Bounded Quantifier Instantiation for Checking Inductive Invariants

Yotam M. Y. Feldman[1], Oded Padon[1], Neil Immerman[2], Mooly Sagiv[1], and Sharon Shoham[1]

[1] Tel Aviv University, Tel Aviv, Israel
[2] UMass, Amherst, USA

Abstract. We consider the problem of checking whether a proposed invariant φ expressed in first-order logic with quantifier alternation is *inductive*, i.e. preserved by a piece of code. While the problem is undecidable, modern SMT solvers can sometimes solve it automatically. However they employ powerful quantifier instantiation methods that may diverge, especially when φ is not preserved. A notable difficulty arises due to counterexamples of infinite size.

This paper studies *Bounded-Horizon instantiation*, a natural method for guaranteeing the termination of SMT solvers. The method bounds the depth of terms used in the quantifier instantiation process. We show that this method is surprisingly powerful for checking quantified invariants in uninterpreted domains. Furthermore, by producing partial models it can help the user diagnose the case when φ is not inductive, especially when the underlying reason is the existence of infinite counterexamples.

Our main technical result is that Bounded-Horizon is at least as powerful as *instrumentation*, which is a manual method to guarantee convergence of the solver by modifying the program so that it admits a purely universal invariant. We show that with a bound of 1 we can simulate a natural class of instrumentations, without the need to modify the code and in a fully automatic way. We also report on a prototype implementation on top of Z3, which we used to verify several examples by Bounded-Horizon of bound 1.

1 Introduction

This paper addresses a fundamental problem in automatic program verification: how to prove that a piece of code preserves a given invariant. In Floyd-Hoare style verification this means that we want to automatically prove the validity of the Hoare triple $\{P\}C\{P\}$ where P is an assertion and C is a command. Alternatively, this can be shown by proving the unsatisfiability of the formula $P(V) \wedge \delta(V, V') \wedge \neg P(V')$ (the *verification condition*) where $P(V)$ denotes the assertion P before the command, $P(V')$ denotes the assertion P after the command, and $\delta(V, V')$ is a formula expressing the meaning of the command C as a transition relation between pre- and post-states. When C is a loop body, such a P is an inductive invariant and can be used to prove safety properties of the loop (if it also holds initially and implies the desired property).

For programs with infinite state space, proving the validity of $\{P\}C\{P\}$ is generally undecidable even when C does not include loops. Indeed, existing SMT solvers

can diverge even for simple assertions and simple commands. Recent attempts to apply program verification to prove the correctness of critical system's design and code [16] identify this as the main hurdle for using program verification.

The difficulty is rooted in powerful constructs used in SMT-based verification of interesting programs. Prominent among these constructs are arithmetic and other program operations modeled using background theories, and logical quantifiers. In this paper we target the verification of applications in which the problem can be modeled without interpreted theories. This is in line with recent works that show that although reasoning about arithmetic is crucial for low-level code, in many cases the verification of high-level programs and designs can be performed by reasoning about quantification in uninterpreted theories. Specifically, the decidable Effectively Propositional logic (EPR) has been successfully applied to domains such as linked-list manipulation [21], Software-Defined Networks [6] and some distributed protocols [29]. Without interpreted theories it remains to address the complications induced by the use of quantifier alternation.

In the presence of quantifier alternation, the solver's ability to check assertions is hindered by the following issues: (1) an infinite search space of proofs that needs to be explored for correct assertions, a problem which is sometimes manifested in matching loops [12], and (2) a difficulty of finding counterexamples for invalid assertions, notably when counterexamples may be of infinite size. Current SMT techniques often fail to produce models of satisfiable quantified formulas [15,34], which is somewhat unfortunate since one of the main values of program verification is early detection of flaws in designs and programs. The existence of infinite counterexamples is a major complication as they are difficult to find. In uninterpreted domains, infinite counterexamples usually do not indicate a real violation of the verification conditions and are counterintuitive to programmers, yet render assertions invalid in the context of general first-order logic (on which SMT proof techniques are based). Hence infinite counter-models pose a real problem in the verification process.

Previous work on EPR [21,6,29] used universally quantified invariants with programs expressed by $\exists^*\forall^*$ formulas[3], in which case checking inductive invariants is decidable, hence problems (1) and (2) do not occur. In particular, EPR enjoys the finite-model property and so counterexamples are of finite size. EPR programs are in fact Turing-complete [29], but universal invariants are not always sufficient to express the required program properties.

For example, [16] describes a client server scenario where the invariant is "For every reply message sent by the server, there exists a corresponding request message sent by a client". (See Example 1 for further details.) This invariant is $\forall^*\exists^*$ and thus leads to verification conditions with quantifier alternation. This kind of quantifier alternation may lead to divergence of the solver as problems (1) and (2) re-emerge.

The current work aims to expand the applicability of the EPR-based verification approach to invariants of more complex quantification. We focus on the class $\forall^*\exists^*$ invariants, which arise in interesting programs. As we show, checking inductiveness of invariants in this class is undecidable. We thus study problems (1),(2) above for this

[3] Automated tools that extract EPR transition relation from code exist for C code manipulating linked lists [21,22,23] and for the modeling language RML [29] which is Turing-complete.

setting using the notion of *bounded quantifier instantiations*, which we term *Bounded-Horizon*.

Main results. This paper explores the utility of limited quantifier instantiations for checking $\forall^*\exists^*$invariants, and for dealing with the problems that arise from quantifier alternation: divergence of the proof search and infinite counter-models.

We consider instantiations that are *bounded in depth* of terms. Bounded instantiations trivially prevent divergence while maintaining soundness. Although for a given bound the technique is not complete, i.e. unable to prove every correct invariant, we provide completeness guarantees by comparing bounded instantiations to the method of *instrumentation*, a powerful technique implicitly employed in previous works [21,23,29]. Instrumentation tackles a $\forall^*\exists^*$invariant by transforming the program in a way that allows the invariant to be expressed in a universal form, and, accordingly, makes the verification conditions fall in EPR. We show that for invariants that can be proven using a typical form of instrumentation, bounded instantiations of a small bound are also complete. Namely, they are sufficiently powerful to prove the original program without modifications and in a fully automatic way. This is encouraging since instrumentation is labor-intensive and error-prone while bounded instantiations are completely automatic.

This result suggests that in many cases correct $\forall^*\exists^*$invariants of EPR programs can be proven using a simple proof technique. Typically in such cases tools such as Z3 will also manage to automatically prove the verification conditions. However, bounded instantiations guarantee termination a-priori even when the invariant is not correct. When it terminates, the procedure returns a logical structure which is not necessarily a true counterexample but "approximates" it, as it satisfies all the bounded instantiations. Interestingly, this suggests a way to overcome the problem of infinite models. This problem arises when the user provides an invariant that is correct for finite models but is incorrect in general first-order logic. In such cases, state-of-the-art SMT solvers typically produce "unknown" or timeout since they fail to find infinite models. Thus the user is left with very little aid from the solver when attempting to make progress and successfully verify the program. In contrast, bounded quantifier instantiation can be used to find finite models with increasing sizes, potentially indicating the existence of an infinite model, and provide hints as to the source of the error. This information allows the user to modify the program or the invariant to exclude the problematic models. We demonstrate this approach on a real example in which such a scenario occurred in one of our verification attempts. We show that the provided models assist in identifying and fixing the error, allowing the user to successfully verify the program.

We also implemented a prototype tool that performs bounded instantiations of bound 1, and used it to verify several distributed protocols and heap manipulating programs. The implementation efficiently reduces the problem of checking inductiveness with bound 1 to a Z3 satisfiability check on which the solver always terminates , thereby taking advantage of Z3's instantiation techniques while guaranteeing termination.

2 Preliminaries

In this section we provide background and explain our notation. Σ will always denote a relational first-order vocabulary, which may contain constant symbols, c_i, and relation

symbols, r_j, but no function symbols. For a formula φ we denote by const$[\varphi]$ the set of constants that appear in φ. We write that $\varphi \in \exists^*(\Sigma)$ to mean that φ is an *existential* formula defined over vocabulary Σ. Similarly, the class of *universal* formulas is denoted by $\forall^*(\Sigma)$. We say that φ is *quantifier-free*, denoted $\varphi \in \text{QF}(\Sigma)$ if it contains no quantifiers, and that it is *alternation free*, denoted $\varphi \in \text{AF}(\Sigma)$, if it can be written as a Boolean combination of formulas in $\exists^*(\Sigma)$. FOL(Σ) stands for arbitrary first-order formulas over Σ. A *sentence* is a closed formula.

EPR. The effectively-propositional (EPR) fragment of first-order logic, also known as the Bernays-Schönfinkel-Ramsey class, consists of $\exists^*\forall^*(\Sigma)$ sentences. Such sentences enjoy the *small model property*. Thus satisfiability of EPR sentences is decidable [31].

EPR Transition Relation. We specify a transition relation via an EPR sentence, δ, over a vocabulary $\Sigma \uplus \Sigma'$ where Σ is a relational vocabulary used to describe the source state of a transition and $\Sigma' = \{a' \mid a \in \Sigma\}$ is used to describe the target state.

Inductive Invariants. A first-order sentence I over Σ is an *inductive invariant* for δ if $I \wedge \delta \rightarrow I'$ is valid, or, equivalently, if $I \wedge \delta \wedge \neg I'$ is unsatisfiable [4], where I' results from substituting every constant and relation symbol in I by its primed version.

Skolemization. Let $\varphi(z_1, \ldots, z_n) \in \text{FOL}(\Sigma)$. The *Skolemization* of φ, denoted φ_S, is a universal formula over $\Sigma \uplus \Sigma_S$, where Σ_S consists of fresh constant symbols and function symbols, obtained as follows. We first convert φ to negation normal form (NNF) using the standard rules. For every existential quantifier $\exists y$ that appears under the scope of the universal quantifiers $\forall x_1, \ldots, \forall x_m$, we introduce a fresh function symbol $f_y \in \Sigma_S$ of arity $n + m$. We replace each bound occurrence of y by $f_y(z_1, \ldots, z_n, x_1, \ldots, x_m)$, and remove the existential quantifier. If $n + m = 0$ (i.e., φ has no free variables and $\exists y$ does not appear in the scope of a universal quantifier) a fresh constant symbol is used to replace y. It is well known that $\varphi_S \rightarrow \varphi$ is valid and φ_S and φ are equi-satisfiable.

3 Bounded-Horizon

In this section, we define a systematic method of quantifier instantiation called *Bounded-Horizon* as a way of checking the inductiveness of first-order logic formulas, and explore some of its basic properties. We start with the undecidability of the problem.

Undecidability of inductiveness. For a universal formula $I \in \forall^*(\Sigma)$, checking inductiveness amounts to checking unsatisfiability of an EPR formula, and is therefore decidable. The same holds for $I \in AF(\Sigma)$. However, this is no longer true when quantifier alternation is introduced. For example, checking inductiveness of $I \in \forall^*\exists^*(\Sigma)$ amounts to checking unsatisfiability of a formula in a fragment for which satisfiability is undecidable. In fact we prove that:

Theorem 1. *The problem of determining on input $I \in \forall^*\exists^*(\Sigma)$ and $\delta \in \exists^*\forall^*(\Sigma, \Sigma')$, whether I is an inductive invariant for δ, is undecidable.*

Proof Sketch. The proof is by reduction from the halting problem, which can be encoded using a $\forall^*\exists^*$ formula via tiling (see e.g. [20]). For the setting of checking invariants, we start with a Turing machine M, and construct $\delta \in \exists^*\forall^*(\Sigma, \Sigma')$, and

[4] In this paper, satisfiability and validity refer to general models, not restricted to finite models.

$I \in \forall^*\exists^*(\Sigma)$ s.t. I is an inductive invariant for δ iff M halts on the empty tape. In case M does not halt, the counter-model that shows that I is not inductive is an infinite structure which encodes an infinite run of M. □

Bounded-Horizon Instantiations. Let $\delta \in \exists^*\forall^*(\Sigma, \Sigma')$ be an EPR transition relation and $I \in \text{FOL}(\Sigma)$ a candidate invariant. We would like to check the satisfiability of $I \wedge \delta \wedge \neg I'$, and equivalently of $\mathit{Ind} = I_S \wedge \delta_S \wedge (\neg I')_S$. Recall that φ_S denotes the Skolemization of φ, and note that I_S and $(\neg I')_S$ possibly add Skolem functions to the vocabulary. Roughly speaking, for a given $k \in \mathbb{N}$, Bounded-Horizon instantiates the universal quantifiers in *Ind*, while restricting the instantiations to produce ground-terms of function nesting at most k.

Below we provide the formal definitions and discuss soundness and (in)completeness. We start with the notion of instantiations, and recall Herbrand's theorem which establishes completeness of proof by (unrestricted) instantiations. Suppose that some vocabulary $\tilde{\Sigma}$ including constants and function symbols is understood (e.g., $\tilde{\Sigma} = \Sigma \uplus \Sigma_S$, where Σ_S includes Skolem constants and function symbols).

Definition 1 (Instantiation). *Let $\varphi(\overline{x}) \in \forall^*(\tilde{\Sigma})$ be a universal formula with n free variables and m universal quantifiers. An* instantiation *of φ by a tuple $\bar{t}$ of $n + m$ ground terms, denoted by $\varphi[\bar{t}\,]$, is obtained by substituting $\bar{t}$ for the free variables and the universally quantified variables, and then removing the universal quantifiers.*

Note that an instantiation is a quantifier-free sentence.

Theorem 2 (Herbrand's Theorem). *Let $\varphi \in \forall^*(\tilde{\Sigma})$. Then φ is satisfiable iff the (potentially infinite) set $\left\{\varphi[\bar{t}\,] \mid \bar{t} \text{ is a tuple of ground terms over } \tilde{\Sigma}\right\}$ is satisfiable.*

We now turn to restrict the depth of terms used in instantiations.

Definition 2 (Bounded-Depth Terms). *For every $k \in \mathbb{N}$, we define BHT_k to be the set of ground terms over $\tilde{\Sigma}$ with function symbols nested to depth at most k. BHT_k is defined by induction over k. Let C be the set of constants in $\tilde{\Sigma}$, F the set of functions, and for every $f \in F$ let Arity_f be the arity of f. Then $\text{BHT}_0 = C$ and for $k > 0$:*

$$\text{BHT}_k = \text{BHT}_{k-1} \cup \{f(t_1, \ldots t_m) \mid f \in F,\ m = \mathit{Arity}_f,\ t_1, \ldots, t_m \in \text{BHT}_{k-1}\}.$$

We will also write $\bar{t} \in \text{BHT}_k$ for a tuple of terms $\bar{t}$, to mean that every entry of $\bar{t}$ is in BHT_k (the number of elements in $\bar{t}$ should be clear from the context). Note that the set of ground terms is $\text{BHT}_\infty = \bigcup_{k\in\mathbb{N}} \text{BHT}_k$.

Definition 3 (Depth of Instantiation). *Let $\varphi \in \forall^*(\tilde{\Sigma})$ and $\bar{t} \in \text{BHT}_\infty$. The* depth of instantiation*, denoted depth$(\varphi[\bar{t}\,])$, is the smallest k such that all ground terms that appear in $\varphi[\bar{t}\,]$ are included in BHT_k.*

Bounded-Horizon algorithm. Given a candidate invariant $I \in \text{FOL}(\Sigma)$, a transition relation δ over $\Sigma \uplus \Sigma'$, and $k \in \mathbb{N}$, the Bounded-Horizon algorithm constructs the formula $\mathit{Ind} = I_S \wedge \delta_S \wedge (\neg I')_S$, and checks if the set

$$\left\{\mathit{Ind}[\bar{t}\,] \mid \bar{t} \in \text{BHT}_k,\ \text{depth}(\mathit{Ind}[\bar{t}\,]) \leq k\right\} \tag{1}$$

```
req := ∅; resp := ∅; match := ∅;
action new_request(u) {
  q := new request;
  req := req ∪ {(u, q)}
  /@   r := r ∪ {(u, y) | match(q, y)}
}
action respond(u, q) {
  assume req(u, q);
  p := new response;
  match := match ∪ {(q, p)};
  /@   r := r ∪ {(x, p) | req(x, q)}
  resp := resp ∪ {(u, p)}
}
```

```
action check(u, p) {
  if resp(u, p) ∧ ∀q. req(u, q) → ¬match(q, p)
  /@ ↪ if resp(u, p) ∧ ¬r(u, p)
    then abort
}

Invariant I = ∀u, p. resp(u, p) →
                ∃q. req(u, q) ∧ match(q, p)
/@   r(x, y) ≡ ∃z. req(x, z) ∧ match(z, y)
/@   Invariant Î = ∀u, p. resp(u, p) → r(u, p)
```

Fig. 1. Example demonstrating a $\forall^*\exists^*$ invariant that is provable with bound 1. The reader should first ignore the instrumentation code denoted by /@ (see Example 2). The complete program is provided in [2] (files `client_server_ae.ivy`, `client_server_instr.ivy`).

is unsatisfiable. If it is, then I is provably inductive w.r.t. δ with Bounded-Horizon of bound k. Otherwise we report that I is not known to be inductive.

Note that the satisfiability check performed by Bounded-Horizon is decidable since the set of instantiations is finite, and each of them is a ground formula.

Bounded-Horizon for $\forall^*\exists^*$ Invariants. We illustrate the definition of Bounded-Horizon in the case that $I \in \forall^*\exists^*(\Sigma)$. Assume that $I = \forall\overline{x}.\ \exists\overline{y}.\ \alpha(\overline{x}, \overline{y})$ where $\alpha \in$ QF. Then $I_S = \forall\overline{x}.\ \alpha(\overline{x}, \overline{f}(\overline{x}))$ where $\overline{f}$ are new Skolem function symbols. δ_S introduces Skolem constants but no function symbols, and in this case so does $(\neg I')_S$. Bounded-Horizon check of bound k can be approximately understood as checking the satisfiability of

$$\Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_{k-1}} I_S[\overline{t}\,]\Big) \wedge \Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_{k}} \delta_S[\overline{t}\,]\Big) \wedge \Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_{k}} (\neg I')_S[\overline{t}\,]\Big). \tag{2}$$

(In fact, it is possible that I_S contains sub-formulas for which instantiations of depth k do not increase the total depth of instantiations beyond k, and are thus also included.)

Lemma 1 (Soundness). *For every $k \in \mathbb{N}$, Bounded-Horizon with bound k is sound, i.e., if it reports that $I \in \mathrm{FOL}(\Sigma)$ is inductive w.r.t. δ, then I is indeed inductive.*

Proof. Assume that I is not inductive w.r.t. δ, so there is a structure $\mathcal{A}$ such that $\mathcal{A} \models I_S \wedge \delta_S \wedge (\neg I')_S$. In particular $\mathcal{A} \models \mathit{Ind}[\,\overline{t}\,]$ for every $\overline{t} \in \mathrm{BHT}_\infty$ and in particular for every $\overline{t} \in \mathrm{BHT}_k$ such that $\mathrm{depth}(\mathit{Ind}[\,\overline{t}\,]) \le k$. Hence, Bounded-Horizon of bound k will not report that I is inductive. □

Example 1. Fig. 1 presents a simple model of the client server scenario described in [16]. The program induces an EPR transition relation, and its invariant is provable by Bounded-Horizon of bound 1.

We first explain this example ignoring the annotations denoted by "/@". The system state is modeled using three binary relations. The *req* relation stores pairs of users and requests, representing requests sent by users. The *resp* relation similarly stores pairs

of users and replies, representing replies sent back from the server. The *match* relation maintains the correspondence between a request and its reply.

The action `new_request` models an event where a user u sends a new request to the server. The action `respond` models an event where the server responds to a pending request by sending a reply to the user. The request and response are related by the *match* relation. The action `check` is used to verify the safety property that every response sent by the server has a matching request, by aborting the system if this does not hold.

A natural inductive invariant for this system is

$$I = \forall u, p.\ \mathit{resp}(u, p) \rightarrow \exists q.\ \mathit{req}(u, q) \wedge \mathit{match}(q, p).$$

The invariant proves that the `then` branch in action `check` will never happen and thus the system will never abort. This invariant is preserved under execution of all actions, and is provable by Bounded Horizon of bound 1.

Lemma 2 (Completeness for some k). *If $I \in \mathrm{FOL}(\Sigma)$ is inductive w.r.t. δ then there exists $k \in \mathbb{N}$ s.t. I is provably inductive w.r.t. δ with Bounded-Horizon of bound k.*

Proof. From Theorem 2 and compactness there is a finite set S of instantiations that is unsatisfiable. Take k to be the maximal depth of instantiations in S. □

For example, if $I \in \forall^*$ then Bounded-Horizon of bound 0 is complete. However, as expected due to the undecidability of checking inductiveness, for arbitrary invariants Bounded-Horizon is *not* necessarily complete for a given k: An example for which a bound of 1 is insufficient appears in the extended version [1].

Small Bounded-Horizon for $\forall^*\exists^*$ Invariants. Despite the incompleteness, we conjecture that a small depth of instantiations typically suffices to prove inductiveness. The intuition is that an EPR transition relation has a very limited "horizon" of the domain: it interacts only with a small fraction of the domain, namely elements pointed to by program variables (that correspond to logical constants in the vocabulary).

When performing the Bounded-Horizon check with bound 1 on a $\forall^*\exists^*$ invariant $I = \forall \overline{x}.\ \exists \overline{y}.\ \alpha(\overline{x}, \overline{y})$, we essentially assume that the existential part of the invariant $\psi(\overline{x}) = \exists \overline{y}.\ \alpha(\overline{x}, \overline{y})$ holds on all program variables — but not necessarily on all elements of the domain — and try to prove that it holds on all elements of the domain after the transition. We expect that for most elements of the domain, the correctness of ψ is maintained simply because they were not modified at all by the transition. For elements that are modified by the transition, we expect the correctness after modifications to result from the fact that ψ holds for the elements of the domain that the transition directly interacts with. If this is indeed the reason that ψ is maintained, a bound of 1 sufficiently uses ψ in the pre-state to prove the invariant in the post-state, i.e. it is inductive.

This is the case in Example 1. Additional examples are listed in Section 6.

4 Power of Bounded-Horizon for Proving Inductiveness

We now turn to investigate the ability of Bounded-Horizon to verify inductiveness. In this section we provide sufficient conditions for its success by relating it to the notion of instrumentation (which we explain below). We show that Bounded-Horizon with a

low bound of 1 or 2 is as powerful as a natural class of sound program instrumentations, those that do not add existential quantifiers. Section 6 demonstrates the method's power on several interesting programs we verified using Bounded-Horizon of bound 1.

4.1 Instrumentation

We present our view of the instrumentation procedure used in previous works [21,23,29] to eliminate the need for quantifier-alternation, thus reducing the verification task to a decidable fragment. The procedure begins with a program that induces a transition relation $\delta \in \exists^*\forall^*(\Sigma \cup \Sigma')$. The purpose of instrumentation is to modify δ into another transition relation $\widehat{\delta}$ that admits an inductive invariant with simpler quantification (e.g., universal, in which case it is decidable to check). We note that instrumentation is generally a manual procedure. For simplicity, we describe the instrumentation process informally, but provide the semantic soundness requirement in Definition 4. The instrumentation procedure consists of the following three steps:

1. Identify a formula $\psi(\overline{x}) \in \text{FOL}(\Sigma)$ (usually ψ will be existential) that captures information that is needed in the inductive invariant. Extend the vocabulary with an instrumentation relation $r(\overline{x})$ that intentionally should capture the derived relation defined by $\psi(\overline{x})$. Let $\widehat{\Sigma} = \Sigma \cup \{r\}$ denote the extended vocabulary[5].
2. Add update code that updates r when the original ("core") relations are modified, and maintains the meaning of r as encoding ψ. The update code must not block executions of real code, and can possibly be a sound approximation. Sometimes it can be generated automatically via finite differencing [32].
3. Modify the program to use r. Often this is performed by rewriting some program conditions, keeping in mind that r encodes ψ. This means replacing some quantified expressions by uses of r.

Example 2. In the example of Fig. 1, to achieve a universal invariant we add an instrumentation relation *r* defined by $r(x, y) \equiv \exists z.\ \mathit{req}(x, z) \wedge \mathit{match}(z, y)$ (step 1). The simple form of ψ allows us to obtain precise update code, which appears as annotations marked with /@ in lines that mutate *req* and *match* (step 2). We also replace the `if` condition in the action `check` by an equivalent condition that uses *r* (step 3). The line marked with /@ $\hookrightarrow$ in the `check` action replaces the line above it. The resulting program has the invariant $\widehat{I} = \forall u, p.\ \mathit{resp}(u, p) \rightarrow r(u, p)$, which is universal.

Let $\widehat{\delta} \in \exists^*\forall^*(\widehat{\Sigma} \cup \widehat{\Sigma}')$ denote the transition relation induced by the modified program (modifications occur in steps 2,3). The soundness of the instrumentation procedure is formalized in the following connection between ψ, δ, and $\widehat{\delta}$:

Definition 4 (Sound Instrumentation). $\widehat{\delta} \in \exists^*\forall^*(\widehat{\Sigma} \cup \widehat{\Sigma}')$ *is a* sound instrumentation *for* $\delta \in \exists^*\forall^*(\Sigma \cup \Sigma')$ *and* $\psi \in \text{FOL}(\Sigma)$ *if* $\big(\forall \overline{x}.\ r(\overline{x}) \leftrightarrow \psi(\overline{x}) \ \wedge\ \delta \ \wedge\ \forall \overline{x}.\ r'(\overline{x}) \leftrightarrow \psi'(\overline{x})\big) \rightarrow \widehat{\delta}$ *is valid, or equivalently,* $\delta \rightarrow \widehat{\delta}[\psi/r, \psi'/r']$ *is valid.*

[5] It is also possible to instrument the program with constants . This can be emulated by adding a unary relation $c(x)$ representing the constant, and adding the assumption that c contains exactly one element to the invariant. This is also aligned with the conditions of Theorem 5.

Definition 4 ensures that the instrumented program includes at least all the behaviors of the original program, when r is interpreted according to ψ. Thus, if the instrumented program is safe, then it is sound to infer that the original program is safe.

The instrumentation procedure does not require the user to know an inductive invariant for the original program. However, if a sound instrumentation which leads to an invariant exists, then an inductive invariant for the original δ can be produced by substituting back the "meaning" of r as ψ (thus, safety of the original program is implied):

Lemma 3. *Let $\widehat{\delta}$ be a sound instrumentation for δ and ψ, and $\widehat{I} \in \mathrm{FOL}(\widehat{\Sigma})$ be an inductive invariant for $\widehat{\delta}$. Then $I = \widehat{I}[\psi/r]$ is inductive w.r.t. δ.*

Proof. $\widehat{I} \wedge \widehat{\delta} \rightarrow \widehat{I}'$ is valid, thus, so is $(\widehat{I} \wedge \widehat{\delta} \rightarrow \widehat{I}')[\psi/r, \psi'/r']$. $\widehat{\delta}$ is a sound instrumentation for δ, so (using Definition 4) $I \wedge \delta \rightarrow I'$ is valid. □

Note that typically the quantification structure of I is more complex than that of $\widehat{I}$.

Instrumentation without additional existential quantifiers. In order to relate instrumentation to Bounded-Horizon instantiations, we consider the typical case where the instrumentation process of δ does not add new existential quantifiers to $\widehat{\delta}$. This happens when the update code does not introduce additional existential quantifiers. Formally:

Definition 5 (Existential Naming). *Let $\widehat{\delta} = \exists z_1, \ldots z_m.\ \varphi(z_1, \ldots, z_m)$ where $\varphi \in \forall^*(\widehat{\Sigma}, \widehat{\Sigma}')$. An* existential naming *η for $(\widehat{\delta}, \delta)$ is a mapping $\eta : \{z_1, \ldots, z_m\} \rightarrow$ const$[\delta_S] \cup$ const$[\widehat{\delta}_S]$. We define $\eta(\widehat{\delta})$ to be $\varphi[\eta(z_1)/z_1, \ldots, \eta(z_m)/z_m]$.*

An existential naming provides a Skolemization procedure which uses existing constants rather than fresh ones. If such η exists, it maps the (Skolemized) existential quantifiers in $\widehat{\delta}$ to their counterparts in δ. For example, the instrumentation in Fig. 1 results in $\widehat{\delta}$ that has an existential naming w.r.t. the original δ. Note that it is possible that $\widehat{\delta}$ has in fact *fewer* existential quantifiers than δ, for example due to the rewriting of conditions (as happens in the example of Fig. 1 — see the `if` statement in action `check`).

Definition 6 (Instrumentation Without Additional Existenials). *$\widehat{\delta}$ is a* sound instrumentation without additional existentials *for δ if there exists an existential naming η such that $\delta_S \rightarrow \eta(\widehat{\delta})[\psi/r, \psi'/r']$ is valid.*

4.2 From Instrumentation to Bounded-Horizon

The results described in this section show that if there is an instrumentation without additional existentials, then Bounded-Horizon with a low bound is able to prove the original invariant, without specific knowledge of the instrumentation and without manual assistance from the programmer. This is the case in the example of Fig. 1, which admits an instrumentation that transforms the invariant to a universal invariant (see Example 2) in a form that matches Theorem 3, and indeed the original invariant is provable by Bounded-Horizon of bound 1.

Interestingly, in case Bounded-Horizon with a small bound does not prove inductiveness (see the example in the extended version [1]), the results imply that either the

invariant is not inductive or *no instrumentation* that does not add existential quantifiers can be used to show that it is inductive (even with the programmer's manual assistance).

In the remainder of this section we will assume that $\widehat{\delta}$ is a sound instrumentation without additional existentials for δ, and η is the corresponding naming of existentials. Further, we assume that $\widehat{I}$ is an inductive invariant for $\widehat{\delta}$ and denote $I = \widehat{I}[\psi/r]$.

The following theorems state our results for $I \in \forall^*\exists^*$.

Theorem 3. *Let $\widehat{I} \in \forall^*$. Assume $\psi \in \exists^*$ and r appears only positively in $\widehat{I}$, or $\psi \in \forall^*$ and r appears only negatively in $\widehat{I}$. Then $I = \widehat{I}[\psi/r]$ is inductive for δ with Bounded-Horizon of bound* 1. *(Note that $I \in \forall^*\exists^*$.)*

Proof Sketch. Let $I = \forall\overline{x}.\ \alpha(\overline{x})$ where $\alpha \in \exists^*$. Assume for the sake of contradiction that I is not inductive for δ with Bounded-Horizon of bound 1. By the assumptions on ψ and $\widehat{I}$, this means that there is a structure $\mathcal{A}$ such that

$$\mathcal{A} \models \left(\bigwedge_{\overline{c}} \alpha(\overline{c})\right) \ \wedge\ \delta_S \ \wedge\ (\neg I')_S\ .$$

From the assumption (Definition 6) and properties of Skolemization, it follows that

$$\mathcal{A} \models \left(\bigwedge_{\overline{c}} \alpha(\overline{c})\right) \ \wedge\ \left(\eta(\widehat{\delta})\right)[\psi/r, \psi'/r'] \ \wedge\ \left((\neg\widehat{I}')_S\right)[\psi/r, \psi'/r']\ .$$

From the assumptions on the way ψ appears in $\widehat{I}$, when we write $\widehat{I} = \forall\overline{x}.\ \widehat{\alpha}(\overline{x})$ where $\widehat{\alpha} \in$ QF we have $\alpha = \widehat{\alpha}[\psi/r]$. Thus, from properties of substitution (interpreting r, r' according to ψ, ψ' in $\mathcal{A}$) it follows that there is a structure $\widehat{A}$ such that

$$\widehat{\mathcal{A}} \models \left(\bigwedge_{\overline{c}} \widehat{\alpha}(\overline{c})\right) \ \wedge\ \eta(\widehat{\delta}) \ \wedge\ (\neg\widehat{I}')_S.$$

By reducing $\widehat{\mathcal{A}}$'s domain to the constants we have that $\left(\forall\overline{x}.\ \widehat{\alpha}(\overline{x})\right) \wedge \eta(\widehat{\delta}) \wedge (\neg\widehat{I}')_S$ is satisfiable. (This is a use of complete instantiation for universal formulas.)

This in turn implies (by properties of Skolemization) that $\widehat{I} \wedge \widehat{\delta} \wedge \neg\widehat{I}'$ is satisfiable, which is a contradiction to the assumption that $\widehat{I}$ is inductive for $\widehat{\delta}$. □

Theorem 4. *Let $\widehat{I} \in \forall^*$. If $\psi \in$ AF then $I = \widehat{I}[\psi/r]$ is inductive for δ with Bounded-Horizon of bound* 2. *(Note that $I \in \forall^*\exists^*$.)*

The following theorem generalizes the above result to *1-alternation invariants*. A formula is 1-alternation if it can be written as a Boolean combination of $\forall^*\exists^*$ formulas.

Theorem 5. *Let $\widehat{I} \in$ AF. If $\psi \in$ AF then $I = \widehat{I}[\psi/r]$ is inductive for δ with Bounded-Horizon of bound* 2. *(Note that $I \in$ 1-alternation.)*

The full proofs appear in the extended version [1]. The results of this section also apply when multiple instrumentation relations $\psi_1, \ldots, \psi_t \in \text{FOL}(\Sigma)$ are simultaneously substituted instead of the relation symbols $r_1, \ldots, r_t$ in $\widehat{\delta}$ and $\widehat{I}$.

Instrumentations for higher bounds. While instrumentation that does not add existentials is at most as powerful as Bounded-Horizon with a low bound, sound instrumentations that do add existentials to the program (thereby not satisfying Definition 6) can be used to simulate quantifier instantiation of an arbitrary depth. A simple way is to add r as an instrumentation that tracks the existential part of a $\forall^*\exists^*$ invariant. Instantiations are performed by introducing existentially quantified variables to the program and using `assume` statements to make these variables function as witnesses for a tuple of variables that instantiate the universal quantifiers. Doing this recursively generates instantiations of an arbitrary depth. See the extended version [1] for further details.

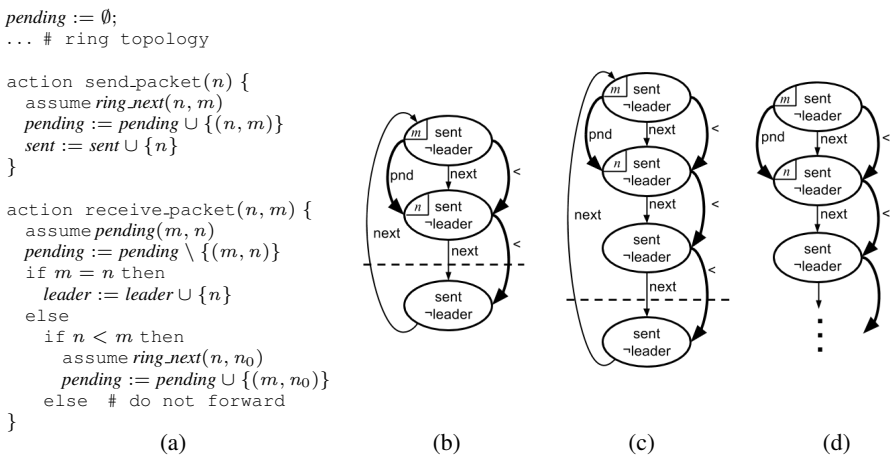

Fig. 2. Leader-election in a ring protocol as an illustration of the use of partial models for incorrect programs and invariants. (a) sketches the protocol (the complete program appears in [2], file `ring_leader_termination.ivy`). (b),(c) show partial models of bound 1 and 2, respectively, and (d) illustrates an infinite structure that explains the root cause of the non-inductiveness.

5 Partial Models for Understanding Non-Inductiveness

When conducting SMT-based deductive verification (e.g., using Dafny [26]), the user constructs both the formal representation of the system and its invariants. In many cases, the invariant I is initially not inductive w.r.t. the given program, due to a bug in the program or in the invariant. Therefore, deductive verification is typically an iterative process in which the user attempts to prove inductiveness, and, when this fails, adapts the program, the invariant, or both.

In such scenarios, it is extremely desirable to present the user with a *counterexample to induction* in the form of a state that satisfies I but makes a transition to a state that violates it. Such a state can be obtained from a model of the formula $\mathit{Ind} = I \wedge \delta \wedge \neg I'$ which is used to check inductiveness. It explains the error, and guides the user towards fixing the program and/or the invariant [26,13]. However, in many cases where the check involves quantifier alternation, current SMT solvers are unable to produce counterexamples. Instead, SMT solvers usually diverge or report "unknown" [15,33]. In such cases, Bounded-Horizon instantiations can be used to present a concrete logical structure which is comprehensible to the user, and is obtained as a model of the (finite) instantiations of the formula *Ind*. While this structure is not a true counterexample (as it is only a model of a subset of the instantiations of the formula), it can still guide the user in the right direction towards fixing the program and/or the invariant.

We illustrate this using a simple leader-election protocol in a ring [10], whose model is presented in Fig. 2(a). The protocol assumes that nodes are organized in a directional ring topology with unique IDs, and elects the node with the highest ID as the leader. Each node sends its own ID to its successor, and forwards messages when they contain an ID higher than its own ID. A node that receives its own ID is elected as leader. We wish to prove a termination property which states that once all nodes have sent their

ID, and there are no pending messages in the network, then there is an elected leader. To verify this we use a relational model of the protocol similar to [29], and specify the property via the following formula:

$$(\exists n.\ \mathit{leader}(n)) \vee (\exists n_1, n_2.\ \neg \mathit{sent}(n_1) \vee \mathit{pending}(n_1, n_2)) \tag{3}$$

A natural attempt of proving this using an inductive invariant is by conjoining Equation (3) (which is not inductive by itself) with the following property (this was the authors' actual next step in proving this termination property):

$$\forall n_1.\ \mathit{sent}(n_1) \wedge \neg \mathit{leader}(n_1) \rightarrow ((\exists n_2.\ \mathit{pending}(n_1, n_2)) \vee (\exists n_2.\ n_1 < n_2)) \tag{4}$$

meaning that if a node has sent its own ID but has not (yet) become leader, then there is either a message pending in the network with the node's ID, or a node with a higher ID.

Alas, the conjunction of Equations (3) and (4) is still not an inductive invariant for the protocol (as we explain below). Since Equation (4) contains $\forall^*\exists^*$ quantification, the associated inductiveness check is outside of the decidable EPR fragment. Indeed, Z3 diverges when it is used to check *Ind*. This is not surprising since the formula has no satisfying finite structures, but has an infinite model (a scenario that is not unusual for $\forall^*\exists^*$ formulas).

On the other hand, applying Bounded-Horizon (with any bound) to *Ind* results in a formula that has finite models. These concrete models are *partial models* of *Ind*. Figs. 2(b) and (c) show partial models (restricted to the pre-states) obtained with bounds of 1 and 2, respectively, on this example.

These models are not true counterexamples to induction: the sub-formula of Equation (4) residing under the universal quantifier does not hold for all the elements of the domain. It does, however, hold for all elements with which the quantifier was instantiated, which are the elements above the dashed line. These elements have all sent their own ID, which was blocked by their successor that has a higher ID, so none of them is the leader. In a finite model, this has to end somewhere, because one of the nodes must have the highest ID. Hence, no finite counter-model exists. However, extrapolating from Fig. 2(b) and (c), we can obtain the infinite model depicted in Fig. 2(d). This model represents an infinite ("open") ring in which each node has a lower ID than its successor. This model is a true model of the formula *Ind* generated by the invariant in Equations (3) and (4), but the fact that it is infinite prevented Z3 from producing it.

Since we use tools that check general (un)satisfiability, which is not limited to finite structures, the only way to prove that an invariant is inductive is to exclude infinite counterexamples to induction as well. Using Bounded-Horizon instantiations, we are able to obtain meaningful partial models that provide hints to the user about what is missing. In this case, the solution is to add an axiom to the system model which states that there is a node with maximal ID: $\exists n_1.\ \forall n_2.\ n_2 \leq n_1$. With this additional assumption, the formula *Ind* is unsatisfiable so the invariant is inductive, and this is proven both by Z3's instantiation heuristics and by Bounded-Horizon with a bound of 1. This illustrates the usefulness of Bounded-Horizon when the invariant is not inductive.

6 Implementation and Initial Evaluation

We implemented a prototype of Bounded-Horizon of bound 1 on top of Z3 [11] and used it within Ivy [29] and the framework of [21]. We applied the procedure to the

incorrect example of Section 5, and successfully verified several correct programs and invariants using bound 1. These examples are (the examples' code can be found in [2]):

- The client-server example of Fig. 1.
- List reverse [21], where the invariant states that the n edges ("next" pointers) are reversed. The invariant is $\forall^*\exists^*$ due to the encoding of n via n^* as explained in [21].
- Learning switch [6], where the invariant states every routing node has a successor.
- Hole-punching firewall [6], where the invariant states that every allowed external node was contacted by some internal node. We explored two modeling alternatives: using a ghost history relation, or existensially quantifying over time.
- Leader election in a ring [10,29] with the invariant discussed in Section 5.

Table 1. Experimental results.

Program	#∀	#Func	#Consts	#$\forall^{\downarrow}$	B1 Total	B1 Solve	Baseline Z3
Client-server	14	1	15	2	58 ms	3 ms	3 ms
List reverse	47	3	15	4	319 ms	211 ms	50 ms
Learning switch	70	1	7	37	245 ms	65 ms	33 ms
Hole-punching firewall with ghost	15	1	18	3	75 ms	4 ms	4 ms
Hole-punching firewall ∃ time	32	2	21	3	102 ms	4 ms	4 ms
Leader-election in a ring (correct)	41	1	21	1	113 ms	36 ms	27 ms
Leader-election in a ring (incorrect)	40	1	20	1	1112 ms	1008 ms	—

B1 Total is the time in milliseconds for the bound 1 implementation. It is compared to **Baseline Z3** which is the solving time in milliseconds of *Ind* as is (with quantifier alternation) by Z3. **B1 Solve** measures the solving time of the formula restricted to bound 1, which demonstrates that most of the overhead occurs when constructing the formula. #∀ is the number of universal quantifiers in *Ind*, **#Func** the number of different Skolem function symbols, and **#Consts** the number of constants. #$\forall^{\downarrow}$ is the number of universally quantified variables that were restricted in the bound 1 check. Measurements were performed on a 3.5GHz Intel i5-4690 CPU with 8GB RAM running Linux 3.13 x86_64.

An initial evaluation of the method's performance appears in Table 1.

Our implementation works by adding "guards" that restrict the range of universal quantifiers to the set of constants where necessary. Technically, recall that we are considering the satisfiability of $\mathit{Ind} = I_S \wedge \delta_S \wedge (\neg I')_S$. Let $\forall x.\ \theta$ be a subformula of *Ind*. If θ contains function symbol applications[6], we transform the subformula to $\forall x.\ \left(\bigvee_c x = c\right) \rightarrow \theta$ where c ranges over const[*Ind*]. The resulting formula is then dispatched to the solver. This is a simple way to encode the termination criterion of bound 1 while leaving room for the solver to perform the necessary instantiations cleverly. The translation enlarges the formula by $O(\#\text{Consts} \cdot \#\forall)$ although the number of bounded instantiations grows exponentially with #∀. The exponential explosion is due to combinations of constants in the instantiation, a problem we defer to the solver.

Z3 terminates on the class of formulas because during the Model-Based Quantifier Instantiation process every instantiation of a universally quantified formula has the same truth value in the model as an instantiation using one of the existing ground terms

[6] This in fact implements the approximation as of Equation (2). The exact bound 1 per Equation (1) can be implemented by a more careful consideration of which universally quantified variables should be restricted, but this was not necessary for our examples.

(constants and then BHT_1 terms). Z3's instantiation engine will produce instantiations using existing terms rather than create superfluous new terms [8].

The results are encouraging because they suggest that the termination strategy of Bounded-Horizon, at least for bound 1, can be combined with existing instantiation techniques to assure termination with only a slight performance penalty. Most encouraging is the satisfiable example of Section 5. On this instance, Z3 was able to return "sat" within seconds, although to do so, in theory, the solver must exhaust the entire set of bounded instantiations. This suggests that the Bounded-Horizon termination criterion might indeed be useful for "sat" instances on which the solver may diverge.

A different approach to the implementation is to integrate the termination criterion of the bound with the solver's heuristics more closely (see [7]).

7 Related Work

Quantifier instantiation. The importance of formulas with quantifier-alternations for program verification has led to many developments in the SMT and theorem-proving communities that aim to allow automated reasoning with quantifier-alternations. The Simplify system [12] promoted the practical usage of quantifier triggers, which let the user affect the quantifier instantiation in a clever way. Similar methods are integrated into modern SMT solvers such as Z3 [11]. Recently, a method for annotating the source code with triggers has been developed for Dafny [27]. The notion of instantiation depth is related to the notions of matching-depth [12] and instantiation-level [14] which are used for prioritization within the trigger-based instantiation procedure.

In addition to user-provided triggers, many automated heuristics for quantifier instantiation have been developed, such as Model-Based Quantifier Instantiation [15]. Even when quantifier instantiation is refutation-complete, it is still important and challenging to handle the SAT cases, which are especially important for program verification. Accordingly, many works (e.g., [33]) consider the problem of model finding.

Local Theory Extensions and Psi-Local Theories [36,19,7] identify settings in which limited quantifier instantiations are complete. They show that completeness is achieved exactly when every partial model can be extended to a (total) model. In such settings Bounded-Horizon instantiations are complete for invariant checking. However, Bounded-Horizon can also be useful when completeness cannot be guaranteed.

Classes of SMT formulas that are decidable by complete instantiations have been studied by [15]. In the uninterpreted fragment, a refined version of Herbrand's Theorem generates a finite set of instantiations when the dependencies are stratified. Bounded-Horizon is a way to bound unstratified dependencies.

Natural Proofs. Natural proofs [30] provide a sound and incomplete proof technique for deductive verification. The key idea is to instantiate recursive definitions over the terms appearing in the program. Bounded-Horizon is motivated by a similar intuition, but focuses on instantiating quantifiers in a way that is appropriate for the EPR setting.

Decidable logics. Different decidable logics can be used to check inductive invariants. For example, Monadic second-order logic [17] obtains decidability by limiting the underlying domain to consist of trees only, and in particular does not allow arbitrary relations, which are useful to describe properties of programs. There are also many decidable fragments of first-order logic [9]. Our work aims to transcend the class of invariants

checkable by a reduction to the decidable logic EPR. We note that the example of Section 5 does not fall under the Loosely-Guarded Fragment of first-order logic [18] due to a use of a transitivity axiom, and does not enjoy the finite-model property.

Abstractions for verification of infinite-state systems. Our work is closely related to abstractions of infinite state systems. These abstractions aim at automatically inferring inductive invariants in a sound way. We are interested in checking if a given invariant is inductive either for automatic and semi-automatic verification.

The View-Abstraction approach [4,5,3] defines a useful abstraction for the verification of parameterized systems. This abstraction is closely related to universally quantified invariants. An extension of this approach [5] adds contexts to the abstraction, which are used to capture $\forall^*\exists^*$ invariants in a restricted setting where nodes have finite-state and are only related by specific topologies. Our work is in line with the need to use $\forall^*\exists^*$ invariants for verification, but applies in a more general setting (with unrestricted high-arity relations) at the cost of losing completeness of invariant checking.

Our work is related to the TVLA system [28,35] which allows the programmers to define instrumentation relations. TVLA also employs finite differencing to infer sound update code for updating instrumentation relations [32], but generates non-EPR formulas and does not guarantee completeness. The focus operation in TVLA implements materialization which resembles quantifier-instantiation. TVLA shows that very few built-in instrumentation relations can be used to verify many different programs.

Instrumentation and update formulas. The idea of using instrumentation relations and generating update formulas is not limited to TVLA and was also used for more predictable SMT verification [24,25].

8 Conclusion

We have provided an initial study of the power of bounded instantiations for tackling quantifier alternation. This paper shows that quantifier instantiation with small bounds can simulate instrumentation. This is a step in order to eliminate the need for instrumenting the program, which can be error-prone. The other direction, i.e. simulating quantifier instantiation with instrumentation, is also possible but is less appealing from a practical point of view, and is presented in the extended version [1].

We are encouraged by our initial experience that shows that various protocols can be proven with small instantiation bounds, and that partial models are useful for understanding the failures of the solver to prove inductiveness. Some of these failures correspond to non-inductive claims, especially those due to infinite counterexamples. In the future we hope to leverage this in effective deductive verification tools, and explore meaningful ways to display infinite counterexamples to the user.

Acknowledgments. We would like to thank Nikolaj Bjørner, Shachar Itzhaky, and Bryan Parno for helpful discussions, and Gilit Zohar-Oren for help and feedback. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no [321174]. This research was partially supported by BSF grant no. 2012259, and by Len Blavatnik and the Blavatnik Family foundation.

References

1. Extended version. http://www.cs.tau.ac.il/research/yotam.feldman/papers/tacas17/tacas17_extended.pdf.
2. Full code materials. http://www.cs.tau.ac.il/research/yotam.feldman/papers/tacas17/examples_code.zip.
3. P. Abdulla, F. Haziza, and L. Holík. Parameterized verification through view abstraction. *International Journal on Software Tools for Technology Transfer*, pages 1–22, 2015.
4. P. A. Abdulla, F. Haziza, and L. Holík. All for the price of few. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 476–495, 2013.
5. P. A. Abdulla, F. Haziza, and L. Holík. Block me if you can! In *International Static Analysis Symposium*, pages 1–17. Springer, 2014.
6. T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky. Vericon: towards verifying controller programs in software-defined networks. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, page 31, 2014.
7. K. Bansal, A. Reynolds, T. King, C. W. Barrett, and T. Wies. Deciding local theory extensions via e-matching. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, pages 87–105, 2015.
8. N. Bjørner. personal communication.
9. E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem*. Springer Science & Business Media, 2001.
10. E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 22(5):281–283, 1979.
11. L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, 2008.
12. D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.
13. C. Flanagan, K. R. M. Leino, M. Lillibridge, G. Nelson, J. B. Saxe, and R. Stata. Extended static checking for java. In *Proceedings of the 2002 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Berlin, Germany, June 17-19, 2002*, pages 234–245, 2002.
14. Y. Ge, C. W. Barrett, and C. Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Ann. Math. Artif. Intell.*, 55(1-2):101–122, 2009.
15. Y. Ge and L. D. Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *International Conference on Computer Aided Verification*, pages 306–320. Springer, 2009.
16. C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. T. V. Setty, and B. Zill. Ironfleet: proving practical distributed systems correct. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP*, pages 1–17, 2015.
17. J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19-20, 1995, Proceedings*, pages 89–110, 1995.
18. I. Hodkinson. Loosely guarded fragment of first-order logic has the finite model property. *Studia Logica*, 70(2):205–240, 2002.
19. C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 265–281, 2008.

20. N. Immerman, A. M. Rabinovich, T. W. Reps, S. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL*, 2004.
21. S. Itzhaky, A. Banerjee, N. Immerman, A. Nanevski, and M. Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *CAV*, volume 8044 of *LNCS*, pages 756–772, 2013.
22. S. Itzhaky, N. Bjørner, T. W. Reps, M. Sagiv, and A. V. Thakur. Property-directed shape analysis. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 35–51, 2014.
23. A. Karbyshev, N. Bjorner, S. Itzhaky, N. Rinetzky, and S. Shoham. Property-directed inference of universal invariants or proving their absence. In *CAV*, 2015.
24. S. K. Lahiri and S. Qadeer. Verifying properties of well-founded linked lists. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11-13, 2006*, pages 115–126, 2006.
25. S. K. Lahiri and S. Qadeer. Back to the future: revisiting precise program verification using SMT solvers. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 171–182, 2008.
26. K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 348–370. Springer, 2010.
27. K. R. M. Leino and C. Pit-Claudel. Trigger selection strategies to stabilize program verifiers. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, pages 361–381, 2016.
28. T. Lev-Ami and S. Sagiv. TVLA: A system for implementing static analyses. In *Static Analysis, 7th International Symposium, SAS 2000, Santa Barbara, CA, USA, June 29 - July 1, 2000, Proceedings*, pages 280–301, 2000.
29. O. Padon, K. L. McMillan, A. Panda, M. Sagiv, and S. Shoham. Ivy: safety verification by interactive generalization. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, pages 614–630, 2016.
30. X. Qiu, P. Garg, A. Stefanescu, and P. Madhusudan. Natural proofs for structure, data, and separation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, pages 231–242, 2013.
31. F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930.
32. T. W. Reps, M. Sagiv, and A. Loginov. Finite differencing of logical formulas for static analysis. *ACM Trans. Program. Lang. Syst.*, 32(6), 2010.
33. A. Reynolds, C. Tinelli, A. Goel, and S. Krstic. Finite model finding in SMT. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 640–655, 2013.
34. A. Reynolds, C. Tinelli, A. Goel, S. Krstic, M. Deters, and C. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, pages 377–391, 2013.
35. S. Sagiv, T. W. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, 2002.
36. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, pages 219–234, 2005.