Kalpa Publications in Computing

Volume 10, 2019, Pages 1–8

Selected Student Contributions and
Workshop Papers of LuxLogAI 2018

On Inductive Verification and Synthesis

Dennis Peuter[1] and Viorica Sofronie-Stokkermans[1]

University Koblenz-Landau, Koblenz, Germany
dpeuter@uni-koblenz.de, sofronie@uni-koblenz.de

Abstract

We study possibilities of using symbol elimination in program verification and synthesis. We consider programs for which a property is given, which is supposed to hold for all states reachable from the initial states. If it can not be proven that such a formula is an inductive invariant, the task is to find conditions to strengthen the property in order to make it an inductive invariant. We propose a method for property-directed invariant generation and analyze its properties.

1 Introduction

In the verification of parametric systems one has to show that for all states reachable from the initial state a certain property holds. One way to solve such problems is to identify an inductive invariant which entails the property to be proved. Finding suitable inductive invariants is non-trivial – the problem is undecidable in general; solutions have been proposed for specific cases e.g. in [4, 1, 3, 8, 9]. In this extended abstract we continue our work on automated verification and synthesis in parametric systems [6, 11, 12]. We present an incremental method which starts with a given property and successively strengthens it to obtain an inductive invariant. The theories of data types and of updates in the systems we analyze are complex (typically extensions or combinations of theories) and not required to have the finite model property – which is required for instance in [8, 9]. While we rely on methods similar up to a certain extent with the ones used in IC3 [2], we here exploit efficient methods for symbol elimination in theory extensions established in [13] which allow to devise a correct algorithm for finding invariants in a certain language. We identify situations in which the method we propose terminates.

Our main results can be summarized as follows:

- We present a method for invariant synthesis which uses the symbol elimination method described in [13].
- We identify a class of transition systems and theories having infinite models for which our method for invariant generation is correct.
- We analyze situations in which the invariant generation method terminates.
- We illustrate the main steps of the method on an example.

C. Benzmüller, X. Parent and A. Steen (eds.), LuxLogAI 2018 (Kalpa Publications in Computing, vol. 10), pp. 1–8

Structure of the paper. The paper is structured as follows. In Section 1.1 we illustrate the ideas on an example. In Section 2 we present earlier results on efficient reasoning in local theory extensions and discuss methods for quantifier elimination and symbol elimination in local theory extensions. In Section 3 we present an approach to invariant synthesis.

1.1 Illustration

Consider the program in Fig. 1 (a variation of an example from [1]). The task is to prove that if a is an array with its elements increasingly sorted, then the formula $\Psi := d_2 \geq a[d_1+1]$ is an invariant of the program. Ψ holds in the initial state since $d_2 = a[4] = a[3+1] = a[d_1+1]$. In order to show that Ψ is an inductive invariant of the while loop, we would need to prove that

$$d_2 \geq a[d_1+1] \wedge \mathsf{Sorted}(a) \wedge \mathsf{Update}(\overline{d}, \overline{d}') \wedge d_2' < a[d_1'+1]$$

is unsatisfiable, where the updates of the variables d_1, d_2, d_3 in the while loop are described by a formula $\mathsf{Update}(\overline{d}, \overline{d}') := d_1' = a[d_1{+}1] \ \wedge \ d_2' = a[d_2{+}1]{+}(1{-}d_3) \ \wedge \ d_3' = d_3/2$ and the fact that a is sorted is described by $\mathsf{Sorted}(a) := \forall i, j (i \leq j \rightarrow a[i] \leq a[j])$. The formula above is satisfiable, so Ψ is not an inductive invariant. We will show how to synthesize the condition $d_3 \leq 1$ which can be used to strengthen Ψ to an inductive invariant.

```
d1 = 3; d2 = a[4]; d3 = 1;
while (nondet()) {
    d1 = a[d1+1];
    d2 = a[d2+1] + (1-d3);
    d3 = d3/2;   }
```

Figure 1: A simple program

2 Preliminaries

We assume known standard definitions from first-order logic (e.g. Π-structures, satisfiability, unsatisfiability). We consider signatures $\Pi = (\Sigma, \mathsf{Pred})$, where Σ is a family of function symbols and Pred a family of predicate symbols. In this paper, (logical) theories are simply sets of sentences. We denote "falsum" with $\bot$. If F and G are formulae we write $F \models G$ (resp. $F \models_{\mathcal{T}} G$ – also written as $\mathcal{T} \cup F \models G$) to express the fact that every model of F (resp. every model of F which is also a model of $\mathcal{T}$) is a model of G. $F \models \bot$ means that F is unsatisfiable; $F \models_{\mathcal{T}} \bot$ means that there is no model of $\mathcal{T}$ in which F is true.

2.1 Verification problems for parametric systems

One of the application domains we consider is related to the verification of parametric systems, for instance of programs (resp. parametric programs, in which some parts are not or only partially specified). For modeling (parametric) systems we use transition constraint systems $T = (V, \Sigma, \mathsf{Init}, \mathsf{Update})$ which specify: the variables (V) and function symbols (Σ) whose values change over time; a formula Init specifying the properties of initial states; a formula Update with variables in $V \cup V'$ and function symbols in $\Sigma \cup \Sigma'$ (where V' and Σ' are copies of V resp. Σ, denoting the variables resp. functions after the transition) which specifies the relationship between the values of variables v and function symbols f before a transition and their values (v', f') after the transition. Such descriptions can be obtained from system specifications (for an example cf. [5]). With every specification of a system S, a *background theory* $\mathcal{T}_S$ – describing the data types used in the specification and their properties – is associated.

We can check in two steps whether a formula Ψ is an inductive invariant of a transition constraint system $T{=}(V, \Sigma, \mathsf{Init}, \mathsf{Update})$: first, we prove that $\mathcal{T}_S, \mathsf{Init} \models \Psi$; and second, we prove that $\mathcal{T}_S, \Psi, \mathsf{Update} \models \Psi'$, where Ψ' results from Ψ by replacing each $v \in V$ by v' and each $f \in \Sigma$

by f'. Failure to prove the second step means that Ψ is not an invariant or Ψ is not inductive w.r.t. T. In this case we can consider two orthogonal problems:

(1) Determine constraints on parameters which guarantee that Ψ is an invariant.

(2) Determine a formula I such that $\mathcal{T}_S \models I \rightarrow \Psi$ and I is an inductive invariant.

Checking whether a formula Ψ is an invariant can be reduced to checking whether $\neg\Psi'$ is satisfiable w.r.t. a theory $\mathcal{T}$. Even if Ψ is a universally quantified formula (and thus $\neg\Psi'$ is a ground formula) the theory $\mathcal{T}$ is quite complex: it contains the axiomatization $\mathcal{T}_S$ of the datatypes used in the specification of the system, the formalization of the update rules, as well as the formula Ψ itself. In [6, 11, 12] we show that in many cases the theory $\mathcal{T}$ can be expressed using a chain of extensions, typically including $\mathcal{T}_0 \subseteq \mathcal{T}_1 = \mathcal{T}_0 \cup \Psi \subseteq \mathcal{T} = \mathcal{T}_0 \cup \Psi \cup \mathsf{Update}$, with the property that checking satisfiability of ground formulae w.r.t. $\mathcal{T}$ can be reduced to checking satisfiability w.r.t. $\mathcal{T}_1$ and ultimately to checking satisfiability w.r.t. $\mathcal{T}_0$. This is the case for instance when the theory extensions in the chain above are *local*. We present results on local theory extensions established so far in Section 2.2. These results yield efficient methods for checking whether a formula is an invariant; the method for symbol elimination in local theory extension described in Section 2.3 allow us to also address the two synthesis problems mentioned above.

2.2 Local Theory Extensions

Let $\Pi_0{=}(\Sigma_0, \mathsf{Pred})$ be a signature, and $\mathcal{T}_0$ be a "base" theory with signature Π_0. We consider extensions $\mathcal{T} := \mathcal{T}_0 \cup \mathcal{K}$ of $\mathcal{T}_0$ with new function symbols Σ (*extension functions*) whose properties are axiomatized using a set $\mathcal{K}$ of clauses in the extended signature $\Pi = (\Sigma_0 \cup \Sigma, \mathsf{Pred})$, which contain function symbols in Σ. If G is a finite set of ground Π^C-clauses[1] and $\mathcal{K}$ a set of Π-clauses, we will denote by $\mathsf{st}(\mathcal{K}, G)$ (resp. $\mathsf{est}(\mathcal{K}, G)$) the set of all ground terms (resp. extension ground terms, i.e. terms starting with a function in Σ) which occur in G or $\mathcal{K}$.[2] If T is a set of ground terms in the signature Π^C, we denote by $\mathcal{K}[T]$ the set of all instances of $\mathcal{K}$ in which the terms starting with a function symbol in Σ are in T. We define:

(Loc_f) For every finite set G of ground clauses in Π^C it holds that $\mathcal{T}_0 \cup \mathcal{K} \cup G \models \bot$ if and only if $\mathcal{T}_0 \cup \mathcal{K}[(G)] \cup G$ is unsatisfiable.

Extensions satisfying condition (Loc_f) are called *local theory extensions* [10]. Local extensions can be recognized by showing that certain partial models embed into total ones [7]. This allowed us to identify many classes of local theory extensions. Especially well-behaved are the theory extensions which have the property (Comp_f), stating that partial models can be made total without changing the universe of the model.

Hierarchical Reasoning in Local Theory Extensions For local theory extensions hierarchical reasoning is possible. If $\mathcal{T}_0 \cup \mathcal{K}$ is a local extension of $\mathcal{T}_0$ and G is a set of ground $\Sigma_0 \cup \Sigma_1 \cup \Sigma_c$-clauses, then $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is unsatisfiable iff $\mathcal{T}_0 \cup \mathcal{K}[G] \cup G$ is unsatisfiable. We can reduce this satisfiability test to a satisfiability test w.r.t. $\mathcal{T}_0$. The idea is to purify $\mathcal{K}[G] \cup G$ by: introducing (bottom-up) new constants c_t for subterms $t = f(g_1, \ldots, g_n)$ with $f \in \Sigma$, g_i ground $\Sigma_0 \cup \Sigma_c$-terms; replacing the terms t with the constants c_t; and adding the definitions $c_t = t$ to

[1] Π^C is the extension of Π with constants in a countable set C of fresh constants.

[2] We here regard every finite set G of ground clauses as the ground formula $\bigwedge_{C \in G} C$.

Algorithm 1: Symbol elimination in theory extensions

Step 1 Let $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ be the set of Π_0^C clauses obtained from $\mathcal{K}[T] \cup G$ after the purification step described in Theorem 1 (with set of extension symbols Σ_1).

Step 2 Let $G_1 = \mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$. Among the constants in G_1, we identify

(i) the constants $\overline{c}_f$, $f \in \Sigma_P$, where either $c_f = f \in \Sigma_P$ is a constant parameter, or c_f is introduced by a definition $c_f := f(c_1, \ldots, c_k)$ in the hierarchical reasoning method,

(ii) all constants $\overline{c}_p$ occurring as arguments of functions in Σ_P in such definitions.

Let $\overline{c}$ be the remaining constants. We replace the constants in $\overline{c}$ with existentially quantified variables $\overline{x}$ in G_1, i.e. replace $G_1(\overline{c}_p, \overline{c}_f, \overline{c})$ with $G_1(\overline{c}_p, \overline{c}_f, \overline{x})$, and consider the formula $\exists \overline{x} G_1(\overline{c}_p, \overline{c}_f, \overline{x})$.

Step 3 Using the method for quantifier elimination in $\mathcal{T}_0$ (if Condition (C1) holds) or in $\mathcal{T}_0^*$ (if Condition (C2) holds) we can construct a formula $\Gamma_1(\overline{c}_p, \overline{c}_f)$ equivalent to $\exists \overline{x} G_1(\overline{c}_p, \overline{c}_f, \overline{x})$ w.r.t. $\mathcal{T}_0$ (resp. $\mathcal{T}_0^*$).

Step 4 Let $\Gamma_2(\overline{c}_p)$ be the formula obtained by replacing back in $\Gamma_1(\overline{c}_p, \overline{c}_f)$ the constants c_f introduced by definitions $c_f := f(c_1, \ldots, c_k)$ with the terms $f(c_1, \ldots, c_k)$. We replace $\overline{c}_p$ with existentially quantified variables $\overline{y}$.

Step 5 Let $\forall \overline{y} \Gamma_T(\overline{y})$ be $\forall \overline{y} \neg \Gamma_2(\overline{y})$.

a set D. We denote by $\mathcal{K}_0 \cup G_0 \cup D$ the set of formulae obtained this way. Then G is satisfiable w.r.t. $\mathcal{T}_0 \cup \mathcal{K}$ iff $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ is satisfiable w.r.t. $\mathcal{T}_0$, where

$$\mathsf{Con}_0 = \{(\bigwedge_{i=1}^{n} c_i{=}d_i) \rightarrow c{=}d \mid f(c_1, \ldots, c_n){=}c, f(d_1, \ldots, d_n){=}d \in D\}.$$

Theorem 1 ([10]). *If $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ is a local extension and G is a set of ground clauses, then we can reduce the problem of checking whether G is satisfiable w.r.t. $\mathcal{T}_0 \cup \mathcal{K}$ to checking the satisfiability w.r.t. $\mathcal{T}_0$ of the formula $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ constructed as explained above.*
If $\mathcal{K}_0 \cup G_0 \cup \mathsf{Con}_0$ belongs to a decidable fragment of $\mathcal{T}_0$, we can use the decision procedure for this fragment to decide whether $\mathcal{T}_0 \cup \mathcal{K} \cup G$ is unsatisfiable.

2.3 Quantifier elimination and symbol elimination

A theory $\mathcal{T}$ over signature Π *allows quantifier elimination* if for every formula ϕ over Π there exists a quantifier-free formula ϕ^* over Π which is equivalent to ϕ modulo $\mathcal{T}$. Examples of theories which allow quantifier elimination are rational and real linear arithmetic ($\mathsf{LI}(\mathbb{Q})$, $\mathsf{LI}(\mathbb{R})$), the theory of real closed fields, and the theory of absolutely-free data structures.

In [13] we proved that in theory extensions $\mathcal{T}_0 \subseteq \mathcal{T} = \mathcal{T}_0 \cup \mathcal{K}$ for which $\mathcal{T}_0$ allows quantifier elimination, for every ground formula G containing function symbols considered to be "parameters" we can generate a (universal) constraint Γ on the parameters of G such that $\mathcal{T} \cup \Gamma \cup G \models \bot$.

Let $\Pi_0 = (\Sigma_0, \mathsf{Pred})$. Let $\mathcal{T}_0$ be a Π_0-theory and Σ_P be a set of parameters (function and constant symbols). Let Σ be a signature such that $\Sigma \cap (\Sigma_0 \cup \Sigma_P) = \emptyset$. Let $\mathcal{K}$ be a set of clauses in the signature $\Pi = \Pi_0 \cup \Sigma_P \cup \Sigma$ in which all variables occur also below functions in $\Sigma_1 = \Sigma_P \cup \Sigma$. Let G be a finite set of ground Π^C-clauses, and T a finite set of ground terms over the signature $\Pi_0 \cup \Sigma_P \cup \Sigma \cup C$, where C is a set of additional constants. We construct a universal $\Pi_0 \cup \Sigma_P$-formula $\forall y_1 \ldots y_n \Gamma_T(y_1, \ldots, y_n)$ by following Steps 1–5 described in Algorithm 1.

Theorem 2 ([13]). *Assume that $\mathcal{T}_0$ allows quantifier elimination. For every finite set of ground Π^C-clauses G, and every finite set T of terms over the signature $\Pi_0 \cup \Sigma \cup \Sigma_P \cup C$ with $\mathsf{est}(G) \subseteq T$, steps 1–5 of Algorithm 1 yield a universally quantified $\Pi_0 \cup \Sigma_P$-formula $\forall \overline{x} \Gamma_T(\overline{x})$ with the*

properties that (1) for every structure $\mathcal{A}$ with signature $\Pi_0 \cup \Sigma \cup \Sigma_P \cup C$ which is a model of $\mathcal{T}_0 \cup \mathcal{K}$, if $\mathcal{A} \models \forall \overline{y} \Gamma_T(\overline{y})$ then $\mathcal{A} \models \neg G$; and (2) $\mathcal{T}_0 \cup \forall \overline{y} \Gamma_T(\overline{y}) \cup \mathcal{K} \cup G$ is unsatisfiable.

Theorem 3 ([13]). *Assume that the extension $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}$ satisfies condition (Comp_f) and $\mathcal{K}$ is flat and linear* [3]*. Let G be a set of ground Π^C-clauses, and $\forall \overline{y} \Gamma_G(\overline{y})$ be the formula obtained with Algorithm 1 for $T = \mathsf{est}(\mathcal{K}, G)$. Then $\forall y \Gamma_G(y)$ is entailed by every universal formula Γ with $\mathcal{T}_0 \cup \Gamma \cup \mathcal{K} \cup G \models \bot$.*

A similar result holds if T is the set of instances obtained from the instantiation of a chain of theory extensions $\mathcal{T}_0 \subseteq \mathcal{T}_0 \cup \mathcal{K}_1 \subseteq \cdots \subseteq \mathcal{T}_0 \cup \mathcal{K}_1 \cup \cdots \cup \mathcal{K}_n$, all satisfying condition (Comp_f), and where $\mathcal{K}_1, \ldots, \mathcal{K}_n$ are all flat and linear.

3 Invariant generation

We now study the problem of inferring – in a goal-oriented way – universally quantified inductive invariants for transition systems described by axioms which define local theory extensions. We focus on invariant generation here, because the problem of invariant checking and constraint synthesis for parametric systems (finding constraints on parameters to ensure that a given formula is an inductive invariant) was already investigated in [6] and [11].

Let S be a system, $\mathcal{T}_S$ be the theory and T=$(V, \Sigma, \mathsf{Init}, \mathsf{Update})$ the transition constraint system associated with S. Let $\mathsf{LocSafe}$ be a class of universal formulae over $V \cup \Sigma$. We make the following assumptions:

(A1) There exists a chain of local theory extensions $\mathcal{T}_0 \subseteq \cdots \subseteq \mathcal{T}_S \cup \mathsf{Init}$ such that in each extension all variables occur below an extension function.

(A2) For every $\Psi \in \mathsf{LocSafe}$ there exists a chain of local theory extensions $\mathcal{T}_0 \subseteq \cdots \subseteq \mathcal{T}_S \cup \Psi$ such that in each extension all variables occur below an extension function.

(A3) The update axioms describe the change of the Σ-functions depending on a finite set $\{\phi_i \mid i \in I\}$ of mutually exclusive conditions over non-primed symbols, i.e. $\mathsf{Update}(\Sigma, \Sigma') = \bigcup_{f \in \Sigma} \mathsf{Update}_f$, where Update_f has the form $\mathsf{Def}_f := \{\forall \overline{x}(\phi_i^f(\overline{x}) \rightarrow F_i^f(f'(\overline{x}), \overline{x})) \mid i \in I\}$, using Σ_0-formulae F_i^f such that (i) $\phi_i(\overline{x}) \wedge \phi_j(\overline{x}) \models_{\mathcal{T}_0} \bot$ for $i \neq j$ and (ii) $\mathcal{T}_0 \models \forall \overline{x}(\phi_i(\overline{x}) \rightarrow \exists y(F_i(y, \overline{x})))$ for all $i \in I$.

(A4) Ground satisfiability in $\mathcal{T}_0$ is decidable; $\mathcal{T}_0$ allows quantifier elimination.

(A5) All candidate invariants I computed in the while loop in Fig. 2 are in $\mathsf{LocSafe}$, and all local extensions in $\mathsf{LocSafe}$ satisfy condition ($\mathsf{Comp_f}$).

The algorithm we propose is shown in Fig. 2. We can prove partial correctness of the algorithm (Theorem 4) and identify a condition under which it terminates (Theorems 5).

Theorem 4 (Partial Correctness). *Under Assumptions* **(A1)**–**(A5)**, *if the algorithm in Fig. 2 terminates, then its output is correct.*

[3] An extension clause D is *flat* when all symbols below an extension function symbol in D are variables. D is *linear* if whenever a variable occurs in two terms of D starting with an extension functions, the terms are equal, and no term contains two occurrences of a variable.

Input:	T transition system; signature Σ_P; $\Psi \in \mathsf{LocSafe}$, formula over Σ_P
Output:	Inductive invariant I of T that entails Ψ and contains only function symbols in Σ_P (if such an invariant exists).

1: $I := \Psi$
2: **while** I is not an inductive invariant for T **do:**
if $\mathsf{Init} \not\models I$ **then return** "no universal inductive invariant entails Ψ"
if I is not preserved under $\mathsf{Update}(\Sigma, \Sigma')$ **then** Let Γ be obtained by eliminating all primed variables and symbols not in Σ_P from $I \wedge \mathsf{Update}(\Sigma, \Sigma') \wedge \neg I'$;
$I := I \wedge \Gamma$
3: **return** I is an inductive invariant

Figure 2: Successively strengthening a formula to obtain an inductive invariant

Theorem 5 (A termination condition)**.** *Assume that the candidate invariants I generated at each iteration are conjunctions of clauses which contain, up to renaming of the variables, terms in a given, finite, family* Ter *of terms. Then the algorithm must terminate with an invariant I or after detecting that* $\mathsf{Init} \not\models I$.

Example 1. *Consider Example A in Section 1.1. In order to prove that $\Psi := d_2 \geq a[d_1 + 1]$ is an inductive invariant of the program, we need to prove that the formula $\mathsf{Sorted}(a) \wedge G$, where $G = d_2 \geq a[d_1 + 1] \wedge d_1' \approx a[d_1 + 1] \wedge d_3' \approx d_3/2 \wedge d_2' \approx a[d_2 + 1] + (1 - d_3) \wedge d_2' < a[d_1' + 1]$ and $\mathsf{Sorted}(a) := \forall i, j(i \leq j \rightarrow a[i] \leq a[j])$, is unsatisfiable.*

The updates change only constants and Ψ is a ground formula. If $\mathcal{T} = \mathbb{Z} \cup \mathsf{Sorted}(a)$, then $\mathcal{T} \wedge G$ is satisfiable iff the formula $\exists d_1 \exists d_2 \exists d_3 \exists d_1' \exists d_2' \exists d_3' G$ is valid w.r.t. $\mathcal{T}$. The quantified variables d_1', d_2' and d_3' can be eliminated, the problem is thus reduced to checking the satisfiability of $\mathsf{Sorted}(a) \wedge d_2 \geq a[d_1 + 1] \wedge a[d_2 + 1] + (1 - d_3) < a[a[d_1 + 1] + 1$. The axiom $\mathsf{Sorted}(a)$ defines a local theory extension $\mathbb{Z} \subseteq \mathbb{Z} \cup \mathsf{Sorted}(a) = \mathcal{T}$; after flattening of the ground part and instantiation of $\mathsf{Sorted}(a)$ we obtain:

$$
\begin{array}{ll}
G: & c_1 \approx a[d_1 + 1] \wedge d_2 \geq a[d_1 + 1] \wedge a[d_2 + 1] + (1 - d_3) < a[c_1 + 1] \\
\mathsf{Sorted}(a)[G]: & d_1 + 1 \rhd d_2 + 1 \rightarrow a[d_1 + 1] \rhd a[d_2 + 1] \\
& d_1 + 1 \rhd c_1 + 1 \rightarrow a[d_1 + 1] \rhd a[c_1 + 1] \\
& d_2 + 1 \rhd c_1 + 1 \rightarrow a[d_2 + 1] \rhd a[c_1 + 1], \rhd \in \{\leq, \geq\}
\end{array}
$$

After purification in which the definitions $\mathsf{Def} := \{c_1 \approx a[d_1 + 1], c_2 \approx a[d_2 + 1], c_3 \approx a[c_1 + 1]\}$ are introduced and further simplification we obtain:

$$
\begin{array}{ll}
G_0: & d_2 \geq c_1 \wedge c_2 + (1 - d_3) < c_3 \\
\mathsf{Sorted}(a)[G]_0: & d_1 \rhd d_2 \rightarrow c_1 \rhd c_2 \wedge d_1 \rhd c_1 \rightarrow c_1 \rhd c_3 \wedge d_2 \rhd c_1 \rightarrow c_2 \rhd c_3, \quad \rhd \in \{\leq, \geq\}
\end{array}
$$

$G_0 \wedge \mathsf{Sorted}(a)[G]_0$ is satisfiable; the formula obtained by negating it and universally quantifying the constants can be used to strengthen Ψ. If we are looking for a universal invariant in a more restricted language (e.g. containing only d_1, d_2 and d_3), we can eliminate c_1, c_2 and negate the result to obtain $d_3 \leq 1$. The formula $d_2 \geq a[d_1 + 1] \wedge d_3 \leq 1$ can be proved to be a loop invariant.

4 Conclusion

In this extended abstract we proposed a method for property-directed invariant generation and analyzed its properties. Our results can be seen as extensions of the results in [4] and [2], since we consider more complex theories. Although there are similarities to the method in [9], our approach is different: The theories we analyze do not typically have the finite model

property – which is required in [8, 9], and we do not use diagrams associated to finite models for strengthening the invariants. While our method is not guaranteed to terminate in general, we identified situations in which termination is guaranteed.

In future work we would like, on the one hand to identify additional situations in which our invariant generation method is correct and terminates, and on the other hand to use similar ideas for the goal-oriented generation of inductive properties for recursively defined functions.

References

[1] Dirk Beyer, Thomas A. Henzinger, Rupak Majumdar, and Andrey Rybalchenko. Invariant synthesis for combined theories. In Byron Cook and Andreas Podelski, editors, *Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings*, volume 4349 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2007.

[2] Aaron R. Bradley. IC3 and beyond: Incremental, inductive verification. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, page 4. Springer, 2012.

[3] Aaron R. Bradley and Zohar Manna. Property-directed incremental invariant generation. *Formal Asp. Comput.*, 20(4-5):379–405, 2008.

[4] Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. Inductive invariant generation via abductive inference. In Antony L. Hosking, Patrick Th. Eugster, and Cristina V. Lopes, editors, *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*, pages 443–456. ACM, 2013.

[5] Johannes Faber, Swen Jacobs, and Viorica Sofronie-Stokkermans. Verifying CSP-OZ-DC specifications with complex data types and timing parameters. In Jim Davies and Jeremy Gibbons, editors, *Integrated Formal Methods, 6th International Conference, IFM 2007, Oxford, UK, July 2-5, 2007, Proceedings*, volume 4591, pages 233–252. Springer, 2007.

[6] Carsten Ihlemann, Swen Jacobs, and Viorica Sofronie-Stokkermans. On local reasoning in verification. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2008.

[7] Carsten Ihlemann and Viorica Sofronie-Stokkermans. On hierarchical reasoning in combinations of theories. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2010.

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

[9] Oded Padon, Neil Immerman, Sharon Shoham, Aleksandr Karbyshev, and Mooly Sagiv. Decidability of inferring inductive invariants. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 217–231. ACM, 2016.

[10] Viorica Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In Robert Nieuwenhuis, editor, *Automated Deduction - CADE-20, 20th International Conference on Automated De-*

duction, Tallinn, Estonia, July 22-27, 2005, Proceedings, volume 3632 of *Lecture Notes in Computer Science*, pages 219–234. Springer, 2005.

[11] Viorica Sofronie-Stokkermans. Hierarchical reasoning for the verification of parametric systems. In Jürgen Giesl and Reiner Hähnle, editors, *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 171–187. Springer, 2010.

[12] Viorica Sofronie-Stokkermans. Hierarchical reasoning and model generation for the verification of parametric hybrid systems. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 360–376. Springer, 2013.

[13] Viorica Sofronie-Stokkermans. On interpolation and symbol elimination in theory extensions. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2016.