert V \rvert = 2$	9	1.42	0.01	`sat`
FloodMin [45]	k-Agreement	$k = 1 \wedge \lvert V \rvert = 2$	9	1.59	0.03	`sat`
FloodMin [45]	Weak Validity	$k = 1 \wedge \lvert V \rvert = 3$	17	11.84	0.05	`sat`
FloodMin [45]	Strong Validity	$k = 1 \wedge \lvert V \rvert = 3$	17	10.30	0.02	`sat`
FloodMin [45]	k-Agreement	$k = 1 \wedge \lvert V \rvert = 3$	17	13.03	0.08	`sat`
FloodMin [45]	Weak Validity	$k = 2 \wedge \lvert V \rvert = 3$	17	11.78	0.06	`sat`
FloodMin [45]	Strong Validity	$k = 2 \wedge \lvert V \rvert = 3$	17	10.31	0.02	`sat`
FloodMin [45]	k-Agreement	$k = 2 \wedge \lvert V \rvert = 3$	17	13.06	0.05	`sat`
FC [50]	(Strong) Validity	$\lvert V \rvert = 2$	9	13.92	0.03	`sat`
FC [50]	Agreement	$\lvert V \rvert = 2$	9	15.29	0.07	`sat`
EDAC [15]	(Weak) Validity	$\lvert V \rvert = 2$	35	46.95	0.03	`sat`
EDAC [15]	Agreement	$\lvert V \rvert = 2$	35	49.66	0.02	`sat`
UTRB1 [8]	Validity	–	2	0.13	0.06	`sat`
UTRB1 [8]	Unif.Agreem.	–	2	0.19	0.00[15]	`sat`
UTRB1 [8]	Integrity (I)	–	2	0.16	0.00[15]	`sat`
UTRB1 [8]	Integrity (II)	–	2	0.15	0.00[15]	`sat`

FloodMin [45] is a Consensus algorithm that solves the k-agreement problem: the algorithm runs for $\lfloor f/k \rfloor + 1$ rounds—with f defined as before—after which it is requested that the values decided upon by the processes are in a set of cardinality at most k. This reduces to classical Consensus for $k = 1$. FloodMin guarantees both the Weak Validity and the Strong Validity properties defined before. We performed experiments with two values of k, and with a set V of initial values of cardinality either 2 or 3.

FC [50] is a Consensus algorithm that guarantees both Agreement and Strong Validity as defined above; it terminates at round $t + 1$ with $t < N$ the algorithm resilience, and $f \leq t$ number of actually crashed processes; the decision is the smallest received value.

EDAC [15] as well solves Consensus, but it considers the Weak Validity property. This algorithm is *early-deciding* in the sense that—if a process does not detect new failures in the current round—it decides, differently from the above three algorithms.The problem of modeling this algorithm is that it would require each process to record the observed crashes (a process is assumed having crashed when no expected message is received from it in the current round), which should be modeled with bi-dimensional arrays, not available in ArcaSim. Considering the syntactic constraints of ArcaSim, the fact that (i) no message is anymore received from a process after it crashes and (ii) a crashed process is detected within at most the following round by all alive processes, we abstracted this part of the algorithm by using a global variable $cnum$ that counts the crashed processes and is incremented for all processes as soon as a new crash is observed by the first process.

UTRB1 [8] solves the Uniform Timed Reliable Broadcast problem, which guarantees the following properties:[15]

Validity: If a correct process broadcasts a message m, then all correct processes eventually deliver m.

Integrity: For any message m, each process delivers m at most once and only if some process actually broadcasts m.

Uniform Agreement: If any process delivers a message m then all correct processes eventually deliver m.

Timeliness: There exists a known constant Δ such that if the broadcast of a message m is initiated at time t, no process delivers m after $t + \Delta$.

Following the pen-and-paper correctness proof of UTRB1, we applied the Timeliness property to instantiate the "eventual" attribute and verified that both uniform agreement and validity are reached by round $f + 1$. As far as Integrity is concerned, we separately checked its two components, by verifying that (i) if no process broadcasts m then no process delivers it, and (ii) each process delivers m at most once; both conditions are checked at round $f + 1$ when the algorithm run terminates.

6.1.3 Cache Coherence and Mutual Exclusion Algorithms

The MESI [47] and MOESI [53] algorithms have been designed to guarantee cache coherence. This problem affects shared-memory multi-processors systems where more than one process at a time may access the same memory location and copy the content to its processor's cache, but *the algorithms must guarantee that at most one process at a time is allowed to modify its copy*, and successive read operations to the same location must return the most updated value. The problem of verifying these algorithms w.r.t. the safety property above lies in the fact that there may be more processes in the same state—and thus satisfying the same guard—but just one of them is allowed to fire at a time; this feature differentiates the algorithms in question from pure counter-based algorithms. In order to model this characteristic, we used both a global variable $flag$ that is initialized to 0, becomes 1 when one of the processes in either the invalid or the shared state prepares to fire, and returns to 0 after the process has performed its operation, and a local array variable $F[x]$ that is initialized to 0, becomes 1 for a process in either invalid or shared state that is selected to fire, and returns to 0 when that process fires. We constrain the system such that $\#\{x|F[x] = 1\} \leq 1$, that is, at most one process at a time—in one between the invalid and the shared state—may be selected to perform some operation. By contrast, in the system just one process at a time may be in either the modified or the exclusive state, and hence the constraint is not applied to those processes.

Dekker [20] is a classical mutual exclusion algorithm that guarantees that no more than one process is in critical section. We used local variables $T[x]$ to record what process has the turn, and $WE[x]$ as the local flag to record that process x wants to enter the critical section. As before, we use a constraint imposing that $\#\{x|T[x] = 1\} \leq 1$, that is, it is the turn of at most one process at a time; this way $T[x]$ takes the place of the global variable $turn$ in Dekker's algorithm. A variable $ST[x]$ records whether x is currently in the critical section.

The results achieved with these algorithms are shown in Table 3.

[15] The time spent by Z3 was not measurable because it is below the clock tick.

Table 3 Cache coherence and mutual exclusion algorithms

Algorithm	Property	Conditions	ARCA_SIM		Z3	
			#Trans.	Time (s)	Time (s)	Answer
MESI [47]	Cache coherence	–	15	0.21	0.05	`sat`
MOESI [53]	Cache coherence	–	20	0.41	0.07	`sat`
Dekker [20]	Mutual exclusion	–	5	0.04	0.00	`sat`

6.2 Qualitative Comparison with Other Tools

Other tools in the literature aim at verifying safety properties for distributed algorithms. In [46], Consensus algorithms are considered, a specification language ConsL is proposed, and cutoff bounds are supplied to reduce the parameterized verification to a setting with finite number of processes. Amongst the algorithms studied in [46], there are the algorithms OT, UV, CoUV, SiCoUV and Ben-Or described above, whose safety we verified for an unlimited number of processes using ArcaSim. Although the time spent by ArcaSim for verification was greater than that obtained by ConsL–ConsL latency is on the order of tens of milliseconds for the cited algorithms—it succeeded in performing the computation and within reasonable time.

In this section, we present a qualitative comparison between ArcaSim and ByMC [38], which is a verification framework proposed for the validation of threshold algorithms, that is, algorithms where an action is performed when a certain number of messages are received, regardless of who generated them. The comparison was performed by manually "transcripting" some ByMC examples—provided by the virtual machine version 2.4.0 downloaded from [36]—into ArcaSim. The transcription was conducted trying to maintain maximum fidelity to the ByMC models, by developing ArcaSim models that use the same variables and counters as the equivalent ByMC model, define arithmetic assertions equivalent to the assumptions in the ByMC model, and include one transition for each computation step in the ByMC model. For the sake of better comparison, ByMC was run using Z3 as underlying SMT solver. Clearly, ByMC and ArcaSim are two really different approaches to the verification of distributed algorithms: ByMC considers asynchronous algorithms while ArcaSim is more oriented to model synchronicity and some properties are modeled by introducing a bound—usually provided in the original algorithms' descriptions—on the time at which safety must be checked; the two underlying logical paradigm have different expressiveness. Yet, we were interested in investigating whether ArcaSim is able to deal with the same benchmarks as ByMC. Table 4 reports the results; the ByMC time was obtained by adding the displayed system and user times.

The considered algorithms are as follows: STRB [55] is the same as the previously described BBP algorithm (we adopt here the ByMC nomenclature to emphasize that this is a different model mimicking the ByMC one). ABA (Asynchronous Byzantine Agreement) [12] slightly modifies the Agreement problem definition supplied before, and satisfies the following properties:

Correctness If the transmitter is correct, all the correct processes decide on its value.
Unforgeability If the transmitter is malicious, then either no correct process will decide or they will all decide on the same value.

Table 4 Comparison with ByMC

Algorithm	Property	Condition	ARCA_SIM		Z3		ByMC
			#Trans.	Time (s)	Time (s)	Answer	Time (s)
STRB [55]	Correctness	$N > 3t$	7	0.73	0.05	`sat`	0.23
STRB [55]	Unforgeability	$N > 3t$	7	0.74	0.03	`sat`	0.16
ABA [12]	Correctness	$N > 3t$	6	1.79	0.01	`sat`	15.68
ABA [12]	Unforgeability	$N > 3t$	6	1.79	0.01	`sat`	0.95
NBAC [30]	Agreement	$N > f$	5	0.14	0.01	`sat`	5.01
NBAC [30]	Abort-validity	$N > f$	5	0.14	0.01	`sat`	4.81
NBAC [30]	Commit-validity	$N > f$	5	0.15	0.01	`sat`	4.89
FRB [23]	Unforgeability	$N > f$	4	0.59	0.01	`sat`	0.14
FRB [23]	Correctness	$N > f$	4	0.73	0.01	`sat`	0.18

ABA tolerates $< N/3$ malicious failures; it may work in *asynchronous* systems, which are modeled with processes that wait until a sufficient number of messages is received to perform some action.

NBAC (Non-blocking atomic commitment) [30] aims at reaching an agreement amongst processes about an action to perform, and satisfies the following properties:

Agreement No two processes decide differently.
Termination Every correct process eventually decides.
Abort-validity Abort is the only possible decision if some process votes no.
Commit-validity Commit is the only possible decision if every process votes yes and no process crashes.

We did not consider the Termination property as it is a liveness property that cannot be managed by ArcaSim.

FRB (Folklore Reliable broadcast) [23] is an algorithm aiming at distributing a message to all alive processes in a system; it works in *asynchronous* systems and tolerates just crash failures. It satisfies both a Correctness and an Unforgeability property equal to those previously defined for BBP.

To sum up, we believe that the main strength of our approach lies in its declarative (thus expressive and flexible) nature: nevertheless, our first ArcaSim implementation for the restricted fragment of Sect. 5 shows that many benchmarks from disparate literature can be effectively handled in our uniform framework with reasonable performances.

7 Conclusions and Related Work

We introduced a plain technique for automatically building counter simulations of system specifications: the technique consists of modeling system specifications in higher order logic, then of introducing counters for definable sets and finally of exploiting quantifier elimination results to get rid of higher order variables. Such technique is quite flexible and since, whenever it applies, it always supplies the *best* simulation, it should be in principle capable to cover all results obtainable via counter abstractions. We underline some further important specific features of our approach.

First of all, the approach is purely *declarative*: our starting point is the informal description of the algorithms (e.g. in some pseudo-code) and the first step we propose is a direct translation into a standard logical formalism (typically, classical Church type theory), without relying for instance on ad hoc automata devices or on ad hoc specification formalisms. We believe that this choice can ensure flexibility and portability of our methods.

Secondly, the amount of human interaction we require is nevertheless *confined to design choices*: although the final outcome of our investigations should be the integration of our techniques into some logical framework, the key leading to their success relies almost entirely on results (satisfiability and quantifier elimination algorithms) belonging to the realm of decision procedures. In fact, the quantifier elimination results presented in this paper (Theorems 1, 2, 3) are of interest by themselves and go far beyond the well-known case of Boolean Algebras with Presburger Arithmetic (BAPA) [42], because they are not confined to fragments where sets are uninterpreted.

Although we are claiming that the fully declarative approach is the main merit of our approach, we are aware that a full automation is, in the present status of the art, far from reachable. In particular, contrarily to what happens in other verification areas, a uniform established standard for the specification of distributed system is not available. Moreover, even when some form of pseudo-code for an algorithm is supplied in the original source papers, its translation into higher order logic is not completely obvious and requires a careful analysis. In principle, higher order logic is a quite powerful and expressive formalism, so that one might claim that "everything" is expressible in it; however, designing a detailed translation from some fragment of pseudo-code into transition systems expressed in higher order logic is far from obvious and surely requires deep specific work, to be attacked in quite differently oriented papers. In this paper, we just made a syntactic classification of higher-order specifications to which quantifier elimination applies; the question whether a given higher-order specification can be symbolically tranformed up to logical equivalence into a specification falling within the syntactic shape to which our theorems apply, is another, terribly difficult, question to which only very limited answers can be provided. All in all, manual intervention is required even in this stage, especially if one wants to go beyond strict syntactic matching. However, in concrete cases our approach can be really effective: in the online available supplementary material to this paper, we made a careful detailed analysis of three concrete examples, starting from informal description, to pseudo-code specification, to translation into higher-order transition systems, to mechanical verification via counter abstractions.

A potentially weak point to be taken care is the *complexity* of our algorithms: in fact, the procedure for quantifier elimination used in the proof of Theorems 1, 2, 3 produces super-exponential blow-up of the formulæ it is applied to. Notice however that, when building a counters simulation of a concrete algorithm, such a heavy procedure is applied to each instruction (or to each block of instructions) separately, i.e. not to the whole code instance. Moreover, it is not difficult to realize (going through e.g. the computation details from the online available suplementary material) that it is hardly the case that the quantifier elimination procedure is applied in its full generality: in fact, it is always applied to smaller fragments, where complexity presumably drastically reduces (similarly to what happens for satisfiability algorithms, compare e.g. Theorems 2 and 3 in [4]). The same observation applies also to the instances of the Presburger quantifier elimination procedure that are invoked in our manipulations: usually, they are confined to difference bounds formulæ or to formulæ where quantifiers can be eliminated by simple instantiations. The identification of such small fragments and the study of the related complexities is important for future work and preliminary to any implementation effort.

Another delicate point is related to the *syntactic limitations* we require on the formulæ describing system specifications (see the statements of Theorems 4 and 5): such syntactic limitations are needed to ensure higher order quantifier elimination. Although it seems that a significant amount of benchmarks are captured despite such limitations, it is essential to develop techniques applying in more general cases. To this aim, we observe that just over-approximations are needed to build simulations and that, even if the best simulation may not exist, still practically useful simulations might be produced. In fact, quantifier elimination is just an extreme solution to symbol elimination problems. Symbol elimination and interpolation are well-known techniques to build invariants, abstractions and over-approximations, and for this reason their investigation has deserved considerable attention in the automated reasoning literature [41]; still, extensions to higher-order fragments need to be attacked and might be useful in our context.

To conclude, we mention some recent work on the verification of fault-tolerant distributed systems, starting from our own previous contributions. The additional original contributions with respect to our previous paper [4] and its journal version [5] are due to the fact that we are moving from bounded model-checking and invariant checking to the much more challenging task of full model-checking via *invariant synthesis*. As discussed in [5] (Sect. 7), standard model-checking techniques are difficult to apply in the present context of fault-tolerant distributed systems because Pre- and Post-image computations are very expensive and lead to fragments for which full decision procedures seem not to be available. That is why we tried here a different approach, via counter simulations. While developing such a different approach, we established further *new quantifier elimination results for higher-order fragments* (Theorems 2 and 3above): we believe that these new results, although tailored to our specific applications, are of some independent interest from a logical and a technical point of view.

Papers [34,35,37,39] represent a very interesting and effective research line (summarized in [38]), where cardinality constraints are not directly handled but abstracted away using counters. In this sense, this research line looks similar to the methodology we applied in this paper (and in contrast to the alternative methodology we adopted in our previous paper [4]); however, abstraction in [38] and in related papers is not obtained via logical formalizations and quantifier elimination, but via a special specification language (‘parametric Promela’) and/or via special devices, called ‘threshold automata’. A comparison with the counter systems we obtain is not immediate and not always possible because the authors of [38] work on asynchronous (not round-based) versions of the algorithms and because their method suffers of some lack of expressiveness whenever local counters are unavoidable. On the other hand, they are able to certify also liveness properties, whereas at the actual stage we can only do that by making reductions to safety or bounded model checking problems (we applied this method in our experiments, usually taking bounds for reductions from the literature on distributed algorithms—such bounds are often trivially suggested by the round-based structure of our benchmarks).

Paper [11] directly handles cardinality constraints for interpreted sets by employing specifically tailored abstractions and some incomplete inference schemata at the level of the decision procedures. Nontrivial invariant properties are synthesized and checked, based on Horn constraint solving technology; this is the same technology we rely on in our final step, however the counter systems we get are ‘as accurate as possible’, in the sense stated in our main Theorems 4 and 5.

Paper [21] introduces an 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 naturally embedded in a higher order logic context. Paper [21] is not concerned with simulations and bisimulations, rather it uses an incomplete algorithm in order to certify invariants. A smaller fragment (identified via several syntactic restrictions) is introduced in the final part of the paper and a decidability proof for it is sketched.

Cutoff approaches are often employed in the literature on verification of distributed systems: in such approaches, problems involving unboundly many processes are reduced to particular cases where only a finite number of processes has to be considered (as soon as such number is determined, finite state model checking techniques apply). For a recent paper in the area, specifically tailored to fault-tolerant distributed algorithms, see [46].

Finally, we mention the effort made by the interactive theorem proving community in formalizing and verifying fault-tolerant distributed algorithm (see e.g. [14]); such approach is a natural complement to ours.

Funding Open access funding provided by Università degli Studi di Milano within the CRUI-CARE Agreement. The work was conducted in the framework of SEED Project "Situational awareness in critical Infrastructure Environment (SENTINEL)" funded by the Università degli Studi di Milano.

References

1. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: Proceedings of LICS, pp. 313–321 (1996)
2. Abdulla, P.A., Delzanno, G., Henda, N.B., Rezine, A.: Regular model checking without transducers. In: TACAS, volume 4424 of LNCS, pp. 721–736 (2007)
3. Alberti, F., Ghilardi, S., Orsini, A., Pagani, E.: Counter abstractions in model checking of distributed broadcast algorithms: Some case studies. In: Proceedings of CILC, CEUR Proceedings, pp. 102–117 (2016)
4. Alberti, F., Ghilardi, S., Pagani, E.: Counting constraints in flat array fragments. In: Proceedings of the IJCAR, volume 9706 of Lecture Notes in Computer Science, pp. 65–81 (2016)
5. Alberti, F., Ghilardi, S., Pagani, E.: Cardinality constraints for arrays (decidability results and applications). Formal Methods in System Design (2017)
6. Alberti, F., Ghilardi, S., Sharygina, N.: Decision procedures for flat array properties. J. Autom. Reason. **54**(4), 327–352 (2015)
7. Andrews, P.B.: An introduction to mathematical logic and type theory: to truth through proof. Applied Logic Series, vol. 27, 2nd edn. Kluwer, Dordrecht (2002)
8. Babaoglu, O., Toueg, S.: Non-blocking atomic commitment. In: Distributed Systems, 2nd edn., pp. 147–168, Sape Mullender Ed., Addison-Wesley (1993)
9. Ben-Or, M.: Another advantage of free choice: completely asynchronous agreement protocols. In: Proceedings of the PODC, pp. 27–30 (1983)
10. Biely, M., Charron-Bost, B., Gaillard, A., Hutle, M., Schiper, A., Widder, J.: Tolerating corrupted communication. In: Proceedings of the PODC, pp. 244–253 (2007)
11. Bjørner, N., von Gleissenthall, K., Rybalchenko, A.: Cardinalities and universal quantifiers for verifying parameterized systems. In: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) (2016)
12. Bracha, G., Toueg, S.: Asynchronous consensus and broadcast protocols. JACM **32**(4), 824–840 (1985)
13. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: CAV, pp. 334–342 (2014)

14. Charron-Bost, B., Debrat, H., Merz, S.: Formal verification of consensus algorithms tolerating malicious faults. In: Stabilization, Safety, and Security of Distributed Systems, LNCS. pp. 120–134. Springer, Berlin (2011)
15. Charron-Bost, B., Schiper, A.: Uniform consensus is harder than consensus. J. Algorithms **51**(1), 15–37 (2004)
16. Charron-Bost, B., Schiper, A.: The heard-of model: computing in distributed systems with benign faults. In: Distributed Computing, pp. 49–71 (2009)
17. Cimatti, A., Griggio, A.: Software model checking via IC3. In: CAV, pp. 277–293 (2012)
18. Delzanno, G.: Constraint-based verification of parameterized cache coherence protocols. Form. Methods Syst. Design **23**(3), 257–301 (2003)
19. Delzanno, G., Esparza, J., Podelski, A.: Constraint-based analysis of broadcast protocols. In: Proceedings of CSL, volume 1683 of LNCS, pp. 50–66 (1999)
20. Dijkstra, E.W.: Cooperating sequential processes. In: Genuys, F. (ed.) Programming Languages. Academic Press, New York (1968)
21. Dragoj, C., Henzinger, T., Veith, H., Widder, J., Zufferey, D.: A logic-based framework for verifying consensus algorithms. In: Proceedings of the VMCAI (2014)
22. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: Proceedings of LICS, pp. 352–359. IEEE Computer Society (1999)
23. Fisman, D., Kupferman, O., Lustig, Y.: On verifying fault tolerance of distributed protocols. In: Proceedings of the TACAS, 2008. Full paper at https://www.researchgate.net/publication/220852375_On_Verifying_Fault_Tolerance_of_Distributed_Protocols
24. Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: POPL, pp. 191–202 (2002)
25. Gabbay, D.M., Schmidt, R., Szalas, A.: Second Order Quantifier Elimination: Foundations, Computational Aspects and Applications. ACM Digital Library (2008)
26. Ghilardi, S., Pagani, E.: Counter simulations via higher order quantifier elimination: a preliminary report. In: Proceedings of the Fifth Workshop on Proof eXchange for Theorem Proving, PxTP 2017, Brasília, Brazil, 23–24 Sept. 2017, pp. 39–53 (2017)
27. Ghilardi, S., Pagani, E.: Second order quantifier elimination: towards verification applications. In: Proceedings of the Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2017), Dresden, Germany, December 6–8, 2017, pp. 36–50 (2017)
28. Ghilardi, S., Ranise, S.: Backward reachability of array-based systems by SMT solving: termination and Invariant Synthesis. Log. Methods Comput. Sci. **1**, 2 (2010). https://doi.org/10.2168/LMCS-6(4:10)2010
29. Ghilardi, S., Ranise, S.: MCMT: a model checker modulo theories. In: IJCAR, pp. 22–29 (2010)
30. Guerraoui, R.: On the hardness of failure-sensitive agreement problems. Inf. Process. Lett. **79**, 99–104 (2001)
31. Gurfinkel, A., Kahsai, T., Komuravelli, A., Navas, J.A.: The SeaHorn verification framework. In: CAV, pp. 343–361 (2015)
32. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: SAT, pp. 157–171 (2012)
33. Hoder, K., Bjørner, N., de Moura, L.: μZ—an efficient engine for fixed points with constraints. In: CAV, pp. 457–462 (2011)
34. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Parameterized model checking of fault-tolerant distributed algorithms by abstraction. In: Proceedings of International Conference on Formal Methods in Computer-Aided Design (FMCAD), pp. 201–209 (2013, Aug)
35. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Towards modeling and model checking fault-tolerant distributed algorithms. In: Proceedings of the International SPIN Symposium on Model Checking of Software, volume 7976 of Lecture Notes in Computer Science, pp. 209–226. Springer, Berlin (2013, July)
36. Konnov, I.: ByMC: Byzantine Model Checker. http://forsyte.at/software/bymc/, last visit: July 17 (2018)
37. Konnov, I., Veith, H., Widder, J.: On the completeness of bounded model checking for threshold-based distributed algorithms: reachability. In: Proceedings of CONCUR, LNCS, pp. 125–140 (2014)
38. Konnov, I.V., Veith, H., Widder, J.: What you always wanted to know about model checking of fault-tolerant distributed algorithms. In: Perspectives of System Informatics—10th International Andrei Ershov Informatics Conference, PSI 2015, in Memory of Helmut Veith, Kazan and Innopolis, Russia, August 24–27, 2015, Revised Selected Papers, pp. 6–21 (2015)
39. Konnov, I.V., Veith, H., Widder, J.: On the completeness of bounded model checking for threshold-based distributed algorithms: reachability. Inf. Comput. **252**, 95–109 (2017)
40. Koopmann, P., Rudolph, S., Schmidt, R.A., Wernhard, C. (eds.): Proceedings of the Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2017), Dresden, Germany, December 6–8, 2017, volume 2013 of CEUR Workshop Proceedings. CEUR-WS.org (2017)

41. Kovács, L., Voronkov, A.: Interpolation and symbol elimination. In: Automated Deduction—CADE-22. In: 22nd International Conference on Automated Deduction, Montreal, Canada, August 2–7, 2009. Proceedings, pp. 199–213 (2009)
42. Kuncak, V., Nguyen, H.H., Rinard, M.: Deciding boolean algebra with Presburger arithmetic. J. Autom. Reas. **36**(3), 213–239 (2006)
43. Kuncak, V., Nguyen, H.H., Rinard, M.: An Algorithm for deciding BAPA: boolean algebra with Presburger arithmetic. In: Proceedings of CADE-20, volume 3632 of LNCS (July 2005)
44. Lambek, J., Scott, P.J.: Introduction to higher order categorical logic, volume 7 of Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, 1988. Reprint of the 1986 original
45. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, Burlington (1996)
46. Maric, O., Sprenger, C., Basin, D.A.: Cutoff bounds for consensus algorithms. In: Computer Aided Verification—29th International Conference, CAV 2017, Heidelberg, July 24–28, 2017, Proceedings, Part II, pp. 217–237 (2017)
47. Papamarcos, M.S., Patel, J.H.: A low-overhead coherence solution for multiprocessors with private cache memories. In: Proceedings of the ISCA (1984). https://dl.acm.org/citation.cfm?id=808204
48. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. JACM **27**(2), 228–234 (1980)
49. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. Kluwer, Warszawa (1929)
50. Raynal, M.: Fault-tolerant Agreement in Synchronous Message-Passing Systems. Morgan & Claypool—Synthesis Lectures on Distributed Computing Theory series (2010)
51. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.W.: Counterexample-guided quantifier instantiation for synthesis in SMT. In: Proceedings of the CAV, pp. 198–216 (2015)
52. Schweikhart, N.: Arithmetic, first-order logic, and counting quantifiers. In: ACM TOCL, pp. 1–35 (2004)
53. Solihin, Y.: Fundamentals of Parallel Computer Architecture Multichip and Multicore Systems. Solihin Publishing & Consulting LLC, Raleigh (2008)
54. Srikanth, T.K., Toueg, S.: Optimal clock synchronization. JACM **34**(3), 626–645 (1987)
55. Srikanth, T.K., Toueg, S.: Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. Distrib. Comput. **2**(2), 80–94 (1987)