Goal-directed Invariant Synthesis for Model Checking Modulo Theories

Silvio Ghilardi[1] and Silvio Ranise[2]

[1] Dipartimento di Informatica, Università degli Studi di Milano (Italy)
[2] Dipartimento di Informatica, Università di Verona (Italy)

Abstract. We are interested in automatically proving safety properties of infinite state systems. We present a technique for invariant synthesis which can be incorporated in backward reachability analysis. The main theoretical result ensures that (under suitable hypotheses) our method is guaranteed to find an invariant if one exists. We also discuss heuristics that allow us to derive an implementation of the technique showing remarkable speed-ups on a significant set of safety problems in parametrised systems.

1 Introduction

Backward reachability analysis has been widely adopted in model checking safety properties of infinite state systems (see, e.g., [1]). This verification procedure repeatedly computes pre-images of a set of unsafe states, usually obtained by complementing a safety property that a system should satisfy. Potentially infinite sets of states are represented by constraints so that pre-image computation can be done symbolically. A key advantage of backward reachability is to be *goal-directed*; the goal being the set of unsafe states from which pre-images are computed. Furthermore, safety properties for some classes of systems (e.g., broadcast protocols [8, 6]) can be decided by backward reachability.

Despite these advantages, backward reachability can unnecessarily explore (large) portions of the symbolic state space of a system which are actually not required to verify the safety property under consideration. Even worse, in some cases the analysis may not detect a fix-point, thereby causing non-termination. In order to avoid visiting irrelevant parts of the symbolic state space during backward reachability, techniques for analyzing pre-images and guessing invariants have been devised (see, e.g., [5, 15, 9, 4, 13] to name a few). The success of these techniques depend crucially on the heuristics used to guess the invariants. Our framework is similar in spirit to [5], but employs techniques which are specific for our different intended application domains.

Along this line of research, we present a technique for interleaving pre-image computation and invariant synthesis which tries to eagerly prune irrelevant parts of the search space. Formally, we work in the framework of the model checking (based on Satisfiability) Modulo Theories approach of [10, 12], where *array-based systems* have been introduced as an abstraction of several classes of infinite

state systems (such as parametrized systems, lossy channels, and algorithms manipulating arrays). The **main result** (cf. Theorems 4.9 and 4.11) of the paper ensures that the technique *will find an invariant—provided one exists—under suitable hypotheses*, which are satisfied for important classes of array-based systems (e.g., mutual exclusion algorithms or cache coherence protocols). The key ingredient in the proof of the result is the model-theoretic notion of configuration and configuration ordering (introduced in [1] at an abstract level) which allows us to finitely characterize the search space of candidate invariants. Although the technique is developed for array-based systems, we believe that the underlying idea can be adapted to other symbolic approaches to model checking (e.g., [2, 3]).

Plan of the paper. We briefly introduce the notion of array-based system (Sec. 2). We revisit the backward reachability procedure *as a Tableaux-like calculus* (Sec. 3) so as to give a firm basis for implementation. We show how invariants can help backward reachability (Sec. 4), recall the duality between this and the synthesis of invariants (Sec. 4.1), and describe how to interleave backward analysis and invariant synthesis (Sec. 4.2) together with some heuristics (Sec. 4.3). Finally (Sec. 5), we discuss how a prototype implementation of our techniques shows remarkable speed-ups. Full proofs and more examples can be found in the technical report [11].

2 Formal Preliminaries

We assume the usual syntactic (e.g., signature, variable, term, atom, literal, and formula) and semantic (e.g., structure, sub-structure, truth, satisfiability, and validity) notions of first-order logic (see, e.g., [7]). The equality symbol $=$ is included in all signatures considered below. A signature is *relational* if it does not contain function symbols and it is *quasi-relational* if its function symbols are all constants. An *expression* is a term, an atom, a literal, or a formula. Let $\underline{x}$ be a finite tuple of variables and Σ a signature; a $\Sigma(\underline{x})$-expression is an expression built out of the symbols in Σ where at most the variables in $\underline{x}$ may occur free (we will write $E(\underline{x})$ to emphasize that E is a $\Sigma(\underline{x})$-expression). Let $\underline{e}$ be a finite sequence of expressions and σ a substitution; $\underline{e}\sigma$ is the result of applying the substitution σ to each element of the sequence $\underline{e}$.

According to the current practice in the SMT literature [16], 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 Σ-formulae.

T admits *quantifier elimination* iff for every formula $\varphi(\underline{x})$ one can compute a quantifier-free formula $\varphi'(\underline{x})$ such that $T \models \forall \underline{x}(\varphi(\underline{x}) \leftrightarrow \varphi'(\underline{x}))$. A theory

$T = (\Sigma, \mathcal{C})$ is said to be *locally finite* iff Σ is finite and, for every finite set of variables $\underline{x}$, there are finitely many $\Sigma(\underline{x})$-terms $t_1, \ldots, t_{k_{\underline{x}}}$ such that for every further $\Sigma(\underline{x})$-term u, we have that $T \models u = t_i$ (for some $i \in \{1, \ldots, k_{\underline{x}}\}$). The terms $t_1, \ldots, t_{k_{\underline{x}}}$ are called $\Sigma(\underline{x})$-*representative terms*; if they are effectively computable from $\underline{x}$ (and t_i is computable from u), then T is said to be *effectively locally finite* (in the following, when we say 'locally finite', we in fact always mean 'effectively locally finite'). If Σ is relational or quasi-relational, then any Σ-theory T is locally finite. An *enumerated data-type theory* T is a theory in a quasi-relational signature whose class of models contains only a single finite Σ-structure $\mathcal{M} = (M, \mathcal{I})$ such that for every $m \in M$ there exists a constant $c \in \Sigma$ such that $c^{\mathcal{I}} = m$.

A T-*partition* is a finite set $C_1(\underline{x}), \ldots, C_n(\underline{x})$ of quantifier-free formulae 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 $t_1(\underline{x}), \ldots, t_n(\underline{x})$; (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 Σ-reduct is a model of T and such that $\mathcal{M} \models \bigwedge_{i=1}^{n} \forall \underline{x}\ (C_i(\underline{x}) \rightarrow F(\underline{x}) = t_i(\underline{x}))$. 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 translate any $SMT(T')$ problem into an equivalent $SMT(T)$-problem, by repeatedly applying the following transformation: given the quantifier free formula ϕ to be tested for T'-satisfiability, replace it by $\bigvee_i (C_i \sigma \wedge \phi_i)$, where ϕ_i is a formula obtained from ϕ by replacing each term of the kind $F\sigma$ by $t_i\sigma$ (the C_i's are the partition formulae for the case definition of F and the t_i's are the related 'value' terms).

From now on, we use many-sorted first-order logic. All notions introduced above can be easily adapted to a many-sorted framework. **In the rest of the paper, we fix** (i) a theory $T_I = (\Sigma_I, \mathcal{C}_I)$ for indexes whose only sort symbol is `INDEX`; (ii) a theory $T_E = (\Sigma_E, \mathcal{C}_E)$ for data whose only sort symbol is `ELEM` (the class $\mathcal{C}_E$ of models of this theory is usually a singleton). The **theory** $A_I^E = (\Sigma, \mathcal{C})$ **of arrays with indexes** I **and elements** E is obtained as the combination of T_I and T_E as follows: `INDEX`, `ELEM`, and `ARRAY` are the only sort symbols of A_I^E, the signature is $\Sigma := \Sigma_I \cup \Sigma_E \cup \{\_[\_]\}$ where $\_[\_] : \texttt{ARRAY}, \texttt{INDEX} \longrightarrow \texttt{ELEM}$ (intuitively, $a[i]$ denotes the element stored in the array a at index i); a three-sorted structure $\mathcal{M} = (\texttt{INDEX}^{\mathcal{M}}, \texttt{ELEM}^{\mathcal{M}}, \texttt{ARRAY}^{\mathcal{M}}, \mathcal{I})$ is in $\mathcal{C}$ iff $\texttt{ARRAY}^{\mathcal{M}}$ is the set of (total) functions from $\texttt{INDEX}^{\mathcal{M}}$ to $\texttt{ELEM}^{\mathcal{M}}$, the function symbol $\_[\_]$ is interpreted as function application, and $\mathcal{M}_I = (\texttt{INDEX}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_I})$, $\mathcal{M}_E = (\texttt{ELEM}^{\mathcal{M}}, \mathcal{I}_{|\Sigma_E})$ are models of T_I and T_E, respectively (where $\mathcal{I}_{|\Sigma_X}$ is the restriction of the interpretation $\mathcal{I}$ to the symbols in Σ_X for $X \in \{I, E\}$).

Notational conventions. For the sake of brevity, we introduce the following notational conventions: d, e range over variables of sort `ELEM`, a over variables of sort `ARRAY`, i, j, k, and z over variables of sort `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 tuple of terms $a[i_1], \ldots, a[i_n]$. Possi-

bly 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 $\phi(\underline{i},\underline{e})$.

3 Backward Reachability and Tableaux

Following [12], we focus on a particular yet large class of array-based systems corresponding to guarded assignments. A *(guarded assignment) array-based (transition) system (for (T_I, T_E))* is a triple $\mathcal{S} = (a, I, \tau)$ where (i) a is the *state* variable of sort `ARRAY`;[3] (ii) $I(a)$ is the *initial* $\Sigma(a)$-formula; and (iii) $\tau(a, a')$ is the *transition* $(\Sigma \cup \Sigma_D)(a, a')$-formula, where a' is a renamed copy of a and Σ_D is a finite set of case-defined function symbols not in $\Sigma_I \cup \Sigma_E$. Below, we also **assume $I(a)$ to be a $\forall^I$-*formula***, i.e. a formula of the form $\forall\underline{i}.\phi(\underline{i}, a[\underline{i}])$, and $\tau(a, a')$ **to be in *functional form***, i.e. a *disjunction of* formulae of the form

$$\exists\underline{i}\,(\phi_L(\underline{i}, a[\underline{i}]) \wedge \forall j\, a'[j] = F_G(\underline{i}, a[\underline{i}], j, a[j])) \tag{1}$$

where ϕ_L is the *guard* (also called the local component in [10]), and F_G is a case-defined function (called the *global* component). To understand why formulae (1) are in functional form, consider λ-abstraction; then, the sub-formula $\forall j\, a'[j] = F_G(\underline{i}, a[\underline{i}], j, a[j]))$ can be re-written as $a' = \lambda j.F_G(\underline{i}, a[\underline{i}], j, a[j])$. (By abuse of notation, any case-definable extension of A_I^E will be denoted by A_I^E).

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

$$I(a_0) \wedge \tau(a_0, a_1) \wedge \cdots \wedge \tau(a_{n-1}, a_n) \wedge U(a_n) \tag{2}$$

is A_I^E-satisfiable. If there is no such n, then $\mathcal{S}$ is *safe* (w.r.t. U); otherwise, it is *unsafe* since the A_I^E-satisfiability of (2) implies the existence of a run (of length n) leading the system from a state in I to a state in U. From now on, we **assume $U(a)$ to be a $\exists^I$-*formula***, i.e. a formula of the form $\exists\underline{i}.\phi(\underline{i}, a[\underline{i}])$.

A general approach to solve instances of the safety problem is based on 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

$$Pre(\tau, K) := \exists a'.(\tau(a, a') \wedge K(a')). \tag{3}$$

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

[3] For simplicity (and without loss of generality), we limit ourselves to array-based systems having just one variable a of sort `ARRAY`. This limitation is however dropped in the examples, where in addition T_E may be many-sorted.

```
function BReach(U : ∃^I-formula)
1  P ⟵ U; B ⟵ ⊥;
2  while (P ∧ ¬B is A_I^E-sat.) do
3      if (I ∧ P is A_I^E-sat.)
           then return unsafe;
4      B ⟵ P ∨ B;
5      P ⟵ Pre(τ, P);
6  end
7  return (safe, B);
```

(a)

```
function SInv(U : ∃^I-formula)
1  P ⟵ ChooseCover(U); B ⟵ ⊥;
2  while (P ∧ ¬B is A_I^E-sat.) do
3      if (I ∧ P is A_I^E-sat.)
           then return failure;
4      B ⟵ P ∨ B;
5      P ⟵ ChooseCover(Pre(τ, P));
6  end
7  return (success, ¬B);
```

(b)

Fig. 1. The basic backward reachability (a) and the invariant synthesis (b) algorithms

basic backward reachability algorithm, depicted in Figure 1 (a), 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. Indeed, for BReach (Figure 1 (a)) to be a true (possibly non-terminating) procedure, it is mandatory that (i) $\exists^I$-formulae are closed under pre-image computation and (ii) both the A_I^E-satisfiability test for safety (line 3) and that for fix-point (line 2) are effective.

Concerning (i), it is sufficient to recall the following result from [12].

Proposition 3.1. *Let* $K(a) := \exists \underline{k}\, \phi(\underline{k}, a[\underline{k}])$ *and* $\tau(a, a') := \bigvee_{h=1}^{m} \exists \underline{i}\, (\phi_L^h(\underline{i}, a[\underline{i}]) \wedge a' = \lambda j.F_G^h(\underline{i}, a[\underline{i}], j, a[j]))$. *Then,* $Pre(\tau, K)$ *is* A_I^E*-equivalent to an (effectively computable)* $\exists^I$*-formula.*

The proof of Proposition 3.1 (see [12]) consists of applying simple logical manipulations to show that $Pre(\tau_h, K)$ is A_I^E-equivalent to the following $\exists^I$-formula, where τ_h is one of the m disjuncts of τ (cf. Proposition 3.1 above):

$$\exists \underline{i}\, \exists \underline{k}.(\phi_L^h(\underline{i}, a[\underline{i}]) \wedge \phi(\underline{k}, F_G^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))) \tag{4}$$

where $\phi(\underline{k}, F_G^h(\underline{i}, a[\underline{i}], \underline{k}, a[\underline{k}]))$ is the formula obtained from $\phi(\underline{k}, a'[\underline{k}])$ by replacing $a'[k_m]$ with $F_G^h(\underline{i}, a[\underline{i}], k_m, a[k_m])$ for $m = 1, ..., l$ and $\underline{k}$ is the tuple $k_1, \ldots, k_l$ (the F_G^h can then be eliminated as shown in Section 2). Notice that the existentially quantified prefix $\exists\ \underline{k}$ is augmented with $\exists\ \underline{i}$ in (4) with respect to K. Concerning (ii), observe that the formulae involved in the satisfiability checks are $I \wedge BR^n(\tau, K)$ and $BR^{n+1}(\tau, K) \wedge \neg BR^n(\tau, K)$. Since we have closure under pre-image computation, both formulae are of the form $\exists \underline{a}\ \exists \underline{i}\ \forall \underline{j}\ \psi(\underline{i}, \underline{j}, \underline{a}[\underline{i}], \underline{a}[\underline{j}])$ and are called $\exists^{A,I}\forall^I$*-sentences* [10].

Theorem 3.2 ([10]). *The A_I^E-satisfiability of $\exists^{A,I}\forall^I$-sentences is decidable if (i) T_I is locally finite and is closed under substructures; (ii) the $SMT(T_I)$ and $SMT(T_E)$ problems are decidable.*

Hypothesis (i) concerns the *topology* of the system (not the data manipulated by the components of the system) and it is satisfied in many practical cases, e.g., when the models of T_I are all finite sets, linear orders, graphs, forests, etc. For example, the topology of virtually any cache coherence protocol can be formalized by finite sets while that of mutual exclusion protocols by linear orders. Under assumption (i), it is possible to show (see [10]) that an $\exists^{A,I}\forall^I$-sentence is A_I^E-satisfiable iff it is satisfiable in a finite index model of A_I^E (a *finite index model* is a model $\mathcal{M}$ in which the set $\texttt{INDEX}^{\mathcal{M}}$ has finite cardinality). This suggests the following quantifier instantiation algorithm, which is indeed complete [10]. Let $\exists \underline{a}\ \exists \underline{i}\ \forall \underline{j}\ \psi(\underline{i}, \underline{j}, \underline{a}[\underline{i}], \underline{a}[\underline{j}])$ be an $\exists^{A,I}\forall^I$-sentence: first, consider the $\underline{i}$'s as Skolem constants and replace the $\underline{j}$'s with the representative $\underline{i}$-terms (by using the local finiteness of T_I); then, invoke the available SMT solver for checking the A_I^E-satisfiability of the resulting quantifier-free formula. The decidability of the $SMT(A_I^E)$ problem can be shown by using *generic combination techniques* from the decidability of those for $SMT(T_I)$ and $SMT(T_E)$ (see [10] for details).

We summarize our working hypotheses.

Assumption 3.3 *We fix an array-based system $\mathcal{S} = (a, I, \tau)$ such that the initial formula I is a $\forall^I$-formula, $\tau(a, a') := \bigvee_{h=1}^{m} \tau_h(a, a')$ where τ_h is a formula in functional form for $h = 1, ..., m$. We also assume that hypotheses (i)-(ii) of Theorem 3.2 are satisfied.*

3.1 Tableaux-like Implementation of Backward Reachability

A naive implementation of the algorithm in Figure 1 (a) does not scale up. The main problem is the size of the formula $BR^n(\tau, U)$ which contains many redundant or unsatisfiable sub-formulae. We now discuss how Tableaux-like techniques can be used to circumvent these difficulties. We need one more definition: an $\exists^I$-formula $\exists i_1 \cdots \exists i_n \phi$ is said to be *primitive* iff ϕ is a conjunction of literals and is said to be *differentiated* iff ϕ contains as a conjunct the negative literal $i_k \neq i_l$ for all $1 \leq k < l \leq n$. By applying various distributive laws together with the rewriting rules

$$\exists j(i = j \wedge \theta) \rightsquigarrow \theta(i/j) \ \text{ and } \ \theta \rightsquigarrow (\theta \wedge i = j) \vee (\theta \wedge i \neq j) \tag{5}$$

it is possible to transform every $\exists^I$-formula into a disjunction of primitive differentiated ones.

We initialize our tableau with the $\exists^I$-formula $U(a)$ representing the set of unsafe states. The key observation is to revisit the computation of the pre-image as the following inference rule (we use square brackets to indicate the applicability condition of the rule):

$$\frac{K \quad [K \text{ is primitive differentiated}]}{Pre(\tau_1, K) \mid \cdots \mid Pre(\tau_m, K)}\ \mathsf{PreImg}$$

where $Pre(\tau_h, K)$ computes the $\exists^I$-formula which is logically equivalent to the pre-image of K w.r.t. τ_h (this is possible according to Proposition 3.1).

Since the $\exists^I$-formulae labeling the consequents of the rule PreImg may not be primitive and differentiated, we need the following Beta rule

$$\frac{K}{K_1 \mid \cdots \mid K_n}\ \mathsf{Beta}$$

where K is first transformed by eliminating the case-defined functions as explained in Section 2, and then by applying rewriting rules like (5) together with standard distributive laws, in order to get $K_1, \ldots, K_n$ which are primitive, differentiated and whose disjunction is A_I^E-equivalent to K.

By repeatedly applying the above rules, it is possible to build a tree whose nodes are labelled by $\exists^I$-formulae describing the set of backward reachable states. Indeed, it is not difficult to see that the disjunction of the $\exists^I$-formulae labelling all the nodes in the (potentially infinite) tree is A_I^E-equivalent to the (infinite) disjunction of the formulae $BR^n(\tau, U)$, where $\tau := \bigvee_{h=1}^m \tau_h$. Indeed, there is no need to fully expand our tree. For example, it is useless to apply the rule PreImg to a node ν labelled by an $\exists^I$-formula which is A_I^E-unsatisfiable as all the formulae labelling nodes in the sub-tree rooted at ν will also be A_I^E-unsatisfiable. This observation can be formalized by the following rule closing a branch in the tree (we mark the terminal node of a closed branch by $\times$):

$$\frac{K \quad [K \text{ is } A_I^E\text{-unsatisfiable}]}{\times}\ \mathsf{NotAppl}$$

This rule is effective since $\exists^I$-formulae are a subset of $\exists^{A,I}\forall^I$-sentences and the A_I^E-satisfiability of these formulae is decidable by Theorem 3.2.

According to procedure BReach, there are two more situations in which we can stop expanding a branch in the tree. One terminates the branch because of the safety test (cf. line 3 of Figure 1 (a)):

$$\frac{K \quad [I \wedge K \text{ is } A_I^E\text{-satisfiable}]}{\mathsf{UnSafe}}\ \mathsf{Safety}$$

Interestingly, if we label with τ_h the edge connecting a node labeled with K with that labeled with $Pre(\tau_h, K)$ when applying rule PreImg, then the transitions $\tau_{h_1}, ..., \tau_{h_e}$ labelling the edges in the branch terminated by UnSafe (from the leaf node to the root node) give a *bad trace*, i.e. a sequence of transitions leading the array-based system from a state satisfying I to one satisfying U. Again, rule UnSafe is effective since $I \wedge K$ is equivalent to an $\exists^{A,I}\forall^I$-sentence and its A_I^E-satisfiability is decidable by Theorem 3.2. The other situation in which one can close a branch corresponds to the fix-point test (cf. line 2 of Figure 1 (a))

$$\frac{K \quad [K \wedge \bigwedge\{\neg K' | K' \preceq K\} \text{ is } A_I^E\text{-unsatisfiable}]}{\times}\ \mathsf{FixPoint}$$

where $K' \preceq K$ means that K' is a primitive differentiated $\exists^I$-formula labeling a node preceding the node labeling K (nodes can be ordered according to the

strategy for expanding the tree). Once more, this rule is effective since $K \wedge \bigwedge\{\neg K' | K' \preceq K\}$ can be straightforwardly transformed into an $\exists^{A,I}\forall^I$-sentence whose A_I^E-satisfiability is decidable by Theorem 3.2.

From the implementation viewpoint, further heuristics are needed, in order to reduce the instances needed for the satisfiability test of Theorem 3.2 and to trivialize the recognition of the unsatisfiable premise of the rule NotAppl.

4 Invariants and Backward Reachability

Termination of our tableaux calculus (and of the algorithm of Figure 1 (a)) is not guaranteed in general, but follows under certain restrictions covering important applications (see below). In the general case, nothing can be said because safety problems are undecidable.

Theorem 4.1. *The problem: "given an $\exists^I$-formula U, decide whether the array-based system $\mathcal{S}$ is safe w.r.t. U" is undecidable (even if T_E is locally finite).*

It is well-known that invariants are useful for pruning the search space of backward reachability procedures and may help either to obtain or to speed up termination.

Definition 4.2 (Safety invariants). *The $\forall^I$-formula $J(a)$ is a* safety invariant *for the safety problem consisting of the array-based system $\mathcal{S} = (a, I, \tau)$ and unsafe $\exists^I$-formula $U(a)$ iff the following conditions hold:*

(i) $A_I^E \models \forall a(I(a) \rightarrow J(a))$,
(ii) $A_I^E \models \forall a \forall a'(J(a) \wedge \tau(a, a') \rightarrow J(a'))$, *and*
(iii) $\exists a.(U(a) \wedge J(a))$ *is A_I^E-unsatisfiable.*

If we are not given the $\exists^I$-formula $U(a)$ and conditions (i)–(ii) hold, then $J(a)$ is an invariant for $\mathcal{S}$.

Checking whether conditions (i), (ii), and (iii) above hold can be reduced, by trivial logical manipulations, to the A_I^E-satisfiability of $\exists^{A,I}\forall^I$-formulae, which is decidable by Theorem 3.2. So, establishing whether a given $\forall^I$-formula $J(a)$ is a safety invariant can be completely automated.

Property 4.3. Let U be an $\exists^I$-formula. If there exists a safety invariant for U, then the array-based system $\mathcal{S} = (a, I, \tau)$ is safe with respect to U.

So, if we are given a suitable safety invariant, Property 4.3 can be used as the basis of the safety invariant method, which turns out to be more powerful than the basic Backward Reachability algorithm of Figure 1 (a):

Property 4.4. Let the procedure BReach in Figure 1(a) terminate on the safety problem consisting of the array-based system $\mathcal{S} = (a, I, \tau)$ and unsafe formula $U(a)$. If BReach returns (safe, B), then $\neg B$ is a safety invariant for U.

The converse of Proposition 4.4 do not hold: there might be a safety invariant even when BReach diverges, as illustrated by the following example.

Example 4.5. Let us consider a simple algorithm for inserting an element $b[0]$ into a sorted array $b[1], \ldots, b[n]$. Let Σ_I consist of one binary relation symbol S and one constant symbol 0 and T_I be the theory whose class of models consists of the substructures of the structure having the naturals as domain, with 0 interpreted in the obvious way, and S interpreted as the graph of the successor function. To simplify the matter, we shall use a two-sorted theory and two array variables. T_E is the two-sorted theory whose class of models consists of the single two-sorted structure given by the Booleans (with the constants $\top, \bot$ interpreted in the obvious way) and the rationals (with the standard ordering $<$). The array variable a is a Boolean flag, whereas the array variable b is the sorted numerical array where $b[0]$ is to be inserted. The initial $\forall^I$-formula is

$$\forall i\,(a[i] = \bot \leftrightarrow i \neq 0) \wedge \forall i_1, i_2\,(S(i_1, i_2) \rightarrow i_1 = 0 \vee b[i_1] \leq b[i_2])$$

saying that the elements in the array b, whose corresponding Boolean flag, is set to false are arranged in increasing order (namely, all except that at position 0). The transition has the following guard and global component:

$$\begin{array}{l} \phi_L(i_1, i_2, a[i_1], a[i_2]) := S(i_1, i_2) \wedge a[i_1] = \top \wedge a[i_2] = \bot \wedge b[i_1] > b[i_2] \\ F_G(i_1, i_2, a[i_1], a[i_2], b[i_1], b[i_2], j) := \texttt{case of } \{ \quad j = i_1 \;:\; \langle \top, b[i_2] \rangle, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad j = i_2 \;:\; \langle \top, b[i_1] \rangle, \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j \neq i_1 \wedge j \neq i_2 \;:\; \langle a[j], b[j] \rangle \;\; \}, \end{array}$$

which swaps two elements in the array b if their order is decreasing and sets the Boolean fields appropriately. The obvious correctness property is that there are no two values in decreasing order in the array b whose corresponding Boolean flags do not allow the transition to fire:

$$\exists i_1, i_2\,(S(i_1, i_2) \wedge \neg(a[i_1] = \top \wedge a[i_2] = \bot) \wedge b[i_1] > b[i_2]). \tag{6}$$

Unfortunately, BReach in Figure 1 (a) applied to (6) diverges. However, it is not difficult to see that a safety invariant for (6) exists and is given by the following formula:

$$\forall i, j.(S(i, j) \rightarrow \neg(a[i] = \bot \wedge a[j] = \top)) \tag{7}$$

saying that two adjacent indexes cannot have their Boolean flags set to $\bot$ and $\top$, respectively.

4.1 Synthesis of Invariants as the Dual of Backward Reachability

The main difficulty to exploit Property 4.3 is to find suitable $\forall^I$-formulae satisfying conditions (i)—(iii) of Definition 4.2. Unfortunately, the set of $\forall^I$-formulae which are candidates to become safety invariants is infinite. Such a search space can be dramatically restricted when T_E is locally finite, although it is still infinite because there is no bound on the length of the universally quantified prefix. To formalize this, we need to summarize some notions about pre-orders and configurations.

A *pre-order* $(P, \leq)$ is a set endowed with a reflexive and transitive relation; an *upset* of such a pre-order is a subset $U \subseteq P$ such that ($p \in U$ and $p \leq q$ imply $q \in U$). An upset U is *finitely generated* iff it is a finite union of cones, where a *cone* is an upset of the form $\uparrow p = \{q \in P \mid p \leq q\}$ for some $p \in P$. Two elements $p, q \in P$ are *incomparable* (*equivalent*) if neither (both) $p \leq q$ nor (and) $q \leq p$. A pre-order $(P, \leq)$ is a *well-quasi-ordering* (wqo) iff every upset of P is finitely generated (this is equivalent to the standard definition, see [10] for a proof).

An A_I^E*-configuration* (or, briefly, a configuration) is a pair $(s, \mathcal{M})$ such that s is an array of a finite index model $\mathcal{M}$ of A_I^E ($\mathcal{M}$ is omitted whenever it is clear from the context). We associate a Σ_I-structure s_I and a Σ_E-structure s_E with an A_I^E-configuration $(s, \mathcal{M})$ as follows: the Σ_I-structure s_I is simply the finite structure $\mathcal{M}_I$, whereas s_E is the smallest Σ_E-substructure of $\mathcal{M}_E$ containing the image of s (in other words, if $\texttt{INDEX}^{\mathcal{M}} = \{c_1, \ldots, c_k\}$, then s_E is the smallest Σ_E-substructure containing $\{s(c_1), \ldots, s(c_k)\}$). Let s, s' be configurations: we say that $s' \leq s$ holds iff there are a Σ_I-embedding $\mu : s'_I \longrightarrow s_I$ and a Σ_E-embedding $\nu : s'_E \longrightarrow s_E$ such that the set-theoretical compositions of μ with s and of s' with ν are equal. In [10], termination of BReach is proved under the hypotheses that T_E is locally finite and the configuration order is a wqo. This implies the decidability of the safety problem for, among others, broadcast protocols and lossy channel systems and can be seen as the declarative counterpart of general results formulated within an algebraic framework (see, e.g., [1]). In the following, we show that using the notions of configuration and configuration order, it is possible to design a method for invariant synthesis.

Finitely generated upsets of configurations and $\exists^I$-formulae can be used interchangeably under a suitable assumption. Let $K(a)$ be an $\exists^I$-formula; we let $[\![K]\!] := \{(s, \mathcal{M}) \mid \mathcal{M} \models K(s)\}$.

Proposition 4.6 (Extended version of [10]). *Let T_E be locally finite. Finitely generated upsets of A_I^E-configurations coincide with sets of A_I^E-configurations of the kind $[\![K]\!]$, for some $\exists^I$-formula K. In particular, for each A_I^E-configuration s, there exists an $\exists^I$-formula K_s such that $[\![K_s]\!] = \uparrow s$.*

The notion of a basis for a configuration upset will be useful in the following.

Definition 4.7. *A* basis *for a finitely generated upset S (resp., for an $\exists^I$-formula K) is a minimal finite set $\{s_1, \ldots, s_n\}$ such that S (resp., $[\![K]\!]$) is equal to $\uparrow s_1 \cup \cdots \cup \uparrow s_n$.*

It is easy to see that two bases for the same upset are essentially the same, in the sense that they are formed by pairwise equivalent configurations. Our goal is to integrate the safety invariant method into the basic Backward Reachability algorithm of Figure 1(a). To this end, we introduce the notion of 'sub-reachability'.

Definition 4.8 (Subreachable configurations). *Suppose T_E is locally finite and let s be a configuration. A* predecessor *of s is any s' that belongs to a basis for $Pre(\tau, K_s)$. Let s, s' be configurations: s is* sub-reachable *from s' iff there exist configurations $s_0, \ldots, s_n$ such that (i) $s_0 = s$, (ii) $s_n = s'$, and (iii) either*

$s_{i-1} \leq s_i$ *or* s_{i-1} *is a predecessor of* s_i*, for each* $i = 1, \ldots, n$*. If* K *is an* $\exists^I$*-formula,* s is sub-reachable from K *iff* s *is sub-reachable from some* s' *taken from a basis of* K*.*

The following theorem is our main technical result.

Theorem 4.9. *Let* T_E *be locally finite. If there exists a safety invariant for* U*, then there are finitely many* A_I^E*-configurations* $s_1, \ldots, s_k$ *which are sub-reachable from* U *and such that* $\neg(K_{s_1} \vee \cdots \vee K_{s_k})$ *is also a safety invariant for* U*.*

The intuition underlying the theorem is as follows. Let us call 'finitely representable' an upset which is of the kind $[\![K]\!]$ for some $\exists^I$-formula K and let B be the set of backward reachable states. Usually B is infinite and it is finitely representable only in special cases (e.g., when the configuration ordering is a wqo like in the case of broad-cast protocols). Despite this, it may sometimes exist a set $B' \supseteq B$ which is finitely representable and whose complement is an invariant of the system. Theorem 4.9 ensures us to find such a B', if any. This is the case of Example 4.5 where not all configurations satisfying the negation of (7) are in B.

In practice, Theorem 4.9 suggests the following procedure to find the superset B'. At each iteration of BReach, the algorithm represents symbolically in the variable B the configurations which are backward reachable in n steps; before computing the next pre-image of B, non deterministically replace some of the configurations in a basis of B with some sub-configurations and update B by a symbolic representation of the obtained upset. In this way, if an invariant exists, we are guaranteed to find it; otherwise, the process may diverge. This is so because the search space of the configurations which are sub-reachable in n steps is finite, although this number is infinite if no bound on n is fixed. To illustrate, the negation of (7) in Example 4.5 identifies sub-reachable only configurations. This shows that sub-reachability is crucial for Theorem 4.9 to hold.

The algorithm sketched above can be furtherly refined so as to obtain a completely symbolic method working with formulae without resorting to configurations. The key idea towards this result is to identify an $\exists^I$-formula which is the symbolic counterpart of the (sub-reachable) configurations $s_1, \ldots, s_k$ of the theorem above which can be directly computed from the available safety invariant for U. Formally, we introduce the following definition:

$$Min(\phi, a, \underline{i}) := \phi(\underline{i}, a[\underline{i}]) \wedge \bigwedge_{\sigma} (\phi(\underline{i}\sigma, a[\underline{i}\sigma]) \rightarrow \bigwedge_{i \in \underline{i}} \bigvee_{t} (t\sigma = i))$$

where $\phi(\underline{i}, a[\underline{i}])$ is a quantifier-free formula, t ranges over representative $\Sigma_I(\underline{i})$-terms, and σ ranges over the substitutions with domain $\underline{i}$ and co-domain included in the set of representative $\Sigma_I(\underline{i})$-terms. The formula $\exists \underline{i}.Min(\phi, a, \underline{i})$ is A_I^E-equisatisfiable to the $\exists^I$-formula $\exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$; moreover if (as it often happens in applications) the signature Σ_I is relational and the formula $\phi(\underline{i}, a[\underline{i}])$ is differentiated, $Min(\phi, a, \underline{i})$ is A_I^E-equivalent to $\phi(\underline{i}, a[\underline{i}])$.

Proposition 4.10. *Let T_E be locally finite, $K := \exists \underline{i}.\phi(\underline{i}, a[\underline{i}])$ be an $\exists^I$-formula, and L be a further $\exists^I$-formula. The following two conditions are equivalent:*

(i) *for every s in a basis for K, there exists a configuration s' in a basis for L such that $s \leq s'$;*
(ii) *L is (up to A_I^E-equivalence) of the form $\exists \underline{i}, \underline{j}.\psi(\underline{i}, \underline{j}, a[\underline{i}], a[\underline{j}])$ for a quantifier-free formula ψ and*

$$\text{if } A_I^E \models Min(\psi, a, \underline{i}\,\underline{j}) \rightarrow \theta(\underline{t}, a[\underline{t}]) \text{ then } A_I^E \models Min(\phi, a, \underline{i}) \rightarrow \theta(\underline{t}, a[\underline{t}]),$$

for all quantifier free $(\Sigma_E \cup \Sigma_I)$-formula θ and for all tuple of terms $\underline{t}(\underline{i})$ taken from the set of the representative $\Sigma_I(\underline{i})$-terms.

In the following, we will write $K \leq L$ whenever one of the (equivalent) conditions in Proposition 4.10 holds. Under the assumption that T_E is locally finite, it is possible to compute all the finitely many (up to A_I^E-equivalence) $\exists^I$-formulae K such that $K \leq L$. Furthermore, we say that K *covers* L iff both $K \leq L$ and $A_I^E \models L \rightarrow K$. Let $\mathsf{ChooseCover}(L)$ be a procedure that returns (according to some criteria) one of the $\exists^I$-formulae K such that K covers L. We are now ready to give the procedure SInv in Figure 1 (b) for the computation of safety invariants and prove its correctness.

Theorem 4.11. *Let T_E be locally finite. Then, there exists a safety invariant for U iff the procedure* SInv *in Figure 1 (b) returns a safety invariant for U, for a suitable* ChooseCover *function.*

When $\mathsf{ChooseCover}(L) = L$, i.e. ChooseCover is the identity (indeed, L covers L), the procedure SInv is the (exact) dual of BReach in Figure 1 (a) and, hence it can only return (the negation of) a symbolic representation of all backward reachable states as a safety invariant.

4.2 Integrating Invariant Synthesis within Backward Reachability

The main drawback of procedure SInv is the difficulty of defining an appropriate function ChooseCover. Although finite, the number of formulae covering a certain $\exists^I$-formula is so large that makes any implementation of SInv impractical. Instead, we prefer to study how to integrate the synthesis of invariants in the backward reachability algorithm in Figure 1 (a). The idea is to use invariants for the unsafe configuration U so as to prune the search space of the backward reachability algorithm. In our symbolic framework, at the n-th iteration of the loop of the procedure BReach, the set of backward reachable states is represented by the formula stored in the variable B (which is equivalent to $BR^n(\tau, U)$). So, 'pruning the search space of the backward reachability algorithm' amounts to disjoining the negation of the available invariants to B. In this way, the extra information encoded in the invariants makes the satisfiability test at line 2 (for fix-point checking) more likely to be successful and possibly decreasing the number of iterations of the loop.

Indeed, the problem is to synthesize such invariants. A way to do this is to consider the set B of reachable states, to extract an $\exists^I$-formula representing a set of sub-reachable configurations, and then checking whether this is an invariant. We assume the existence of a function ChooseSub that takes an $\exists^I$-formula P and returns a (possibly empty) finite set S of $\exists^I$-formulae such that $K \leq P$ if $K \in S$. The formulae in S represent sub-reachable configurations that may contribute to an invariant in the sense of Theorem 4.9.

To summarize, it is possible to integrate the synthesis of invariants within the backward reachability algorithm by inserting between lines 4 and 5 in Figure 1 (a) the following instructions:

$4'$ **foreach** $CINV \in \mathsf{ChooseSub}(P)$ **do**
if $\mathsf{BReach}(CINV) = (\mathsf{safe}, B_{CINV})$ **then** $B \longleftarrow B \vee \neg B_{CINV}$;

where *CINV* stands for 'candidate invariant.' The resulting procedure will be indicated with BReach+Inv (notice that BReach is used here as a sub-procedure).

Proposition 4.12. *Let T_E be locally finite. If the procedure* BReach+Inv *terminates and returns safe (unsafe), then $\mathcal{S}$ is safe (unsafe) with respect to U.*

The procedure BReach+Inv is incomplete (in the sense that it is not guaranteed to terminate even in case a safety invariant exists), deterministic (no backtracking is required), and highly parallelizable (it is possible to run in parallel as many instances of BReach as formulae in the set returned by ChooseSub), and it performs well, as witnessed by the experimental evidence supplied in the next Section. In this way, invariant synthesis has become a powerful *heuristics* within a sophisticated version of the basic backward reachability algorithm. Furthermore, its integration in the Tableaux calculus of Sec. 3.1 is particularly easy: just use the calculus itself with some bounds on the resources, such as a limit on the depth of the tree to check if a candidate invariant is a true invariant.

4.3 Heuristics

There is a delicate trade-off between the number of candidate invariants produced by the function ChooseSub and their effects in pruning the search space of the basic backward reachability algorithm. More candidate invariants implies a higher probability of finding an invariant and, ultimately, to prune the search space. However, looking at line $4'$, it is evident that more candidate invariants implies many more calls to the basic backward reachability algorithms to establish if they are invariant or not. Indeed, on "simpler" candidate invariants, the procedure BReach is likely to perform well, i.e. to terminate in few iterations. The following two remarks are helpful in finding the right trade-off.

First, it is possible to limit the resources of the basic backward reachability algorithm BReach when invoking it at line $4'$; e.g., it is possible to bound the number of iterations of the loop or its run time. This allows us to avoid slowing down too much each iteration of the main loop in BReach+Inv.

The second remark concerns the implementation of the function ChooseSub when the theories T_I and T_E satisfy some additional requirements, which are

often satisfied when modelling classes of parametrised systems such as mutual exclusion algorithms or cache coherence protocols. The goal of this discussion is to design a function ChooseSub returning few "simple" candidate invariants which are likely to become true invariants.

Claim. Let Σ_I be relational and let T_E be locally finite and admit elimination of quantifiers. (When T_I is the theory of all finite sets—this is appropriate for cache coherence protocols—or the theory of linear orders—this is appropriate for mutual exclusion algorithms—and T_E is the theory of an enumerated datatype, these assumptions are satisfied.) Let

$$L := \exists \underline{i}\, \underline{j}.(\psi_E(a[\underline{i}], a[\underline{j}]) \wedge \psi_I(\underline{i}, \underline{j}) \wedge \delta_I(\underline{i})) \tag{8}$$

be a primitive differentiated A_I^E-satisfiable $\exists^I$-formula such that (i) $\underline{i} \cap \underline{j} = \emptyset$, (ii) $\psi_E(\underline{e}, \underline{d})$ is a conjunction of Σ_E-literals; (iii) $\psi_I(\underline{i}, \underline{j})$ is a conjunction of Σ_I-literals; (iv) $\delta_I(\underline{i})$ is a maximal conjunction of $\Sigma_I(\underline{i})$-literals (i.e. for every $\Sigma(\underline{i})$-atom $A(\underline{i})$, δ_I contains either $A(\underline{i})$ or its negation). If

$$K := \exists \underline{i}\, (\delta_I(\underline{i}) \wedge \phi_E(a[\underline{i}])), \tag{9}$$

where $\phi_E(\underline{e})$ is T_E-equivalent to $\exists \underline{d}\, \psi_E(\underline{e}, \underline{d})$ (which is guaranteed to exist as T_E admits elimination of quantifiers), then K covers L and in particular $K \leq L$.

When ChooseSub is applied to a disjunction of primitive differentiated $\exists^I$-formulae, we need to transform each disjunct $P := \exists \underline{k}.\theta(\underline{k}, a[\underline{k}])$ to the form of (8) so as to obtain a candidate invariant. To do this, we can decompose $\underline{k}$ into two disjoint sub-sequences $\underline{i}$ and $\underline{j}$ such that $\underline{k} = \underline{i} \cup \underline{j}$ according to some criteria: if the conjunction of $\Sigma_I(\underline{i})$ literals occurring in θ is maximal, we get a candidate invariant by returning the corresponding $\exists^I$-formula (9). This is quite feasible in many concrete cases. For instance, quantifier elimination reduces to a trivial substitution if T_E is an enumerated datatype theory and the Σ_E-literals in θ are all positive. Maximality of θ is guaranteed (by differentiatedness) if T_I is the theory of finite sets; maximality of θ is also guaranteed if T_I is the theory of linear orders and $\underline{i} = i_1$ or ($\underline{i} = i_1, i_2$ and θ contains the atom $i_1 < i_2$).

5 Experiments and Discussion

To test the practical viability of our approach, we have implemented MCMT, a prototype tool which uses Yices (`http://yices.csl.sri.com`) as the backhand SMT solver. MCMT is the successor of the system in [12] which is not capable of solving almost any of the problems considered here. The starting point of our implementation is the Tableaux-like calculus of Section 3.1. As Yices is guaranteed to behave as a decision procedure on quantifier-free formulae only, universally quantified variables in $\exists^{A,I}\forall^I$-sentences are instantiated according to the procedure sketched after Theorem 3.2: this is required for the application of rules NotAppl, Safety, FixPoint. Invariants have been integrated in the basic backward reachability algorithm along the lines of Section 4.2.

	depth	#nodes	#calls	time	depth	#nodes	#calls	#inv	time
Bakery	9	29	221	0.104	7	8	129	5	**0.052**
Burns	14	57	497	0.216	2	2	59	3	**0.016**
Java M-lock	9	23	353	**0.156**	9	22	2390	1	0.772
Dijkstra	13	40	392	0.148	2	1	41	2	**0.012**
Dijkstra (rv)	14	138	6905	5.756	2	1	57	2	**0.016**
Szymanski	17	143	3266	2.208	11	22	1185	8	**0.288**
Szymanski (a)	23	2358	902017	24m19s	16	90	8547	16	**5.188**

Table 1. Mutual exclusion algorithms

	depth	#nodes	#calls	time	depth	#nodes	#calls	#inv	time
Berkeley	2	1	102	**0.020**	2	1	190	0	0.032
Mesi	3	2	175	**0.032**	3	2	231	0	0.036
Moesi	3	2	304	**0.048**	3	2	384	0	0.052
Dec Firefly	4	4	163	**0.052**	4	4	222	0	0.068
Xerox P.D.	7	13	607	**0.288**	7	13	1059	0	0.432
Illinois	4	8	998	**0.196**	4	8	1114	0	0.216
Futurebus	4	19	1318	**0.460**	4	19	3824	0	1.096
German	26	2985	322335	**8m39s**	26	2856	544429	10	10m37s
German (pfs)	42	26004	3062165	176m51s	42	22808	2656282	40	**173m42s**

Table 2. Cache coherence protocols

As benchmarks, we have derived safety problems in our format from two sets of benchmarks in [2]: one is of mutual exclusion protocols (with 7 problems, cf. Table 1) and the other is of cache coherence protocols (with 9 problems, cf. Table 2).[4] We used the theory of finite linear orders as T_I for mutual exclusion algorithms and the theory of finite sets as T_I for cache coherence protocols. The theory T_E for the various systems is the combination of an enumerated datatype theory for the control locations with theories for the data manipulated by the processes. A difficulty in the translation was the presence of global (i.e. universally quantified) guards which are not directly supported by our formalism. It is possible to eliminate universal quantifiers in guards (see [12] for details) by adopting the well-known *stopping failure model* (see, e.g., [14]) which is quite close to the approximate model in [2, 3]. This is without loss of generality since establishing a safety property for the stopping failures model of a system trivially implies that the same property is enjoyed by the original system. The elimination of global guards can be easily mechanized as it is purely syntactic.

Columns 2-5 of both Tables report the statistics of our implementation of the procedure BReach while columns 6-10 show the results for BReach+Inv. (All timings are in seconds and obtained on a Pentium Dual-Core 3.4 GHz with 2 Gb Sdram). Table 1 clearly shows the usefulness of invariant search as the size

[4] The files containing such specifications and an executable of the tool are available at `http://homes.dsi.unimi.it/~ghilardi/mcmt`.

of the problem grows. Table 2 seems to suggest that invariant search is useless or even detrimental to performances on cache coherence protocols. However, we remark that all these problems, except the German, are quite small and a brute force search of the tiny search space (see the column '#nodes') is likely to be more successful. Furthermore, the overhead of searching for invariants can be eliminated by implementing a parallel version of the tool. Interestingly, there is some gain in using invariant synthesis on the last problem in this set (a difficult version of the German protocol [15], which is well-known to be a significant benchmark for verification tools). Although a comparative analysis is somewhat difficult in lack of a standard for the specifications of safety problems, we report that MCMT performs comparably with the model checker PFS [2] on small to medium sized problems and outperforms the latter on larger instances.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proc. of LICS*, pages 313–321, 1996.
2. P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine. Regular model checking without transducers. In *TACAS*, volume 4424 of *LNCS*, pages 721–736, 2007.
3. P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *CAV*, volume 4590 of *LNCS*, pages 145–157, 2007.
4. D. Beyer, T. A. Henzinger, R. Majumdar, and A. Rybalchenko. Invariant Synthesis for Combined Theories. In *VMCAI'07*, volume 4349 of *LNCS*, 2007.
5. Aaron R. Bradley and Zohar Manna. Property-Directed Incremental Invariant Generation. *Formal Aspects of Computing*, 2009. To appear.
6. G. Delzanno, J. Esparza, and A. Podelski. Constraint-based analysis of broadcast protocols. In *Proc. of CSL*, volume 1683 of *LNCS*, pages 50–66, 1999.
7. Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York-London, 1972.
8. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *Proc. of LICS*, pages 352–359. IEEE Computer Society, 1999.
9. C. Flanagan and S. Qadeer. Predicate abstraction for software verification. In *Proc. of POPL'02*, pages 191–202. ACM, 2002.
10. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Towards SMT Model-Checking of Array-based Systems. In *Proc. of IJCAR*, LNCS, 2008.
11. S. Ghilardi and S. Ranise. Goal-directed Invariant Synthesis for Model Checking Modulo Theories. Technical Report RI325-09, Univ. di Milano, 2009.
12. S. Ghilardi, S. Ranise, and T. Valsecchi. Light-Weight SMT-based Model-Checking. In *Proc. of AVOCS 07-08*, ENTCS, 2008.
13. S. K. Lahiri and R. E. Bryant. Predicate Abstraction with Indexed Predicate. *ACM Trans. on Comp. Logic*, 9(1), 2007.
14. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
15. A. Pnueli, S. Ruath, and L. D. Zuck. Automatic deductive verification with invisible invariants. In *Proc. of TACAS 2001*, volume 2031 of *LNCS*, 2001.
16. 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.