Logical Methods in Computer Science
Volume 15, Issue 3, 2019, pp. 18:1–18:47
https://lmcs.episciences.org/

Submitted Oct. 26, 2017
Published Aug. 21, 2019

BOUNDED QUANTIFIER INSTANTIATION FOR CHECKING INDUCTIVE INVARIANTS

YOTAM M. Y. FELDMAN [a], ODED PADON [a], NEIL IMMERMAN [b], MOOLY SAGIV [a], AND SHARON SHOHAM [a]

[a] Tel Aviv University, Tel Aviv, Israel
e-mail address: yotam.feldman@gmail.com

[b] 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. Often this is shown by proving the unsatisfiability of a formula of the form $P(V) \wedge \delta(V, V') \wedge \neg P(V')$ (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 two-vocabulary formula expressing the meaning of the command

Key words and phrases: Decidable Logic, Quantifier Instantiation, EPR, Inductive Invariants, Deductive Verification, Decision Procedures.

* A preliminary version of this paper appeared in [FPI+17].

LOGICAL METHODS IN COMPUTER SCIENCE DOI:10.23638/LMCS-15(3:18)2019

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 infinite-state programs, proving the validity of $\{P\}C\{P\}$ is generally undecidable even when C does not include loops. Indeed, existing Satisfiability Modulo Theory (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 [HHK+15] identify this as the main hurdle for using program verification.

The difficulty is rooted in powerful constructs used in the SMT-based verification of interesting programs. Prominent among these constructs are arithmetic and other program operations modelled using background theories, and logical quantifiers. In this paper we target the verification of applications in which the problem can be modelled 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 application domains such as linked-list manipulation [IBI+13], Software-Defined Networks [BBG+14] and some distributed protocols [PMP+16, PLSS17]. Without interpreted theories it remains to address the complications induced by the use of quantifiers, and specifically by the use of alternating universal ($\forall$) and existential ($\exists$) quantifiers.

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 must be explored for correct assertions. A standard form of proofs with quantified formulas is *instantiation*, in which the solver attempts to replace universal quantifiers by a set of ground terms. The problem of exploring the infinite set of candidates for instantiation is sometimes manifested in matching loops [DNS05].
(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 [GM09, RTG+13]. This is somewhat unfortunate since one of the main values of program verification is the early detection of flaws in designs and programs. The possibility of infinite counterexamples is a major complication in this task, as they are especially difficult to find. In uninterpreted domains, infinite counterexamples usually do not indicate a real violation 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 works on EPR-based verification [IBI+13, BBG+14, PMP+16] used universally quantified invariants with programs expressed by $\exists^*\forall^*$ formulas (EPR programs)[1]. In that setting, 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 [PMP+16], but universal invariants are not always sufficient to express the program properties required for verification.

[1] $\exists^*\forall^*$ transition relations can be extracted from code by existing tools for C code manipulating linked lists [IBI+13, IBR+14, KBI+17] and for the modeling language RML [PMP+16] which is Turing-complete.

For example, [HHK$^+$15] describes a client-server scenario with the invariant that "For every reply message sent by the server, there exists a corresponding request message sent by a client". (See Example 3.7 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.

This paper aims to expand the applicability of the EPR-based verification approach to invariants of more complex quantification. We focus on the class of $\forall^*\exists^*$ invariants. $\forall^*\exists^*$ invariants arise in interesting programs, but, as we show, checking inductiveness of invariants in this class is undecidable. We thus study problems (1), (2) above for this setting using the notion of *bounded quantifier instantiation*, a technique 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 the 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 [IBI$^+$13, KBI$^+$17, PMP$^+$16]. Instrumentation tackles a $\forall^*\exists^*$ invariant by transforming the program in a way that allows the invariant to be expressed using just universal quantifiers, 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, meaning 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 existing 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. In this case, when the bounded instantiation procedure terminates, it returns a logical structure which satisfies all the bounded instantiations. This structures is not necessarily a true counterexample but "approximates" one. Interestingly, this capability 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 not correct in general first-order logic. In such cases, state-of-the-art SMT solvers typically produce "unknown" or timeout as they fail to find infinite models. The user is thus 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.

Outline. The rest of the paper is organized as follows: Section 2 provides some technical background and notations. In Section 3 we define the Bounded-Horizon algorithm and discuss its basic properties. Section 4 defines the concept of instrumentation as used in this work, and shows that Bounded-Horizon with a low bound is at least as powerful. Section 5 relates instrumentation to bounded instantiation in the converse direction, showing that other forms of instrumentation can simulate quantifier instantiation of arbitrarily high depth. In Section 6 we show how bounded instantiations can be used to tackle the problem of infinite counterexamples to induction when the verification conditions are not valid. Section 7 describes our implementation of Bounded-Horizon of bound 1, and evaluates its ability to prove some examples correct by bound 1 instantiation while ensuring termination. Section 8 discusses related work, and Section 9 concludes. The discussion of the undecidability of checking inductiveness of $\forall^*\exists^*$ invariants is deferred to Appendix A.

2. Preliminaries

In this section we provide background and explain our notation.

2.1. First-Order Logic.

We use standard relational first-order logic with equality [RS67].

Syntax. A first-order *vocabulary*, denoted Σ, consists of constant symbols c_i, relation symbols r_j, and function symbols f_k. A *relational vocabulary* is a vocabulary without function symbols. In this paper we principally use relational vocabularies, and employ function symbols only when they are generated through Skolemization (see below). Terms and formulas are constructed according to the syntax

$$t ::= c \mid x \mid f(t_1, \ldots, t_n)$$
$$f ::= r(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \neg f \mid f_1 \vee f_2 \mid f_1 \wedge f_2 \mid f_1 \rightarrow f_2 \mid f_1 \leftrightarrow f_2 \mid \forall x.\, f \mid \exists x.\, f$$

where c is a constant symbol, x is a variable, f is an n-ary function symbol and r is an n-ary relation symbol. We always assume terms and formulas are well-formed. FOL(Σ) stands for the set of first-order formulas over Σ. For a formula φ we denote by const[φ] the set of constants that appear in φ. The set of free variables in a term or a formula is defined as usual. A term without free variables is called a *closed term* or *ground term*. A formula without free variables is called a *sentence* or a *closed formula*. We sometimes write $\varphi(x_1, \ldots, x_k)$ (respectively, $t(x_1, \ldots, x_k)$) to indicate that $x_1, \ldots, x_k$ are free in φ (respectively, t). Substitution in φ of terms $t_1, \ldots, t_n$ instead of free variables $x_1, \ldots, x_n$ is denoted by $\varphi[t_1/x_1, \ldots, t_n/x_n]$.

Semantics. Given a vocabulary Σ, a *structure* of Σ is a pair $\sigma = (D, \mathcal{I})$: D is a *domain*, and $\mathcal{I}$ is an *interpretation*. The domain D is a set (of elements). If this set if finite, we say that the structure σ is finite. The interpretation $\mathcal{I}$ maps each symbol in Σ to its meaning in σ: $\mathcal{I}$ associates each k-ary relation symbol r with a relation $\mathcal{I}(r) \subseteq D^k$, and associates each k-ary function symbol f with a function $\mathcal{I}(f) : D^k \rightarrow D$.

We use the standard semantics of first-order logic. Given a vocabulary Σ and a structure $\sigma = (D, \mathcal{I})$, a *valuation* v maps every logical variable x to an element in D. We write $\sigma, v \models \varphi$ to denote that the structure σ and valuation v *satisfy* the formula φ. We write $\sigma \models \varphi$ to mean that $\sigma, v \models \varphi$ for any v, and we will reserve this for whenever φ is a closed formula.

Satisfiability and Validity. We say that a formula φ is *satisfiable* if there are some σ and v such that $\sigma, v \models \varphi$. Otherwise, we say that φ is *unsatisfiable*. We say that a formula φ is valid if $\sigma, v \models \varphi$ for any σ and v. Note that φ is valid if and only if $\neg\varphi$ is unsatisfiable. A formula $\varphi(x_1, \ldots, x_n)$, whose free variables are $x_1, \ldots, x_n$, is valid if and only if the closed formula $\forall x_1, \ldots, x_n.\ \varphi(x_1, \ldots, x_n)$ is valid; it is satisfiable if and only of the closed formula $\exists x_1, \ldots, x_n.\ \varphi(x_1, \ldots, x_n)$ is satisfiable. Two formulas φ and ψ are *equivalent* if $\psi \leftrightarrow \varphi$ is valid. We say that two formulas φ and ψ are *equisatisfiable* to mean that φ is satisfiable if and only if ψ is satisfiable. Note that if two formulas are equivalent then they are also equisatisfiable, but the converse does not necessarily hold.

Syntactical Classes of Formulas. We say that a formula is in *negation normal form* if negation is only applied to its atomic subformulas, namely $r(t_1, \ldots, t_n)$ or $t_1 = t_2$. Every formula can be transformed to an equivalent formula in negation normal form, and the transformation is linear in the size of the formula. For a formula φ we denote by $\mathrm{nnf}(\varphi)$ an equivalent formula in negation normal form obtained by the standard procedure, pushing negation inwards when it is applied on a quantifier or a connective. For example, the negation normal form of $\neg\exists x, y.\ r(x, y) \wedge x \neq y$ is $\forall x, y.\ \neg r(x, y) \vee x = y$.

We say that a formula is *quantifier-free* if it contains no quantifiers. $\mathrm{QF}(\Sigma)$ denotes the set of quantifier-free formulas over Σ. We say that a formula is in *prenex normal form* (PNF) if it is of the form $Q_1 . \cdots Q_n . \psi$ where ψ is quantifier-free, and each Q_i is either $\forall x$ or $\exists x$ for some variable x. Every formula can be transformed to an equivalent formula in prenex normal form, and the transformation is linear in the size of the formula. For example, the prenex normal form of $\forall x.\ r(x) \rightarrow \exists y.\ p(x, y)$ is $\forall x.\ \exists y.\ r(x) \rightarrow p(x, y)$.

We say a formula is *universally quantified* or *universal*, if it is in prenex normal form and has only universal quantifiers. $\forall^*(\Sigma)$ denotes the set of universal formulas over Σ. An *existentially quantified* or *existential* formula is similarly defined, and $\exists^*(\Sigma)$ denotes the set of existential formulas over Σ. Whenever an existential quantifier is in the scope of a universal quantifier or vice versa, we call this a *quantifier alternation*. A formula is *alternation-free* if it contains no quantifier alternations; namely, if it is a Boolean combination of universal and existential formulas. The set of alternation-free formulas is denoted $\mathrm{AF}(\Sigma)$. With quantifier alternations present, we denote by $\exists^*\forall^*(\Sigma)$ the set of formulas over Σ in prenex normal form where all the existential quantifiers appear before the universal ones, and $\forall^*\exists^*(\Sigma)$ for the case where all universal quantifiers appear before the existentials (in both cases this is a single quantifier alternation).

EPR. The effectively-propositional (EPR) fragment of first-order logic, also known as the Bernays-Schönfinkel-Ramsey class, consists of $\exists^*\forall^*(\Sigma)$ sentences, which we denote by $\mathrm{EPR}(\Sigma)$. Such sentences enjoy the *small model property*; in fact, a satisfiable EPR sentence has a model of size no larger than the number of its constants plus existential quantifiers. Thus satisfiability of EPR sentences is decidable [Ram30].

Skolemization. Let $\varphi(z_1, \ldots, z_n) \in \mathrm{FOL}(\Sigma)$. 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 $\varphi_S \to \varphi$ is valid and that φ_S, φ are equi-satisfiable.

2.2. Transition Systems and Inductive Invariants.

Transition Relation. A *transition relation* is a sentence δ over a vocabulary $\Sigma \uplus \Sigma'$ where Σ is a relational vocabulary used to describe the source (or pre-) state of a transition and $\Sigma' = \{a' \mid a \in \Sigma\}$ is used to describe the target (or post-) state.

Inductive Invariants. A first-order sentence I over Σ is an *inductive invariant* for δ if $I \wedge \delta \to I'$ is valid, or, equivalently, if $I \wedge \delta \wedge \neg I'$ is unsatisfiable[2], where I' results from substituting every constant and relation symbol in I by its primed version (i.e. $I' \in \mathrm{FOL}(\Sigma')$). Candidate invariants I are always sentences (i.e. ground).

Remark 2.1. Often in the literature an inductive invariant is considered with respect to a set of initial states of the system φ_0 and a safety property φ_P, requiring from an inductive invariant I also that $\varphi_0 \to I$ and $I \to \varphi_P$ are valid. We refer to this setting in Appendix A. Elsewhere in the paper we focus on checking the validity of $I \wedge \delta \to I'$ which is typically the difficult part: when $\varphi_0 \in \exists^*\forall^*(\Sigma)$, if $I \in \forall^*\exists^*(\Sigma)$ then the validity of $\varphi_0 \to I$ is decidable (Section 2.1), and φ_P is usually part of I. Furthermore, checking simply the inductiveness of $(\varphi_0 \vee I) \wedge \varphi_P$ suffices to establish safety.

Counterexample to Induction. Given a first-order sentence I over Σ and transition relation δ (over $\Sigma \uplus \Sigma'$), a *counterexample to induction* is a structure $\mathcal{A}$ (over $\Sigma \uplus \Sigma'$) s.t. $\mathcal{A} \models I \wedge \delta \wedge \neg I'$.

EPR Transition Relation. In this paper we focus on transition relations that are EPR sentences. Namely, 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 (or pre-) state of a transition and $\Sigma' = \{a' \mid a \in \Sigma\}$ is used to describe the target (or post-) state.

2.3. RML: Relational Modeling Language with Effectively Propositional Logic.

We now review a simple imperative modeling language, a variant of the *relational modeling language* (RML) [PMP+16, Pad18], and its translation to EPR transition relations.

[2] In this paper, unless otherwise stated, satisfiability and validity refer to general models and are not restricted to finite models. Note that for EPR formulas, finite satisfiability and general satisfiability coincide.

$$
\begin{array}{rcll}
\langle rml \rangle & ::= & \langle decls \rangle \ ; \ \langle actions \rangle & \\
\langle decls \rangle & ::= & \epsilon \mid \langle decls \rangle ; \langle decls \rangle & \\
& \mid & \texttt{relation}\ \mathbf{r} & \\
& \mid & \texttt{variable}\ \mathbf{v} & \\
& \mid & \texttt{axiom}\ \varphi_{\texttt{EA}} & \\
& \mid & \texttt{init}\ \varphi & \\
\langle actions \rangle & ::= & \epsilon \mid \langle actions \rangle \ ; \ \langle actions \rangle & \\
& \mid & \textbf{action}\ \texttt{action}\ \{\ \langle cmd \rangle\ \} & \\
\langle cmd \rangle & ::= & \texttt{skip} & \text{do nothing} \\
& \mid & \texttt{abort} & \text{terminate-abnormally} \\
& \mid & \mathbf{r}\,(\overline{x}) := \varphi_{\texttt{QF}}\,(\overline{x}) & \text{quantifier-free update of relation } \mathbf{r} \\
& \mid & \mathbf{v} := * & \text{havoc of variable } \mathbf{v} \\
& \mid & \texttt{assume}\ \varphi_{\texttt{EA}} & \text{assume } \exists^*\forall^* \text{ formula holds} \\
& \mid & \langle cmd \rangle \ ; \ \langle cmd \rangle & \text{sequential composition} \\
& \mid & \langle cmd \rangle \mid \langle cmd \rangle & \text{non-deterministic choice}
\end{array}
$$

FIGURE 1. Syntax of RML. $\mathbf{r}$ denotes a relation identifier. $\mathbf{v}$ denotes a variable identifier. `action` denotes an action identifier. $\overline{x}$ denotes a vector of logical variables. $\varphi_{\texttt{QF}}\,(\overline{x})$ denotes a quantifier-free formula with free logical variables $\overline{x}$. $\varphi_{\texttt{EA}}$ denotes a closed formula with quantifier prefix $\exists^*\forall^*$. The syntax of terms and formulas is of first-order logic.

2.3.1. *RML Syntax and Informal Semantics.* Figure 1 shows the abstract syntax of RML. RML imposes two main programming limitations:

(1) the only data structures are uninterpreted relations,
(2) program conditions and update formulas have restricted quantifier structure.

An RML program is composed of a set of actions. Each action consists of loop-free code, and a transition of the RML program corresponds to (non-deterministically) selecting an action and executing its code atomically. Thus, an RML program can be understood as a single loop, where the loop body is a non-deterministic choice between all the actions. The restriction of each action to loop-free code simplifies the presentation, and it does not reduce RML's expressive power, as nested loops can always be converted to a flat loop.

Declarations and states. The declarations of an RML program define a set of relations $\mathcal{R}$, a set of program variables $\mathcal{V}$, and a set of axioms $\mathcal{A}$ in the form of (closed) $\exists^*\forall^*$-formulas.

A state of an RML program is a first-order structure over the vocabulary that contains a relation symbol for every relation in $\mathcal{R}$, and a constant symbol for every variable in $\mathcal{V}$, such that all axioms are satisfied. The values of program variables and relations are all mutable by the program.

Actions. An RML program is composed of a set of actions. Each action consists of a name and a loop-free body, given by an RML command. The transition relation formula of the whole program will be given by the disjunction of the transition relation formulas associated with each action of the program. Effectively, this means that each transition is a non-deterministic choice between all the actions of the program, and that each action

Syntactic Sugar	**Desugared RML**
local v := $*$	**v** is syntactically declared inside the current scope **v** := $*$

action `action` $(\mathbf{v}_1, \ldots, \mathbf{v}_n)\,\{C\}$	**action** `action` {
	  local $\mathbf{v}_1 := *$;
	  ...
	  local $\mathbf{v}_n := *$;
	  C
	}

`if` φ C_1 `if` φ C_1 `else` C_2	{`assume` φ ; C_1} | {`assume` $\neg\varphi$} {`assume` φ ; C_1} | {`assume` $\neg\varphi$; C_2}
$\mathbf{r}$.`insert` $(\overline{y} \mid \varphi_{QF}(\overline{y}))$ $\mathbf{r}$.`insert` $(\overline{g})$	$\mathbf{r}(\overline{x}) := \mathbf{r}(\overline{x}) \vee (\overline{x} = \overline{y} \wedge \varphi_{QF}(\overline{y}))$ $\mathbf{r}$.`insert` $(\overline{y} \mid \overline{y} = \overline{g})$

FIGURE 2. Syntactic sugars for RML. In addition to using the notations of Figure 1, g denotes a ground term, $\overline{y}$ denotes a tuple of terms where each y_i is x_i or a ground term, $\overline{g}$ denotes a tuple of ground terms, and equality and between tuples denotes the conjunction of the component-wise equalities.

is executed atomically. Below we given an intuitive description of RML commands, and Section 2.3.2 presents their axiomatic semantics and explains how to translate a command C to its associated transition relation formula $\delta[C]$.

Commands. Each command investigates and potentially updates the state of the program. The semantics of `skip` and `abort` are standard. The command $\mathbf{r}(x_1, \ldots, x_n) := \varphi_{\mathtt{QF}}(x_1, \ldots, x_n)$ is used to update the n-ary relation $\mathbf{r}$ to the set of all n-tuples that satisfy the quantifier-free formula $\varphi_{\mathtt{QF}}$. For example, $\mathbf{r}(x_1, x_2) := (x_1 = x_2)$ updates the binary relation $\mathbf{r}$ to the identity relation; $\mathbf{r}(x_1, x_2) := \mathbf{r}(x_2, x_1)$ updates $\mathbf{r}$ to its inverse relation; $\mathbf{r}_1(x) := \mathbf{r}_2(x, \mathbf{v})$ updates $\mathbf{r}_1$ to the set of all elements that are related by $\mathbf{r}_2$ to the current value (interpretation) of program variable $\mathbf{v}$.

The havoc command $\mathbf{v} := *$ performs a non-deterministic assignment to $\mathbf{v}$. The `assume` command is used to restrict the executions of the program to those that satisfy the given (closed) $\exists^*\forall^*$-formula. Sequential composition and non-deterministic choice are defined in the usual way.

The commands given in Figure 1 are the core of RML. Figure 2 provides several useful syntactic sugars for RML which we use in the examples in this paper, including an `if-then-else` command and convenient update commands for relations and functions.

Turing-completeness. To see that RML is Turing-complete, we can encode a (Minsky) counter machine in RML. Each counter c_i can be encoded with a unary relation r_i. The value of counter c_i is the number of elements in r_i. Testing for zero, incrementing, and decrementing counters can all be easily expressed by RML commands.

$$
\begin{aligned}
wp\left(\texttt{skip}, Q\right) &= Q \\
wp\left(\texttt{abort}, Q\right) &= \mathit{false} \\
wp\left(\mathbf{r}\left(\overline{x}\right) := \varphi_{\texttt{QF}}\left(\overline{x}\right), Q\right) &= \left(\mathcal{A} \rightarrow Q\right)\left[\varphi_{\texttt{QF}}\left(\overline{s}\right) \ / \ \mathbf{r}\left(\overline{s}\right)\right] \\
wp\left(\mathbf{v} := *, Q\right) &= \forall x.\left(\mathcal{A} \rightarrow Q\right)\left[x \ / \ \mathbf{v}\right] \\
wp\left(\texttt{assume}\ \varphi_{\texttt{EA}}, Q\right) &= \varphi_{\texttt{EA}} \rightarrow Q \\
wp\left(C_1 \ ; \ C_2, Q\right) &= wp\left(C_1, wp\left(C_2, Q\right)\right) \\
wp\left(C_1 \mid C_2, Q\right) &= wp\left(C_1, Q\right) \wedge wp\left(C_2, Q\right)
\end{aligned}
$$

FIGURE 3. Rules for wp for RML. $\overline{s}$ denotes a vector of terms.

2.3.2. *Axiomatic Semantics.* We now provide a formal semantics for RML by defining a weakest precondition operator for RML commands with respect to assertions expressed in first-order logic, which also allows us to define the transition relation formula of an RML command. We start with a formal definition of program states as structures, and program assertions as formulas in first-order logic.

States. Recall that an RML program declares a set of program variables $\mathcal{V}$ and relations $\mathcal{R}$. We define a first-order vocabulary Σ, that contains a relation symbol for every relation in $\mathcal{R}$, and a constant symbol for every variable in $\mathcal{V}$. A state of the program is given by a first-order structure over Σ. The states of an RML program are structures of Σ that satisfy all the axioms $\mathcal{A}$ declared by the program.

Assertions. Assertions on program states are specified by sentences in first-order logic over Σ. A state satisfies an assertion if it satisfies it in the usual semantics of first-order logic.

Remark 2.2. Note that program variables, modeled as constant symbols in Σ, should not be confused with logical variables used in first-order formulas.

Weakest precondition of RML commands. Figure 3 presents the definition of a *weakest precondition* operator for RML, denoted wp. The weakest precondition [Dij76] of a command C with respect to an assertion Q, denoted $wp(C, Q)$, is an assertion Q' such that every execution of C starting from a state that satisfies Q' leads to a state that satisfies Q. Further, $wp(C, Q)$ is the weakest such assertion. Namely, $Q' \Rightarrow wp(C, Q)$ for every Q' as above.

The rule for wp of `skip` is standard, as are the rules for `abort`, `assume`, sequential composition and non-deterministic choice. The rules for updates of relations and functions and for havoc are instances of Hoare's assignment rule [Hoa69], applied to the setting of RML and adjusted for the fact that state mutations are restricted by the axioms $\mathcal{A}$.

Transition relation formulas of RML commands. The weakest precondition operator is closely related to transition relation formulas. Recall that a transition relation formula has vocabulary $\Sigma \uplus \Sigma'$, where the primed symbols represent the state after executing the command. Here, we use the connection between transition relations and the weakest precondition (pointed out by [Dij82, Nel89]), to define the transition relation of a command C, denoted by $\delta[C]$, as follows:

$$\delta[C] = \neg wp\left(C, \neg \psi_{\Sigma = \Sigma'}\right)$$

where

$$\psi_{\Sigma=\Sigma'} = \bigwedge_{r\in\Sigma} \forall \overline{x}.\ r(\overline{x}) \leftrightarrow r'(\overline{x}) \ \wedge \ \bigwedge_{c\in\Sigma} c = c'$$

This makes a slight abuse of the definition of wp, since it applies wp to a formula over $\Sigma \uplus \Sigma'$. However, in this context, the symbols in Σ' can be treated as additional auxiliary symbols, without special meaning.

Intuitively, there is a transition from s to s' if and only if s does not satisfy the weakest precondition of "not being s'". This is captured by the above connection, and "not being s'" is captured by $\neg\psi_{\Sigma=\Sigma'}$. Note further that non-deterministic choice between commands results in a conjunction in the weakest precondition, and a disjunction in the transition relation. Similarly, a havoc command results in a universal quantifier in the weakest precondition, and an existential quantifier in the transition relation.

The transition relation of the entire RML program δ is given by the disjunction of the transition relations of each action in the RML program, where the transition relation of each action is computed from its body via the above definition. Formally, if the bodies of actions are $C_1, , \ldots, , C_k$ then the transition relation of the whole program δ is given by:

$$\delta = \mathcal{A} \wedge \bigvee_i \delta[C_i]$$

Note that when using the syntactic sugar of Figure 2 to define actions with parameters, these parameters are existentially quantified in the transition relation (through the negation of the weakest-precondition rule of a havoc command).

RML Produces EPR Transition Relations. RML is designed so that the transition relations associated with RML programs are EPR transition relations. This is formalized in the following claims:

Lemma 2.3. *Let C be an RML command. If $Q \in \forall^*\exists^*(\Sigma)$-formula, then so is the prenex normal form of $wp(C, Q)$.*

Proof. Straightforward from the rules of Figure 3, the fact that all assignments to relations use quantifier-free formulas, and all `assume` commands and axioms are of formulas with $\exists^*\forall^*$ prenex normal form. □

Corollary 2.4. The transition relation δ of an RML program is an EPR transition relation.

Proof. Follows from Lemma 2.3, the construction of $\delta[C]$ as the negation of a weakest-precondition, and the transition relation of the entire program δ as the disjunction of transition relations of individual actions. □

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.

Undecidability. We first justify the use of sound but incomplete algorithms, such as the Bounded-Horizon algorithm, for the problem of checking inductiveness of $\forall^*\exists^*$ formulas. For a universal sentence $I \in \forall^*(\Sigma)$, the sentence $I \wedge \delta \wedge \neg I'$ is in EPR (recall that δ is specified in EPR). Hence, checking inductiveness amounts to checking the 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. In Appendix A we show that checking inductiveness of $\forall^*\exists^*$ formulas is indeed undecidable, even when the transition relation is restricted to EPR (see Theorem A.3). Thus, an attempt to check inductiveness must sacrifice either soundness, completeness, or termination. Techniques based on quantifier instantiation usually prefer completeness over termination. In contrast, the Bounded-Horizon algorithm guarantees termination a-priori, possibly at the expense of completeness (but is surprisingly powerful nonetheless). We now move to define the Bounded-Horizon algorithm for checking invariants with quantifier alternation and discuss its basic properties in checking inductiveness.

Bounded-Horizon Instantiations. Let $\delta \in \exists^*\forall^*(\Sigma, \Sigma')$ be an EPR transition relation and $I \in \mathrm{FOL}(\Sigma)$ a candidate invariant. We would like to check the satisfiability of $I \wedge \delta \wedge \neg I'$, and equivalently of $Ind = I_S \wedge \delta_S \wedge (\neg I')_S$. 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. (δ is an EPR sentence and so its Skolemization adds only constants.) 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. We then check if this (finite) set of instantiations is unsatisfiable; if it is already unsatisfiable then we have a proof that I is inductive. Otherwise we report that I is not known to be inductive. The idea is to choose a (preferably small) number k and perform instantiations bounded by k instead of full-blown instantiation. As we will show, this algorithm is sound but not necessarily complete for a given k.

Below we provide the formal definitions. 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 3.1 (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 $\overline{t}$ of $n+m$ ground terms, denoted by $\varphi[\overline{t}\,]$, is obtained by substituting $\overline{t}$ 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 3.2 (Herbrand's Theorem)**.** *Let $\varphi \in \forall^*(\tilde{\Sigma})$. Then φ is satisfiable iff the (potentially infinite) set $\left\{\varphi[\overline{t}\,] \mid \overline{t} \text{ is a tuple of ground terms over } \tilde{\Sigma}\right\}$ is satisfiable.*

Remark 3.3. Herbrand's theorem provides a simple proof of the decidability of the satisfiability of EPR sentences: Let $\varphi \in \mathrm{EPR}(\Sigma)$. Then its Skolemization φ_S may introduce constant symbols but not function symbols (since there is no $\forall^*\exists^*$ quantification). Function symbols are not present in the vocabulary, so the set of possible instantiations is finite. From Herbrand's theorem, it suffices to check the satisfiability of the finite set of quantifier-free sentences $\{\varphi_S[\overline{c}\,]\}$, which is decidable.

While EPR sentences introduce only instantiations on constant symbols (bound 0 when considering the bound of function applications), arbitrary sentences may introduce instantiations of unbounded depths. We now turn to restrict the depth of terms used in instantiations.

Definition 3.4 (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, as follows. 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*

$$\mathrm{BHT}_0 = C$$
$$\mathrm{BHT}_k = \mathrm{BHT}_{k-1} \cup \{f(t_1, \ldots t_m) \mid f \in F,\ m = Arity_f,\ t_1, \ldots, t_m \in \mathrm{BHT}_{k-1}\}.$$

We will also write $\bar{t} \in \mathrm{BHT}_k$ 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 $\mathrm{BHT}_\infty = \bigcup_{k \in \mathbb{N}} \mathrm{BHT}_k$.

Definition 3.5 (Depth of Instantiation)**.** *Let $\varphi \in \forall^*(\tilde{\Sigma})$ and $\bar{t} \in \mathrm{BHT}_\infty$. 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*.*

We are now ready to define the algorithm and discuss its basic soundness and completeness properties.

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

$$\left\{ Ind[\bar{t}\,] \mid \bar{t} \in \mathrm{BHT}_k,\ \mathrm{depth}(Ind[\bar{t}\,]) \leq k \right\} \tag{3.1}$$

is unsatisfiable. If it is unsatisfiable, 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 instantiation is a ground quantifier-free formula.

Bounded-Horizon for $\forall^*\exists^*$ Invariants. We illustrate the definition of Bounded-Horizon in the case that $I \in \forall^*\exists^*(\Sigma)$. Let $I = \forall \overline{x}.\ \exists \overline{y}.\ \alpha(\overline{x}, \overline{y})$ where $\alpha \in \mathrm{QF}$. 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 $(\neg I')_S$. The Bounded-Horizon check of bound k can be approximately[3] understood as checking the (un)satisfiability of

$$\Big(\bigwedge_{\bar{t} \in \mathrm{BHT}_{k-1}} I_S[\bar{t}\,] \Big) \wedge \Big(\bigwedge_{\bar{t} \in \mathrm{BHT}_k} \delta_S[\bar{t}\,] \Big) \wedge \Big(\bigwedge_{\bar{t} \in \mathrm{BHT}_k} (\neg I')_S[\bar{t}\,] \Big). \tag{3.2}$$

The Bounded-Horizon algorithm is sound for all $I \in \mathrm{FOL}(\Sigma)$, as formalized in the next lemma:

[3] Equation (3.2) is an under-approximation of the set of instantiations used for bound k; variables that do not appear in I_S under a function symbol can be taken from BHT_k in the conjunction without increasing the total depth of instantiation beyond k, and are therefore allowed in bounded instantiation of bound k. This approximation is illustrative nonetheless, and will be useful in the proofs in Section 4.

```
relation req(u, q)
relation resp(u, p)
relation match(q, p)

action new_request(u) {
  local q := *; # new request
  assume ∀w, s. ¬req(w, q) ∧
                ¬match(q, s);
  req.insert ((u, q))
  /@   r.insert ((u, y) | match(q, y))
}
action respond(u, q) {
  assume req(u, q);
  local p := *; # new response
  assume ∀w, x. ¬req(w, p) ∧
                ¬match(x, p);
  match.insert ((q, p)) ;
  /@   r.insert ((x, p) | req(x, q))
  resp.insert ((u, p))
}
```

```
init ∀u, q. ¬req(u, q)
init ∀u, p. ¬resp(u, p)
init ∀u, p. ¬match(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)
```

FIGURE 4. Example demonstrating a $\forall^*\exists^*$ invariant that is provable with bound 1. The reader should first ignore the instrumentation code denoted by /@ (see Example 4.1). This example models a simple client-server scenario, with the safety property that every response sent by the server was triggered by a request from a client. Verification of this example requires a $\forall^*\exists^*$ invariant. This example is inspired by [HHK+15]. The complete program is provided in [add] (files `client_server_ae.ivy`, `client_server_instr.ivy`).

Lemma 3.6 (Soundness)**.** *For every $k \in \mathbb{N}$, Bounded-Horizon with bound k is sound, i.e., if Bounded-Horizon of bound k 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 $\mathcal{A} \models \mathit{Ind}[\,\bar{t}\,]$ for every $\bar{t} \in \mathrm{BHT}_\infty$ and in particular for every $\bar{t} \in \mathrm{BHT}_k$ such that $\mathrm{depth}(\mathit{Ind}[\,\bar{t}\,]) \leq k$. Hence, Bounded-Horizon of bound k will not report that I is inductive. □

As the algorithm is sound for any k, the crucial question that remains is an appropriate choice of k. A small k is preferable for efficiency, but a larger k could allow for proving more invariants. In the following example, a bound of even 1 suffices for proving that the invariant is inductive. We then show that for every correct invariant there is a suitable bound k, but a single choice of k cannot prove all correct invariants. Later, in Section 4, we show that bound of 1 or 2 is surprisingly powerful nonetheless.

Example 3.7. Example 3.7 presents a simple model of the client-server scenario described in [HHK+15]. The program induces an EPR transition relation, and its invariant is provable by Bounded-Horizon of bound 1.

We first explain this example while 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

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

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 this fact is provable by Bounded Horizon of bound 1.

Lemma 3.8 (Completeness for some k)**.** *For every $I \in \mathrm{FOL}(\Sigma)$ and δ such that I is inductive w.r.t. δ there exists a finite $k \in \mathbb{N}$ s.t. I is provably inductive w.r.t. δ with Bounded-Horizon of bound k.*

Proof. From Theorem 3.2 and compactness there is a finite unsatisfiable set S of instantiations. Take k to be the maximal depth of the 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 (see Appendix A), Bounded-Horizon is *not* complete for a given k for arbitrary invariants.

Example 3.9. An example of a program and an inductive invariant for which a bound of 0 or 1 is insufficient appears in Example 3.9. In this example the server operates as a middleman between clients and the database (DB), and is used to anonymize user requests before they reach the database. The server performs a translation p between clients' identity and an anonymous unique id, sends a translated request to the DB, and forwards the DB's response to the clients. The safety property is that every response sent by the server was triggered by a request from a client. The inductive invariant states, in addition to the safety property, that every server request to the DB was triggered by a client's request from the server, and that every DB response was triggered by a server's request. Proving that the invariant is inductive under the action `server_recv_db_response` requires the prover to understand that for the response from the DB there is a matching request from the server to the DB, and that for this request to the DB there is a matching request from the client to the server. Every such translation requires another level of nesting in the instantiation. In this example, a bound of 2 manages to prove inductiveness. This example can be lifted to require an even larger depth of instantiation by adding more translation entities similar to the server, and describing the invariant in a similar, modular, way.

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

```
relation req(u, q)
relation resp(u, p)
relation db_req(id, p)
relation db_resp(id, p)
relation t(id, u)

action new_request(u) {
  local q := *; # new request
  assume ∀w, j. ¬req(w, q) ∧
                ¬db_req(q, j);
  req.insert ((u, q))
}
action db_recv_request(id, q) {
  assume db_req(id, q);
  local p := *;
  assume db(q, p);
  db_resp.insert ((id, p))
}
action check(u, p) {
  assume resp(u, p);
  if ∀q. req(u, q) → ¬db(q, p)
    then abort
}

init ∀u, q. ¬req(u, q)
init ∀u, p. ¬resp(u, p)
init ∀id, p. ¬db_req(id, p)
init ∀id, p. ¬db_resp(id, p)
init ∀id, u. ¬t(id, u)

action server_recv_request(u, q) {
  assume req(u, q);
  local id := *; # new DB request id
  assume ∀w. ¬t(id, w);
  t.insert ((id, u)) ;
  db_req.insert ((id, q))
}


action server_recv_db_response(id, p) {
  assume db_resp(id, p);
  resp.insert ((x, p) | t(id, x))
}
```

$$\begin{aligned}\text{Invariant } I = {} & \forall u, p.\ resp(u, p) \rightarrow \exists q.\ req(u, q) \wedge db(q, p) \wedge \\ & \forall id, q.\ db\_req(id, q) \rightarrow \exists u.\ t(id, u) \wedge req(u, q) \wedge \\ & \forall id, p.\ db\_resp(id, p) \rightarrow \exists q.\ db\_req(id, q) \wedge db(q, p) \wedge \\ & \forall id, u_1, u_2.\ t(id, u_1) \wedge t(id, u_2) \rightarrow u_1 = u_2\end{aligned}$$

FIGURE 5. Example demonstrating a $\forall^*\exists^*$ invariant that is provable only with bound 2. The server anonymizes requests from clients to the database (DB) and forwards the answer to the client. The server performs a translation t between clients' identity and an anonymous unique id. The safety property is that every response sent by the server to a client was triggered by a request from the client. The inductive invariant further states that every server request to the DB was triggered by a client's request from the server, and that every DB response was triggered by a server's request. The complete program corresponding to this Figure appears in [add] (file `client_server_db_ae.ivy`).

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 modification to result from the fact that ψ holds for the elements of the domain that are directly involved in the transition. If this is indeed the reason that ψ is maintained, a bound of 1 sufficiently utilizes ψ in the pre-state to prove the invariant in the post-state, i.e. to prove that it is inductive.

This is the case in Example 3.7. Additional examples are listed in Section 7. The example of Example 3.9 itself also admits a different invariant that is provable by bound 1. Section 4 further studies the power of Bounded-Horizon with a low bound.

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 7 demonstrates the method's power on several interesting programs that we verified using Bounded-Horizon of bound 1.

4.1. Instrumentation.

In this section we present our view of an instrumentation procedure, a form of which was used in previous works [IBI$^+$13, KBI$^+$17, PMP$^+$16], aiming to eliminate the need for quantifier-alternation, thus reducing the verification task to a decidable fragment. Generally speaking, instrumentation begins with a program that induces an EPR 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) in a sound way. Instrumentation is generally a manual procedure. We now describe the instrumentation procedure used in previous works [IBI$^+$13, KBI$^+$17, PMP$^+$16], but stress that the results of this paper do not depend on this specific recipe but on the semantic soundness condition below (Definition 4.2). This instrumentation procedure is also thoroughly described in a recent work [PLSS17].

The instrumentation procedure used previously [IBI$^+$13, KBI$^+$17, PMP$^+$16] consists of the following three steps:

(1) Identify a formula $\psi(\overline{x}) \in \mathrm{FOL}(\Sigma)$ (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.
(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 [RSL10].
(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 4.1. In the example of Example 3.7, 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 $\widehat{I} = \forall u, p.\ resp(u, p) \rightarrow r(u, p)$, 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 4.2 (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* $\psi \in \mathrm{FOL}(\Sigma)$ *if*

$$\delta \rightarrow \widehat{\delta}[\psi/r, \psi'/r']$$

is valid.

Definition 4.2 requires 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 subtle point in instrumentation as opposed to, e.g., ghost code, is that instrumentation may affect the executions, for example by changing conditions in the code. Soundness ensures that no executions are omitted.

Example 4.3. In the example of Example 3.7, the instrumentation described in Example 4.1 is a sound instrumentation, where the transition relation of the original program δ and that of the instrumented program $\widehat{\delta}$ are produced from the example's code as in Section 2.3.2. To see that $\widehat{\delta}$ forms a sound instrumentation for δ and $\psi(x, y) = \exists z.\ req(x, z) \wedge match(z, y)$, consider a transition of δ. This induces a transition of $\widehat{\delta}$ by interpreting the instrumentation relation r according to ψ: the update code of r in the instrumented program translates to a restriction in $\widehat{\delta}$ relating r, r', which holds since the code updates r according to its meaning as ψ. Furthermore, if the transition of δ is of the action `check` and the condition of the `if` statement holds, then $\widehat{\delta}[\psi/r, \psi'/r']$ allows the matching transition—of that same action, with the same action parameters (formally, a valuation of the existential quantifiers in $\widehat{\delta}$ as the valuation of the existentials for the action paramters in δ). This is due to the fact that the rewritten `if` statement in the instrumented program is equivalent to the original when interpreting r as ψ.

Remark 4.4. Note that the definition of sound instrumentation ensures that r is updated in a way that is consistent with its interpretation as ψ. To see this, note that in the expression $\widehat{\delta}[\psi/r, \psi'/r']$ the update code of r in $\widehat{\delta}$ becomes a constraint over the core relations in Σ. In a sound instrumentation this constraint is required to follow from the way the core relations are updated by δ, essentially implying that the update code is correct.

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 4.5. *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 4.2) $I \wedge \delta \rightarrow I'$ is valid. □

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

Example 4.6. In the example of Example 3.7, the instrumented program has the inductive invariant $\widehat{I} = \forall u, p.\ resp(u, p) \rightarrow r(u, p)$, which is universally quantified. Substituting ψ instead of r we get an inductive invariant of the original program, $I = \forall u, p.\ resp(u, p) \rightarrow \exists q.\ req(u, q) \wedge match(q, p)$. This invariant is $\forall^*\exists^*$.

Remark 4.7. In this paper we focus on instrumentation by a derived relation. It is also possible to consider instrumentations by a *constant*, as used for example in handling the alternation-free invariant of the shared-tail example in [KBI+17]. Instrumentation by a constant can be emulated with an instrumentation by a derived relation, conforming to the results of this paper. This is performed by adding a unary relation $c(x)$ representing the constant c, and adding to the invariant a clause stating that $c(x)$ holds for exactly one element. The resultant invariant is alternation-free, and thus Theorem 4.16 can account for the power of bounded instantiations in this case.

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. To formally define this notion (Definition 4.9) we first define *existential naming* to relate the existential quantifiers of two transition relations:

Definition 4.8 (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* $\eta : \{z_1, \ldots, z_m\} \rightarrow const[\delta_S] \cup const[\widehat{\delta}]$. *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 for $\widehat{\delta}$ which uses existing constants rather than fresh ones. The existential naming fixes existentially quantified variables to constants of $\widehat{\delta}$, constants of δ, or existential quantifiers of δ (manifested as constants from δ_S). Without further requirements, such a mapping η always exists. However, we are interested in mappings such that $\eta(\widehat{\delta})$ is a sound over-approximation of δ, despite fixing the existential quantifiers of $\widehat{\delta}$ according to η (note that fixing the quantifiers makes the formula stronger, or, when viewing it as an over-approximation of δ, tighter). Intuitively, this means that η fixes the existential quantifiers in a sound way. When such a mapping exists, we refer to the corresponding instrumentation as instrumentation without additional existentials:

Definition 4.9 (Instrumentation Without Additional Existentials)**.** $\widehat{\delta}$ *is a* sound instrumentation without additional existentials *for* δ *and* ψ *if there exists an existential naming* η *such that*

$$\delta_S \rightarrow \eta(\widehat{\delta})[\psi/r, \psi'/r']$$

is valid.

Definition 4.9 ensures that the existential quantifiers in $\widehat{\delta}$ have counterparts in (the Skolemized) δ, which are identified by η, and suffice to establish soundness of $\widehat{\delta}$.

Example 4.10. The instrumentation in Example 3.7 results in $\widehat{\delta}$ whose soundness can be established with an existential naming w.r.t. the original δ. The existnential naming is as

follows: existential quantifiers in $\widehat{\delta}$ result (per the procedure in Section 2.3.2) only from action parameters and the havoc statements (see Figure 2).[4] The existential naming maps these to the *same* (Skolemized) action parameters and variables in the original program. Note that the update code of r uses quantifier-free updates and does not utilize *additional* existential quantifiers. A proof of soundness with the interpretation of the existential quantifiers of $\widehat{\delta}$ is just as in Example 4.3: for every transition of the original program there is a matching transition of the instrumented program when r is interpreted according to ψ *and* the existential quantifiers of the instrumented program are interpreted to match the existential quantifiers of the original program as described here. Therefore, $\widehat{\delta}$ is a sound instrumentation without additional existentials.

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 Example 3.7 — see the `if` statement in action `check`).

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 Example 3.7, which admits an instrumentation that transforms the invariant to a universal invariant (see Example 4.1) in a form that matches Theorem 4.13, 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 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). This is the case in the example of Example 3.9, where a bound of 1 does not suffice.[5]

While we show that 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 4.9) can be used to simulate quantifier instantiation of an arbitrary depth. This topic is explored in Section 5.

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 $I = \widehat{I}[\psi/r]$.

Results. We now state the results whose proofs are presented in the rest of this section. Theorem 4.13 and Theorem 4.14 consider $I \in \forall^*\exists^*(\Sigma)$ that is transformed to $\widehat{I} \in \forall^*(\widehat{\Sigma})$. In Theorem 4.13 we show that a bound of 1 suffices to prove that I is inductive for δ when $\psi \in \exists^*(\Sigma)$ (that is, the instrumentation defining formula is existential) and the instrumentation relation r appears only positively in $\widehat{I}$, or when $\psi \in \forall^*(\Sigma)$ and r appears only negatively in $\widehat{I}$. This is an attempt to explain the success of bound 1 instantiations in

[4]Another potential source of existential quantifiers is existentially quantified `assume` statements, which are not present in the instrumented program.

[5]Strictly speaking this shows that there is no such instrumentation where the instrumentation relation appears only positively in the invariant, which is the most common case. Examples that require an even larger bound (sketched above) do not admit any instrumentation without additional existential quantifiers that transforms the invariant to a universal form.

proving our examples (see Section 7). In Theorem 4.14 we show that a bound of 2 suffices in the more general setting of $\psi \in \mathrm{AF}(\Sigma)$ (with no restriction on appearances of r in $\widehat{I}$).

Theorem 4.16 considers a generalization to I that is 1-alternation and transformed to $\widehat{I} \in \mathrm{AF}(\widehat{\Sigma})$. We show that a bound of 2 suffices in this case.

Proof idea. The rest of the section is devoted to proofs of these claims. The idea of the proof concentrates around the instantiations necessary to prove inductiveness of the instrumented and original invariants. To highlight the main points in the formal proof, the crux of the argument is as follows.

(1) Assume for the sake of contradiction that bounded instantiations of a low bound on the *original* invariant I do not suffice to prove it inductive w.r.t. the *original* program δ, and take a counterexample to induction of the instantiated I (see Equation (4.2) in the proof of Lemma 4.12).
(2) Exploiting properties of substitution, connect instantiations of the original and of the instrumented invariants through the assumption that $\widehat{\delta}$ is an instrumentation without additional existentials for δ to obtain a counterexample to induction for the instantiated $\widehat{I}$ w.r.t. $\widehat{\delta}$ (see Equation (4.5)).
(3) Rely on the assumption that the instrumented invariant $\widehat{I}$ is universal and $\widehat{\delta} \in \mathrm{EPR}(\widehat{\Sigma})$. By Remark 3.3, this means that instantiations of bound 0—namely, with just the constants—suffice to prove $\widehat{I}$ inductive w.r.t. the instrumented program. In other words, a counterexample to induction obtained for the *instantiated* $\widehat{I}$ w.r.t. $\widehat{\delta}$ is a *true* counterexample to induction of $\widehat{I}$ w.r.t. $\widehat{\delta}$ (see Equation (4.9)), in contradiction to the premise.

In essence, the proof translates an instantiation-based proof of the instrumented invariant to a proof of the original invariant by *the same set of terms* instantiating the universal quantifiers; the set of constants, sufficient for the instrumented invariant, must thus be sufficient also for the original invariant, where this amounts to Equation (3.2), thus constituting a proof by bound 1 instantiations.

The formal proofs handle the fine details of relating between the universal quantifiers and the constants of the original and instrumented invariant, to complete the transformation of instantiations between them.

Remark 4.11. The results of this section also apply when multiple instrumentation relations $\psi_1, \ldots, \psi_t \in \mathrm{FOL}(\Sigma)$ are simultaneously substituted for the relation symbols $r_1, \ldots, r_t$ in $\widehat{\delta}$ and $\widehat{I}$.

4.3. **Power for $\forall^*\exists^*$ Invariants.** We now establish that low bounds are sufficient for the Bounded-Horizon check, assuming that a sound instrumentation without additional existentials exists, in the case of $\widehat{I} \in \forall^*(\widehat{\Sigma})$ and $I \in \forall^*\exists^*(\Sigma)$. To do so, we first prove the following lemma.

Lemma 4.12. *Let $\widehat{\delta}$ be a sound instrumentation of δ, ψ without new existentials and with naming η, and let $\widehat{I} \in \forall^*(\widehat{\Sigma})$ be an inductive invariant for $\widehat{\delta}$. Write $\widehat{I} = \forall \overline{x}.\ \widehat{\alpha}(\overline{x})$ where $\widehat{\alpha} \in \mathrm{QF}(\widehat{\Sigma})$ and let $\alpha = \mathrm{nnf}(\widehat{\alpha}[\psi/r])$. Then,*

$$\Big(\bigwedge_{\overline{c} \in C^n} \alpha(\overline{c})\Big) \ \wedge\ \delta_S \ \wedge\ (\neg I')_S \tag{4.1}$$

is unsatisfiable, where $C = const[\delta_S \wedge (\neg I')_S]$ *and* n *is the number of universal quantifiers in* $\widehat{I}$.

Proof. Assume not, so there exists a structure $\mathcal{A}_0$ satisfying Equation (3.2), namely

$$\mathcal{A}_0 \models \big(\bigwedge_{\overline{c} \in C^n} \alpha(\overline{c}) \big) \ \wedge \ \delta_S \ \wedge \ (\neg I')_S \ . \tag{4.2}$$

We will show that $\widehat{I}$ is not inductive for $\widehat{\delta}$. Let $\widehat{C} = \text{const}[\eta(\widehat{\delta}) \wedge (\neg \widehat{I}')_S]$. Then,

$$\mathcal{A}_1 \models \big(\bigwedge_{\overline{c} \in (C \cup \widehat{C})^n} \alpha(\overline{c}) \big) \ \wedge \ \delta_S \ \wedge \ (\neg I')_S \tag{4.3}$$

where $\mathcal{A}_1$ is the same as $\mathcal{A}_0$ but also interprets any constant in $\widehat{C} \setminus C$ as the interpretation of some arbitrary constant in C. Thus $\alpha(\overline{c})$ holds in $\mathcal{A}_1$ for the new constants as well.

Removing some conjuncts from Equation (4.3), we get,

$$\mathcal{A}_1 \models \big(\bigwedge_{\overline{c} \in \widehat{C}^n} \alpha(\overline{c}) \big) \ \wedge \ \delta_S \ \wedge \ (\neg I')_S \ . \tag{4.4}$$

By assumption (Definition 4.9), it follows that,

$$\mathcal{A}_1 \models \big(\bigwedge_{\overline{c} \in \widehat{C}^n} \alpha(\overline{c}) \big) \ \wedge \ \eta(\widehat{\delta})[\psi/r, \psi'/r'] \ \wedge \ (\neg I')_S. \tag{4.5}$$

Recall that $I' = \widehat{I}[\psi'/r']$. Since $\mathcal{A}_1 \models (\neg \widehat{I}[\psi'/r'])_S$, it follows that $\mathcal{A}_1 \models (\neg \widehat{I}')_S[\psi'/r']$. In the latter formula, some existentially quantified variables from ψ or $\neg\psi$ may remain, whereas in the former formula they were replaced by Skolem constants. Thus this is just a corollary of the fact that $\gamma_S \to \gamma$ is valid for any γ.

Thus we have shown (recalling that $\alpha = \text{nnf}(\widehat{\alpha}[\psi/r])$ and $\widehat{\alpha}[\psi/r]$ are equivalent),

$$\mathcal{A}_1 \models \Big(\big(\bigwedge_{\overline{c} \in \widehat{C}^n} \widehat{\alpha}(\overline{c}) \big) \ \wedge \ \eta(\widehat{\delta}) \ \wedge \ (\neg \widehat{I}')_S \Big)[\psi/r, \psi'/r']. \tag{4.6}$$

Now, consider the structure $\widehat{\mathcal{A}}$ that expands $\mathcal{A}_1$ by interpreting r and r' the way that $\mathcal{A}_1$ interprets ψ and ψ', respectively. Then,

$$\widehat{\mathcal{A}} \models \big(\bigwedge_{\overline{c} \in \widehat{C}^n} \widehat{\alpha}(\overline{c}) \big) \ \wedge \ \eta(\widehat{\delta}) \ \wedge \ (\neg \widehat{I}')_S. \tag{4.7}$$

The formula in Equation (4.7) is a universal sentence: $\widehat{\alpha}(\overline{c})$ is quantifier free and closed, $\eta(\widehat{\delta}) \in \forall^*$ from the definition of existential naming, and $\neg \widehat{I}' \in \exists^*$ and thus its Skolemization introduces only constants. It follows that the formula in Equation (4.7) is also satisfied by $\widehat{\mathcal{A}}_{|\widehat{C}}$, the substructure of $\widehat{\mathcal{A}}$ with universe $\widehat{C}^{\widehat{\mathcal{A}}}$, i.e., $\widehat{\mathcal{A}}$'s interpretation of the constant symbols $\widehat{C}$; recall that $\widehat{C} = \text{const}[\eta(\widehat{\delta}) \wedge (\neg \widehat{I}')_S]$ so indeed $\widehat{\mathcal{A}}$ provides an interpretation to every constant in the formula. Thus,

$$\widehat{\mathcal{A}}_{|\widehat{C}} \models \big(\forall \overline{x}. \ \widehat{\alpha}(\overline{x}) \big) \ \wedge \ \eta(\widehat{\delta}) \ \wedge \ (\neg \widehat{I}')_S. \tag{4.8}$$

Finally, since $\gamma_S \to \gamma$ is valid and so is $\eta(\widehat{\delta}) \to \widehat{\delta}$ (for the same reasons), we know,

$$\widehat{\mathcal{A}}_{|\widehat{C}} \models \widehat{I} \ \wedge \ \widehat{\delta} \ \wedge \ \neg \widehat{I}'. \tag{4.9}$$

But this contradicts the fact that $\widehat{I}$ is inductive for $\widehat{\delta}$. □

The following results are corollaries of Lemma 4.12.

Theorem 4.13. *Let $\widehat{I} \in \forall^*(\widehat{\Sigma})$ be an inductive invariant for $\widehat{\delta}$, which is a sound instrumentation for δ, ψ without additional existentials. Assume $\psi \in \exists^*(\Sigma)$ and r appears only positively in $\widehat{I}$, or $\psi \in \forall^*(\Sigma)$ 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 $I \in \forall^*\exists^*(\Sigma)$.)*

Proof. Let $\widehat{I} = \forall \overline{x}.\, \widehat{\alpha}(\overline{x})$ where $\widehat{\alpha} \in \mathrm{QF}$. In both cases of the claim $\alpha = \mathrm{nnf}(\widehat{\alpha}[\psi/r]) \in \exists^*$, and so all the universal quantifiers in I (more accurately, in $\mathrm{nnf}(I)$) are those of $\widehat{I}$. This implies that the satisfiability check of Lemma 4.12 is simply the Bounded-Horizon satisfiability check with bound 1, and it shows that the result must be unsatisfiable.

More formally, assume for the sake of contradiction that I is not inductive w.r.t. δ with Bounded-Horizon of bound 1. Let $\alpha(\overline{x}) = \exists y_1, \ldots, y_m.\, \theta(\overline{x}, y_1, \ldots, y_m)$ where $\theta \in \mathrm{QF}$, and let

$$\alpha_S(\overline{x}) = \theta(\overline{x}, f_1(\overline{x}), \ldots, f_m(\overline{x}))$$

be its Skolemization with fresh Skolem function symbols $f_1, \ldots, f_m$ (introduced for $y_1, \ldots, y_m$, respectively). Then there is a structure $\mathcal{A}$ satisfying Equation (3.2), which reads

$$\Big(\bigwedge_{\overline{t} \in \mathrm{BHT}_0} \alpha_S[\overline{t}]\Big) \wedge \Big(\bigwedge_{\overline{t} \in \mathrm{BHT}_1} \delta_S[\overline{t}]\Big) \wedge \Big(\bigwedge_{\overline{t} \in \mathrm{BHT}_1} \big(\neg I'\big)_S[\overline{t}]\Big). \tag{4.10}$$

Since α_S has no universal quantifiers, the instantiation is just a substitution of the free variables, and $\mathcal{A}$ satisfies

$$\Big(\bigwedge_{\overline{t} \in \mathrm{BHT}_0} \alpha_S(\overline{t})\Big) \wedge \Big(\bigwedge_{\overline{t} \in \mathrm{BHT}_1} \delta_S[\overline{t}]\Big) \wedge \Big(\bigwedge_{\overline{t} \in \mathrm{BHT}_1} \big(\neg I'\big)_S[\overline{t}]\Big). \tag{4.11}$$

By reducing $\mathcal{A}$ to the elements pointed to by BHT_1 terms we have that

$$\mathcal{A}_{|_{\mathrm{BHT}_1}} \models \Big(\bigwedge_{\overline{t} \in \mathrm{BHT}_0} \alpha_S(\overline{t})\Big) \wedge \delta_S \wedge \big(\neg I'\big)_S \tag{4.12}$$

Note that in $\mathcal{A}_{|_{\mathrm{BHT}_1}}$ the interpretations of the Skolem functions are possibly partial functions. The functions appear in the formula of Equation (4.12) as closed terms, and applied on BHT_0, and these cases the interpretations of the functions are defined. (In particular, they can be extended to total functions in an arbitrary way, and the resulting structure still satisfies Equation (4.12).)

We now move from the Skolem functions back to existential quantifiers. By the valuation that to every existentially quantified variable y_i in α assigns the interpretation of $f_i(\overline{t})$ in $\mathcal{A}_{|_{\mathrm{BHT}_1}}$ (recall that $f_i(\overline{t})$ appears in α_S instead of the quantifier $\exists y_i$ in α), we know that

$$\mathcal{A}_{|_{\mathrm{BHT}_1}} \models \Big(\bigwedge_{\overline{t} \in \mathrm{BHT}_0} \alpha(\overline{t})\Big) \wedge \delta_S \wedge \big(\neg I'\big)_S. \tag{4.13}$$

The set of constants referred to by BHT_0 is the set of constants in Equation (4.10), which is $\mathrm{const}[\delta_S \wedge \big(\neg I'\big)_S]$. Therefore, Equation (4.13) can be rewritten as

$$\mathcal{A}_{|_{\mathrm{BHT}_1}} \models \Big(\bigwedge_{\overline{c} \in C^n} \alpha(\overline{c})\Big) \wedge \delta_S \wedge \big(\neg I'\big)_S \tag{4.14}$$

where $C^n = \mathrm{const}[\delta_S \wedge \big(\neg I'\big)_S]^n$ and n is the number of universal quantifiers in I (and $\widehat{I}$).

By Lemma 4.12 this is a contradiction to the assumption that $\widehat{I}$ is inductive w.r.t. $\widehat{\delta}$, and the claim follows. □

Theorem 4.14. *Let $\widehat{I} \in \forall^*$. If $\psi \in \mathrm{AF}(\Sigma)$ then $I = \widehat{I}[\psi/r]$ is inductive for δ with Bounded-Horizon of bound 2. (Note that $I \in \forall^*\exists^*(\Sigma)$.)*

Proof. As before, let $\widehat{I} = \forall\overline{x}.\ \widehat{\alpha}(\overline{x})$ where $\widehat{\alpha} \in \mathrm{QF}$, and let $\alpha = \mathrm{nnf}(\widehat{\alpha}[\psi/r])$. Since $\widehat{\alpha} \in \mathrm{QF}$ and $\psi \in \mathrm{AF}$, α is a positive Boolean combination of formulas of the form $\forall\overline{v_i}\theta_{i,1}(\overline{x},\overline{v_i})$ and $\exists\overline{z_j}\theta_{j,2}(\overline{x},\overline{z_j})$ where $\overline{v_i} \cap \overline{z_j} = \emptyset$ for all i, j. In α_S, each formula $\exists\overline{z_j}\theta_{j,2}(\overline{x},\overline{z_j})$ is replaced by $\theta_{j,2}(\overline{x},\overline{g_i}(\overline{x}))$ where $\overline{g_i}$ are fresh Skolem function symbols. (Note that $\overline{x}$ is free in α.)

Assume for the sake of contradiction that I is not inductive w.r.t. δ with Bounded-Horizon of bound 2.

For brevity denote

$$\xi(k) = \big(\bigwedge_{\overline{t}\in\mathrm{BHT}_k} \delta_S[\overline{t}]\big) \wedge \big(\bigwedge_{\overline{t}\in\mathrm{BHT}_k} \big(\neg I'\big)_S[\overline{t}]\big).$$

By the assumption that inductiveness is not provable using Bounded-Horizon of bound 2,

$$\big(\bigwedge_{\overline{t}\in\mathrm{BHT}_1} \alpha_S[\overline{t}]\big) \wedge \xi(2) \tag{4.15}$$

is satisfiable by a structure $\mathcal{A}$.

In particular

$$\big(\bigwedge_{\substack{\overline{c}\in\mathrm{BHT}_0,\\ \overline{d_1},\ldots,\overline{d_r}\in\mathrm{BHT}_1}} \alpha_S[\overline{t}]\big) \wedge \xi(1) \tag{4.16}$$

is satisfied by $\mathcal{A}$ (the arity of $\overline{d_i}$ is understood from the number of universal quantifiers in the respective universal term). The reason that Equation (4.16) is satisfied by $\mathcal{A}$ is that Equation (4.16) differs from Equation (4.15) by having *fewer* conjuncts, as the Bounded-Horizon check with bound 2 has conjuncts for each $\overline{c} \in \mathrm{BHT}_1$ and not just BHT_0, and we take $\xi(1)$ instead of $\xi(2)$ ($\mathrm{BHT}_1 \subseteq \mathrm{BHT}_2$ so the conjuncts of $\xi(1)$ are included in those of $\xi(2)$).

Reduce $\mathcal{A}$ to the elements pointed by BHT_1 terms, let $\mathcal{A}^{\downarrow} = \mathcal{A}_{|_{\mathrm{BHT}_1}}$.

Now,

$$\big(\bigwedge_{\overline{c}\in\mathrm{BHT}_0} \alpha[\overline{t}]\big) \wedge \xi(1) \tag{4.17}$$

is satisfied by $\mathcal{A}^{\downarrow}$. This is because:

- The universal quantifiers are semantically equivalent to a conjunction over all BHT_1 elements because the domain was reduced, and
- The existential quantifiers are justified by the following valuation: the valuation assigns every $\overline{z_i}$ the interpretation of $\overline{g_i}(\overline{c})$.

With this valuation the conjunctions of formula 4.17 are all guaranteed by the conjunctions in formula 4.16.

Now formula 4.17 exactly means that

$$\mathcal{A}^{\downarrow} \models \big(\bigwedge_{\overline{c}\in\mathrm{BHT}_0} \alpha(\overline{c})\big) \wedge \xi(1). \tag{4.18}$$

As in the proof of Theorem 4.13, using Lemma 4.12, this is a contradiction to the assumption that $\widehat{I}$ is inductive w.r.t. $\widehat{\delta}$, and the claim follows. □

4.4. **Generalization to 1-Alternation Invariants.** We now generalize the results of Section 4.3 to *1-alternation invariants*. A formula is 1-alternation if it can be written as a Boolean combination of $\forall^*\exists^*$ formulas. In the sequel, $\widehat{I} \in \mathrm{AF}(\widehat{\Sigma})$ and $I = \widehat{I}[\psi/r] \in$ 1-alternation(Σ).

Lemma 4.15. *Let $\psi \in \mathrm{FOL}(\Sigma)$. Let $\delta \in \mathrm{EPR}(\Sigma \uplus \Sigma')$ and let $\widehat{\delta} \in \mathrm{EPR}(\widehat{\Sigma} \uplus \widehat{\Sigma}')$ be a sound instrumentation of δ, ψ. Let $\widehat{I} \in \mathrm{AF}(\widehat{\Sigma})$ be an inductive invariant for $\widehat{\delta}$, and write $\widehat{I}_S = \forall \overline{x}.\ \widehat{\alpha_1}(\overline{x})$ and $(\neg\widehat{I})_S = \forall \overline{x}.\ \widehat{\alpha_2}(\overline{x})$, where $\widehat{\alpha_1}, \widehat{\alpha_2}$ are quantifier free. Let $\alpha_1 = \mathrm{nnf}(\widehat{\alpha_1}[\psi/r])$ and $\alpha_2 = \mathrm{nnf}(\widehat{\alpha_2}[\psi/r])$. Then,*

$$\Big(\bigwedge_{\overline{c_1} \in C^n} \alpha_1(\overline{c_1})\Big) \ \wedge\ \delta_S \ \wedge\ \Big(\bigwedge_{\overline{c_2} \in C^m} \alpha_2(\overline{c_2})\Big) \tag{4.19}$$

is unsatisfiable, where $C = \mathit{const}[\widehat{I}_S \wedge \delta_S \wedge (\neg I')_S]$, n is the number of universal quantifiers in $\widehat{I}_S$ and m is the number of universal quantifiers in $(\neg\widehat{I})_S$.

Proof. The proof is similar to that of Lemma 4.12; this proof follows the same reasoning to transform satisfiable formulas, but transforms not only the quantifier and conjunctions related to the invariant in the pre-state, but also to its negation in the post-state.

Assume not, i.e., there exists a structure $\mathcal{A}_0$ such that,

$$\mathcal{A}_0 \models \Big(\bigwedge_{\overline{c_1} \in C^n} \alpha_1(\overline{c_1})\Big) \ \wedge\ \delta_S \ \wedge\ \Big(\bigwedge_{\overline{c_2} \in C^m} \alpha_2(\overline{c_2})\Big). \tag{4.20}$$

We will show that $\widehat{I}$ is not inductive for $\widehat{\delta}$. Let $\widehat{C} = \mathrm{const}[\widehat{I}_S \wedge \eta(\widehat{\delta}) \wedge (\neg\widehat{I}')_S]$. Then,

$$\mathcal{A}_1 \models \Big(\bigwedge_{\overline{c_1} \in \widehat{C}^n} \alpha_1(\overline{c_1})\Big) \ \wedge\ \delta_S \ \wedge\ \Big(\bigwedge_{\overline{c_2} \in \widehat{C}^m} \alpha_2(\overline{c_2})\Big). \tag{4.21}$$

where $\mathcal{A}_1$ is the same as $\mathcal{A}_0$ but also interprets any constant in $\widehat{C} \setminus C$ as some arbitrary constant in C.

By the assumption (Definition 4.9), it follows that,

$$\mathcal{A}_1 \models \Big(\bigwedge_{\overline{c_1} \in \widehat{C}^n} \alpha_1(\overline{c_1})\Big) \ \wedge\ \eta(\widehat{\delta})[\psi/r, \psi'/r'] \ \wedge\ \Big(\bigwedge_{\overline{c_2} \in \widehat{C}^m} \alpha_2(\overline{c_2})\Big). \tag{4.22}$$

where η is the existential naming between $\delta, \widehat{\delta}$.

Thus we have shown (recalling that $\alpha_1 = \mathrm{nnf}(\widehat{\alpha_1}[\psi/r])$ and $\widehat{\alpha_1}[\psi/r]$ are equivalent, and similarly for α_2),

$$\mathcal{A}_1 \models \Big(\Big(\bigwedge_{\overline{c_1} \in \widehat{C}^n} \widehat{\alpha_1}(\overline{c_1})\Big) \ \wedge\ \eta(\widehat{\delta}) \ \wedge\ \Big(\bigwedge_{\overline{c_2} \in \widehat{C}^m} \widehat{\alpha_2}(\overline{c_2})\Big)\Big)[\psi/r, \psi'/r']. \tag{4.23}$$

Now, consider the structure $\widehat{\mathcal{A}}$ that expands $\mathcal{A}_1$ by interpreting r and r' the way that $\mathcal{A}_1$ interprets ψ and ψ', respectively. Then,

$$\widehat{\mathcal{A}} \models \Big(\bigwedge_{\overline{c_1} \in \widehat{C}^n} \widehat{\alpha_1}(\overline{c_1})\Big) \ \wedge\ \eta(\widehat{\delta}) \ \wedge\ \Big(\bigwedge_{\overline{c_2} \in \widehat{C}^m} \widehat{\alpha_2}(\overline{c_2})\Big) \tag{4.24}$$

The formula in Equation (4.24) is a universal sentence: $\widehat{\alpha_1}(\overline{c}), \widehat{\alpha_2}(\overline{c})$ are quantifier free and closed, and $\eta(\widehat{\delta}) \in \forall^*$. It follows that the formula in Equation (4.24) $\widehat{\mathcal{A}}_{|\widehat{C}}$, the substructure of $\widehat{\mathcal{A}}$ with universe $\widehat{C}^{\widehat{\mathcal{A}}}$, i.e., $\widehat{\mathcal{A}}$'s interpretation of the constant symbols $\widehat{C}$; recall

that $\widehat{C} = \text{const}[\widehat{I}_S \wedge \eta(\widehat{\delta}) \wedge (\neg\widehat{I}')_S]$ (and $\widehat{I}_S = \forall\overline{x}.\ \widehat{\alpha_1}(\overline{x})$ and $(\neg\widehat{I})_S = \forall\overline{x}.\ \widehat{\alpha_2}(\overline{x})$) so indeed $\widehat{\mathcal{A}}$ provides an interpretation to every constant in the formula. Thus,

$$\widehat{\mathcal{A}}_{|\widehat{C}} \models \big(\forall\overline{x}.\ \widehat{\alpha_1}(\overline{x})\big) \ \wedge\ \eta(\widehat{\delta}) \ \wedge\ \big(\forall\overline{x}.\ \widehat{\alpha_2}(\overline{x})\big). \tag{4.25}$$

Recall that $\widehat{C}$ was defined as $\widehat{C} = \text{const}[\widehat{I} \wedge \eta(\widehat{\delta}) \wedge (\neg\widehat{I}')_S]$.

Finally, since $\gamma_S \to \gamma$ is valid and for the same reasons $\eta(\widehat{\delta}) \to \widehat{\delta}$ is valid, we know,

$$\widehat{\mathcal{A}}_{|\widehat{C}} \models \widehat{I} \ \wedge\ \widehat{\delta} \ \wedge\ \neg\widehat{I}'. \tag{4.26}$$

But this contradicts the fact that $\widehat{I}$ is inductive for $\widehat{\delta}$. □

The following result is a corollary of Lemma 4.15.

Theorem 4.16. *Let $\widehat{I} \in \text{AF}(\widehat{\Sigma})$ an inductive invariant for $\widehat{\delta}$ which is a sound instrumentation of δ without additional existentials. If $\psi \in \text{AF}(\Sigma)$ then $I = \widehat{I}[\psi/r]$ is inductive for δ with Bounded-Horizon of bound 2. (Note that $I \in$ 1-alternation(Σ).)*

Proof. $\psi \in$ AF implies that $\alpha_1(\overline{x}) = \text{nnf}(\widehat{\alpha_1}[\psi/r]) \in$ AF and $\alpha_2(\overline{x}) = \text{nnf}(\widehat{\alpha_2}[\psi/r]) \in AF$ (recall that $\widehat{\alpha_1}, \widehat{\alpha_2} \in$ QF).

By the assumption that inductiveness is not provable using Bounded-Horizon of bound 2,

$$\Big(\bigwedge_{\overline{t}\in\text{BHT}_1} (\alpha_1)_S[\overline{t}]\Big) \wedge \Big(\bigwedge_{\overline{t}\in\text{BHT}_2} \delta_S[\overline{t}]\Big) \wedge \Big(\bigwedge_{\overline{t}\in\text{BHT}_1} (\alpha_2)_S[\overline{t}]\Big) \tag{4.27}$$

is satisfiable, where $(\alpha_1)_S, (\alpha_2)_S$ introduce Skolem functions. Let $\mathcal{A}$ be such a satisfying structure, and $\mathcal{A}^{\downarrow} = \mathcal{A}_{|\text{BHT}_1}$.

Because $\alpha_1 \in$ AF, in the same way as in the proof of Theorem 4.14,

$$\mathcal{A}^{\downarrow} \models \bigwedge_{\overline{c}\in\text{BHT}_0} \alpha_1(\overline{c}) \tag{4.28}$$

and in the same way, since $\alpha_2 \in$ AF as well, the same structure has

$$\mathcal{A}^{\downarrow} \models \bigwedge_{\overline{c}\in\text{BHT}_0} \alpha_2(\overline{c}). \tag{4.29}$$

Note that from Equation (4.27), $\mathcal{A}^{\downarrow} \models \widehat{\delta}$. Overall we have

$$\mathcal{A}^{\downarrow} \models \Big(\bigwedge_{\overline{c}\in\text{BHT}_0} \alpha_1(\overline{c})\Big) \ \wedge\ \delta_S \ \wedge\ \Big(\bigwedge_{\overline{c}\in\text{BHT}_0} \alpha_2(\overline{c})\Big) \tag{4.30}$$

and by Lemma 4.15 this is a contradiction to the assumption that $\widehat{I}$ a is inductive w.r.t. $\widehat{\delta}$. □

5. Instrumentation for High Depth Instantiations

In this section we discuss the connection between quantifier instantiation and program instrumentation in the converse direction, i.e. simulating quantifier instantiation by the process of instrumentation. In Section 4 we showed that instrumentation without adding existential quantifiers is at most as powerful as bounded instantiations with a low bound. In this section we show that allowing additional existentials does increase the power of instrumentation in proving $\forall^*\exists^*$ invariants. In particular, we show how to systematically construct instrumented programs in a way that corresponds to quantifier instantiation for $\forall^*\exists^*$-invariants: performing the required instantiations within the program allows expressing

the invariant in a form that falls in the decidable fragment. Together with Section 4, this makes the point that, in the context of invariant checking, the form of instrumentation by a derived relation studied in this paper directly corresponds to quantifier instantiation.

We are again interested in an original program δ with a $\forall^*\exists^*$ inductive invariant $I \in \forall^*\exists^*(\Sigma)$. Our goal is to prove that I is inductive w.r.t. δ by a reduction to a decidable class. We therefore identify some existential formula $\psi(\overline{x}) \in \exists^*(\Sigma)$ that expresses needed information, and encode it using an instrumentation relation r, with the meaning that "$r(\overline{x}) \equiv \psi(\overline{x})$". Instrumentation then modifies δ to produce an instrumented program $\widehat{\delta}$ with a universal inductive invariant $\widehat{I}$ such that $I = \widehat{I}[\psi/r]$ (at least up to redundant tautological clauses in I).

Intuitively, adding the instrumentation relation r lets $\widehat{I} \in \forall^*(\widehat{\Sigma})$ express existential information by referring to r instead. The modifications to the program must encode "enough" of "$r(\overline{x}) \equiv \psi(\overline{x})$" to make $\widehat{I}$ inductive. In Section 4.1, this was performed by adding update code and rewriting program conditions. In this section, we will take a different approach to instrumentation: instantiating the correspondence (using `assume` statements) between ψ and r for specific variables in the program—relating $r(\overline{t})$ to $\psi(\overline{t})$ where $\overline{t}$ is a specific tuple of closed terms. While different from the process discussed in Section 4.1, the result is still a sound instrumentation per Definition 4.2. We show this approach—closely following quantifier instantiation—can systematically construct instrumentations $\widehat{\delta}, \widehat{I}$ whose effect is to prove that I is inductive w.r.t. δ, by encoding the necessary instantiations. We begin by defining the instrumentation process we use in this section.

Instrumentation by Local Instantiations. We would like to enforce r to be interpreted according to ψ in the pre-state, i.e., to enforce $\forall\overline{x}.\ \psi(\overline{x}) \leftrightarrow r(\overline{x})$. The direction $\forall\overline{x}.\ \psi(\overline{x}) \rightarrow r(\overline{x})$ is an EPR formula (since $\psi \in \exists^*$), and thus we can simply conjoin it to the verification conditions without sacrificing decidability.

The converse implication, $\forall\overline{x}.\ r(\overline{x}) \rightarrow \psi(\overline{x})$, is a $\forall^*\exists^*$ formula, and adding it to the verification condition will lead to a formula that does not belong to the decidable EPR class. Note that Bounded-Horizon with bound 1 is analogous to enforcing $r(\overline{t}) \rightarrow \psi(\overline{t})$ for every $\overline{t}$ that is a tuple of program variables. Inspired by this, we define the following instrumentation that lets the user locally enforce the definition of r for program variables.

Definition 5.1 (Local Instantiation)**.** *Let $\psi(\overline{x}) = \exists\overline{y}.\ \varphi(\overline{x},\overline{y})$ where $\varphi \in \mathrm{QF}(\Sigma)$. To generate a* local instantiation *of $\forall\overline{x}.\ r(\overline{x}) \rightarrow \exists\overline{y}.\ \varphi(\overline{x},\overline{y})$ on some tuple of program variables $\overline{t}$, we instrument the program by adding new local program variables $\overline{c}$, and inserting the following code:*

$$\textbf{local}\ \overline{c} := *;\ \texttt{assume}\ r(\overline{t}) \rightarrow \varphi(\overline{t},\overline{c}). \tag{5.1}$$

The code in Equation (5.1) uses the havoc statement, which sets the value of $\overline{c}$ to arbitrary values, followed by an assume statement that restricts the execution such that if $r(\overline{t})$ holds, then $\varphi(\overline{t},\overline{c})$ holds. Thus, this code realizes the restriction that $r(\overline{t}) \rightarrow \exists\overline{y}.\ \varphi(\overline{t},\overline{y})$, and also assigns to the new program variables $\overline{c}$ the "witnesses" for the existential quantifiers. Note that the new variables translate to existential quantifiers in the transition relation (through the semantics of havoc—see Figure 3, recalling that the transition relation is obtained by negation). We call this addition to the program a local instantiation, as it imposes the connection between r and ψ locally for some program variables $\overline{t}$.

Lemma 5.2 (Soundness of Local Instantiations)**.** *If $\widehat{\delta}$ is obtained from δ by a local instantiation then $\widehat{\delta}$ is a sound instrumentation with ψ per Definition 4.2.*

Proof. It suffices to show that the logical constraint generated in $\widehat{\delta}$ as a result of the `assume` command does not exclude any transition allowed by δ. The code added by a local instantiation for $\bar{t}$ translates to a new constraint in $\widehat{\delta}$ of the form $\gamma = \exists \overline{y}.\ r(\bar{t}) \rightarrow \varphi(\bar{t}, \overline{y})$ through the semantics of havoc and `assume` (see Figure 3). Note that the variables added through local instantiation are translated to existential quantifiers, not new constants. Since $(\forall \overline{x}.\ r(\overline{x}) \leftrightarrow \psi(\overline{x})) \rightarrow \gamma$ is valid (recall that $\psi(\overline{x}) = \exists \overline{y}.\ \varphi(\overline{x}, \overline{y})$), we have $((\forall \overline{x}.\ r(\overline{x}) \leftrightarrow \psi(\overline{x})) \ \wedge \ \delta) \rightarrow \widehat{\delta}$ is valid, which implies the condition of Definition 4.2. □

Remark 5.3. The combination of adding $\forall \overline{x}.\ \psi(\overline{x}) \rightarrow r(\overline{x})$ to the verification condition and allowing the user to perform local instantiations on the program variables is at least as powerful as rewriting program conditions, since any rewrite of $\psi(\bar{t})$ to $r(\bar{t})$ can be simulated by a local instantiation on $\bar{t}$.

Instantiations for the Invariant. The mechanism of local instantiations is designed to support instantiations of $\forall \overline{x}.\ \psi(\overline{x}) \leftrightarrow r(\overline{x})$ required to prove that the invariant $I = \widehat{I}[\psi/r]$ is preserved by the program. This proof is carried out by showing that $(\forall \overline{x}.\ \psi(\overline{x}) \leftrightarrow r(\overline{x})) \wedge \widehat{I} \wedge \delta \ \wedge \ \neg(\widehat{I}[\psi/r])'$ is unsatisfiable. This may require instantiating the definition of r on Skolem constants that come from the negation of the invariant in the post-state. Thus, we extend our instrumentation method by adding new "program variables" that represent the elements of the domain on which the invariant is potentially violated in the post-state. For every existentially quantified variable x in $\neg\widehat{I}$, we add a special program variable sk_x which can be used in local instantiations, enhancing their power to prove that $\widehat{I}$ is inductive.

To this end we formally use a slightly modified inductiveness check, termed *Skolemization-aware inductiveness*. In this check, existentially quantified variables can be shared between $\widehat{\delta}$ and $(\neg\widehat{I})_S$, facilitating local instantiations on Skolem constants coming from the negation of the invariant. Formally, denote $\widehat{I} = \forall \overline{x}.\ \theta(\overline{x})$ where $\theta(\overline{x}) \in \mathrm{QF}(\widehat{\Sigma})$ (recall that $\widehat{I} \in \forall^*(\widehat{\Sigma})$). Let $\overline{s}$ be free variables, intended to be shared between $\widehat{\delta}$ and $(\neg\widehat{I})_S$. Then $\overline{s}$ replace the existentially quantified variables $\overline{x}$ in $\neg\widehat{I}$, and also replace the existentially quantified variables in $\widehat{\delta}$ introduced by local instantiations performed on $\overline{sk_x}$, thus linking them. The inductiveness check now translates to the unsatisfiability of

$$(\forall \overline{x}.\ \theta(\overline{x})) \wedge \exists \overline{s}.\ \left(\widehat{\delta} \wedge \neg\theta'(\overline{s})\right). \tag{5.2}$$

Obtaining Deep Instantiations. Applying local instrumentation on a tuple $\bar{t}$ that consists of original program variables, or variables that represent Skolem constants, corresponds to instantiations of Bounded-Horizon with bound 1. However, once a local instantiation is performed, new program variables $\bar{c}$ are added. Performing a local instantiation on these new variables now corresponds to instantiation of bound 2. By iteratively applying local instantiations, where each iteration adds new program variables (corresponding to existential quantifiers in the transition relation), we can thus simulate quantifier instantiations of arbitrary depth.

Overall, the procedure leads to the following claim:

Lemma 5.4. *Let $I \in \forall^*\exists^*(\Sigma)$ be an inductive invariant for $\delta \in \mathrm{EPR}(\Sigma \uplus \Sigma')$, provable using Bounded-Horizon of bound k. Let $\psi \in \exists^*(\Sigma)$ be its existential sub-formula, i.e., $I = \forall \overline{x}.\ \psi(\overline{x})$.*

Then it is possible to construct $\widehat{\delta} \in \mathrm{EPR}(\widehat{\Sigma} \uplus \widehat{\Sigma}')$ *where* $\widehat{\Sigma} = \Sigma \uplus \widehat{S} \uplus \{r\}$, $\widehat{S}$ *being the Skolem constants from* $(\neg I)_S$ *and* r *being a fresh relation symbol, such that*

- $\widehat{\delta}$ *is a sound instrumentation of* δ *and* ψ*, and*
- $\widehat{I} = (\forall \overline{x}.\ r(\overline{x})) \wedge (\forall \overline{x}.\ \psi(\overline{x}) \rightarrow r(\overline{x}))$ *is an inductive invariant for* $\widehat{\delta}$ *with a Skolemization aware check.*

Proof (sketch). Construct $\widehat{\delta}$ by performing local instantiations, iteratively constructing variables corresponding to the terms used by the instantiations of $I = \forall x.\ \psi(x)$ required to prove that $I_S \wedge \delta_S \wedge (\neg I')_S$ is unsatisfiable. We translate instantiations for the Skolem constants from $(\neg I')_S$ to the corresponding Skolem constants of $(\neg \widehat{I})_S$: each existential quantifier in $\neg I'$ comes from a universal quantifier in I, which is also present in $\forall \overline{x}.\ r(\overline{x})$ which is part of $\widehat{I}$. Instantiations of higher depth are obtained through local instantiation over the terms corresponding to the base terms, i.e. the instantiation for a term $f(t_1, \ldots, t_n)$ is simulated by a local instantiation over variables $c_{t_1}, \ldots, c_{t_n}$ which were introduced for the instantiation of $t_1, \ldots, t_n$. We also conjoin to $\widehat{\delta}$ the clause $\forall \overline{x}.\ \psi'(\overline{x}) \rightarrow r'(\overline{x})$.

The proof is by contradiction: assume that $\widehat{I} \wedge \widehat{\delta} \wedge \neg \widehat{I}$ is satisfiable. Without loss of generality[6], take a Herbrand model $\widehat{\mathcal{A}}$ of this formula. From this we construct a model $\mathcal{A}$ satisfying the instantiations of $I_S \wedge \delta_S \wedge (\neg I')_S$: the domain and interpretation of constants and relations (omitting r, r') are without change. The key point is that the Skolem functions are interpreted according to the hierarchy between variables introduced through local instantiation. (Note that $(\neg I)_S$ is satisfied by $\mathcal{A}$, which reads $\mathcal{A} \models (\exists \overline{x}.\ \neg \psi'(\overline{x}))_S$, because $\widehat{A} \models (\exists \overline{x}.\ \neg r'(\overline{x}))_S$ and $\widehat{A} \models \forall \overline{x}.\ \neg r'(\overline{x}) \rightarrow \neg \psi'(\overline{x})$ (the latter is part of $\widehat{I}$ and guaranteed to hold through $\widehat{\delta}$).) □

Illustrating Example. Figure 6 illustrates the local instantiation procedure on the example of Example 3.9. Recall that the $\forall^*\exists^*$ invariant I of Example 3.9 is not provable using Bounded-Horizon of bound 1, but is provable using Bounded-Horizon of bound 2. The instrumentation presented in Figure 6 introduces three instrumentation relations to encode the existential parts of I, thereby producing the instrumented universal invariant $\widehat{I}$.

To prove the inductiveness of $\widehat{I}$, we use local instantiations in the actions `check` and `server_process_db_response`. In `server_process_db_response`, we instantiate the definition of r_3 on (id, sk_p), and assign the existential witness to c_1. Intentionally, c_1 gets the request that was sent from the server to the DB that led to the response sk_p being sent from the DB to the server. sk_p is the response that supposedly causes a violation of the invariant when the action `server_process_db_response` is executed (the instantiations are used to prove that a violation does not occur). This instantiation is of depth 1. Next, we make an instantiation of the definition of r_2 on (id, c_1) and obtain a new existential witness c_2. The use of c_1 here makes this instantiation depth 2. The `check` action includes another instantiation of depth 1, which is simply used to prove that the `abort` cannot happen (similarly to rewriting a program condition).

The reader can observe that the local instantiations introduced during the instrumentation process closely correspond to the instantiations required to prove the original $\forall^*\exists^*$ invariant I (see Section 3).

[6] The theory of equality, which is universally quantified, can be conjoined to the verification conditions, and thus the rest of the reasoning can be performed in first-order logic without equality.

```
/@  r1(x, y) ≡ ∃z. req(x, z) ∧ db(z, y)
/@  r2(x, y) ≡ ∃z. t(x, z) ∧ req(z, y)
/@  r3(x, y) ≡ ∃z. db_req(x, z) ∧ db(z, y)
/@  Invariant Î = ∀u, p. resp(u, p) → r1(u, p) ∧
/@          ∀id, q. db_req(id, q) → r2(id, q) ∧
/@          ∀id, p. db_resp(id, p) → r3(id, p) ∧
/@          ∀id, u1, u2. t(id, u1) ∧ t(id, u2) → u1 = u2
action server_process_db_response(id, p) {
  # instantiate r3 on (id, sk_p) (depth 1)
  /@  c1 := ∗;
  /@  assume r3(id, sk_p) → db_req(i, c1) ∧ db(c1, sk_p);
  # instantiate r2 on (id, c1) (depth 2)
  /@  c2 := ∗;
  /@  assume r2(id, c1) → t(id, c2) ∧ req(c2, c1);
  ...
}
action check(u, p) {
  # instantiate r1 on (u, p) (depth 1)
  /@  c3 := ∗;
  /@  assume r1(u, p) → req(u, c3) ∧ db(c3, p);
  ...
}
```

FIGURE 6. An illustration of instrumentation by local instantiations for the example of Example 3.9. The instrumentation adds three instrumentation relations r_1, r_2, r_3, and performs three local instantiations in order to prove that the invariant is inductive. Note that an instantiation depth of 2 is used, in accordance with the fact that the original invariant is provable using bound 2 but not bound 1. The complete model corresponding to this Figure appears in [add] (file `client_server_db_instr.ivy`).

It is important to note that the process of instrumentation by local instantiations discussed here is different in spirit from those of Section 4: instrumentation by local instantiation consists of almost nothing but adding existential quantifiers to the transition relation, as opposed to the condition in Definition 4.9 where we do not allow the instrumentation to add new existential quantifiers.

6. Partial Models for Understanding Non-Inductiveness

When conducting SMT-based deductive verification (e.g., using Dafny [Lei10]), 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 the user adapts the program, the invariant, or both.

```
relation pending(m, n)
init ∀m, m. ¬pending(m, n)
... # ring topology

action send_packet(n) {
  assume ring_next(n, m)
  pending.insert ((n, m))
  sent.insert (n)
}

action receive_packet(n, m) {
  assume pending(m, n)
  pending := pending \ {(m, n)}
  if m = n then
    leader.insert (n)
  else
    if n < m then
      assume ring_next(n, n0)
      pending.insert ((m, n0))
    else  # do not forward
}
```

FIGURE 7. A sketch of the leader election protocol discussed in this section (the complete program appears in [add], file `ring_leader_termination.ivy`).

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 $Ind = I \wedge \delta \wedge \neg I'$ which is used to check inductiveness. It explains the error, and guides the user towards fixing the program and/or the invariant [Lei10, FLL+02]. 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" [GM09, RTGK13]. In such cases, Bounded-Horizon instantiations can be used to present a concrete logical structure which is comprehensible to the user, 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 two examples.

6.1. **Leader Election in a Ring.** Our first example is a simple leader-election protocol in a ring [CR79], whose model is presented in Figure 7. 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, there must be an elected leader. To verify this we use a relational model of the protocol, similar to [PMP+16], and specify the property via the following formula:

$$(\exists n.\ leader(n)) \vee (\exists n_1, n_2.\ \neg sent(n_1) \vee pending(n_1, n_2)) \tag{6.1}$$

A natural attempt of proving this using an inductive invariant is by conjoining Equation (6.1) (which is not inductive by itself) with the following property (this was the authors' actual next step in proving this termination property):

$$\forall n_1.\ sent(n_1) \wedge \neg leader(n_1) \rightarrow ((\exists n_2.\ pending(n_1, n_2)) \vee (\exists n_2.\ n_1 < n_2)) \tag{6.2}$$

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.

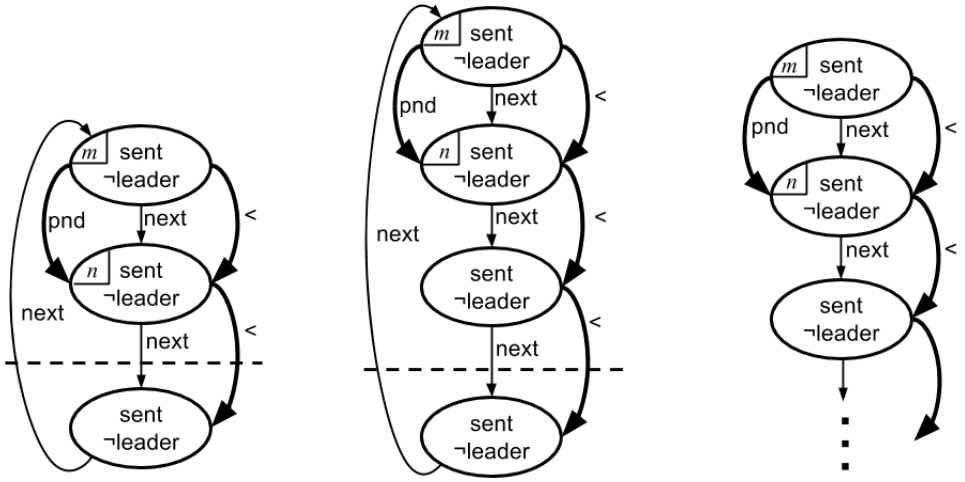

FIGURE 8. Leader-election in a ring protocol as an illustration of the use of partial models for incorrect programs and invariants. (A), (B) show partial models of bound 1 and 2, respectively, and (C) illustrates an infinite structure that explains the root cause of the non-inductiveness.

Alas, the conjunction of Equations (6.1) and (6.2) is still not an inductive invariant for the protocol (as we explain below). Since Equation (6.2) 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. 8(a) and (b) show partial models (restricted to the pre-states) obtained with bounds of 1 and 2, respectively, on this example[7]: they show (finite) rings in which all elements have advertised themselves (sent), none are considered leaders, and there is a message pending to n with the ID of m, which n would not forward as n's ID is greater than m's. Therefore, once the message is received there are no pending messages and no leader, which contradicts the safety property.

These models are not true counterexamples to induction: the sub-formula of Equation (6.2) 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 in the figure. For these, there exists a node with a higher ID, which is mandated by the invariant of Equation (6.2) (unless corresponding messages are present). Intuitively, 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.

[7]Ivy [MP18], a language and tool for the verification of distributed protocols, can visualize counterexamples with relations of arity up to 2 [PMP+16]. The dashed line marks the boundary between elements on which all $\forall^*\exists^*$ clauses holds and those on which it does not. It is meaningful when the $\forall^*\exists^*$ part consists of a single universal quantifier over the existential formula, and is readily computed automatically.

Hence, no finite counter-model exists. However, extrapolating from Figure 8(a) and (b), we can obtain the infinite model depicted in Figure 8(c). 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 (6.1) and (6.2), 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 of 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.

6.2. **Trusted Chain.** Another example which exhibits similar behavior is that of a *trusted chain*. In this example, nodes are organized in a linear total order between nodes s and t. A node is marked trusted when it receives a message. When a node processes a message it propagates the message to its successor. The safety property requires that all nodes are trusted when there are no pending messages. The invariant further states that if a node is trusted then its successor is trusted or has a pending message. Referring to the successor in the linear total order introduces $\forall^*\exists^*$ quantification[8], as well as that if a message is pending to a node after s then s is already trusted. This invariant is *not* inductive due to a counterexample with an *infinite* chain of trusted nodes and an ω untrusted node (similar to the construction in Theorem A.3). Partial models for this case would have the form of a finite chain in which all nodes are trusted apart from t and nodes lying "beyond the horizon"—meaning, $\forall^*\exists^*$ invariant was not instantiated on them—including t's predecessor.[9] In this case, the invariant can be made inductive by adding an axiom expressing an induction principle of being trusted over the order ("if s is trusted, and every successor of a trusted node is trusted, then all nodes are trusted"). This fix causes the invariant to become inductive, and this invariant is now provable using Bounded-Horizon of bound 1.

Generally, using the Bounded Horizon technique with increasing bounds produces a sequence of partial counter-models. This is unlike the usual scenario upon the solver's divergence, where the user is typically provided with no evidence of what went wrong. By analyzing this sequence of partial counter-models, the user can extrapolate to an infinite counter-model, and identify a strengthening (e.g., an axiom) that will overrule it.

7. Implementation and Evaluation

In this section, we describe our implementation of Bounded-Horizon of bound 1 and evaluate it on several correct and incorrect examples. While discussing the examples, we note that certain types of ghost code that records properties of the history of an execution can be

[8]This can be understood as saying "*for every* pair of nodes, either *exists* a node between them or...".

[9]Ivy seeks a counterexample over a domain of minimal cardinality [PMP+16], in which case t and t's predecessor would be the *only* untrusted nodes.

viewed as a special case of instrumentation, and demonstrate how the Bounded-Horizon technique circumvents the need for augmenting the program with such ghost code.

7.1. **Implementation.** We implemented a prototype of Bounded-Horizon of bound 1 on top of Z3 [DMB08] and used it within Ivy [MP18] and the framework of [IBI+13].

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 $Ind = I_S \wedge \delta_S \wedge (\neg I')_S$.[10] Let $\forall x.\ \theta$ be a subformula of Ind. If θ contains function symbol applications[11], 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 a bound of 1 on the depth of instantiations performed (instead of trying instantiations of higher and higher depth), 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 $\#\forall$. The exponential explosion is due to combinations of constants in the instantiation, a problem we defer to the solver.

Our encoding restricts the constants necessary to decide satisfiability of the formula while avoiding an exponential blowup in the translation. The task of exploring the space of possible instantiations is left for Z3's instantiation heuristics. It not immediate, however, that Z3 is guaranteed to terminate on the resultant formula, which syntactically does not fall into a decidable class (despite of encoding bounded instantiations). Fortunately, employing Model-Based Quantifier Instantiation [GM09], Z3 is guaranteed to terminate on this formula, as desired. This is because during the Model-Based Quantifier Instantiation process every instantiation of a universal formula has the same truth value in the model as an instantiation using one of the existing ground terms (constants and then BHT_1 terms). Z3's instantiation engine produces instantiations using existing terms rather than create superfluous new terms, and so must terminate on the formulas our procedure produces [Bjø].

An alternative approach of implementation is to integrate the instantiation bound with the solver's heuristics more closely (see [BRK+15]). It would also be interesting to efficiently integrate this approach with resolution based first-order theorem-provers such as Vampire [KV13].

7.2. **Examples.** We applied the procedure to the incorrect examples of Section 6, and also successfully verified several correct programs and invariants using bound 1. These examples are (the examples' code can be found in [add]):

- The client-server example of Example 3.7.
- List reverse [IBI+13], where the invariant states that the n edges ("next" pointers) are reversed. The invariant is $\forall^*\exists^*$ due to the encoding of n via its (reflexive) transitive closure n^* as explained in [IBI+13].

[10] Skolemization is performed via Z3, taking advantage of heuristics that reduce the number of different Skolem functions.

[11] This in fact implements the approximation as of Equation (3.2). The exact bound 1 per Equation (3.1) can be implemented by a more careful consideration of which universally quantified variables should be restricted, but this was not necessary for our examples.

- Learning switch [BBG+14], in which the routing tables of the switches in the network are automatically constructed through observing the source packets arriving on each switch's links. The invariant states that in every routing path, every routing node has a successor.
- Hole-punching firewall [BBG+14], in which the firewall allows a packet from an external host to enter the network only if the host is recorded as trusted by the firewall. The invariant states that if a packet from an external host then is allowed, then there previously was an internal host that contacted the external host. We explored two modeling alternatives: using a ghost history relation, or existentially quantifying over time. We elaborate on this topic and this example below.
- Trusted chain, as explained in Section 6.
- Leader election in a ring [CR79, PMP+16] with the invariant discussed in Section 6. (See Section 6 for full details.)

Derived Relations Over the History. Sometimes expressing verification conditions requires modifications to the program in which the state is augmented with additional relations, but these relations cannot be defined as derived relations over the core program relations. Often the reason is that the property of interest depends not only on the current program state, but also on previous states in the execution history. Such a scenario occurs in the example of the *hole-punching firewall*. In this example, a firewall controls packets entering and leaving the organization's network. Packets from outside the organization are allowed if they originate from a host that is considered trusted. A host $host_o$ is considered trusted if some host $host_i$ in the organization's network has previously sent a packet to $host_o$. This correctness condition depends on previous states in the execution history, and cannot be expressed in an inductive invariant without changing the vocabulary.

A common method to overcome this problem is to introduce *ghost state* to record historical information, and *ghost code* to mutate the ghost state, implementing this book-keeping. In the hole-punching firewall example, the user can add a relation $\mathit{ever\text{-}pending}(host_1, host_2)$, and add ghost code that adds the tuple $(host_1, host_2)$ to *ever-pending* whenever $host_1$ sends a packet to $host_2$.

We observe that it is sometimes possible to think of such ghost relations as standard derived relations, defined over a vocabulary in which past states are available, and view the procedure of adding appropriate ghost code as an instrumentation by a derived relation as defined in this paper (Definition 4.2). The idea is as follows:

In our first-order setting, we can lift the vocabulary to encode the time explicitly with a classical encoding (see e.g. [Aba89]), by adding a new parameter t to every relation and constant symbol. In the transition relation, references to $p(\cdot)$ are replaced by $p(t_{\text{now}}, \cdot)$ and $p'(\cdot)$ by $p(t'_{\text{now}}, \cdot)$, where t_{now} is a new constant that represents the current time. t_{now} is incremented in every transition (according to some total order on time). The transition relation also needs to include the requirement that the previous states are not modified by the current transition, meaning that $\forall t < t_{\text{now}}.\ \forall \overline{x}.\ p(t, \overline{x}) \leftrightarrow p'(t, \overline{x})$. Call this modified transition relation δ_t.

We now proceed to express the ghost relation as a derived relation over the history. In the example of the hole-punching firewall, the relation *ever-pending* can now be expressed by the derived relation $\psi(host_1, host_2) \equiv \exists t.\ t \leq t_{\text{now}} \wedge \mathit{pending}(t, host_1, host_2)$.

With this construction, we can directly use the derived relation in the inductive invariant instead of the ghost relation. Viewing the addition of the ghost relation and ghost code as a sound instrumentation, Lemma 4.5 implies that an inductive invariant for the program

with ghost state induces an inductive invariant for δ_t. This invariant is defined over the vocabulary that records the entire history, but without the ghost relation symbol.

The value of this point of view is that the results of this paper imply that in certain cases the user can prove an invariant expressed over the history using bounded instantiations with a low bound, *without resorting to ghost code manipulations*. The reasoning is as follows: If adding the ghost code does not add existential quantifiers, then augmenting the program with the ghost code can be thought of as instrumentation without additional existentials. If, additionally, the derived relation — now defined over the entire execution history — can be expressed as a combination of universal and existential properties, Theorem 4.14 applies, showing that Bounded-Horizon with a low-bound is guaranteed to prove the inductiveness of the invariant expressed for δ_t, with no need to add the ghost code.

These conditions are satisfied by the hole-punching firewall example. We manually performed the transformation of the transition relation to a vocabulary over the history in Ivy, and successfully proved the inductive invariant with Bounded-Horizon of bound 1.

7.3. **Evaluation.** In this section, we attempt to answer the following questions regarding the applicability of Bounded-Horizon of bound 1:

- Is the method sufficiently **powerful** to prove correct inductive invariants of interesting programs? (Section 7.3.1)
- Can our implementation achieve quick **termination** (with a partial counterexample) when standard (complete) methods diverge due to an infinite counterexample? (Section 7.3.2)
- Is the **overhead** associated with Bounded-Horizon on correct invariants reasonable compared to a baseline implementation (which does not bound instantiation a-priori)? (Section 7.3.3)

Table 1 compares the running time of our implementation of Bounded-Horizon of bound 1 with Z3 as a baseline. "—" means the solver did not terminate in a 60 seconds timeout. We elaborate on each of the questions referring to the results of this table.

Table 1. Experimental results.

Program	#∀	#Func	#Consts	#∀$^{\downarrow}$	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	2004 ms	83 ms	91 ms
Hole-punching firewall with ghost	15	1	18	3	354 ms	14 ms	14 ms
Hole-punching firewall ∃ time	32	2	21	3	485 ms	14 ms	14 ms
Trusted chain (correct)	27	1	19	2	435 ms	30 ms	37 ms
Leader-election in a ring (correct)	41	1	21	1	517 ms	33 ms	47 ms
Trusted chain (incorrect)	23	15	1	2	393 ms	95 ms	—
Leader-election in a ring (incorrect)	40	1	20	1	899 ms	417 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. "—" means the solver did not terminate in a 60 seconds timeout. **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. #∀$^{\downarrow}$ is the number of universally quantified variables that were restricted in the bound 1 check. Measurements were performed on a Linux 4.15.0 VM running on a 2.9GHz Intel i7-7500U CPU, except for client-server and list reverse which were measured on a 3.5GHz Intel i5-4690 CPU with 8GB RAM running Linux 3.13 x86_64.

Summary of Results. 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 ensure termination with only a slight performance penalty. Bounded-Horizon successfully terminating on incorrect examples with a partial example suggests that the Bounded-Horizon termination criterion may indeed be useful for "sat" instances on which the solver may diverge.

7.3.1. *Bound 1 Suffices for Interesting Programs and Invariants.* Bound 1 successfully proves the invariant inductive in all correct examples list in Table 1. This demonstrates in practice the power of bounded instantiations shown theoretically in Section 4.

7.3.2. *Bound 1 Terminates on Incorrect Invariants.* In this section we consider the incorrect invariants of leader election termination in a ring and trusted chain, outlined in Section 6. On these examples, our encoding of bounded instantiations allowed Z3 to terminates in seconds. In contrast, baseline Z3 does not converge in 60 seconds on either of these examples. The same happens with the first-order theorem prover Vampire [KV13], invoked on an SMTLIB2 translation of the verification condition. This demonstrates that the encoding of bounded instantiations outlined in Section 7.1 allows Z3 to terminate efficiently, although to do so the solver must in theory exhaust all possible bounded instantiations.

7.3.3. *Bound 1 Has Reasonable Overhead.* In this section we consider the correct invariants in Table 1. Both Bounded-Horizon and baseline Z3 successfully prove these inductive correct. Bounded-Horizon of bound 1 introduces an overhead in this computation; this is to be expected since the chief benefit of our approach is the termination guarantee on incorrect examples. We deem this overhead a reasonable price to ensure termination (in general the user does not know in advance whether the invariant is inductive or not before querying the solver). Comparing the columns **B1 Total** and **B1 Solve** with **Baseline Z3** in Table 1, it is apparent that most of the overhead is due to the transformation of the formula before it is sent to the solver, which may be improved by further engineering.

8. Related Work

Quantifier Instantiation. The importance of formulas with quantifier-alternation for program verification has led to many developments in the SMT and theorem-proving communities that aim to allow automated reasoning with such formulas. The Simplify system [DNS05] promoted the practical usage of quantifier triggers, which let the user affect the quantifier instantiation procedure in a clever way. Similar methods are integrated into modern SMT solvers such as Z3 [DMB08]. Recently, a method for annotating the source code with triggers has been developed for Dafny [LP16]. The notion of instantiation depth is related to the notions of matching-depth [DNS05] and instantiation-level [GBT09] 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 [GM09]. 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., [RTGK13]) consider the problem of model finding.

Local Theory Extensions and Psi-Local Theories [Sof05, IJS08, BRK$^+$15] 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 [GM09]. 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.

Finally, first-order theorem provers, such as Vampire [KV13] and SPASS [Wei01], employ other proof techniques such as resolution. These techniques are also prone to divergence on formulas with quantifier-alternation and in general unable to generate infinite counterexamples.

Natural Proofs. Natural proofs [QGSM13] 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 [HJJ+95] 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 [BGG01]. 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 6 does not fall under the Loosely-Guarded Fragment of first-order logic [Hod02] 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 [AHH13, AHH14, AHH15] defines a useful abstraction for the verification of parameterized systems. This abstraction is closely related to universally quantified invariants. An extension of this approach [AHH14] 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 [LS00, SRW02] which allows the programmers to define instrumentation relations. TVLA also employs finite differencing to infer sound update code for updating instrumentation relations [RSL10], 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 [LQ06, LQ08].

Instrumentation for Decidable Logics. The technique of instrumentation by derived relations to allow decidable reasoning is further discussed in a recent work [PLSS17]. Several variants of Paxos are proved safe using models in the decidable logic of EPR with stratified functions, whose use is enabled by instrumentation. Efficiently implementing a bounded instantiations scheme that bounds instantiations only where they are unstratified is an interesting challenge, in hope of relieving the need for instrumentation also for such cases.

9. 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, was also presented for conceptual purposes, although it is less appealing from a practical point of view.

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. Other interesting directions include further investigation into the automation of program transformations for the purpose of verification (of which instrumentation is an example), including types of ghost code, and the use of Bounded-Horizon for automatically inferring invariants with quantifier-alternation.

Acknowledgments. We would like to thank Nikolaj Bjørner, Shachar Itzhaky, and Bryan Parno for helpful discussions, Gilit Zohar-Oren for help and feedback, and the anonymous referees whose comments have improved the paper. 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], and the European Union's Horizon 2020 research and innovation programme (grant agreement No [759102-SVIS]). This research was partially supported by BSF grant no. 2012259, and by Len Blavatnik and the Blavatnik Family foundation, the Blavatnik Interdisciplinary Cyber Research Center, Tel Aviv University, the United States-Israel Binational Science Foundation (BSF) grants No. 2016260 and 2012259, and the Israeli Science Foundation (ISF) grant No. 2005/17.

References

[Aba89] Martín Abadi. The power of temporal proofs. *Theor. Comput. Sci.*, 65(1):35–83, June 1989.

[add] Full code materials. `http://www.cs.tau.ac.il/research/yotam.feldman/papers/tacas17/examples_code.zip`.

[AHH13] Parosh Aziz Abdulla, Frédéric Haziza, and Lukás 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.

[AHH14] Parosh Aziz Abdulla, Frédéric Haziza, and Lukáš Holík. Block me if you can! In *International Static Analysis Symposium*, pages 1–17. Springer, 2014.

[AHH15] Parosh Abdulla, Frédéric Haziza, and Lukáš Holík. Parameterized verification through view abstraction. *International Journal on Software Tools for Technology Transfer*, pages 1–22, 2015.

[BBG+14] Thomas Ball, Nikolaj Bjørner, Aaron Gember, Shachar Itzhaky, Aleksandr Karbyshev, Mooly Sagiv, Michael Schapira, and Asaf 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.

[BGG96] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1996.

[BGG01] Egon Börger, Erich Grädel, and Yuri Gurevich. *The classical decision problem*. Springer Science & Business Media, 2001.

[Bjø] Nikolaj Bjørner. personal communication.

[BRK+15] Kshitij Bansal, Andrew Reynolds, Tim King, Clark W. Barrett, and Thomas 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.

[CR79] Ernest Chang and Rosemary Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 22(5):281–283, 1979.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[Dij82] Edsger W. Dijkstra. From predicate transformers to predicates (Dedicated by the Tuesday Afternoon Club to C.A.R. Hoare at the occasion of his being elected Fellow of the Royal Society.). circulated privately, April 1982.

[DMB08] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS*, 2008.

[DNS05] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.

[FLL+02] Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, and Raymie 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.

[FPI+17] Yotam M. Y. Feldman, Oded Padon, Neil Immerman, Mooly Sagiv, and Sharon Shoham. Bounded quantifier instantiation for checking inductive invariants. In *TACAS*, 2017.

[GBT09] Yeting Ge, Clark W. Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. *Ann. Math. Artif. Intell.*, 55(1-2):101–122, 2009.

[GM09] Yeting Ge and Leonardo De Moura. Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In *International Conference on Computer Aided Verification*, pages 306–320. Springer, 2009.

[HHK+15] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath T. V. Setty, and Brian Zill. Ironfleet: proving practical distributed systems correct. In *Proceedings of the 25th Symposium on Operating Systems Principles, SOSP*, pages 1–17, 2015.

[HJJ+95] Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders 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.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[Hod02] Ian Hodkinson. Loosely guarded fragment of first-order logic has the finite model property. *Studia Logica*, 70(2):205–240, 2002.

[IBI+13] Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, and Mooly Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *CAV*, volume 8044 of *LNCS*, pages 756–772, 2013.

[IBR+14] Shachar Itzhaky, Nikolaj Bjørner, Thomas W. Reps, Mooly Sagiv, and Aditya 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.

[IJS08] Carsten Ihlemann, Swen Jacobs, and Viorica 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.

[Imm99] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

[IRR+04] Neil Immerman, Alexander Moshe Rabinovich, Thomas W. Reps, Shmuel Sagiv, and Greta Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL*, 2004.

[KBI+17] Aleksandr Karbyshev, Nikolaj Bjørner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham. Property-directed inference of universal invariants or proving their absence. *J. ACM*, 64(1):7:1–7:33, 2017.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 1–35, 2013.

[Lei10] K Rustan M Leino. Dafny: An automatic program verifier for functional correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 348–370. Springer, 2010.

[LP16] K. Rustan M. Leino and Clément 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.

[LQ06] Shuvendu K. Lahiri and Shaz 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.

[LQ08] Shuvendu K. Lahiri and Shaz 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.

[LS00] Tal Lev-Ami and Shmuel 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.

[MP18] Kenneth L. McMillan and Oded Padon. Deductive verification in decidable fragments with ivy. In *Static Analysis - 25th International Symposium, SAS 2018, Freiburg, Germany, August 29-31, 2018, Proceedings*, pages 43–55, 2018.

[Nel89] Greg Nelson. A generalization of dijkstra's calculus. *ACM Trans. Program. Lang. Syst.*, 11(4):517–561, 1989.

[Pad18] Oded Padon. *Deductive Verification of Distributed Protocols in First-Order Logic*. PhD thesis, Tel Aviv University, 2018.

[PLSS17] Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. Paxos made epr: Decidable reasoning about distributed protocols. *Proc. ACM Program. Lang.*, 1(OOPSLA):108:1–108:31, October 2017.

[PMP+16] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon 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.

[QGSM13] Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and Parthasarathy 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.

[Ram30] F. P. Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, s2-30(1):264–286, 1930.

[RS67] Joseph R. Shoenfield. *Mathematical Logic*. Addison-Wesley, Reading, Massachusetts, 1967.

[RSL10] Thomas W. Reps, Mooly Sagiv, and Alexey Loginov. Finite differencing of logical formulas for static analysis. *ACM Trans. Program. Lang. Syst.*, 32(6), 2010.

[RTG+13] Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstic, Morgan Deters, and Clark 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.

[RTGK13] Andrew Reynolds, Cesare Tinelli, Amit Goel, and Sava 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.

[Sof05] Viorica 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.

[SRW02] Shmuel Sagiv, Thomas W. Reps, and Reinhard Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Program. Lang. Syst.*, 24(3):217–298, 2002.

[Wei01] Christoph Weidenbach. Combining superposition, sorts and splitting. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 1965–2013. Elsevier, 2001.

Appendix A. Undecidability

For a universal formula $I \in \forall^*(\Sigma)$, the formula $I \wedge \delta \wedge \neg I'$ is in EPR (recall that δ is specified in EPR). Hence, checking inductiveness amounts to checking the 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 this section we show that checking inductiveness of $\forall^*\exists^*$ formulas is indeed undecidable, even when the transition relation is restricted to EPR. The undecidability of the problem justifies sound but incomplete algorithms for checking inductiveness, one of which is the Bounded-Horizon algorithm (defined in Section 3) which we study in this paper.

Finite and infinite structures. We begin by showing that the problem is undecidable when structures, or program states, are assumed to be finite. This is the intention in most application domains [Imm99] (including the examples in Section 7), especially when the program does not involve numerical computations. Nevertheless, in this paper we mostly concern ourselves with the problem of checking inductiveness when structures may also be infinite. This is because SMT-based deductive verification relies on proof techniques from standard first-order logic, whose semantics are defined over *general* structures, i.e. both finite and infinite. We thus establish an undecidability result for this setting as well. It is interesting to note that the discrepancy between the intended finiteness of the domain and the proof techniques, which cannot incorporate this assumption, re-emerges in Section 6.

We refer to *inductiveness over finite structures* when the validity of $I \wedge \delta \rightarrow I'$ is considered over finite structures, and to *inductiveness over general structures* when it is considered over both finite and infinite structures.

Scope of the proofs. The undecidability proofs of this section are by reductions from tiling problems. Although technically it is also possible to prove the results by a trivial reduction from the satisfiability of $\forall^*\exists^*$ formulas, since invariants for the transition relation *true* are necessarily either valid or unsatisfiable, we believe that the proofs presented here demonstrate the intuition behind the inherent difficulty of checking inductiveness of $\forall^*\exists^*$ formulas in a more profound and robust way; the reduction using the transition relation *true* leaves more room for questions about decidability of the problem w.r.t. classes of transition relations more realistic and structured than *true*. In contrast, the reductions we present here use transition systems (based on tiling problems) that we consider rather realistic, so the undecidability of checking $\forall^*\exists^*$ invariants seems inherent rather than an artifact of just degenerate transition relations.

To further provide intuition, we prove the undecidability of the overarching problem of checking not only inductiveness of a candidate I, but also that I holds initially and implies a given safety property, as is typically done when inductive invariants are used as a means to verify safety of a transition system. Formally, the problem of *checking inductive invariants for safety of transition systems* is defined as follows. Given a transition relation δ (over $\Sigma \uplus \Sigma'$), a sentence φ_0 (over Σ) describing the set of initial states, a sentence φ_P (over Σ) describing the safety property, and a sentence I (the candidate inductive invariant), the problem is to check whether $\varphi_0 \rightarrow I$ (initiation), $I \wedge \delta \rightarrow I'$ (consecution), and $I \rightarrow \varphi_P$ (safety) are valid (over finite or general structures). We will consider this problem when $I \in \forall^*\exists^*$, $\varphi_0 \in \forall^*$, $\varphi_P \in \forall^*$ and our reductions will generate instances where $\varphi_0 \rightarrow I$ and

$I \to \varphi_P$ are valid (so it only remains to check whether $I \wedge \delta \to I'$ is valid). With these restrictions, the undecidability of the problem of checking inductive invariants for safety of transition systems over general structures implies the undecidability of the problem of checking inductiveness as used elsewhere in this paper.

We now proceed to state the undecidability results and their proofs. We present different reductions for the problems of inductiveness over finite structures and over general structures. Both reductions are from the halting problem, albeit in opposite directions: Over finite structures, the invariant is inductive iff there is no halting tiling iff the machine does not halt. Over general structures, the invariant is inductive iff there is no infinite lower triangular tiling iff the machine halts. This reflects Trakhtenbrot's theorem: over finite structures, model enumeration is possible but not proof enumeration, while over general structures the situation is reversed.

A.1. **Inductiveness Over Finite Structures.**

Theorem A.1. *It is undecidable to check given $I \in \forall^*\exists^*(\Sigma)$ and $\delta \in \mathrm{EPR}(\Sigma)$ whether I is inductive for δ over finite structures.*

The proof is based on a reduction from a variant of tiling problems. We start by defining the specific tiling problem used in the proof of this theorem:

Definition A.2 [IRR$^+$04]**.** A *halting-tiling problem* consists of a finite set of tiles T with designated tiles $T_{\text{start}}, T_{\text{halt}} \in T$, along with horizontal and vertical adjacency relations $\mathcal{H}, \mathcal{V} \subseteq T \times T$. A *solution* to a halting-tiling problem is an arrangement of instances of the tiles in a finite rectangular grid ("board") such that the tile T_{start} appears in the top left position, the tile T_{halt} appears in the end of a row (the rightmost position in some row), and the adjacency relationships $\mathcal{H}, \mathcal{V}$ are respected, meaning: if a tile t_2 appears immediately to the right of t_1 it must hold that $(t_1, t_2) \in \mathcal{H}$, and if a tile t_2 appears immediately below t_1 it must hold that $(t_1, t_2) \in \mathcal{V}$.

The problem is undecidable [IRR$^+$04]. The proof is by a reduction from the halting problem: given a Turing machine we can compute a halting-tiling problem such that the problem has a solution iff the machine halts (on the empty input). In the reduction, rows represent the tape of the Turing machine as it evolves over time (computation steps). The tiles encode the location of the head and the current (control) state of the machine. The horizontal and vertical constraints ensure that successive tiled rows correspond to a correct step of the machine, and the locality of constratins is possible by the locality of computation in a Turing machine. See [BGG96] for further details.

Proof of Theorem A.1. The proof is by a reduction from non-tilability in the halting-tiling problem (Definition A.2) to the problem of checking inductive invariants for safety of a transition system over finite structures where the initiation and safety requirements are valid. We think of the transition relation δ as incrementally placing tiles in a rectangle.

Vocabulary. To express locations on the board, we use a total order for both the horizontal and vertical dimensions of the board. We add an immediate predecessor relation $j = i - 1$ which is true if $j < i$ and there is no element of the order between j, i. We use a constant 0 for the minimal element of the order. These notions can be defined using a universally

quantified formula.[12] A *location* is a pair of elements of the order, a *vertical* and *horizontal* component. We sometimes use the term *board order* to refer to the lexicographic order of pairs of elements of the order.

The transition system keeps track of the last tile placed on the board by a relation $M(i, j)$ which is true only for the last updated location. Since the placing of tiles occurs in a sequential manner we also call this board location *maximal*, and a location *active* if it comes before the maximal location in the board order. The *active area* is the set of active locations.

The state of tiles on the board is represented by a set of relations $\{T_k\}$, one for each tile type, encoding the locations on the board where a tile of type T_k is placed.

In this proof we also use a constant *max* to be an element of the total order, representing the width of the rectangle.

Transitions. In every step the transition system places a valid tile in the next board location. The next board location is considered while respecting the width of the rectangle, moving to the next row if the horizontal component of the current tile is *max*.

Placing a tile of type T_{next} on the board is done by an EPR update of the (two-vocabulary) form

$$\begin{aligned}
&\forall i, j.\ M'(i, j) \leftrightarrow ((j = 0 \wedge M(i - 1, max)) \vee (j \neq 0 \wedge j \leq max \wedge M(i, j - 1))) \\
&\forall i, j.\ T'_{\text{next}}(i, j) \leftrightarrow \big(T_{\text{next}}(i, j) \vee M'(i, j)\big) \\
&\forall i, j.\ T'_k(i, j) \leftrightarrow T_k(i, j) \qquad \forall T_k \neq T_{\text{next}}.
\end{aligned} \tag{A.1}$$

The transition system nondeterministically chooses a tile T_{next} type that respects the adjacency relationships. These relationships are with the tiles in the board location preceding the current location in the horizontal and vertical components, expressible using the immediate predecessor relation and existential quantification on these predecessors. (Note that the existential quantifiers do not need to reside in the scope of universal quantifiers — they depend only on the current location.) Because the set of tile types T is finite, expressing the allowed tile types given the two adjacent locations can be done by a quantifier-free formula. Overall the EPR formula describing a step of the system consists of a disjunction between choices for T_{next}. Each of these possible choices is described via a conjunction of the guard that makes sure that it is legal to place T_{next}, and a corresponding update to the relation that is a conjunction of the formulas in Equation (A.1).

Initial state. Initially we only have T_{start} placed in the upper-left corner, so $\forall i, j.\ T_{\text{start}}(i, j) \leftrightarrow (i = 0 \wedge j = 0)$ and $\forall i, j.\ \neg T_k(i, j)$ for every other tile type T_k.

Safety property. The safety property states that the special tile T_{halt}, is not placed on the board in the end of a row (in a *max* position) in the active area.

[12] The assumption that there *exists* a predecessor is left for the invariant to state explicitly when necessary, as this is the heart of the $\forall^*\exists^*$ quantification in the proof.

Invariant. The invariant states that in the active area we have a valid partial tiling. We require this by a $\forall^*\exists^*$ formula saying that for every tile placed in an active location (except for the maximal location) there is a successor tile, placed in the next board location, that conforms to the (local) ajdacency relations.[13] We also conjoin the safety property to the invariant.

Reduction argument. The invariant holds for the initial state, and trivially implies the safety property.

If there exists a valid tiling with T_{halt} in the end of a row, a counterexample to induction can be obtained by encoding this valid tiling in the post-state and that same tiling without T_{halt}, which is the last-placed tile, in the pre-state.

For the converse, assume that the invariant is not inductive over finite structures, i.e., there exists a finite counterexample to induction, and show that there exists a solution to the halting-tiling problem. The reasoning is as follows: A finite state satisfying the invariant induces a valid finite partial tiling (defined by the active area of the board in the structure). Since the transition system always places a tile that respects the adjacency relations, it is easy to see that a counterexample to induction must be such that the transition places T_{halt} on the board in the end of a row, and that this also induces a valid partial finite tiling in the post-state. Thus a finite counterexample to induction implies the existence of a valid finite tiling with T_{halt} in the end of a row, which is a solution to the halting-tiling problem.

Thus the invariant is inductive iff the halting-tiling problem does not have a solution. □

A.2. **Inductiveness Over General Structures.**

Theorem A.3. *It is undecidable to check given $I \in \forall^*\exists^*(\Sigma)$ and $\delta \in \mathrm{EPR}(\Sigma)$ whether I is inductive for δ over general (finite and infinite) structures.*

The proof is based on a reduction from a variant of tiling problems. We start by defining the specific tiling problem used in the proof of this theorem:

Definition A.4. *A* lower-triangular infinite-tiling problem *consists of a finite set of tiles T, along with horizontal and vertical adjacency relations $\mathcal{H}, \mathcal{V} \subseteq T \times T$. A* solution *to a lower-triangular infinite-tiling problem is an arrangement of instances of the tiles in the lower-triangular plane (i.e., a total function $\{(i,j) \in \mathbb{N} \times \mathbb{N} \mid i \leq j\} \to T$) where the adjacency relationships $\mathcal{H}, \mathcal{V}$ are respected, meaning: if a tile t_2 appears immediately to the right of t_1 it must hold that $(t_1, t_2) \in \mathcal{H}$, and if a tile t_2 appears immediately below t_1 it must hold that $(t_1, t_2) \in \mathcal{V}$.*

The problem is undecidable. The proof is by a reduction from the non-halting problem: given a Turing machine we can compute a lower-triangular infinite-tiling problem such that the problem has a solution iff the machine does not halt (on the empty input). The encoding is similar to [IRR$^+$04].

Proof of Theorem A.3. The proof is by a reduction from non-tilability in the lower-triangular infinite-tiling problem (Definition A.4) to the problem of checking inductive invariants for safety of a transition system (over general structures) where the initiation and safety requirements are valid.

[13] We specify the requirement in this forward fashion, rather than requiring that every tile has a valid predecessor, in order to easily reuse this invariant in the proof of Theorem A.3.

We construct a transition relation similar to the one in the proof of Theorem A.1, with some changes, as described below.

Discussion and motivation. To provide some intuition to the difference between the reductions, we remark that both of the proofs in this section are in essence a reduction from the halting (or non-halting) problem. The proof of Theorem A.1 encodes runs of the machine as finite tilings, and asks whether a tiling that represents a terminating computation, encoded by T_{halt}, is possible. This reduction is no longer adequate when structures may be infinite. The reason is that an infinite valid partial tiling may not correspond to reachable configurations of the Turing machine, so there may be such an infinite tiling with T_{halt} even though the Turing machine never halts.[14]

In fact, the reduction in this proof must be in the opposite direction: the invariant should be inductive iff the machine terminates, whereas in the proof of Theorem A.1 the invariant is inductive iff the machine does not terminate. This is because satisfiability is recursively-enumerable over finite structures and co-recursively-enumerable over general structures (due to the existence of proofs), which reflects on checking inductiveness through the satisfiability check of the formula $I \wedge \delta \wedge \neg I'$.

Thus, we would like to have a counterexample to induction when the machine never halts, i.e. has an infinite run. As before, runs of the machine are encoded via tiling, only that now an infinite structure can encode an infinite run of the machine. (It is not necessary that an infinite tiling represents a valid infinite run of the machine, but every infinite run can be represented by such a structure.) We would like to "detect" this situation. Our way to do this is by the observation that induction on the number of rows, or execution steps, must hold when the number of rows is finite (but unbounded), as in Theorem A.1, but does not necessarily hold when there may be an infinite number of rows. This idea is implemented by a relation P with the invariant that it is preserved under successive board locations. In an infinite structure this does not imply that P is true for all locations. A flag f is used to express a transition that is aware of P not being globally true.

Another technical detail is the lower-triangular formulation of the tiling problem, which is used to construct the infinite computation of the transition system by placing a single tile in each step.

Returning to the proof, we describe the reduction and highlight its differences from the reduction in Theorem A.1. Following the lower-triangular formulation of the tiling problem, we restrict the board order to the lower-triangular part (locations (i, j) such that $i \leq j$) and ignore other locations when considering successor in the board order.

Vocabulary. We add a relation P over board locations, and a Boolean flag (nullary predicate) f.

Transitions. In each step the transition system places a valid tile in the next board location, similar to the proof of Theorem A.1. The difference is that the criterion for moving to place tiles in the next tile is when the current location (i, j) has $i = j$ (whereas in Theorem A.1 the criterion was $j = max$).

[14] One way to construct such a tiling, using a tile in row ω of the board, is utilized in the proof that follows.

To maintain the invariant that P is preserved under successor of active tiles in the board, when we place a new tile, if P holds for the maximal location before the step, set P to true for the new maximal location.

If P does not hold for the new maximal location, turn f to *false*.

Initial state. P is true for the first location $(0, 0)$ only, and f is *true*. In this proof, initially the board is empty.

Safety property. The safety property now asserts that f is *true*.

Invariant. As before, the invariant states that the active board represents a valid partial tiling, i.e. every active tile except for the maximal one has a valid successor.

The invariant also states that P is preserved under successor of board, i.e., if (i_1, j_1) and (i_2, j_2) are successive active board locations w.r.t. the board order, then if P holds for (i_1, j_1) is must also hold for (i_2, j_2). We also conjoin the safety property to the invariant.

Reduction argument. The invariant holds for the initial state, and trivially implies the safety property.

Assume that there is no solution to the lower-triangular infinite-tiling problem, and show that the invariant is inductive. The reasoning is as follows: A state satisfying the invariant induces a partial valid tiling — either finite or infinite — over the active area of the board. Since there is no valid partial tiling with an infinite number of rows, the number of active locations must be finite (the number of columns in the active domain is bounded by the number of rows, since we are discussing lower-triangular tilings). Because P is preserved under successor of the board order, by induction on the number of locations, P must hold for the maximal location. After a transition is taken, f remains *true*. Since the transition system always places a tile that respects the horizontal and vertical ajdacency relations and sets P to true for the new maximal location, it is easy to see that the rest of the invariant is preserved by a transition as well.

For the converse direction, if there is a solution to the lower-triangular infinite-tiling problem, then there is an infinite structure encoding this tiling. The transition begins with the infinite valid tiling, with a new additional row *after* this infinite sequence of tiled rows. (Recall that the board dimensions are axiomatized using a total order; the additional row index corresponds to ordinal ω of vertical order.) We place some tile in the first column of this row as in some valid row in the tiling. Note that when placing tiles in this row we need not worry about vertical constraints, because they were expressed in a forward fashion, and this row is not a successor of any other row. The first leftmost location in the new row is set to be the maximal active one, and we set P to be *false* for this location. Note that this does not violate the invariant: P is preserved under successor of the board location, but nonetheless does not hold for the location in the additional row (it is not the successor of any location). The transition will now place a new tile and turn f to *false*, P does not hold for the current maximal location, thereby violating the invariant.

Thus the invariant is inductive iff the infinite tiling problem does not have a solution. □