Journal on Satisfiability, Boolean Modeling and Computation 8 (2012) 29-61

Universal Guards, Relativization of Quantifiers, and Failure Models in Model Checking Modulo Theories

Francesco Alberti francesco.alberti@usi.ch
Università della Svizzera Italiana,
via G. Buffi, 13,
CH-6904, Lugano

Silvio Ghilardi ghilardi@dsi.unimi.it
Elena Pagani pagani@dico.unimi.it
Department of Computer Science, Università degli Studi,
via Comelico 35,
I-20135 Milano

Silvio Ranise ranise@fbk.eu
FBK (Fondazione Bruno Kessler),
Povo - Via Sommarive 18
I-38123 Trento

Gian Paolo Rossi rossi@dico.unimi.it
Department of Computer Science, Università degli Studi,
via Comelico 35,
I-20135 Milano

Abstract

Model Checking Modulo Theories is a recent approach for the automated verification of safety properties of a class of infinite state systems manipulating arrays, called array-based systems. The idea is to repeatedly compute pre-images of a set of (unsafe) states by using certain classes of first-order formulae representing sets of states and transitions, and then reduce fix-point checks to Satisfiability Modulo Theories problems. Unfortunately, if the guards contain universally quantified index variables, the backward procedure cannot be fully automated. In this paper, we overcome the problem by describing a syntactic transformation on array-based systems, which can be seen as an instance of the well-known operation of relativization of quantifiers in first-order logic. Interestingly, when specifying and verifying distributed systems, the proposed syntactic transformation can be interpreted as the adoption of the crash-failure model, which is well-known in the literature of fault-tolerant systems. By eliminating universal quantifiers from guards, the transformation significantly extends the scope of applicability of the symbolic backward reachability procedure. To provide empirical evidence of this claim, we discuss our findings in applying the proposed technique to a significant case-study in the verification of some classical algorithms for reliable broadcast.

Keywords: *model checking modulo theories, failure models, quantifiers instantiation, reliable broadcast*

Submitted February 2011; revised February 2011; published January 2012

1. Introduction

In [15], a declarative approach to the specification and automated verification of a class of infinite state systems manipulating arrays, called *array-based systems*, is introduced. The key idea is to represent the algebraic structures of the indexes and the elements stored in the arrays by using theories and certain classes of first-order formulae for representing sets of states and transitions. On top of this, a symbolic backward reachability procedure can be automated by reducing tests for fix-point and safety to Satisfiability Modulo Theories (SMT) problems. Safety problems for several classes of systems and algorithms can be handled in this way; e.g., parameterized protocols for mutual exclusion or imperative algorithms manipulating arrays (see, e.g., [17, 13, 15]).

In [17], it is realized that formulae representing transitions of array-based systems can be restricted to, so-called, *guarded assignments*, i.e. array variables are updated according to a certain function provided that a guard is satisfied. Guards are expressed as first-order formulae involving array state variables. Usually guards can be expressed as conjunctions of two conditions: one involving a limited number of indexes of array variables, called *local*, and another one universally quantifying over the (finite but unbounded) domain of indexes, called *global*. When the guard is purely local (i.e. the global condition is equivalent to true), it is possible to show that the class of formulae used to represent sets of states is closed under pre-image computation (see, e.g., [17]). This is one of the crucial requirements for the automation of the symbolic backward reachability procedure used to solve safety problems. Unfortunately, in several situations, the global condition must be considered and the closure under pre-image is lost. Indeed, this severely restricts the scope of applicability of verification procedures based on symbolic backward reachability as described in [15]. To overcome this problem, in this paper, we provide a fully formal account of a purely syntactic transformation, its main properties relating the set of possible executions of the original and the transformed system, as well as its implementation in MCMT, acronym for Model Checker Modulo Theories [16]. We also discuss in details its relationship with the crash-failure model (in which, a process may halt at any moment and cannot recover from its crashed state) developed in the context of distributed systems (see, e.g., [22, 27]), when array-based systems model fault-tolerant algorithms. We also sketch how more complex failure models (e.g., the send-omission failure model [27]) can be formalized in our framework.

The main source of inspiration for designing our transformation is the work on monotonic abstraction [3, 4, 5, 1] developed by Abdulla *et al*, which is more semantic and needs to be adapted each time a new of class of systems is to be verified. On the contrary, our transformation is purely syntactic and may be applied to arbitrary array-based systems, even those not modelling distributed systems, and put to productive work for verification problems where a semantic approach *à la* Abdulla *et al* would be much more difficult to adapt. For example, we used our transformation in the verification of imperative programs manipulating arrays or strings in [16, 15]. So, the work described in this paper can be seen as the reinterpretation and generalization of Abdulla *et al* monotonic abstraction in the fully declarative framework of model checking modulo theories. This has been anticipated in [17] where we informally describe a syntactic transformation of array-based systems with global conditions in the guards of transitions so that if the transformed system is proved safe with respect to a reachability property, then also the original system is so. Along the lines of [17], the idea underlying the transformation described here can be informally illustrated as follows. Consider a parametric system formed by identical processes which are finite state automata. Each automaton has a finite set $Q = \{q_1, ..., q_n\}$ of control locations. Let us extend Q to a set $Q' = Q \cup \{q_{crash}\}$, where $q_{crash} \notin Q$, and augment the set of transitions of each process as follows: it is always possible to go from state q_i to q_{crash}, for each $i = 1, ..., n$ (this formalizes the fact that in the crash-failure model, a process can halt without warning at any time).[1.] Now, consider a transition with a global condition saying that a process i can execute a transition if a certain predicate C is satisfied by *all* processes $j \neq i$. In the crash-failure model, this can be expressed without the universal quantification as follows: the *non-crashed* process i takes the transition without checking the global condition C and, concurrently, all *non-crashed* processes $j \neq i$ not satisfying the condition C move to the state q_{crash}; moreover, all *non-crashed* processes $j \neq i$ satisfying C behave as originally prescribed. The key observation is that in the transformed system all the processes involved in the transition are required to be non-crashed. Formally, this can be expressed by a standard syntactic transformation known in the literature about first-order logic as *relativization of quantifiers* (see, e.g., [12]). By eliminating global conditions, the transformed array-based system can be automatically analyzed by the model checking modulo theories procedure. Crucially, since it is possible to prove that the transformed system satisfies a subset of the class of safety (or even recurrence) properties satisfied by the original system (since the latter has fewer runs), establishing a safety property for the system in the crash-failure model *implies* that the same property is fulfilled by the original system.

Given the connection between our syntactic transformation and the crash-failure model, we have found it natural to consider the verification of an important class of parameterized fault-tolerant algorithms to test the usefulness of our approach. Reliable broadcast algorithms are such a class and we consider some of the classical algorithms developed in [28]. In particular, our experimental results show that it is possible to formalize the step-wise specification and verification of an important safety property (called *agreement* in the literature) confirming the pen-and-paper proofs in [28]. For scalability, we found the use of invariants crucial for pruning of the search space of the backward reachability procedure, as discussed in [13, 15]. Indeed, when considering the syntactically transformed array-based system for verification, also the invariants to be employed should be suitably transformed. We show that this is particularly easy in our setting. We believe that the experiments reported in this paper (together with those already discussed in [15]) show that our technique, despite its conceptual simplicity, significantly widens the scope of applicability of our symbolic reachability procedure based on SMT solving.

Plan of the paper. Section 2 introduces selected notions from [17, 13, 15]. Sections 3 and 4 give an overview of the specification and automated analysis of safety problems of array-based systems. Section 5 describes the syntactic transformations to eliminate global conditions from the guards of the transition, formally states and proves its main properties, and discusses how they affect invariants and their use in pruning the search

1. An alternative (equivalent) solution would be to keep the original set Q of control locations and add a Boolean flag saying that a given process is crashed or not. This will be the solution adopted in this paper.

space of the backward reachability procedure. Section 6 considers the interpretation of the transformation as the adoption of the crash-failure model and compares our verification technique with others available in the literature, such as the monotonic abstraction of Abdulla *et al.* Section 7 describes the implementation and integration of the syntactic transformation in MCMT and discusses our experiments with the verification of some of the algorithms for reliable broadcast in [28]. Section 8 concludes. A simple protocol for mutual exclusion taken from [1] is used as a running example throughout the paper to illustrate the various notions. The material concerning the case study of Section 7 has appeared in preliminary form in [7, 6], the remaining content of the paper is published for the first time.

How to read the paper. Readers interested only in the transformation to handle global conditions in the guards of transitions can focus on Sections 2, 3, and 5; although Sections 2 and 3 can be omitted if already familiar with the formal framework underlying MCMT. Particular attention should be paid to Table 2 that contains the formal definition of the transformation together with Sections 5.2 and 5.3 that prove the key properties of the transformation and Table 1 that reports the assumptions under which the properties are proved. Those readers interested in understanding the relationship between the proposed transformation and the crash-failure model of distributed systems should also read Section 6. The specification of more complex failure models is discussed in Section 7.3. Finally, those interested in the implementation and evaluation of the technique may also read Section 7.

2. Formal Preliminaries

We assume the usual syntactic (e.g., signature, variable, term, atom, literal, and formula) and semantic (e.g., structure, sub-structure, assignment, truth, satisfiability, and validity) notions of many-sorted first-order logic (see, e.g., [12]). The equality symbol $=$ is included in all signatures considered below. If $\mathcal{M}$ is a structure for a signature Σ (briefly, a Σ-structure), we denote by $S^{\mathcal{M}}, f^{\mathcal{M}}, P^{\mathcal{M}}, \ldots$ the interpretation in $\mathcal{M}$ of the sort S, the function symbol f, the predicate symbol P, etc. If Σ_0 is a sub-signature of Σ, the structure $\mathcal{M}_{|\Sigma_0}$ results from $\mathcal{M}$ by forgetting about the interpretation of the sorts, function and predicate symbols that are not in Σ_0 and $\mathcal{M}_{|\Sigma_0}$ is called the *restriction* of $\mathcal{M}$ to Σ_0.

According to the current practice in the SMT literature [25], a *theory* T is a pair $(\Sigma, \mathcal{C})$, where Σ is a signature and $\mathcal{C}$ is a class of Σ-structures; the structures in $\mathcal{C}$ are the *models* of T. Below, we let $T = (\Sigma, \mathcal{C})$. A Σ-formula ϕ is *T-satisfiable* if there exists a Σ-structure $\mathcal{M}$ in $\mathcal{C}$ such that ϕ is true in $\mathcal{M}$ under a suitable assignment to the free variables of ϕ (in symbols, $\mathcal{M} \models \phi$); it is *T-valid* (in symbols, $T \models \varphi$) if its negation is T-unsatisfiable. Two formulae φ_1 and φ_2 are *T-equivalent* if $\varphi_1 \Leftrightarrow \varphi_2$ is *T-valid*. The *satisfiability modulo the theory T ($SMT(T)$) problem* amounts to establishing the T-satisfiability of quantifier-free (i.e. not containing quantifiers) Σ-formulae.

A *T-partition* is a finite set $C_1(\underline{x}), \ldots, C_n(\underline{x})$ of quantifier-free formulae (with free variables contained in the tuple $\underline{x}$) such that $T \models \forall \underline{x} \bigvee_{i=1}^{n} C_i(\underline{x})$ and $T \models \bigwedge_{i \neq j} \forall \underline{x} \neg (C_i(\underline{x}) \wedge C_j(\underline{x}))$. A *case-definable extension* $T' = (\Sigma', \mathcal{C}')$ of a theory $T = (\Sigma, \mathcal{C})$ is obtained from T by applying (finitely many times) the following procedure: (i) take a T-partition $C_1(\underline{x}), \ldots, C_n(\underline{x})$ together with Σ-terms $o_1(\underline{x}), \ldots, o_n(\underline{x})$; (ii) let Σ' be $\Sigma \cup \{F\}$, where F is a "fresh" function symbol (i.e. $F \notin \Sigma$) whose arity is equal to the length of $\underline{x}$; (iii) take as

$\mathcal{C}'$ the class of Σ'-structures $\mathcal{M}$ whose restriction to Σ is a model of T and such that

$$\mathcal{M} \models \bigwedge_{i=1}^{n} \forall \underline{x}\, (C_i(\underline{x}) \Rightarrow F(\underline{x}) = o_i(\underline{x})).$$

Thus a case-definable extension T' of a theory T contains finitely many additional function symbols, called *case-defined functions*. It is not hard to effectively translate any $SMT(T')$ problem into an equivalent $SMT(T)$-problem, see [15] for details. In the following, by abuse of notation, we shall identify a theory T and its case-definable extensions T'. In the rest of the paper, a case-defined function F will be written as follows:

$$F(\underline{x}) \ := \ \texttt{case of} \ \ \{\ C_1(\underline{x}) \ : \ o_1; \cdots \ C_n(\underline{s}) \ : \ o_r \ \},$$

where $C_1, ..., C_n$ is a T-partition for a suitable theory T. When $n = 2$, the partition has the form $C(\underline{x}), \neg C(\underline{x})$ for some formula C so that we will write $F(\underline{x}) \ := \ \texttt{if}\ C(\underline{x})\ \texttt{then}\ o_1\ \texttt{else}\ o_2$. To improve readability, we will use λ-abstractions to define functions; although, we emphasize that all expressions can be rewritten in pure first-order logic.

3. Array-Based systems

The theory A_I^E specifies the array data structure manipulated by the class of transition systems considered in the paper. It is parametric with respect to the indexes and elements stored in the arrays, whose algebraic structures are again specified as theories T_I and T_E, respectively. We assume $T_I = (\Sigma_I, \mathcal{C}_I)$ to have only one sort symbol $\texttt{INDEX}$. The sorts of the theory $T_E = (\Sigma_E, \mathcal{C}_E)$ are given names $\texttt{ELEM}_\ell$, where ℓ varies in a given finite index set. We define the composed theory $A_I^E = (\Sigma, \mathcal{C})$ of arrays with indexes in T_I and elements in T_E as follows. The signature of A_I^E contains the sort symbols of T_I, T_E, together with a new sort symbol $\texttt{ARRAY}_\ell$ for each $\texttt{ELEM}_\ell$ of Σ_E, and all the function and predicate symbols in $\Sigma_I \cup \Sigma_E$ together with a new function symbol $\_[\_]_\ell \ : \ \texttt{ARRAY}_\ell, \texttt{INDEX} \longrightarrow \texttt{ELEM}_\ell$ for each $\texttt{ELEM}_\ell$ of Σ_E. Intuitively, $a[i]_\ell$ denotes the element of sort $\texttt{ELEM}_\ell$ stored in the array a of sort $\texttt{ARRAY}_\ell$ at index i; when the sort $\texttt{ELEM}_\ell$ is clear from the context, we simply write $a[i]$. The class $\mathcal{C}$ of models of A_I^E contains a multi-sorted structure $\mathcal{M}$ iff for each sort $\texttt{ELEM}_\ell$ of Σ_E, we have that $\texttt{ARRAY}_\ell^{\mathcal{M}}$ is interpreted as the set $[\texttt{INDEX}^{\mathcal{M}} \to \texttt{ELEM}_\ell^{\mathcal{M}}]$ of (total) functions from $\texttt{INDEX}^{\mathcal{M}}$ to $\texttt{ELEM}_\ell^{\mathcal{M}}$, the function symbol $\_[\_]$ is interpreted as function application, and $\mathcal{M}_{|\Sigma_I}, \mathcal{M}_{|\Sigma_E}$ are models of T_I and T_E, respectively.

An *array-based (transition) system (for (T_I, T_E))* is a triple $\mathcal{S} = (a, I, \tau)$ where (i) $a = a_1, \ldots, a_s$ is the tuple of the array *state variables* (these arrays encode local data of sorts $\texttt{ELEM}_1, \ldots, \texttt{ELEM}_s$, respectively); (ii) $I(a)$ is the *initial* formula; and (iii) $\tau(a, a')$ is the *transition* formula. Notice that in $\tau(a, a')$ not only the tuple a occurs but also its renamed copy a': this is because the transition formula τ is meant to express a constraint relating the values of the state variables before and after the execution of τ (usually, such a constraint is a conditional update).

Given an array-based system $\mathcal{S} = (a, I, \tau)$ and a formula $U(a)$, (an instance of) the *safety problem* is to establish whether there exists a natural number n such that the formula

$$I(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge U(a_n) \qquad (1)$$

is A_I^E-satisfiable. If there is no such n, then $\mathcal{S}$ is *safe* (w.r.t. U); otherwise, it is *unsafe* and there exists a run (i.e. a sequence of transitions) of length n leading the system from a state in I to a state in U.

Array-based systems have been used to model a variety of systems coming from several verification domains; the interested reader is pointed to [15] for an overview.

3.1 Running Example

We consider a protocol (taken from [1]) ensuring mutual exclusion for an arbitrary number of processes with a linear topology. Each process has four control locations: I(dle), R(equesting), W(aiting), and C(ritical section). The processes are linearly ordered: the set of processes on the left (right) of a given process p are those processes coming before (after, respectively) p in the linear order. Initially, all processes are in control location I. Each process can perform the following transitions:

t_1: if a process is in location I and all other processes (on its left and right) are either in location I or R, then it can move to location R;

t_2: if a process is in location R, then it can move to location W;

t_3: if a process is in location W and all the processes on its left are in location I, then it can move to location C;

t_4: if a process is in location C, then it can move to location R; and

t_5: if a process is in location R, then it can move to location I.

All those (bad) states containing at least two distinct processes in the control location C violate the mutual exclusion property that the protocol is intended to guarantee.

To formalize this protocol as an array-based system, we start defining the theories over indexes and elements. Let T_I be the theory of linear orders, which contains the binary predicate symbol $<$ (written infix) interpreted as a transitive, antisymmetric, and total relation over a set of elements. Let T_E be the theory of an enumerated data type containing only the constant symbols I, R, W, and C whose interpretations are distinct elements of a set containing just those four elements (thus the class of models of T_E is a singleton). Then, we consider an array-based system with only one state variable a, the set of initial states is characterized by

$$I(a) := \forall z.a[z] = \mathsf{I},$$

and the transitions are formalized by

$$\begin{array}{lcll}
t_1(a,a') & := & \exists i.(a[i] = \mathsf{I} \wedge \forall j.(j \neq i \Rightarrow a[j] \in \{\mathsf{I},\mathsf{R}\})\wedge & a' = \mathtt{upd}(a,i,\mathsf{R})) \\
t_2(a,a') & := & \exists i.(a[i] = \mathsf{R}\wedge & a' = \mathtt{upd}(a,i,\mathsf{W})) \\
t_3(a,a') & := & \exists i.(a[i] = \mathsf{W} \wedge \forall j.(j < i \Rightarrow a[j] = I)\wedge & a' = \mathtt{upd}(a,i,\mathsf{C})) \\
t_4(a,a') & := & \exists i.(a[i] = \mathsf{C}\wedge & a' = \mathtt{upd}(a,i,\mathsf{R})) \\
t_5(a,a') & := & \exists i.(a[i] = \mathsf{R}\wedge & a' = \mathtt{upd}(a,i,\mathsf{I})),
\end{array}$$

where $t \in \{c_1, ..., c_n\}$ abbreviates $t = c_1 \vee \cdots \vee t = c_n$ for t a term and c_i a constant ($i = 1, ..., n$) and $\mathtt{upd}(a,i,c)$ abbreviates the case-defined function $\lambda j.(\mathtt{if}\ (j = i)\ \mathtt{then}\ c\ \mathtt{else}\ a[j])$

```
function BReach((a, I, τ), U)
1  $P \longleftarrow U$; $B \longleftarrow \bot$;
2  while ($P \wedge \neg B$ is $A_I^E$-sat.) do
3      if ($I \wedge P$ is $A_I^E$-sat.) then return unsafe;
4      $B \longleftarrow P \vee B$;
5      $P \longleftarrow Pre(\tau, P)$;
6  end
7  return (safe, $B$);
```

Figure 1. The symbolic backward reachability algorithm

for a, i, and c constants of appropriate sorts. So, the array-based system (a, I, τ) formalizes the protocol for mutual exclusion introduced above, where $\tau(a, a') := \bigvee_{\ell=1}^{5} t_\ell(a, a')$. The formula describing the set of bad states is

$$U(a) \quad := \quad \exists i, j.(i < j \wedge a[i] = \mathsf{C} \wedge a[j] = \mathsf{C}).$$

The safety problem for this protocol consists of establishing if there exists $n \geq 0$ such that

$$\forall z.a_0[z] = \mathsf{I} \wedge t_{\ell_1}(a_0, a_1) \wedge \cdots \wedge t_{\ell_n}(a_{n-1}, a_n) \wedge \exists i, j.(i \neq j \wedge a_n[i] = \mathsf{C} \wedge a_n[j] = \mathsf{C})$$

is A_I^E-satisfiable for $\ell_1, ..., \ell_n \in \{1, ..., 5\}$. Notice that for $n = 0$, we have that

$$\forall z.a_0[z] = \mathsf{I} \wedge \exists i, j.(i < j \wedge a_0[i] = \mathsf{C} \wedge a_0[j] = \mathsf{C})$$

is A_I^E-unsatisfiable since its quantifier-free instance (obtained by considering the existentially quantified variables i, j as Skolem constants and instantiating the universally quantified variable z with i and j):

$$a_0[i] = \mathsf{I} \wedge a_0[j] = \mathsf{I} \wedge i < j \wedge a_0[i] = \mathsf{C} \wedge a_0[j] = \mathsf{C},$$

is obviously A_I^E-unsatisfiable (to see why, consider the first and fourth, or the second and last, literals and derive $\mathsf{I} = \mathsf{C}$, which is T_E-unsatisfiable as I and C denote distinct elements in the model of T_E).

4. Backward Reachability for Array-based Systems

A general approach to solve instances of the safety problem is based on symbolically computing the set of backward reachable states. For $n \geq 0$, the *n-pre-image* of a formula $K(a)$ is $Pre^0(\tau, K) := K$ and $Pre^{n+1}(\tau, K) := Pre(\tau, Pre^n(\tau, K))$, where

$$Pre(\tau, K) \quad := \quad \exists a'.(\tau(a, a') \wedge K(a')). \qquad (2)$$

Given $\mathcal{S} = (a, I, \tau)$ and $U(a)$, the formula $Pre^n(\tau, U)$ describes the set of backward reachable states in n steps (for $n \geq 0$). At the n-th iteration of the loop, the *basic backward reachability algorithm*, depicted in Figure 1, stores in the variable B the formula

$BR^n(\tau, U) := \bigvee_{i=0}^{n} Pre^i(\tau, U)$ representing the set of states which are backward reachable from the states in U in at most n steps (whereas the variable P stores the formula $Pre^n(\tau, U)$). While computing $BR^n(\tau, U)$, BReach also checks whether the system is unsafe (cf. line 3, which can be read as $I \wedge Pre^n(\tau, U)$ is A_I^E-satisfiable) or a fix-point has been reached (cf. line 2, which can be read as $\neg(BR^n(\tau, U) \Rightarrow BR^{n-1}(\tau, U))$ is A_I^E-unsatisfiable or, equivalently, that $(BR^n(\tau, U) \Rightarrow BR^{n-1}(\tau, U))$ is A_I^E-valid). When BReach returns the safety of the system (cf. line 7), the variable B stores the formula describing the set of states which are backward reachable from U which is also a fix-point. For BReach (Figure 1) to be a true (possibly non-terminating) procedure, it is mandatory to identify a class of formulae for representing sets of (backward) reachable states (line 5), in such a way that the class is closed under pre-image computation and the safety (line 3) and fix-point (line 2) checks are effective.

In [15], we identified such a class of formulae and sufficient conditions on the component theories T_I and T_E to guarantee decidability of the satisfiability problem; we also identified the shapes of the formulae of an array-based transition system and of the formula representing the set of bad states to guarantee closure under pre-image computation. To precisely state these assumptions and the related results, we need to introduce the following notational convention and definitions: d, e range over variables of a sort $\mathtt{ELEM}_S$ of Σ_E, $i, j, k, z, \ldots$ over variables of sort $\mathtt{INDEX}$. An underlined variable name abbreviates a tuple of variables of unspecified (but finite) length and, if $\underline{i} := i_1, \ldots, i_n$, the notation $a[\underline{i}]$ abbreviates the $s * n$-tuple of terms

$$a_1[i_1], \ldots, a_1[i_n], \ldots, a_s[i_1], \ldots, a_s[i_n].$$

Notice that we use the underlined notation $\underline{i}, \underline{e}, \ldots$ for tuples of elements and index variables, whereas we use just a (not underlined!) for the tuple $a_1, \ldots, a_s$ of array variables: the reason for this asymmetric choice is due to the fact that the length s of the tuple a is fixed during model checking (it is given in the definition of an array-based system); whereas the length of the tuples $\underline{i}$ arising during backward reachability is finite but unknown. As a general rule, whenever we do not underline a variable name, we mean that it represents a single variable or a tuple of variables of fixed length s; on the contrary, whenever a variable name is underlined, it means that it represents a tuple of variables of unspecified length.

Possibly sub/super-scripted expressions of the form $\phi(\underline{i}, \underline{e}), \psi(\underline{i}, \underline{e})$ denote *quantifier-free* $(\Sigma_I \cup \Sigma_E)$*-formulae in which at most the variables* $\underline{i} \cup \underline{e}$ *occur*. Also, $\phi(\underline{i}, \underline{t}/\underline{e})$ (or simply $\phi(\underline{i}, \underline{t})$) abbreviates the substitution of the Σ-terms $\underline{t}$ for the variables $\underline{e}$. Thus, for instance, $\phi(\underline{i}, a[\underline{i}])$ denotes the formula obtained by replacing $\underline{e}$ with $a[\underline{i}]$ in the quantifier-free formula $\phi(\underline{i}, \underline{e})$. A $\forall^I$*-formula* is a formula of the form $\forall \underline{i}.\phi(\underline{i}, a[\underline{i}])$. An $\exists^I$*-formula* is a formula of the form $\exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$. An $\exists^{A,I}\forall^I$*-sentence* is a sentence of the form $\exists a \, \exists \underline{i} \, \forall \underline{j} \, \psi(\underline{i}, \underline{j}, a[\underline{i}], a[\underline{j}])$.

To mechanize BReach, the additional assumptions reported in Table 1 on the theories T_I and T_E, and the form of the formulae I, τ, and U specifying safety problems are mandatory.[2] The following theorem—a consequence of slightly more general decidability results in [15]—supports the automation of safety and fix-point checks, that are reduced in BReach to the satisfiability of $\exists^{A,I}\forall^I$-sentences.

2. In the table, "$\mathcal{M}$ is closed under sub-structures" means means that if $\mathcal{M}$ is in $\mathcal{C}_I$, X is a subset of $\mathtt{INDEX}^{\mathcal{M}}$, and $\mathcal{M}'$ is the structure having X as support set and whose interpretation of predicate symbols is given by the restriction to X, then also $\mathcal{M}'$ is in $\mathcal{C}_I$.

Table 1. Assumptions on T_I, T_E, and the form of I, τ, and U

(HT1)	the $STM(T_I)$ and $STM(T_E)$ problems are decidable
(HT2)	Σ_I does not contain function or constant symbols and its class $\mathcal{C}_I$ of models is closed under sub-structures
(HF1)	I is a $\forall^I$-formula
(HF2)	U is a $\exists^I$-formula
(HF3)	τ is in *functional form*, i.e. a disjunction of formulae of the form $$\exists \underline{i}\, (\phi_L(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j.F(\underline{i}, a[\underline{i}], j, a[j])\,) \qquad (3)$$ where the quantifier-free formula ϕ_L is called the *(local) guard* and $F = F_1, \ldots, F_s$ is a tuple of case-defined functions, called the *(global) updates*
(wHF3)	τ is in *weak functional form*, i.e. a disjunction of formulae of the form (3) or *universally guarded transition formulae in functional form* $$\exists \underline{i}\, (\phi_L(\underline{i}, a[\underline{i}]) \wedge \forall k\, \psi(\underline{i}, k, a[\underline{i}], a[k]) \wedge a' = \lambda j.F(\underline{i}, a[\underline{i}], j, a[j])), \qquad (4)$$ where ϕ_L and F are as for **(HF3)** and ψ is a quantifier-free formula, called the *(global) guard*.

Theorem 4.1. *Under the assumptions **(HT1)** and **(HT2)** in Table 1, we have that*

(RT1) *the A_I^E-satisfiability of $\exists^{A,I}\forall^I$-sentences is decidable;*

(RT2) *an $\exists^{A,I}\forall^I$-sentence is A_I^E-satisfiable iff it is satisfiable in a finite index model, i.e. in a model in which the interpretation of the sort `INDEX` is a finite set.*

The following property states that sets of backward reachable states can always be represented by $\exists^I$-formulae provided that the formulae representing the array-based systems and the set of unsafe states satisfy suitable restrictions (identified in [17, 15]).

Proposition 4.1. *Under the assumptions **(HF1)**, **(HF2)**, and **(HF3)** in Table 1, the following hold:*

(RF1) *if K is an $\exists^I$-formula, then $Pre(\tau, K)$ is equivalent to an (effectively computable) $\exists^I$-formula;*

(RF2) *the A_I^E-satisfiability tests at lines 2 and 3 of the procedure* BReach *in Figure 1 involve only $\exists^{A,I}\forall^I$-sentences.*

The proof of **(RF1)** consists of simple logical manipulations that will be illustrated on the running example below, while that of **(RF2)** is immediate.

In the rest of the paper, unless explicitly stated, we will assume that **(HT1)**, **(HT2)**, **(HF1)**, and **(HF2)** in Table 1 always hold. This is reasonable since it is possible to specify several classes of systems—such as parameterized and distributed systems, imperative programs, and protocols for mutual exclusion or cache coherence—as array-based systems for which these assumptions hold [17, 15]. However, for several of these problems, **(HF3)** is not satisfied, but its weakening, called **(wHF3)** in Table 1, is so. To widen the scope of applicability of BReach to array-based systems satisfying **(wHF3)**, we describe a syntactic transformation in Section 5. Before, we illustrate how BReach works on the running example of Section 3.1 and the difficulties encountered to automate it when dealing with transitions satisfying **(wHF3)** but not **(HF3)**.

4.1 Running example: pre-images and backward reachability

One of the key functionalities of procedure BReach in Figure 1 is $Pre(\tau, P)$ at line 5. We apply Proposition 4.1 for the computation of some pre-images with the transitions considered in Section 3.1. First, we observe that $Pre(\tau, P)$ is logically equivalent to $\bigvee_{\ell=1}^{5} Pre(t_\ell, P)$ when $\tau := \bigvee_{\ell=1}^{5} t_\ell$. Furthermore, the assumptions of Theorem 4.1 are satisfied: **(HT1)** holds since the SMT problem for the theory T_E of the enumerated datatype is obviously decidable and that of the theory T_I of linear orders is also decidable (see, e.g., [9]) and **(HT2)** is satisfied because the signature of T_I contains only predicate symbols while closure under substructures holds because T_I can be axiomatized by universal sentences only (see again [9]). Finally, establishing that assumptions **(HF1)** and **(HF2)** hold is immediate by inspection of the formulae I and U in Section 3.1. The situation for **(HF3)** is more complex and leads us to the main problem considered in this paper. Before discussing this issue, let us observe that t_2, t_4, and t_5 are all in functional form and thus do satisfy **(HF3)**. We illustrate the application of Proposition 4.1 by computing $Pre(t_2, U)$, which is logically equivalent to

$$\exists a'.(\exists k.(a[k] = \mathsf{R} \wedge a' = \mathtt{upd}(a, k, \mathsf{W})) \wedge \exists i, j.(i < j \wedge a'[i] = \mathsf{C} \wedge a'[j] = \mathsf{C}))$$

by (2). By simple logical manipulations, the formula can be rewritten as

$$\exists a'.\exists i, j, k.(a' = \mathtt{upd}(a, k, \mathsf{W}) \wedge a[k] = \mathsf{R} \wedge i < j \wedge \mathtt{upd}(a, k, \mathsf{W})[i] = \mathsf{C} \wedge \mathtt{upd}(a, k, \mathsf{W})[j] = \mathsf{C}).$$

Now, observe that the first conjunct can be dropped, and, recalling the definition of `upd`, we derive:

$$\exists i, j, k.(i < j \wedge k \neq i \wedge k \neq j \wedge a[i] = \mathsf{C} \wedge a[j] = \mathsf{C} \wedge a[k] = \mathsf{R});$$

in fact the cases $k = i$ and $k = j$ yield the T_E-unsatisfiable equality $\mathsf{W} = \mathsf{R}$.[3.] This formula is subsumed by U, hence it can be discarded during backward search.[4.]

3. Notice that disjunctions of $\exists^I$-formulae are equivalent to an $\exists^I$-formula by simple logical manipulation. So, we can conveniently keep $\exists^I$-formulae in the so called *primitive differentiated* format (see, e.g., [15] for a precise definition), namely as disjunctions of existentially quantified conjunctions of literals containing all pairwise distinctions of their existentially quantified variables. Primitive differentiated formulae are used by MCMT [16].
4. With "subsumption", we refer to "partial fix-point tests" according to what is implemented in MCMT [16]. In fact, instead of having a single large $\exists^I$-formula K equivalent to $Pre(\tau, U)$, MCMT maintains several primitive differentiated formulae $K_1, ..., K_r$ (see the previous footnote for a characterization of this class of formulae), whose disjunction is logically equivalent to K but which are smaller and are easier to simplify (roughly, $K_1, ..., K_r$ derive from the computation of $Pre(\widehat{t_\ell}, U)$, for each ℓ). In this way, instead of checking K for fix-point, we check for "partial" fix-point each $K_1, ..., K_r$. This way of implementing BReach seems to be crucial for performance as witnessed by the results of MCMT reported in [16, 15].

It is easy to see that $Pre(t_4, U)$ and $Pre(t_5, U)$ also give rise to subsumed formulae (in fact, the tool MCMT does not even generate these formulae because it implements some optimizations which allow it to detect such subsumptions statically, see [17] for details).

Let us now consider formulae t_1 and t_3, which are not in functional form because they contain a universally quantified formula in their guards. Although Proposition 4.1 does not apply, we can still use (2) to compute, e.g., $Pre(t_3, U)$:

$$\begin{array}{l}\exists a'.(\exists k.(a[k] = \mathsf{W} \wedge \forall l.(l < k \Rightarrow a[l] = I) \wedge a' = \mathtt{upd}(a, k, \mathsf{C}))\wedge \\ \exists i, j.(i < j \wedge a'[i] = \mathsf{C} \wedge a'[j] = \mathsf{C})).\end{array}$$

In turn, this is equivalent to

$$\exists i, j, k.\, \forall l. \left(\begin{array}{l} a[k] = \mathsf{W} \wedge (l < k \Rightarrow a[l] = I)\wedge \\ i < j \wedge \mathtt{upd}(a, k, \mathsf{C})[i] = \mathsf{C} \wedge \mathtt{upd}(a, k, \mathsf{C})[j] = \mathsf{C} \end{array} \right),$$

which is easily seen to be logically equivalent to a $\exists^{A,I}\forall^{I}$-sentence by recalling the definition of upd. So, it does not seem possible to compute an $\exists^{I}$-formula which is logically equivalent to the pre-image of an $\exists^{I}$-formula when the transition contains a universal formula in its guard. At first sight, this does not seem to prevent the mechanization of BReach. In fact, the satisfiability checks at lines 2 and 3 are still covered by Theorem 4.1 since the formulae which are involved ($Pre(t_3, U) \wedge \neg U$ and $I \wedge Pre(t_3, U)$, respectively) are transformed into $\exists^{A,I}\forall^{I}$-sentences with simple logical manipulations. The problem arises at the next iteration of the main loop in BReach since the pre-image of $Pre(t_3, U)$ will again be a $\exists^{A,I}\forall^{I}$-sentence (even if the considered transition is in functional form) so that the satisfiability check at line 2 will involve the conjunction of a $\exists^{A,I}\forall^{I}$-sentence with its negation. This is not covered by Theorem 4.1 and we are no more guaranteed to be able to give fully automated support to BReach. A possible solution would be to study the decidability of sentences of the form $\exists a, \underline{i}.\forall \underline{j}, a' \exists \underline{j}'.\varphi(a, \underline{i}, \underline{j}, a')$. However, classical results about the (un-)decidability of fragments of first-order logic allowing alternation of quantifiers as those considered above suggest that it is very difficult to obtain practically useful conditions. Fortunately, there is a different approach which consists in introducing a failure model and adapting a standard technique in first-order logic, called *relativization of quantifiers*, to translate formulae of many-sorted to unsorted logic (see, e.g., [12]). This is the topic of the following section.

5. Eliminating Universal Conditions in Guards

As illustrated in Section 4.1, it is difficult to automate BReach when the transition formula τ is in weak functional form, see **(wHF3)** in Table 1 for the definition. The main problem is to establish that the global guard of a disjunct of the form (4) in τ holds for all processes comprising the system (i.e. a finite but unknown number of processes) rather than on a bounded number as it is the case for a local guard. To overcome this problem, the key idea is to design a syntactic transformation of the array-based system so as to obtain a sufficiently precise *abstraction* that preserves the safety properties. The abstraction is induced by the adoption of the following simple *failure model*: *we allow processes to crash and we stipulate that crashed processes cannot take an active role in the protocol.* Once the reference model has changed because crashes may occur, it is not difficult to see that (modulo bisimulation) universally guarded transitions in functional form (4) can be replaced by transitions in

Table 2. Definitions of the mappings $(\tilde{\cdot})$ and $(\widehat{\cdot})$

Original Array-based System $\mathcal{S}$ and unsafe states U

(a)	T_I, T_E
(b)	$a = a_1, ..., a_s$
(c)	$U(a) := \exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$
(d)	$I(a) := \forall \underline{i}.\phi(\underline{i}, a[\underline{i}])$
(e)	$t_\ell(a, a') := \exists \underline{i} \begin{pmatrix} \phi_L(\underline{i}, a[\underline{i}]) \wedge \forall k\, \psi(\underline{i}, k, a[\underline{i}], a[k]) \\ a' = \lambda j.F(\underline{i}, a[\underline{i}], j, a[j]) \end{pmatrix}$ where $F(\underline{i}, \underline{e}, j, d) :=$ `case of` $\{$ $C_1(\underline{i}, \underline{e}, j, d) \;:\; o_1(\underline{i}, \underline{e}, j, d)$ $\cdots$ $C_k(\underline{i}, \underline{e}, j, d) \;:\; o_k(\underline{i}, \underline{e}, j, d) \quad \}$
(f)	$\tau(a, a') := \bigvee_{\ell=1}^{n} t_\ell(a, a')$

$\downarrow (\tilde{\cdot})$

Intermediate Array-based System $\tilde{\mathcal{S}}$ and unsafe states $\tilde{U}$

(a)	$\tilde{T}_I := T_I$ and $\tilde{T}_E := T_E$ plus constants $\mathtt{t} \neq \mathtt{f}$ of sort $\mathtt{ELEM}_{s+1}$
(b)	$\tilde{a} := a, a_{s+1}$ where a_{s+1} is of sort $\mathtt{ARRAY}_{s+1}$
	for $\underline{i} := i_1, ..., i_k$, the notation $\mathtt{A}(\underline{i})$ abbreviates $\bigwedge_{\ell=1}^{k}(a_{s+1}[i_\ell] = \mathtt{t})$
(c)	$\tilde{U}(\tilde{a}) := \exists \underline{i}.(\mathtt{A}(\underline{i}) \wedge \phi(\underline{i}, a[\underline{i}]))$
(d)	$\tilde{I}(\tilde{a}) := \forall \underline{i}.(\mathtt{A}(\underline{i}) \Rightarrow \phi(\underline{i}, a[\underline{i}]))$
(e)	$\tilde{t}_\ell(\tilde{a}, \tilde{a}') := \exists \underline{i} \begin{pmatrix} \mathtt{A}(\underline{i}) \wedge \phi_L(\underline{i}, a[\underline{i}]) \wedge \forall k\, (\mathtt{A}(k) \Rightarrow \psi(\underline{i}, k, a[\underline{i}], a[k])) \wedge \\ a' = \lambda j.F_{\mathtt{A}}(\underline{i}, a[\underline{i}], j, a[j], a_{s+1}[j]) \wedge a'_{s+1} = a_{s+1} \end{pmatrix}$ where $F_{\mathtt{A}}(\underline{i}, \underline{e}, j, d, \beta) :=$ `case of` $\{$ $C_1(\underline{i}, \underline{e}, j, d) \wedge \beta = \mathtt{t} \;:\; o_1(\underline{i}, \underline{e}, j, d)$ $\cdots$ $C_k(\underline{i}, \underline{e}, j, d) \wedge \beta = \mathtt{t} \;:\; o_k(\underline{i}, \underline{e}, j, d)$ $\beta = \mathtt{f} \;:\; d \quad \}$
(e′)	$\tilde{t}_{n+1}(\tilde{a}, \tilde{a}') := \exists i\, (\mathtt{A}(i) \wedge a' = a \wedge a'_{s+1} = \mathtt{upd}(a_{s+1}, i, \mathtt{f}))$
(f)	$\tilde{\tau}(\tilde{a}, \tilde{a}') := \bigvee_{\ell=1}^{n+1} \tilde{t}_\ell(\tilde{a}, \tilde{a}')$

$\downarrow (\widehat{\cdot})$

Target Array-based System $\widehat{\mathcal{S}}$ and unsafe states $\widehat{U}$

(a)	$\widehat{T_I} := \tilde{T}_I$ and $\widehat{T_E} := \tilde{T}_E$
(b)	$\widehat{a} := \tilde{a}$
(c)	$\widehat{U}(\widehat{a}) := \tilde{U}(\tilde{a})$
(d)	$\widehat{I}(\widehat{a}) := \tilde{I}(\tilde{a})$
(e)	$\widehat{t}_\ell(\widehat{a}, \widehat{a}') := \exists \underline{i} \begin{pmatrix} \mathtt{A}(\underline{i}) \wedge \phi_L(\underline{i}, a[\underline{i}]) \wedge \bigwedge_{j \in \underline{i}} \psi(\underline{i}, j, a[\underline{i}], a[j]) \\ a' = \lambda j.F_{\mathtt{A}}^{\forall}(\underline{i}, a[\underline{i}], j, a[j], a_{s+1}[j]) \wedge \\ a'_{s+1} = \lambda j.f_{\mathtt{A}}^{\forall}(i, a[j], j, a_{s+1}[j]) \end{pmatrix}$ where $F_{\mathtt{A}}^{\forall}(\underline{i}, \underline{e}, j, d, \beta) :=$ `case of` $\{$ $C_1(\underline{i}, \underline{e}, j, d) \wedge \beta = \mathtt{t} \wedge \psi(\underline{i}, \underline{e}, j, d) \;:\; o_1(\underline{i}, \underline{e}, j, d)$ $\cdots$ $C_k(\underline{i}, \underline{e}, j, d) \wedge \beta = \mathtt{t} \wedge \psi(\underline{i}, \underline{e}, j, d) \;:\; o_k(\underline{i}, \underline{e}, j, d)$ $\beta = \mathtt{f} \vee \neg\psi(\underline{i}, \underline{e}, j, d) \;:\; d \quad \}$ and $f_{\mathtt{A}}^{\forall}(\underline{i}, \underline{e}, j, d, \beta) :=$ `if` $(\beta = \mathtt{f} \vee \neg\psi(\underline{i}, j, \underline{e}, d))$ `then f else` β
(e′)	$\widehat{t_{n+1}}(\widehat{a}, \widehat{a}') := \tilde{t}_{n+1}(\tilde{a}, \tilde{a}')$
(f)	$\widehat{\tau}(\widehat{a}, \widehat{a}') := \bigvee_{\ell=1}^{n+1} \widehat{t}_\ell(\widehat{a}, \widehat{a}')$

simple functional form (3). Intuitively, this is because runs in which processes not satisfying the condition in the universal guards crash are allowed in the abstract system. We shall make this observation rigorous in the technical developments below. Here, we point out that the array-based system obtained by this transformation is more liberal, i.e. it has more runs than the original system. As a consequence, if a set of bad states U is shown to be unreachable for the more liberal system, then it will also be unreachable for the original system. In practice, many common distributed algorithms are fault-tolerant and this explains why it is not by chance that our approach usually works and produces the desired safety certifications, as it is evident from the extensive experiments reported in [15].

Let T_I and T_E be theories, $\mathcal{S} = (a, I, \tau)$ be an array-based system, and U a $\exists^I$-formula (describing a set of unsafe states) satisfying the assumptions **(HT1)**, **(HT2)**, **(HF1)**, **(HF2)**, and **(wHF3)** in Table 1. To simplify the formal definition of the syntactic transformation, we split it in two maps, namely $(\tilde{\cdot})$ and $(\hat{\cdot})$, defined on safety problems, i.e. pairs of an array-based transition system and an $\exists^I$-formula. The first mapping $(\tilde{\cdot})$ *introduces failures* and transforms $(\mathcal{S}, U)$ into $(\tilde{\mathcal{S}}, \tilde{U})$. The second mapping $(\hat{\cdot})$ *removes the universal guards* and takes $(\tilde{\mathcal{S}}, \tilde{U})$ to $(\widehat{\mathcal{S}}, \widehat{U})$. The two mappings are in Table 2.

First Step: $\boxed{(\mathcal{S}, U) \longmapsto (\tilde{\mathcal{S}}, \tilde{U})}$. The technique for introducing the failure version of $(\mathcal{S}, U)$ consists in adding an auxiliary variable and then in relativizing all quantifiers to the formula describing non-crashed processes. This is done as follows.

- In (a), we add a new sort $\mathtt{ELEM}_{s+1}$ with two constant symbols $\mathtt{t}$ and $\mathtt{f}$ to T_E so that its models are expanded in such a way that $\mathtt{ELEM}_{s+1}$ is interpreted as a set of two elements, and $\mathtt{t}$ and $\mathtt{f}$ are mapped to distinct elements of the set.

- In (b), the auxiliary (array state) variable a_{s+1} of sort $\mathtt{ARRAY}_{s+1}$ is also added to the original variables a of the system and the resulting tuple of variables is denoted with $\tilde{a}$. The abbreviation $\mathtt{A}(\underline{i})$ is introduced in order to identify the conjunction of equalities between the terms $a_{s+1}[\underline{i}]$ and the constant $\mathtt{t}$: the formula $\mathtt{A}(i_\ell)$ means that "i_ℓ is not crashed".

- In (c), the existential quantifier of formula U is "relativized" with respect to $\mathtt{A}$.

- In (d), the universal quantifier of I is also relativized.

- In (e), the existential and universal quantifiers of the transition formula are relativized with respect to $\mathtt{A}$; similarly, the case-defined functions of the updates are also relativized in order to modify the values of the arrays a only whenever the auxiliary variable a_{s+1} is set to $\mathtt{t}$. Formally, this is done by adding an extra parameter β to the case-defined function and creating a new partition by splitting on its two possible values (either $\mathtt{t}$ or $\mathtt{f}$) and then distributing over $C_1, ..., C_k$. According to this transformation, formulae of the form (3), obtained from $t(a, a')$ in Table 2 by taking ψ to be equivalent to *true*, are mapped to

$$\exists \underline{i}\, (\mathtt{A}(\underline{i}) \wedge \phi_L(\underline{i}, a[\underline{i}]) \wedge \tilde{a}' = \lambda j. \tilde{F}(\underline{i}, \tilde{a}[\underline{i}], j, \tilde{a}[j])) \tag{5}$$

and those of the form (4) to

$$\exists \underline{i} \left(\begin{array}{l} \mathtt{A}(\underline{i}) \wedge \phi_L(\underline{i}, a[\underline{i}]) \wedge \forall k\, (\mathtt{A}(k) \Rightarrow \psi(\underline{i}, k, a[\underline{i}], a[k])) \wedge \\ \tilde{a}' = \lambda j. \tilde{F}(\underline{i}, \tilde{a}[\underline{i}], j, \tilde{a}[j]) \end{array} \right), \tag{6}$$

where $\tilde{a}' = \lambda j.\tilde{F}(\underline{i}, \tilde{a}[\underline{i}], j, \tilde{a}[j])$ abbreviates $a' = \lambda j.F_{\mathtt{A}}(\underline{i}, a[\underline{i}], j, a[j], a_{s+1}[j]) \wedge a'_{s+1} = a_{s+1}$.[5] Notice that the value of a_{s+1} is not changed by the transition $\tilde{t}_\ell$.

- In (e′), we formalize the fact that a process can crash at any time: we add the $(n+1)$-transition which leaves the original variables in a unchanged and non-deterministically changes the value of a cell of a_{s+1} from $\mathtt{t}$ to $\mathtt{f}$.
- Finally, in (f), we construct the new transition formula $\tilde{\tau}$ by simply taking the disjunction of the "relativized" transition formulae $\tilde{t}_1, ..., \tilde{t}_n$ with the additional disjunct $\tilde{t}_{n+1}$.

It is not difficult to show that the safety problems $(\mathcal{S}, U)$ and $(\tilde{\mathcal{S}}, \tilde{U})$ have the same answer provided that τ is in functional form (see Proposition 5.2 in Section 5.2 below).

Second Step: $\boxed{(\tilde{\mathcal{S}}, \tilde{U}) \longmapsto (\widehat{\mathcal{S}}, \widehat{U})}$. We now eliminate the universal guards. The definition of this mapping is simpler since most of the work has already been done.

- We do not modify further the component theories (a), the tuple of the state variables (b), the unsafe (c) and the initial formulae (d), and (e′) the additional transition non-deterministically updating the auxiliary variable a_{s+1}.
- Our main purpose is (e) to transform the formulae $\tilde{t}_1, ..., \tilde{t}_n$ by replacing the universally quantified formula $\forall k\, (\mathtt{A}(k) \Rightarrow \psi(\underline{i}, k, a[\underline{i}], a[k]))$ in the guard by the finite conjunction of quantifier-free formulae $\bigwedge_{k \in \underline{i}} \psi(\underline{i}, k, a[\underline{i}], a[k])$; in addition, we modify the case-defined function of the updates as follows. First, we update only the values j of the arrays a having the auxiliary variable $a_{s+1}[j]$ set to $\mathtt{t}$ (as before) and for which the (removed) global condition $\psi(\underline{i}, j, a[\underline{i}], a[j])$ holds; furthermore—and most importantly—we force the value of $a_{s+1}[j]$ for those cells j for which $\psi(\underline{i}, j, a[\underline{i}], a[j])$ does not hold to become $\mathtt{f}$. Formally, we create a new partition by splitting on $\psi(\underline{i}, \underline{e}, j, d)$ and then distributing over the intermediate partition $(C_1 \wedge \beta = \mathtt{t}), ..., (C_k \wedge \beta = \mathtt{t}), \beta = \mathtt{f}$. According to this transformation, formulae of the form (5) are mapped to
$$\exists \underline{i}\, (\mathtt{A}(\underline{i}) \wedge \phi_L(\underline{i}, a[\underline{i}]) \wedge \widehat{a}' = \lambda j.\widehat{F}(\underline{i}, \widehat{a}[\underline{i}], j, \widehat{a}[j])) \tag{7}$$
and those of the form (6) to
$$\exists \underline{i}\, (\mathtt{A}(\underline{i}) \wedge \phi_L(\underline{i}, a[\underline{i}]) \wedge \bigwedge_{k \in \underline{i}} \psi(\underline{i}, k, a[\underline{i}], a[k]) \wedge \widehat{a}' = \lambda j.\widehat{F}(\underline{i}, \widehat{a}[\underline{i}], j, \widehat{a}[j])), \tag{8}$$
where $\widehat{a}' = \lambda j.\widehat{F}(\underline{i}, \widehat{a}[\underline{i}], j, \widehat{a}[j])$ abbreviates $a' = \lambda j.F^{\forall}_{\mathtt{A}}(\underline{i}, a[\underline{i}], j, a[j], a_{s+1}[j]) \wedge {a_{s+1}}' = \lambda j.f^{\forall}_{\mathtt{A}}(i, a[i], j, a[j], a_{s+1}[j])$, where $F_{\mathtt{A}}, f_{\mathtt{A}}$ are as explained above and as formally defined in Table 2.

We will show (Theorem 5.1 below) that the safety problems $(\tilde{\mathcal{S}}, \tilde{U})$ and $(\widehat{\mathcal{S}}, \widehat{U})$ have the same answer under the assumptions **(HT1)**, **(HT2)**, **(HF1)**, **(HF2)**, and **(wHF3)** of Table 1.

5. In Table 2, we made some notational simplifications that deserve a comment. First, in the lines marked with (e), the length of the tuple $\underline{i}$ as well as the vectors of function symbols $F, F_{\mathtt{A}}, \ldots$ should depend on ℓ; we omitted this dependence to avoid the notation overhead. Second, since the F's are vectors (having s-components, like the a's), the tuple $o_1, \ldots, o_k$ is a tuple of vectors of terms, each of which having s-components. (All this is consistent with our policy to use non-underlined names for tuples of expressions having fixed length s.) Third, since any two T-partitions have a common refinement, one can freely assume that the T-partition used in the definitions of the F's is the same for all the s-components of the vector F.

5.1 Running example: elimination of universal conditions

We illustrate the two mappings on the transitions of our running example in Section 3.1. First of all, we introduce the auxiliary variable v (corresponding to a_{s+1} above) together with the extension to theory T_E of the enumerated data-type for control locations.

The sequential application of the two mappings on the initial state formula I and the unsafe formula U (recall them from Section 3.1) yields

$$\begin{array}{rcl} \widehat{I}(a,v) & := & \forall i.(v[i] = \mathtt{t} \Rightarrow a[i] = \mathsf{I}) \text{ and} \\ \widehat{U}(a,v) & := & \exists i,j.(v[i] = \mathtt{t} \wedge v[j] = \mathtt{t} \wedge i < j \wedge a[i] = \mathsf{C} \wedge a[j] = \mathsf{C}), \end{array}$$

respectively. Notice that this initialization does not exclude that some processes are crashed from the very beginning, but this liberal assumption is in fact immaterial.

The effect of the composition of the mappings $(\tilde{\cdot})$ and $(\hat{\cdot})$ on t_2, t_4, and t_5 (of the form (3)) is very limited: quantifiers are relativized with respect to $v[i]$ and v is updated identically. we show the result of transforming t_2, namely

$$\widehat{t_2}(a,v,a',v') \quad := \quad \exists i.(v[i] = \mathtt{t} \wedge a[i] = \mathsf{R} \wedge a' = \mathtt{upd}(a,i,\mathsf{W}) \wedge v' = v),$$

and omit those for t_4 and t_5 as they are very similar. The additional transition $\widehat{t_6}$ for the non-deterministic update of v is

$$\widehat{t_6}(a,v,a',v') \quad := \quad \exists i.(v[i] = \mathtt{t} \wedge a' = a \wedge v' = \mathtt{upd}(v,i,\mathtt{f})),$$

which arbitrarily selects a process i and sets the corresponding value in v to `f`. Now, we consider the more interesting cases of t_1 and t_3, which both contain a global condition in their guards. The result of syntactically transforming t_1 is

$$\widehat{t_1}(a,v,a',v') \quad := \quad \exists i. \left(\begin{array}{l} v[i] = \mathtt{t} \wedge a[i] = \mathsf{I} \wedge (i \neq i \Rightarrow a[i] \in \{\mathsf{I},\mathsf{R}\}) \wedge \\ a' = \mathtt{upd}(a,i,\mathsf{R}) \wedge \\ v' = \lambda j. \left(\begin{array}{l} \mathtt{if}\ (v[j] = \mathtt{f} \vee \neg(j \neq i \Rightarrow a[j] \in \{\mathsf{I},\mathsf{R}\})) \\ \mathtt{then}\ \mathtt{f}\ \mathtt{else}\ v[j] \end{array} \right) \end{array} \right).$$

Notice that $(i \neq i \Rightarrow a[i] \in \{\mathsf{I},\mathsf{R}\})$ can be removed from $\widehat{t_1}(a,v,a',v')$ since it is always true. Similarly, the transformation of t_3 yields the formula:

$$\widehat{t_3}(a,v,a',v') \quad := \quad \exists i. \left(\begin{array}{l} v[i] = \mathtt{t} \wedge a[i] = \mathsf{W} \wedge (i < i \Rightarrow a[i] = I) \wedge \\ a' = \mathtt{upd}(a,i,\mathsf{C}) \wedge \\ v' = \lambda j. \left(\begin{array}{l} \mathtt{if}\ (v[j] = \mathtt{f} \vee \neg(j < i \Rightarrow a[j] = I)) \\ \mathtt{then}\ \mathtt{f}\ \mathtt{else}\ v[j] \end{array} \right) \end{array} \right),$$

where $(i < i \Rightarrow a[i] = I)$ can be deleted.

It should now be clear that $\widehat{\tau} := \bigvee_{\ell=1}^{6} \widehat{t_\ell}$ is in functional form; whereas τ was not, because of t_1 and t_3. The transformed safety problem allows us to overcome the difficulties of applying BReach in discussed in Section 4.1. For example, recall that we were not able to compute an $\exists^I$-formula logically equivalent to $Pre(t_3, U)$ in Section 4.1: the application of the definition (2) of pre-image yielded an $\exists^{A,U}\forall^I$-sentence. Let us now compute $Pre(\widehat{t_3}, \widehat{U})$

and recall that **(RF1)** in Proposition 4.1 ensures us to be able to find a logically equivalent $\exists^I$-formula since $\widehat{t_3}$ is in functional form:

$$\exists a', v'. \left(\begin{array}{l} \exists k. \left(\begin{array}{l} v[k] = \mathtt{t} \wedge a[k] = \mathsf{W} \wedge a' = \mathtt{upd}(a, k, \mathsf{C}) \wedge \\ v' = \lambda j. \left(\begin{array}{l} \mathtt{if}\ (v[j] = \mathtt{f} \vee \neg(j < k \Rightarrow a[j] = I)) \\ \mathtt{then}\ \mathtt{f}\ \mathtt{else}\ v[j] \end{array} \right) \end{array} \right) \wedge \\ \exists i, j.(v'[i] = \mathtt{t} \wedge v'[j] = \mathtt{t} \wedge i < j \wedge a'[i] = \mathsf{C} \wedge a'[j] = \mathsf{C}) \end{array} \right),$$

which rewrites to

$$\exists i, j, k. \left(\begin{array}{l} v[k] = \mathtt{t} \wedge a[k] = \mathsf{W} \wedge i < j \wedge \\ \mathtt{upd}(a, k, \mathsf{C})[i] = \mathsf{C} \wedge \mathtt{upd}(a, k, \mathsf{C})[j] = \mathsf{C} \wedge \\ (\mathtt{if}\ (v[i] = \mathtt{f} \vee \neg(i < k \Rightarrow a[i] = I))\ \mathtt{then}\ (\mathtt{f} = \mathtt{t})\ \mathtt{else}\ (v[i] = \mathtt{t})) \wedge \\ (\mathtt{if}\ (v[j] = \mathtt{f} \vee \neg(j < k \Rightarrow a[j] = I))\ \mathtt{then}\ (\mathtt{f} = \mathtt{t})\ \mathtt{else}\ (v[j] = \mathtt{t})) \end{array} \right)$$

by logical manipulations similar to those discussed in Section 4.1 when computing $Pre(t_3, U)$. By case-splitting on $k = i$, $k = j$, and $k \neq i \wedge k \neq j$, further simplifications, and satisfiability checks modulo T_E, the formula above can be simplified to the disjunction of

$$\exists i, j.(i < j \wedge v[i] = \mathtt{t} \wedge v[j] = \mathtt{t} \wedge a[i] = \mathsf{W} \wedge a[j] = \mathsf{C}),$$

and

$$\begin{array}{rl} \exists i, j.k & (i < j \wedge i > k \wedge j > k \wedge v[k] = \mathtt{t} \wedge v[i] = \mathtt{t} \wedge \\ & v[j] = \mathtt{t} \wedge a[k] = \mathsf{W} \wedge a[i] = \mathsf{C} \wedge a[j] = \mathsf{C}); \end{array}$$

the third disjunct is unsatisfiable as it contains $i < j \wedge j < i$. Since the second disjunct is subsumed by U, only the first one needs to be kept. The computation of $Pre(\widehat{t_\ell}, U)$ for $\ell = 1, 2, 3, 5, 6$ yields an unsatisfiable or subsumed $\exists^I$-formula and thus $U \vee Pre(\widehat{\tau}, U)$ is equivalent to $U \vee Pre(\widehat{t_3}, U)$. Now, observe that $Pre(\widehat{\tau}, U) \wedge \widehat{I}$ is $\widehat{A_I^E}$-unsatisfiable (this means that unsafety has not been detected after one iteration of BReach) and $Pre(\widehat{\tau}, U) \Rightarrow \widehat{U}$ is $\widehat{A_I^E}$-invalid (hence, a fix-point has not been reached). We should proceed to the next iteration and compute the pre-image of $U_1 := Pre(\widehat{t_3}, U)$ with respect to $\widehat{\tau}$. Using simplification and subsumption, only the following new $\exists^I$-formula remains:

$$\exists i, j.(i < j \wedge v[i] = \mathtt{t} \wedge v[j] = \mathtt{t} \wedge a[i] = \mathsf{R} \wedge a[j] = \mathsf{C}).$$

It is then possible to see that $Pre(U_1, \widehat{\tau}) \wedge \widehat{I}$ is $\widehat{A_I^E}$-unsatisfiable (hence, unsafety has not been detected), and $Pre(U_1, \widehat{\tau}) \Rightarrow (U_1 \vee \widehat{U})$ is again $\widehat{A_I^E}$-invalid (so that a fix-point has not been detected). One more iteration of BReach is sufficient to discover that a fix-point is reached and that the intersection with the initial set of states is empty. As a consequence, BReach returns the answer safe for the (target) safety problem $(\widehat{\mathcal{S}}, \widehat{U})$.

What can we infer about the (original) safety problem $(\mathcal{S}, U)$? We will see that it has the same answer because of the properties of the two mappings $(\tilde{\cdot})$ and $(\hat{\cdot})$. The key point is that $\widehat{\mathcal{S}}$ admits more runs than $\mathcal{S}$ and if $\widehat{U}$ is unreachable from $\widehat{I}$ by any finite sequence of applications of $\widehat{\tau}$, then—*a fortiori*—U is unreachable from I by applying τ. To see an example of a run in $\widehat{\mathcal{S}}$ which is not allowed in $\mathcal{S}$, consider the set of states of $\mathcal{S}$ described by the following $\exists^I$-rule:

$$P \quad := \quad \exists i_1, ..., i_7.(a[i_1, ..., i_7] = \mathsf{I}, \mathsf{R}, \mathsf{I}, \mathsf{C}, \mathsf{W}, \mathsf{W}, \mathsf{R}).$$

Transition t_3 is not enabled in P since its universal condition, namely $\forall j.(j < i \Rightarrow a[j] = \mathsf{I})$, cannot be satisfied. In fact, in order to satisfy the local condition of the guard $a[i] = \mathsf{W}$, one should take either $i = i_5$ or $i = i_6$ so that the universal condition turns out to be violated at, e.g., i_2 where $a[i_2] = \mathsf{R} \neq \mathsf{I}$. Now, consider the corresponding set of states in $\widehat{\mathcal{S}}$, described by the following relativized $\exists^I$-formulae:

$$\widehat{P} \quad := \quad \exists i_1, ..., i_7.(v[i_1, ..., i_7] = \mathtt{t}, ..., \mathtt{t} \wedge a[i_1, ..., i_7] = \mathsf{I}, \mathsf{R}, \mathsf{I}, \mathsf{C}, \mathsf{W}, \mathsf{W}, \mathsf{R}).$$

if we apply twice $\widehat{t_6}$ to i_2 and i_4, we obtain

$$\exists i_1, ..., i_7.(v[i_2, i_4] = \mathtt{f}, \mathtt{f} \wedge f[i_1, i_3, i_5, i_6, i_7] = \mathtt{t}, ..., \mathtt{t} \wedge a[i_1, ..., i_7] = \mathsf{I}, \mathsf{R}, \mathsf{I}, \mathsf{C}, \mathsf{W}, \mathsf{W}, \mathsf{R}),$$

and the transition $\widehat{t_3}$ becomes enabled as the values of f at i_2 and i_4 make them irrelevant for its application.

5.2 Properties of $(\mathcal{S}, U) \mapsto (\tilde{\mathcal{S}}, \tilde{U})$

When the answer to a safety problem $(\mathcal{S}, U)$ is safe (unsafe), we say that $\mathcal{S}$ *is safe (unsafe, respectively) with respect to* U.

Proposition 5.1. *If $\tilde{\mathcal{S}}$ is safe with respect to $\tilde{U}$, then $\mathcal{S}$ is safe with respect to U.*

Proof. Suppose (by contra-position) that

$$I(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge U(a_n)$$

is A_I^E-satisfiable in a model $\mathcal{M}$ under a certain assignment $\mathtt{a}$. We expand the model $\mathcal{M}$ to a model $\tilde{\mathcal{M}}$ of $\tilde{A}_I^E$ in the obvious way by interpreting $\mathtt{ELEM}_{s+1}$ as a set containing two distinct elements, say 0 and 1 for the constants $\mathtt{f}$ and $\mathtt{t}$, respectively, and interpreting $\mathtt{ARRAY}_{s+1}$ as the set of (total) functions from $\mathtt{INDEX}^{\mathcal{M}}$ to $\{0, 1\}$. Then, we expand the assignment $\mathtt{a}$ by assigning to n-copies $(a_{s+1})_1, \cdots, (a_{s+1})_n$ of the new array variable a_{s+1} the array whose constant value is 1. In this way, it is easy to see that

$$\mathcal{M}, \mathtt{a} \models I(\tilde{a}_0) \wedge \tau(\tilde{a}_0, \tilde{a}_1) \wedge \cdots \wedge \tau(\tilde{a}_{n-1}, \tilde{a}_n) \wedge U(\tilde{a}_n)$$

holds under the assignment $\mathtt{a}$ as expanded above and hence $\tilde{\mathcal{S}}$ is unsafe with respect to $\tilde{U}$. □

Safety of the intermediate system $\tilde{\mathcal{S}}$ implies safety of the original system $\mathcal{S}$. The contrary is false for arbitrary array-based systems although we can prove it for systems in functional form. In the rest of the current Subsection 5.2, we assume that $((a, I, \tau), U)$ is a safety problem for which assumptions **(HT1)**, **(HT2)**, **(HF1)**, **(HF2)**, and **(HF3)** hold (thus, we consider $\tau(a, a')$ to be in functional form).

We show that the sets of backward reachable states are the same for the original and the intermediate systems. For a $\exists^I$-formula $K := \exists \underline{i}\, \psi)$, we let $\tilde{K}$ be $\exists \underline{i}.(\mathtt{A}(\underline{i}) \wedge \psi)$; similarly, for a $\forall^I$-formula $J := \forall \underline{i}\, \phi)$, we let $\tilde{J}$ be $\forall \underline{i}.(\mathtt{A}(\underline{i}) \Rightarrow \psi)$ (this is consistent with the notation used in Table 2).

Lemma 5.1. *Let K be an $\exists^I$-formula and K_0 be the $\exists^I$-formula which is logically equivalent to $Pre(\tau, K)$. Then, the $\exists^I$-formulae $\tilde{K} \vee \tilde{K}_0$ and $\tilde{K} \vee Pre(\tilde{\tau}, \tilde{K})$ are $\tilde{A}_I^E$-equivalent.*

Proof. We need three observations. First, according to Proposition 4.1, $\exists^I$-formulae are closed under pre-images when transitions are in functional form. Second, $\exists^I$-formulae are closed under disjunctions, modulo simple logical manipulations. Third, the disjunct $\tilde{K}$ in the statement of the Lemma 5.1 is needed because $\tilde{\tau}$ contains also the disjunct $\tilde{t}_{n+1}(a, a')$; see (e$'$) in Table 2.

The Lemma follows straightforwardly from the definition of $(\tilde{\cdot})$, in particular (c), (d), (e), and (e$'$) in Table 2; and the fact that it is not possible to change the value of $a_{s+1}[i]$ when this has been set to f by construction of $\tilde{\tau}$. □

Safety checks for the original and the intermediate systems are equivalent.

Proposition 5.2. *$\tilde{\mathcal{S}}$ is safe with respect to $\tilde{U}$ if and only if $\mathcal{S}$ is safe with respect to U.*

Proof. Unsafety of $\tilde{\mathcal{S}}$ with respect to $\tilde{U}$ means that $\bigvee_{k \le n} Pre^k(\tilde{\tau}, \tilde{U})$ is $\tilde{A}_I^E$-consistent with $\tilde{I}$ for some n. Now, if we put $K := \bigvee_{k \le n} Pre^k(\tau, U)$, we have that $\tilde{K} \wedge \tilde{I}$ is $\tilde{A}_I^E$-consistent iff $K \wedge I$ is A_I^E-consistent by Lemma 5.2 below. This concludes the proof in view of Lemma 5.1 above. □

As a side remark, we observe that Proposition 5.2 does not apply to properties different from safety, like deadlock-freedom.

Lemma 5.2. *Let I be a $\forall^I$-formula and K be a $\exists^I$-formula. Then, $\tilde{I} \wedge \tilde{K}$ is $\tilde{A}_I^E$-satisfiable iff $I \wedge K$ is A_I^E-satisfiable.*

Proof. One side of the equivalence (from right to left) is trivial. For the other side, suppose that $\tilde{\mathcal{M}} \models \tilde{I} \wedge \tilde{K}$ and let K be $\exists \underline{i} \phi(\underline{i}, a[\underline{i}])$. Thus $\tilde{\mathcal{M}} \models \tilde{I} \wedge \mathtt{A}(\underline{i}) \wedge \phi(\underline{i}, a[\underline{i}])$ holds for some assignment to a, a_{s+1} and to the $\underline{i}$. Let the A_I^E-model $\mathcal{M}$ be obtained from the $\tilde{A}_I^E$-model $\tilde{\mathcal{M}}$ by restricting the interpretation of the sort `INDEX` to the elements assigned to the $\underline{i}$ (notice that hypothesis **(HT2)** is used here). It is clear that all processes selected for $\mathcal{M}$ are active in $\tilde{\mathcal{M}}$ (i.e. $\tilde{\mathcal{M}} \models \mathtt{A}(j)$ holds for every $j \in \mathtt{INDEX}^{\mathcal{M}}$), hence if we assign in $\mathcal{M}$ to a the arrays so restricted in the domain, we get $\mathcal{M} \models I(a) \wedge K(a)$ from $\tilde{\mathcal{M}} \models \tilde{I} \wedge \phi(\underline{i}, a[\underline{i}])$. □

5.3 Properties of $(\tilde{\mathcal{S}}, \tilde{U}) \mapsto (\widehat{\mathcal{S}}, \widehat{U})$

Consider the transition $\widehat{t_\ell}$ of the target system in Table 2: the universal condition in the guard of $\tilde{t}_\ell$ of the intermediate system has been removed and the corresponding values in a_{s+1} of those processes violating it are set to f. We prove that the elimination of the universal conditions from transitions is without loss of precision when verifying safety properties. The reason why $\tilde{\mathcal{S}}$ and $\widehat{\mathcal{S}}$ are equivalent is that the transition $\widehat{\tau}$ can be simulated by finitely many application of the additional transition $\tilde{t}_{n+1}$ followed by the application of $\bigvee_{\ell=1}^n \tilde{t}_\ell$. In fact, the universal condition of the guard has been instantiated in $\widehat{t_\ell}$ of Table 2 only to the processes enabling the transition, so that the transition can fire anyway if the values of a_{s+1} of those processes violating the universal condition of the guard have been set to f before the transition applies. To make this intuition precise, we need to introduce some technical notions and, in particular, that of bisimulation.

Let $\mathcal{S} = (a, I, \tau)$ be an array-based system. A *state* of $\mathcal{S}$ is a pair $(s, \mathcal{M})$ where $\mathcal{M}$ is a model of A_I^E and s is a tuple of elements in $\mathtt{ARRAY}_1^{\mathcal{M}}, \ldots, \mathtt{ARRAY}_s^{\mathcal{M}}$. A *configuration* of $\mathcal{S}$ is a state $(s, \mathcal{M})$ where $\mathcal{M}$ is a finite index model, i.e. $\mathtt{INDEX}^{\mathcal{M}}$ is a finite set. Below, we will write $\mathcal{M}_s$ to denote the finite index model where s is taken from. The n-th iteration of $\tau(a, a')$ is inductively defined as follows: (i) $\tau^0(a, a') := (a = a')$ and (ii) $\tau^{n+1}(a, a') := \exists a''.(\tau(a, a'') \wedge \tau^n(a'', a'))$.

Definition 5.1. Suppose we are given two array-based systems $\mathcal{S}_1 = (a, I_1, \tau_1)$ and $\mathcal{S}_2 = (a, I_2, \tau_2)$ (with the same background theory A_I^E). A *bisimulation* between them is a relation R between configurations of $\mathcal{S}_1$ and $\mathcal{S}_2$ satisfying the following properties:

- for every s_1, there exists s_2 such that $R(s_1, s_2)$ holds (and vice versa);
- if $R(s_1, s_2)$, then $\mathcal{M}_{s_1} \models I_1(s_1)$ iff $\mathcal{M}_{s_2} \models I_2(s_2)$;
- if $R(s_1, s_2)$ and $\mathcal{M}_{s_1} \models \tau(s_1, s_1')$, then there exist s_2' and $n \geq 0$ such that $R(s_1', s_2')$ and $\mathcal{M}_{s_2} \models \tau^n(s_2, s_2')$ (and vice versa).

The safety problems $(\mathcal{S}_1, U_1)$ and $(\mathcal{S}_2, U_2)$ are *compatible with the bisimulation* R iff $R(s_1, s_2)$ implies that $\mathcal{M}_{s_1} \models U_1(s_1)$ iff $\mathcal{M}_{s_2} \models U_2(s_2)$.

The following key property for bisimulations of array-based systems is an immediate consequence of the fact that formula (1) in the definition of safety problem is a $\exists^{A,I}\forall^I$-sentence and that an $\exists^{A,I}\forall^I$-sentence is A_I^E-satisfiable iff it is satisfiable in a finite index model (see **(RT2)** of Theorem 4.1).

Lemma 5.3. *Suppose that R is a bisimulation between $\mathcal{S}_1 = (a, I_1, \tau_1)$ and $\mathcal{S}_2 = (a, I_2, \tau_2)$ and that the safety problems $(\mathcal{S}_1, U_1)$ and $(\mathcal{S}_2, U_2)$ are compatible with R. Then, $\mathcal{S}_1$ is safe with respect to U_1 iff $\mathcal{S}_2$ is safe with respect to U_2.*

We are now ready to formally state and prove our main result.

Theorem 5.1. *Let $\mathcal{S} = (a, I, \tau)$ be an array-based system and $(\mathcal{S}, U)$ be a safety problem satisfying assumptions **(HT1)**, **(HT2)**, **(HF1)**, **(HF2)**, and **(wHF3)** in Table 1, i.e. τ is in weak functional form. Then, $\tilde{\mathcal{S}}$ is safe with respect to $\tilde{U}$ iff $\widehat{\mathcal{S}}$ is safe with respect to $\widehat{U}$.*

Proof. We apply Lemma 5.3 with the identity relation as the bisimulation. The proof can be concluded by observing that the transition $\widehat{\tau}$ can be simulated by finitely many applications of the additional transition $\tilde{t}_{n+1}$ in the intermediate system of Table 2 followed by a single application of $\tilde{\tau}$; conversely, each application of $\tilde{\tau}$ is obviously also an application of $\widehat{\tau}$. □

The importance of Theorem 5.1 lies in the fact that the transition $\widehat{\tau}$ of $\widehat{\mathcal{S}}$ is in functional form and it does not have universal quantifiers in guards. This allows us to automate BReach as explained in Section 4.

5.4 Invariants and relativization of quantifiers

It is well-known that invariants can dramatically prune the search space of backward reachability procedures; see [13, 15] for an in-depth discussion about this issue and related experiments. Here, we briefly recall the notion of invariant for array-based systems and explain how they can be used in combination with the technique for eliminating universal conditions from guards.

A $\forall^I$-formula $\forall \underline{i}\, \phi(\underline{i}, a[\underline{i}])$ is an *(inductive) invariant* for an array-based system $\mathcal{S} = (a, I, \tau)$ iff the formulae (a) $I \Rightarrow \forall \underline{i}\, \phi(\underline{i}, a[\underline{i}])$ and (b) $\forall \underline{i}\, \phi(\underline{i}, a[\underline{i}]) \wedge \tau(a, a') \Rightarrow \forall \underline{i}\, \phi(\underline{i}, a'[\underline{i}])$ are both A_I^E-valid. If, in addition to (a) and (b), also the formula (c) $\forall \underline{i}\, \phi(\underline{i}, a[\underline{i}]) \wedge U(a)$ is A_I^E-unsatisfiable, then we say that the $\forall^I$-formula $\forall \underline{i}\, \phi(\underline{i}, a[\underline{i}])$ is a *safety invariant* for the safety problem $(\mathcal{S}, U)$. Notice that the A_I^E-validity and A_I^E-satisfiability tests required in (a), (b), and (c) are decidable by Theorem 4.1 under the hypothesis **(HT1)**, **(HT2)**, **(HF1)**, **(HF2)**, and **(wHF3)** in Table 1. Thus, in a sense, using invariant for the verification of safety properties—called the *invariant method* in the literature—is more general than using a backward reachability procedure as we propose in Section 4. Unfortunately, the invariant method is more difficult to apply because finding safety invariants cannot be mechanized (see again [13, 15] for details). However, if an invariant has been found in some way (for instance, by using some heuristics) or it is supplied by a trusted third-party, the invariant can be put to productive use (even if it is not a safety invariant) in the backward reachability procedure by adding it as an extra hypothesis in the fix-point test. More precisely, line 2 of Breach in Figure 1 can be substituted by the following instruction:

$$2' \ \textbf{while}\ (P \wedge Inv \wedge \neg B \text{ is } A_I^E\text{-sat.})\ \textbf{do}$$

where Inv is a conjunction of the available invariants, i.e. $\forall^I$-formulae satisfying conditions (a) and (b) above. In this way, we try to augment the possibility to detect unsatisfiability by further constraining the formula $P \wedge \neg B$ with the invariants in Inv. As shown in [13, 15], this modification may have dramatic effects on the performance of the backward reachability procedure; in some cases, we may even get a safety certificate by using invariants while the procedure does not terminate without them.

Combining invariants with the technique for eliminating universal conditions in guards described above is straightforward; it is sufficient to relativize the universal quantifiers occurring in $\forall^I$-formulae. More precisely, we extend the mapping $(\tilde{\cdot})$ to invariants as follows:

$$\forall \underline{i}\, \phi(\underline{i}, a[\underline{i}]) \quad \longmapsto \quad \forall \underline{i}\, (\mathtt{A}(\underline{i}) \Rightarrow \phi(\underline{i}, a[\underline{i}]))$$

where $\mathtt{A}$ is defined as in Table 2. The mapping $(\hat{\cdot})$ acts as the identity on invariants. It is easy to adapt the results of Sections 5.2 and 5.3 for the two mappings extended in this way.

6. Related Work

Relationship with failure models. A (partial) failure of a distributed system may happen when one of its components fails (see, e.g., [27]). The failure may affect the operation of some components while leaving others unaffected. It is crucial to build distributed systems which are tolerant to partial failures. The task of designing and verifying such systems is complex and error prone; for this reason, some methodologies have been proposed for their modular design (see, e.g., [19]). Abstractly, a (partial) failure can be characterized as a deviation from the correct behavior. For example, a process can crash at any time and after doing so it stops executing further computation steps. This is called the *crash failure model* (CFM) or the stopping-failure model in [22]. It is possible to characterize a hierarchy of increasingly more complex failure models, one including the type of failures considered by the simpler ones. For example, in the *send-omission failure model* (SOFM), a process may omit to send a message besides crashing as in the CFM; in the *general-omission failure model* (GOFM) a process may fail to perform some prescribed (e.g. send or receive) action besides crashing and, worst of all, in the *Byzantine* failure model a process may arbitrarily deviate from its behavior, i.e. it can exhibit any behavior whatsoever. The reader interested in having more details about failure models is pointed to standard textbooks on distributed systems, e.g., [22, 27].

We claim that the transformation $\tilde{(\cdot)}$ is equivalent to adopting the CFM when array-based systems model distributed systems. To see this, let us assume that the array-based system $\mathcal{S} = (a, I, \bigvee_{\ell=1}^{n} t_\ell(a, a'))$ models a certain distributed system and consider the intermediate system $\tilde{\mathcal{S}}$ of Table 2. If we interpret $a_{s+1}[i] = \mathtt{f}$ as "the process i has crashed" or, equivalently, $\mathtt{A}(i)$ as "the process i is not faulty according to the CFM," then the transition formula $\tilde{t}_{n+1}(\tilde{a}, \tilde{a}')$ models the fact that a process may crash at any time. Furthermore, the way we modified the update functions from F to $\tilde{F}$ models the fact that crashed processes remain crashed in the CFM; the relativization of quantifiers in the guards of transitions implies that faulty processes cannot contribute to any run of the system; finally, the relativization of quantifiers in the unsafe formula means that faulty processes cannot be used as witnesses of undesired behaviors of the system. As a consequence, $\tilde{\mathcal{S}}$ models the computations of $\mathcal{S}$ when assuming the CFM.

Notice that the syntactic transformation described in Section 5 can be applied to any array-based system, regardless of it modeling a distributed system or not. When modeling other types of systems (e.g., imperative programs manipulating arrays), finding an intuitive reading of the proposed transformations is more difficult, if possible at all. However, from the point of view of formal verification, the transformation can be performed and if the backward reachability analysis terminates with a global fixpoint, meaning that it is not possible to reach an unsafe state from an initial one, then we are entitled to conclude the safety of the original system.

Relationship with other verification techniques. The use of *monotonic abstractions*, which are similar to the CFM, has been advocated by Abdulla *et al* [3, 4, 5, 1]. The key idea is to use backward reachability on configurations, which are used as finite representations of infinite sets of state. More precisely, a well-quasi-order (roughly, a pre-order which guarantees the termination of the backward reachability procedure) on the set of configurations is defined, so that a configuration may be used as a finite symbolic representation for a certain class of sets of states (see, e.g., [3] for more details). To overcome the problem that universal conditions in the guards of transitions prevent monotonicity, an abstract transition relation is defined that forces monotonicity whereby processes inside a configuration violating a universal condition are deleted from the configuration. In this way, an over-approximation of the original transition is obtained and the safety of this new system implies the safety of the original one. The results in [3, 4, 5] can be derived in our framework by exploiting the notion of configuration induced by $\exists^I$-formulae (recall the formal definition at the beginning of Section 5.3). For a formal account of the relationship between our approach and that of Abdulla *et al*, the reader is pointed to [15]. We emphasize that our transformation, being purely syntactic, can be applied to any array-based system whereas the approach in [3, 4, 5, 1], being more semantic, needs to be adapted each time a new class of systems is considered. This adaptation can become problematic whenever the modeling language becomes expressive; for instance, when registers containing process identifiers are required.

More recently, monotonic abstractions have been combined with the paradigm of counter-example-guided abstraction (CEGAR) [2], by computing interpolants in the theory of the elements manipulated by a parameterized system. It would be natural to adapt such an approach in our framework which is fully declarative. We leave this to future work. In the context of the CEGAR approach to model checking infinite state systems, an alternative and long line of works goes back to the seminal paper [18] which introduces the technique of *predicate abstraction*. The idea is to abstract a parametric system to a finite state system, perform model checking, and then refine spurious traces (if any) by using decision procedures or SMT solvers. This approach has been implemented in several tools and is often combined with interpolation algorithms for the refinement phase. A major problem with predicate abstraction (as already pointed out in [21]) is that it must be carefully adapted when (universal) quantification is used to specify the transitions of the system or its properties, as it is the case for the verification problems considered in this paper.

An alternative approach to the verification of parametric system was initiated by McMillan in [23]. This work starts by observing that some classes of parametric systems can be specified in terms of the cardinality n of a parameter set (e.g., the set of processes in a fault-tolerant algorithm). So, the idea is to establish a cut-off value $\overline{n}$ for n so that the corresponding instance of the parametric system exposes all the possible behaviors which are relevant for the verification of certain properties. There are two drawbacks to this approach. First, finding the cut-off value is difficult and crucially depends on the topology of systems. Second, the value of the cut-off is usually high and the verification of the corresponding instance of the parametric system turns out to be a computationally expensive task. Several papers—such as [20, 26, 8, 24, 10]—propose refinements in order to alleviate the two problems identified above and make the approach more scalable.

7. The Case Study: Reliable Broadcast

As discussed above, the task of designing and verifying fault tolerant distributed systems is complex and error prone. Methodologies for their modular construction usually start with the simplest failure model (i.e. CFM) and then refine the algorithms in order to consider more realistic types of failures. This is exemplified in [28] where some classical algorithms for reliable broadcast (i.e. a process broadcasts a message and all non-faulty processes consistently decide on delivering that message) are developed and pen-and-paper proofs of correctness are also given. The paper is part of a long line of research which focuses on providing communication primitives, which are perceived as one of the weakest points of any distributed system. The algorithms described in [28] are designed for reliable broadcast in increasingly more complex failure models and thus constitute ideal benchmarks for the evaluation of the practical viability of our techniques. We discuss how the safety of three of the algorithms in [28] can be automatically verified by MCMT. Before describing our experiments, we explain how the technique to handle global conditions in guards have been implemented in the tool.

7.1 Eliminating universal conditions: implementation

MCMT [16] implements the procedure BReach in Figure 1 combined with the syntactic transformation for the elimination of universal conditions described in Section 5.[6.] It is based on a client-server architecture: the client iteratively computes pre-images and generates the satisfiability problems encoding the safety and fix-point checks, and the server is the combination of the SMT solver Yices [11] with a dedicated quantifier-instantiation procedure for universal quantifiers in $\exists^{A,I}\forall^{I}$-sentences (see [15, 14] for details).

In MCMT, the backward reachability procedure attempts to prune the search space by using invariants as described in Section 5.4. Powerful heuristics for the synthesis of invariants have been designed and implemented [13, 15, 16]. The advantages of using invariant synthesis on a substantial set of benchmarks is documented in [15]. Both synthesized and user-provided invariants can be used in MCMT without compromising its correctness as the tool may be instructed to preliminarily check that the invariant under consideration is really so and only afterwards be used in the fix-point checks of the backward reachability procedure. Invariants can also be used to simplify the specification of an array-based system. For example, it is possible to model *shared registers*, i.e. variables shared among the processes of a distributed systems, as follows. Let c be an array variable that should model a shared register. Then, we add $\forall i, j.(c[i] = c[j])$ to the initial formula, update the array c by using a constant update function f satisfying the invariant $\forall i, j.(f[i] = f[j])$. In an MCMT input file, all this is done simply by declaring c as a "global" variable. In a similar vein, it is possible to model the situation where the values stored in an array, say b, are integers to be used as indexes of other processes. To cope with this, we introduce an integer valued array id, whose interpretation is fixed over the runs of the array-based system (or, equivalently, there is no transition modifying its values) and is constrained by the following invariant $\forall i, j.(id[i] = id[j] \Rightarrow i = j)$, saying that the functions interpreting id are injective. As a consequence, we may regard id as an array of process identifiers. If we then write $b[j] = id[j]$, or, for simplicity, $b[i] = j$, we can express the fact that $b[i]$ is equal to the identifier of process j.

MCMT implements the transformations of Table 2 in Section 5 from the original to the target array-based systems and invokes the backward reachability procedure on the latter. Upon termination with unsafe, if at least one universal condition in the guard of a transition has been encountered, the tool prints out a message that warns the user about the adoption of a possibly approximated model. The user is responsible to verify if the conditions of Theorem 5.1 hold or not (this is reasonable since, in general, checking conditions **(HT1)** and **(HT2)** is undecidable). The implementation in MCMT of the syntactic transformations for eliminating universal conditions slightly differs from Table 2. This is so because MCMT manipulates sub-classes of the formulae considered in this paper to describe sets of states

6. The executable of MCMT, several benchmark problems, the documentation, and related papers can be downloaded at `http://homes.dsi.unimi.it/~ghilardi/mcmt`.

```
Initialization:
    if ($p$ is the sender)
        then $estimate[p] \leftarrow m$; $\underline{coord\_id[p] \leftarrow 0}$;
        else $estimate[p] \leftarrow \bot$; $\underline{coord\_id[p] \leftarrow -1}$;
    $state[p] \leftarrow undecided$;
End Initialization

for $c \leftarrow 1, 2, ..., n+1$ do        // Process $c$ becomes coordinator for four rounds
    Round 1:
        All $undecided$ processes $p$ send $request(\underline{estimate[p], coord\_id[p]})$ to $c$;
        if ($c$ does not receive any request) then it skips rounds 2 to 4;
            $\underline{\textbf{else } estimate[c] \leftarrow estimate[p] \text{ with largest } coord\_id[p]}$;
    Round 2:
        $c$ multicasts $estimate[c]$;
        All $undecided$ processes $p$ that receive $estimate[c]$ do
            $estimate[p] \leftarrow estimate[c]$ $\underline{\textbf{and } coord\_id[p] \leftarrow c}$;
    Round 3:
        All undecided processes $p$ that do not receive $estimate[c]$ send(NACK) to $c$;
    Round 4:
        if ($c$ does not receive any NACK) then $c$ multicasts $Decide$; else $c$ HALTS;
        All $undecided$ processes $p$ that receive $Decide$ do
            $decision[p] \leftarrow estimate[p]$;
            $state[p] \leftarrow$ DECIDED;
end for
```

Pseudo-code 1: *Algorithms 1, 1e, and 2 of [28]*

and transitions (see footnote 3. at page 38 and, e.g., [13, 15] for more details). This implies a more involved definition of the syntactic transformation aimed to produce formulae in the accepted classes. We omit the details as they are conceptually irrelevant for the discussion of the results in this paper. Another difference concerns MCMT's (implicit) assumption that *all the quantifiers over indexes it works with are relativized* so that it never displays those parts of formulae concerning the relativization; thereby greatly improving the readability of its output.

7.2 Reliable Broadcast

Roughly, a reliable broadcast algorithm ensures that when a process broadcasts a message, all non-faulty (with respect to a given failure model) processes consistently decide on the value of the message. Its crucial safety property is called *agreement* and can be formulated as follows: if a non-faulty process decides the message m, then all non-faulty processes must also decide m. In [28], some algorithms providing efficient (both in time and number of messages) reliable broadcasts are described for the CFM, the SOFM, and the GOFM. Here, for simplicity, we discuss only the first three algorithms of [28] for the CFM and the SOFM. The algorithm for the GOFM is omitted because, besides containing universal conditions in guards, it also features substantial arithmetic reasoning for which we are still developing suitable techniques.

The Pseudo-code 1 contains three round-based algorithms. Each round is split in two phases: in the former, each process sends a message to some or all of the other processes (the content of the messages will depend on the current state of the sending process) and in the latter, each process changes its state according to its current state and the collection of messages it received in the first phase. Messages are transferred instantaneously from senders to recipients between the two phases. The processes operate in lockstep: all of them perform the two phases of the current round, then move on to the first phase of the next round, and so on. The algorithms are assumed to be parametric in the number n of processes and each process is connected to any other process of the system. Cyclically, n processes become coordinator for a constant number of rounds each. Once elected, a coordinator determines a decision value and convinces all the other processes to agree on the same value. Since at most $n-1$ coordinators can be faulty and, in each round, all the messages are to/from the coordinator, the algorithms are efficient both in time, as agreement is reached in $O(n)$ rounds, and number of messages, which are $O(n \cdot k)$ where $O(k)$ is the number of messages sent per round. Each process p can be in one of two states, DECIDED or UNDECIDED, and maintains a variable $\mathit{estimate}[p]$ representing p's current estimate of the final decision. Algorithm 1 in [28] can be read in the Pseudo-code 1 by ignoring the instructions which are underlined; to derive algorithm 1e, the reader should consider also the underlined instructions involving only the NACK variable; and for algorithm 2, consider all the instructions in the pseudocode regardless of the fact that they are underlined or not. The goal of the algorithms is to ensure the agreement property in the CFM (algorithm 1) or in the SOFM (algorithms 1e, which is bugged, and 2). For a more detailed discussion of the algorithms, the reader is pointed to [28].

We have translated the three algorithms in a high-level syntax accepted by the tool which provides some convenient syntactic sugar to specify array-based systems and safety problems as defined in this paper. For the sake of conciseness, we illustrate only some of our modelling choices.

The theories T_I and T_E that we need to model the topology and the data-structures manipulated by the algorithms are similar to those used in the running example of this paper and certainly satisfy assumptions **(HT1)** and **(HT2)** above for each one of the three algorithms of Pseudo-code 1.

If an algorithm comprises, say, three rounds (as it is the case for algorithm 1), we introduce a shared register variable *round* (see Section 7.1) storing the current *round* of execution encoded as an integer. Since the number of rounds is bound to three, besides the invariant constraining the array variable round to be constant, we can also require that $\forall i.(1 \leq \mathit{round}[i] \wedge \mathit{round}[i] \leq 3)$.

At the beginning of each algorithm, no process of the system has yet decided on the value of the message being broadcast (represented as $\mathit{state}[i] = \mathit{false}$), no one is or has been the coordinator ($\mathit{coord}[i] = \mathit{false}$, $\mathit{aCoord}[i] = \mathit{false}$), no message request has been sent by any process ($\mathit{request}[i] = \mathit{false}$) and the system is at the beginning of the first round ($\mathit{round}[i] = 1$, $\mathit{done}[i] = \mathit{false}$). This can be formalized by the following $\forall^I$-formula:

$$\forall i. \left(\begin{array}{lclcll} \mathit{round}[i] = 1 & \wedge & \mathit{state}[i] = \mathit{false} & \wedge & \mathit{coord}[i] = \mathit{false} & \wedge \\ \mathit{aCoord}[i] = \mathit{false} & \wedge & \mathit{done}[i] = \mathit{false} & \wedge & \mathit{request}[i] = \mathit{false} & \end{array} \right).$$

Thus, assumption **(HF1)** is also satisfied.

Recall that the key safety property for our algorithms is agreement. The complement of this property (characterizing the set of unsafe states) can be represented by the following $\exists^I$-formula:

$$\exists i_1, i_2.(i_1 \neq i_2 \wedge \mathit{state}[i_1] = \mathit{true} \wedge \mathit{state}[i_2] = \mathit{true} \wedge \mathit{decision}[i_1] \neq \mathit{decision}[i_2])$$

so that also assumption **(HF2)** holds.

To illustrate how we specify that a process goes from one round to the next, let us consider the following universally guarded transition formula in functional form—thus, also the last assumption **(wHF3)** is satisfied—modeling a transition from the first to the second round:[7.]

$$\exists i. \left(\begin{array}{l} \mathit{round}[i] = 1 \wedge \mathit{request}[i] = \mathit{true} \wedge \\ \mathit{coord}[i] = \mathit{true} \wedge \forall j.(\mathit{done}[j] = \mathit{true}) \wedge \\ \mathit{round}' = \lambda j.2 \end{array} \right),$$

i.e. if the current round is 1, process i has already received a request, it is the coordinator, and all processes have finished the execution of that round, then the shared register variable *round* is set to 2. Notice that all the other state variables are unchanged by the transition and the update of *round* maintains the invariant $\forall i, j.(\mathit{round}[i] = \mathit{round}[j])$ introduced when declaring it as a shared register.

We recall that the correct reading of the $\exists^I$-formula expressing the complement of the agreement property is the following: there exist two distinct and *non-faulty* processes i_1 and i_2 such that the values of *state* at i_1 and i_2 are true and the values of *decision* at i_1 and i_2 are distinct. Notice that the two processes to be considered should be non-faulty since in MCMT, when using the CFM, all the process (index) variables are implicitly relativized to non-faulty (see again Section 7.1). Similar remarks hold also for the initial state formula and the transition formulae above.

7.3 Reliable broadcast and more complex failure models

While MCMT provides native support for the CFM, the handling of the SOFM (or more complex failure models, such as the GOFM) is difficult as it must be done manually. To simplify this task, we have developed a methodology that can also be mechanized in MCMT although we leave this to future work. The capability of specifying different failure models may be important for developing new abstractions to be used in the automated verification of distributed systems. In fact, as the syntactic transformation in Table 2 can be seen as the formal counterpart of the CFM (as discussed in Section 6), more complex failure models (such as the SOFM) may induce new over-approximations that can be put to productive work in model checking. We leave the evaluation of this kind of abstractions to future work. Instead, in the rest of this section, we show how the capability of specifying complex failure models gives us the possibility to more faithfully represent the environments in which fault tolerant algorithms are supposed to work for the case of reliable broadcast algorithms. As a preliminary, we sketch our methodology which is based on the notion of *action omission*

7. In this section, for simplicity, when writing transition formulae, we omit to specify the update of the array variable which are unchanged.

model (AOM). Let $\mathcal{S} = (a, I, \tau)$ be an array-based system such that $\tau := \bigvee_{\ell=1}^{n} t_\ell(a, a')$. Assume that τ_ℓ has the form

$$\exists k\, \exists \underline{i}\ (\phi_L(\underline{i}, k, a[\underline{i}], a[k]) \wedge a_h[k] = \mathtt{f} \wedge a' = \lambda j.F(\underline{i}, k, a[\underline{i}], a[k], j, a[j]))\,, \qquad (9)$$

for some $\ell = 1, \ldots, n$, where a_h is an array variable whose values can either be $\mathtt{t}$ or $\mathtt{f}$, and the h-th component of the vector F of functions is $\mathtt{upd}(a, k, \mathtt{t})$. The role of a_h is that of a flag indicating that a process k has performed the action (9). Once the action has been executed, the "*done?*" flag a_h changes its value from $\mathtt{f}$ to $\mathtt{t}$. We want to introduce the possibility for a process k to *omit the action expressed by* τ_ℓ: in this case, the flag $a_h[k]$ will again change its value from $\mathtt{f}$ to $\mathtt{t}$ but, since the action has been omitted, the update function F is replaced by an identical update function for all array variables a_l ($l \neq h$). An extra flag is added to record the fact that, from now on, process k has to be considered faulty according to the AOM, because it failed to execute the action expressed by the transition τ_ℓ. The *AOM version* $(\overline{\mathcal{S}}, \overline{U})$ *of a safety problem* $(\mathcal{S}, U)$ *for* U *an unsafe formula* is defined by the following purely syntactic transformations. Let $\overline{a}$ be a extended with an auxiliary array variable a_{s+1} of sort $\mathtt{ARRAY}_{s+1}$ as specified in Table 2, which indicates that a process is faulty or not according to the AOM, i.e. that the process has omitted an action of the form (9). Each t_m (for $m \neq \ell$) is left unchanged (in particular, the auxiliary variable a_{s+1} is updated identically); t_ℓ is replaced by the following pair of disjuncts:

$$\exists k\, \exists \underline{i} \left(\begin{array}{l} \phi_L(\underline{i}, k, a[\underline{i}], a[k]) \wedge a_h[k] = \mathtt{f} \wedge \\ a' = \lambda j.F(\underline{i}, k, a[\underline{i}], a[k], j, a[j]) \wedge a'_{s+1} = a_{s+1} \end{array} \right)$$

$$\exists k\, \exists \underline{i} \left(\begin{array}{l} \phi_L(\underline{i}, k, a[\underline{i}], a[k]) \wedge a_h[k] = \mathtt{f} \wedge \\ \bigwedge_{l \leq s, l \neq h} a'_l = a_l \wedge a'_h = \mathtt{upd}(a_h, k, \mathtt{t}) \wedge a'_{s+1} = \mathtt{upd}(a_{s+1}, k, \mathtt{f}) \end{array} \right)$$

corresponding to the cases in which the process k performed or omitted to perform, respectively, the action (9). In the second case, the process k is considered faulty (from there on) and the value of $a'_{s+1}[k]$ is updated accordingly. The initial formula $\overline{I}$ becomes $I \wedge \forall j.(a_{s+1}[j] = \mathtt{t})$. The precise specification of the transformation into the AOM of an array-based system depends on the details of the application we are considering. In particular, when building $(\overline{\mathcal{S}}, \overline{U})$, we may or may not relativize existentially quantified variables in the unsafe formula (and in the guards of some transitions) to processes which are non-faulty with respect to AOM. For example, a process that has been elected as coordinator in a distributed algorithm can continue (or not) to play its role even after omitting an important action, depending on what is the behavior of the system. This is the main difference with respect to the CFM, which always assume that, after crashing, the process is unable to perform any further computations. Interestingly, we can modularly combine AOM transformations by starting with an array-based system $(\mathcal{S}, U)$, deriving its AOM version $(\overline{\mathcal{S}}, \overline{U})$, and then obtaining an AOM version $(\overline{\overline{\mathcal{S}}}, \overline{\overline{U}})$ of the latter by selecting a different disjunct of the form (9) in τ. This chain of AOM transformations is useful to refine the failure model under consideration. We will see an application of this idea in the next section when the CFM is considered first and then refined to the SOFM.

We now use the notion of AOM to derive the SOFM for the reliable broadcast algorithm considered here. The key point is the addition of the array state variable a_{s+1} which will be denoted by $faulty$, mapping indices to Booleans such that $faulty[i]$ is false (true) when the process i is correct (faulty, respectively) with respect to the SOFM (i.e. besides crashing, the process i may also omit to send a message). Indeed, the new variable must be mentioned in the safety properties to be checked and explicitly updated by the transitions specified by the user so as to model the situation when the process executing a transition is correct or faulty. Since the failures due to the CFM are still implicitly handled by mcmt, we only need to take into consideration the send omission failures. To illustrate the impact of doing this on the specification, let us consider the $\exists^I$-formula describing the complement of the agreement property in the SOFM:

$$\exists i_1, i_2.(i_1 \neq i_2 \wedge \left(\begin{array}{c} \mathit{state}[i_1] = \mathit{true} \wedge \mathit{faulty}[i_1] = \mathit{false} \wedge \\ \mathit{state}[i_2] = \mathit{true} \wedge \mathit{faulty}[i_2] = \mathit{false} \wedge \\ \mathit{decision}[i_1] \neq \mathit{decision}[i_2] \end{array} \right)),$$

where it is explicitly required that the two processes i_1 and i_2 are non-faulty (i.e. the flag *faulty* is false at both i_1 and i_2). For transitions, let us consider the situation where the undecided processes send a request to the coordinator in the first round. In CFM, this transition can be formalized by the following formula:

$$\exists i_1, i_2.(i_1 \neq i_2 \wedge \left(\begin{array}{l} \mathit{round}[i_1] = 1 \wedge \mathit{done}[i_1] = \mathit{false} \wedge \mathit{state}[i_1] = \mathit{false} \wedge \\ \mathit{coord}[i_2] = \mathit{true} \wedge \mathit{request}' = \lambda j.\mathit{true} \wedge \\ \mathit{done}' = \mathtt{upd}(\mathit{done}, i_1, \mathit{true}) \end{array} \right)).$$

When considering the SOFM, we also need to specify the situation when the sending of the request to the coordinator (process i_2) from process i_1 is not successful. This can be formalized as follows:

$$\exists i_1, i_2.(i_1 \neq i_2 \wedge \left(\begin{array}{l} \mathit{round}[i_1] = 1 \wedge \mathit{done}[i_1] = \mathit{false} \wedge \\ \mathit{state}[i_1] = \mathit{false} \wedge \mathit{coord}[i_2] = \mathit{true} \wedge \\ \mathit{faulty}' = \mathtt{upd}(\mathit{faulty}, i_1, \mathit{true}) \wedge \\ \mathit{done}' = \mathtt{upd}(\mathit{done}, i_1, \mathit{true}) \end{array} \right)).$$

A final remark is in order. Besides the transitions formalizing the various instructions in the pseudo-code of the algorithm, some transitions have been added so as to (i) allow progress of the system in case no requests are received in Round 1; (ii) describe the switching to a new coordinator either at the end of Round 4 for the current one, or when the current coordinator crashes; (iii) specify the behavior of already decided processes. For a description of the high-level syntax and the formal specifications of all the problems considered here, the reader is referred to `http://www.inf.usi.ch/phd/alberti/mcmt/reliableBroadcast/`.

7.4 Experimental results

In [28], the algorithms are presented step-wise. Algorithm 1 is first proved correct for the CFM. Then, the unsafety for the SOFM is shown. An error trace violating agreement is considered and a first refinement is proposed and called algorithm 1e. This is still proved unsafe for the SOFM so that a further refinement, called algorithm 2, is proposed and shown safe for the SOFM, but unsafe for the GOFM. In the following, we follow the exposition of [28] by replacing pen-and-paper proofs or the manual definition of error traces with the use of mcmt. The behavior of the tool on the various problems is summarized in Table 3. The first line reports whether the algorithm satisfies the agreement property in the chosen failure model; the second line shows the time (in seconds) taken by mcmt to find the answer to the safety problem. The following two lines gives a rough idea of the dimension of the specifications by reporting the number of array state variables (line 3) and that of transitions (line 4) in the array-based system. The remaining lines record some statistics output by the tool: the number of disjuncts of the formula (in disjunctive normal form) representing the set of backward reachable states, when exiting the procedure of backward reachability (line 5) gives a rough idea of the size of the explored search space, and the number of invocations of the SMT solvers (line 6) gives an idea of the complexity of solving the satisfiability problems encoding the safety and fix-point checks. The last three lines deserve some more comments.

The length of the unsafe trace (line 7) suggests how complex it is to consider all the possible interleavings of the transitions for this class of algorithms (notice that the demonstration of the unsafety of algorithm 1e in the SOFM requires a sequence of 33 transitions). Unsafe traces are automatically computed by mcmt and are extremely useful for debugging. For example, when considering algorithm 1 with the SOFM, mcmt discovers an error trace of length 11 that, manually translated in the form of a message sequence chart for better readability, is shown in Figure 2. This trace involves only one coordinator whereas the trace considered in [28] is longer as it involves two coordinators. Each time a new coordinator is elected, the algorithm re-starts from Round 1; thus, the more coordinators are elected, the longer is the trace. By analyzing the error trace, according to what is suggested in [28], one can come up with the idea of using an additional variable for negative acknowledgment (nack) to be used in the additional Round 3 and the conditional statement in Round 4 (i.e. algorithm 1e in Pseudo-code 1). Algorithm 2 is obtained in a similar way by analyzing the longer trace produced by mcmt when checking the safety of algorithm 1e in the SOFM. The idea here is to augment the information sent around by processes; namely, when sending a request, they should also include their current *estimate* and the identity of the coordinator from which it has been received. In this way, the current coordinator does not impose its

Table 3. Performances of (dis-)proving agreement for algorithms 1, 1e, and 2

	Algo. 1, CFM	Algo. 1, SOFM	Algo. 1e, SOFM	Algo. 2, SOFM
Safe (agreement)	Yes	No	No	Yes
time (sec)	0.87	11.35	1,255.51	3,846.82
# state vars	8	9	11	15
# transitions	13	16	22	28
size fix-point	92	438	8,909	9,916
# SMT calls	2,399	19,733	1,338,058	2,563,756
Length unsafe trace	×	11	33	×
Max # processes	4	3	4	6
# invariants	×	×	×	19 (+**7**)

Timings are obtained by running mcmt on a machine equipped with an Intel Core i7 @ 2.80GHz CPU with 8 GB of RAM.

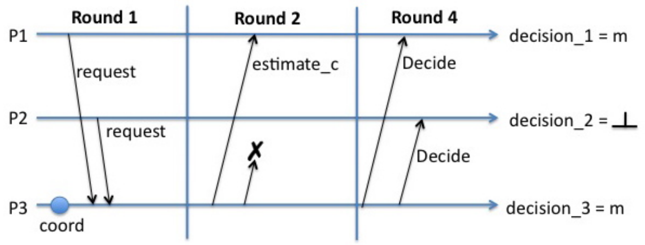

Figure 2. Unsafe trace found by MCMT for algorithm 1 in the SOFM

own decision; rather, it adopts and circulates the *estimate* associated with the most recent coordinator (i.e. algorithm 2 in Pseudo-code 1).

Line 8 shows the maximum number of (distinct) process identifiers which are mentioned in the formula representing the set of backward reachable states. Initially, only two such identifiers are mentioned in the unsafe formula (representing the complement of agreement). However, MCMT may insert additional processes identifiers when needed to enable the symbolic (backward) execution of a transition. For efficiency, heuristics have been implemented in the tool to perform this addition lazily (see [14, 16] for more details on this point). Here, it is worth noticing that all problems need to consider twice or three times the number of process identifiers mentioned in the set of unsafe states. Automatically detecting when enough process identifiers have been inserted to prove safety is one of the main sources of complexity in handling parametric verification problems.

The last line of Table 3 shows the number of invariants which have been used to prune the search space of the backward reachability procedure as explained in Section 7.1. We have used invariants only for solving the last safety problem as the others are solved in a reasonable time without them. The fully automatic verification of agreement for algorithm 2 in the SOFM is quite problematic: MCMT runs for several hours without finding a fix-point. Although quite effective in pruning the search space for the verification of many safety problems [13, 15], even the invariant synthesis capabilities of the tool [13, 15] do not help pruning the search space enough to find a fix-point in a reasonable amount of time. Fortunately, the possibility to interact with the tool by proving simpler properties (that an expert in fault reliable broadcast algorithms is able to identify) and then telling MCMT to exploit them for proving more difficult ones is the key to the result recorded in the last column, last line of Table 3: 19 is the number of invariants that have been automatically synthesized by the heuristics explained in [13, 15] whereas 7 is the number of invariants that have been suggested by us. As discussed above, the inclusion of these invariants does not affect the validity of the final result of the verification process, since they are verified by the tool before being used for the validation of other properties. The 7 invariants suggested by us are simple properties that do not require deep understanding of the algorithm; e.g., "there is only one coordinator at a time."

The tool also easily validates the three lemmas used in [28] to perform the pen-and-paper proof of the correctness of the algorithm.

In summary, these experimental results suggest the practical viability of our techniques to handle global conditions in guards when combined with symbolic backward reachability for the verification of fault tolerant algorithms.

8. Conclusions

We have formally described a syntactic transformation, which can be seen as an instance of the well-known operation of relativization of quantifiers in first-order logic, for array-based systems. The transformation allows us to eliminate global conditions in the transition formulae of array-based systems. Being syntactic in nature, the transformation is easy to grasp, simple to implement, and can be applied to a large class of array-based systems without further modifications. Crucially, it significantly extends the scope of applicability of the symbolic backward reachability procedure introduced in [15], which has proved to be useful in several verification problems. A significant case-study concerning the verification of some classical algorithms for reliable broadcast shows its usefulness and practical viability.

Acknowledgements. The work of the first author was supported by the Hasler Foundation under project 09047. The work of the fourth author was partially supported by the "Automated Security Analysis of Identity and Access Management Systems (SIAM)" project funded by Provincia Autonoma di Trento in the context of the "team 2009 - Incoming" COFUND action of the European Commission (FP7).

References

[1] P. A. Abdulla. Forcing monotonicity in parameterized verification: From multisets to words. In *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM '10, pages 1–15. Springer-Verlag, 2010.

[2] P. A. Abdulla, Y.-F. Chen, G. Delzanno, F. Haziza, C.-D. Hong, and Rezine A. Constrained monotonic abstraction: A cegar for parameterized verification. In *CONCUR 2010 - Concurrency Theory, 21th International Conference, Paris, France, August 31-September 3, 2010. Proceedings*, pages 86–101, 2010.

[3] P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Regular model checking without transducers. In *TACAS*, **4424** of *LNCS*, pages 721–736, 2007.

[4] P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *CAV*, **4590** of *LNCS*, pages 145–157, 2007.

[5] P. A. Abdulla, N. B. Henda, G. Delzanno, and A. Rezine. Handling parameterized systems with non-atomic global conditions. In *Proc. of VMCAI*, **4905** of *LNCS*, pages 22–36, 2008.

[6] F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, and G. P. Rossi. Automated Support for the Design and Validation of Fault Tolerant Parameterized Systems: a case study. In *Proc. of AVOCS 10*, Electr. Comm. of the EASST, 2010.

[7] F. Alberti, S. Ghilardi, E. Pagani, S. Ranise, and G. P. Rossi. Brief Announcement: Automated Support for the Design and Validation of Fault Tolerant Parameterized Systems—a case study. In *Proc. of DISC 10*, number 6343 in LNCS, pages 392–394, 2010.

[8] J. Bingham. Automatic Non-interference Lemmas for Parameterized Model Checking. In *Formal Methods in Computer Aided Design (FMCAD)*, 2008.

[9] Chen-Chung Chang and Jerome H. Keisler. *Model Theory*. North-Holland, Amsterdam-London, third edition, 1990.

[10] C.-T. Chou, P. K. Mannava, and S. Park. A simple method for parameterized verification of cache coherence protocols. In *FMCAD'04*, pages 382–398, 2004.

[11] B. Dutertre and L. De Moura. The yices smt solver. Technical report, Computer Science Laboratory, SRI International, 2006. Available at http://yices.csl.sri.com.

[12] H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York-London, 1972.

[13] S. Ghilardi and S. Ranise. Goal Directed Invariant Synthesis for Model Checking Modulo Theories. In *(TABLEAUX 09)*, LNAI, pages 173–188. Springer, 2009.

[14] S. Ghilardi and S. Ranise. Model Checking Modulo Theory at work: the integration of Yices in MCMT. In *AFM 09 (co-located with CAV09)*, 2009.

[15] S. Ghilardi and S. Ranise. Backward reachability of array-based systems by SMT-solving: termination and invariant synthesis. *Logical Methods in Computer Science*, **6**(4), 2010.

[16] S. Ghilardi and S. Ranise. MCMT: A Model Checker Modulo Theories. In *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, **6173** of *Lecture Notes in Computer Science*, pages 22–29. Springer, 2010.

[17] S. Ghilardi, S. Ranise, and T. Valsecchi. Light-Weight SMT-based Model-Checking. In *Proc. of AVOCS 07-08*, ENTCS, 2008.

[18] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Proc. of CAV 1997*, **1254** of *LNCS*. Springer, 1997.

[19] V. Hadzilacos and S. Toueg. *Fault-tolerant broadcasts and related problems*, pages 97–145. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.

[20] S. Kristić. Parameterized system veriication with guard strengthening and parameter abstraction. In *Automated Verification of Infinite-State Systems*, 2005.

[21] S. K. Lahiri and R. E. Bryant. Predicate abstraction with indexed predicates. *ACM Transactions on Computational Logic (TOCL)*, **9**(1), 2007.

[22] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[23] K. L. McMillan. Parameterized verification of the FLASH cache coherence protocol by compositional model checking. In *Correct Hardware Design and Verification Methods (CHARME)*, pages 179–195, 2001.

[24] A. Pnueli, S. Ruath, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *Proc. of TACAS 2001*, **2031** of *LNCS*, 2001.

[25] S. Ranise and C. Tinelli. The SMT-LIB Standard: Version 1.2. Technical report, Dep. of Comp. Science, Iowa, 2006. Available at http://www.SMT-LIB.org/papers.

[26] M. Talupur and M. R. Tuttle. Going with the flow: Parameterized Verification Using Message Flows. In *Formal Methods in Computer Aided Design (FMCAD)*, 2008.

[27] A. S. Tanenbaum and M. Van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, 1st edition, 2001.

[28] S. Toueg and T. D. Chandra. Time and Message Efficient Reliable Broadcast. In *Proc. 4th Int. Workshop on Distributed Algorithms*, LNCS, pages 289–303, 1990.