Tel Aviv University
Raymond and Beverly Sackler Faculty of Exact Sciences
The Blavatnik School of Computer Science

BOUNDED QUANTIFIER INSTANTIATION FOR CHECKING INDUCTIVE INVARIANTS

by

Yotam M. Y. Feldman

under the supervision of
Prof. Mooly Sagiv
and
Dr. Sharon Shoham

Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science

2017

Abstract

Bounded Quantifier Instantiation for Checking Inductive Invariants

Yotam M. Y. Feldman
Master of Science
School of Computer Science
Tel Aviv University

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 thesis 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.

To my parents.

Acknowledgements

I would like to extend my gratitude to those without whom this thesis would not have come to be.

To my supervisor, Prof. Mooly Sagiv, whose vitality, expertise and faith in me made the difference for this work. I admire Prof. Sagiv's approach to doing research, as well as to any other task in hand.

To my supervisor, Dr. Sharon Shoham, for her invaluable guidance and constant support. Her strive for clarity, professionalism and kindness are remarkable.

To Prof. Neil Immerman, for a very special and highly enjoyable opportunity to work with, and learn from, an extraordinay expert.

To Dr. Nikolaj Bjørner, for constructive discussions, help with SMT solving internals, and much appreciated encouragement.

To Oded Padon, for his support, and for always challenging and seeking new directions, in hope of further fruitful collaboration.

To Kalev Alpernas, Asya Frumkin, Elazar Gershuni, Oded Padon, Hila Peleg and Orr Tamir, for support, feedback, and hours of grace.

To my family and friends, who had to endure a never-ending season of deadlines, provided relief, and made everything worthwile.

And to my parents, who brought me thus far.

Contents

Chapter 1

Introduction

A preliminary version of this work will appear in the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2017 [FPI+17].

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

For programs with infinite state space, proving the validity of $\{P\}C\{P\}$ is generally undecidable even when C does not include loops. Indeed, existing SMT solvers can diverge even for simple assertions and simple commands. Recent attempts to apply program verification to prove the correctness of critical system's design and code [HHK+15] identify this as the main hurdle for using program verification.

The difficulty is rooted in powerful constructs used in SMT-based verification of interesting programs. Prominent among these constructs are arithmetic and other program operations modeled using background theories, and logical quantifiers. In this thesis we target the verification of applications in which the problem can be modeled without interpreted theories. This is in line with recent works that show that although reasoning about arithmetic is crucial for low-level code, in many cases the verification of high-level programs and designs can be performed by reasoning about quantification in uninterpreted theories. Specifically, the decidable Effectively Propositional logic (EPR) has been successfully applied to domains such as linked-list manipulation [IBI+13], Software-Defined Networks [BBG+14] and some distributed protocols [PMP+16]. Without interpreted theories it remains to address the complications induced by the use of quantifier alternation.

In the presence of quantifier alternation, the solver's ability to check assertions is hindered by the following issues: (1) an infinite search space of proofs that needs to be explored for correct assertions, a problem which is sometimes manifested in matching loops [DNS05], and (2) a difficulty of finding counterexamples for invalid assertions, notably when counterexamples may be of infinite size. Current SMT techniques often fail to produce models of satisfiable quantified formulas [GM09, RTG$^+$13], which is somewhat unfortunate since one of the main values of program verification is early detection of flaws in designs and programs. The existence of infinite counterexamples is a major complication as they are difficult to find. In uninterpreted domains, infinite counterexamples usually do not indicate a real violation of the verification conditions and are counterintuitive to programmers, yet render assertions invalid in the context of general first-order logic (on which SMT proof techniques are based). Hence infinite counter-models pose a real problem in the verification process.

Previous work on EPR [IBI$^+$13, BBG$^+$14, PMP$^+$16] used universally quantified invariants with programs expressed by $\exists^*\forall^*$ formulas[1], in which case checking inductive invariants is decidable, hence problems (1) and (2) do not occur. In particular, EPR enjoys the finite-model property and so counterexamples are of finite size. EPR programs are in fact Turing-complete [PMP$^+$16], but universal invariants are not always sufficient to express the required program properties.

For example, [HHK$^+$15] describes a client server scenario where the invariant is "For every reply message sent by the server, there exists a corresponding request message sent by a client". (See Example 1 for further details.) This invariant is $\forall^*\exists^*$ and thus leads to verification conditions with quantifier alternation. This kind of quantifier alternation may lead to divergence of the solver as problems (1) and (2) re-emerge.

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

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

We consider instantiations that are *bounded in depth* of terms. Bounded instantiations trivially prevent divergence while maintaining soundness. Although for a given bound the technique is not complete, i.e. unable to prove every correct invariant, we provide completeness guarantees by comparing bounded instantiations to the method of *instrumentation*, a powerful technique implicitly employed in previous works [IBI$^+$13, KBI$^+$15, PMP$^+$16]. Instrumentation tackles a $\forall^*\exists^*$invariant by transforming the program in a way that allows the invariant to be expressed in a universal form, and, accordingly, makes the verification conditions fall in EPR. We show that for invariants that can be proven using a typical form of instrumentation, bounded instantiations of a small bound are also complete. Namely, they are sufficiently powerful to prove the original program without modifications and in a fully automatic way. This is encouraging since instrumentation is labor-intensive and error-prone while bounded instantiations are

[1]Automated tools that extract EPR transition relation from code exist for C code manipulating linked lists [IBI$^+$13, IBR$^+$14, KBI$^+$15] and for the modeling language RML [PMP$^+$16] which is Turing-complete.

completely automatic.

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

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

Outline.. The rest of the thesis is organized as follows: Chapter 2 provides some background and presents required notation. Chapter 3 sets the scene by showing the undecidability of the problem of checking inductiveness. In Chapter 4 we define the Bounded-Horizon algorithm and discuss its basic properties. Chapter 5 defines the concept of instrumentation as used in this work, and shows that Bounded-Horizon with a low bound is at least as powerful. Chapter 6 relates instrumentation to bounded instantiation in the converse direction, showing that other forms of instrumentation can simulate quantifier instantiation of higher depth. In Chapter 7 we show how bounded instantiations can be used to tackle the problem of infinite counterexamples to induction when the verification conditions are not valid. Chapter 8 describes our implementation of Bounded-Horizon of bound 1, and provides initial evaluation of its ability to prove some examples correct using bound 1. Chapter 9 discusses related work, and Chapter 10 concludes.

Chapter 2

Preliminaries

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

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

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

Inductive Invariants. A first-order sentence I over Σ is an *inductive invariant* for δ if $I \wedge \delta \to I'$ is valid, or, equivalently, if $I \wedge \delta \wedge \neg I'$ is unsatisfiable [1], where I' results from substituting every constant and relation symbol in I by its primed version (i.e. $I' \in \text{FOL}(\Sigma')$).

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 $\mathcal{A}$ s.t. $\mathcal{A} \models I \wedge \delta \wedge \neg I'$.

Skolemization. Let $\varphi(z_1, \ldots, z_n) \in \text{FOL}(\Sigma)$. The *Skolemization* of φ, denoted φ_S, is a universal formula over $\Sigma \uplus \Sigma_S$, where Σ_S consists of fresh constant symbols and function symbols, obtained as follows. We first convert φ to negation normal form (NNF) using the standard rules. For every existential quantifier $\exists y$ that appears under the scope of the universal quantifiers $\forall x_1, \ldots, \forall x_m$, we introduce a fresh function symbol $f_y \in \Sigma_S$ of arity $n + m$. We replace each bound occurrence of y

[1]In this thesis, 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.

by $f_y(z_1, \ldots, z_n, x_1, \ldots, x_m)$, and remove the existential quantifier. If $n + m = 0$ (i.e., φ has no free variables and $\exists y$ does not appear in the scope of a universal quantifier) a fresh constant symbol is used to replace y. It is well known that $\varphi_S \rightarrow \varphi$ is valid and φ_S and φ are equi-satisfiable.

Chapter 3

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 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 chapter we show that indeed checking inductiveness of $\forall^*\exists^*$ formulas is 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 Chapter 4) which we study in this thesis.

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 Chapter 8), especially when the program does not involve numerical computations. However, in this thesis we will 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 is defined over *general* structures, i.e. both finite and infinite. We thus show 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, that cannot incorporate this assumption, will re-emerge in Chapter 7.

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

Scope of the proofs. The undecidability proofs of this chapter 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* necessarily are 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 way.

To further provide intuition, in the proofs we prove the undecidability of a closely related problem, that of checking *inductive invariants for safety of transition systems*. Given a transition relation δ (over $\Sigma \uplus \Sigma'$), a sentence φ_0 (over Σ) describing the set of initial states and a sentence φ_P (over Σ) describing the safety property, 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 $I \in \forall^*\exists^*$, $\varphi_0 \in \forall^*$, $\varphi_P \in \forall^*$. We will consider instances where $\varphi_0 \rightarrow I$ and $I \rightarrow \varphi_P$ are valid, and it only remains to check whether $I \wedge \delta \rightarrow I'$ 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 thesis.

3.1 Inductiveness Over Finite Structures

Theorem 1. *It is undecidable to check given $I \in \forall^*\exists^*$ and $\delta \in$ EPR 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 1 ([IRR+04]). *A* halting tiling problem *consists of a finite set of tiles T with designated tiles $T_{start}, T_{halt} \in T$, 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 line (the rightmost position in some line), 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 under 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). Additionally the tiles encode the location of the head and the current (control) state of the machine. The horizontal and vertical constraints impose that successive tiled rows correspond to a correct step of the machine, and their locality is due to the locality of computation in a Turing machine. For further details see [BGG96].

Proof of Theorem 1. The proof is by a reduction from non-tilability in the halting tiling problem (Definition 1) 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. The two dimensions of the rectangle are encoded by two total orders: a horizontal order and a vertical order. When the order is clear from the context, we use $i - 1$ as a shorthand for the predecessor of i in the order, and 0 as a shorthand for the minimal element of the order. *max* is a constant axiomatized to be the maximal element of the horizontal order. We sometimes refer to the lexicographic order of the vertical and horizontal orders as the *board 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.

Transitions. In every step the transition system places a valid tile in the next board location. Placing a tile of type T_{next} on the board is done by an EPR update of the (two-vocabulary) form

$$\begin{aligned} &\forall i,j.\ M'(i,j) \leftrightarrow ((j=0 \wedge M(i-1,\mathit{max}) \vee (j \neq 0 \wedge M(i,j-1))) \\ &\forall i,j.\ T'_{\text{next}}(i,j) \leftrightarrow \big(T_{\text{next}}(i,j) \vee M'(i,j)\big) \\ &\forall i,j.\ T'_k(i,j) \leftrightarrow T_k(i,j) \qquad \forall T_k \neq T_{\text{next}}. \end{aligned} \tag{3.1}$$

The transition system nondeterministically chooses a tile type that respects the adjacency relationships. These relationships are with the tile in the board location immediately before the current location in the horizontal and vertical orders, which is EPR expressible. 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 (3.1).

Initial state. Initially we 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 line (in a *max* position) in the active area.

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 there is a successor tile, placed in the next board location , that conforms to the (local) tiling rules.[1] We also add the safety property to the invariant.

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

[1] We specify the requirement in this forward fashion, rather than requiring that every tile has a valid predecessor, in order to reuse this invariant in the proof of Theorem 2.

If there exists a valid tiling with T_{halt} in the end of a line, a counterexample to induction can be obtained by encoding this valid tiling in the post-state and that same tiling without the last T_{halt} 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 horizontal and vertical constraints, it is easy to see that a counterexample to induction must place T_{halt} on the board in the end of a line, 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 line, which is a solution to the halting tiling problem.

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

3.2 Inductiveness Over General Structures

Theorem 2. *It is undecidable to check given $I \in \forall^*\exists^*$ and $\delta \in$ EPR 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 2 ([BGG96]). *An* 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 an infinite-tiling problem is an arrangement of instances of the tiles in the entire plane (i.e. a total function $\mathbb{N} \times \mathbb{N} \to T$) 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 under t_1 it must hold that $(t_1, t_2) \in \mathcal{V}$.*

The problem is undecidable [BGG96]. The proof is by a reduction from the non-halting problem: given a Turing machine we can compute an 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 the one from Definition 1.

Proof of Theorem 2. The proof is by a reduction from non-tilability in the infinite-tiling problem (Definition 2) to the problem of checking inductive invariants for safety of a transition system (over general structures) where the initiation and safety requirements are valid.

We construct a transition relation similar to the one in the proof of Theorem 1, with some additions, 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 chapter are in essence a reduction from the halting (or non-halting) problem. The proof of Theorem 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 no longer holds when structures may be infinite. The reason is that an infinite valid partial tiling may not correspond to reachable configuration of the Turing machine, so there may be such an infinite tiling with T_{halt} even though the Turing machine never halts[2].

In fact, the reduction in this proof must be in the opposite direction: the invariant should be inductive iff the machine does not terminate, whereas in the proof of Theorem 1 the invariant is inductive iff the machine does 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 $I \wedge \delta \wedge \neg I'$.

We thus want 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 1, but does not necessarily hold when there may be an infinite number of rows. This idea is implemented by a relation H with the invariant that it is preserved under successive rows. In an infinite structure this does not imply that H is true for all rows. A flag f is used to express a transition that is aware of H not being globally true.

A technical complication arises because to make sure that the infinite number of rows truely represents an infinite number of computation steps, we also need to make sure that the number of columns is infinite (representing the entire tape). This is done using a relation P that "detects" the infinite number of columns similarly to H. We are assured in the existence of an infinite execution when both H and P "detect" infinity of the tiling, and we turn the flag f to *false* to express this.

Returning to the proof, the new elements of the transition system in the current proof are as follows:

Vocabulary. We add a relation H over the row indices (elements of the vertical order), a relation P over board locations, and a Boolean flag (nullary predicate) f.

Transitions. As before, in each step the transition system places a valid tile in the next board location (see the proof of Theorem 1). To maintain the invariant that H is preserved under successor of active rows (see below), when we place a new tile, if H holds for the current line (the element of the vertical order of the maximal active location) after placing the tile, set H to true for the row after (next element of the vertical order) as well. (For ease of expression, in this proof we let the transition system place the first tile itself.)

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

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

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

Initial state. Initially H is true for the first row (0 of the vertical order) only, P is true for the first location of every row (0 of the horizontal order with any value of the vertical order) 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.

In addition, the invariant states that H is preserved under successor of active board rows, i.e.: If i_1, i_2 are active rows — meaning $i_1, i_2 \leq i$ where $M(i, j)$ — and i_2 is the successor of i_1 w.r.t. the vertical order, then if H holds for i_1 it must also hold for i_2.

The invariant also states that P is preserved under successor of board locations in any line, i.e.: if (i, j_1) and (i, j_2) are successive active board locations, then if P holds for (i, j_1) is must also hold for (i, j_2). We also add 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 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 infinite valid partial tiling, the number of rows in the active area must be finite, or the number of (active) columns must be finite. If the number of active rows is finite, because H is preserved under successor for active rows, we have that H must hold for the current row, by induction on the number of rows in the active area. If the number of columns is finite, because P is preserved under successor in every line, we have that P must hold for the maixmal location, by induction on the number of columns (in the current line). Either way, after a transition is taken, f remains *true*. Since the transition system always places a tile that respects the horizontal and vertical ajdacency relations, sets H to true for the next row and 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 infinite tiling problem, then there is an infinite structure encoding this tiling. (We take *max* to be an infinite ordinal of the horizontal order, to allow an infinite number of columns.) From this we can construct a counterexample to induction: the transition begins with the infinite valid tiling, with a new valid row in an additional row *after* this infinite sequence of tiled rows. (Recall that the board dimensions are axiomatized to be total orders; such a tile is placed in a location corresponding to ordinal ω of the vertical order.) We can tile this row exactly the as some other arbitrary line in the tiling and 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. We set H to be *false* for this additional row.

We also add two successive columns (successive elements of the horizontal order) *after* the infinite tiling (such elements correspond to ordinals ω and $\omega + 1$ of the horizontal order). Since the constraints were expressed in a forward fashion, there are no horizontal constraints on the first of these columns, as it is not the successor of any column. Therefore we use two successive tiled columns from the valid tiling and use the same tiling for the new columns. We set P to be *false* for the location in these columns in the new row.

The location in the newly added row and the first new column (i.e. (ω, ω)) is set to be the maximal active one. Note that this does not violate the invariant: H is preserved under successor of active rows, but nonetheless does not hold for all active rows (in essence, the induction fails). Similarly, P is preserved under successor of columns in the last line, but nonetheless does not hold for all columns. The transition will now place a new tile and turn f to *false*, because H does not hold for the current line and 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. $\square$

Chapter 4

Bounded-Horizon

In this chapter, 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.

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

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

Definition 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. 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*

$$\begin{aligned}
&\mathrm{BHT}_0 = C \\
&\mathrm{BHT}_k = \mathrm{BHT}_{k-1} \cup \{f(t_1, \dots t_m) \mid f \in F,\ m = \mathit{Arity}_f,\ t_1, \dots, t_m \in \mathrm{BHT}_{k-1}\}.
\end{aligned}$$

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 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.

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 $\mathit{Ind} = I_S \ \wedge\ \delta_S \ \wedge\ (\neg I')_S$, and checks if the set

$$\big\{\mathit{Ind}[\bar{t}\,] \mid \bar{t} \in \mathrm{BHT}_k,\ \mathrm{depth}(\mathit{Ind}[\bar{t}\,]) \leq k\big\} \tag{4.1}$$

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

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

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

$$\Big(\bigwedge_{\bar{t} \in \mathrm{BHT}_{k-1}} I_S[\bar{t}\,]\Big) \wedge \Big(\bigwedge_{\bar{t} \in \mathrm{BHT}_k} \delta_S[\bar{t}\,]\Big) \wedge \Big(\bigwedge_{\bar{t} \in \mathrm{BHT}_k} (\neg I')_S[\bar{t}\,]\Big). \tag{4.2}$$

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

Lemma 1 (Soundness)**.** *For every $k \in \mathbb{N}$, Bounded-Horizon with bound k is sound, i.e., if 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. □

Example 1. Figure 4.1 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.

```
$req := \emptyset$; $resp := \emptyset$; $match := \emptyset$;
action new_request($u$) {
  $q :=$ new request;
  $req := req \cup \{(u, q)\}$
  /@   $r := r \cup \{(u, y) \mid match(q, y)\}$
}
action respond($u, q$) {
  assume $req(u, q)$;
  $p :=$ new response;
  $match := match \cup \{(q, p)\}$;
  /@   $r := r \cup \{(x, p) \mid req(x, q)\}$
  $resp := resp \cup \{(u, p)\}$
}
```

```
action check($u, p$) {
  if $resp(u, p) \land \forall q.\ req(u, q) \to \neg match(q, p)$
  /@ $\hookrightarrow$ if $resp(u, p) \land \neg r(u, p)$
    then abort
}

Invariant $I = \forall u, p.\ resp(u, p) \to$
                    $\exists q.\ req(u, q) \land match(q, p)$
/@   $r(x, y) \equiv \exists z.\ req(x, z) \land match(z, y)$
/@   Invariant $\widehat{I} = \forall u, p.\ resp(u, p) \to r(u, p)$
```

Figure 4.1: Example demonstrating a $\forall^*\exists^*$ invariant that is provable with bound 1. The reader should first ignore the instrumentation code denoted by /@ (see Example 3). 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`).

We first explain this example ignoring the annotations denoted by "/@". The system state is modeled using three binary relations. The *req* relation stores pairs of users and requests, representing requests sent by users. The *resp* relation similarly stores pairs of users and replies, representing replies sent back from the server. The *match* relation maintains the correspondence between a request and its reply.

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

A natural inductive invariant for this system is

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

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

Lemma 2 (Completeness for some k)**.** *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 and compactness there is a finite set S of instantiations that is unsatisfiable. Take k to be the maximal depth of instantiations in S. □

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

```
$req := \emptyset$; $resp := \emptyset$;
$db\_req := \emptyset$; $db\_resp := \emptyset$; $t := \emptyset$;
action new_request($u$) {
  $q :=$ new request;
  $req := req \cup \{(u, q)\}$
}
action db_recv_request($id, q$) {
  assume $db\_req(id, q)$;
  $p :=$ DB response ensuring $db(q, p)$;
  $db\_resp := db\_resp \cup \{(id, p)\}$
}
action check($u, p$) {
  assume $resp(u, p)$;
  if $\forall q.\ req(u, q) \to \neg db(q, p)$
    then abort
}

action server_recv_request($u, q$) {
  assume $req(u, q)$;
  $id :=$ new DB request id;
  $t := t \cup \{(id, u)\}$;
  $db\_req := db\_req \cup \{(id, q)\}$
}

action server_recv_db_response($id, p$) {
  assume $db\_resp(id, p)$;
  $resp := resp \cup \{(x, p) \mid t(id, x)\}$
}
```

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

Figure 4.2: 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`).

Example 2. An example of a program and an inductive invariant for which a bound of 0 or 1 is insufficient appears in Figure 4.2. 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 nested instantiation. In this example, a bound of 2 manages to prove inductiveness. This example can be lifted to require an even larger depth of instantiations 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 only with a small fraction of the domain, namely elements pointed to by program variables (that correspond to logical constants in the vocabulary).

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

This is the case in Example 1. Additional examples are listed in Chapter 8. The example of Figure 4.2 itself also admits a different invariant that is provable by bound 1.

Chapter 5

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. Chapter 8 demonstrates the method's power on several interesting programs we verified using Bounded-Horizon of bound 1.

5.1 Instrumentation

We present our view of the instrumentation procedure used in previous works [IBI$^+$13, KBI$^+$15, PMP$^+$16] to eliminate the need for quantifier-alternation, thus reducing the verification task to a decidable fragment. The procedure begins with a program that induces a transition relation $\delta \in \exists^*\forall^*(\Sigma \cup \Sigma')$. The purpose of instrumentation is to modify δ into another transition relation $\widehat{\delta}$ that admits an inductive invariant with simpler quantification (e.g., universal, in which case it is decidable to check). We note that instrumentation is generally a manual procedure. For simplicity, we describe the instrumentation process informally, but provide the semantic soundness requirement in Definition 6. The instrumentation procedure consists of the following three steps:

1. Identify a formula $\psi(\overline{x}) \in \text{FOL}(\Sigma)$ (usually ψ will be existential) that captures information that is needed in the inductive invariant. Extend the vocabulary with an instrumentation relation $r(\overline{x})$ that intentionally should capture the derived relation defined by $\psi(\overline{x})$. Let $\widehat{\Sigma} = \Sigma \cup \{r\}$ denote the extended vocabulary[1].
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

[1]It is also possible to instrument the program with constants rather than relations. This can be emulated by adding a unary relation $c(x)$ representing the constant, and adding the assumption that c contains exactly one element to the invariant. This is also aligned with the conditions of Theorem 6.

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 3. In the example of Figure 4.1, to achieve a universal invariant we add an instrumentation relation *r* defined by $r(x, y) \equiv \exists z.\ \mathit{req}(x, z) \wedge \mathit{match}(z, y)$ (step 1). The simple form of ψ allows us to obtain precise update code, which appears as annotations marked with /@ in lines that mutate *req* and *match* (step 2). We also replace the `if` condition in the action `check` by an equivalent condition that uses *r* (step 3). The line marked with /@ $\hookrightarrow$ in the `check` action replaces the line above it. The resulting program has the invariant $\widehat{I} = \forall u, p.\ \mathit{resp}(u, p) \rightarrow r(u, p)$, which is universal.

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

Definition 6 (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* $\big(\forall \overline{x}.\ r(\overline{x}) \leftrightarrow \psi(\overline{x}) \ \wedge\ \delta\ \wedge\ \forall \overline{x}.\ r'(\overline{x}) \leftrightarrow \psi'(\overline{x})\big) \rightarrow \widehat{\delta}$ *is valid, or equivalently,* $\delta \rightarrow \widehat{\delta}[\psi/r, \psi'/r']$ *is valid.*

Definition 6 ensures that the instrumented program includes at least all the behaviors of the original program, when r is interpreted according to ψ. Thus, if the instrumented program is safe, then it is sound to infer that the original program is safe.

Remark 1. 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 meaning that r is updated in a way that is consistent with its interpretation as ψ.

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

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

Proof. $\widehat{I} \wedge \widehat{\delta} \rightarrow \widehat{I}'$ is valid, thus, so is $(\widehat{I} \wedge \widehat{\delta} \rightarrow \widehat{I}')[\psi/r, \psi'/r']$. $\widehat{\delta}$ is a sound instrumentation for δ, so (using Definition 6) $I \wedge \delta \rightarrow I'$ is valid. □

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

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

Definition 7 (Existential Naming). *Let* $\widehat{\delta} = \exists z_1, \dots z_m.\ \varphi(z_1, \dots, z_m)$ *where* $\varphi \in \forall^*(\widehat{\Sigma}, \widehat{\Sigma}')$. *An* existential naming η *for* $(\widehat{\delta}, \delta)$ *is a mapping* $\eta : \{z_1, \dots, z_m\} \to$ *const*$[\delta_S] \cup$ *const*$[\widehat{\delta}_S]$. *We define* $\eta(\widehat{\delta})$ *to be* $\varphi[\eta(z_1)/z_1, \dots, \eta(z_m)/z_m]$.

An existential naming provides a Skolemization procedure which uses existing constants rather than fresh ones. If such η exists, it maps the (Skolemized) existential quantifiers in $\widehat{\delta}$ to their counterparts in δ. For example, the instrumentation in Figure 4.1 results in $\widehat{\delta}$ that has an existential naming w.r.t. the original δ. Note that it is possible that $\widehat{\delta}$ has in fact *fewer* existential quantifiers than δ, for example due to the rewriting of conditions (as happens in the example of Figure 4.1 — see the `if` statement in action `check`).

Definition 8 (Instrumentation Without Additional Existenials). $\widehat{\delta}$ *is a* sound instrumentation without additional existentials *for* δ *if there exists an existential naming* η *such that* $\delta_S \to \eta(\widehat{\delta})[\psi/r, \psi'/r']$ *is valid.*

5.2 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 Figure 4.1, which admits an instrumentation that transforms the invariant to a universal invariant (see Example 3) in a form that matches Theorem 4, 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 inductivenessthe 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 Figure 4.2[2].

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 8) can be used to simulate quantifier instantiation of an arbitrary depth. This topic is explored in Chapter 6.

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 chapter.

Theorem 4 and Theorem 5 consider $I \in \forall^*\exists^*$ that is transformed to $\widehat{I} \in \forall^*$. In Theorem 4 we show that a bound of 1 suffices to prove that I is inductive for δ when the instrumentation defining formula

[2]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 (only sketched for simplicity) do not have any instrumentation without additional existential quantifiers that transforms the invariant to a universal form.

$\psi \in \exists^*$ and the instrumentation relation r appears only positively in $\widehat{I}$, or when $\psi \in \forall^*$ and r appears only negatively in $\widehat{I}$. This is an attempt to explain the success of bound 1 instantiations in proving our examples (see Chapter 8). In Theorem 5 we show that a bound of 2 suffices in the more general setting of $\psi \in AF$ (with no restriction on appearances of r in $\widehat{I}$).

Theorem 6 considers a generalization to I that is 1-alternation and transformed to $\widehat{I} \in$ AF. We show that a bound of 2 suffices in this case.

The rest of the chapter is devoted to proofs of these claims. The main points in the proof are as follows: Assume for the sake of contradiction that I cannot be shown to be inductive for δ by Bounded-Horizon of a low bound, and take a counterexample to induction of the instantiated I (see Equation (5.2) in the proof of Lemma 4). By the assumption that $\widehat{\delta}$ is an instrumentation without additional existentials for δ, we can utilize properties of substitution to obtain a counterexample to induction for the instantiated $\widehat{I}$ w.r.t. $\widehat{\delta}$ (see Equation (5.5)). By the assumption that $\widehat{I} \in \forall^*$ and $\widehat{\delta} \in \exists^*\forall^*$, we can use complete instantiation to argue that we have obtained a true counterexample to induction of $\widehat{I}$ w.r.t. $\widehat{\delta}$ (see Equation (5.9)), in contradiction to the premise.

Remark 2. The results of this section also apply when multiple instrumentation relations $\psi_1, \ldots, \psi_t \in$ FOL(Σ) are simultaneously substituted instead of the relation symbols $r_1, \ldots, r_t$ in $\widehat{\delta}$ and $\widehat{I}$.

5.2.1 Power for $\forall^*\exists^*$ Invariants

We now establish that low bounds are sufficient for the Bounded-Horizon check when $\widehat{I} \in \forall^*(\widehat{\Sigma})$ and $I \in \forall^*\exists^*$. To do so, we first prove the following lemma.

Lemma 4. *Let $\widehat{\delta}$ be a sound instrumentation of δ without new existentials and with naming η. Write $\widehat{I} = \forall \overline{x}.\ \widehat{\alpha}(\overline{x})$ where $\widehat{\alpha} \in$ QF$(\widehat{\Sigma})$ and let $\alpha = \widehat{\alpha}[\psi/r]$. Then,*

$$\Big(\bigwedge_{\overline{c} \in C^n} \alpha(\overline{c})\Big) \ \wedge\ \delta_S \ \wedge\ (\neg I')_S \tag{5.1}$$

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, i.e., there exists a structure $\mathcal{A}_0$ such that,

$$\mathcal{A}_0 \models \Big(\bigwedge_{\overline{c} \in C^n} \alpha(\overline{c})\Big) \ \wedge\ \delta_S \ \wedge\ (\neg I')_S\ . \tag{5.2}$$

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

$$\mathcal{A}_1 \models \Big(\bigwedge_{\overline{c} \in (C \cup \widehat{C})^n} \alpha(\overline{c})\Big) \ \wedge\ \delta_S \ \wedge\ (\neg I')_S \tag{5.3}$$

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. Thus $\alpha(\overline{c})$ holds for the new constants as well.

Removing some conjuncts from Equation (5.3), we get,

$$\mathcal{A}_1 \models \big(\bigwedge_{\overline{c} \in \widehat{C}^n} \alpha(\overline{c}) \big) \ \wedge \ \delta_S \ \wedge \ (\neg I')_S \,. \tag{5.4}$$

By assumption (Definition 8), it follows that,

$$\mathcal{A}_1 \models \big(\bigwedge_{\overline{c} \in \widehat{C}^n} \alpha(\overline{c}) \big) \ \wedge \ \eta(\widehat{\delta})[\psi/r, \psi'/r'] \ \wedge \ (\neg I')_S. \tag{5.5}$$

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,

$$\mathcal{A}_1 \models \Big(\big(\bigwedge_{\overline{c} \in \widehat{C}^n} \widehat{\alpha}(\overline{c}) \big) \ \wedge \ \eta(\widehat{\delta}) \ \wedge \ (\neg \widehat{I}')_S \Big)[\psi/r, \psi'/r']. \tag{5.6}$$

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,

$$\widehat{\mathcal{A}} \models \big(\bigwedge_{\overline{c} \in \widehat{C}^n} \widehat{\alpha}(\overline{c}) \big) \ \wedge \ \eta(\widehat{\delta}) \ \wedge \ (\neg \widehat{I}')_S. \tag{5.7}$$

Since the formula in Equation (5.7) is universal, it 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 $\widehat{C} = \mathrm{const}[\eta(\widehat{\delta}) \wedge (\neg \widehat{I}')_S]$). Thus,

$$\widehat{\mathcal{A}}_{|\widehat{C}} \models \big(\forall \overline{x}.\ \widehat{\alpha}(\overline{x}) \big) \ \wedge \ \eta(\widehat{\delta}) \ \wedge \ (\neg \widehat{I}')_S. \tag{5.8}$$

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

$$\widehat{\mathcal{A}}_{|\widehat{C}} \models \widehat{I} \ \wedge \ \widehat{\delta} \ \wedge \ \neg \widehat{I}'. \tag{5.9}$$

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

The following results are corollaries of Lemma 4.

Theorem 4. *Let $\widehat{I} \in \forall^*$. Assume $\psi \in \exists^*$ and r appears only positively in $\widehat{I}$, or $\psi \in \forall^*$ and r appears only negatively in $\widehat{I}$. Then $I = \widehat{I}[\psi/r]$ is inductive for δ with Bounded-Horizon of bound* 1. *(Note that $I \in \forall^*\exists^*$.)*

Proof. Let $\widehat{I} = \forall \overline{x}.\ \widehat{\alpha}(\overline{x})$ where $\widehat{\alpha} \in \mathrm{QF}$. In both cases of the claim $\alpha = \widehat{\alpha}[\psi/r] \in \exists^*$, and so all the universal quantifiers in I are those of $\widehat{I}$. This implies that the satisfiability check of Lemma 4 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 $\alpha(\overline{x}) = \exists y_1, \dots, y_m.\ \theta(\overline{x}, y_1, \dots, y_m)$ where $\theta \in$ QF, and let

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

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

$$\Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_0} \alpha_S[\overline{t}]\Big) \wedge \Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_1} \delta_S[\overline{t}]\Big) \wedge \Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_1} \left(\neg I'\right)_S[\overline{t}]\Big) \tag{5.10}$$

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

$$\Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_0} \alpha_S(\overline{t})\Big) \wedge \Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_1} \delta_S[\overline{t}]\Big) \wedge \Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_1} \left(\neg I'\right)_S[\overline{t}]\Big) \tag{5.11}$$

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

$$\mathcal{A}_{|_{\mathrm{BHT}_1}} \models \Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_0} \alpha_S(\overline{t})\Big) \wedge \delta_S \wedge \left(\neg I'\right)_S \tag{5.12}$$

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

$$\mathcal{A}_{|_{\mathrm{BHT}_1}} \models \Big(\bigwedge_{\overline{t}\in \mathrm{BHT}_0} \alpha(\overline{t})\Big) \wedge \delta_S \wedge \left(\neg I'\right)_S. \tag{5.13}$$

Recall that $\mathrm{BHT}_0 = \mathrm{const}[\delta_S \wedge \left(\neg I'\right)_S]$. Therefore, Equation (5.13) can be rewritten as

$$\mathcal{A}_{|_{\mathrm{BHT}_1}} \models \Big(\bigwedge_{\overline{c}\in C^n} \alpha(\overline{c})\Big) \wedge \delta_S \wedge \left(\neg I'\right)_S \tag{5.14}$$

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

By Lemma 4 this is a contradiction to the assumption that $\widehat{I}$ is inductive w.r.t. $\widehat{\delta}$, and the claim follows. □

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

Proof. As before, Let $\widehat{I} = \forall\overline{x}.\ \widehat{\alpha}(\overline{x})$ where $\widehat{\alpha} \in$ QF. $\psi \in$ AF implies that $\alpha = \widehat{\alpha}[\psi/r] \in AF$. Let

$$\begin{aligned}\alpha(\overline{x}) = &\big(\forall\overline{v_1}\theta_{1,1}(\overline{x},\overline{v_1}) \vee \exists\overline{z_1}\theta_{1,2}(\overline{x},\overline{z_1})\big) \wedge \dots\\ &\wedge\big(\forall\overline{v_r}\theta_{r,1}(\overline{x},\overline{v_r}) \vee \exists\overline{z_r}\theta_{r,2}(\overline{x},\overline{z_r})\big)\end{aligned} \tag{5.15}$$

where $\theta_{1,1}, \theta_{1,2}, \dots, \theta_{r,1}, \theta_{r,2} \in$ QF.

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

For brevity denote

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

and let $\overline{g_i}(\overline{x})$ denote the fresh Skolem function introduced for $\overline{z_i}$ in α_S.

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

$$\big(\bigwedge_{\bar{t}\in \text{BHT}_1} \alpha_S[\bar{t}]\big) \wedge \xi(2) \tag{5.16}$$

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

In particular

$$\begin{aligned}\bigwedge \Big(&\big(\theta_{1,1}(\overline{c}, \overline{d_1}) \vee \theta_{1,2}(\overline{c}, \overline{g_1}(\overline{c}))\big) \wedge \\ &\ldots \wedge \big(\theta_{r,1}(\overline{c}, \overline{d_r}) \vee \theta_{r,2}(\overline{c}, \overline{g_r}(\overline{c}))\big)\Big) \wedge \xi(1)\end{aligned} \tag{5.17}$$

is satisfied by $\mathcal{A}$, where the conjunction is over $\overline{c} \in \text{BHT}_0$ and $\overline{d_1}, \ldots, \overline{d_r} \in \text{BHT}_1$ (with the relevant arity). Note that the difference from Equation (5.16) is that there are fewer conjuncts here, because (i) the full Bounded-Horizon check with bound 2 has conjuncts for each $\overline{c} \in \text{BHT}_1$ and not just BHT_0, and (ii) we now have $\xi(1)$ instead of $\xi(2)$.

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

Now,

$$\begin{aligned}\bigwedge_{\overline{c}\in \text{BHT}_0} \Big(&\big(\forall \overline{v_1} \theta_{1,1}(\overline{c}, \overline{v_1}) \vee \exists \overline{z_1} \theta_{1,2}(\overline{c}, \overline{z_1})\big) \wedge \ldots \\ &\wedge \big(\forall \overline{v_r} \theta_{r,1}(\overline{c}, \overline{v_r}) \vee \exists \overline{z_r} \theta_{r,2}(\overline{c}, \overline{z_r})\big)\Big) \wedge \xi(1)\end{aligned} \tag{5.18}$$

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 5.18 are all guaranteed by the conjunctions in formula 5.17.

Now formula 5.18 exactly means that

$$\mathcal{A}^{\downarrow} \models \big(\bigwedge_{\overline{c}\in \text{BHT}_0} \alpha(\overline{c})\big) \wedge \xi(1). \tag{5.19}$$

As in the proof of Theorem 4, using Lemma 4, this is a contradiction to the assumption that $\widehat{I}$ is inductive w.r.t. $\widehat{\delta}$, and the claim follows. □

5.2.2 Generalization to 1-Alternation Invariants

We now generalize the results of Section 5.2.1 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, $\widehat{I} \in \text{AF}(\widehat{\Sigma})$ and $I = \widehat{I}[\psi/r] \in$ 1-alternation(Σ).

Lemma 5. *Let $\psi \in \text{FOL}(\Sigma)$. Let $\widehat{I} \in \text{AF}(\widehat{\Sigma})$ be an inductive invariant for $\widehat{\delta} \in \exists^*\forall^*(\widehat{\Sigma})$. 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 $\widehat{\alpha_1}, \widehat{\alpha_2} \in \text{QF}(\widehat{\Sigma})$. Let $\alpha_1 = \widehat{\alpha_1}[\psi/r]$ and $\alpha_2 = \widehat{\alpha_2}[\psi/r]$. Let $\delta \in \exists^*\forall^*(\Sigma)$ and let $\widehat{\delta} \in \exists^*\forall^*(\widehat{\Sigma}, \widehat{\Sigma}')$ be a sound instrumentation of δ without new existentials and with naming η. Then,*

$$\Big(\bigwedge_{\overline{c_1}\in C^n} \alpha_1(\overline{c_1})\Big) \ \wedge\ \delta_S \ \wedge\ \Big(\bigwedge_{\overline{c_2}\in C^m} \alpha_2(\overline{c_2})\Big) \tag{5.20}$$

is unsatisfiable, where $C = const[\widehat{I}_S \wedge \delta_S \wedge (\neg I')_S]$, 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, with the transformations applied to the conjunction are performed both on the invariant in the pre-state and in the post-state.

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

$$\mathcal{A}_0 \models \Big(\bigwedge_{\overline{c_1}\in C^n} \alpha_1(\overline{c_1})\Big) \ \wedge\ \delta_S \ \wedge\ \Big(\bigwedge_{\overline{c_2}\in C^m} \alpha_2(\overline{c_2})\Big). \tag{5.21}$$

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

$$\mathcal{A}_1 \models \Big(\bigwedge_{\overline{c_1}\in \widehat{C}^n} \alpha_1(\overline{c_1})\Big) \ \wedge\ \delta_S \ \wedge\ \Big(\bigwedge_{\overline{c_2}\in \widehat{C}^m} \alpha_2(\overline{c_2})\Big). \tag{5.22}$$

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 8), it follows that,

$$\mathcal{A}_1 \models \Big(\bigwedge_{\overline{c_1}\in \widehat{C}^n} \alpha_1(\overline{c_1})\Big) \ \wedge\ \eta(\widehat{\delta})[\psi/r, \psi'/r'] \ \wedge\ \Big(\bigwedge_{\overline{c_2}\in \widehat{C}^m} \alpha_2(\overline{c_2})\Big). \tag{5.23}$$

Thus we have shown,

$$\mathcal{A}_1 \models \Big(\Big(\bigwedge_{\overline{c_1}\in \widehat{C}^n} \widehat{\alpha_1}(\overline{c_1})\Big) \ \wedge\ \eta(\widehat{\delta}) \ \wedge\ \Big(\bigwedge_{\overline{c_2}\in \widehat{C}^m} \widehat{\alpha_2}(\overline{c_2})\Big)\Big)[\psi/r, \psi'/r']. \tag{5.24}$$

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,

$$\widehat{\mathcal{A}} \models \big(\bigwedge_{\overline{c_1} \in \widehat{C}^n} \widehat{\alpha_1}(\overline{c_1}) \big) \ \wedge \ \eta(\widehat{\delta}) \ \wedge \ \big(\bigwedge_{\overline{c_2} \in \widehat{C}^m} \widehat{\alpha_2}(\overline{c_2}) \big) \tag{5.25}$$

Since the formula in Equation (5.25) is universal, it 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 $\widehat{C}$. Thus,

$$\widehat{\mathcal{A}}_{|\widehat{C}} \models \big(\forall \overline{x}.\ \widehat{\alpha_1}(\overline{x}) \big) \ \wedge \ \eta(\widehat{\delta}) \ \wedge \ \big(\forall \overline{x}.\ \widehat{\alpha_2}(\overline{x}) \big) \tag{5.26}$$

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,

$$\widehat{\mathcal{A}}_{|\widehat{C}} \models \widehat{I} \ \wedge \ \widehat{\delta} \ \wedge \ \neg \widehat{I}'. \tag{5.27}$$

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

The following result is a corollary of Lemma 5.

Theorem 6. *Let* $\widehat{I} \in$ AF*. If* $\psi \in$ AF *then* $I = \widehat{I}[\psi/r]$ *is inductive for* δ *with Bounded-Horizon of bound* 2*. (Note that* $I \in$ 1-alternation*.)*

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

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

$$\big(\bigwedge_{\overline{t} \in \text{BHT}_1} (\alpha_1)_S[\overline{t}] \big) \wedge \big(\bigwedge_{\overline{t} \in \text{BHT}_2} \delta_S[\overline{t}] \big) \wedge \big(\bigwedge_{\overline{t} \in \text{BHT}_1} (\alpha_2)_S[\overline{t}] \big) \tag{5.28}$$

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 5,

$$\mathcal{A}^{\downarrow} \models \bigwedge_{\overline{c} \in \text{BHT}_0} \alpha_1(\overline{c}) \tag{5.29}$$

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

$$\mathcal{A}^{\downarrow} \models \bigwedge_{\overline{c} \in \text{BHT}_0} \alpha_2(\overline{c}). \tag{5.30}$$

Note that from Equation (5.28), $\mathcal{A}^{\downarrow} \models \widehat{\delta}$. Overall we have

$$\mathcal{A}^{\downarrow} \models \big(\bigwedge_{\overline{c} \in \text{BHT}_0} \alpha_1(\overline{c}) \big) \ \wedge \ \delta_S \ \wedge \ \big(\bigwedge_{\overline{c} \in \text{BHT}_0} \alpha_2(\overline{c}) \big) \tag{5.31}$$

and by Lemma 5 this is a contradiction to the assumption that $\widehat{I}$ a is inductive w.r.t. $\widehat{\delta}$. $\square$

Chapter 6

Instrumentation for High Depth Instantiations

In this chapter we discuss the connection between quantifier instantiation and program instrumentation in the converse direction, i.e. simulating quantifier instantiation with instrumentation. In Chapter 5 we showed that instrumentation without adding existential quantifiers is powerful at most as bounded instantiations with a low bound. In this chapter 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 to make the verification conditions decidable to check. Together with Chapter 5, this makes the point that, in the context of invariant checking, the form of instrumentation by a derived relation studied in this thesis directly corresponds to quantifier instantiation.

As before, the instrumentation process begins with a program that does not have a universal inductive invariant. We would like to transform the program in a sound way to a program that has a universal inductive invariant $\widehat{I}$. As before, we start by identifying 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 "$r(\overline{x}) \equiv \psi(\overline{x})$". As before, we present the procedure for a single formula and instrumentation relation, but in practice it can be used with multiple formulas and instrumentation relations (see Remark 2).

Adding the instrumentation relation r lets $\widehat{I} \in \forall^*(\widehat{\Sigma})$ express existential information by referring to r. The modifications to the program must encode enough of "$r(\overline{x}) \equiv \psi(\overline{x})$" to make $\widehat{I}$ inductive. As opposed to the instrumentation described in Section 5.1, this will be done by instantiating the correspondence between $\psi(\overline{t})$ and $r(\overline{t})$ for specific variables in the program using `assume` statements. This replaces the rewriting of program conditions that use ψ to use r. This approach is also extended to obtain deep instantiations which simulate Bounded-Horizon with arbitrary bound.

Local Instantiations Rather than rewriting $\psi(\overline{t})$ to $r(\overline{t})$ in the program, 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

$\forall \overline{x}.\ \psi(\overline{x}) \to r(\overline{x})$ is an EPR formula (since $\psi \in \exists^*$), and thus we can simply conjoin it to the verification condition without sacrificing decidability.

The converse implication, $\forall \overline{x}.\ r(\overline{x}) \to \psi(\overline{x})$, is a $\forall^*\exists^*$ formula, and adding it to the verification condition will lead to undecidability. Note that Bounded-Horizon with bound 1 is analogous to enforcing $r(\overline{t}) \to \psi(\overline{t})$ for every t that is a program variable. Inspired by this, we define the following instrumentation that lets the user locally enforce the definition of r for program variables.

Definition 9 (Local Instantiation). *Let $\psi(\overline{x}) = \exists \overline{y}.\ \varphi(\overline{x}, \overline{y})$. To generate an instantiation of $\forall \overline{x}.\ r(\overline{x}) \to \exists \overline{y}.\ \varphi(\overline{x}, \overline{y})$ on some tuple of program variables $\overline{t}$, we instrument the program by adding new program variables $\overline{c}$, and inserting the following code:*

$$\overline{c} := *;\ \mathtt{assume}\ r(\overline{t}) \to \varphi(\overline{t}, \overline{c})$$

This code 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 $r(\overline{t}) \to \exists \overline{y}.\ \varphi(\overline{t}, \overline{y})$, and also assigns to the new program variables $\overline{c}$, the witnesses for the existential quantifiers. 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 6 (Soundness of Local Instantiations). *If $\widehat{\delta}$ is obtained from δ by a local instantiation then $\widehat{\delta}$ is a sound instrumentation by Definition 6.*

Proof. The code added by a local instantiation for $\overline{t}$ that uses new variables $\overline{c}$ translates a new constraint in $\widehat{\delta}$ of the form $\gamma = \exists \overline{c}.\ r(\overline{t}) \to \varphi(\overline{t}, \overline{c})$. Since $\forall \overline{x}.\ r(\overline{x}) \leftrightarrow \psi(\overline{x}) \Rightarrow \gamma$, we have $\forall \overline{x}.\ r(\overline{x}) \leftrightarrow \psi(\overline{x})\ \wedge\ \delta \Rightarrow \widehat{\delta}$, which implies the condition of Definition 6. □

Remark 3. The combination of adding $\forall \overline{x}.\ \psi(\overline{x}) \to r(\overline{x})$ 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 $\psi(\overline{t})$ to $r(\overline{t})$ can be simulated by a local instantiation on $\overline{t}$.

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 universally quantified variable x in $\widehat{I}$, 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.

Obtaining Deep Instantiations Applying local instrumentation on a tuple $\overline{t}$ that consists of original program variables, or variables that represent Skolem constants, corresponds to instantiations of

```
/@  $r_1(x,y) \equiv \exists z.\ \mathit{req}(x,z) \land \mathit{db}(z,y)$
/@  $r_2(x,y) \equiv \exists z.\ t(x,z) \land \mathit{req}(z,y)$
/@  $r_3(x,y) \equiv \exists z.\ \mathit{db\_req}(x,z) \land \mathit{db}(z,y)$
/@  Invariant $\widehat{I} = \forall u,p.\ \mathit{resp}(u,p) \to r_1(u,p) \land$
/@        $\forall id,q.\ \mathit{db\_req}(id,q) \to r_2(id,q) \land$
/@        $\forall id,p.\ \mathit{db\_resp}(id,p) \to r_3(id,p) \land$
/@        $\forall id,u_1,u_2.\ t(id,u_1) \land t(id,u_2) \to u_1 = u_2$
action server_process_db_response($id,p$) {
  # instantiate $r_3$ on $(id, sk_p)$ (depth 1)
  /@  $c_1 := *$;
  /@  assume $r_3(id,sk_p) \to \mathit{db\_req}(i,c_1) \land \mathit{db}(c_1,sk_p)$;
  # instantiate $r_2$ on $(id,c_1)$ (depth 2)
  /@  $c_2 := *$;
  /@  assume $r_2(id,c_1) \to t(id,c_2) \land \mathit{req}(c_2,c_1)$;
  ...
}
action check($u,p$) {
  # instantiate $r_1$ on $(u,p)$ (depth 1)
  /@  $c_3 := *$;
  /@  assume $r_1(u,p) \to \mathit{req}(u,c_3) \land \mathit{db}(c_3,p)$;
  ...
}
```

Figure 6.1: An illustration of instrumentation by local instantiations for the example of Figure 4.2. 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`).

Bounded-Horizon with bound 1. However, once a local instantiation is performed, new program variables $\overline{c}$ are added. Performing a local instantiation on these new variables now corresponds to instantiation from depth 2. By iteratively applying local instantiations, where each iteration adds new program variables, we can thus obtain quantifier instantiations of arbitrary depth.

Illustrating Example Figure 6.1 illustrates the local instantiation procedure on the example of Figure 4.2. Recall that the $\forall^*\exists^*$ invariant I of Figure 4.2 is not provable using Bounded-Horizon of bound 1, but is provable using Bounded-Horizon of bound 2. The instrumentation presented in Figure 6.1 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 both the `server_process_db_response` action and the `check` action. 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 that the original $\forall^*\exists^*$ invariant I (see Chapter 4).

We note that the process of instrumentation by local instantiations discussed here is different in spirit from those of Chapter 5: instrumentation by local instantiation consists of almost nothing but adding existential quantifiers to the transition relation, as opposed to the condition in Definition 8 where we do not allow the instrumentation to add new existential quantifiers.

Chapter 7

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, adapts the program, the invariant, or both.

In such scenarios, it is extremely desirable to present the user with a *counterexample to induction* in the form of a state that satisfies I but makes a transition to a state that violates it. Such a state can be obtained from a model of the formula $\mathit{Ind} = I \wedge \delta \wedge \neg I'$ which is used to check inductiveness. It explains the error, and guides the user towards fixing the program and/or the invariant [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, and is obtained as a model of the (finite) instantiations of the formula *Ind*. While this structure is not a true counterexample (as it is only a model of a subset of the instantiations of the formula), it can still guide the user in the right direction towards fixing the program and/or the invariant.

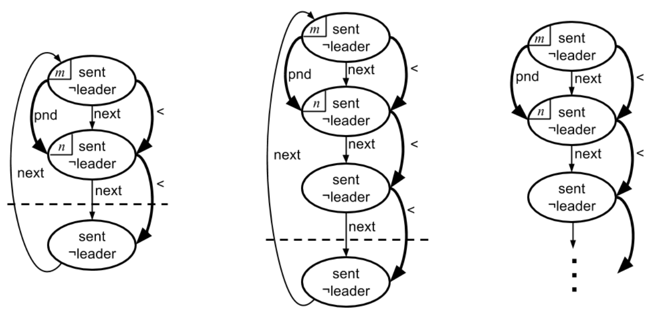
We illustrate this using a simple leader-election protocol in a ring [CR79], whose model is presented in Figure 7.1(a). The protocol assumes that nodes are organized in a directional ring topology with unique IDs, and elects the node with the highest ID as the leader. Each node sends its own ID to its successor, and forwards messages when they contain an ID higher than its own ID. A node that receives its own ID is elected as leader. We wish to prove a termination property which states that once all nodes have sent their ID, and there are no pending messages in the network, then there is an elected leader. To verify this we use a relational model of the protocol similar to [PMP$^+$16], and specify the property via the following formula:

$$(\exists n.\ \mathit{leader}(n)) \vee (\exists n_1, n_2.\ \neg \mathit{sent}(n_1) \vee \mathit{pending}(n_1, n_2)) \tag{7.1}$$

```
$pending := \emptyset$;
... # ring topology

action send_packet($n$) {
  assume $ring\_next(n, m)$
  $pending := pending \cup \{(n, m)\}$
  $sent := sent \cup \{n\}$
}

action receive_packet($n, m$) {
  assume $pending(m, n)$
  $pending := pending \setminus \{(m, n)\}$
  if $m = n$ then
    $leader := leader \cup \{n\}$
  else
    if $n < m$ then
      assume $ring\_next(n, n_0)$
      $pending := pending \cup \{(m, n_0)\}$
    else # do not forward
}
```



(a) (b) (c) (d)

Figure 7.1: Leader-election in a ring protocol as an illustration of the use of partial models for incorrect programs and invariants. (a) sketches the protocol (the complete program appears in [add], file `ring_leader_termination.ivy`). (b),(c) show partial models of bound 1 and 2, respectively, and (d) illustrates an infinite structure that explains the root cause of the non-inductiveness.

A natural attempt of proving this using an inductive invariant is by conjoining Equation (7.1) (which is not inductive by itself) with the following property (this was the authors' actual next step in proving this termination property):

$$\forall n_1.\ sent(n_1) \land \neg leader(n_1) \rightarrow ((\exists n_2.\ pending(n_1, n_2)) \lor (\exists n_2.\ n_1 < n_2)) \tag{7.2}$$

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.

Alas, the conjunction of Equations (7.1) and (7.2) is still not an inductive invariant for the protocol (as we explain below). Since Equation (7.2) 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. 7.1(b) and (c) show partial models (restricted to the pre-states) obtained with bounds of 1 and 2, respectively, on this example.

These models are not true counterexamples to induction: the sub-formula of Equation (7.2) 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. These elements have all sent their own ID, which was blocked by their successor that has a higher ID, so none of them is the leader. In a finite model, this has to end somewhere, because one of the nodes must have the highest ID. Hence, no finite counter-model exists. However, extrapolating from

Figure 7.1(b) and (c), we can obtain the infinite model depicted in Figure 7.1(d). This model represents an infinite ("open") ring in which each node has a lower ID than its successor. This model is a true model of the formula *Ind* generated by the invariant in Equations (7.1) and (7.2), 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 about what is missing. In this case, the solution is to add an axiom to the system model which states that there is a node with maximal ID: $\exists n_1.\ \forall n_2.\ n_2 \leq n_1$. With this additional assumption, the formula *Ind* is unsatisfiable so the invariant is inductive, and this is proven both by Z3's instantiation heuristics and by Bounded-Horizon with a bound of 1. This illustrates the usefulness of Bounded-Horizon when the invariant is not inductive.

Chapter 8

Implementation and Initial Evaluation

We implemented a prototype of Bounded-Horizon of bound 1 on top of Z3 [DMB08] and used it within Ivy [PMP$^+$16] and the framework of [IBI$^+$13]. We applied the procedure to the incorrect example of Chapter 7, and 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 Figure 4.1.
- 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 n^* as explained in [IBI$^+$13].
- Learning switch [BBG$^+$14], where the invariant states every routing node has a successor.
- Hole-punching firewall [BBG$^+$14], where the invariant states that every allowed external node was contacted by some internal node. We explored two modeling alternatives: using a ghost history relation, or existentially quantifying over time.
- Leader election in a ring [CR79, PMP$^+$16] with the invariant discussed in Chapter 7.

Table 8.1: Experimental results.

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

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

An initial evaluation of the method's performance appears in Table 8.1.

Our implementation works by adding "guards" that restrict the range of universal quantifiers to the set of constants where necessary. Technically, recall that we are considering the satisfiability of *Ind* $= I_S \wedge \delta_S \wedge (\neg I')_S$. [1]Let $\forall x.\ \theta$ be a subformula of *Ind*. If θ contains function symbol applications[2], we transform the subformula to $\forall x.\ \left(\bigvee_c x = c\right) \rightarrow \theta$ where c ranges over const[*Ind*]. The resulting formula is then dispatched to the solver. This is a simple way to encode the termination criterion of bound 1 while leaving room for the solver to perform the necessary instantiations cleverly. The translation enlarges the formula by $O(\#\text{Consts} \cdot \#\forall)$ although the number of bounded instantiations grows exponentially with #∀. The exponential explosion is due to combinations of constants in the instantiation, a problem we defer to the solver.

Z3 terminates on the class of formulas because during the Model-Based Quantifier Instantiation process every instantiation of a universally quantified formula has the same truth value in the model as an instantiation using one of the existing ground terms (constants and then BHT_1 terms). Z3's instantiation engine will produce instantiations using existing terms rather than create superfluous new terms [Bjø].

The results are encouraging because they suggest that the termination strategy of Bounded-Horizon, at least for bound 1, can be combined with existing instantiation techniques to assure termination with only a slight performance penalty. Most encouraging is the satisfiable example of Chapter 7. On this instance, Z3 was able to return "sat" within seconds, although to do so, in theory, the solver must exhaust the entire set of bounded instantiations. This suggests that the Bounded-Horizon termination criterion might indeed be useful for "sat" instances on which the solver may diverge.

A different approach to the implementation is to integrate the termination criterion of the bound with the solver's heuristics more closely (see [BRK$^+$15]).

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

[2]This in fact implements the approximation as of Equation (4.2). The exact bound 1 per Equation (4.1) can be implemented by a more careful consideration of which universally quantified variables should be restricted, but this was not necessary for our examples.

Chapter 9

Related Work

Quantifier instantiation. The importance of formulas with quantifier-alternations for program verification has led to many developments in the SMT and theorem-proving communities that aim to allow automated reasoning with quantifier-alternations. The Simplify system [DNS05] promoted the practical usage of quantifier triggers, which let the user affect the quantifier instantiation in a clever way. Similar methods are integrated into modern SMT solvers such as Z3 [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.

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 Chapter 7 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].

Chapter 10

Conclusion

We have provided an initial study of the power of bounded instantiations for tackling quantifier alternation. This thesis 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.

Bibliography

[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.

[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.

[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$^+$15] Aleksandr Karbyshev, Nikolaj Bjorner, Shachar Itzhaky, Noam Rinetzky, and Sharon Shoham. Property-directed inference of universal invariants or proving their absence. In *CAV*, 2015.

[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.

[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.

[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.

תקציר

עבודה זו עוסקת בבדיקה אם אינווריאנטה מוצעת שמבוטאת בלוגיקה מסדר-ראשון עם חילופי כמתים היא *אינדוקטיבית*, כלומר משתמרת על-ידי קטע קוד (ללא לולאות). על אף שהבעיה אינה כריעה, כלי SMT (ספיקות מודולו תורה לוגית) מודרניים לפעמים מצליחים לבצע את הבדיקה באופן אוטומטי. עם זאת, הם משתמשים בטכניקות רבות-עוצמה של אינסטנסיאציה של כמתים אשר עלולות להתבדר, במיוחד כשהאינווריאנטה המוצעת אינה אינדוקטיבית. קושי משמעותי הוא קיומן של דוגמאות-נגד לאינדוקטיביות בגודל אינסופי.

עבודה זו חוקרת אלגוריתם בשם *Bounded-Horizon* (אופק חסום), שהוא דרך טבעית להבטיח התכנסות של כלי SMT. השיטה חוסמת את העומק של ביטויים בתהליך האינסטנציאציה של הכתמים. אנחנו מראים ששיטה זו רבת-עוצמה במידה מפתיעה כשהיא מופעלת לצורך בדיקת אינדוקטיביות של אינווריאנטות מעל תחומים לוגיים לא-מפורשים. מעבר לזאת, על-ידי הפקת דוגמאות-נגד חלקיות, השיטה יכולה לסייע למשתמש לאבחן שהאינווריאנטה אינה אינדוקטיבית, במיוחד כאשר הסיבה היא קיומה של דוגמת-נגד אינסופית.

התוצאה הטכנית העיקרית של העבודה היא שאלגוריתם Bounded-Horizon חזק לפחות כמו *אינסטרומנטציה*, שהיא שיטה ידנית להבטיח את ההתכנסות של כלי ה-SMT על ידי שינוי התוכנית כך שהאינווריאנטה המתאימה ניתנת לביטוי באמצעות כימות אוניברסלי בלבד. אנו מראים שעם חסם של 1 האלגוריתם יכול לחקות מחלקה טבעית של אינסטרומנטציות, ללא צורך לשנות את התוכנית ובצורה אוטומטית לחלוטין.

בנוסף, אנו מתארים מימוש ראשוני של האלגוריתם, מעל הכלי Z3, בו השתמשנו לוודא מספר דוגמאות באמצעות Bounded-Horizon עם חסם של 1.

אוניברסיטת תל-אביב TEL AVIV UNIVERSITY

הפקולטה למדעים מדויקים ע״ש ריימונד ובברלי סאקלר

בית הספר למדעי המחשב ע״ש בלבטניק

אינסטנציאציית כמתים חסומה לבדיקת אינווריאנטות אינדוקטיביות

חיבור זה הוגש כחלק מהדרישות לקבלת התואר

"מוסמך האוניברסיטה" (M.Sc.)

על ידי

יותם פלדמן

עבודת המחקר בוצעה בהנחייתו של

פרופ' מולי שגיב

ד"ר שרון שוהם