Foundations for Natural Proofs and Quantifier Instantiation

CHRISTOF LÖDING, RWTH Aachen University, Germany
P. MADHUSUDAN, University of Illinois at Urbana-Champaign, U.S.A.
LUCAS PEÑA, University of Illinois at Urbana-Champaign, U.S.A.

We give foundational results that explain the efficacy of heuristics used for dealing with quantified formulas and recursive definitions. We develop a framework for first order logic (FOL) over an uninterpreted combination of background theories. Our central technical result is that systematic term instantiation is *complete* for a fragment of FOL that we call *safe*. Coupled with the fact that unfolding recursive definitions is essentially term instantiation and with the observation that heap verification engines generate verification conditions in the safe fragment explains the efficacy of verification engines like natural proofs that resort to such heuristics. Furthermore, we study recursive definitions with least fixpoint semantics and show that though they are not amenable to complete procedures, we can systematically introduce induction principles that in practice bridge the divide between FOL and FOL with recursive definitions.

CCS Concepts: • **Theory of computation → Logic and verification**; **Automated reasoning**; **Programming logic**; **Hoare logic**; **Separation logic**;

Additional Key Words and Phrases: Natural proofs, Quantifier instantiation, Completeness, Program analysis, Program logics, Verification

ACM Reference Format:
Christof Löding, P. Madhusudan, and Lucas Peña. 2018. Foundations for Natural Proofs and Quantifier Instantiation. *Proc. ACM Program. Lang.* 2, POPL, Article 10 (January 2018), 30 pages. https://doi.org/10.1145/3158098

1 INTRODUCTION

Automated reasoning for quantified logics is an important and integral component of program verification. Automated engines for checking validity and satisfiability of quantifier-free logics for combinations of several important theories has made a significant impact in systems verification, especially through their efficient realization in SMT solvers, resulting in robust and scalable engines both for verification and finding bugs using symbolic execution. However, automated quantified reasoning remains mostly driven by heuristics, resulting in tools that are extremely fragile and unpredictable [Kovács and Voronkov 2013; Schulz 2002; Weidenbach et al. 2009]. Furthermore, software verification requires reasoning in logics that are even beyond FOL, especially for reasoning with recursive/inductive definitions, which are not expressible in first-order logic.

The goal of this paper is to provide foundations for a class of heuristics currently used to reason with quantified logics in practice that are based on two paradigms— systematic quantifier

Authors' addresses: Christof Löding, Department of Computer Science, RWTH Aachen University, Germany, loeding@automata.rwth-aachen.de; P. Madhusudan, Department of Computer Science, University of Illinois at Urbana-Champaign, U.S.A., madhu@illinois.edu; Lucas Peña, Department of Computer Science, University of Illinois at Urbana-Champaign, U.S.A., lpena7@illinois.edu.

2475-1421/2018/1-ART10
https://doi.org/10.1145/3158098

instantiation using terms for FOL and the unfold-and-match paradigm to deal with recursive definitions. In particular, we are interested in settings where we can show these heuristics in fact lead to *sound and complete reasoning*, even though the logics are undecidable.

We work in a reasoning framework that is *refutation-based*, where, in order to prove a formula φ valid, under some theory T, we build procedures that show that $\neg\varphi$ is not T-satisfiable.

Given a formula in FOL that we need to show unsatisfiable, we can Skolemize it to get a universally quantified formula $\forall\overline{x}\ \psi$. By Herbrand's Theorem and compactness theorem for first-order logic [Ebbinghaus et al. 1994; Schöning 2008], it follows that the formula is unsatisfiable iff there is a finite subset of ground terms which when substituted for the variables $\overline{x}$ renders it unsatisfiable. Term instantiation heuristics aim to find such a set of ground term substitutions for the variables in some systematic fashion, guided by the formula to be shown unsatisfiable. The widely applied resolution-based reasoning [Davis and Putnam 1960; Robinson 1965] is also a refutation-based method that employs such a targeted procedure [Ganzinger and Korovin 2003, 2006].

1.1 Uninterpreted Combination of Theories

A primary motivation of our work is to build reasoning for FOL (with and without recursive definitions) that can utilize the efficient solvers for quantifier free theories realized as SMT solvers available today.

Consequently, we work with a framework called Uninterpreted Combination of Theories (UCT) that explicitly allows reasoning modulo background theories. A UCT is a many-sorted universe where there is a special sort that is declared to be a *foreground sort*, while the other sorts are declared *background sorts*. The signature allows functions and relations of any type on the underlying sorts. Furthermore, we allow arbitrary background theories for the various individual background sorts (these will mention only functions and relations that are restricted to the particular background sort, of course). And we insist the foreground sort be entirely uninterpreted.

Formulas over a UCT can, of course, include any *finite* axiomatization of the foreground sort; hence any finite axiomatizations are permitted. UCTs can hence be seen as a combination of the background theories that communicate through uninterpreted functions and relations and a new foreground type.

We seek procedures that are both sound and complete for checking unsatisfiability of formulas over UCTs. More precisely, given a formula $\forall\overline{x}\psi$, the goals of this paper are to perform term instantiation for universally quantified variables over the foreground sort, resulting in a sequence of formulas $\varphi_1, \varphi_2, \varphi_3, \ldots$ corresponding to each level of instantiation so that each φ_i has no foreground quantification and hence is over the combined theory of uninterpreted functions and the background theories. This term instantiation is sound if each φ_i is implied by $\forall\overline{x}\psi$ and complete if whenever $\forall\overline{x}\psi$ is unsatisfiable, there is some $i \in \mathbb{N}$ such that φ_i is unsatisfiable.

When instantiated formulas fall into a class that has a decidable satisfiability problem, a sound and complete term instantiation coupled with checking satisfiability of the successively instantiated formulas results in a *recursively enumerable* procedure for unsatisfiability, and hence for validity.

1.2 The Safe Fragment and Completeness

One technique for term instantiation that is followed by many practical tools today is to simply instantiate using the terms (of the appropriate type) occurring in the formula at the previous level of instantiation (i.e., where φ_{i+1} has variables instantiated with ground terms occurring in φ_i, where $i \geq 0$ and φ_0 is interpreted to be $\forall\overline{x}\psi$). This kind of term instantiation is often effective in practice as it is targeted towards the formula being verified, and is used by the E-matching and pattern-based quantification in SMT solvers as well as in the work on natural proofs [Madhusudan et al. 2012; Pek et al. 2014; Qiu et al. 2013; Suter et al. 2010].

Our first result shows that when there are no background theories present, such a term instantiation is actually complete for UCTs. The proof is a slight generalization of the proof that arbitrary term instantiation (not guided by the formula) is sound and complete using Herbrand's theorem and the compactness theorem [Ebbinghaus et al. 1994; Schöning 2008].

However, in the presence of background theories, we show that this kind of term instantiation is *not complete*. The *central technical contribution* of this paper is to identify a safe fragment of FOL over UCTs for which term instantiation is actually a *complete* procedure.

The safe fragment consists of safe formulas, where a formula is safe if every term with the foreground type occurring in it has all its variables also of the foreground type. This essentially rules out applying functions on terms of the background sorts, that involve universally quantified variables, and that map to the foreground sort. Two simple ways of satisfying the safety requirement are (a) to have no functions that have as part of the domain a background type and map to the foreground type (we call such a UCT as having *one-way functions*) or (b) that the formula has no quantification over the background sort.

The safe fragment hence admits a much more predictable validity checking procedure, when the background theories are decidable. In particular, for formulas that do not quantify over the background sorts directly, term instantiation is complete, and furthermore results in formulas that are quantifier-free over a combination of the background theories and the theory of uninterpreted functions. When the background theories are individually decidable and stably infinite, these formulas will be decidable using a Nelson-Oppen combination decision procedure, and amenable to solutions provided by current SMT solvers [Nelson and Oppen 1979].

1.3 Beyond FOL: Recursive Definitions and Least Fixpoints

In program verification, not only do we often need quantification to express specifications and invariants, we also require recursively defined properties. For instance, data-structures such as lists, doubly linked lists, trees, binary search trees, AVL trees, etc., are typically defined using recursion. Furthermore, we often need *pure* recursive functions to aggregate properties of such structures— for example, to define the set of keys stored in a list, to define the height of a tree, to define the sequence defined by a list, and in separation logic, to express the heaplet defined by a data-structure and, more generally, to describe frames that a function modifies in order to support modular reasoning.

Using a simple reduction, we can show that the validity problem for FOL enriched with recursive definitions with least fixpoint semantics is *not* recursively enumerable, even when there are no background theories present. In fact, there is a fundamental problem in building complete verification engines in the context of program verification when the underlying logic is undecidable. Program configurations are typically *enumerable*— configurations after all have finitary descriptions, even if they have unbounded datatypes (like integers) or unbounded heaps. Consequently, satisfiability for the logic is recursively enumerable (r.e.) since model-checking formulas for FO enriched with recursive definitions (in fact, for any reasonable logic) over any given finite structure is decidable. Consequently, validity cannot be in r.e. if the validity for the logic is undecidable, for if it were, one could enumerate all valid formulas and all finite models. If the formula is valid, we will find it in the enumeration of valid formulas, and if it is not, we will find a model that satisfies its negation.

There are several existing program verification systems for reasoning with recursion that use a heuristic roughly known as "unfold-and-match" [Chu et al. 2015; Jacobs and Piessens 2008; Leino 2012; Madhusudan et al. 2012; Nguyen and Chin 2008; Pek et al. 2014; Qiu et al. 2013; Suter et al. 2010]. These systems involve unfolding recursive definitions a few times, and then finding a simple proof of validity using the unfolded formulas, with recursive definitions treated as uninterpreted after unfolding. The "match" in the paradigm refers to matching variables arising from unfolding recursive definitions to variables in the proof obligation. There is in fact a distinct similarity between unfolding recursive definitions and term instantiation— given an occurrence $r(\overline{t})$ in the formula where r has a recursive definition, unfolding the recursive definition of r at $\overline{t}$ is essentially the same as instantiating the variables $\overline{x}$ in the universally quantified formula $\forall \overline{x}.\ r(x) \leftrightarrow \phi(r, \overline{x})$ expressing the recursive definition. Furthermore, the instantiated formula will result in new terms that can be instantiated in the next round, corresponding to unfolding the definitions further.

The second contribution of this paper is to show that the completeness result for the safe fragment sheds light on the efficacy of the "unfold-and-match" heuristics used in practice. First, when recursive definitions are just unfolded and used to prove validity, the method essentially ignores the least fixpoint semantics and using only the fact that the definitions define *some* fixpoint. Moreover, if we are expressing only the fixpoint semantics, then it can be expressed in FOL, and our completeness result says that the method is in fact complete for the safe fragment. We consider one particular work called *natural proofs* [Madhusudan et al. 2012; Pek et al. 2014; Qiu et al. 2013; Suter et al. 2010] in the literature, and show that the verification conditions obtained for heap verification therein naturally fall into the safe fragment.

In fact, in the work on natural proofs [Pek et al. 2014; Qiu et al. 2013], several valid verification conditions are unprovable using "unfold-and-match" techniques, and we show that the reason they are not provable is because they are not valid with respect to *fixpoint* semantics, though they are valid under least fixpoint semantics. In the context of a UCT with Presburger arithmetic as a background theory, this is at first glance puzzling, as recursive definitions can be formulated using fixpoint semantics through induction on natural numbers. However, it turns out that Presburger arithmetic (or any arithmetic) as background theory admits *non-standard* models of arithmetic that prohibit defining such recursive definitions accurately.

In natural proofs [Pek et al. 2014; Qiu et al. 2013], the user is required to give additional axioms that provide help in dealing with the incompleteness. The third contribution of this paper is to provide general induction principles that are driven by the property that can be introduced to prove properties involving recursive definitions. These induction principles can be seen as formulae that are valid with respect to least fixpoint semantics of recursive definitions, but not valid with respect to fixpoint semantics. We show an evaluation of these principles in the context of heap verification using natural proofs, showing that the user-given unproven axioms are in fact provable automatically using our automatic insertion of induction principles.

We provide several additional results not mentioned above. One set of results involve logics over arrays. We show that the known decision procedure for the *array property fragment* (APF), which uses finite term instantiation, can be seen as a UCT framework where our term instantiation gives the same decision procedure. Furthermore, we show that non-standard models, being quite non-intuitive, has led to errors in proofs in earlier work. In particular, we point to several errors involving the undecidability of extensions of APF; this leads us to revert one problem, the validity of APF extended with strict inequality between indices, back to being an open problem.

The paper is structured as follows. Section 2 defines many-sorted logic, the UCT framework, and FOL enriched with recursively defined relations with least fixpoint as well as fixpoint semantics. Section 3 considers the simpler setting where there are no background theories, and shows that systematic term instantiation followed by a decision procedure for quantifier-free uninterpreted functions is sound and complete. Section 4 present our main technical results on the safe fragment of FOL with background theories and proves that term instantiation is sound and complete for it. In

Section 5 we discuss several ramifications of our main result, including explaining the "unfold-and-match" heuristics in natural proofs, the subtle issues that nonstandard models create when using recursive definitions, and discussions of APF and its extensions. In Section 6, we show automatic ways of adding induction principles that bridge the gap created when recursive definitions are treated using fixpoint semantics and show their power in validating the user-provided axioms in the work on natural proofs [Pek et al. 2014; Qiu et al. 2013]. Related work is discussed in Section 7 and we conclude with future directions in Section 8.

2 DEFINITIONS AND NOTATION

In this section we define some basic notions of many-sorted first-order logic (FOL) with equality, and define the framework of uninterpreted combination of theories (UCT). We also define some formal notions of uninterpreted first-order recursively defined relations.

2.1 Many-Sorted Logic

We define a signature as $\Sigma = (S; F; \mathcal{R})$, where $S = \{\sigma_1, \ldots, \sigma_n\}$ is a finite non-empty set of sorts. F is a set of function symbols, where each function $f \in F$ has a type of the form $\tau_1 \times \ldots \times \tau_m \longrightarrow \tau$ for some m, with $\tau_i, \tau \in S$. Function symbols where $m = 0$ represent constant symbols of sort τ in our signature. Similarly, $\mathcal{R}$ is a set of relation symbols, where each relation $R \in \mathcal{R}$ has a type of the form $\tau_1 \times \ldots \times \tau_m$. We use $=$ as our infix equality symbol.

For variables, let Var_τ denote the set of variables of sort τ, where $\tau \in S$. We abbreviate $x \in Var_\tau$ with $x : \tau$. We sometimes simply use x if the sort is clear or unimportant. We let $\overline{x}$ abbreviate $x_1, \ldots, x_n$. The notation $\overline{x} : \tau$ abbreviates $x_1 : \tau, \ldots, x_n : \tau$. If $n = 0$, this denotes the empty sequence of variables. We use similar notation for a sequence of terms. A ground term refers to a term t without any variables.

A Σ-term t of sort τ is formed as

$$t ::= x : \tau \mid f(t_1, \ldots, t_m)$$

where f has type $\tau_1 \times \ldots \times \tau_m \longrightarrow \tau$, and each Σ-term t_i is of sort τ_i.

An atomic Σ-formula ψ is of the form

$$\psi ::= \top \mid \bot \mid s = t \mid R(t_1, \ldots, t_n)$$

where s and t are Σ-terms of the same sort, and R has type $\tau_1 \times \ldots \times \tau_n$, and each term t_i is of sort τ_i. Finally, a Σ-formula φ takes the form

$$\varphi ::= \psi \mid \neg\varphi' \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \varphi_1 \leftrightarrow \varphi_2 \mid \forall x : \tau.\ \varphi' \mid \exists x : \tau.\ \varphi'$$

where φ', φ_1, and φ_2 are Σ-formulas. A quantifier-free Σ-formula is formed using the above grammar, omitting the cases with $\forall$ and $\exists$. We often write term instead of Σ-term and formula instead of Σ-formulas when Σ is clear from the context.

An occurrence of a variable x in a formula is free if it does not occur under the scope of a quantifier for x. By renaming of variables we can assume that each variable only occurs freely in a formula or is quantified by exactly one quantifier in the formula. We write $\varphi(x_1, \ldots, x_k)$ to indicate that the free variables of φ are among $x_1, \ldots, x_k$. Substitution of a term t for all free occurrences of variable x in a formula φ is denoted $\varphi[t/x]$. Multiple variables are substituted simultaneously as $\varphi[t_1/x_1, \ldots, t_n/x_n]$. We abbreviate this by $\varphi[\overline{t}/\overline{x}]$.

A model $M = (U; [\![\cdot]\!]_M)$ where $U = \{U_{\sigma_1}, \ldots, U_{\sigma_n}\}$ contains a set of universes for each sort, and an interpretation function $[\![\cdot]\!]_M$. The interpretation function maps each variable x of sort σ_i to some concrete element $u \in U_{\sigma_i}$, each function symbol $f : \tau_1 \times \ldots \times \tau_m \longrightarrow \tau$ to a concrete function $[\![f]\!]_M : U_{\tau_1} \times \ldots \times U_{\tau_m} \longrightarrow U_\tau$, and each relation symbol R of type $\tau_1 \times \ldots \times \tau_m$ to a concrete relation

$[\![R]\!]_M \subseteq U_{\tau_1} \times \ldots \times U_{\tau_m}$. An interpretation function can naturally be extended to terms, such that $[\![t]\!]_M$ is the element denoted by the term t in the model M.

2.2 Many-Sorted Uninterpreted Combination of Theories (UCT)

We now define the notion of Uninterpreted Combination of Theories (UCT) for many-sorted logic. We use the sort σ_1 as the sort of the "foreground" theory, and σ_i for $i > 1$ as the sort for a "background" theory. For each background type, we allow a background theory B_{σ_i} that refers only to function symbols that have type $\sigma_i^n \rightarrow \sigma_i$ for some n, and relation symbols have type σ_i^m for some m. The background theories can be arbitrary (can be just a set of formulas, or presented as an arbitrary axiomatization, even an infinite axiomatization or axiom schema). However, the foreground theory must have no axioms. We refer to this framework as a many-sorted Uninterpreted Combination of Theories (UCT).

We write $M \models \varphi$ if M satisfies a formula φ and $M \restriction_{U_{\sigma_i}} \models B_{\sigma_i}$ for each sort σ_i, under the given interpretation function and standard semantics for the propositional connectives and quantifiers. $M \restriction_{U_{\sigma_i}}$ denotes a restricted model of M with the carrier set as U_{σ_i} and the interpretation function restricted to the set U_{σ_i}. This is well-defined as all relation and function symbols of the background theories only operate over a single sort. We say $M \models X$ for a set of formulas X if $M \models \varphi$ for all $\varphi \in X$. For a given model M, a formula $\varphi(x_1, \ldots, x_k)$ with free variables $x_1, \ldots, x_k$ of sorts $\tau_1, \ldots, \tau_k$ defines a relation $[\![\varphi]\!]_M \subseteq U_{\tau_1} \times \ldots \times U_{\tau_k}$ that contains the tuples $(u_1, \ldots, u_k)$ such that M satisfies φ when each x_i is interpreted by u_i.

2.3 Recursive Definitions

Besides standard first-order logic we use recursive first-order definitions of uninterpreted relations. Before we enter into formal definitions let us give a small example.

Example 2.1. Assume that there is an uninterpreted unary function $f : \sigma_1 \rightarrow \sigma_1$. We can use a recursive definition to define the reflexive and transitive closure of f, which we denote by $rch(x, y)$ for reachability:

$$\varphi_{rch} := \forall x, y. rch(x, y) \leftrightarrow_{\text{LFP}} \underbrace{x = y \vee (x \neq y \wedge rch(f(x), y))}_{\rho_{rch}(x, y, rch)}$$

The right-hand side is a formula ρ_{rch} with two free variables of the same sort as the free variables for the predicate *rch*. Furthermore, *rch* occurs only positively in ρ. The formula is satisfied if *rch* is interpreted as the least fixpoint of ρ, which is formalized below.

The recursive definitions that we use here are based on least fixpoints of monotonic functions. In the following, we briefly introduce the corresponding terminology that is required for our purposes. For more background on first-order recursive definitions, we refer the reader to [Moschovakis 2008].

For a domain D, a function $\mu : 2^D \rightarrow 2^D$ that maps subsets of D to subsets of D is called monotonic if it preserves set inclusion, that is, $\mu(V_1) \subseteq \mu(V_2)$ for all $V_1 \subseteq V_2 \subseteq D$. A fixpoint of μ is a subset $V \subseteq D$ of the domain with $\mu(V) = V$. A monotonic function μ has a unique least fixpoint (w.r.t. set inclusion) by the Knaster-Tarski theorem [Granas and Dugundji 2003]. For the induction principles that are presented in Section 6, we use the notion of pre-fixpoint of μ, which is a set $V \subseteq D$ with $\mu(V) \subseteq V$. For a monotonic function, the least fixpoint is the intersection of all its pre-fixpoints. In particular, the least fixpoint is contained in all the pre-fixpoints.

We use first-order definable monotonic functions for our recursive definitions, as explained below.

Let $\tau_1, \ldots, \tau_k \in \{\sigma_1, \ldots, \sigma_n\}$ be sorts, and R be a relation symbol of sort $\overline{\tau} := \tau_1 \times \cdots \times \tau_k$. We say that a formula is of sort $\overline{\tau}$ if it has k free variables $x_1, \ldots, x_k$ with x_i of type τ_i.

Let $\rho(\overline{x})$ be a formula of sort $\overline{\tau}$ in which R occurs only positively (under an even number of negations). We also write $\rho(\overline{x}, R)$ because we view R as a free second-order variable that is used for a recursive definition.

Let $M = (U; [\![\cdot]\!]_M)$ where $U = \{U_{\sigma_1}, \ldots, U_{\sigma_n}\}$ be a structure. Then ρ defines a monotonic function μ_ρ over $D = U_{\tau_1} \times \cdots \times U_{\tau_k}$ by

$$\mu_\rho(V) = [\![\rho(\overline{x})]\!]_M^{\{R \leftarrow V\}} \text{for } V \subseteq D$$

where $[\![\rho(\overline{x})]\!]_M^{\{R \leftarrow V\}}$ denotes the semantics of the formula under the interpretation function $[\![\cdot]\!]_M$ with the interpretation of R changed to V. So $\mu_\rho(V)$ consists of all tuples $\overline{u} \in D$ that make $\rho(\overline{x})$ true when R is interpreted as V, and $\overline{x}$ as $\overline{u}$. Since R occurs only positively in ρ, the function μ_ρ is indeed monotonic. A recursive definition of R using ρ is of the form

$$\varphi_R = \forall \overline{x} : \overline{\tau}.R(\overline{x}) \leftrightarrow_{\text{LFP}} \rho(\overline{x})$$

with $M \models \varphi_R$ if $[\![R]\!]_M$ is the least fixpoint of μ_ρ.

Note that, technically, we only require R to occur positively in ρ. However, if we require the more strict definition that all predicates that are recursively defined only occur positively in all these definitions, then it will ensure that such a relation always exists (by Tarski-Knaster Theorem). This restriction is easily satisfied and most natural in defining recursive data structures in practice. If we do not have this restriction and allow R to occur negatively in other recursive definitions (but not in its own), then the onus is on the verification engineer to ensure that a least fixpoint exists. Interestingly, the correctness of the induction principles from Section 6 remain correct without the more strict definition.

Note that we do not allow recursive *function* definitions. Since we work over arbitrary signatures and domains in first-order logic, it is difficult to provide a generic way of recursively defining functions without having access to a well-order on the domain (or over recursively defined datatypes). Consequently, we require recursive definitions to be formulated using recursively defined relations that are "functional" (model a function). The property of being functional either can be expressed in first-order logic and proven as an additional property or left as an assumption that the engineer guarantees is correct.

The recursive definitions can be seen as axioms that the model is supposed to satisfy, and we want to prove some property assuming that these axioms are true. So the type of formula we are interested in, is defined as follows. A first-order formula with recursive definitions is of the form $\varphi = \varphi_A \rightarrow \varphi_P$, where φ_A is a conjunction of first-order formulas and recursive definitions (the "axioms"), and φ_P is a first order formula (the "property"). We are interested in validity of such formulas, which is equivalent to unsatisfiability of their negation $\varphi_A \wedge \neg\varphi_P$.

Least fixpoints are not definable in first-order logic. If we want to reason about first-order formulas with recursive definitions within first-order logic, then we can replace the $\leftrightarrow_{\text{LFP}}$ in the recursive definitions φ_R by the standard equivalence $\leftrightarrow$, which yields the pure FO formula $\forall \overline{x} : \overline{\tau}.R(\overline{x}) \leftrightarrow \rho(\overline{x})$. One can view this step as an abstraction that admits models in which R is interpreted as any fixpoint of μ_ρ, not necessarily the least fixpoint. Instead of syntactically changing the formula in order to transform recursive definitions into standard FO formulas, we use two different semantics for FO formulas with recursive definitions. We write $M \models_{\text{LFP}} \varphi$ to denote the fact that the FO part of φ is satisfied in M and that recursively defined relations R are interpreted as least fixpoints of their definitions. We write $M \models_{\text{FO}} \varphi$ if φ is satisfied in M when $\leftrightarrow_{\text{LFP}}$ is interpreted as $\leftrightarrow$. We refer to these two semantics as LFP-semantics and FO-semantics, respectively. Since in the FO-semantics the least fixpoint definitions are simply interpreted by (arbitrary) fixpoints, we also refer to this as fixpoint semantics or FP-semantics.

The following observations are direct consequences of the definitions.

Remark 1. *Let φ be an FO formula with recursive definitions.*

(1) $M \models_{LFP} \varphi$ implies that $M \models_{FO} \varphi$ for any model M.

(2) If φ is valid in the FO-semantics, then φ is valid in the LFP-semantics.

In Section 6 we substitute the occurrences of R in $\rho(\overline{x}, R)$ by other first-order formulas ψ of the same sort as R. We denote this by $\rho(\overline{x}, R \leftarrow \psi)$, which is the formula obtained from $\rho(\overline{x}, R)$ by replacing every occurrence of $R(t_1, \dots, t_k)$ for terms $t_1, \dots, t_k$ in ρ by $\psi(t_1, \dots, t_k)$.

In Section 3 and 4 we use first-order logic without recursive definitions. We come back to recursive definitions in Sections 5 and 6.

3 PRESERVATION OF SATISFACTION UNDER TERM INSTANTIATION

In this section, we discuss the completeness of using term instantiation to prove validity of universally quantified formulas in first-order logic. We omit the proof of this result as it is an instance of our main technical contribution in Section 4, but it helps lay the groundwork for the more general case discussed in that section.

Assume one has a signature $\Sigma = (\{\sigma_1\}; F; \mathcal{R})$, with function symbols in F and relation symbols in $\mathcal{R}$ purely uninterpreted. Note a single sort σ_1 is equivalent to the unsorted first-order case. In general we would like to show the validity of a formula of the form $\psi = \varphi_1 \wedge \dots \wedge \varphi_{n-1} \longrightarrow \neg\varphi_n$, where each φ_i is universally quantified. Existential quantifiers can be eliminated through Skolemization: $\exists x.\ \forall y.\ \exists z.\ R(x, y, z)$ can be transformed to the formula $\forall y.\ R(c, y, f(y))$, where we augment our signature with the uninterpreted constant symbol c and uninterpreted function symbol f. Note the validity of ψ is equivalent to the unsatisfiability of $\varphi = \neg\psi = \varphi_1 \wedge \dots \wedge \varphi_n$, which is the form we assume our formulas take in the sequel.

The term instantiation technique for which we show completeness proceeds intuitively as follows, assuming φ is a conjunction of universally quantified formulas: begin with all ground terms present in φ. Then, exhaustively instantiate the universally quantified formulas in φ with those ground terms. If any resulting formula is unsatisfiable, then φ is unsatisfiable. Otherwise, repeat the process while also using new ground terms introduced from the previous instantiation. With this, we are able to reduce validity in first-order logic to unsatisfiability of quantifier-free formulas over uninterpreted functions with equality.

We note the similarity to Herbrand's Theorem [Ebbinghaus et al. 1994; Schöning 2008], which also uses term instantiation to prove completeness of FOL by a reduction to propositional logic. Our instantiation differs from Herbrand's Theorem in two aspects. First of all, we do not need to instantiate the equality axioms since we reduce to quantifier-free formulas over uninterpreted functions with equality instead of propositional logic. Second, we do not use all terms for instantiation but only those occurring as subterms in the formulas during instantiation. For example, if in the formula φ a function f occurs only in a term $f(g(x))$, then our instantiation only builds terms in which f is applied to terms starting with g.

3.1 Example

Consider the following problem:

Example 3.1. Assume some arbitrary property P holds on all elements of an array a. Let b be identical to a on all indices besides the index k, where b holds the element v at that index. Assume $P(v)$ holds. Then, P holds on all elements of the array b.

Encoded in first-order logic, this results in the following formulas:

$$\begin{aligned}
&\varphi_1 := \forall i.\ P(a(i)) && (P \text{ holds for all elements of } a)\\
&\varphi_2 := \forall i.\ i \neq k \to b(i) = a(i) && (b \text{ agrees with } a \text{ on all indices other than } k)\\
&\varphi_3 := b(k) = v \land P(v) &&\\
&\varphi_4 := \forall i.\ P(b(i)) && (P \text{ holds for } b\text{, becomes } \neg P(b(k')) \text{ when negated})
\end{aligned}$$

Note that we do not use any further axioms for arrays or indices but consider them as uninterpreted objects in this example.

Let $\psi = \varphi_1 \land \varphi_2 \land \varphi_3 \to \varphi_4$. Note a and b are uninterpreted functions, with k and v as constants. We want to check the unsatisfiability of $\varphi = \varphi_1 \land \varphi_2 \land \varphi_3 \land \neg\varphi_4$. The negation of φ_4 creates an existential quantifier that we replace by a new constant k'.

Initially, the term instantiation will use the ground terms $\{k, k', v, b(k), b(k')\}$. The term $b(k)$ is superfluous, since it is equal to v. Among formulas present in the first round of instantiation are

$$\begin{aligned}
&\varphi_1' := P(a(k')) && \varphi_2' := k' \neq k \to b(k') = a(k')\\
&\varphi_3' := b(k) = v \land P(v) && \varphi_4' := \neg P(b(k'))
\end{aligned}$$

It is not difficult to see that this set of formulas is unsatisfiable. Of course, other formulas will be present in the instantiation that are not relevant, such as $P(a(v))$, but this will not affect unsatisfiability. Should we not have found the resulting set of formulas unsatisfiable, we would continue instantiation with new ground terms, such as $a(k')$ and $b(k')$.

It is clear that this process is sound: we only are instantiating universal quantifiers with elements already in our domain. The difficulty comes with showing that this process is in fact complete, that if φ is unsatisfiable then we will ultimately find a set of quantifier-free formulas witnessing the unsatisfiability through this process.

3.2 Proof Sketch

We sketch the proof here, and lay out the lemmas needed for the full proof. First, we create a set X^* of all fully-instantiated formulas from our instantiation process. Next, given any model M of X^*, we create a model M^* such that $M \models X^*$ iff $M^* \models \varphi$. For most of this model construction, we can simply restrict the interpretation for function and relation symbols from M to the domain in M^*. The only hitch in this procedure is the interpretation for function symbols f on elements in the universe of M^* whose result may not be in the universe of M. However, for these, it is sufficient to map $[\![f]\!]_{M^*}$ to an arbitrary element in the universe of M^*. Since the result was not in the universe of M, it will not be used in the instantiation process, so it is sufficient to map it to an arbitrary element in the model M^*.

Next, we prove by induction on the term structure that the interpretation of any term arising from the instantiation process is the same in both M and M^*. We use this lemma to prove that all subformulas of formulas in X^* are equisatisfiable, which is used to prove that φ is satisfiable if and only if X^* is satisfiable. Thus, we show that if a formula is unsatisfiable, there will be a level within the instantiation process witnessing the unsatisfiability (and vice versa).

Specifically, the unfolding process proceeds as follows: Let $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_n$ be a closed Σ-formula, where each φ_i is of the form $\forall \overline{x}.\ \psi_i(\overline{x})$ for quantifier-free ψ_i. Define X^* and T^* as follows:

$$\begin{aligned}
T_0 &= \{t \mid t \text{ is a ground term occurring in } \psi\}\\
X_0 &= \{\psi_i[\overline{t}/\overline{x}] \mid \text{each term in } \overline{t} \text{ is in } T_0\}\\
T_{n+1} &= T_n \cup \{t \mid t \text{ is a ground term occurring in a formula from } X_n\}\\
X_{n+1} &= X_n \cup \{\psi_i[\overline{t}/\overline{x}] \mid \text{each term in } \overline{t} \text{ is in } T_{n+1}\}\\
X^* &= \bigcup_{i\geq 0} X_i \text{ and } T^* = \bigcup_{i\geq 0} T_i
\end{aligned}$$

We assume T^* is nonempty. $\psi = \neg\varphi = \varphi_1 \wedge \ldots \wedge \varphi_{n-1} \rightarrow \neg\varphi_n$. Our main theorem in this section is below:

Theorem 3.2 (Completeness of Term Instantiation). *ψ is valid if and only if there is an m such that X_m is unsatisfiable.*

As explained above, to prove this we need we need to construct a model M^*. First, we observe that each formula in X^* is quantifier-free, and is implied by some formula φ_i. X^* is thus *weaker* than φ.

Let $M = (U; [\![\cdot]\!]_M)$ be a model that satisfies ψ. As we are in the single-sorted case here, U simply represents a universe for sort σ_1 rather than a family of universes. Let

$$U^* = \{u \in U \mid u = [\![t]\!]_M \text{ for some } t \in T^*\}$$

Now, pick an arbitrary $u^* \in U^*$. Such a u^* exists since T^* is assumed to be nonempty. Define the model $M^* = (U^*; [\![\cdot]\!]_{M^*})$, where for any $R \in \mathcal{R}$, $[\![R]\!]_{M^*} = [\![R]\!]_M \restriction_{U^*}$, and for any $f \in F$ and $u_1, \ldots, u_n \in U^*$,

$$[\![f]\!]_{M^*}(u_1,\ldots,u_n) = \begin{cases} [\![f]\!]_M(u_1,\ldots,u_n) & \text{if } [\![f]\!]_M(u_1,\ldots,u_n) \in U^* \\ u^* & \text{otherwise} \end{cases}$$

With this concrete model construction, we use the following lemmas to prove Theorem 3.2.

Lemma 3.3. $\forall t \in T^*, [\![t]\!]_M = [\![t]\!]_{M^*}$

Lemma 3.4. *Let ψ' be a subformula of a formula in X^*. Then $M \models \psi' \iff M^* \models \psi'$.*

Lemma 3.5. *For any model M, if $M \models X^*$, then $M^* \models \varphi$.*

Lemma 3.6. *φ is satisfiable if and only if then X^* is satisfiable*

Proof of Theorem 3.2. Note that ψ is valid $\Leftrightarrow$ $\neg\psi = \varphi$ is unsatisfiable $\Leftrightarrow$ (by Lemma 3.6) X^* is unsatisfiable. By compactness of FOL, X^* is unsatisfiable if and only if there is some finite set $Y \subseteq X^*$ such that Y is unsatisfiable. But since $Y \subseteq X^*$, we must have that $Y \subseteq X_m$ for some m. Since Y is unsatisfiable, so is X_m, and this completes the proof. $\square$

3.3 Decidability

Though our main contribution regards checking validity, it is important to note in which cases the unfolding yields a decision procedure for the satisfiability problem.

Lemma 3.7. *Let $m \geq 0$ be some number, and consider a class of formulas such that for all formulas form this class in the instantiation process $X_{m+1} = X_m$. Then the satisfiability for this class of formulas is decidable.*

Proof. Follows immediately from Lemma 3.6, the observation that $X^* = X_m$ and X_m is finite, and that the quantifier-free theory of uninterpreted functions with equality is decidable. □

While this result is not surprising, some decidable classes are subsumed in this fragment. For example, the Bernays-Schönfinkel class [Börger et al. 1997] is the set of sentences with a prefix of the form $\exists^*\forall^*$ followed by a quantifier-free formula containing no function symbols. Each formula in this class will have a finite unfolding as above, since the quantifier-free formula has no function symbols, and no new function symbols are introduced when eliminating the existential variables, as all existentially quantified variables occur before any universally quantified variables. Thus, we can apply Lemma 3.7 for any formula in this class to give an alternate proof of the decidability of the satisfiability problem for the Bernays-Schönfinkel class.

4 TERM INSTANTIATION IN THE PRESENCE OF BACKGROUND THEORIES

We have shown that the method of term instantiation is complete for proving universally quantified statements over uninterpreted functions. We now generalize these ideas to a setting that admits the use of background theories. As explained in Section 2, our logic uses sorts $\sigma_1, \ldots, \sigma_m$, and we assume background theories for the sorts $\sigma_2, \ldots, \sigma_m$. The satisfaction of a formula is considered modulo the background theories, that is, $M \models \varphi$ implicitly also means that M restricted to sort σ_j for $j \geq 2$ is a model of the background theory for σ_j. Validity of φ in this context means that all models that satisfy the the background theories also satisfy φ.

As in the previous section, proving validity of a formula is reduced to unsatisfiability of its negation. In the purely uninterpreted case we can deal with arbitrary formulas by eliminating all existential quantifiers using Skolemization. For the instantiation in the presence of background theories, we require that the quantifiers over sort σ_1 are the outermost ones. Using Skolemization we again reduce those to universal quantifiers only. Hence, the formula that is tested for unsatisfiability is a conjunction $\varphi = \bigwedge_{i=1}^{n} \varphi_i$ where each φ_i is of the form $\forall \overline{x} : \sigma_1.\ \psi_i(\overline{x})$, where $\psi_i(\overline{x})$ may quantify over sorts σ_j (with $j \neq 1$), but does not quantify over sort σ_1.

The instantiation itself is then defined in the same way as in Section 3, iteratively instantiating the universally quantified variables of sort σ_1 by ground terms of sort σ_1:

$$\begin{aligned}
T_0 &:= \{t \mid t \text{ is a ground term of sort } \sigma_1 \text{ occurring in } \varphi\}\\
X_0 &:= \{\psi_i[\overline{t}/\overline{x}] \mid \text{ each term in } \overline{t} \text{ is in } T_0\}\\
T_{n+1} &:= T_n \cup \{t \mid t \text{ is a ground term of sort } \sigma_1 \text{ occurring in a formula from } X_n\}\\
X_{n+1} &= X_n \cup \{\psi_i[\overline{t}/\overline{x}] \mid \text{ each term in } \overline{t} \text{ is in } T_{n+1}\}\\
X^* &= \bigcup_{i\geq 0} X_i \text{ and } T^* = \bigcup_{i\geq 0} T_i
\end{aligned}$$

In this definition, we assume that at least one ground term of sort σ_1 occurs in one of the formulas φ_i. Otherwise, T_0 is defined to contain an arbitrary constant of sort σ_1. Then T^* and X^* are nonempty.

Our aim is to show that X^* is unsatisfiable if, and only if, φ is unsatisfiable. One direction is easy.

Lemma 4.1. *Each model of φ is a model of X^*.*

Proof. A model of φ satisfies each $\psi_i(\overline{x})$ for all interpretations of the variables in $\overline{x}$, in particular for the interpretations by the ground terms in all the T_i. Hence it is also a model of X^*. □

For the other direction, we want to modify a model for X^* such that it becomes a model of φ, such that only the σ_1 part is modified. This is crucial since in applications we assume that the model for X^* satisfies the background theories for sorts $\sigma_2, \ldots, \sigma_m$. However, such a modification is not possible in general, as illustrated by the following example (the example is artificial to keep it small, one can also come up with natural examples involving, e.g., numbers). In fact, the example shows that term instantiation over the foreground theory is not complete, in general. The set of instantiated formulas is not equisatisfiable with the original formula. One can extend this example to a more complex one showing that even an instantiation including the variables of background sorts does not lead to a complete procedure.

Example 4.2. Assume we have sorts σ_1, σ_2 with one constant d of sort σ_1, and an uninterpreted unary function $g : \sigma_2 \longrightarrow \sigma_1$. For sort σ_2 we have two constants c_1, c_2, and the background theory requires $c_1 \neq c_2$.

Let $\varphi = \varphi_1 \wedge \varphi_2$ with $\varphi_1 := \forall x : \sigma_1.\ x = d$ and $\varphi_2 := \forall y, z : \sigma_2.\ y \neq z \longrightarrow g(y) \neq g(z)$. These formulas state that d is the only element of sort σ_1, and that the function g is injective. Note that φ_2 does not contain variables of sort σ_1, so $\varphi_2 = \psi_2$ with an empty quantifier prefix of universal quantifiers over σ_1. The only ground term of sort σ_1 is d. Thus, $X^* = \{d = d, \varphi_2\}$ (x in φ_1 is instantiated by d, and in φ_2 there is no variable to be instantiated).

A possible model for X^* is the following one: There are elements d, d' of sort σ_1, and c, c' of sort σ_2 with $g(c) = d$ and $g(c') = d'$. However, the original formula is certainly unsatisfiable because it requires $d = g(c_1) \neq g(c_2) = d$.

The problem in this example is comes from the fact that properties of the σ_2-part of the model imply properties of the σ_1-part but these properties are not captured by the term instantiation. To avoid this, we assume a restriction on the terms used in φ, as defined below.

4.1 Completeness of Term Instantiation for Safe Formulas

A term t is called σ_1-guarded if it only contains variables of sort σ_1. We say that the formula φ is σ_1-safe (or just safe, for short) if all subterms of sort σ_1 that occur in φ are σ_1-guarded. Note that the formula φ_2 from Example 4.2 violates the condition because it contains, for example, the term $g(y)$ which is of sort σ_1 but contains the variable y of sort σ_2.

We point out that φ being safe is a natural restriction that will usually be satisfied in applications. There are at least two natural scenarios that imply that φ is safe. Obviously, φ is safe if it does not use quantification over sorts $\sigma_2, \ldots, \sigma_m$ because then all variables are of sort σ_1. In the other scenario, every uninterpreted function f with range σ_1 has domain σ_1^k (so there are no functions producing σ_1 elements from arguments that are not of sort σ_1). We refer to such functions as one-way functions. In this case, each term of sort σ_1 only has subterms of sort σ_1 (in particular the variables) and is therefore σ_1-guarded. This is summarized in the following proposition.

Proposition 4.3. *If φ only quantifies over the elements of the foreground theory, or if the signature only contains one-way functions, then φ is safe.*

We now explain how to modify the σ_1-part of a model for X^* such that it becomes a model of φ under the assumption that φ is safe. Its definition does not depend on the fact that φ is safe, only its correctness (in the proof of Lemma 4.5 below).

Let $M = (U; [\![\cdot]\!]_M)$ be a model with $M \models X^*$, where $U = \{U_{\sigma_1}, \ldots, U_{\sigma_m}\}$ is a set of universes for each sort, and $[\![\cdot]\!]_M$ assigns concrete functions $[\![f]\!]_M$ and relations $[\![R]\!]_M$ to the corresponding symbols in the signature.

The construction of M^* is very similar to the construction presented in Section 3. Let

$$U^*_{\sigma_1} = \{u \in U_{\sigma_1} \mid u = [\![t]\!]_M \text{ for some } t \in T^*\}$$

and let $S^* = \{U^*_{\sigma_1}, U_{\sigma_2}, \ldots, U_{\sigma_m}\}$. Now, pick an arbitrary $u^* \in U^*_{\sigma_1}$. Such a u^* exists since T^* is nonempty. Define the model $M^* = (S^*; [\![\cdot]\!]_{M^*})$ as follows:

(1) The relation $[\![R]\!]_{M^*}$ is the restriction of $[\![R]\!]_M$ to the new domain S^*.

(2) Let $f : \sigma_{i_1} \times \cdots \times \sigma_{i_k} \to \sigma_j$ be a function symbol.
  (a) If $j \neq 1$, then $[\![f]\!]_{M^*}$ is the restriction of $[\![f]\!]_M$ to the new domain (the range did not change).
  (b) if $j = 1$, then $[\![f]\!]_{M^*}(u_1, \ldots, u_n) = \begin{cases} [\![f]\!]_M(u_1, \ldots, u_n) & \text{if } [\![f]\!]_M(u_1, \ldots, u_n) \in U^*_{\sigma_1} \\ u^* & \text{otherwise.} \end{cases}$

We first show that the interpretation of the terms in T^* are the same in M and M^*.

Lemma 4.4. *Let t be a subterm of a term in T^*. Then $[\![t]\!]_M = [\![t]\!]_{M^*}$.*

Proof. By induction on the structure of t. Note that the terms in T^* are all of sort σ_1 but may contain subterms of different sorts. So we have to consider these cases, too.

If $t = c$ is a constant of sort σ_1, then $[\![c]\!]_M \in U^*_{\sigma_1}$ by definition of $U^*_{\sigma_1}$ and thus $[\![c]\!]_{M^*} = [\![c]\!]_M$ by definition (2b) of the functions in M^*. If $t = c$ is a constant of sort different from σ_1, then its interpretation is the same in M and M^* by definition (2a) of the functions in M^*.

If $t = f(t_1, \ldots, t_n)$, then $[\![t_i]\!]_M = [\![t_i]\!]_{M^*}$ by induction. If t is of sort different from σ_1, then the claim follows again by definition (2a) of the functions in M^*. If t is of sort σ_1, then

$$[\![t]\!]_{M^*} = [\![f]\!]_{M^*}([\![t_1]\!]_{M^*}, \ldots, [\![t_i]\!]_{M^*}) = [\![f]\!]_{M^*}([\![t_1]\!]_M, \ldots, [\![t_i]\!]_M) =^{(\dagger)} [\![f]\!]_M([\![t_1]\!]_M, \ldots, [\![t_i]\!]_M) = [\![t]\!]_M$$

where ($\dagger$) follows from (2b) of the definition of the functions in M^* and the fact that $[\![t]\!]_M = [\![f]\!]_M([\![t_1]\!]_M, \ldots, [\![t_i]\!]_M)$ is an element of $U^*_{\sigma_1}$. □

We now lift Lemma 4.4 to the formulas in X^*.

Lemma 4.5. *Let $\psi(\overline{y})$ be a subformula of a formula in X^* and let $\overline{u}$ be a sequence of elements from S^* matching the sorts of the variables in $\overline{y}$. If φ is safe, then $M \models \psi(\overline{u})$ iff $M^* \models \psi(\overline{u})$.*

Proof. Note that all variables of sort σ_1 have been instantiated by ground terms in the formulas of X^*, and therefore the variables in $\overline{y}$ are of sorts different from σ_1.

We prove the claim by induction on the structure of ψ. For the base case, ψ is of the form $R(t_1, \ldots, t_n)$ for a predicate R, or of the form $t_1 = t_2$. We claim that all subterms of the terms t_i (with variables from $\overline{y}$ substituted by the corresponding elements from $\overline{u}$) evaluate to the same elements in M and M^*. Subterms of sort σ_1 must be ground terms because φ is safe and all variables of sort σ_1 have been instantiated by ground terms. Therefore they are in T^* and the claim follows from Lemma 4.4. For the other subterms it follows from the fact that the models M and M^* are the same for all sorts different from σ_1. Thus, by definition of the relations in M^*, the case of atomic formulas follows.

The Boolean connectives are immediate, and if ψ is of the form $\forall y' : \sigma.\ \psi'(\overline{y}, y')$ where $\sigma \neq \sigma_1$, we conclude as follows:

$$\begin{aligned} M \models \forall y' : \sigma.\ \psi'(\overline{u}, y') \iff M \models \psi'(\overline{u}, u') \text{ for all } u' \in U_\sigma &\overset{(\dagger)}{\iff} M^* \models \psi'(\overline{u}, u') \text{ for all } u' \in U_\sigma \\ &\iff M^* \models \forall y' : \sigma.\ \psi'(\overline{u}, y') \end{aligned}$$

where ($\dagger$) holds by induction.

The case of existential quantification can be obtained from the Boolean laws and the case of universal quantification. □

Lemma 4.6. *Let φ be safe. If $M \models X^*$, then $M^* \models \varphi$.*

Proof. Since each φ_i is of the form $\forall \overline{x} : \sigma_1.\ \psi_i(\overline{x})$, we need to show $M^* \models \psi_i(\overline{u})$ for all tuples $\overline{u}$ of elements from $U^*_{\sigma_1}$. Indeed, for each $\overline{u}$, we know there is a tuple $\overline{t}$ of terms from T^* such that $\overline{u} = [\![\overline{t}]\!]_M$ by definition of U^*. The formula $\psi_i(\overline{t})$ is contained in X^*, and thus $M \models \psi_i(\overline{t})$ since we assume that $M \models X^*$. Lemma 4.5 yields that $M^* \models \psi_i(\overline{t})$. By Lemma 4.4, $[\![\overline{t}]\!]_M = [\![\overline{t}]\!]_{M^*}$, and thus $M^* \models \psi_i(\overline{u})$. □

We can now formulate the completeness result of term instantiation in the presence of background theories for safe formulas.

Theorem 4.7 (Completeness of Term Instantiation with Background Theories). *Let φ be a safe formula over a combination of theories. If φ is unsatisfiable, then there is a level X_i of the term instantiation such that X_i is unsatisfiable.*

Proof. If φ is unsatisfiable, then X^* is unsatisfiable by Lemma 4.6. By compactness of FOL, there is a finite subset of X^* that is unsatisfiable, which is included in one of the X_i. □

The term instantiation without background theories described in Section 3 produces a sequence of finite sets X_i of quantifier-free formulas over the theory of uninterpreted functions with equality, for which satisfiability is decidable. If φ is a safe formula over a combination of theories, then the formulas in X_i are quantifier free if φ does not quantify over the background sorts. If for all the background theories satisfiability of quantifier-free formulas is decidable, then the decision procedures can be combined to a decision procedure for the combined theory using the method of Nelson-Oppen [Nelson and Oppen 1979]. This combination methods requires the background theories to be stably infinite [Oppen 1980], where a theory is stably infinite if each satisfiable quantifier-free formula of that theory is satisfiable in an infinite model of the theory. Hence, satisfiability is decidable for each set X_i in this setting. By Theorem 4.7, this yields a semi decision procedure for unsatisfiability of safe formulas without quantification over the background structures, as stated in the following corollary.

Corollary 4.8. *Consider a combination of stably infinite theories such that satisfiability of quantifier-free formulas is decidable for the background theories. Then term instantiation yields a semi-decision procedure for formulas that do not quantify over the elements of the background structures.*

If the signature does not contain uninterpreted function symbols (only relation symbols and constants), then the only ground terms of sort σ_1 are constants, which means that $X^* = X_0$ is finite. In the setting without background theories, this observation leads to the decidability of satisfiability for formulas of the form $\exists\overline{x}.\forall\overline{y}.\psi(\overline{x},\overline{y})$ for quantifier-free φ, known as the Bernays-Schönfinkel class of formulas [Börger et al. 1997]. Based on Corollary 4.8, we can extend this result to the setting with background theories assuming that there are no function with range σ_1 in the signature (so there are no terms of sort σ_1 other than constants).

Proposition 4.9. *Consider a combination of stably infinite theories such that satisfiability of quantifier-free formulas is decidable for the background theories, and the signature does not contain functions with range σ_1. Then satisfiability is decidable for formulas of the form $\exists\overline{x} : \sigma_1.\forall\overline{y} : \sigma_1.\psi(\overline{x},\overline{y})$, where ψ is quantifier-free.*

5 RAMIFICATIONS AND APPLICATIONS OF COMPLETE TERM INSTANTIATION FOR THE SAFE FRAGMENT

We now explore some of the ramifications of our main theorem on the soundness and completeness of systematic term instantiation for the safe fragment of FOL (in particular, Theorem 4.7 and Corollary 4.8).

5.1 Explaining the Efficacy of Natural Proofs and the Unfold-and-Match Technique

There are several techniques in current literature where validation engines find proofs using the unfold-and-match technique, wherein the negation of the formula is proved unsatisfiable by "unfolding" recursive definitions followed by a proof of unsatisfiability that uses simple derivations of the resulting formulas. We concentrate in this section on one such paradigm called "natural proofs", which is a formalization of this tactic. Though the results herein are for natural proofs, they apply also to other unfold-and-match paradigms in the literature.

In the setting of natural proofs for heap verification [Pek et al. 2014; Qiu et al. 2013], a dialect of separation logic called Dryad is encoded using first order logic with recursive definitions, and with background theories of numbers (that capture key values in data-structures and recursive functions such as length and height of structures) and background theory of sets (that model heaplets required to capture separation logic semantics and also to capture collections such as the set/multiset of keys stored in a data structure). In this section, we use our main theorem (Theorem 4.7) to give foundational reasons for the efficacy of the heuristics used in natural proofs.

Recall the fundamental problem of finding sound and complete procedures for checking validity for undecidable logics in the realm of program verification. Program configurations are inherently *finite* in nature (even with unbounded heaps, unbounded stacks, unbounded data-structures, and arbitrarily large integers) and hence are enumerable. Since model-checking logics are (almost) always decidable on finite structures, satisfiability is r.e. (recursively enumerable). Consequently, a sound and complete procedure simply cannot exist for validity, if the underlying logic is undecidable.

We propose that natural proofs, as presented in [Pek et al. 2014; Qiu et al. 2013], is best understood in two phases:

Phase 1: Abstracting least fixpoints to fixpoints:
Natural proofs first translate recursive definitions in separation logic that have least fixpoint definitions to recursive equations, which essentially impose only a fixpoint semantics (this corresponds to moving from LFP-semantics to FO-semantics in the terminology of Section 2.3).

Phase 2: Heuristics involving unfolding of recursive definitions:
Natural proofs then unroll recursive definitions (expanding recursive definitions on the footprint of the program segment), and send the resulting quantifier-free formulas to solvers for the combination of background theories and uninterpreted functions.

In the sequel, we assume that natural proofs in Phase 2 unroll recursive definitions systematically, ad infinitum (and call this natural proofs with systematic unbounded unfolding).

Phase 1 of the above procedure is *crucial* in side-stepping the fundamental block of building sound and complete verification engines for validating verification conditions in program verification. By moving from least fixpoints to fixpoint-based definitions, the logic allows *infinite* models, that cannot be enumerated, and hence satisfiability is *not necessarily r.e.* under this semantics, opening the possibility of validity being r.e.

In this section, our main aim is to show that Phase 2, in the context of verification of heap properties, is actually *not a heuristic!* We show that natural proofs with systematic unbounded unfolding is actually a sound and complete procedure for validity.

Before we delve into Phase 2, we describe the subtleties of moving from least fixpoint semantics to fixpoint semantics, and describe the non-standard models that this change brings. We will discuss more sophisticated methods that handle the inaccuracy Phase 1 brings in the next section, Section 6.

5.2 Moving from Least Fixpoint Semantics to Fixpoint Semantics

Let us illustrate the difference in semantics using an example. Consider linked lists with a field pointer n. A formalization of this using heaplets could be:

$$lseg(x,y) \leftrightarrow (x=y \wedge h\_lseg(x,y)=\emptyset) \;\vee (x\neq y \wedge lseg(n(x),y) \wedge x \notin h\_lseg(n(x),y))$$

where h_lseg is defined as follows, using the the standard if-then-else connective *ite*:

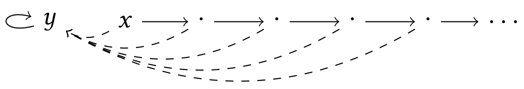

Fig. 1. A model in which *lseg* is not interpreted as the least fixpoint.

$$h\_lseg(x,y) = ite(x = nil,\ \emptyset,\ \{x\} \cup h\_lseg(n(x),y)).$$

In the least fixpoint semantics, this definition of *lseg* does indeed capture the linked lists precisely.

It may be tempting to think that the definition without the least fixpoint semantics is correct, too. In fact, it is easy to see that in any model, if x to y does form a finite list segment, then $lseg(x,y)$ would definitely hold, and if x points to a finite circular list that does not reach y, then the $lseg(x,y)$ would definitely be false. The reason that the above does not capture precisely linked list segments is that if x points to an *infinitely* long list that does not reach y, then one can interpret $lseg(x,y)$ to be true (as well as $lseg(z,y)$ to be true for all nodes z reachable from x) and still satisfy the constraints. Such a model is depicted in Figure 1, where the solid edges indicate the next pointer, and the additional *lseg*-edges (outside the least fixpoint) are indicated by dashed arrows.

One can try to fix this using a theory of *finite* sets for the heaplets or using induction on the length of lists, where length is modeled using a background theory of natural numbers; however, it turns out that that too will not work, as we discuss later in Section 5.4.

However, if we prove that a verification condition is valid with respect to the above fixpoint semantics, then it will of course be valid with respect to the least fixpoint semantics (see Remark 1).

5.3 The Completeness of Natural Proofs with Fixpoint Semantics

We now show that such recursive definitions (interpreted using fixpoint semantics/FO-semantics) are indeed amenable to sound and complete validation using systematic term instantiation. In other words, systematic term instantiation done in Phase 2 is not a heuristic, but a complete method.

Our first goal is to show that heaps can be modeled in our UCT framework consisting of an uninterpreted combination of standard decidable background theories. We will then show that typical verification conditions fall into the safe fragment (as defined in Section 4.1), and in fact satisfy *both conditions* of having one-way functions as well as not quantifying over background theories, which ensure that the formulas are safe according to Proposition 4.3. This will lead us to conclude that systematic instantiation done in natural proofs is in fact sound and complete.

In the heap verification setting, we can model the heap locations and pointers to heap locations using a foreground theory of uninterpreted functions, along with a background theory of natural numbers, integers, and sets of natural numbers/integers, and other theories that are both stably infinite [Oppen 1980] and have a decidable satisfiability problem for quantifier free first-order logics over their respective signatures.

Consider the kind of functions and relations we would need to talk effectively about typical heap properties:

- Pointers to heap locations are naturally modeled as functions from the foreground type to the foreground type.
- Data elements stored in the data-structure, for instance the keys stored in a tree data-structure, can be modeled using a function key that maps heap locations in the foreground type to data-elements in the background type (like integers or natural numbers with addition, or Presburger Arithmetic).

- The *set* of keys stored in a data-structure can be modeled using maps from tuples of heap locations (that define the boundaries of the data structure) to the keys stored in them. For instance, the keys stored in a list-segment between two locations can be defined as a function from pairs of locations to sets of keys. The background types will offer functions and relations (such as $\in$ and $\subseteq$) to compare these sets, and will adhere to standard set-theoretic axioms.
- The *heaplet* of a data-structure can be also be modeled as a function that maps tuples of locations (that define the boundaries of a structure, typically) to sets modeled as first class elements in the background theory. For instance, the heaplet of a linked list segment from a location u to a location v, can be seen as $h\_lseg(u, v)$, mapping to a set of integers that model the addresses of the heap.
- The definitions of data-structures (such as trees, list segments, binary search trees, etc.) can then be defined using the above functions and relations, and form predicates on tuples in the foreground universe.

The crucial aspect of the above modeling to note is that it only uses functions that are from the foreground universe to the foreground universe (such as function modeling pointer fields) or whose range is the background universe (key, set-of-keys, heaplet, etc.). Hence, the logic naturally falls into the *safe fragment* of the many-sorted logic, as it satisfies the one-way function restriction (Proposition 4.3). This guarantees that systematic term instantiation is complete, by Theorem 4.7.

Moreover, in the setting of verification, in the negated verification condition that we check for satisfiability, it is naturally the case that the universal quantification is over the foreground theory. The scalar variables (like integer variables, etc.) the program accesses in a basic path are modeled using existential quantification, while it is natural to use universal quantification to express properties of the unbounded heap. There can be *implicit* unbounded quantification of the background universe by applications of functions on universally quantified variables over the foreground universe (such as saying a linked list is sorted), but this is allowed by the safe fragment.

Consequently, not only are the formulas generated in the heap verification setting in [Pek et al. 2014; Qiu et al. 2013] result in the safe fragment, each level of term instantiation gives a *entirely quantifier-free* formula over uninterpreted functions and the background theories. The resulting instantiated formulas at each level hence fall into the quantifier-free fragments of the combined background theories, which is decidable by Nelson-Oppen combination of the individual decidable theories [Nelson and Oppen 1979; Oppen 1980].

It turns out that the entire class of formulas generated in the work of natural proofs of heap verification reported in [Pek et al. 2014; Qiu et al. 2013], where recursive definitions are treated with fixpoint semantics, falls in the above category, leading us to conclude the following:

Result #1: *All verification conditions generated in the work on natural proofs for heap verification, as reported in [Pek et al. 2014; Qiu et al. 2013], with recursive definitions given fixpoint semantics, fall into the safe fragment of many-sorted logic with background theories, where each background theory has a decidable quantifier-free satisfiability problem and is stably infinite. Consequently, each level of term instantiation results in a formula that is decidable, and furthermore, the procedure that successively performs term instantiation followed by deciding satisfiability of the instantiated formulas is a sound and complete procedure.*

We believe that the above explains the efficacy of the instantiation technique used in natural proofs— the procedure, despite seeming to resort to heuristics, is actually sound and complete for fixpoint semantics of the underlying recursive definitions.

Note that the work reported in [Pek et al. 2014; Qiu et al. 2013] for natural proofs on heap verification further have a simplification that only does a particular fixed term instantiation. Our result only shows that repeated instantiation is complete; we do not claim to give any rationale for the heuristic that deterministically instantiated terms. Of course, for general FOL (or even safe FOL) over a UCT with even Presburger arithmetic as background, no deterministic instantiation can be complete, for otherwise validity would be *decidable*, which we know is false.

5.4 Nonstandard Models and Inductive Definitions

We now describe a subtle aspect of many-sorted logics with background theories that explains the inability of these logics to capture the least fixpoint. First, given a background theory of natural numbers with their underlying induction axioms, it is tempting to define structures that are inductive on a measure based on natural numbers. As a simple example, consider defining a list as follows. This does not follow the definition obtained from separation logic as it does not use heaplets, but a similar one can be defined using heaplets as well; in fact, work in separation logic suggests such a definition that inducts using natural numbers [O'Hearn 2012].

$$n(nil) = nil \wedge list(nil) \wedge len(nil){=}0 \wedge \forall y.\ (list(y) \wedge len(y) = 0) \rightarrow y = nil$$
$$\wedge\ \forall y, z.\ \ (y \neq nil \wedge n(y){=}z) \rightarrow (list(z) \leftrightarrow (list(y) \wedge len(z) = len(y) + 1))$$

The above says that *nil* is a 0-length list and the only one, and whenever $n(y) = z$, one is a list iff the other is a list, and the length of the list at y is one more than the list at z. When *len* is interpreted to map to natural numbers, we invite the reader to verify that the above precisely defines lists—those nodes that reach *nil* in finitely many applications of $n()$. If $list(x)$ is true, we can walk to successive nodes $n(x), n(n(x)), \ldots$ by the last conjunct that point to shorter and shorter lists, and hence must reach *nil*.

However, perhaps surprisingly, the above *does not* capture the intended meaning of lists in the many-sorted logic with the background theory of Presburger arithmetic. The reason is that Presburger arithmetic (in fact, any FO axiomatic scheme for numbers) does not capture precisely the *standard model of arithmetic, but always allows non-standard models as well* (by Skolem's Theorem [Skolem 1934]).

Consequently, we can have a model for the foreground type, as in Figure 2, with one infinite list ending in *nil* (labeled by $\mathbb{N}$), and one list that is infinite in both directions (labeled by $\mathbb{Z}$). We can interpret $list(x)$ to be true for all nodes when defining $len(x)$ to be a nonstandard number.

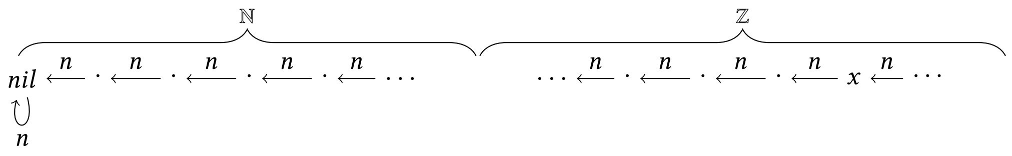

Fig. 2. A non-standard model for *list*

Despite the fact that Presburger arithmetic comes with (an infinite set of) induction axioms, the signature of these axioms do not refer to the foreground universe, and hence do not capture induction on properties of the foreground universe (in this case lists) that can be expressed using uninterpreted functions. Note that the above formulation also is in the safe fragment, as functions are one-way.

Also, note that in the above model with n interpreted as shown, we can also interpret $list(x)$ to be *false* (by interpreting *list* to be true only on the $\mathbb{N}$ prefix).

It turns out that we may hence be unable to prove certain theorems that we expect to be true of lists. For instance, if we defined list-segments in a similar fashion as above, then the formula

$$(lseg(x, y) \land list(y)) \longrightarrow list(x)$$

is not valid as there are models that have a non-standard background model for arithmetic where the negation of the formula holds (for example, there is model that looks like the model above, has *list(x)* to be false, and $lseg(x, y)$ maps to true if y is in the initial $\mathbb{N}$ prefix).

In fact, in the work of natural proofs reported in [Pek et al. 2014; Qiu et al. 2013], the above is *not provable* using term instantiation, and the authors resort to adding user-provided axioms that capture facts such as the above. Our results show that the reason term instantiation cannot prove the above is that it is *not valid* in the FO-semantics where recursive definitions are interpreted as fixpoints. In Section 6, we go into this problem in more detail, exploring how we can automatically add induction principles that can prove such properties correct, alleviating the user from having to provide them (these automatic ways of adding induction principles, however, will not be complete; in fact, no automatic complete scheme can exist, as we explain in Section 6).

5.5 Applications to the Array Property Fragment

The goal of this section is to (a) show that the known decidable theory of arrays for the logic *array property fragment* (APF) [Bradley et al. 2006] can be translated into an equisatisfiable formula in the UCT setting such that the term instantiation given in this paper is a decision procedure, and (b) show certain errors in existing proofs for extensions of APF stemming from the fact that they do not handle non-standard models carefully.

The array property fragment is a restricted fragment of the standard theory of arrays; we do not define them here, but refer the reader to the literature [Bradley 2007; Bradley et al. 2006]. We work with *integer-indexed* array property fragment, where contents are also integers.

There is a standard rewriting of array-writes, as given in [Bradley et al. 2006], that transforms the formula so that there are no more writes. Subsequently, APF really resembles a logic that combines uninterpreted functions from integers to integers (modeling integer-indexed arrays).

However, we can embed this in a UCT framework using the following representation, where we have a single background theory of integers with addition. Let us have a foreground universe of *abstract indices AI*, endowed with a linear order $\leq_a$ and functions that map abstract indices to concrete indices, the latter being integers. Let us denote this map as $r()$. We further introduce a constant a_c over the abstract indices for each constant c that is mentioned in the formula.

Then we have the following requirements:

$$r(a_c) = c, \textit{ for every constant } c \textit{ mentioned in the formula}$$

$$\forall x, y : \sigma_1.\ x \leq_a y \leftrightarrow r(x) \leq r(y)$$

and the axioms for $\leq_a$ being a linear order on the foreground universe.

Now let us transform the APF formula φ (without writes) by replacing all quantification over indices to quantification over the foreground type of abstract indices, and replace every occurrence of such a quantified index i by $r(i)$. Since this framework has only one-way functions (as r maps foreground to background), this falls into the safe fragment of the UCT, and hence by our results, term instantiation is complete. Furthermore, since *all* functions from the foreground theory map to the background theory, term instantiation terminates in the first round as described in Proposition 4.9. Consequently, term instantiation here is a *decision procedure.*

The formula transformed to UCT framework above is different from the original formula as universal quantification is restricted to the indices corresponding to the abstract indices, which can always be realized as a finite model, where we have one element at most for each term instantiated to the foreground type. However, as the proof of decidability of APF [Bradley et al. 2006] shows, term instantiation is complete for APF. Consequently, the original formula in APF and the transformed formula in the UCT fragment are equisatisfiable. We emphasize that we are not claiming a new proof of the APF fragment, but just that it can be mapped to a UCT framework where our term instantiation is precisely the same done in the APF decision procedure.

5.5.1 Negative Results for Extensions of APF. Several extensions of APF have been proved undecidable in [Bradley et al. 2006]; for instance, having an additional inner existential quantifier, allowing + on universally quantified indices, allowing nested reads, etc. These results and more have been proved again in the thesis [Bradley 2007] of one of the authors of the paper [Bradley et al. 2006].

However, it turns out that the *proofs* given in this thesis [Bradley 2007] are incorrect. The proofs reduce finding whether Diophantine equations have a solution to satisfiability of the various logics; however, all these proofs assume that the models for the formula will be standard models of arithmetic and fail for non-standard models.[1] In fact, the structure of the proofs themselves seem wrong— checking whether Diophantine equations are satisfiable (which is clearly a problem in r.e.) is reduced to checking satisfiability of the logic, which should be impossible. For if it were possible, it would mean validity is co-r.e. hard, while we know that validity is in r.e. as these are all theories with a recursive axiomatization.

Note, however, that the APF *decidability* result does indeed work for indices and values satisfying Presburger axioms of arithmetic (not just the standard theory of natural numbers).

One important additional result in the thesis [Bradley 2007] (that is not in [Bradley et al. 2006]) is that APF with $<$ (allowing $<$ and not just $\leq$ to compare indices) is undecidable, and this proof suffers from the same problem. We are unable to confirm the decidability/undecidability status of this logic, and declare this problem to be open:

Open Problem: Does the array property fragment with $<$ allowed in addition to $\leq$ (or equivalently, with indices allowing *disequality* of indices) and contents of arrays being axiomatized by Presburger arithmetic, admit a decidable satisfiability problem?

6 INDUCTION PRINCIPLES

In Sections 3 and 4 we have presented a sound and complete term instantiation method for proving validity (or unsatisfiability) of safe formulas that only quantify over the foreground theory. Section 5.1 links this method to the proof technique of natural proofs for recursive data structures from [Pek et al. 2014; Qiu et al. 2013]. However, since these natural proofs work within the FO-semantics instead of the LFP-semantics (this corresponds to Phase 1 mentioned in Section 5.1) , they cannot prove properties that are valid in the LFP-semantics but not in the FO-semantics. Such an example is explained for list segments in Section 5.1. This is compensated by adding user specified axioms for the recursively defined data structures (which are known to hold in the LFP-semantics).

Often, these properties that hold in the LFP-semantics but not the FO-semantics require some form of an inductive proof. Let us illustrate this on an example. We use recursive transitive closure definitions for illustration purposes in this section in order to keep the formulas simple.

Example 6.1. We modify the definition of reachability from Example 2.1 to a definition of sorted reachability. The following formula refers to $d(x)$ as the data at node x (a natural number, for example), and compares the data values using $\leq$.

$$\varphi_{srch} := \forall x, y. srch(x, y) \leftrightarrow_{\text{LFP}} \underbrace{x = y \lor (x \neq y \land srch(f(x), y)) \land d(x) \leq d(f(x))}_{\rho_{srch}(x, y, srch)}$$

[1]Note to reviewers: We did contact Aaron Bradley, and while he did not confirm that these proofs are incorrect, he said that the flaws we outlined seem entirely plausible, given that the thesis was not peer-reviewed.

We want to prove validity in LFP-semantics of the following formula:

$$\varphi_{rch} \wedge \varphi_{srch} \rightarrow \underbrace{(\forall x, y. srch(x, y) \rightarrow rch(x, y))}_{\varphi}.$$

In the pure FO-semantics, the formula is not valid because of infinite models that do not interpret *srch* as the least fixpoint of its definition. For example, take a model consisting of two infinite f-chains with $d(u) = 0$ for all elements. Then interpret *rch* as the transitive and reflexive closure of f, and *srch* as the transitive and reflexive closure of f plus $srch(u, v)$ for u from the first f-chain and v from the second f-chain. $\dashv$

One can prove validity of the formula $\varphi_{rch} \wedge \varphi_{srch} \rightarrow \varphi$ from Example 6.1 in LFP-semantics by induction. This induction could proceed by using a function that defines the distance of two nodes that are related by *srch*. A more direct approach would proceed by induction directly along the recursive definition of *srch*. The aim of this section is to formalize such inductive proofs along the recursive definitions by first-order definable induction principles.

First-order induction principles are a common way of formalizing inductive proofs. For example, it is well known that Presburger arithmetic (the theory of natural numbers with addition) can be axiomatized by a schema of first-order induction principles for the natural numbers. An axiomatization of the theory of finite trees based on first-order induction schemas is presented in [Backofen et al. 1995]. Induction principles specifically tailored for transitive closure definitions are considered in [Lev-Ami et al. 2009] in the context of first-order theorem proving. We propose in this section a general form of induction principles for recursively defined relations, which subsume all of these induction schemas. Our experiments (presented in Section 6.3) show that properties about recursively defined data structure that require additional axioms in the natural proof context can be proven based on our induction principles.

Note that we cannot hope for a complete set of induction axioms from which we can prove all valid properties because recursive definitions are too powerful for their theories to be recursively axiomatizable. For example, we can define transitive closures with recursive definitions (see Example 2.1) and in [Lev-Ami et al. 2009, Proposition 4.1] it is shown that there is no recursively enumerable axiomatization of first-order logic with transitive closure. One can also easily see that it is possible to define the structure $(\mathbb{N}, +, \cdot, 0, 1)$ of full arithmetic by a recursive predicate $N(x) \leftrightarrow_{\text{LFP}} x = 0 \vee \exists y.\, x = y + 1 \wedge N(y)$ in combination with a formula $\forall x. N(x)$ and the standard axioms for addition and multiplication. So a complete set of induction axioms for recursive definitions would provide a recursive axiomatization of $(\mathbb{N}, +, \cdot, 0, 1)$, which cannot exist by Gödel's incompleteness theorem.

Of course, a lot of research has been done on induction in the context of automated theorem proving. The aim of this section is not to compare to existing methods for automated inductive proofs, but rather to illustrate that there is a simple way of incorporating inductive reasoning in the context of recursive first-order definitions.

6.1 First-Order Induction Principles for Recursive Definitions

We refer to the notation for recursive definitions from Section 2.3. Assume that we want to prove a property of the form

$$\varphi_P := \forall \overline{x}.\, R(\overline{x}) \rightarrow \psi(\overline{x}) \text{ with recursive definition } \varphi_R = \forall \overline{x}.\, R(\overline{x}) \leftrightarrow_{\text{LFP}} \rho(\overline{x}, R).$$

So we are interested in the validity of the formula $\varphi_R \rightarrow \varphi_P$ (in the LFP-semantics). The following induction principle expresses the fact that if $\psi(\overline{x})$ defines a pre-fixpoint of μ_ρ, then $[\![R]\!]_M$ is included in $[\![\psi]\!]_M$ for every model M:

$$\mathrm{IP}(R,\psi) := \mathrm{PFP}(\rho,\psi) \rightarrow (\forall \overline{x}.R(\overline{x}) \rightarrow \psi(\overline{x})) \quad \text{with} \quad \mathrm{PFP}(\rho,\psi) := \forall \overline{x}.\rho(\overline{x}, R \leftarrow \psi) \rightarrow \psi(\overline{x}).$$

The following lemma states that the formula $\mathrm{PFP}(\rho,\psi)$ indeed expresses that $\psi(\overline{x})$ defines a pre-fixpoint of μ_ρ.

Lemma 6.2. *For each model M we have that $M \models_{LFP} \mathrm{PFP}(\rho,\psi)$ if, and only if, $[\![\psi(\overline{x})]\!]_M$ is a pre-fixpoint of μ_ρ on M.*

Proof. From the definition of μ_ρ we get that $\mu_\rho([\![\psi(\overline{x})]\!]_M) = [\![\rho(\overline{x}, R \leftarrow \psi)]\!]_M$. From the definition of $\mathrm{PFP}(\rho,\psi)$ we obtain that $M \models_{\mathsf{LFP}} \mathrm{PFP}(\rho,\psi)$ if, and only if, $[\![\rho(\overline{x}, R \leftarrow \psi)]\!]_M \subseteq [\![\psi(\overline{x})]\!]_M$. Hence, $M \models_{\mathsf{LFP}} \mathrm{PFP}(\rho,\psi)$ if, and only if, $[\![\psi(\overline{x})]\!]_M$ is a pre-fixpoint of μ_ρ. □

We can use Lemma 6.2 to show that our induction principles are satisfied in all models that interpret R as the least fixpoint of its definition, as stated in the following lemma.

Lemma 6.3. *For each model M, if $M \models_{LFP} \varphi_R$ then $M \models_{FO} \mathrm{IP}(R,\psi)$.*

Proof. Assume that $M \models_{\mathsf{LFP}} \varphi_R$ and $M \models_{\mathsf{FO}} \mathrm{PFP}(\rho,\psi)$ (otherwise the statement is trivially true). By Lemma 6.2, this means that $[\![\psi(\overline{x})]\!]_M$ is a pre-fixpoint of μ_ρ. Since $[\![R]\!]_M$ is the least fixpoint of μ_ρ on M, we obtain that $[\![R]\!]_M \subseteq [\![\psi]\!]_M$ and hence $M \models_{\mathsf{FO}} (\forall \overline{x}.R(\overline{x}) \rightarrow \psi(\overline{x}))$. This means that $M \models_{\mathsf{FO}} \mathrm{IP}(R,\psi)$. □

The following proposition states that validity of formulas in LFP-semantics can be shown by strengthening the formula by induction principles and showing validity in FO semantics.

Proposition 6.4. *Let $\varphi = \varphi_A \rightarrow \varphi_P$ be some FO formula with recursive definitions, including the recursive definition φ_R of R, and let ψ be an FO formula. If $(\varphi_A \wedge IP(R,\psi)) \rightarrow \varphi_P$ is valid in FO-semantics, then $\varphi_A \rightarrow \varphi_P$ is valid in LFP-semantics.*

Proof. Assume that $\varphi_A \rightarrow \varphi_P$ is not valid in LFP-semantics. Then there is a model M with $M \models_{\mathsf{LFP}} \varphi_A \wedge \neg\varphi_P$. Since φ_A includes the recursive definition of R, we obtain $M \models_{\mathsf{LFP}} \varphi_R$. By Lemma 6.3, $M \models_{\mathsf{FO}} \mathrm{IP}(R,\psi)$. Combining the formulas and using Remark 1, we get $M \models_{\mathsf{FO}} \varphi_A \wedge \neg\varphi_P \wedge \mathrm{IP}(R,\psi)$, which means that $(\varphi \wedge IP(R,\psi)) \rightarrow \varphi'$ is not valid in FO-semantics. □

We illustrate the use of Proposition 6.4 by continuing Example 6.1.

Example 6.5. Consider the formula that states that *rch* is a pre-fixpoint of the definition of *srch*:

$$\mathrm{PFP}(\rho_{srch}, rch(x,y)) = \forall x,y.(x = y \vee (x \neq y \wedge rch(f(x),y)) \wedge d(x) \leq d(f(x))) \rightarrow rch(x,y)$$

It is not difficult to verify that $\varphi_{rch} \wedge \varphi_{srch} \rightarrow \mathrm{PFP}(\rho_{srch}, rch(x,y))$ is a valid formula in the FO-semantics. We obtain that

$$\varphi_{rch} \wedge \varphi_{srch} \wedge \mathrm{IP}(srch, rch) \rightarrow (\forall x,y.srch(x,y) \rightarrow rch(x,y))$$

is valid in the FO-semantics, which can be shown by using term instantiation as in Section 4 (on the negation of the formula). By Proposition 6.4 this also yields a proof that the original formula $\varphi_{rch} \wedge \varphi_{srch} \rightarrow (\forall x,y.srch(x,y) \rightarrow rch(x,y))$ is valid in the LFP-semantics. ⊣

A slightly stronger version of the induction principle is obtained when substituting R in ρ by the conjunction $R \wedge \psi$ instead of just ψ:

$$\mathrm{IP}_s(R,\psi) := \mathrm{PFP}(\rho, \psi \wedge R) \rightarrow (\forall \overline{x}.R(\overline{x}) \rightarrow \psi(\overline{x}))$$

This induction principle is stronger because if ψ is a pre-fixpoint of μ_ρ, then $R \wedge \psi$ is also a pre-fixpoint of μ_ρ, as formally stated in the lemma below.

Lemma 6.6. *For each model M, if $M \models_{FO} \varphi_R \wedge \mathrm{PFP}(\rho, \psi)$ then $M \models_{FO} \mathrm{PFP}(\rho, \psi \wedge R)$.*

Proof. If $M \models_{\mathrm{FO}} \varphi_R \wedge \mathrm{PFP}(\rho, \psi)$, then $[\![R]\!]_M$ is a fixpoint of μ_ρ and by Lemma 6.2 $[\![\psi]\!]_M$ is a pre-fixpoint of μ_ρ. Then the intersection $[\![\psi]\!]_M \cap [\![R]\!]_M = [\![R \wedge \psi]\!]_M$ is also a pre-fixpoint of μ_ρ, and hence $M \models_{\mathrm{FO}} \mathrm{PFP}(\rho, \psi \wedge R)$. □

This lemma implies that we can replace the induction principle $\mathrm{IP}(R, \psi)$ in Proposition 6.4 by the strong induction principle $\mathrm{IP}_s(R, \psi)$.

Example 6.7. The following formula expresses that the set of points reachable from x is linearly ordered: $\forall x, y, z.\ (rch(x, y) \wedge rch(x, z)) \to (rch(y, z) \vee rch(z, y))$.

This formula is valid in the LFP-semantics of rch but not in the FO-semantics. It turns out that the weak induction principle is not sufficient for proving the formula valid but validity follows from the strong induction principle. Details are given in the supplementary material.

6.2 Deriving New Induction Principles

We now present an example of a formula for which one induction principle is not sufficient to prove validity in the FO-semantics. We need a second induction principle that is obtained from the formula of the first induction principle.

Consider the definition of reachability from Example 2.1 and the following equivalent definition that recurses on the other end of the path ($rchr$ for reachability reversed).

$$\varphi_{rchr} := \forall x, y. rchr(x, y) \quad \leftrightarrow_{\mathrm{LFP}} \quad \underbrace{x = y \vee (x \neq y \wedge \exists v. f(v) = y \wedge rchr(x, v))}_{\rho_{rchr}}$$

In the least fixpoint interpretation, both definitions correspond to the transitive and reflexive closure of the function f. Consider the formula $\varphi := \forall x, y. rch(x, y) \to rchr(x, y)$. Then $\varphi_{rch} \wedge \varphi_{rchr} \to \varphi$ is certainly valid in LFP-semantics but not in the FO-semantics (using a model similar to the one described in Example 6.1). Letting $\psi(x, y) := rch(x, y) \to rchr(x, y)$, the induction principle yields

$$\mathrm{IP}(rch, rchr) = (\forall x, y.(x = y \vee (x \neq y \wedge rchr(f(x), y))) \to rchr(x, y)) \to (\forall x, y.\psi(x, y)).$$

We use the weak induction principle for this example in order to keep the formulas shorter. The reasoning would not change for the strong induction principle. It turns out that this induction principle does not suffice, that is, $\varphi_{rch} \wedge \varphi_{rchr} \wedge \mathrm{IP}(rch, rchr) \to \varphi$ is not valid in the FO-semantics (see supplementary material). However, we can view this first induction principle as providing a new proof obligation, namely proving $\mathrm{PFP}(rch, rchr)$, which then implies φ.

We explain how we can derive a second induction principle from the formula $\mathrm{PFP}(rch, rchr)$ that states that $rchr$ is a pre-fixpoint of ρ_{rch}:

$$\mathrm{PFP}(rch, rchr) = \forall x, y.(x = y \vee (x \neq y \wedge rchr(f(x), y))) \to rchr(x, y)$$

This formula can be rewritten as

$$(\forall x, y.\ x = y \to rchr(x, y)) \wedge (\forall x, y.(x \neq y \wedge rchr(f(x), y)) \to rchr(x, y))$$

The second conjunct is equivalent to

$$\forall x, y.(rchr(f(x), y) \to (x \neq y \to rchr(x, y)))$$

Introducing a variable z for the term $f(x)$, we obtain

$$\forall z, y. rchr(z, y) \to \underbrace{(\forall x.(z = f(x) \wedge x \neq y) \to rchr(x, y))}_{\theta(z, y)}$$

which is of the form required for the induction principle $\mathrm{IP}(rchr(z,y), \theta(z,y))$:

$$\forall z, y.((z = y \lor (z \neq y \land \exists v.\ f(v) = y \land \theta(z,v))) \rightarrow \theta(z,y)) \rightarrow \forall z, y.(rchr(z,y) \rightarrow \theta(z,y))$$

One can check that this second induction principle together with the first one is sufficient to prove the original formula correct. This is confirmed by our experiments where the property for two corresponding definitions of list segments is verified by these two induction principles.

Note that the second induction principle was obtained by simply rewriting the formula $\mathrm{PFP}(rch, rchr)$ into an implication with *rchr* on the left-hand side, a process that can be automated.

6.3 Evaluation

To demonstrate the efficacy of these induction principles, we manually encoded some examples in Z3 [de Moura and Bjørner 2008] most existing systems cannot prove correct, due to the existence of non-standard models discussed in Section 6.

We borrowed many of our examples from [Chu et al. 2015]. Our analysis is not exhaustive, but all lemmas we considered were verified in the LFP semantics using the induction principles. The examples cover traditional data structures encountered in program verification, such as list segments, heaplets, etc. We discuss some of the examples below, please see [Löding et al. 2017] for the full Z3 programs.

In these lemmas, *list* refers to a linked list, and $list_1$ is a linked list with an explicit parameter for the length of the list. The data structures *dlist* and *slist* are similar, but refer to doubly linked lists and sorted lists respectively. The doubly linked lists and sorted lists contain a function *key* on nodes, while the singly linked list does not. The last lemma refers to *h_lseg*, the heaplet segment between two nodes x and y. If y is *nil*, this is the heaplet for the list segment from x. This lemma is proved and then used as an axiom in many of our other lemmas.

$lseg_v$ is a unary definition of a list segment parametrized by a constant v, while *lseg* and *lsegr* are binary definitions for list segments, defined both with the moving pointer moving forwards and backwards (similar to the two definitions *rch* and *rchr* of reachability from Section 6). We similarly define *dlseg* and *slseg* for doubly linked list segments and sorted list segments.

The relation $rev(x,y)$ holds if the reverse of a list beginning at node x is y, and $keys(\cdot)$ represents all the keys reachable from a node. Finally, *bst* and *btree* are data structures for binary search trees and binary trees respectively, and $leftmost(x,v)$ asserts that the leftmost non-nil node reachable from node x contains the key v.

The lemmas in Table 1 include examples similar to those discussed in the previous subsection. Specifically, the fourth lemma proved represents an example similar to Example 6.7 where the stronger induction principle is needed. In most of the other programs, we also use the stronger induction principle as it is implied by the weaker induction principle (see Lemma 6.6). The encoding of the lemmas and the necessary induction principles was done by hand, though an important area for future work is automating this process.

The fifth lemma demonstrates the implication of one list segment definition to another, similar to Example 6.5. Two induction principles are needed for this lemma, in contrast with only one induction principle for all other examples. The major difference between the examples for the reachability definitions in Sections 6.1 and 6.2 and the lemmas in Table 1 is that these lemmas also refer to semantics of separation using an uninterpreted *h_lseg* function that is also used in the unfolding process.

In all examples above, the maximum depth of unfolding required for any lemma was 2, and Z3 took less than 0.2 seconds cumulatively to prove all these lemmas. Though the completeness result

Table 1. Examples of lemmas proved correct in LFP semantics using the induction principles covered in Section 6. The Defs Unfolded column refers to the number of definitions needed to be unfolded for the proof to go through. Depth represents the maximum level of unfolding needed to witness the unsatisfiability. None of the lemmas above are provable without the induction principles.

	Lemma	Defs Unfolded	Depth
1	$\forall x, l.\ list_1(x,l) \longrightarrow list(x)$	1	1
2	$\forall x.\ lseg_v(x) \wedge list(v) \longrightarrow list(x)$	4	2
3	$\forall x,y.\ lseg(x,y) \wedge list(y) \longrightarrow list(x)$	4	2
4	$\forall x,y.\ lseg(x,y) \longrightarrow (\exists z.\ lseg(x,z) \longrightarrow lseg(y,z) \vee lseg(z,y))$	1	1
5	$\forall x,y.\ lsegr(x,y) \longrightarrow lseg(x,y)$	8	2
6	$\forall x.\ dlist(x) \longrightarrow list(x)$	3	1
7	$\forall x,l\ dlist_1(x,l) \longrightarrow dlist(x)$	1	1
8	$\forall x,y.\ dlseg(x,y) \wedge dlist(y) \longrightarrow dlist(x)$	6	2
9	$\forall x,y.\ dlist(x) \wedge dlist(y) \wedge rev(x,y) \longrightarrow keys(x) = keys(y)$	0	0
10	$\forall x,l\ slist_1(x,l) \longrightarrow slist(x)$	1	1
11	$\forall x.\ slist(x) \longrightarrow list(x)$	3	1
12	$\forall x,y.\ slseg(x,y) \wedge slist(y) \longrightarrow slist(x)$	5	2
13	$\forall x.\ bst(x) \longrightarrow btree(x)$	1	1
14	$\forall x.\ bst(x) \wedge leftmost(x,v) \longrightarrow v \leq key(x)$	0	0
15	$\forall x,y,z.\ H = h\_lseg(x,y) \wedge z \in H \longrightarrow n(z) \in H$	2	1

requires a potential infinite depth, these examples show that in practice we are able to witness the unsatisfiability rather quickly.

Besides these lemmas, we can also use our technique for proving functional programs correct. For example, consider the McCarthy 91 function [Manna and McCarthy 1970]. Using our induction principles and unfolding technique, we have no trouble verifying interesting properties of this function. The function evaluates $M(n)$ to $n - 10$ if $n > 100$, and to the recursive $M(M(n+11))$ otherwise. The interesting claim regarding this function is that $M(n) = 91$ for all $n \leq 100$. By encoding M as a relation $M_R(n,m)$ such that $M(n) = m$ iff $M_R(n,m)$ holds, the claim is in the correct form for using our induction principle. The induction principle, together with unfolding the recursive definition just twice (each at depth 1) proves the claim. This example is also included in [Löding et al. 2017].

7 RELATED WORK

Gilmore [Gilmore 1960] proposed one of the first sound and complete procedures for first-order logic, implementing it on the IBM 704 Data Processing Machine in 1960; this was refutation-based and relied on Herbrand's theorems [Ebbinghaus et al. 1994; Schöning 2008] on reducing satisfiability of quantified FOL reasoning to satisfiability of larger and larger classes of propositional formulas. In a sense, our work is in a similar vein— we reduce FOL reasoning to sequences of formulas that are in decidable logics. Given the efficient implementation of Boolean reasoning, quantifier-free formulas over a combination of a variety of theories, and some quantified theories, our goal is to reduce reasoning to these logics rather than just propositional logic. Davis and Putnam [Davis and Putnam 1960], also in 1960, proposed a more efficient algorithm for FOL validity based on ground instantiations and improved the reasoning of propositional logic using propositional resolution (later refinements of the latter led to the DPLL technique used in modern SAT solver algorithms). Robinson [Robinson 1965] lifted resolution from ground instances using unification of clauses, and this led to the superposition calculus for equational logic which many FO theorem provers use today (Vampire [Kovács and Voronkov 2013], SPASS [Weidenbach et al. 2009], E [Schulz 2002]). Apart from these, there is, of course, a vast literature on interactive theorem provers [Harrison et al.

2014; Kaufmann and Moore 1997; Kaufmann et al. 2000; The Coq development team 2016; Norell 2009; Owre et al. 1992; Paulson 1993].

Quantified logics for certain important theories are known to be decidable as well, in particular Presburger arithmetic (natural numbers/integers with addition), rationals, and reals, using quantifier elimination techniques [Bradley and Manna 2007].

The development of quantified reasoning techniques over SMT solvers have typically been based on quantifier instantiation (see [Reynolds 2016] for a recent survey of these methods). The `Simplify` theorem prover [Detlefs et al. 2005; Nelson 1980] introduced the E-matching technique, which chooses instantiations based on matching pattern terms. A number of similar methods have since emerged and these techniques are now implemented in several SMT solvers [Barrett et al. 2011; Barrett and Tinelli 2007; de Moura and Bjørner 2008; Detlefs et al. 2005; Rümmer 2012]. Combining term instantiations with background SMT solvers have also been investigated in [Ganzinger and Korovin 2003, 2006]

One work that is close to ours is iProverEq [Korovin and Sticksel 2010], which is an instantiation-based theorem prover based on the Inst-Gen method first mentioned in [Ganzinger and Korovin 2003] that is complete for first-order logic with equality. The Inst-Gen method differs from our method in that if a set of ground terms is found to be satisfiable, the model is used to guide the instantiation process. Our method only uses new ground terms that arise in the formulas to guide the instantiation. Additionally, iProverEq only covers pure first-order logic with equality, and does consider background theories.

Another similar related work is that of Ge and de Moura [Ge and de Moura 2009], where an instantiation method for combinations of uninterpreted functions and relations with background theories is presented. However, the combinations of theories in [Ge and de Moura 2009] is not over many-sorted logic but uses one sort for all of the theories. Term instantiation is shown to be complete for formulas that are called essentially uninterpreted [Ge and de Moura 2009] which means that only uninterpreted functions or relations are applied directly to the quantified variables. This class of formulas differs from the class of safe formulas for which we show our instantiation procedure to be complete. In particular, equality is an interpreted relation in [Ge and de Moura 2009] and thus variables cannot even be compared for equality. For example, defining injectivity of a function f by $\forall x, y.\ x \neq y \rightarrow f(x) \neq f(y)$ is not an essentially uninterpreted formula (but falls into our safe fragment). Furthermore, most verification conditions arising from programs do not fall into the essentially uninterpreted fragment as properties of scalar variables that are over background sorts are typically required. Hence, the result from [Ge and de Moura 2009] is similar in spirit to ours but the classes of formulas for which the instantiation is complete are different. The model-based quantifier instantiation introduced in [Ge and de Moura 2009] has been implemented in Z3 and has also been shown to be complete when, through external arguments, one can ensure that the formula will generate only a finite number of instantiations before a real model is found [Ihlemann et al. 2008].

Turning to reasoning with recursive definitions, logics with recursive functions have been studied for a long time. The NQTHM prover developed by Boyer and Moore [Boyer and Moore 1980], and its successor ACL2 [Kaufmann and Moore 1997; Kaufmann et al. 2000] had a Lisp-like language that had support for recursive functions and had several induction heuristics to find inductive proofs. Our treatment of induction is similar in spirit but is in the context of a UCT with background theory solvers. Recent work on cyclic proofs [Brotherston et al. 2011; Ta et al. 2016] use heuristics for reasoning about recursive definitions. However, heuristics used in these works are incomparable to our methods. Specifically, [Ta et al. 2016] reasons about recursive definitions by induction on the size of a model, and [Brotherston et al. 2011] adds certain inference rules to the proof system and global conditions on cyclic proofs. In contrast, our heuristic is based on adding FO formulas capturing parts of the least fixpoint semantics of the recursive definition, and is independent of the actual proof system.

The safe fragment and notion of foreground and background theories in our work is similar to the notion of stratified vocabularies present in [Abadi et al. 2007; Padon et al. 2017]. However, our safe fragment notion is a looser restriction than stratification. Safety of formulas ensures that all elements from the foreground theory arise as a result of function application, and thus functions from the foreground universe to the foreground universe are permitted. Stratification does not even allow functions from the foreground universe to the foreground universe, as the main purpose of stratification is to keep the number of instantiated terms finite, which leads to decidability results. In fact, Proposition 4.9 should hold with many-sorted stratified theories.

There are also several *decidable logics*, especially for separation logic, on recursive data-structures, some of which use SMT solvers [Madhusudan et al. 2011; Madhusudan and Qiu 2011; Piskac et al. 2013, 2014]; this is an active area of research and we refer the reader to the recent paper [Le et al. 2017] and references therein. These logics are naturally restricted in expressive power to obtain a decidable validity problem. In contrast to interactive theorem proving and procedures for decidable logics, there is a third class of techniques and tools that allow expressive and undecidable logics, but allow automated sound verification. In a general software verification context with specifications and complex manipulation of ghost structures, techniques in use commonly unfold recursive definitions (using either unfold/fold techniques or just unfolding followed by abstracted reasoning); see for instance work of this kind in Dafny [Leino 2012] and Verifast [Jacobs and Piessens 2008], where the latter accepts such fold/unfold tactic suggestions from the user.

The work of Suter et al [Suter et al. 2010] introduced the notion of verifying *abstractions* of programs manipulating purely functional data-structures, where the procedure did unfolding followed by abstracting recursive functions as uninterpreted functions in order to reduce them to decidable logics, and conditions under which these procedures are complete was investigated. The work on natural proofs [Madhusudan et al. 2012; Pek et al. 2014; Qiu et al. 2013] advocates this as a principle for showing imperative programs manipulating pointer data-structures to be correct, showing its efficacy for a large class of programs against separation logic specifications. The focus on this paper has been more particularly on natural proofs because of its systematic mapping of data-structure properties into first-order logic with recursion. The work presented here however does indeed give foundations to the unfold-and-match method in general, followed in the various works above.

The drawback in the natural proof work that the user needs to provide (unproven) axioms in order to prove properties correct has also been addressed by the work of Nguyen et al [Nguyen and Chin 2008] and Chu et al [Chu et al. 2015]. In the latter, the authors, working in a CLP framework, show how inductive hypotheses can be generated using proof obligations that arise during verification. Incorporating these ideas in addition to our induction principles is an interesting future direction.

In recent work [Feldman et al. 2017], quantifier instantiation is used for verifying inductive invariants for programs. While this technique is related to our work, it does not introduce general formulas expressing the induction principle, but rather uses term instantiation to generate a sequence of quantifier-free formulas that in the limit expresses that the property is not an invariant. The advantage of our induction principles is that they can be used to derive new proof obligations (see Section 6.2).

Induction principles specifically tailored for transitive closure are considered in [Lev-Ami et al. 2009] in a context of first-order theorem proving. If we formalize transitive closure using recursive definitions, then our induction principles imply the ones proposed in [Lev-Ami et al. 2009].

8 CONCLUSIONS AND FUTURE WORK

Reasoning with quantified FOL with recursive definitions has mostly been heuristic in nature. We have, however, provided results that characterize both the power of these heuristics as well as their limitations. In particular, when dealing with quantified logics in a many-sorted uninterpreted combination of background theories, term instantiation heuristics are not in general complete, but are indeed complete for the safe fragment of FOL. The efficacy of practical tools using such heuristics can be reconciled by observing that they reason with safe formulas. When the logic is enriched with recursive definitions with least fixpoint semantics, no technique can be complete, but heuristics for discovering and adding induction principles during term instantiation can be significantly helpful. We believe that our results give a more compelling foundation to support the ideas emerging from practice that recursive definitions be handled using discovery of inductive principles, while quantified logics be handled using term instantiation.

The most interesting future direction is an implementation of tools for the safe fragment, with and without recursive definitions (with LFP-semantics). An efficient realization of our algorithm by incorporating SMT solvers for the safe fragment that works using term instantiation would provide useful, robust and more predictable quantified reasoning for verification engines than what is available today. Designing and implementing program logics that are guaranteed to produce safe formulas would extend this robustness to program verifiers.

On the theoretical front, it would be interesting to see whether the safe fragment can be extended while providing completeness guarantees. Furthermore, it would be interesting to calibrate the effectiveness of the induction principles given in this paper (or some expansion of them) by identifying settings where they can be proved to be complete. The discovery of inductive hypotheses that facilitate recursive reasoning is very similar to invariant synthesis in current literature (like interpolation [McMillan 2003] and abduction [Dillig et al. 2013]), and incorporating ideas developed for the latter to facilitate robust discovery of inductive hypotheses is an interesting direction to pursue.

ACKNOWLEDGEMENTS

We would like to thank Rupak Majumdar for discussions on complete logics for program verification. We would also like to thank Mahesh Viswanathan and Xiaokang Qiu for early discussions on automatic discovery of induction principles for natural proofs.

This material is based upon work supported by the National Science Foundation under Grant No. 1527395.

REFERENCES

Aharon Abadi, Alexander Rabinovich, and Mooly Sagiv. 2007. Decidable Fragments of Many-sorted Logic. In *LPAR'07*. 17–31. http://dl.acm.org/citation.cfm?id=1779419.1779423

Rolf Backofen, James Rogers, and K. Vijay-Shanker. 1995. A First-Order Axiomatization of the Theory of Finite Trees. *Journal of Logic, Language and Information* 4, 1 (1995), 5–39. https://doi.org/10.1007/BF01048403

Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *CAV'11*. 171–177. https://doi.org/10.1007/978-3-642-22110-1_14

Clark Barrett and Cesare Tinelli. 2007. CVC3. In *CAV'07*. 298–302. https://doi.org/10.1007/978-3-540-73368-3_34

Egon Börger, Erich Grädel, and Yuri Gurevich. 1997. *The Classical Decision Problem.* Springer.

Robert S. Boyer and J. Strother Moore. 1980. *A computational logic.* Academic Press.

Aaron Bradley. 2007. *Safety Analysis of Systems.* Ph.D. Dissertation. Stanford University.

Aaron R. Bradley and Zohar Manna. 2007. *The Calculus of Computation: Decision Procedures with Applications to Verification.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. 2006. What's Decidable About Arrays?. In *VMCAI'06*, Vol. 3855. Springer, 427–442. https://doi.org/10.1007/11609773

James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. 2011. Automated Cyclic Entailment Proofs in Separation Logic. In *CADE'11*. Springer-Verlag, 131–146. http://dl.acm.org/citation.cfm?id=2032266.2032278

Duc-Hiep Chu, Joxan Jaffar, and Minh-Thai Trinh. 2015. Automatic induction proofs of data-structures in imperative programs. In *PLDI'15*. 457–466.

Martin Davis and Hilary Putnam. 1960. A Computing Procedure for Quantification Theory. *J. ACM* 7, 3 (1960), 201–215. https://doi.org/10.1145/321033.321034

Leonardo de Moura and Nikolaj Bjørner. 2008. *Z3: An Efficient SMT Solver.* Springer Berlin Heidelberg, Berlin, Heidelberg, 337–340.

David Detlefs, Greg Nelson, and James B. Saxe. 2005. Simplify: A Theorem Prover for Program Checking. *J. ACM* 52, 3 (May 2005), 365–473. https://doi.org/10.1145/1066100.1066102

Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. 2013. Inductive invariant generation via abductive inference. In *OOPSLA'13*. 443–456. https://doi.org/10.1145/2509136.2509511

Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. 1994. *Mathematical logic (2. ed.)*. Springer.

Yotam M. Y. Feldman, Oded Padon, Neil Immerman, Mooly Sagiv, and Sharon Shoham. 2017. Bounded Quantifier Instantiation for Checking Inductive Invariants. In *TACAS 2017 (Lecture Notes in Computer Science)*, Vol. 10205. 76–95. https://doi.org/10.1007/978-3-662-54577-5

Harald Ganzinger and Konstantin Korovin. 2003. New Directions in Instantiation-Based Theorem Proving. In *LICS'03*. IEEE Computer Society, 55–64.

Harald Ganzinger and Konstantin Korovin. 2006. Theory Instantiation. In *LPAR'06*. 497–511. https://doi.org/10.1007/11916277_34

Yeting Ge and Leonardo Mendonça de Moura. 2009. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In *CAV'09 (Lecture Notes in Computer Science)*, Vol. 5643. Springer, 306–320. https://doi.org/10.1007/978-3-642-02658-4

Paul C. Gilmore. 1960. A Proof Method for Quantification Theory: Its Justification and Realization. *IBM Journal of Research and Development* 4, 1 (1960), 28–35. https://doi.org/10.1147/rd.41.0028

Andrzej Granas and James Dugundji. 2003. *Fixed Point Theory.* Springer.

John Harrison, Josef Urban, and Freek Wiedijk. 2014. History of Interactive Theorem Proving. In *Computational Logic*. 135–214. https://doi.org/10.1016/B978-0-444-51624-4.50004-6

Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. 2008. On Local Reasoning in Verification. In *TACAS'08/ETAPS'08*. 265–281. https://doi.org/10.1007/978-3-540-78800-3_19

Bart Jacobs and Frank Piessens. 2008. The VeriFast Program Verifier. (2008).

Matt Kaufmann and J. Strother Moore. 1997. An Industrial Strength Theorem Prover for a Logic Based on Common Lisp. *IEEE Trans. Software Eng.* 23, 4 (1997), 203–213. https://doi.org/10.1109/32.588534

Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. 2000. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, Norwell, MA, USA.

K. Korovin and C. Sticksel. 2010. iProver-Eq: An Instantiation-Based Theorem Prover with Equality. In *IJCAR'10 (Lecture Notes in Computer Science)*, J. Giesl and R. Hähnle (Eds.), Vol. 6173. Springer, 196–202.

Laura Kovács and Andrei Voronkov. 2013. First-Order Theorem Proving and Vampire. In *CAV'13*. 1–35. https://doi.org/10.1007/978-3-642-39799-8_1

Q. L. (Quang Loc) Le, M. Tatsuta, J. (Jun) Sun, and W. N. (Wei-Ngan) Chin. 2017. A Decidable Fragment in Separation Logic with Inductive Predicates and Arithmetic. In *CAV'17*. http://hdl.handle.net/10149/621090

K. Rustan M. Leino. 2012. Automating Induction with an SMT Solver. In *VMCAI'12*. 315–331. https://doi.org/10.1007/978-3-642-27940-9_21

Tal Lev-Ami, Neil Immerman, Thomas W. Reps, Mooly Sagiv, Siddharth Srivastava, and Greta Yorsh. 2009. Simulating reachability using first-order logic with applications to verification of linked data structures. *Logical Methods in Computer Science* 5, 2 (2009). https://doi.org/10.1007/978-3-642-39799-8_53

Christof Löding, Parthasarathy Madhusudan, and Lucas Peña. 2017. Foundations for Natural Proofs and Quantifier Instantiation: Artifact. https://github.com/lucaspena/foundations-np-qi-artifcat. (2017).

P. Madhusudan, Gennaro Parlato, and Xiaokang Qiu. 2011. Decidable logics combining heap structures and data. In *POPL'11*. 611–622. https://doi.org/10.1145/1926385.1926455

P. Madhusudan and Xiaokang Qiu. 2011. Efficient Decision Procedures for Heaps Using STRAND. In *SAS'11*. 43–59. https://doi.org/10.1007/978-3-642-23702-7_8

Parthasarathy Madhusudan, Xiaokang Qiu, and Andrei Stefanescu. 2012. Recursive Proofs for Inductive Tree Data-structures. In *POPL'12*. 123–136. https://doi.org/10.1145/2103656.2103673

Zohar Manna and John McCarthy. 1970. Properties of Programs and Partial Function Logic. *Machine Intelligence* 5 (1970), 79–98.

The Coq development team. 2016. The Coq Proof Assistant Reference Manual. (2016). Version 8.6.

Kenneth L. McMillan. 2003. Interpolation and SAT-Based Model Checking. In *CAV'03*. 1–13. https://doi.org/10.1007/978-3-540-45069-6_1

Yiannis N. Moschovakis. 2008. *Elementary induction on abstract structures*. Dover Publications.

Charles Gregory Nelson. 1980. *Techniques for Program Verification*. Ph.D. Dissertation. Stanford, CA, USA. AAI8011683.

Greg Nelson and Derek C. Oppen. 1979. Simplification by Cooperating Decision Procedures. *ACM Trans. Program. Lang. Syst.* 1, 2 (1979), 245–257. https://doi.org/10.1145/357073.357079

Huu Hai Nguyen and Wei-Ngan Chin. 2008. Enhancing Program Verification with Lemmas. In *CAV'08*. 355–369. https://doi.org/10.1007/978-3-540-70545-1_34

Ulf Norell. 2009. Dependently Typed Programming in Agda. In *TLDI'09*. 1–2. https://doi.org/10.1145/1481861.1481862

Peter W. O'Hearn. 2012. A Primer on Separation Logic (and Automatic Program Verification and Analysis). In *Software Safety and Security - Tools for Analysis and Verification*. Vol. 33. IOS Press, 286–318. https://doi.org/10.3233/978-1-61499-028-4-286

Derek C. Oppen. 1980. Complexity, Convexity and Combinations of Theories. *Theor. Comput. Sci.* 12 (1980), 291–302. https://doi.org/10.1016/0304-3975(80)90059-6

Sam Owre, John M. Rushby, and Natarajan Shankar. 1992. PVS: A Prototype Verification System. In *CADE'11*. 748–752. http://dl.acm.org/citation.cfm?id=648230.752639

Oden Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. 2017. Paxos Made EPR: Decidable Reasoning about Distributed Protocols. In *OOPSLA'17*. To Appear.

Lawrence C. Paulson. 1993. Isabelle: The Next 700 Theorem Provers. *CoRR* cs.LO/9301106 (1993). http://arxiv.org/abs/cs.LO/9301106

Edgar Pek, Xiaokang Qiu, and Parthasarathy Madhusudan. 2014. Natural proofs for data structure manipulation in C using separation logic. In *PLDI'14*. 440–451. http://dl.acm.org/citation.cfm?id=2594291

Ruzica Piskac, Thomas Wies, and Damien Zufferey. 2013. Automating Separation Logic Using SMT. In *CAV'13*. 773–789. https://doi.org/10.1007/978-3-642-39799-8_54

Ruzica Piskac, Thomas Wies, and Damien Zufferey. 2014. Automating Separation Logic with Trees and Data. In *CAV'14*. 711–728. https://doi.org/10.1007/978-3-319-08867-9_47

Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and Parthasarathy Madhusudan. 2013. Natural proofs for structure, data, and separation. In *PLDI'13*. ACM, 231–242.

Andrew Reynolds. 2016. Conflicts, Models and Heuristics for Quantifier Instantiation in SMT. In *Vampire@IJCAR 2016. Proceedings of the 3rd Vampire Workshop, Coimbra, Portugal, July 2, 2016.* 1–15.

John Alan Robinson. 1965. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM* 12, 1 (1965), 23–41. https://doi.org/10.1145/321250.321253

Philipp Rümmer. 2012. E-Matching with Free Variables. In *LPAR'12*. 359–374. https://doi.org/10.1007/978-3-642-28717-6_28

Uwe Schöning. 2008. *Logic for Computer Scientists*. Birkhäuser. https://doi.org/10.1007/978-0-8176-4763-6

Stephan Schulz. 2002. E - a brainiac theorem prover. *AI Commun.* 15, 2-3 (2002), 111–126. http://content.iospress.com/articles/ai-communications/aic260

Thoralf Skolem. 1934. Über die Nicht-charakterisierbarkeit der Zahlenreihe mittels endlich oder abzählbar unendlich vieler Aussagen mit ausschließlich Zahlenvariablen. *Fundamenta Mathematica* 1 (1934), 150–161.

Philippe Suter, Mirco Dotta, and Viktor Kuncak. 2010. Decision Procedures for Algebraic Data Types with Abstractions. In *POPL'10*. ACM, New York, NY, USA, 199–210. https://doi.org/10.1145/1706299.1706325

Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. 2016. Automated Mutual Explicit Induction Proof in Separation Logic. *CoRR* abs/1609.00919 (2016). http://arxiv.org/abs/1609.00919

Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. 2009. SPASS Version 3.5. In *CADE'09*. 140–145. https://doi.org/10.1007/978-3-642-02959-2_10