Verification of SMT Systems with Quantifiers *

Alessandro Cimatti[0000−0002−1315−6990], Alberto Griggio[0000−0002−3311−0893], and Gianluca Redondi[0000−0002−2856−5236]

Fondazione Bruno Kessler
{cimatti, griggio, gredondi}@fbk.eu

Abstract. We consider the problem of invariant checking for transition systems using SMT and quantified variables ranging over finite but unbounded domains. We propose a general approach, obtained by combining two ingredients: exploration of a finite instance, to obtain candidate inductive invariants, and instantiation-based techniques to discharge quantified queries. A thorough experimental evaluation on a wide range of benchmarks demonstrates the generality and effectiveness of our approach. Our algorithm is the first capable of approaching in a uniform way such a large variety of models.

1 Introduction

Model checking algorithms based on efficient quantifier-free SAT and SMT reasoning have seen significant progress in the last few years. However, in many verification areas first-order quantifiers are needed, both in the symbolic description of the system and in the property to prove. This is the case, for example, of verification of parameterized systems.

Unfortunately, dealing with the combined case of transition systems with theories and first-order is far from trivial: SMT-based model checking algorithms can't be naturally extended. In this paper, we discuss the problem of model checking invariant properties in systems containing SMT theories and first-order quantifiers, with quantified variables ranging over finite but unbounded domains. For example, the (finite) size of the domain may depend on the number of processes in a protocol, or the number of components of the station in a railway interlocking system.

We present a simple yet general approach based on the interaction of two key ingredients. First, given a fixed cardinality for the domain, we compute a quantifier-free system (a *ground instance*) that can be model checked with existing techniques. We either get a counterexample, in which case the system is unsafe, or a proof for the property. Such a proof is lifted to a candidate invariant for the quantified system. This step is crucial, and is made effective by combining minimization and generalization techniques [15,17,26]. Second, we check the validity of the candidate invariant using quantified SMT reasoning. If the

* This work has been partly supported by project "AI@TN" funded by the Autonomous Province of Trento.

candidate invariant is valid, then the system is safe. Otherwise, further reasoning is required, e.g. by increasing the cardinality of the domain, and iterating the first step. Such a check can in principle be carried out by any off-the-shelf solver supporting SMT and quantifiers (e.g. Z3 [23]). However, a black-box approach to checking the validity of quantified invariants may cause the procedure to diverge in practice. Therefore, we adopt a more careful, resource-bounded approach to instantiation, that can be used to discharge quantified queries in a more controlled way.

The approach combines in a unique framework different aspects of the recent literature that have never been integrated. Compared to our approach, previous works on verification of parameterized systems with SMT [8, 14] impose strong syntactic restrictions on the formulae used for defining systems, and allow only a very limited form of quantifier alternation. Other approaches based on modern SAT-based model checking algorithms such as [15, 18, 25] are more liberal, but they do not support theories.

The algorithm has been implemented and experimentally evaluated on various families of benchmarks, obtained from different sources, and making use of theories, quantifier alternations, or both. The experimental evaluation demonstrates that the algorithm is very general, being the only one able deal with all the benchmarks. As far as we know, our algorithm is the first capable of approaching in a uniform way such a large variety of systems. Furthermore, the experimental evaluation shows that, despite the relative simplicity of the implementation, the algorithm is quite efficient, and very effective, solving more instances than the competitor approaches in all the benchmarks classes.

2 Preliminaries

Our setting is standard first order logic. A theory $\mathcal{T}$ in the SMT sense is a pair $\mathcal{T} = (\Sigma, \mathcal{C})$, where Σ is a first-order signature and $\mathcal{C}$ is a class of models over Σ. A theory $\mathcal{T}$ is closed under substructure if its class $\mathcal{C}$ of structures is such that whenever $\mathcal{M} \in \mathcal{C}$ and $\mathcal{N}$ is a substructure of $\mathcal{M}$, then $\mathcal{N} \in \mathcal{C}$. We use the standard notions of Tarskian interpretation (assignment, model, satisfiability, validity, logical consequence). We refer to 0-arity predicates as Boolean variables, and to 0-arity uninterpreted functions as (theory) variables. A literal is an atom or its negation. A clause is a disjunction of literals. A ground term is a term which does not contain free variables. A formula is in conjunctive normal form (CNF) iff it is a conjuction of clauses. If $x_1, ..., x_n$ are variables and ϕ is a formula, we might write $\phi(x_1, ..., x_n)$ to indicate that all the variables occurring free in ϕ are in $x_1, ..., x_n$.

If ϕ is a formula, t is a term and v is a variable which occurs free in ϕ, we write $\phi[v/t]$ for the substitution of every occurrence of v with t. If $\underline{t}$ and $\underline{v}$ are vectors of the same length, we write $\phi[\underline{v}/\underline{t}]$ for the simultaneous substitution of each v_i with the corresponding term t_i.

Given a set of variables $\underline{v}$, we denote with $\underline{v}'$ the set $\{v' \mid v \in \underline{v}\}$. A symbolic transition system is a triple $(\underline{v}, I(\underline{v}), T(\underline{v}, \underline{v}'))$, where $\underline{v}$ is a set of variables, and

$I(\underline{v})$, $T(\underline{v}, \underline{v}')$ are first-order formulae over some signature. An assignment to the variables in $\underline{v}$ is a state. A state s is initial iff it is a model of $I(\underline{v})$, i.e. $s \models I(\underline{v})$. The states s, s' denote a transition iff $s \cup s' \models T(\underline{v}, \underline{v}')$, also written $T(s, s')$. A path is a sequence of states $s_0, s_1, \ldots$ such that s_0 is initial and $T(s_i, s'_{i+1})$ for all i. We denote paths with π, and with $\pi[j]$ the j-th element of π. A state s is reachable iff there exists a path π such that $\pi[i] = s$ for some i.

A formula $\phi(\underline{v})$ is an invariant of the transition system $C = (\underline{v}, I(\underline{v}), T(\underline{v}, \underline{v}'))$ iff it holds in all the reachable states. Following the standard model checking notation, we denote this with $C \models \phi(\underline{v})$.[1] A formula $\phi(\underline{v})$ is an inductive invariant for C iff $I(\underline{v}) \models \phi(\underline{v})$ and $\phi(\underline{v}) \wedge T(\underline{v}, \underline{v}') \models \phi(\underline{v}')$. Given a first-order formula ϕ over a signature Σ, containing arbitrary quantifiers, it is well known that it is possible to obtain a universal formula ϕ', called the *Skolemization* of ϕ, defined over a larger signature Σ', which is equisatisfiable to ϕ.

3 Verification of Quantified SMT Systems

3.1 Symbolic Formalism

The problem discussed in this paper is to prove or disprove that a given quantified formula is an invariant of a symbolic transition system. In this section, we describe the formalism that we use for defining systems and we present an overall picture of the algorithm we use to solve the problem.

We introduce a class of symbolic transition systems, which subsumes many formalisms presented in the literature [14,24]. We start by considering two theories; a theory $\mathcal{T}_\mathcal{I} = (\Sigma_I, \mathcal{C}_I)$, called the *index* theory, which is closed under substructures. In practice, this is often the theory of an uninterpreted sort, whose class of models includes all possible finite (but unbounded) structures. In addition, we consider a theory of elements $\mathcal{T}_E = (\Sigma_E, \mathcal{C}_E)$, used to model the data of the system. Relevant examples consider as $\mathcal{T}_E$ the theory of an enumerated datatype, or linear arithmetic (integer or real). Then, with A^E_I we denote the theory whose signature is $\Sigma = \Sigma_I \cup \Sigma_E \cup \{[\_]\}$, and a model for it is given by a set of total functions from a model of $\mathcal{T}_I$ to a model of $\mathcal{T}_E$, where $[\_]$ is interpreted as the function application. In the following, we might refer to variables of sort A^E_I as *arrays*.

We restrict ourselves to one index theory and one element theory for the sake of simplicity, but typically applications include a multi-sorted setting, with several index theories and several element theories.

Definition 1 *In the following, we will consided a subclass of transition systems, defined by triples* $S = (\underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x}'))$ *where:*

[1] Note that we use the symbol $\models$ with three different denotations: if ϕ, ψ are formulae, $\phi \models \psi$ denotes that ψ is a logical consequence of ϕ; if μ is an interpretation, and ψ is a formula, $\mu \models \psi$ denotes that μ is a model of ψ; if C is a transition system, $C \models \psi$ denotes that ψ is an invariant of C.

- $\underline{x}$ *are arrays, i.e. variables of sort* A^E_I *interpreted as functions from a model of* $\mathcal{T}_\mathcal{I}$ *to a model of* $\mathcal{T}_E$*. Note that this includes also 0-ary or constant functions, i.e. variables of sort* $\mathcal{T}_E$*.*
- $\iota(\underline{x}), \tau(\underline{x}, \underline{x}')$ *are first-order formulae over* Σ *possibly containing quantifiers over variables of sorts* $\mathcal{T}_I$*.*

Example 1 (A simple train station). In this example, we describe an abstract simple train station, with an arbitrary number of tracks and routes; routes can be activated by locking the corresponding tracks. As index theories, we use two uninterpeted sorts: track and route. As element theory, we use two enumeratives, T_{E_1} with model $\{\mathsf{locked}, \mathsf{free}\}$, and T_{E_2} with model $\{\mathsf{active}, \mathsf{inactive}\}$. As state variables, we use an array $state\_t : \mathsf{track} \to \{\mathsf{locked}, \mathsf{free}\}$, and an array $state\_r : \mathsf{route} \to \{\mathsf{active}, \mathsf{inactive}\}$. Moreover, we define a relation symbol $\mathsf{UsedBy} : \mathsf{track} \times \mathsf{route} \to \mathit{Bool}$ to model the correspondences between tracks and routes. The initial formula of our model is:

$$\forall r : \mathsf{route}.state\_r[r] = \mathsf{inactive} \ \wedge \ \forall t : \mathsf{track}.state\_t[t] = \mathsf{free}.$$

The transition formula of the system is the disjunction of two formulae $\tau_1 \vee \tau_2$, corresponding to the activation or the deactivation of a route. The first disjunct is:

$$\begin{aligned}\exists r : \mathsf{route}.\big(&state\_r[r] = \mathsf{inactive} \wedge \forall t : \mathsf{track}.(\mathsf{UsedBy}(t, r) \to state\_t[t] = \mathsf{free}) \\ &\wedge state\_r'[r] = \mathsf{active} \wedge \forall r1 : \mathsf{route}.(r1 \neq r \to state\_r'[r1] = state\_r[r1]) \\ &\wedge \forall t1 : \mathsf{track}.(\mathsf{UsedBy}(t1, r) \to state\_t'[t1] = \mathsf{locked}) \\ &\wedge \forall t1 : \mathsf{track}.(\neg\mathsf{UsedBy}(t1, r) \to state\_t'[t1] = state\_t[t1])\big).\end{aligned}$$

The second disjunct is:

$$\begin{aligned}\exists r : \mathsf{route}.\big(&state\_r[r] = \mathsf{active} \wedge state\_r'[r] = \mathsf{inactive} \\ &\wedge \forall r1 : \mathsf{route}.(r1 \neq r \to state\_r'[r1] = state\_r[r1]) \\ &\wedge \forall t1 : \mathsf{track}.(\mathsf{UsedBy}(t1, r) \to state\_t'[t1] = \mathsf{free}) \\ &\wedge \forall t1 : \mathsf{track}.(\neg\mathsf{UsedBy}(t1, r) \to state\_t'[t1] = state\_t[t1])\big).\end{aligned}$$

The *Invariant Problem* we consider is the problem of proving (or disproving) that a given formula ϕ, possibly containing quantified variables of sort $\mathcal{T}_I$, is an invariant for S. The problem is well-known to be undecidable, since it subsumes undecidable problems such as safety of parameterized systems [2].

Example 2. In the example before, we want to prove mutual exclusion of routes which are using a same track. To do this, we define a new relational symbol $\mathsf{Incompatible} : \mathsf{route} \times \mathsf{route} \to \mathit{Bool}$ and we introduce the following axiom:

$$\begin{aligned}\forall r1 : \mathsf{route}, r2 : \mathsf{route}.\big(&\mathsf{Incompatible}(r1, r2) \leftrightarrow \\ &(r1 \neq r2 \wedge \exists t : \mathsf{track}.\mathsf{UsedBy}(t, r1) \wedge \mathsf{UsedBy}(t, r2))\big)\end{aligned}$$

Axioms are not defined in Definition 1, but they are common in the literature regarding symbolic transition systems. An axiom is a formula which is implicitly considered in conjunction to both the initial and the transition formula. The invariant we want to prove is the formula

$$\forall r1 : \mathsf{route}, r2 : \mathsf{route}.\big(\mathsf{Incompatible}(r1, r2) \rightarrow \\ \neg(state\_r[r1] = \mathsf{active} \wedge state\_r[r2] = \mathsf{active})\big)$$

i.e. incompatible routes are never active together. □

To solve the invariant problem affirmatively, we search for an inductive strengthening, i.e. a first-order formula ψ such that $\psi \wedge \phi$ is an inductive invariant for S.

Definition 2 *Let $S = (\underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x'}))$ a transition system, and ϕ a candidate invariant. An invariant strengthening ψ is a first-order fomula such that the following formulae are A_I^E-unsatisfiable:*

$$\iota(\underline{x}) \wedge \neg(\phi(\underline{x}) \wedge \psi(\underline{x})), \tau(\underline{x}, \underline{x'}) \wedge \phi(\underline{x}) \wedge \psi(\underline{x}) \wedge \neg(\phi(\underline{x'}) \wedge \psi(\underline{x'})). \quad (1)$$

Since a formula is valid iff its negation is unsatisfiable, it follows from the definition that $\psi \wedge \phi$ is an inductive invariant for S.

Our method will first automatically synthesize a *candidate* invariant strengthening ψ, and then try to discharge inductive queries with instantiation-based methods to see if the guess was correct. In fact, after Skolemizing inductive queries (1) to a universal form, our method will search for a set of ground terms G such that the ground formula obtained by instantiating universal quantifiers (in G) is unsatisfiable. We have, however, many open problems, which we can summarize with the following questions: (i) How to find such candidate invariant strengthenings? (ii) How to choose the set of ground terms G? (iii) If the query is SAT, how to detect real counterexamples?

In the method we propose, we will try to address these problems with a common approach, which is *ground* instance exploration. A ground instance of the system is obtained by fixing the cardinality of models of $\mathcal{T}_I$ to a fixed integer. In this way we can obtain (after removing quantifiers by instantiation) a transition system defined by quantifier-free formulae, which can be analyzed by standard SMT-based techniques.

We will describe our approach more thoroughly in the next sections. Here, we give a high-level overview of our method, depicted also in Fig. 1.

3.2 Overview

As an input, we have a symbolic transition system S and a candidate invariant ϕ. We set n, a counter for the size of the ground instance we explore, equal to 1. We perform the following steps:

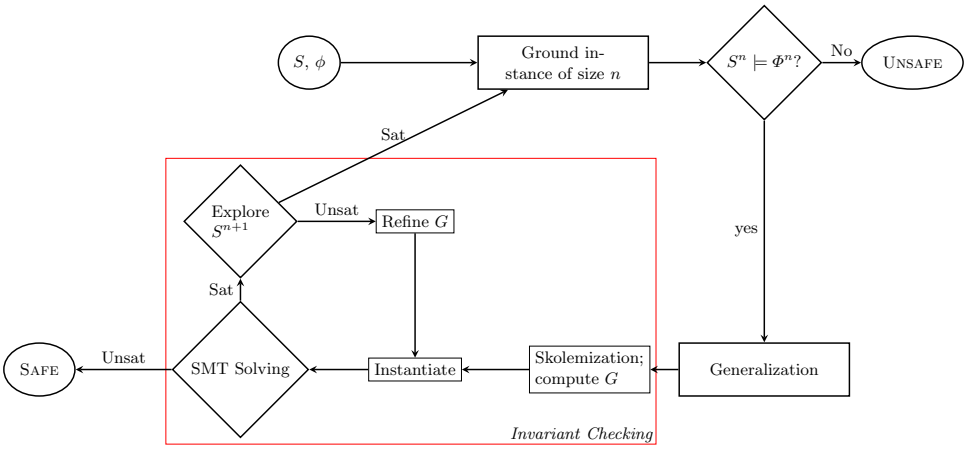

Fig. 1. An overview of the algorithm.

- we consider a ground instance of cardinality n, and then use a model checker to get either a counterexample for the property (thus terminating the algorithm with UNSAFE result), or an inductive invariant in size n. More details about the computation of ground instances are given in §3.3.
- From the invariant of size n, we synthesize a candidate invariant strenghtening ψ (Generalization).
- We consider the quantified queries (1), and try to prove their unsatisfiability (Invariant Checking box). In case of a success, the property is proved and we have found an inductive invariant. In case of a failure, we need a better candidate invariant: we restart the loop with a new exploration from size $n + 1$.

Since we are dealing with undecidable problems, there are many possible causes of non-termination of the algorithm: the main problems are the invariant cheking box, which involves quantified reasoning, and the existence of a cut-off, i.e. an integer n such that the generalized formula obtained afted model checking a ground instance of size n is inductive also for all other instances. Note that the procedure of invariant checking could be be implemented with the usage of any prover supporting SMT reasoning and quantifiers. However, especially for satisfiabile instances, such solvers can diverge easily. Thus, since many queries can be SAT, a naive usage of such tools will cause the procedure to get stuck in quantified reasoning with no progress obtained.

Therefore, we proposed a 'bounded' sub-procedure of Invariant Checking, explained in detail in §3.5, in which instead of relying on an off-the-shelf SMT solver supporting quantifiers, we 'manually' apply standard instantiation-based techniques for quantified SMT reasoning [10], in which however we carefully manage the set of terms used to instantiate the quantifiers, in order to prevent divergence.

We now describe each step in more detail.

3.3 Ground instances

We start by describing in detail the computation of a ground instance from the quantified system S. Traditionally, the exploration of ground instances has always been recognized as a source of helpful heuristics, especially in the verification of parameterized systems [9, 26]. The intuition is that in most cases, if a counterexample to a property exists, it can be detected for small values of the parameter. Moreover, if a property holds, the reason for that should be the same for all values of the parameter (at least after a certain threshold value). We try to use this intuition in a symbolic setting, where we use as a parameter the cardinality of the models of the theory $\mathcal{T}_I$. In the following, we denote with n an integer, and with $\underline{c} = c_1, \ldots, c_n$ a set of fresh constants of index sort. These will be frozen variables of the ground instance, i.e. we will implicitly consider a constraint $\underline{c}' = \underline{c}$ as a conjunction of the transition formula; moreover, they will be also considered all implicitly different.

In the following, if $\phi = Q_1 i_1, ..., Q_m i_m . \phi'(\underline{i}, \underline{x}[\underline{i}])$, with $Q_j \in \{\forall, \exists\}$ is a formula with quantifiers of only sort $\mathcal{T}_I$, we denote $\phi^n(\underline{c}, \underline{x}[\underline{c}])$ the ground formula obtained by expanding the quantifiers in $\underline{c}$.

Definition 3 *Given $S = (\underline{x}, \iota(\underline{x}), \tau(\underline{x}, \underline{x}'))$ a transition system and n an integer, the* ground instance *of S of size n, denoted with S^n, is obtained in the following way:*

- *for each function symbol a in Σ whose codomain type is $\mathcal{T}_I$, consider the formula*
$$\forall i_1, \ldots, i_m \exists j . a(i_1, \ldots, i_m) = j^2,$$
where m is the arity of a, and $i_1, \ldots, i_m, j$ are fresh variables of appropriate sort;
- *add the formulae generated in this way in conjuction to the initial formula ι and the transition formula τ;*
- *Instantiate all the quantifiers in the modified formulae with $\underline{c}$, thus obtaining a* ***quantifier-free*** *transition system*
$$S^n = \big(\underline{c} \cup \underline{x}, \iota^n(\underline{c}, \underline{x}[\underline{c}]), \tau^n(\underline{c}, \underline{x}[\underline{c}], \underline{x}'[\underline{c}])\big).$$

We observe that a state of S^n is given by: (i) an assignment of $\underline{c}$ to a finite model of cardinality n of $\mathcal{T}_I$, and (ii) an interpretation of the state variables as functions from that model to a model of $\mathcal{T}_E$. Note that even if the models of T_I have finte cardinality, the set of states of S^n can be infinite, since $\mathcal{T}_E$ could have an infinite model, e.g. if integer or real variables are in the system. Nevertheless, the system can be model checked efficiently by modern symbolic SMT techniques like [3].

[2] These are 'cardinality axioms', used to restrict the values of functions in appropriate models.

Symmetric presentation of ground instances As already observed in previous works [4, 15, 20], transition systems obtained by instantiating quantified formulae have a certain degree of symmetry. We report here the notion that will be useful to our description.

Definition 4 *If σ is a permutation of $1, ..., n$, and ϕ is a formula in which $c_1, \ldots, c_n$ occur free, we denote with $\sigma\phi$ the formula obtained by substituting every c_i with $c_{\sigma(i)}$.*

The following follows directly from the fact that ι^n and τ^n are obtained by instantiating a quantified formula with a set of fresh constants $\underline{c}$ [20]:

Lemma 1 *For every permutaion σ, we have that: (i) $\sigma\iota^n \equiv \iota^n$; (ii) $\sigma\tau^n \equiv \tau^n$.*

From this lemma and a simple induction proof, the following holds:

Proposition 2 (Invariance for permutation) *Let s be a state of S^n, reachable in k steps. Then $s \models \phi(\underline{c}, x[\underline{c}])$, if and only if, for every σ, there exists a state s' reachable in k steps such that $s' \models \sigma\phi(\underline{c}, x[\underline{c}])$*

This property will be exploited both for the verification of ground instances, and in the generalization process. In fact, from the last proposition we can simplify every invariant problem $S^n \models \bigwedge_\sigma \sigma\phi(\underline{c})$ – where σ ranges over all possible substitutions – to $S^n \models \phi(\underline{c})$. This simplification is of great help when checking properties which are the result of instantiating a formula with only universal quantifiers.

3.4 Generalizing invariants from instances

After computing S^n, let $\phi^n(\underline{c}, \underline{x}[\underline{c}])$ be the result of instantiating the quantifiers of the original candidate invariant ϕ in $\underline{c}$. Then, we suppose to have a model checker capable of proving or disproving that $S^n \models \phi^n(\underline{c}, \underline{x}[\underline{c}])$. If a counterexample is not found, we also suppose to have an formula $I^n(\underline{c}, \underline{x}[\underline{c}])$ which witnesses the proof, i.e. an inductive invariant. From this witness we generalize a candidate invariant for the unbounded case.

Definition 5 (Generalization) *Let S be a transition system and ϕ a candidate invariant. Let S^n be the ground instance of size n, and suppose $S^n \models \phi^n(\underline{c}, \underline{x}[\underline{c}])$. A* generalization from size n *is a (quantified) formula ψ such that ψ^n is an inductive invariant for ϕ^n.*

For generalization, we exploit the same technique that we used in [4], inspired by [26]. Suppose that I^n is in CNF. Then, $I^n = \mathcal{C}_1 \wedge \cdots \wedge \mathcal{C}_m$ is a conjunction of clauses. From every one of such clauses we will obtain a universally quantified formula. Let $\mathsf{AllDiff}(\underline{i})$ be the formula which states that all variables in $\underline{i}$ are different from each other. For all $j \in \{1, \ldots, m\}$, let $\psi_j = \forall \underline{i}.\mathsf{AllDiff}(\underline{i}) \rightarrow \mathcal{C}_j[\underline{c}/\underline{i}]$. Let $\Psi = \bigwedge_{j=1}^{m} \psi_j$. It follows from Proposition (2) than such a Ψ is a generalization from size n.

It should be clear that our technique can infer invariant strenghtnenings with only universal quantifiers, but more generalizations are possible [11, 15]. For example, if a clause of the inductive invariant is $l(c_1) \vee ... \vee l(c_n)$, a naive generalization of that clause would be $\exists x.l(x)$.

Example 3. Continuing our example, suppose to model check a ground instance with exactly two tracks (t_1, t_2) and two routes (r_1, r_2). Suppose that after succeeding in proving the property, a clause of the inductive invariant is the formula

$$\neg \mathsf{UsedBy}(t_1, r_1) \vee \neg state\_r[r_1] = \mathsf{active} \vee \neg state\_t[t_1] = \mathsf{free}$$

Our generalization is simply:

$$\forall t_1 : \mathsf{track}, r_1 : \mathsf{route}.(\neg \mathsf{UsedBy}(t_1, r_1) \vee \neg state\_r[r_1] = \mathsf{active} \vee \neg state\_t[t_1] = \mathsf{free}).$$

This is actually an inductive strengthening for the property we wanted to prove.

Minimizing modulo symmetries Recall that our next step will be to try to prove that the generalized invariant from size n is inductive also for all other ground instances. Therefore, it is intuitive to try to weaken as much as possible the candidate strenghtening Ψ, to increase the chances that its inductiveness will be preserved in other instances. So, before generalization, we use the invariant minimization techniques described in [17] to weaken the inductive invariant I^n by removing unnecessary clauses. However, note that, with our generalization technique, two symmetric clauses produce the same quantified formula: if σ is a substitution of the $\underline{c}$'s, the formulae obtained by generalizing a clause $\mathcal{C}(\underline{c})$ or $\sigma\mathcal{C}(\underline{c})$ are logically equivalent. So, we apply the following strategy: given a clause $\mathcal{C}(\underline{c})$ in I^n, we add to the invariant all the 'symmetric' versions $\sigma\mathcal{C}$, where σ ranges over all possible substitutions of the $\underline{c}$'s. By Proposition (2), we can safely add those clause to I^n and it will remain inductive. Then, during the minimization process, a clause is removed from the invariant only if all its 'symmetric' versions are. In our experiments, minimizing invariants with this method has proved to be crucial for the effectiveness of our approach.

3.5 Invariant Checking

Having described how we synthesize candidate inductive invariants from a ground instance of size n, we now describe how we try to prove that our generalization is correct. Given a candidate inductive invariant, we perform Skolemization on the inductive query (1), obtaining a universal formula. Then, we look for a set of terms G such that the ground formula obtained by instantiating the universals with G is unsatisfiable. This is the standard approach used in SMT solvers for detecting unsatisfiability of quantied formulae [10,13]. The main difference is that instead of relying on heuristics to perform the instantiation lazily during the SMT search (e.g [10,13]), we carefully control the quantifier instantiation procedure, and expand the quantifiers eagerly, so that we can use only quantifer-free SMT reasoning.

Let $\phi_S = \forall \underline{i}.\phi'_S(\underline{i}, \underline{x}[\underline{i}])$ be the result of the Skolemization process, where ϕ'_S is a quantifier-free formula over a signature Σ', obtained by expanding Σ with new Skolem symbols. Initially, we simply let G to be the set of 0-ary symbols of the index sort in the formula. Note that apart from constants in the original signature, new (Skolem) constants arise by eliminating existential quantifiers. Since we use only universal quantification the generalized invariant strengthening, Ψ is a conjunction of universal formulae, and we can commute the conjunction and the universal quantification to obtain a formula with only n universal quantified variables. Notice that, since the candidate inductive strengthening occurs also negated in the quantified formula, this will produce n new Skolem constants.

Finally, we can add to the inductive query an additional constraint. By induction on the structure of our algorithm, if Ψ is generalized from size n, we have proven already that the property ϕ holds in S for all the ground instances of size equal or less than n. Thus, we impose that in our universe G there are at least n different terms.

To sum up, let $\phi_S = \forall \underline{i}.\phi'_S(\underline{i}, \underline{x}[\underline{i}])$ be the universal formula obtained after Skolemization, and let m be the length of $\underline{i}$. Let n be the cardinality of the last visited ground instance. Let G be the set of constants of index sort in ϕ'_S (by the previous discussion, $|G| \geq n$). Let $c_1, \ldots, c_n$ be a set of fresh variables of index sort. We test with an SMT solver the satisfiability of the following formula

$$\bigwedge_{\underline{g} \in G^m} \phi'_S[\underline{i}/\underline{g}] \wedge \mathit{AllDiff}(\underline{c}) \wedge \bigwedge_{j=1}^{n} \Big(\bigvee_{g \in G} c_j = g \Big) \tag{2}$$

We have that:

Proposition 3 *For any set of Σ_I-terms G, if (2) is unsatisfiable, then ψ is an inductive strengthening for ϕ.*

Refinement If the former formula is SAT, there are two possibilities. Either we have a real counterexample to induction, and we need a better candidate, or our instantiation set G was too small to detect unsatisfiability. In general, if G covers all possible Σ_I-terms, then we can deduce that the counterexample is not spurious.

Definition 6 *Given an index theory $\mathcal{T}_I$ with signature Σ_I, we say that a set of Σ_I-terms G is saturated if, for all terms Σ_I-term t, there exists a $g \in G$ such that $\mathcal{T}_I \models t = g$.*

So, if G is saturated, any model of (2) correspond to a counterexample to induction, and we need a better strenghtening. However, in case (2) is satisfiable, but G is not saturated, we use the following heuristic to decide whether we need a better candidate or a larger G. We consider the inductive query in S^{n+1},using as a candidate inductive invariant $(\psi \wedge \phi)^{n+1}$. If the candidate invariant is still good (the query is UNSAT), we try to increase G to get the unsatisfiablity of the unbounded case. Our choice is to add to G terms of the form $f(\underline{x})$ where f is a

function symbol of index type, and $\underline{x}$ are constants already in G. Note that if no function symbols are avaible, i.e. if Σ_I is a relational signature, then saturation of G follows already by considering 0-ary terms. Therefore, in case G is initially not saturated, the existence of at least one function symbol is guaranteed [3].

If the query (2) is now UNSAT, we have succeeded. Otherwise, we continue to add terms to G, until either all function symbols have been used, or an UNSAT result is encountered. If the candidate invariant strengthening is not inductive for size $n+1$ (the query is SAT), we search for a better candidate. To do so, before completely discharging the invariant generalized from size n, we can run an additional minimization procedure (see §3.4) in size S^{n+1}, to try to remove unnecessary clauses. If a new invariant is obtained, we repeat the instantiation procedure. Otherwise, we repeat the whole loop, starting by model checking the ground instance S^{n+1} and obtaining a new strenghtening from size $n+1$.

3.6 Termination

In general, we do not have theoretical guarantees that our algorithm eventually terminates. In fact, there can be many causes of non termination: note that in case of infinite theories, the model checking of ground instances already can be non terminating. Moreover, in general universal formulae are not enough to strengthen an arbitrary invariant property [18], and existential quantification in the invariant strengthening might be needed.

However, we want to remark that even with our simple generalization technique we have obtained termination in many cases. An important remark is necessary to put more insight on the reasons of why our instantiation procedure is effective for the benchmarks we considered. In many systems descriptions, especially the ones arising from parameterized verification, the signature Σ_I is relational and all the formulae describing inductive queries contain only $\exists^*\forall^*$ quantifiers alternation. In this case, no function symbols are introduced during Skolemization: therefore, the set G of 0-ary terms already is saturated. Even in case of $\forall\exists$ alternation (but in a multi-sorted setting), saturation can be achieved after few refinement steps (as long as the Skolem functions introduced in the signature do not combine in cycles). More details about completeness of instantiation methods, especially for the verification of parameterized systems, can be found in [12, 14]. Since we limit ourselves to terms of depth one, our method can fail to prove invariants requiring some more complex instantiations. Note that in that case it is always possible to change the choice and the refinement of the set G with more sophisticated methods [13, 27]. Finally, we remark that, by limiting the possible refinements of G, our method has a notion of progress: given a transition system S and a candidate invariant ϕ, if there exists an n such that $S^n \not\models \phi^n$, and if all the model checking problems $S^{n'} \models \phi^{n'}$, with $n < n'$, terminate, then our algorithm eventually finds a counterexample.

[3] In our implementation, the saturation of G is detected when no new function symbols are available

4 Related Work

Verification of systems with quantifiers ranging over finite but unbounded domains has always received a lot of attention from the literature. A main area of application, for which our method is designed, is parameterized verification, where the parameters represent the cardinality of components of the system.

Many proposed methods for solving the problem are based on *cut-off* results. In our terms, a cut-off is an integer n such that the ground system S^n bisimulates the quantified system S. Cut-off values exist for large varieties of classes of systems, but such results strongly depend on the assumptions such as topology, data, etc (see [2] for a survey). Nonetheless, we can see a posteriori that, when our algorithm terminates with a candidate invariant from size n, such a size is a candidate cut-off, since the proof of the property for that size holds also for all integers $n' > n$.

The method of invisible invariants [26] was to our knowledge the first which proposed the usage of finite instance exploration to produce universally quantified invariant strengthening. In that paper the invariant is generalized from the formula describing the set of reachable states of the finite instance. Systems considered in that work are, however, a subclass of ours.

Tools designed for the verification of systems with a combination of first-order quantifiers and SMT theories are MCMT [14] and Cubicle [8]. These tools use the framework of array-based transition systems, of which our formalism is an extension. They implement a fully symbolic backward reachability algorithm, where pre-images of states can be described by symbolic quantified formulae. Quantified queries are then discharged with an instantiation approach similar to ours. Nonetheless, many approximations can be introduced during the backward computation, which may cause spurious counterexamples [1]. Cubicle extends this algorithm by using finite instance exploration to speed up pre-image computation [9].

Ivy [12,24] is a tool for the verification of inductive invariants of parameterized systems. Again, the formalism for defining systems considered in Ivy can be seen as a subclass of ours: a translation can be obtained if we put T_E to be the theory of Booleans, and T_I is the theory of an uninterpreted sort. In Ivy, the quantified queries can always be embedded in EPR (Effective Propositional Logic), a decidable fragment of first-order logic where formulae have a $\exists^*\forall^*$ quantifier prefix, and do not contain function symbols. Therefore, the set of possible Σ_I-terms is always finite, and it is always possible to do complete instantiations. Inspired by Ivy, MyPyvy [19] is a tool which implements algorithms for the automatic discovery of inductive invariants. Among those we have **updr** [18], a version of the IC3 algorithm capable of inferring universally quantified invariants, **fol-ic3** [19], which extends IC3 by using separators to find invariants with quantifier alternation during the construction of frames, and **PdH**, a recent algorithm which combines the duality between states and predicates to discover invariants [25].

Various tools in the literature are designed to use finite instance exploration to guess invariants to lift to the unbounded case [11, 16, 21, 22, 28]. These tools

either rely on cut-off results, or some external prover to discharge the quantified queries. Instead, we propose a tighter integration between finite instance exploration and quantified queries. Moreover, most of this approaches rely on enumerating possible models for finding inductive invariants; such an apprach cannot be extended naively to support theories. The tool IC3PO [15] proposes a generalization technique that can also infer formulae with quantifier alternation by detecting symmetries in a clausal proof. As a future work, we will try to combine their generalization technique with our method.

Abstraction methods are another major trend in verification of quantified systems. In our previous paper [4], we designed an CEGAR approach for the verification of array-based systems based on the Parameter Abstraction [20,21], which also used the ground instance exploration technique presented here. However, that approach imposed some syntactic restrictions on the definition of the systems, allowing only a single quantifier alternation with outermost existential quantification in transition formulae, and only universal quantification in the initial formula and in the candidate invariant. Our new approach, instead, is more general, and allows for arbitrary quantification in the definition of the transition system and properties to check.

5 Experimental Evaluation

To evalute our approach, we have implemented our algorithm in the tool LAMBDA, which was initially developed in a previous work [4] for model checking parameterized systems. The tool accepts as input transition systems specified either in the language of MCMT [14], IC3PO, or in VMT format (a light-weight extension of SMT-LIB to model transition systems [6]).

For clarity, we use BI-LAMBDA (for bounded induction) to denote the algorithm described in this paper, while LAMBDA will denote the previously developed method. We used the SMT-based IC3 with implicit predicate abstraction of [3] as underlying verification engine for the finite instances, and MATHSAT5 [5] as the solver for ground checks. We also have implemented a version of the algorithm which uses Z3 [23] to discharge quantified queries, to compare the effectiveness of our instantiation-based procedure (this version will be referred to as BI-LAMBDA-Z3).

In case of successful termination, we generate either a counterexample trace (for violated properties) in a concrete instance of the quantified system, or a quantified inductive invariant that proves the property. In the latter case, we can also generate proof obligations that can be independently checked with an SMT solver supporting quantifiers.

For our evaluation, we have collected a total of 183 benchmarks, divided in five different groups:

Protocols consists of 42 instances taken from the MCMT or the CUBICLE distributions. Due to the very different format, we could not run IC3PO or the MYPYVY algorithms on them. This was not only a syntactic problem: many (but not all) benchmarks contain theory variables (like integers or reals), which

are not supported in MyPyvy and Ic3po.
DynArch consists of 57 instances of verification problems of dynamic architectures, taken from [7]. These benchmarks make use of arithmetic constraints on index terms, which are not supported by Cubicle, MyPyvy or Ic3po.
PdH consists of 20 benchmarks used in the experimental evaluation of [25], written in MyPyvy and Ic3po format. We could not run MCMT, Cubicle or Lambda on them, for the different input language and for not supported features such as multi-sorted indexes.
Ic3po consists of 37 benchmarks from the experimental evaluation of [15], written in Ic3po format and in (old) MyPyvy format. For the same reasons as the previous set, we could not run MCMT, Cubicle and Lambda on them. For compatibility issues, we could not run the **PdH** algorithm on them.
Trains consists of 17 instances derived by (a simplified version of) verification problems of railway interlocking. These benchmarks contains theory variables and synctactic requirements currently supported only by Lambda and bi-Lambda.
Trains-AE consists of 10 instances derived again by verification problems on railway interlocking logics; in these systems a forall-exists quantification alternation in a multi-sorted setting occurs, along with theory variables. Therefore, they are supported only by bi-Lambda.

We have run our experiments on a cluster of machines with a 2.90GHz Intel Xeon Gold 6226R CPU running Ubuntu Linux 20.04.1, using a time limit of 1 hour and a memory limit of 4GB for each instance. For MCMT, we used standard settings. For Cubicle, we used the `--brab 2` option. For **fol-ic3** and **updr**, we used the implementation in the artifact given in [19]. For Ic3po, we used the option `--finv=2` to discharge unbounded checks with Z3. Without this option, Ic3po terminates when the tool finds a proof for size n which is still valid for size $n+1$; this was conjectured to be enough [15] to ensure that the proof was correct for all the instances, but, without unbounded checks, we have encountered errors of the tool in the PdH benchmarks. We also remark that in case of termination with Unsafe result, our tool produces always concrete counterexamples in finite instances; on the other hand, counterexamples of MCMT, Cubicle and **updr** can be spurious [1, 18] (and in theory should be checked manually). The results of bi-Lambda-z3 are obtained with a timeout of 120 seconds on every Z3 query, to avoid the prover being stuck in quantified reasoning as discussed in §3.2 (without such timeout, we have obtained worse results, both in resource usage and number of instances solved). A summary of our experimental evaluation is presented in Table 1. An extended version of the paper, containing more data of the experiments, and a virtual machine with our implementation and all the benchmarks, can be found at https://drive.google.com/file/d/1JTASD-qp5ZzJR_ADqM9qjEdSCnlANNxi.

As we can see from the table, bi-Lambda is applicable on a large set of benchmarks and in every set it is competitive with other approaches. When comparing bi-Lambda with bi-Lambda-z3, we see that our simple instantiation procedure is more effective than relying on the built-in support for quantifiers

Table 1. Summary of experimental results. In every column, we have reported the number of solved instances or '-' for incompatibility.

	Tot	**bi-Lambda**	**bi-Lambda-z3**	**Lambda**	**MCMT**	**Cubicle**	**updr**	**fol-ic3**	**PdH**	**Ic3po**
Protocols	42	34	31	34	24	30	–	–	–	–
DynArch	57	50	50	48	49	–	–	–	–	–
Trains	17	16	16	17	–	–	–	–	–	–
Trains-AE	10	10	10	–	–	–	–	–	–	–
PdH	20	11	11	–	–	–	12	11	5	12
Ic3po	37	18	16	–	–	–	18	17	–	25
Tot	183	139	134	99	73	30	30	28	5	37

in Z3, allowing to solve 5 more instances (and in general reducing the execution time, though this is not reported in Table 1 for lack of space).

6 Conclusions and Future Work

In this paper we have presented a general approach for model checking systems with quantifiers and SMT variables; the novelty in the presented algorithm relies in the tight integration between finite instance exploration and instantiation-based techniques. However, our proposed method currently synthesizes only universal invariants, which in some cases are not enough to prove properties of quantified systems. In our future works, we will investigate how to combine our approach with techniques that infer invariants with quantifier alternations. Moreover, we will study how to combine our approach with more sophisticated instantiation techniques exploited in state-of-the-art provers.

References

1. Alberti, F., Ghilardi, S., Pagani, E., Ranise, S., Rossi, G.P.: Universal guards, relativization of quantifiers, and failure models in model checking modulo theories. Journal on Satisfiability, Boolean Modeling and Computation **8**, 29–61 (2012)
2. Bloem, R., Jacobs, S., Khalimov, A.: Decidability of Parameterized Verification. Morgan & Claypool Publishers (2015)
3. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Infinite-state invariant checking with IC3 and predicate abstraction. Formal Methods Syst. Des. (2016)
4. Cimatti, A., Griggio, A., Redondi, G.: Universal invariant checking of parametric systems with quantifier-free SMT reasoning. In: CADE 28 (2021)
5. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The mathsat5 smt solver. TACAS'13, Springer-Verlag, Berlin, Heidelberg (2013)
6. Cimatti, A., Griggio, A., Tonetta, S.: The VMT-LIB language and tools. CoRR **abs/2109.12821** (2021)
7. Cimatti, A., Stojic, I., Tonetta, S.: Formal specification and verification of dynamic parametrized architectures. In: FM 2018 (2018)
8. Conchon, S., Goel, A., Krstic, S., Mebsout, A., Zaïdi, F.: Cubicle: A Parallel SMT-based Model Checker for Parameterized Systems. In: CAV 2012
9. Conchon, S., Goel, A., Krstic, S., Mebsout, A., Zaïdi, F.: Invariants for finite instances and beyond. In: Formal Methods in Computer-Aided Design, FMCAD 2013

10. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM **52**(3), 365–473 (2005)
11. Dooley, M., Somenzi, F.: Proving parameterized systems safe by generalizing clausal proofs of small instances. In: CAV 2016 (2016)
12. Feldman, Y.M.Y., Padon, O., Immerman, N., Sagiv, M., Shoham, S.: Bounded quantifier instantiation for checking inductive invariants. Log. Methods Comput. Sci. (2019)
13. Ge, Y., Barrett, C., Tinelli, C.: Solving quantified verification conditions using satisfiability modulo theories. Annals of Mathematics and Artificial Intelligence p. 101–122 (feb 2009)
14. Ghilardi, S., Ranise, S.: Backward reachability of array-based systems by SMT solving: Termination and invariant synthesis. Log. Methods Comput. Sci. **6**(4) (2010)
15. Goel, A., Sakallah, K.A.: On symmetry and quantification: A new approach to verify distributed protocols. In: NFM 2021 (2021)
16. Hance, T., Heule, M., Martins, R., Parno, B.: Finding invariants of distributed systems: It's a small (enough) world after all. In: NSDI 2021. pp. 115–131. USENIX Association (2021)
17. Ivrii, A., Gurfinkel, A., Belov, A.: Small inductive safe invariants. In: Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014. pp. 115–122. IEEE (2014)
18. Karbyshev, A., Bjørner, N., Itzhaky, S., Rinetzky, N., Shoham, S.: Property-directed inference of universal invariants or proving their absence. In: Kroening, D., Păsăreanu, C.S. (eds.) Computer Aided Verification (2015)
19. Koenig, J.R., Padon, O., Immerman, N., Aiken, A.: First-order quantified separators. In: PLDI (2020)
20. Krstic, S.: Parametrized system verification with guard strengthening and parameter abstraction (2005)
21. Li, Y., Duan, K., Jansen, D.N., Pang, J., Zhang, L., Lv, Y., Cai, S.: An automatic proving approach to parameterized verification. ACM Trans. Comput. Logic (Nov 2018)
22. Ma, H., Goel, A., Jeannin, J.B., Kapritsos, M., Kasikci, B., Sakallah, K.A.: I4: Incremental inference of inductive invariants for verification of distributed protocols. SOSP '19 (2019)
23. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: TACAS (2008)
24. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: Safety verification by interactive generalization. SIGPLAN Not. **51**(6), 614–630 (Jun 2016)
25. Padon, O., Wilcox, J.R., Koenig, J.R., McMillan, K.L., Aiken, A.: Induction duality: Primal-dual search for invariants **6**(POPL) (2022)
26. Pnueli, A., Ruah, S., Zuck, L.D.: Automatic deductive verification with invisible invariants. In: TACAS (2001)
27. Reynolds, A.: Quantifier instantiation beyond e-matching. In: Brain, M., Hadarean, L. (eds.) (CAV 2017) (2017)
28. Yao, J., Tao, R., Gu, R., Nieh, J., Jana, S., Ryan, G.: DistAI: Data-Driven automated invariant learning for distributed protocols. In: (OSDI 21) (Jul 2021)