Towards Complete Reasoning about Axiomatic Specifications

Swen Jacobs and Viktor Kuncak

École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
firstname.lastname@epfl.ch

Abstract. To support verification of expressive properties of functional programs, we consider algebraic style specifications that may relate multiple user-defined functions, and compare multiple invocations of a function for different arguments. We present decision procedures for reasoning about such universally quantified properties of functional programs, using local theory extension methodology. We establish new classes of universally quantified formulas whose satisfiability can be checked in a complete way by finite quantifier instantiation. These classes include single-invocation axioms that generalize standard function contracts, but also certain many-invocation axioms, specifying that functions satisfy congruence, injectivity, or monotonicity with respect to abstraction functions, as well as conjunctions of some of these properties. These many-invocation axioms can specify correctness of abstract data type implementations as well as certain information-flow properties. We also present a decidability-preserving construction that enables the same function to be specified using different classes of decidable specifications on different partitions of its domain.

1 Introduction

A promising approach to precision and scalability in software verification is to let developers write the specifications of procedures, then use modern satisfiability-modulo theory solvers to automatically prove verification conditions that use such specifications. Recent results in modular verification have been obtained using e.g. tools VCC [2] and Jahob [12]. In this paper, we follow the idea of the high-level analysis of Hob [10] tool, and focus particularly on checking that specified functions are correctly used to implement the desired higher-level functionality. We simplify the problem by considering purely functional instead of imperative code. However, we consider more complex program properties than those checked by most previous automated verification approaches. Most of the existing approaches for specifying a function f use contracts that establish a relationship between a given input, x, and the output of the function, $f(x)$. Such contracts can be expressed as universally quantified statements $\forall x.\Phi(x, f(x))$, where Φ is a formula expressing the desired property of the function. Contracts are therefore a very special case among the classes of function specifications.

Algebraic specifications. In this paper, we consider a broader class of specifications that follow algebraic specification style. Our specifications may relate

R. Jhala and D. Schmidt (Eds.): VMCAI 2011, LNCS 6538, pp. 278–293, 2011.

multiple different functions, using, for example, abstraction functions to specify the behavior of an abstract data type implementation. Moreover, our properties may include *multiple universal quantifiers*, which allows us to express congruence, monotonicity, injectivity, and non-interference properties of functions. Such properties cannot be directly expressed using standard pre/post condition specifications, which refer to an arbitrary, but *only one* function invocation.

Sound, complete and terminating approach. We study the verification problem for such properties from the theorem proving point of view. In analogy with modular verification of contracts, our verification conditions (VCs) contain as assumptions certain universally quantified formulas (specifying the behavior of basic functions). As the goal, the VCs contain further universally quantified formulas that express the behavior of functions composed from basic ones. The key challenge in proving such VCs is the presence of quantifiers in their assumptions. Current SMT provers typically have incomplete support for quantifiers. Therefore, they typically cannot establish that a quantified formula is satisfiable. This limits their ability to provide meaningful counterexamples to the developer.

In this paper, we overcome the incompleteness of quantifier reasoning for a number of properties of interest. We use the methodology of local theory extensions [13] to obtain complete quantifier-instantiation strategies. We therefore arrive at decision procedures for quantified formulas about functions in the presence of decidable theories of primitive operations (such as linear arithmetic and algebraic data types). The procedure can prove the validity of universally quantified properties of functions, where functions themselves are specified using universally quantified axioms. Our algorithms are counterexample generating decision procedures. They terminate for all specifications in the identified classes. Given a formula, they either prove its validity, or generate a counterexample. The completeness for counterexamples is a significant aspect of usability, because it helps users identify concrete inputs that cause errors. At the same time, the completeness for generating proofs provides a higher degree of confidence compared to bounded-exhaustive test-case generation [4].

Single-invocation and many-invocation axioms. As our first class of properties that specify a function f we consider *single-invocation* axioms of the form $\forall \bar{x}.\Phi(\bar{x}, f(\bar{x}))$ for a formula Φ in a decidable theory. We show that it suffices to instantiate the universal quantifier over $\bar{x}$ only with $\bar{a}$ such that $f(\bar{a})$ occurs in the property being proved. Such completeness of instantiation has not been stated before, and is not to be taken for granted: we show that such instantiation is *incomplete* for general *two-invocation* axioms of the form $\forall \bar{x}, \bar{y}.\Phi(\bar{x}, \bar{y}, f(\bar{x}), f(\bar{y}))$. Despite this limitation, we identify a number of useful subclasses of two-invocation axioms that also remain local and decidable. These include properties of functions such as monotonicity, injectivity, congruence, and non-interference, all under a given abstraction function.

Locality of sufficiently surjective recursive abstractions. The results on both single-invocation and many-invocation axioms are particularly useful in the presence of recursively defined functions. Recent results [15, 17] show

the decidability of theories of algebraic data types with recursive abstraction functions which includes e.g. functions to compute set or multiset content, size, or height of algebraic data type (ADT) values. We verify that, like [15], the results of [17] can be naturally explained using Ψ-locality. We present a more abstract algorithm than the description in [17], possibly leading to more efficient future implementations.

Combination results. Given several classes of axioms for which reasoning is complete, a question arises whether they can be combined. We consider combinations of axioms that are separately local, and show several positive and negative results for the locality of their combinations. Finally, we give a positive result on *piecewise* combinations of local axioms. This is of significant practical value, because it enables functions to be specified using different local axiomatizations on different regions of their input domain, and generates a local axiomatization as a result.

In summary, we show a number of new decidability results for theories that support quantifiers, by showing that these quantifiers can be finitely instantiated in a complete way. Thus, we extend the predictability and efficiency of quantifier-free reasoning to an interesting class of axiomatically specified functions. Proofs have been omitted due to lack of space. They can be found in the technical report [8].

2 Example

Fig. 1 shows an illustrative functional program in the notation of the Scala programming language. We specify functions using global assertions written in Scala extended with a $\forall$ operator for universal quantification.

Many interesting properties of functional programs can be expressed using contracts. Contracts assign a pre- and a postcondition to every function, where the postcondition may depend on the value of function parameters. The specification of function merge in Fig. 1 illustrates how we can encode contracts using universally quantified axioms. The result of the function is a list containing the elements of both input lists. We express this requirement using an abstraction function content, mapping a list to the set of elements it contains. We show locality for such specifications in sections 4 and 5.

However, many other interesting properties of functions are *not* expressible through contracts, because they compare the values of different invocations of the function. Consider a user-defined equality on data structures, defined by the abstraction function content: $x \sim y \iff \mathsf{content}(x) = \mathsf{content}(y)$. By definition, $\sim$ is an equivalence relation, but it may or may not satisfy the *congruence* axiom: $x \sim y \rightarrow f(x) \sim f(y)$. Congruence states that the equivalence is preserved under application of data structure operations, and is generally a desirable property. For example, the function even in Figure 1 takes a list and returns the sublist containing the (integer) elements that are even numbers. The second assertion after even definition shows how to specify the congruence of even with respect to equivalence determined by content.

```
def merge(l1: List, l2: List): List = { ... }
assert(∀l1 : List ⇒ ∀l2 : List ⇒ content(merge(l1,l2)) = content(l1) ∪ content(l2))

def even(l : List): List = { ... }
assert(∀l: List ⇒ content(even(l)) ⊆ content(l))
assert(∀l1 : List ⇒ ∀ l2 : List ⇒
        content(l1) = content(l2) → content(even(l1)) = content(even(l2)))

def insert(c : Int, l: List): List = { ... }
assert(∀c: Int ⇒ ∀l1 : List ⇒ ∀l2 : List⇒
        content(l1) ⊆ content(l2) → content(insert(c,l1)) ⊆ content(insert(c,l2)))

def fill(l: List): List = { ... }
assert(∀l: List ⇒ if (content(l).size < 3) content(fill(l)).size=3
                  else content(fill(l)) = content(l) )

def main(c: Int, l1: List, l2: List): List = merge(even(l1), fill(insert(c,l2)))

case class Employee(name : String, age : Int, bankAccountNo : Int)
def samePublic1(e1 : Employee, e2 : Employee) : Boolean =
  { e1.name = e2.name && e1.age = e2.age }
def samePublic(el1 : List[Employee], el2 : List[Employee]) : Boolean =
    zip(el1,el2).forAll(samePublic1) && length(el1)=length(el2)
assert(equivalence(samePublic))

def averageAge(emp : List[Employee]) : Int = { ... }
// information flow property: averageAge does not depend on private data
assert(∀l1: List[Employee] ⇒ ∀l2: List[Employee] ⇒
  samePublic(l1,l2) → averageAge(l1) = averageAge(l2)
```

Fig. 1. Example of Functional Program Specified Using Axioms

For other functions, such as insert in Fig. 1, we may require stronger properties, such as monotonicity with respect to the pre-order. The pre-order can be induced by an abstraction function that maps into an ordered structure. In case of the content abstraction, the starting point is the subset ordering, and we define $x \preceq y \iff \mathsf{content}(x) \subseteq \mathsf{content}(y)$. Then, we expect a function that inserts a given element into the data structure to satisfy $x \preceq y \rightarrow \mathsf{insert}(c, x) \preceq \mathsf{insert}(c, y)$.

It can also be useful to specify the data that a function is allowed to access. For example, averageAge in Fig. 1 is a function that should not access the private information about employees. We specify this by first defining an equivalence relation samePublic1 on employees, abstracting from the private data bankAccountNo, then an equivalence relation on lists of employees (which are equivalent if they have the same length and elements at the same position in a list are in relation samePublic1), and finally assert that whenever two lists are equivalent wrt. samePublic, then averageAge will give the same result for both lists. The results

in Section 6 ensure complete reasoning about specifications that take the form of congruence, monotonicity, or injectivity.

We may also wish to combine specifications using conjunctions or if-then-else. As an example of combination using conjunction, the function even in Fig. 1 should satisfy the congruence, and, in addition, it should return a subset of the original list. As an example of combination by if-then-else, the fill function should insert additional elements into lists with less than 3 elements (so that the result has 3 distinct elements), and otherwise should not change the list content. Sections 7 and 8 establish results on preserving the decidability of reasoning when combining specifications.

As with the usual contracts, our overall goal is to verify partial correctness of programs in a modular way. We wish to check the properties of functions such as main in Figure 1, assuming the assertions of the functions it calls. The properties we check can be universally quantified and can include contracts, as well as e.g. congruence or monotonicity. When the function verified (e.g. main) is not recursive, proving such contracts reduces to deciding quantifier-free formulas in the theory given by the specifications of the functions called. The approach that we introduce allows us to decide such questions. Moreover, if a property does not hold, we obtain a counterexample that points us to the weakness in our specification or code, which we can use to make the appropriate correction.

3 Background

The notion of *local theory extensions* is key to our decidability results. We give a short introduction to the concept, which was developed by Sofronie-Stokkermans [13]. For more details, we refer to [5, 7, 13].

Theories and models. Consider a signature $\Pi = (\Sigma, \mathsf{Pred})$, where Σ is a set of function symbols and Pred a set of predicate symbols (both with given arities). A *Π-structure* $\mathcal{M}$ consists of a non-empty set of elements M, a total function $f^{\mathcal{M}} : M^n \to M$ for every n-ary function symbol $f \in \Sigma$, as well as a set $P^{\mathcal{M}} \subseteq M^n$ for every n-ary predicate symbol $P \in \mathsf{Pred}$. We regard *theories* as sets of formulas closed under consequences, defined by a set of *axioms*. A given (Π-)structure $\mathcal{M}$ is a *model* of a theory $\mathcal{T}$ iff every axiom of $\mathcal{T}$ is satisfied by $\mathcal{M}$. If a formula F is satisfied by a structure $\mathcal{M}$, we write $\mathcal{M} \models F$. If F is true in all models of $\mathcal{T}$, we write $\models_{\mathcal{T}} F$. If no model of $\mathcal{T}$ satisfies F, we write $F \models_{\mathcal{T}} \Box$, where $\Box$ represents the empty clause.

Local theory extensions. *Theory extensions* extend a given theory with new function symbols, defined by a set of axioms. *Locality* of the extension ensures that reasoning about these symbols can be reduced to reasoning in the base theory by finite instantiation of the axioms. The new symbols are called *extension symbols*, terms starting with extension symbols are *extension terms*.

Consider a background theory $\mathcal{T}$ with signature $\Pi_0 = (\Sigma_0, \mathsf{Pred})$, and an extension $\Pi = (\Sigma_0 \cup \Sigma_1, \mathsf{Pred})$ of this signature with extension symbols in Σ_1. An *augmented Π-clause* is a Π-formula $\forall \overline{x}.\ \Phi(\overline{x}) \vee C(\overline{x})$, where $\Phi(\overline{x})$ is an arbitrary

Π_0-formula and $C(\overline{x})$ is a disjunction of Π-literals. We say that it is Σ_1-*ground* if $C(\overline{x})$ is ground. A *theory extension* of a theory $\mathcal{T}$ with signature Π_0 is given by a set $\mathcal{K}$ of augmented Π-clauses, representing axioms for the extension symbols.

A substitution σ is a function mapping variables to terms. By $F\sigma$ we denote the result of simultaneously replacing each free variable x in F with $\sigma(x)$. For a set of formulas $\mathcal{K}$, define $\mathsf{st}(\mathcal{K})$ as the set of ground subterms appearing in $\mathcal{K}$. For a set of Π-formulas $\mathcal{K}$ and a Π-formula G, let

$$\mathcal{K}[G] = \{\ F\sigma \mid \begin{array}{l} F \in \mathcal{K} \text{ and } \sigma \text{ is such that} \\ f(\overline{t})\sigma \in \mathsf{st}(\mathcal{K} \cup G) \text{ for each extension subterm } f(\overline{t}) \text{ of } F, \\ \text{and } \sigma(x) = x \text{ if } x \text{ does not appear in an extension term } \}, \end{array}$$

i.e. $\mathcal{K}[G]$ is the result of matching extension terms in $\mathcal{K}$ to ground terms in $\mathcal{K} \cup G$. We consider theory extensions with the following locality property (defined in [13]):

(ELoc) For every set G of Σ_1-ground augmented Π-clauses, we have $\mathcal{K} \cup G \models_{\mathcal{T}} \square \iff \mathcal{K}[G] \cup G \models_{\mathcal{T}} \square$

Decidability and model generation. Assuming (ELoc), satisfiability of G modulo $\mathcal{T} \cup \mathcal{K}$ is decidable whenever $\mathcal{K}[G] \cup G$ is finite and belongs to a decidable fragment of $\mathcal{T}$ (plus free functions). In particular, if G is ground, and all variables in $\mathcal{K}$ appear in extension terms, then $\mathcal{K}[G] \cup G$ is ground, and decidability of the ground fragment of $\mathcal{T}$ is sufficient. As mentioned in [5], model-generating decision procedures for $\mathcal{T}$ can be used to produce models for $\mathcal{T} \cup \mathcal{K}$.

Identifying local theory extensions. To formulate a sufficient condition for theory extensions satisfying (ELoc), we need some additional definitions.

A *partial Π-structure* is the same as a Π-structure, except that function symbols may be assigned partial functions. In a partial structure $\mathcal{M}$, terms are evaluated wrt. a variable assignment β as in total structures, except that the evaluation of $\beta(f(t_1, \ldots, t_n))$ is undefined if either $(\beta(t_1), \ldots, \beta(t_n))$ is not in the domain of $f^{\mathcal{M}}$, or at least one of the $\beta(t_i)$ is undefined. A partial Π-structure $\mathcal{M}$ and a variable assignment β *weakly satisfy* a literal L if either all terms in L are defined and the usual notion of satisfaction applies, or if at least one of the terms in L is undefined. Based on weak satisfaction of literals, weak satisfaction of formulas is defined recursively in the usual way. If $\mathcal{M}$ satisfies F for all variable assignments β, $\mathcal{M}$ is a *weak partial model* of F.

For $\Pi = (\Sigma, \mathsf{Pred})$, a total Π-structure $\mathcal{M}$ is a *completion* of a partial Π-structure $\mathcal{M}'$ if (1) $M = M'$, (2) for every $f \in \Sigma$: $f^{\mathcal{M}}(\overline{x}) = f^{\mathcal{M}'}(\overline{x})$ whenever $f^{\mathcal{M}'}(\overline{x})$ is defined, and (3) for every $P \in \mathsf{Pred}$: $P^{\mathcal{M}} = P^{\mathcal{M}'}$.

For extensions of a theory $\mathcal{T}$ with signature $\Pi_0 = (\Sigma_0, \mathsf{Pred})$ with a set of axioms $\mathcal{K}$ with signature $\Pi = (\Sigma_0 \cup \Sigma_1, \mathsf{Pred})$, define the completability property

($\mathsf{Comp_w}$) For every weak partial Π-model $\mathcal{M}$ of $\mathcal{T} \cup \mathcal{K}$ where Σ_0-functions are total, there exists a completion which is a model of $\mathcal{T} \cup \mathcal{K}$

A formula F is Σ_1-*flat* if it does not contain occurrences of function symbols below a Σ_1-symbol. A Σ_1-flat formula F is Σ_1-*linear* if all extension terms in F

which contain the same variable are syntactically equal, and no extension term in F contains two or more occurrences of the same variable.

Theorem 1 (Completability implies extended locality [13]). *If $\mathcal{K}$ consists of Σ_1-linear augmented clauses and the extension of $\mathcal{T}$ with $\mathcal{K}$ satisfies* ($\mathsf{Comp_w}$)*, then it also satisfies* (ELoc)*.*

Combinations and chains of extensions. For combining several local theory extensions with different extension functions, two approaches have been considered in the literature. If we have two extensions of $\mathcal{T}$ with $\mathcal{K}_1$ and $\mathcal{K}_2$, respectively, that introduce disjoint sets of function symbols and individually satisfy ($\mathsf{Comp_w}$), then the extension of $\mathcal{T}$ with $\mathcal{K}_1 \cup \mathcal{K}_2$ also satisfies ($\mathsf{Comp_w}$) [14]. On the other hand, an extended theory can be extended again, so we can extend $\mathcal{T}$ with $\mathcal{K}_1$, then $\mathcal{T} \cup \mathcal{K}_1$ with $\mathcal{K}_2$, and so on, if every extension satisfies ($\mathsf{Comp_w}$).

These combination results allow us to efficiently reason about programs containing multiple user-specified functions: if two functions are defined separately (i.e. they don't appear in the definition of each other), then the combination of both axioms is also a local extension, and if one is defined in terms of the other, we can use a chain of extensions. There are currently no combination results for mutually recursive functions.

In sections 5 to 8, we introduce several classes of axioms and prove that they satisfy ($\mathsf{Comp_w}$), and thus the locality property (ELoc).

4 Reasoning about ADTs with Abstractions

In this section, we introduce a logic that allows us to reason about axiomatic specifications of functional programs. We consider the theory of ADTs with recursive abstraction functions introduced in [17] as our base theory, and use the framework of local theory extensions [13] to reason about axiomatic specifications of additional functions that manipulate these data structures. The syntax of our logic, parametrized by element and collection theory, can be found in Fig. 2. In the rest of the paper, we will prove decidability of this logic.

Several decidability results for recursive functions over algebraic data types are presented in [15, 17]. Whereas [15] uses local theory extensions, [17] uses a criterion of sufficient surjectivity, related to the notion of counting constraints of [19]. In the following, we give a decision procedure that follows [17] but uses local theory extensions. It decides satisfiability of ground formulas in our logic (see Fig. 2; we ignore additional function symbols $f \notin \Sigma_T$ for now, but they could easily be added as free function symbols).

We present our decision procedure for the specific ADT of binary trees, but it generalizes to data types with other constructors. Our decision procedure is parametrized by an *element theory* $\mathcal{T}_E$, a *collection theory* $\mathcal{T}_C$ and an *abstraction function* α, which is recursively defined on the tree structure by $\mathcal{K}_\alpha = \{\alpha(\mathsf{Leaf}) = \mathsf{empty},\ \forall t_1, e, t_2.\ \alpha(\mathsf{Node}(t_1, e, t_2) = \mathsf{combine}(\alpha(t_1), e, \alpha(t_2))\}$, where empty is a ground term in $\mathcal{T}_C$ and $\mathsf{combine}$ is a function which maps two

Element terms:	E	$::= e \mid T_E$
Tree terms:	T	$::= t \mid \mathsf{Leaf} \mid \mathsf{Node}(T,E,T) \mid \mathsf{Left}(T) \mid \mathsf{Right}(T) \mid f(T)$ for $f \notin \Sigma_T$
Collection terms:	C	$::= c \mid \alpha(T) \mid T_C$
Element literals:	L_E	$::= \mathcal{L}_E$ (given by $\mathcal{T}_E$)
Tree literals:	L_T	$::= T = T$
Collection literals:	L_C	$::= C = C \mid C \leq C \mid \mathcal{L}_C$ (given by $\mathcal{T}_C$)
Ground Formulas:	φ	$::= L_E \mid L_T \mid L_C \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi$
Axioms:	ϕ	$::= \forall x.\Phi(x, f(x))^\star \mid (\mathsf{Con}) \mid (\mathsf{Mon}) \mid (\mathsf{Inj})$
Conjunctive comb.:	$\phi^\wedge$	$::= \phi \mid (\mathsf{Con}) \wedge (\mathsf{Mon}) \mid (\mathsf{Con}) \wedge (\mathsf{Inj}) \mid \forall x.\Phi(x, f(x)) \wedge (\mathsf{Con})^\star$ $\mid \forall x.\Phi(x, f(x)) \wedge (\mathsf{Inj})^\star \mid \forall x.\Phi(x, f(x)) \wedge (\mathsf{Con}) \wedge (\mathsf{Inj})^\star$
Piecewise comb.:	$\phi^\vee$	$::= \bigwedge_j \left((\bigwedge_i \varphi_j(x_i)) \rightarrow \phi_j^\wedge \right)^\star$
Formulas:	ϕ^+	$::= \phi^\vee \wedge \varphi$

Cases marked with $\star$ require additional side conditions, see Sect. 5 to 8.

Fig. 2. Syntax of our logic

arguments of collection sort and one argument of element sort to a value of collection sort.[1] We assume that α is *sufficiently surjective* as defined in [17].

Key steps of the decision procedure. We give a decision procedure for conjunctions of literals. It can be lifted to arbitrary ground formulas by using the well-known $DPLL(\mathcal{T})$ approach.

Purification and flattening. We purify the formula s.t. every literal only contains symbols from one of the theories $\mathcal{T}_E, \mathcal{T}_C$ or $\mathcal{T}_T$ (the theory of trees). Then we flatten tree terms s.t. constructors and selectors are only applied to tree variables (and arbitrary element terms), and tree disequalities do not contain non-variable terms. Both can be done by introduction of fresh variables.

Elimination of selectors. We rewrite equations $t = \mathsf{Left}(t_1)$ to $t_1 = \mathsf{Node}(t_L, e, t_R) \wedge t = t_L$, and $t = \mathsf{Right}(t_1)$ to $t_1 = \mathsf{Node}(t_L, e, t_R) \wedge t = t_R$, where t_L, t_R and e are always fresh variables.

Unification of tree literals. We apply unification on the positive tree literals (see [17] for details). If unification fails, we have shown unsatisfiability. Otherwise, we obtain a solution σ that can be seen as a substitution for tree and element variables. We apply this substitution to the formula and call the result G. Tree variables t with $\sigma(t) = t$ will in the following be called *parameter variables*. If for any disequality $x \neq y$ between tree or element variables we have $\sigma(x) = \sigma(y)$, then we also have shown unsatisfiability. Otherwise, we continue.

Adding additional constraint P**.** To ensure equisatisfiability to the original problem, we have to add an additional constraint P that depends on the abstraction function α, as well as on the variables and terms of tree sort in the given formula. Generation of P for $\mathsf{content}$ and other abstraction functions is given in [17].

[1] Typically, we consider theories of sets or multisets as collection theory, but integers or even booleans can also be considered as "collections".

Partial evaluation of α**.** We partially evaluate α in $G \wedge P$ by adding, for every term $\alpha(\mathsf{Node}(T_1, e, T_2))$, the recursive definition $\alpha(\mathsf{Node}(T_1, e, T_2)) = \mathsf{combine}(\alpha(T_1), e, \alpha(T_2))$, and recursively for the subterms T_1 and T_2. The set of all these recursive definitions, including $\alpha(\mathsf{Leaf}) = \mathsf{empty}$, will be called $\mathcal{K}_\alpha[G]$ (note that all terms in P which appear below α are also in G, so the set of needed definitions does not depend on P). G may still contain equalities of the form $\alpha(T_j) = c_j$, where T_j may be either a parameter variable or a term containing parameter variables. In the latter case, $\mathcal{K}_\alpha[G]$ contains all recursive definitions to uniquely determine $\alpha(T_j)$, given the values $\alpha(t_i)$ for all parameter variables t_i and values of element variables e_i appearing in T_j.

Solving the resulting problem. The resulting formula, when considering α as a free function symbol, is equisatisfiable to the original one. We can check its satisfiability with a combined decision procedure for $\mathcal{T}_T \cup \mathcal{T}_C \cup \mathcal{T}_E$.

Correctness of the decision procedure. It is clear that the preprocessing steps are satisfiability-preserving. Therefore, we only state that the constraint G (in the theory $\mathcal{T}_T \cup \mathcal{T}_c \cup \mathcal{T}_E \cup \mathcal{K}_\alpha$, where $\mathcal{K}_\alpha$ are the universal recursive definitions of α) is equisatisfiable to the resulting formula $G \wedge P \wedge \mathcal{K}_\alpha[G]$ in $\mathcal{T}_T \cup \mathcal{T}_c \cup \mathcal{T}_E$.

Theorem 2 (Correctness of Decision Procedure for Abstraction Functions). *Let G be a conjunction of literals which has been preprocessed as mentioned above, let P be as defined in [17] (based on G and α), and $\mathcal{K}_\alpha$ and $\mathcal{K}_\alpha[G]$ as defined above. Then* $G \models_{\mathcal{T}_T \cup \mathcal{T}_c \cup \mathcal{T}_E \cup \mathcal{K}_\alpha} \Box \iff G \cup P \cup \mathcal{K}_\alpha[G] \models_{\mathcal{T}_T \cup \mathcal{T}_c \cup \mathcal{T}_E} \Box$.

Compared to [17], we improved efficiency of the decision procedure by removing the case split on (dis)equalities of variables, which are now partially determined by unification, and then negotiated by a combined decision procedure for the base theory. Also, we do not need to convert formula P to DNF, since our correctness argument works for arbitrary boolean structure of P.

5 Complete Reasoning about Single-Invocation Axioms

Suppose we are given a function f, and we wish to prove $P(f(\overline{t}_1), \ldots, f(\overline{t}_n))$ for all values of free variables in P. In condition P, function f is applied to arbitrary terms $\overline{t}_1, \ldots, \overline{t}_n$ from some decidable theory, such as the theory of linear arithmetic, lists, sets, or trees. We wish to prove P valid using only a contract-like specification of f as an assumption. Consider a contract for f with a quantifier-free precondition $\mathsf{Pre}(\overline{x})$ and a quantifier-free postcondition $\mathsf{Post}(\overline{x}, r)$, with r denoting the resulting value. We model such a contract as the formula $\forall \overline{x}.\Phi(\overline{x}, f(\overline{x}))$, where $\Phi(\overline{x}, r)$ is $\mathsf{Pre}(\overline{x}) \rightarrow \mathsf{Post}(\overline{x}, r)$. Our goal is to show that the contract implies the desired property, that is, that the following formula is valid: $(\forall \overline{x}.\Phi(\overline{x}, f(\overline{x}))) \rightarrow P(f(\overline{t}_1), \ldots, f(\overline{t}_n))$. Equivalently, we aim to show that $(\forall \overline{x}.\Phi(\overline{x}, f(\overline{x}))) \wedge \neg P(f(\overline{t}_1), \ldots, f(\overline{t}_n))$ is an unsatisfiable formula.

In this section we show that we can reduce such satisfiability problems to the simpler *quantifier-free* satisfiability problem $\left(\bigwedge_{i=1}^{n} \Phi(\overline{t}_i, f(\overline{t}_i))\right) \wedge \neg P(f(\overline{t}_1), \ldots, f(\overline{t}_n))$, in which the universal quantifier is instantiated only with

the terms t_i. Note that the above instantiation confirms two important special cases. First, when $\Phi(\overline{x}, r)$ specifies the behavior of f completely, that is, r is unique for a given $\overline{x}$, then the axiom is simply a form of one-point rule applied to f as a variable. Second, if we view P as a program that invokes f, then the instantiation principle above reflects modular reasoning by inlining contracts. While it is known that such reasoning is sound, this is usually shown using operational semantics. In our formulation, the approach is sound thanks to quantifier instantiation. Moreover, the following theorem shows that this approach is also complete, as a consequence of a locality result.

Theorem 3 (Locality for Single-Invocation Axioms). *Let $\mathcal{T}$ be a theory with signature $\Pi_0 = (\Sigma_0, \mathsf{Pred})$, f a fresh function symbol and $\Phi(\overline{x}, f(\overline{x}))$ a $(\Sigma_0 \cup \{f\}, \mathsf{Pred})$-formula with $\overline{x}$ as the vector of all free variables and $f(\overline{x})$ the only non-ground term containing f. Then ($\mathsf{Comp_w}$) holds for the extension of $\mathcal{T}$ with $\forall \overline{x}.\ \Phi(\overline{x}, f(\overline{x}))$ if and only if $\models_{\mathcal{T}} \forall \overline{x} \exists y.\ \Phi(\overline{x}, y)$.*

By Thm. 1, ($\mathsf{Comp_w}$) implies (ELoc), which ensures that we can decide satisfiability of formulas with respect to such axioms by simply adding one instance of the axiom for every occurrence of the function symbol in a given formula.[2]

The side condition $\models_{\mathcal{T}} \forall \overline{x} \exists y.\ \Phi(\overline{x}, y)$ guarantees that the specification is consistent. It holds whenever there exists a function f with $\forall \overline{x}.\ \Phi(\overline{x}, f(\overline{x}))$. Consequently, if we have proved that some function satisfies its contract, we can use finite instantiation to reason about the contract in a complete way.

Let us also remark that Thm. 3 subsumes the decidability result from field constraint analysis [18] in a more systematic form, by expressing it as a locality result. We have shown that this result applies also to contracts of functional programs. Moreover, in addition to better understanding, locality gives us efficient implementations in the form of incremental instance generation [6].

Loss of locality for general many-invocation axioms. In general, the straightforward modification of Thm. 3 for axioms with multiple function invocations does not hold. Consider the background theory of integers $\mathcal{T}_{\mathbb{Z}}$ and its extension with strict monotonicity

$$\forall x_1, x_2.\ x_1 < x_2 \rightarrow f(x_1) < f(x_2). \qquad (\mathsf{SMon})$$

Although the axiom is consistent, the extension of $\mathcal{T}_{\mathbb{Z}}$ with (SMon) is not local (and thus cannot satisfy ($\mathsf{Comp_w}$)). Indeed, take $G = \{f(0) = 0, f(2) = 1\}$. Then ($\mathsf{SMon}$)$[G] \cup G$ is $\mathcal{T}_{\mathbb{Z}}$-satisfiable, but (SMon) $\cup\, G$ is unsatisfiable, because a strictly monotonic function on integers cannot have $f(0) = 0$ and $f(2) = 1$. Despite this negative result, in the next sections we show that in a number of cases of interest we can also obtain locality results for many-invocation axioms.

[2] Note that in this case, we do not need the restriction to augmented clauses, since for any G the local instantiation of $\Phi(\overline{x}, f(\overline{x}))$ can easily be shown to be equivalent to the instantiation of its augmented CNF (where Π_0-formulas are treated as literals).

6 Complete Reasoning about Many-Invocation Axioms

In this section, we push decidability of reasoning over axiomatic specifications beyond function contracts to more expressive axiom classes.

Congruence. We want to consider axiomatic specifications which go beyond contracts, in that they allow multiple function invocations. We start with congruence properties, as defined for function even in Fig. 1.

We consider equivalence relations $\sim$ on data types given by abstraction functions like content, i.e. $x \sim y \iff \mathsf{content}(x) = \mathsf{content}(y)$.

The following result allows us to reason about specifications ensuring congruence of a function wrt. an equivalence relation $\sim$. In the following Theorem, the equivalence relation $\dot{\sim}$ may be defined by $\overline{x} \dot{\sim} \overline{y} \iff \bigwedge_{i=1}^{n} x_i \sim y_i$, but it can also be any other equivalence relation on the domain of f.

Theorem 4 (Locality of Congruence). *If $\mathcal{T}$ is a theory with equivalence relations $\sim$ and $\dot{\sim}$, then* ($\mathsf{Comp_w}$) *holds for the extension of $\mathcal{T}$ with a function f satisfying*

$$\forall \overline{x}, \overline{y}.\ \ \overline{x} \dot{\sim} \overline{y} \ \rightarrow\ f(\overline{x}) \sim f(\overline{y}). \qquad (\mathsf{Con})$$

This theorem allows us to decide satisfiability problems with user-defined functions that are specified to be congruent with respect to a given abstraction, such as the averageAge function in the Fig. 1 example.

Monotonicity. Another important requirement for many functions is monotonicity wrt. a given abstraction, as specified for function insert in Fig. 1.

By ordering data structures wrt. the given abstraction, we define a new order with $x \preceq y \iff \mathsf{content}(x) \subseteq \mathsf{content}(y)$.

Because it is defined by a function, such a user-defined order satisfies the defining properties of the original order, except antisymmetry (unless the abstraction function is injective). That is, if the original order is a total order then the user-defined order will be a total preorder, if it is a lattice the user-defined order will be a prelattice, etc.

The following theorem extends known results on local extensions with monotone functions [16, 7] to the case of preorders and bounded (semi-)prelattices:

Theorem 5 (Locality of Monotonicity). *Let $\mathcal{T}$ be a theory with a binary relation $\preceq$ such that either (i) $\preceq$ is a total order, or (ii) $\preceq$ defines a bounded semi-lattice. Let $R(x, y)$ be a binary predicate which is transitive in $\mathcal{T}$. Then*
(i) (Jacobs, Sofronie-Stokkermans [7])
($\mathsf{Comp_w}$) *holds for the extension of $\mathcal{T}$ with a function f satisfying*

$$\forall \overline{x}, \overline{y}.\ \ R(\overline{x}, \overline{y}) \ \rightarrow\ f(\overline{x}) \preceq f(\overline{y}). \qquad (\mathsf{Mon})$$

(ii) ($\mathsf{Comp_w}$) *is already satisfied if $\preceq$ is a total preorder, or defines a bounded semi-prelattice.*

For the abstraction of data structures to their set of elements, $\preceq$ is a bounded prelattice, and for abstractions to their length or size, it is a total preorder.

Injectivity. Another important property of functions manipulating data structures is injectivity. The following result allows us to decide specifications that require injectivity of a function wrt. a given abstraction, and is a generalization of known locality results wrt. equality [5]:

Theorem 6 (Locality of Injectivity). *If $\mathcal{T}$ is a theory with equivalence relations $\sim$ and $\dot{\sim}$ such that (in every model of $\mathcal{T}$)*[3] *there are infinitely many equivalence classes wrt. $\sim$, then* $(\mathsf{Comp}_{\mathsf{w,f}})$[4] *holds for the extension of $\mathcal{T}$ with a function f satisfying*

$$\forall \overline{x}, \overline{y}.\ \ \neg(\overline{x} \dot{\sim} \overline{y}) \ \rightarrow\ \neg(f(\overline{x}) \sim f(\overline{y})). \qquad (\mathsf{Inj})$$

7 Complete Reasoning about Conjunctive Combinations

In this Section, we consider the problem of reasoning about specifications which impose several axioms from the classes introduced in Sect. 5 and 6 on the same function, as for function even from Fig. 1.

Theorem 7 (Locality of Conjunctive Combinations 1). *Let $\mathcal{T}$ be a theory with equivalence relations $\sim$ and $\dot{\sim}$.*

(i) Let $\Phi(x, f(x))$ be as in Thm. 3. Then $(\mathsf{Comp}_{\mathsf{w}})$ *holds for the extension of $\mathcal{T}$ with a function f satisfying* $(\mathsf{Con}) \cup \forall \overline{x}.\ \Phi(\overline{x}, f(\overline{x}))$ *if and only if* $\models_{\mathcal{T}} \forall \overline{x_1}, \overline{x_2}, y_1.\ (\overline{x_1} \dot{\sim} \overline{x_2} \wedge \Phi(\overline{x_1}, y_1) \rightarrow \exists y_2.\ y_1 \sim y_2 \wedge \Phi(\overline{x_2}, y_2))$.

(ii) Let $\preceq$ and $R(x, y)$ be as in Thm. 5, and furthermore such that $\models_{\mathcal{T}} x \preceq y \wedge y \preceq x \iff x \sim y$. *Then* $(\mathsf{Comp}_{\mathsf{w}})$ *holds for the extension of $\mathcal{T}$ with a function f satisfying* $(\mathsf{Con}) \cup (\mathsf{Mon})$.

(iii) $(\mathsf{Comp}_{\mathsf{w}})$ *holds for the extension of $\mathcal{T}$ with a function f satisfying* $(\mathsf{Con}) \cup (\mathsf{Inj})$ *if and only if* $|dom(f)/\dot{\sim}| \leq |codom(f)/\sim|$.

In contrast to these positive results, an extension with $(\mathsf{Mon}) \cup (\mathsf{Inj})$ or $(\mathsf{Con}) \cup (\mathsf{Mon}) \cup (\mathsf{Inj})$ will in general not be local. This essentially corresponds to strict monotonicity, mentioned as a counterexample to locality at the end of Sect. 5. We can however obtain some more positive results:

Theorem 8 (Locality of Conjunctive Combinations 2). *Let $\mathcal{T}$ be a theory with equivalence relations $\sim$ and $\dot{\sim}$, and $\Phi(\overline{x}, f(\overline{x}))$ as in Thm. 3. Then*

[3] In fact, it would be sufficient to have a notion similar to *stable infiniteness* wrt. the equivalence classes, i.e. whenever a formula is satisfiable, it is also satisfiable in a model with infinitely many equivalence classes.

[4] $(\mathsf{Comp}_{\mathsf{w,f}})$ is a weaker form of $(\mathsf{Comp}_{\mathsf{w}})$ that only considers embeddability of models where partial functions have a finite domain. It implies the slightly weakened locality condition $(\mathsf{ELoc}_{\mathsf{f}})$, which requires G to be finite.

(i) $(\mathsf{Comp_w})$ *holds for the extension of* $\mathcal{T}$ *with a function* f *satisfying* $(\mathsf{Inj}) \cup \forall \overline{x}.\ \Phi(\overline{x}, f(\overline{x}))$ *if* $\models_{\mathcal{T}} \forall \overline{x_1}, \overline{x_2}, y_1.\ (\neg(\overline{x_1} \dot{\sim} \overline{x_2}) \wedge \Phi(\overline{x_1}, y_1) \rightarrow \exists^{\infty} y_2.\ y_1 \not\sim y_2 \wedge \Phi(\overline{x_2}, y_2))$.

(ii) $(\mathsf{Comp_w})$ *holds for the extension of* $\mathcal{T}$ *with a function* f *satisfying* $(\mathsf{Con}) \cup (\mathsf{Inj}) \cup \forall x.\ \Phi(x, f(x))$ *if and only if* $\models_{\mathcal{T}} \quad \forall \overline{x_1}, \overline{x_2}, y_1.\quad (\overline{x_1} \dot{\sim} \overline{x_2} \wedge \Phi(\overline{x_1}, y_1) \rightarrow \exists y_2.\ y_1 \sim y_2 \wedge \Phi(\overline{x_2}, y_2))$ *and* $|dom(f)/\dot{\sim}| \leq |codom(f)/\sim|$.

Above, $\exists^{\infty} y_2$ means that there exist infinitely many values y_2 in different equivalence classes wrt. $\sim$. Combination of contracts with (Mon) is in general not local, not even with the restriction to contracts which allow monotone functions.

8 Complete Reasoning about Piecewise Combinations

In some cases, it is desirable to use specifications with a case distinction over different partitions of the function domain, as for function fill in Fig. 1.

While single-invocation axioms allow case distinctions intrinsically, this is not the case for the other axiom classes we introduced, or local theory extensions in general. In the following, we consider sets of axioms $\mathcal{K}(f(\overline{x}_1), \ldots, f(\overline{x}_n))$, where in each axiom there may be up to n invocations of f. We first show that we can have different local axiomatizations in disjoint subsets of the domain, and the resulting "piecewise" axiomatization will again be local. Furthermore, we show that a piecewise local axiomatization can also be obtained for non-disjoint subsets of the domain, as long as the overlaps are finite.

In the following, we use restrictions of formulas $\mathcal{K}(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ to a subset of the domain of f, specified by a formula $\Phi(\overline{x})$. We denote by $\bigwedge_{i=1}^{n} \Phi(\overline{x}_i) \rightarrow \mathcal{K}(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ the set of augmented Π-clauses $\{\forall \overline{x}_1, \ldots, \overline{x}_n.\ \bigwedge_{i=1}^{n} \Phi(\overline{x}_i) \rightarrow F \vee C \mid F \vee C \in \mathcal{K}(f(\overline{x}_1), \ldots, f(\overline{x}_n))\}$. We state that such restrictions do not destroy locality properties:

Lemma 1. *Let* $\mathcal{T}$ *be a* Π_0*-theory and* $\mathcal{K}(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ *a set of augmented* Π*-clauses such that* $\mathcal{T} \subseteq \mathcal{T} \cup \mathcal{K}(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ *satisfies* $(\mathsf{Comp_w})$*. For any* Π_0*-formula* $\Phi(\overline{x})$*, the extension* $\mathcal{T} \subseteq \mathcal{T} \cup (\Phi(\overline{x}) \rightarrow \mathcal{K}(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ *also satisfies* $(\mathsf{Comp_w})$*.*

Lem. 1 and the next theorem allow piecewise combinations satisfying $(\mathsf{Comp_w})$:

Theorem 9 (Locality of Disjoint Piecewise Combinations). *Let* $\mathcal{T}$ *be a* Π_0*-theory and consider* Π_0*-formulas* $\Phi_1(\overline{x}), \Phi_2(\overline{x})$ *such that* $\models_{\mathcal{T}} \neg(\Phi_1(\overline{x}) \wedge \Phi_2(\overline{x}))$*. If* $\mathcal{K}_1(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ *and* $\mathcal{K}_2(f(\overline{x}_1), \ldots, f(\overline{x}_m))$ *are* Π*-formulas such that* $(\mathsf{Comp_w})$ *holds for both* $\mathcal{T} \subseteq \mathcal{T} \cup (\bigwedge_{i=1}^{n} \Phi_1(\overline{x}_i) \rightarrow \mathcal{K}_1(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ *and* $\mathcal{T} \subseteq \mathcal{T} \cup (\bigwedge_{i=1}^{m} \Phi_2(\overline{x}_i) \rightarrow \mathcal{K}_2(f(\overline{x}_1), \ldots, f(\overline{x}_m))$*, then* $(\mathsf{Comp_w})$ *also holds for* $\mathcal{T} \subseteq \mathcal{T} \cup \mathcal{K}$*, with*

$$\begin{aligned} \mathcal{K} = \quad & (\textstyle\bigwedge_{i=1}^{n} \Phi_1(\overline{x}_i)) \rightarrow \mathcal{K}_1(f(\overline{x}_1), \ldots, f(\overline{x}_n)) \\ \cup\ & (\textstyle\bigwedge_{i=1}^{m} \Phi_2(\overline{x}_i)) \rightarrow \mathcal{K}_2(f(\overline{x}_1), \ldots, f(\overline{x}_m)). \end{aligned}$$

Repeated application of Thm. 9 directly gives a locality result for arbitrarily many case distinctions.

Non-disjoint subsets with a finite intersection. If the overlap between cases is finite, we can preserve completeness by instantiating the axioms additionally for all elements in the overlap. To this end, consider a closure operator Ψ on ground terms and the more general notion of Ψ-completability [5]

($\mathsf{Comp}_{\mathsf{w}}^{\Psi}$) For every weak partial Π-model $\mathcal{M}$ of $\mathcal{T} \cup \mathcal{K}$ where Σ_0-functions are total and the definition domain of Σ_1-functions is closed under Ψ, there exists a completion which is a model of $\mathcal{T} \cup \mathcal{K}$,

which implies the Ψ-locality condition

(ELoc^{Ψ}) For every set G of Σ_1-ground augmented Π-clauses, we have $\mathcal{K} \cup G \models_{\mathcal{T}} \square \iff \mathcal{K}^{\Psi}[G] \cup G \models_{\mathcal{T}} \square$

with $\mathcal{K}^{\Psi}[G]$ defined like $\mathcal{K}[G]$, except extension terms may be in $\Psi(\mathsf{st}(\mathcal{K} \cup G))$.

Then, with a suitable Ψ we can prove Ψ-locality for piecewise combinations with finite overlaps:

Theorem 10 (Locality of Piecewise Combinations with Finite Intersection). *Let $\mathcal{T}$ be a theory with signature $\Pi_0 = (\Sigma_0, \mathsf{Pred})$ and consider Π_0-formulas $\Phi_1(\overline{x}), \Phi_2(\overline{x})$ such that in every $\mathcal{T}$-model $\mathcal{M}$, the set $O = \{\overline{x} \in M | \Phi_1(\overline{x}) \wedge \Phi_2(\overline{x})\}$ is finite. Let furthermore T_0 be a set of Σ_0-terms such that in every such model $\mathcal{M}$, $O \subseteq \{t^{\mathcal{M}} \mid t \in T_0\}$.*

If $\mathcal{K}_1(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ and $\mathcal{K}_2(f(\overline{x}_1), \ldots, f(\overline{x}_m))$ are sets of augmented Π-clauses such that ($\mathsf{Comp}_{\mathsf{w}}$) holds for both the extensions of $\mathcal{T}$ with $(\bigwedge_{i=1}^{n} \Phi_1(\overline{x}_i) \rightarrow \mathcal{K}_1(f(\overline{x}_1), \ldots, f(\overline{x}_n))$ and $(\bigwedge_{i=1}^{m} \Phi_2(\overline{x}_i) \rightarrow \mathcal{K}_2(f(\overline{x}_1), \ldots, f(\overline{x}_m))$, then ($\mathsf{Comp}_{\mathsf{w}}^{\Psi}$) holds for the extension of for $\mathcal{T}$ with $\mathcal{K}$, where $\Psi(T) = T \cup \{f(t) | t \in T_0\}$ and

$$\begin{aligned} \mathcal{K} = \quad & (\textstyle\bigwedge_{i=1}^{n} \Phi_1(\overline{x}_i)) \rightarrow \mathcal{K}_1(f(\overline{x}_1), \ldots, f(\overline{x}_n)) \\ \cup \; & (\textstyle\bigwedge_{i=1}^{m} \Phi_2(\overline{x}_i)) \rightarrow \mathcal{K}_2(f(\overline{x}_1), \ldots, f(\overline{x}_m)). \end{aligned}$$

Thm. 10 can easily be extended to a combination of arbitrarily many pieces, where $O = \{\overline{x} \in M | \Phi_i(\overline{x}) \wedge \Phi_j(\overline{x})$, for some $i \neq j\}$.

9 Related Work

Local theory extensions are one of the few approaches that allow us to obtain decision procedures for quantified satisfiability problems modulo a background theory. They have proved useful in the verification of parametrized systems [9] and properties of data structures [5,15], and in reasoning about certain properties of numerical functions [14] and functions in ordered domains [16]. Algebraic data types with size functions are shown to be decidable in [19]. Further results on handling quantified formulas in a complete way include decidable fragments

of the theory of arrays [1] and of pointer data structures [11, 18], as well as certain classes of formulas that can be decided with an appropriate instantiation strategy [3]. An overview of results on local theory extensions can be found in [7].

Note that each locality result needs to be proved separately for each class of axioms of interest. A contribution of our paper is to identify new classes of local theories, and to show that they are useful in verification of new classes of properties of functional programs.

Acknowledgements. We thank Viorica Sofronie-Stokkermans for detailed comments on drafts of this paper, as well as the original suggestion to generalize the monotonicity axiom from [16] to the version in [7]. We thank Philippe Suter for discussions related to Section 4.

References

1. Bradley, A.R., Manna, Z., Sipma, H.B.: What's decidable about arrays? In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 427–442. Springer, Heidelberg (2005)
2. Cohen, E., Dahlweid, M., Hillebrand, M., Leinenbach, D., Moskal, M., Santen, T., Schulte, W., Tobies, S.: VCC: A practical system for verifying concurrent C. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 23–42. Springer, Heidelberg (2009)
3. Ge, Y., de Moura, L.: Complete instantiation for quantified SMT formulas. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 306–320. Springer, Heidelberg (2009)
4. Gligoric, M., Gvero, T., Jagannath, V., Khurshid, S., Kuncak, V., Marinov, D.: Test generation through programming in UDITA. In: International Conference on Software Engineering, ICSE (2010)
5. Ihlemann, C., Jacobs, S., Sofronie-Stokkermans, V.: On local reasoning in verification. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 265–281. Springer, Heidelberg (2008)
6. Jacobs, S.: Incremental instance generation in local reasoning. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 368–382. Springer, Heidelberg (2009)
7. Jacobs, S.: Hierarchic Decision Procedures for Verification. PhD thesis, Saarland University, Germany (2010)
8. Jacobs, S., Kuncak, V.: On complete reasoning about axiomatic specifications. Technical Report EPFL-REPORT-151486, EPFL (2010)
9. Jacobs, S., Sofronie-Stokkermans, V.: Applications of hierarchical reasoning in the verification of complex systems. Electronic Notes in Theoretical Computer Science 174(8), 39–54 (2007)
10. Lam, P., Kuncak, V., Rinard, M.: Generalized typestate checking for data structure consistency. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 430–447. Springer, Heidelberg (2005)
11. McPeak, S., Necula, G.C.: Data structure specifications via local equality axioms. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 476–490. Springer, Heidelberg (2005)
12. Podelski, A., Wies, T.: Counterexample-guided focus. In: 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (2010)

13. Sofronie-Stokkermans, V.: Hierarchic reasoning in local theory extensions. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 219–234. Springer, Heidelberg (2005)
14. Sofronie-Stokkermans, V.: Efficient hierarchical reasoning about functions over numerical domains. In: Dengel, A.R., Berns, K., Breuel, T.M., Bomarius, F., Roth-Berghofer, T.R. (eds.) KI 2008. LNCS (LNAI), vol. 5243, pp. 135–143. Springer, Heidelberg (2008)
15. Sofronie-Stokkermans, V.: Locality results for certain extensions of theories with bridging functions. In: Schmidt, R.A. (ed.) CADE-22. LNCS, vol. 5663, pp. 67–83. Springer, Heidelberg (2009)
16. Sofronie-Stokkermans, V., Ihlemann, C.: Automated reasoning in some local extensions of ordered structures. Journal of Multiple-Valued Logic and Soft Computing 13(4-6), 397–414 (2007)
17. Suter, P., Dotta, M., Kuncak, V.: Decision procedures for algebraic data types with abstractions. In: 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (2010)
18. Wies, T., Kuncak, V., Lam, P., Podelski, A., Rinard, M.: Field constraint analysis. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 157–173. Springer, Heidelberg (2005)
19. Zhang, T., Sipma, H.B., Manna, Z.: Decision procedures for recursive data structures with integer constraints. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 152–167. Springer, Heidelberg (2004)