Counter Simulations via Higher Order Quantifier Elimination: a preliminary report

Silvio Ghilardi
Università degli Studi di Milano, Milano, Italy*

Elena Pagani
Università degli Studi di Milano, Milano, Italy

Quite often, 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 a specifically designed technique for second order quantifier elimination. Our method, once applied to specifications of verification problems for parameterized distributed systems, produces integer variables systems that are ready to be model-checked by current SMT-based tools. We demonstrate the feasibility of the approach with a prototype implementation and first experiments.

1 Introduction

In this paper we introduce a methodology moving from higher order specifications down to simulations expressible inside first-order theories, where SMT techniques can be effectively applied. We believe that this methodology, requiring user intervention only for initial choices at design phase, can supply a good example of the interaction between logic engines operating at different expressivity levels. The motivation of our research lies in the area of the verification of distributed (especially fault-tolerant) algorithms, where benchmarks for our first experiments were taken from.

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 [20], [21] and, in the fault-tolerant case, also some fragment of higher order logic [16], [4] (this is needed in order to have some form of comprehension to handle cardinalities of definable sets). On the other hand, for a long time, it has been observed that *counter systems* [14, 15, 17] 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, 25–27]. It should be noticed that, unlike what happens in the old framework of [14, 15, 17], 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 first authar was supported by the INdAM's GNSAGA group.

C. Dubois and B. Woltzenlogel Paleo (Eds.): Fifth Workshop on Proof eXchange for Theorem Proving (PxTP 2017)
EPTCS 262, 2017, pp. 39–53, doi:10.4204/EPTCS.262.5

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 [18], IC3 [13, 23], etc.) developed inside the SMT community lead to impressively performing tools like μZ [24], nuXmv [10], SeaHorn [22], ... which are nowadays being used to solve many verification problems regarding counter systems.

Being conscious that building such 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, 29]; as pointed out in [5, 19, 29], sometimes such decision techniques can be modified so as to supply quantifier elimination results and, via these quantifier elimination results, we shall show how to *automatically build the best possible counter simulations* users can obtain once they fixed (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 benchmarks, counter systems simulations which are effectively model-checked by current SMT-based tools.

1.1 A four-steps strategy

Our general four-steps strategy can be summarized as follows: (1) system specifications (together with their safety problems) 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. The reader is referred to Section 4 for a detailed example.

In this plan, only steps (1) and (2) require manual intervention; step (3) is effective every time the syntactic restrictions for quantifier elimination procedures are matched; step (4) is subject to two risks, namely to the fact that model-checkers may not terminate on such (undecidable) problems and to 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 and there are also positive theoretical results - both classical [1, 20] and more recent [28] - that guarantee termination in some interesting cases. Concerning the second risk, notice that if spurious traces arise, they can be recognized because SMT tools supply concrete values for counterexamples; then, one can go back to step (2) and refine the abstraction by adding more counters.

The paper is structured as follows: Section 2 gives general foundations; Section 3 outlines the formalizations we use and supplies a basic quantifier elimination result; Section 4 analyzes a concrete benchmark; Section 5 describes our implementation and our first experiments. Section 6 concludes.

2 System Specifications in Higher Order Logic

The behavior of a computer system can be modeled through a *transition system*, which is a tuple

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

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* $\mathscr{T}' = (W', W'_0, R', AP, V')$ simulates *the transition system* $\mathscr{T} = (W, W_0, R, AP, V)$ *(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* wRv*, 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* $\mathscr{T}'$ *and* $\mathscr{T}$ *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 $\mathscr{T} \leq \mathscr{T}'$ iff $W \subseteq W'$ and the inclusion is a simulation. This relation is a partial order; notice that if $\mathscr{T}'$ simulates $\mathscr{T}$ and $\mathscr{T}' \leq \mathscr{T}''$, then $\mathscr{T}''$ also simulates T; in this case, the simulation supplied by $\mathscr{T}'$ is said to be *stronger* or *better* than the simulation supplied by $\mathscr{T}''$ (in fact, one has more chances of establishing e.g. a safety property of $\mathscr{T}$ by using $\mathscr{T}'$ than by using $\mathscr{T}''$).

The above formalism of transition systems is often too poor, because it cannot cover rich features arising in concrete applications. 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).[1] 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' $\{\mathtt{a}_0, \ldots, \mathtt{a}_k\}$, where the $\mathtt{a}_i$ 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 `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 (usually, the parameter we choose for this is named `N`) to indicate the 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,[2] together with a class $\mathscr{C}_\Sigma$ of Σ-structures, called the *models* of Σ.[3] Using primitive

[1] Some notation we use might look slightly non-standard; it is similar to the notation of [30].

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

[3] In the standard model-checking literature $\mathscr{C}_\Sigma$ is a singleton; here we must allow *many* structures in $\mathscr{C}_\Sigma$, 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 [31]).

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.

Our constrained signatures always include the sort Ω of truth-values; terms of type Ω are called *formulae* (we use greek letters $\alpha, \beta, \ldots, \phi, \psi, \ldots$ 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. [30]). We shall use the roman letters $x, y, \ldots, i, j, \ldots, v, w, \ldots$ 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 $\mathscr{M} \in \mathscr{C}_\Sigma$ is a function $\mathscr{I}$ mapping each variable onto an element of the correponding type (as interpreted in $\mathscr{M}$). The evaluation of a term $t(\mathbf{v})$ according to $\mathscr{I}$ is recursively defined in the standard way and is written as $t_{\mathscr{M},\mathscr{I}}$. A Σ-formula $\phi(\mathbf{v})$ is *true* under $\mathscr{M}, \mathscr{I}$ iff it evaluates to $\mathtt{tt}$ (in this case, we may also say that $\mathbf{v}_{\mathscr{M},\mathscr{I}}$ *satisfies* ϕ); ϕ is *valid* iff it is true for all models $\mathscr{M} \in \mathscr{C}_\Sigma$ and all interpretations $\mathscr{I}$ of $\mathbf{v}$ over $\mathscr{M}$. 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.

Constrained signatures are used for our system specifications as follows:

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

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

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*

$$\iota(\mathbf{v}) \models_\Sigma \Phi(\mathbf{v}), \qquad \Phi(\mathbf{v}) \wedge \tau(\mathbf{v}, \mathbf{v}') \models_\Sigma \Phi(\mathbf{v}') \quad . \tag{1}$$

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 (1), describes an invariant of the system (known to the user). Of course, using the expressive power of our type theory, 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 (like fixpoints) lying outside the tractable fragments we plan to use. On the other hand, invariants are quite useful - and often essential - in concrete verification tasks, this 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 of the system specification* $\mathscr{S} = (\Sigma, \mathbf{v}, \Phi, \iota, \tau, AP)$ *is the transition system* $T^{\mathscr{S}}$ *given by* $(W^{\mathscr{S}}, W_0^{\mathscr{S}}, R^{\mathscr{S}}, AP^{\mathscr{S}}, V^{\mathscr{S}})$*, where: (i) the set of states* $W^{\mathscr{S}}$ *is the set of the tuples* $\mathbf{v}_{\mathscr{M},\mathscr{I}}$ *satisfying* $\Phi(\mathbf{v})$*, varying* $\mathscr{M}, \mathscr{I}$ *among the* Σ*-models and* Σ*-interpretations of* $\mathbf{v}$*; (ii)* $W_0^{\mathscr{S}}$ *is the set of states satisfying* $\iota(\mathbf{v})$*; (iii)* $R^{\mathscr{S}}$ *contains the couples of states* $\mathbf{v}_{\mathscr{M},\mathscr{I}}, \mathbf{v}'_{\mathscr{M},\mathscr{I}'}$[4] *satisfying* $\tau(\mathbf{v}, \mathbf{v}')$*; (iv)* $AP^{\mathscr{S}}$ *is AP; (v) for* $\psi(\mathbf{v}) \in AP^{\mathscr{S}}$*, we have that* $V(\psi)$ *contains precisely the states satisfying* $\psi(\mathbf{v})$*.*

[4]Notice that $\mathscr{M}$ is the same; $W^{\mathscr{S}}$ might be a proper class, but to avoid this it is sufficient to ask for the set of models $\mathscr{C}_\Sigma$ of our constrained signature Σ to be a *set* (not a proper class).

3 Simulations and Counter Abstractions

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

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 $\mathscr{C}_{\Sigma_0}$ of the restrictions $\mathscr{M}_{|\Sigma_0}$ to the Σ_0-symbols of the structures $\mathscr{M} \in \mathscr{C}_\Sigma$.

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

$$\Phi(\mathbf{v}) \models_\Sigma \Phi_0(\mathbf{v}_0), \quad \iota(\mathbf{v}) \models \iota_0(\mathbf{v}_0), \quad \Phi(\mathbf{v}) \wedge \tau(\mathbf{v}, \mathbf{v}') \models \tau_0(\mathbf{v}_0, \mathbf{v}_0') \tag{2}$$

The following fact is immediate:

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

Projection simulations are ordered according to the ordering of the simulations of $\mathscr{S}$ they produce, i.e. we say that $\mathscr{S}_0$ is *stronger* or *better* than $\mathscr{S}_0'$ iff $\mathscr{T}^{\mathscr{S}_0} \leq \mathscr{T}^{\mathscr{S}_0'}$. 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 straightforward result supplies a (practically useful) sufficient condition:

Proposition 2. *Let $\mathscr{S} = (\Sigma, \mathbf{v}, \Phi, \iota, \tau, AP)$ 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 $\mathscr{S}_0$ be the subsystem specification $(\Sigma_0, \mathbf{v}_0, \Phi_0, \iota_0, \tau_0, AP)$, then the projection simulation $\pi_{\mathscr{S}_0}$ is the best projection simulation over $\Sigma_0, \mathbf{v}_0$.

Proof. That $\mathscr{S}_0 = (\Sigma_0, \mathbf{v}_0, \Phi_0, \iota_0, \tau_0, AP)$ is a subsystem specification of $\mathscr{S}$ is clear; let us now pick another subsystem specification $\mathscr{S}' = (\Sigma_0, \mathbf{v}_0, \Phi', \iota', \tau', AP)$ of $\mathscr{S}$ inducing a projection simulation over the same subsignature Σ_0 and the same sub-tuple of variables $\mathbf{v}_0$. According to (2), we have

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

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 $\mathscr{T}^{\mathscr{S}_0} \leq \mathscr{T}^{\mathscr{S}_0'}$. □

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 $\exists \mathbf{v}_1 \Phi, \exists \mathbf{v}_1 \iota, \exists \mathbf{v}_1 \exists \mathbf{v}_1' (\Phi \wedge \tau)$.

3.1 Counter Abstractions for Parameterized Systems

We now give a closer look at the signatures we need for modeling parameterized systems (i.e. systems composed by a finite - but arbitrary! - number of indistinguishable processes). We 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, $0, 1, +, -, =, <, \equiv_n$);

(ii) the enumerated truth value sort Ω, with the constants $\mathtt{tt}, \mathtt{ff}$ and the Boolean operations on them;

(iii) a finite sort `Proc`, whose cardinality is constrained to be equal to the arithmetic parameter `N` (this sort models the processes - all identical to each other - taking part in our parameterized system as actors); equality is the only predicate/function symbol defined on this sort;

(iv) further enumerated sorts `Data`, modeling local status, local flags, etc.

The subsignature Σ_0 comprising only the items (i)-(ii) above is called the *arithmetic subsignature* of Σ; the subsignature Σ_2 comprising only the items (ii) and (iv) above is called the *data subsignature* of Σ. 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 *enumerated* and *arithmetic array-ids*: an enumerated array-id is a variable of type $\mathtt{Proc} \to \mathtt{Data}$ and an arithmetic array-id is a variable of type $\mathtt{Proc} \to \mathbb{Z}$.

Let now $\mathscr{S} = (\Sigma, \mathbf{v}, \Phi, \iota, \tau, AP)$ be a system specification based on the above signature Σ. The variables $\mathbf{v}$ of $\mathscr{S}$ include some integer variables $\mathbf{v}_0$ and in addition variables for arithmetic and enumerated arrays-ids. Let us suppose that $\mathbf{v} = \mathbf{v}_0\mathbf{v}_1$, where $\mathbf{v}_1$ is the tuple of array 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 $\mathscr{S} = (\Sigma, \mathbf{v}_0\mathbf{v}_1, \Phi, \iota, \tau, AP)$ be as above. A *counter abstraction* of $\mathscr{S}$ is a subsystem specification of the kind $\mathscr{S}_0 = (\Sigma_0, \mathbf{v}_0, \Phi_0, \iota_0, \tau_0, AP)$; counter abstractions are ordered according to the ordering of the simulations of $\mathscr{S}$ they produce, i.e. we say that $\mathscr{S}_0$ is stronger than $\mathscr{S}_0'$ iff $\mathscr{T}^{\mathscr{S}_0} \leq \mathscr{T}^{\mathscr{S}_0'}$. We are interested in sufficient conditions on Φ, ι, τ ensuring the existence of a strongest counter abstraction. We describe below the sufficient conditions for which we have a first implementation (for stronger conditions, requiring heavier machinery, see [19]).

Below we use notations like $\phi(\underline{x}), t(\underline{x}), \ldots$ to mean that the formula ϕ, the term $t, \ldots$ contains at most the free variables in the tuple $\underline{x}$; notice also that, since there are no operation symbols defined on the sort `Proc`, all `Proc`-atoms [5] must be equalities between `Proc`-variables; for the same reasons, all subterms involving arrays-ids are flat, i.e. must be of the kind $a(i)$ where i is a variable of sort `Proc`. Since `Data` is enumerated, all `Data`-atoms must be of the kind $a(i) = b(k)$ or $a(k) = \mathtt{a_i}$, where a, b are enumerated arrays-ids, i, k are `Proc`-variables, and $\mathtt{a_i}$ is a constant for a value of type `Data`.[6] We call `Data`-*formula* a Boolean combination of `Data`-atoms; we also call *extended arithmetic term* a term of type $\mathbb{Z}$ which is an arithmetic parameter, a numeral, an arithmetic variable, a term of the kind $a(i)$ (where a is an arithmetic array-id and i a `Proc`-variable) or a term of the kind $\sharp\{k \mid \psi(k)\}$, where $\psi(k)$ is a `Data`-formula in which only the single `Proc`-variable k occurs. An *extended arithmetic atom* is a

[5] By a `Proc`-atom (resp. `Data`-atom) we mean an atomic formula whose root predicate is applied to terms denoting an element of sort `Proc` (resp. `Data`).

[6] Atoms of the kind $\mathtt{a_i} = \mathtt{a_j}$ are equivalent to `ff` or to `tt` because enumerated values are assumed to be distinct.

formula obtained from extended arithmetic terms by applying to them the arithmetic operations $+, -$ and the arithmetic predicates $=, <, \leq, \equiv_n$.

Theorem 1. *The system specification $\mathscr{S} = (\Sigma, \mathbf{v}_0\mathbf{v}_1, \Phi, \iota, \tau, AP)$ has a strongest (computable) counter abstraction in case Φ, ι, τ are disjunctions of formulæ of the kind*

$$\forall i\, \phi(i) \tag{3}$$

where $\phi(i)$ is a Boolean combination of Data*-atoms and of extended arithmetic atoms (both containing just the* Proc*-variable i).*

Proof. In view of Proposition 2, it is sufficient to show that if $a_1, \ldots, a_n$ are array-ids and $\forall i\, \phi(i)$ is a formula like (3), then $\exists a_1 \cdots \exists a_n \forall i \phi(i)$ is equivalent to a formula in pure Presburger arithmetic.

We first show how to eliminate an existential arithmetic array-id quantifier $\exists a$. This is eliminated (in favour of an extra existentially quantified arithmetic variable) by reverse skolemization [6, 31]: one observes that $\exists a \forall i\, \phi(i)$ is equivalent to the formula $\forall i \exists x\, \phi(i, x/a(i))$ (see the above observation about the 'flatness' of array-ids terms). Then the extra arithmetic existentially quantified variables introduced above are eliminated via Presburger quantifier elimination (notice that they do not occur inside Data-atoms or inside abstraction like terms $\sharp\{k \mid \psi(k)\}$, because ψ is a Data-formula).

Finally, enumerated array-ids quantifiers can be eliminated using the methods of [5]. Alternatively, since only arithmetic variables and enumerated array-ids are left at this point, it is also possible to make a BAPA-encoding and to use the quantifier elimination procedure for BAPA [29]. Such encoding can be obtained as follows. Notice that Data-atoms involving only the variable j can be written as $a(j) = \mathtt{a_k}$ for some enumerated value $\mathtt{a_k}$;[7] thus if we introduce set variables $S_{a,\mathtt{a_k}}$ for the sets $\{j \mid a(j) = \mathtt{a_k}\}$, we can write the terms $\{j \mid \psi(j)\}$ as Boolean combinations of these set variables $S_{a,\mathtt{a_k}}$. Finally, if $\psi(i)$ is a Boolean combination of Data-atoms and of extended arithmetic atoms without arithmetic array-ids, extended arithmetic atoms can be abstracted out of $\forall i\, \psi(i)$ by 'guessing' which of them hold (formally, we introduce a big disjunction, indexed by all Boolean assignments to such extended arithmetic atoms) and, when $\psi(i)$ is reduced to a Data-formula, $\forall i\, \psi(i)$ is equivalent to $\mathtt{N} = \sharp\beta$, where β is a Boolean combination of the $S_{a,\mathtt{a_k}}$ introduced above. □

4 An Example

In this section, we show how to apply the four-step methodology presented in Subsection 1.1 to a concrete problem. All results below have been certified via our prototype ARCA_SIM explained in Section 5 below.

The One-Third (OT) algorithm is designed to reach agreement in presence of benign transient faults [8, 9, 12]; the specification is reported in Algorithm 1. The protocol is supposed to work with an unlimited number of failures, but failures are supposed to be transient (processes may behave correctly in some rounds and not correctly in other rounds) and benign (processes, if they send any value, they send their own real value - which might or might not be received by the others - i.e. no fake value is sent or received). To be able to apply our techniques, *we need the extra assumption that the value to be agreed on is taken from a finite preassigned set* - let it be $\{0, 1\}$ for simplicity. We apply our four-steps plan.

Step (1): *we produce a formalization in higher order logic.* We employ:

- an array-id $V : \mathtt{Proc} \longrightarrow \{0, 1\}$ ($V(x)$ is the value currently held by x);

[7] Atoms like $a(j) = b(j)$ can be eliminated via $\bigvee_k (a(j) = \mathtt{a_k} \wedge b(j) = \mathtt{a_k})$.

Algorithm 1 One-Third Algorithm:

Round k**:** each process executes the following
send *val* to all;
if received values from more than $2\mathtt{N}/3$ distinct processes
then set *val* to the smallest most often received value;
if more than $2\mathtt{N}/3$ received values equal to *val*,
then accept *val*.

- an array-id $A : \mathtt{Proc} \longrightarrow \{\perp, 0, 1\}$ ($A(x)$ is the value accepted by x, initially $A(x) = \perp$);
- arithmetic array-ids R_0, R_1 ($R_0(x)$ is the number of 0-values received by x and $R_1(x)$ is the number of 1-values received by x).

We initialize the system using the following formula ι:

$$\mathtt{N} > 2 \ \wedge \ \forall x \, A(x) = \perp \tag{4}$$

(the assumption $\mathtt{N} > 2$ is not needed, but produces a more readable output). The transition relation is specified by the formula τ below:

$$\begin{array}{l} \forall i \, [0 \leq R_0'(i) \leq \sharp\{x \mid V(x) = 0\} \wedge 0 \leq R_1'(i) \leq \sharp\{x \mid V(x) = 1\}] \quad \wedge \\ \wedge \ \forall i \left[\begin{array}{l} (R_0'(i) + R_1'(i) > 2\mathtt{N}/3 \wedge R_1'(i) > R_0'(i) \wedge V'(i) = 1) \ \vee \\ \vee \ (R_0'(i) + R_1'(i) > 2\mathtt{N}/3 \wedge R_0'(i) \geq R_1'(i) \wedge V'(i) = 0) \ \vee \\ \vee \ (R_0'(i) + R_1'(i) \leq 2\mathtt{N}/3 \wedge V'(i) = V(i)) \end{array} \right] \quad \wedge \\ \wedge \ \forall i \left[\begin{array}{l} (R_1'(i) > 2\mathtt{N}/3 \wedge A'(i) = 1) \ \vee \\ \vee \ (R_0'(i) > 2\mathtt{N}/3 \wedge A'(i) = 0) \ \vee \\ \vee \ (R_0'(i) \leq 2\mathtt{N}/3 \ \wedge \ R_1'(i) \leq 2\mathtt{N}/3 \wedge A'(i) = A(i)) \end{array} \right] \end{array}$$

As usual, the primed variables R_0', R_1', V', A' denote the updated values of the arrays R_0, R_1, V, A (the arrays R_0, R_1 actually do not occur in τ, because the update of the local status of the processes only depends on the messages received in the current round - and the numbers of such messages are stored in R_0', R_1'). Notice that the formula τ matches the syntactic requirements of Theorem 1 (just swap the universal quantifier $\forall i$ and the conjunctions).

Step (2): *we manually add counters to our specification*. We introduce six counters, namely

$$\begin{array}{ll} z_{00} = \sharp\{i \mid A(i) = 0 \wedge V(i) = 0\}, & z_{10} = \sharp\{i \mid A(i) = 1 \wedge V(i) = 0\}, \\ z_{\perp 0} = \sharp\{i \mid A(i) = \perp \wedge V(i) = 0\}, & z_{01} = \sharp\{i \mid A(i) = 0 \wedge V(i) = 1\}, \\ z_{11} = \sharp\{i \mid A(i) = 1 \wedge V(i) = 1\}, & z_{\perp 1} = \sharp\{i \mid A(i) = \perp \wedge V(i) = 1\} \end{array}$$

(notice that the counters z_{10}, z_{01} will in fact be constantly equal to 0 during a system run, but we do not assume that this is known in advance). The formulæ ι, τ are modified as follows

$$\iota^+ :\equiv \ \iota \ \wedge \ \delta, \qquad \tau^+ :\equiv \ \tau \ \wedge \ \delta \ \wedge \delta', \tag{5}$$

where the auxiliary formulæ δ, δ' are the counters definitions supplied in Figure 1. Our system specification

$$\mathscr{S} = (\Sigma, \{V, A, R, z_{00}, z_{10}, z_{\perp 0}, z_{01}, z_{11}, z_{\perp 1}\}, \Phi, \iota^+, \tau^+)$$

is now complete (we do not need any invariant, so we take Φ to be $\top$).

Step (3): using the quantifier elimination procedure of Theorem 1, *we get rid of higher order variables and we compute the projected system*

$$\mathscr{S}_0 = (\Sigma_0, \{z_{00}, z_{10}, z_{\perp 0}, z_{01}, z_{11}, z_{\perp 1}\}, \Phi_0, \iota_0, \tau_0) \ .$$

We give the result produced by ARCA_SIM, with some hand-made manipulations aiming at making the output more human-readable (all such manipulations are up to logical equivalence). We use the auxiliary formulæ from Figure 1. The formula Φ_0 turns out to be $\top$, whereas ι_0 is

$$\psi \wedge z_{00} = 0 \wedge z_{01} = 0 \wedge z_{10} = 0 \wedge z_{11} = 0 \wedge \mathtt{N} = z_{\perp 0} + z_{\perp 1} \wedge \mathtt{N} > 2 \ .$$

In order to introduce τ_0, we need some extra notation. If u is an arithmetic term involving our counters, we let u' be the same term in which all counter variables are primed; we let also $\Delta(u)$ be $u' - u$ and $Decr(u)$ (resp. $Incr(u)$) be $\Delta(u) \leq 0$ (resp. $\Delta(u) \geq 0$). Thus, for instance, $\Delta(z_{00})$ is $z'_{00} - z_{00}$ and $Incr(z_{00})$ is $z'_{00} - z_{00} \geq 0$. Now τ_0 is the conjunction of $\psi \wedge \psi'$ (where ψ and ψ' are as defined in Figure 1) with the disjunction of the 7 formulæ below:

$$\neg p_0 \wedge \neg p_1 \wedge \neg p_2 \wedge p_3 \wedge Incr(z_{00}) \wedge Incr(z_{\perp 0}) \wedge Incr(z_{10}) \wedge$$
$$\wedge \Delta(z_{00} + z_{01}) = 0 \wedge \Delta(z_{10} + z_{11}) = 0 \wedge \Delta(z_{\perp 0} + z_{\perp 1}) = 0$$

$$\neg p_0 \wedge \neg p_1 \wedge p_2 \wedge \neg p_3 \wedge Decr(z_{00}) \wedge Decr(z_{\perp 0}) \wedge Decr(z_{10}) \wedge$$
$$\wedge \Delta(z_{00} + z_{01}) = 0 \wedge \Delta(z_{10} + z_{11}) = 0 \wedge \Delta(z_{\perp 0} + z_{\perp 1}) = 0$$

$$\neg p_0 \wedge \neg p_1 \wedge p_2 \wedge p_3 \wedge \Delta(z_{00} + z_{01}) = 0 \wedge \Delta(z_{10} + z_{11}) = 0 \wedge$$
$$\wedge \Delta(z_{\perp 0} + z_{\perp 1}) = 0$$

$$\neg p_0 \wedge p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge Incr(z_{00}) \wedge Decr(z_{01}) \wedge Decr(z_{10}) \wedge$$
$$\wedge Decr(z_{11}) \wedge Decr(z_{\perp 0}) \wedge Decr(z_{\perp 1})$$

$$\neg p_0 \wedge p_1 \wedge \neg p_2 \wedge p_3 \wedge Decr(z_{\perp 1}) \wedge Decr(z_{01}) \wedge Decr(z_{11}) \wedge$$
$$\wedge Decr(z_{10} + z_{11}) \wedge Incr(z_{00} + z_{01} + z_{10} + z_{11})$$

$$p_0 \wedge \neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge Decr(z_{00}) \wedge Decr(z_{01}) \wedge Decr(z_{10}) \wedge$$
$$\wedge Decr(z_{\perp 0}) \wedge Decr(z_{\perp 1})$$

$$p_0 \wedge \neg p_1 \wedge p_2 \wedge \neg p_3 \wedge Decr(z_{00}) \wedge Decr(z_{10}) \wedge Decr(z_{\perp 0}) \wedge$$
$$\wedge Decr(z_{00} + z_{01}) \wedge Decr(z_{\perp 0} + z_{\perp 1})$$

Step (4): *we express the safety properties we are interested in using our projected counters and we use an SMT-based tool to check them.* The relevant properties are agreement, weak validity and irrevocability (see Table 1). *Agreement* can be formalized with our counters by saying that the system

$$
\begin{array}{ll}
\delta: \equiv & z_{00} = \sharp\{i \mid A(i)=0 \wedge V(i)=0\} \wedge z_{10} = \sharp\{i \mid A(i)=1 \wedge V(i)=0\} \wedge \\
& \wedge z_{\perp 0} = \sharp\{i \mid A(i)=\perp \wedge V(i)=0\} \wedge z_{01} = \sharp\{i \mid A(i)=0 \wedge V(i)=1\} \\
& \wedge z_{11} = \sharp\{i \mid A(i)=1 \wedge V(i)=1\} \wedge z_{\perp 1} = \sharp\{i \mid A(i)=\perp \wedge V(i)=1\} \\
\\
\delta': \equiv & z'_{00} = \sharp\{i \mid A'(i)=0 \wedge V'(i)=0\} \wedge z'_{10} = \sharp\{i \mid A'(i)=1 \wedge V'(i)=0\} \wedge \\
& \wedge z'_{\perp 0} = \sharp\{i \mid A'(i)=\perp \wedge V'(i)=0\} \wedge z'_{01} = \sharp\{i \mid A'(i)=0 \wedge V'(i)=1\} \\
& \wedge z'_{11} = \sharp\{i \mid A'(i)=1 \wedge V'(i)=1\} \wedge z'_{\perp 1} = \sharp\{i \mid A'(i)=\perp \wedge V'(i)=1\} \\
\\
\psi: \equiv & 0 \le z_{00} \le \mathtt{N} \wedge 0 \le z_{10} \le \mathtt{N} \wedge 0 \le z_{\perp 0} \le \mathtt{N} \wedge \\
& 0 \le z_{01} \le \mathtt{N} \wedge 0 \le z_{11} \le \mathtt{N} \wedge 0 \le z_{\perp 1} \le \mathtt{N} \wedge \\
& \mathtt{N} = z_{00} + z_{10} + z_{\perp 0} + z_{01} + z_{11} + z_{\perp 1} \\
\\
\psi': \equiv & 0 \le z'_{00} \le \mathtt{N} \wedge 0 \le z'_{10} \le \mathtt{N} \wedge 0 \le z'_{\perp 0} \le \mathtt{N} \wedge \\
& 0 \le z'_{01} \le \mathtt{N} \wedge 0 \le z'_{11} \le \mathtt{N} \wedge 0 \le z'_{\perp 1} \le \mathtt{N} \wedge \\
& \mathtt{N} = z'_{00} + z'_{10} + z'_{\perp 0} + z'_{01} + z'_{11} + z'_{\perp 1} \\
\\
p_0: \equiv & t < z_1 \\
p_1: \equiv & t < z_0 \\
p_2: \equiv & t+1 < 2z_1 \wedge 0 < z_0 \\
p_3: \equiv & t < 2z_0 \wedge 0 < z_1
\end{array}
$$

Figure 1: Auxiliary formulæ for OT (we let $t := \llcorner 2\mathtt{N}/3 \lrcorner$, $z_0 := z_{00} + z_{10} + z_{\perp 0}$ and $z_1 := z_{01} + z_{11} + z_{\perp 1}$).

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

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 the decision later.

Table 1: Properties to be certified for OT.

never reaches a status satisfying $z_{00} + z_{01} > 0 \wedge z_{10} + z_{11} > 0$. *Weak validity* can be tested by checking that the system never reaches a status satisfying $z_{10} + z_{11} > 0$, once initialized to $\iota \wedge z_{\perp 0} = \mathtt{N}$. *Irrevocability* cannot be fully expressed with our counters, but can be approximated by adding a switch S that is turned to $\mathtt{tt}$ as soon as we have $z_{00} > 0$ and then checking that the system cannot reach a status satisfying $S = \mathtt{tt} \wedge z_{00} = 0$. All the above problems can be formulated with a different choice of counters (we employed a maximum choice above); in all variants,[8] ARCA_SIM takes 1-2 seconds to produce the HORN SMT_LIB file for μZ and the latter solves the related fixpoint problem in at most half a second, see the experimental data in Section 5 below.

5 A First Implementation

We implemented the procedure of Theorem 1 in a prototype tool called ARCA_SIM. Such tool accepts system specifications matching the syntactic restrictions of Theorem 1 and produces as output a file in

[8] The expected obvious property that $z_{01} + z_{10}$ is always equal to 0 can also be checked by our tool combination.

the HORN SMT_LIB format, ready to be model-checked e.g. by μZ [24], 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 ARCA_SIM should first contain *declarations* for parameters, integer variables and arithmetic and 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 $z = \sharp\{k \mid \psi(k)\}$, where ψ is a data formula. The system *transition* is given as a single variable universally quantified disjunction of cases $\forall x \bigvee_i \tau_i$, where each τ_i is specified via a formula of the kind $\phi_{i1} \wedge \phi_{i2}$, where: (i) $\phi_{i1}(x)$ is a conjunction of extended arithmetic atoms (in such atoms, terms like $\sharp\{k \mid \psi(k)\}$ must have been replaced by the corresponding counters); (ii) $\phi_{i2}(x)$ is a Data-formula. The *initial* formula follows the same syntax as the transition formula (but only one case is allowed), whereas the formula expressing the (negation of the) *safety* property must be an arithmetic formula containing only counters, integer variables and parameters.

ARCA_SIM produces a file for μZ basically following the proof of Theorem 1; it uses a BAPA-quantifier elimination algorithm adapted to the shape of the formulæ arising from our benchmarks. More specifically, the tool proceeds as follows:

(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;

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

$$\bigwedge_i (z_i = \sharp\{k \mid \psi_i(k)\}) \wedge \alpha \wedge \forall k\, \theta(k) \tag{6}$$

where we have, besides the counter definitions $z_i = \sharp\{k \mid \psi_i(k)\}$, a Boolean assignment α (seen as a conjunction of literals) to the arithmetic atoms occurring in the problem, and a single-variable universally quantified Data-formula $\forall k\, \theta(k)$;

(iii) auxiliary counters are now introduced: we have one counter z_f for each function f associating values to enumerated array-ids (z_f counts the cardinality of the set $\{k \mid \bigwedge_a a(k) = f_a \wedge \bigwedge_a a'(k) = f_{a'}\}$); the previous counters are expressed as linear combinations of these new counters; in addition, in each disjunct (6), the universally quantified formula $\forall k \theta(k)$ is replaced by the equation $\mathtt{N} = \sum \varepsilon_f z_f$, where ε_f is 0 or 1 depending on whether the Data-formula defining 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 (6); the new counters are eliminated by quantifier elimination and the resulting formulæ give the disjuncts of the transition of the projected counter 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 ARCA_SIM 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 $z = t$).[9] Notice also that, in case a difficult integer quantifier elimination problem arises, shifting to the

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

(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 ARCA_SIM was prepared to make such a shifting to Fourier-Motzkin procedure, it never did 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 counters: future work should concentrate on improving heuristics here. Notice that, even in the case the user made an (exponentially expensive) maximum choice of counters, the counters we need in (iii) are even more, because the auxiliary counters in (iii) must take into consideration both the actual and the updated enumerated array-ids (by a 'maximum choice of counters' we mean the introduction of a counter for each of the sets $\{k \mid \bigwedge_a a(k) = f_a\}$, varying f among the functions associating values to enumerated array-ids).

Some Experiments. In this Subsection we report our first experiments; the related files, as well as ARCA_SIM executables are available at the following link:

`http://users.mat.unimi.it/users/ghilardi/arca/arcasim.zip` .

Unfortunately, for various reasons, the specifications for the tool ARCA we used in [5] for invariant checking and bounded model-checking are insufficient and not compatible with the specifications accepted by ARCA_SIM. We only analyzed three representative benchmarks: (i) the One-Third (OT) algorithm from [12], whose formalization is described in Section 4 above; (ii) the Byzantine Broadcast Primitive (BBP) algorithm from [33], whose formalization is described in [5], Section 7.3; (iii) the Send Receive Broadcast Primitive (SRBP) algorithm from [32], whose formalization is described in [5], Section 6. For each of these benchmarks, we checked the relevant properties mentioned in the literature (for OT also the emptyness of the sets counted by z_{01}, z_{10}, see Section 4).[10]

In the table below, we report the time employed by ARCA_SIM to produce the Horn SMT-LIB problem and the time employed by μZ to solve the latter problem. Timings are all in seconds. We used a PC equipped with Intel Core i7 processor and operating system Linux Ubuntu 16.04 (64 bits). We also tried (and included in the distribution) some buggy versions - taken from [5] - of the above algorithms; we obtained the expected `unsat` answer from μZ (with performances similar to those in the above table). Such `unsat` answers just mean that the system is 'possibly unsafe': they do not certify bugs, because our counters simulations are, in fact, just simulations. Sometimes, with a maximum choice of counters, it is possible to prove (only offline with the actual techniques) that we are in presence of a bisimulation of the original system and in this case an `unsat` answer reveals the real presence of a bug.

Algorithm	**Property**	ARCA_SIM **Time**	μZ **Time**	**Total Time**
SRBP [32]	Correctness	2.68	0.09	2.77
SRBP [32]	Unforgeability	2.73	0.06	2.79
SRBP [32]	Relay I	2.68	0.06	2.74
SRBP [32]	Relay II	2.72	0.03	2.32
BBP [33]	Correctness	3.20	0.03	3.23
BBP [33]	Unforgeability	3.23	0.07	3.30
BBP [33]	Relay I	3.21	0.02	3.23
BBP [33]	Relay II	3.21	0.13	3.34
OT [12]	Agreement	0.76	0.26	1.02
OT [12]	Weak Validity	0.76	0.02	0.78
OT [12]	Irrevocability	2.03	0.42	2.45
OT [12]	Empty Counters	0.24	0.11	0.35

[10] Relay properties are split into two safety properties, as explained in [5].

6 Conclusions

We introduced a technique for automatically building counter simulations: the technique consists in modeling system specifications in higher order logic, then in introducing counters for definable sets and finally in 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 method.

Secondly, the amount of human interaction we require is nevertheless very limited and *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.

A delicate point is related to the *syntactic limitations* we require on the formulæ describing system specifications (see the statement of Theorem 1): 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. In fact, Theorem 1 can be extended in various directions [19]; in particular, extensions covering specifications with formulæ containing an extra layer of existentially quantified variables of sort `Proc` cover classical benchmarks like those in [2] and look to be relatively easily implementable.

The *integration of the methodology explained in this paper with proof assistants* is another interesting challenge to be pursued; such integration could on one hand *double-check the invariants and the related proof certificates given by the SMT-solvers* and on the other hand *use counters invariants supplied by our techniques as lemmata* inside complex interactive verifications tasks.

To conclude, we mention some recent work on the verification of fault-tolerant distributed systems, starting with our own previous work. The additional original contributions with respect to our previous paper [4] and its journal version [5] are due to the fact that in this paper we moved from bounded model-checking and invariant checking to the much more challenging task of full model-checking via *invariant synthesis*. As discussed in [5] (Section 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. This is why we tried a different approach, via counter simulations.

Papers [25–27] represent a very interesting and effective research line (summarized in [27]), 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 [27] 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 [27] 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

(whenever possible) to safety or bounded model checking problems.

Paper [9] directly handles cardinality constraints for interpreted sets by employing specifically tailored abstractions and some incomplete inference schemata at the level of the decision procedures. Non-trivial 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 that they supply 'the best simulations' as stated in Theorem 1.

Paper [16] 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 [16] 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.

Finally, we mention the effort made by the interactive theorem proving community in formalizing and verifying fault-tolerant distributed algorithms (see e.g. [11]); such approach is a natural complement to ours.

References

[1] P. A. Abdulla, K. Cerans, B. Jonsson & Y.-K. Tsay (1996): *General Decidability Theorems for Infinite-State Systems*. In: *Proc. of LICS*, pp. 313–321, doi:10.1109/LICS.1996.561359.

[2] P. A. Abdulla, G. Delzanno, N. B. Henda & A. Rezine (2007): *Regular Model Checking Without Transducers*. In: *TACAS*, *LNCS* 4424, pp. 721–736, doi:10.1007/978-3-540-71209-1_56.

[3] F. Alberti, S. Ghilardi, A. Orsini & E. Pagani (2016): *Counter Abstractions in Model Checking of Distributed Broadcast Algorithms: Some Case Studies*. In: *Proc. CILC*, CEUR Proceedings, pp. 102–117. Available at `http://ceur-ws.org/Vol-1645/paper_4.pdf`.

[4] F. Alberti, S. Ghilardi & E. Pagani (2016): *Counting Constraints in Flat Array Fragments*. In: *Proc. IJCAR*, *Lecture Notes in Computer Science* 9706, pp. 65–81, doi:10.1007/978-3-319-40229-1_6.

[5] F. Alberti, S. Ghilardi & E. Pagani (2017): *Cardinality Constraints for Arrays (decidability results and applications)*. *Formal Methods in System Design*, doi:10.1007/s10703-017-0279-6. To appear.

[6] F. Alberti, S. Ghilardi & N. Sharygina (2015): *Decision Procedures for Flat Array Properties*. *Journal of Automated Reasoning* 54(4), pp. 327–352, doi:10.1007/s10817-015-9323-7.

[7] Peter B. Andrews (2002): *An introduction to mathematical logic and type theory: to truth through proof*, 2nd edition. *Applied Logic Series* 27, Kluwer Academic Publishers, Dordrecht, doi:10.1007/978-94-015-9934-4.

[8] M. Biely, B. Charron-Bost, A. Gaillard, M. Hutle, A. Schiper & J. Widder (2007): *Tolerating corrupted communication*. In: *Proc. PODC*, pp. 244–253, doi:10.1145/1281100.1281136.

[9] N. Bjørner, K. von Gleissenthall & A. Rybalchenko (2016): *Cardinalities and Universal Quantifiers for Verifying Parameterized Systems*. In: *Proc. of PLDI*, doi:10.1145/2980983.2908129.

[10] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri & S. Tonetta (2014): *The nuXmv Symbolic Model Checker*. In: *CAV*, pp. 334–342, doi:10.1007/978-3-319-08867-9_22.

[11] B. Charron-Bost, H. Debrat & S. Merz (2011): *Formal Verification of Consensus Algorithms Tolerating Malicious Faults*. In: *Stabilization, Safety, and Security of Distributed Systems*, Springer LNCS, pp. 120–134, doi:10.1007/978-3-642-24550-3_11.

[12] B. Charron-Bost & A. Schiper (2009): *The heard-of model: computing in distributed systems with benign faults*. *Distributed Computing*, pp. 49–71, doi:10.1007/s00446-009-0084-6.

[13] A. Cimatti & A. Griggio (2012): *Software model checking via IC3*. In: *CAV*, pp. 277–293, doi:10.1007/978-3-642-31424-7_23.

[14] G. Delzanno (2003): *Constraint-Based Verification of Parameterized Cache Coherence Protocols*. Formal Methods in System Design 23(3), pp. 257–301, doi:10.1023/A:1026276129010.

[15] G. Delzanno, J. Esparza & A. Podelski (1999): *Constraint-Based Analysis of Broadcast Protocols*. In: Proc. of CSL, LNCS 1683, pp. 50–66, doi:10.1007/3-540-48168-0_5.

[16] C. Dragoj, T. Henzinger, H. Veith, J. Widder & D. Zufferey (2014): *A Logic-based Framework for Verifying Consensus Algorithms*. In: Proc. of VMCAI, doi:10.1007/978-3-642-54013-4_10.

[17] J. Esparza, A. Finkel & R. Mayr (1999): *On the Verification of Broadcast Protocols*. In: Proc. of LICS, IEEE Computer Society, pp. 352–359, doi:10.1109/LICS.1999.782630.

[18] C. Flanagan & S. Qadeer (2002): *Predicate abstraction for software verification*. In: POPL, pp. 191–202, doi:10.1145/565816.503291.

[19] S. Ghilardi & E. Pagani (2017): *Second Order Quantifier Elimination: towards Verification Applications*. Technical Report. In preparation.

[20] S. Ghilardi & S. Ranise (2010): *Backward Reachability of Array-based Systems by SMT solving: Termination and Invariant Synthesis*. Logical Methods in Computer Science 6(4), doi:10.2168/LMCS-6(4:10)2010.

[21] S. Ghilardi & S. Ranise (2010): *MCMT: A Model Checker Modulo Theories*. In: IJCAR, pp. 22–29, doi:10.1007/978-3-642-14203-1_3.

[22] Arie Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli & Jorge A. Navas (2015): *The SeaHorn Verification Framework*. In: CAV, pp. 343–361, doi:10.1007/978-3-319-21690-4_20.

[23] K. Hoder & N. Bjørner (2012): *Generalized Property Directed Reachability*. In: SAT, pp. 157–171, doi:10.1007/978-3-642-31612-8_13.

[24] K. Hoder, N. Bjørner & L. deMoura (2011): *μZ– An Efficient Engine for Fixed Points with Constraints*. In: CAV, pp. 457–462, doi:10.1007/978-3-642-22110-1_36.

[25] A. John, I. Konnov, U. Schmid, H. Veith & J. Widder (2013): *Parameterized model checking of fault-tolerant distributed algorithms by abstraction*. In: Proc. FMCAD, pp. 201–209, doi:10.1109/FMCAD.2013.6679411.

[26] A. John, I. Konnov, U. Schmid, H. Veith & J. Widder (2013): *Towards Modeling and Model Checking Fault-Tolerant Distributed Algorithms*. In: Proc. SPIN , 7976, pp. 209–226, doi:10.1007/978-3-642-39176-7_14.

[27] I. Konnov, H. Veith & J. Widder (2015): *What You Always Wanted to Know About Model Checking of Fault-Tolerant Distributed Algorithms*. In: PSI , pp. 6–21, doi:10.1007/978-3-319-41579-6_2.

[28] I.. Konnov, H. Veith & J. Widder (2017): *On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability*. Inf. Comput. 252, pp. 95–109, doi:10.1007/978-3-662-44584-6_10.

[29] Viktor Kuncak, Huu Hai Nguyen & Martin Rinard (2006): *Deciding Boolean Algebra with Presburger Arithmetic*. Journal of Automated Reasoning 36(3), doi:10.1007/s10817-006-9042-1.

[30] J. Lambek & P. J. Scott (1988): *Introduction to higher order categorical logic*. Cambridge Studies in Advanced Mathematics 7, Cambridge University Press, Cambridge.

[31] Andrew Reynolds, Morgan Deters, Viktor Kuncak, Cesare Tinelli & Clark W. Barrett (2015): *Counterexample-Guided Quantifier Instantiation for Synthesis in SMT*. In: Proc. CAV, pp. 198–216, doi:10.1007/978-3-319-21668-3_12.

[32] T.K. Srikanth & S. Toueg (1987): *Optimal Clock Synchronization*. Journal of the ACM 34(3), pp. 626–645, doi:10.1145/28869.28876.

[33] T.K. Srikanth & S. Toueg (1987): *Simulating authenticated broadcasts to derive simple fault-tolerant algorithms*. Distributed Computing 2(2), pp. 80–94, doi:10.1007/BF01667080.