Proving Infinite Satisfiability

Peter Baumgartner and Joshua Bax

NICTA[⋆] and Australian National University, Canberra, Australia
Firstname.Lastname@nicta.com.au

Abstract. We consider the problem of automatically disproving invalid conjectures over data structures such as lists and arrays over integers, in the presence of additional hypotheses over these data structures. We investigate a simple approach based on refutational theorem proving. We assume that the data structure axioms are satisfiable and provide a template language for additional hypotheses such that satisfiability is preserved. Then disproving is done by proving that the negated conjecture follows. By means of examples we demonstrate that our template language is reasonably expressive and that our approach works well with current theorem provers (Z3, SPASS+T and Beagle).

1 Introduction

We consider the problem of automatically disproving invalid conjectures over data structures such as lists and arrays over integers, in the presence of additional hypotheses over these data structures. Such invalid conjectures come up frequently in applications of automated reasoning to software verification and the analysis of data-rich state-based systems, for example. More formally, the disproving problem is to show that $\mathsf{AX} \cup \mathsf{HYP}$ does not entail a sentence CON, where AX are list and/or array axioms and CON is the conjecture in question. The obvious approach to disproving is to show satisfiability of $\mathsf{AX} \cup \mathsf{HYP} \cup \{\neg\mathsf{CON}\}$ by means of a (complete) theorem prover. Unfortunately, current theorem proving technology is of limited usefulness for that: finite model finders cannot be used because the list axioms do not admit finite models, SMT-solvers are typically incomplete on quantified formulas and face the same problem, and theorem provers based on saturation often do not terminate on satisfiable input (let alone completeness issues in presence of arithmetic background theories).

We propose a different, conceptually simple approach based on refutational theorem proving. It consists in assuming that AX is satisfiable and providing templates for HYP that are guaranteed to preserve satisfiability of $\mathsf{AX} \cup \mathsf{HYP}$. Then disproving is attempted simply by proving that $\mathsf{AX} \cup \mathsf{HYP}$ entails $\neg\mathsf{CON}$, i.e., that $\mathsf{AX} \cup \mathsf{HYP} \cup \{\mathsf{CON}\}$ is unsatisfiable.

The main point of this paper is to demonstrate the practical feasibility of our approach. By means of examples, we demonstrate that our template language covers useful cases. We also report on our experiences disproving sample conjectures using current theorem provers (Z3 [11], SPASS+T [18] and Beagle [3]), and we compare their performance.

⋆ NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

Related Work. Kapur and Zarba [8] show by way of reductions to sub-theories how to decide the satisfiability of conjunctions of ground literals wrt. various theories, including arrays and lists. Armando, Bonacina, Ranise and Schulz [2] use the superposition calculus as a decision procedure, again for conjunctions of ground literals wrt. these (and other) theories. In a similar way, Lynch and Morawska [9] aim at superposition as decision procedure based on finite saturation. Ihlemann, Jacobs and Sofronie-Stokkermans [7] develop decidability results for the theory of arrays and others using the framework of local theory extensions. DeMoura and Bjoerner [12] give decidability results for a theory extending the basic theory of arrays. McPeak and Necula [10] provide decision procedures for pointer data structures. Bradley, Manna and Sipma [4] give a decidability result for an expressive fragment of the theory of arrays, the *array property* fragment. Certain desirable formulas are not included in this fragment, for example totality axioms for functions or an injectivity predicate for arrays (see distinct in Section 4). Ghilardi, Nicolini, Ranise and Zucchelli [6] provide a decision procedure for an extension of the array theory and demonstrate how decision procedures may be derived for extensions to this theory, many of which lie outside the array property fragment. This relies on the existence of a "standard model" for the theory and extension, whose existence must be demonstrated a priori.
In contrast to these works, we do not provide decision procedures for specific fragments. This is intentionally so, in order to support disproving tasks in the presence of liberally formulated additional axioms (the set HYP above). Although we employ superposition-based provers in our experiments (like some of the approaches above), our approach does not hinge on finite saturation. Claessen and Lillieström [5] present a method for showing that a set of formulas does not admit finite models. It does not answer the question whether infinite models exists, and this way our work is complementary to theirs. Suter, Köksal and Kuncak [17] have developed a semi-decision procedure for checking satisfiability of correctness properties of recursive functional programs on algebraic data types. It overlaps with out method on lists (Section 3) by imposing similar syntactic restrictions. Their method works differently, by partial unrolling of function definitions into quantifier-free logic instead of theorem proving on (quantified) formulas. In [15], Rümmer and Shah use a program logic for Java to prove the incorrectness of programs. It utilizes a sequent calculus for unfolding lists and reasoning with arithmetic constraints, and this way is somewhat more spcialised than our approach.

Preliminaries. We work in the context of many-sorted logic with first-order signatures comprised of sorts and operator symbols (i.e., function symbols and predicate symbols) of given arities over these sorts. In this paper we focus on theorem proving modulo the fixed background theory of (linear) integer arithmetic. Our signatures Σ are comprised of sort symbols $s_1, \ldots, s_n$ where $s_n = \mathbb{Z}$, the integer sort. Let $sorts(\Sigma) = \{s_1, \ldots, s_n\}$. We assume Σ contains an equality symbol $\approx_{s_i}$ for each sort s_i. We usually drop the sort annotion from $\approx_{s_i}$. We also assume infinite supplies of variables of each sort. When x is a variable and s is a sort we write x_s to make clear that the sort of x is s.

We use the notions commonly used in automated theorem proving in a standard way. The (well-sorted Σ-) terms, atoms, and formulas are defined as usual. Let $x_1, \ldots, x_n$ be pairwise different variables of corresponding sorts $s_1, \ldots, s_n$. We write $F[x_1, \ldots, x_n]$ to indicate that the formula F has free variables at most $x_1, \ldots, x_n$, and we say that F has the *arity* $s_1 \times \cdots \times s_n$. We write $F[t_1, \ldots, t_n]$ for the formula obtained from $F[x_1, \ldots, x_n]$ by replacing every free occurrence of x_i in F by t_i, for all $1 \leq i \leq n$.

Our logical language is essentially the same as the TPTP-language TFA ("Typed Formulas with Arithmetic") and we adopt the semantics given for it in [16]. In brief, a *(Σ-)interpretation I* consists of a *(Σ-)domain* $D = D_{s_1} \uplus \ldots \uplus D_{s_n}$ with disjoint, non-empty sub-domains for each sort, and an arity-respecting mapping of function symbols to functions and predicate symbols to relations (representing the tuples of which the predicate holds true). We work with E-interpretations only. That is, $I(\approx_{s_i}) = \{(d, d) \mid d \in D_{s_i}\}$, where $I(op)$ is the interpretation of the operator op. Furthermore, we consider only interpretations that *extend arithmetic*, that is, (i) the domain $D_{\mathbb{Z}}$ of the integer sort $\mathbb{Z}$ is the set of all integer numbers and, (ii) the numeric operators such as $>$, $\geq$, $+$, $-$ and $\cdot$ are interpreted as expected. The usual notions of satisfaction, validity, model etc. apply in the standard way. In particular, when N is a set of sentences we write $I \models N$ to indicate that I is a model of (all elements of) N, and we say that N *entails* a formula F, written as $N \models F$ iff every model of N is a model of F.

2 Approach

Our approach consists in starting with a signature Σ and a set of Σ-sentences Ax that is known to be satisfiable. Our main interest is in lists and arrays, and so Ax will be corresponding axioms, see below. Then we stepwise expand Σ and Ax with new user-defined operators and additional definitions for these.

More formally, for two signatures Σ and Σ' over the same sorts we use set operators to relate the sets of their operators in the obvious way. For instance, we write $\Sigma' = \Sigma \cup \{op\}$ to indicate that Σ' is obtained from Σ by adding the operator op. We consider sequences $(Ax, \mathrm{Def}_{op_1}, \ldots, \mathrm{Def}_{op_n})$ such that Def_{op_i} is a set of Σ_i-sentences ("Definition for op_i") of a certain form explained below, where $\Sigma_0 = \Sigma$, $op_i \notin \Sigma_{i-1}$ and $\Sigma_i = \Sigma_{i-1} \cup \{op_i\}$ for all $1 \leq i \leq n$. We call any such sequence an *extension of Ax*.

Definition 2.1 (Admissible Definition). *Let Σ be a signature, D a Σ-domain, and $op \notin \Sigma$ an operator with an arity over sorts(Σ). We say that a set of $(\Sigma \cup \{op\})$-sentences N is an* admissible definition of op (wrt. Σ and D) *iff every Σ-interpretation I with domain D can be expanded to a $(\Sigma \cup \{op\})$-interpretation I' with domain D such that $I' \models N$.*

That is, I' differs from I only by adding an interpretation for op which satisfies N. We indicate this by writing $I' = I \cup I(op)$.

Proposition 2.2. *Let $(Ax, \mathrm{Def}_{op_1}, \ldots, \mathrm{Def}_{op_n})$ be an extension of Ax. Suppose there is a Σ_0-model $I \models Ax$ with domain D. If Def_{op_i} is an admissible definition of op_i wrt. Σ_{i-1} and D, for all $1 \leq i \leq n$, then there is a Σ_n-interpretation I' such that $I' \models Ax \cup \bigcup_{1 \leq i \leq n} \mathrm{Def}_{op_i}$.*

Proof. By induction over the length n of extensions, using the given model I in the induction start and using admissibility in the induction step. □

As said, in this paper we are mainly interested in disproving conjectures. With the current terminology, the problem is to show that $N = Ax \cup \bigcup_{1 \leq i \leq n} \mathrm{Def}_{op_i}$ does not entail a given Σ_n-sentence *Con*, the conjecture in question. Assuming admissible definitions, Proposition 2.2 gives us $I' \models N$, for some Σ_n-interpretation I'. Now, suppose we are able to *prove* (by a theorem prover) the entailment $N \models \neg Con$. It follows $I' \models \neg Con$, and so $I' \not\models Con$. By definition, then $N \not\models Con$, and so the conjecture is disproven.

Our intended application context is that of dynamically evolving systems. By this we mean computations that start in a (typically partially) specified initial state, modify some data until a final state is reached, and then the resulting (partially specified) final state is queried as to whether a property P holds in it. This leads to universally quantified implications *Con* in which the premise encodes both the initial state and computation, while the conclusion encodes property P.

A trivial example of this situation is the formula $Con = \forall\, l_{\mathsf{LIST}}\, l'_{\mathsf{LIST}} \,.\, l \not\approx \mathsf{nil} \wedge l' \approx \mathsf{tail}(l) \Rightarrow l' \not\approx \mathsf{nil}$. Here, $l \not\approx \mathsf{nil}$ is meant to represent the initial state; $l' \approx \mathsf{tail}(l)$ the computation; and $P = l' \not\approx \mathsf{nil}$. Where $\mathsf{Ax}_{\mathsf{LIST}}$ are the list axioms of Section 3 below, we wish to show $\mathsf{Ax}_{\mathsf{LIST}} \not\models Con$. With the approach indicated above, we have to prove $\mathsf{Ax}_{\mathsf{LIST}} \models \exists\, l_{\mathsf{LIST}}\, l'_{\mathsf{LIST}} \,.\, l \not\approx \mathsf{nil} \wedge l' \approx \mathsf{tail}(l)$ instead, which is a theorem proving task.

3 Lists

We consider lists over integers. To this end let the signature Σ_{LIST} consist of sorts LIST and $\mathbb{Z}$ and the operators $\mathsf{nil} : \mathsf{LIST}$, $\mathsf{cons} : \mathbb{Z} \times \mathsf{LIST} \mapsto \mathsf{LIST}$, $\mathsf{head} : \mathsf{LIST} \mapsto \mathbb{Z}$, $\mathsf{tail} : \mathsf{LIST} \mapsto \mathsf{LIST}$. The *list axioms* $\mathsf{AX}_{\mathsf{LIST}}$ are the following formulas, each implicitly universally quantified, where k is $\mathbb{Z}$-sorted and l is LIST-sorted:

$$\begin{array}{rr} \mathsf{head}(\mathsf{cons}(k,l)) \approx k & \mathsf{cons}(k,l) \not\approx \mathsf{nil} \\ \mathsf{tail}(\mathsf{cons}(k,l)) \approx l & \mathsf{cons}(\mathsf{head}(l),\mathsf{tail}(l)) \approx l \vee l \approx \mathsf{nil} \end{array}$$

Structurally identical axioms have been mentioned in [13]. The satisfiability of the list axioms is well known. It can also be determined automatically. For example, the theorem prover Beagle [3] in a complete setting and after adding the axioms $\exists\, d_{\mathbb{Z}} \,.\, \mathsf{head}(\mathsf{nil}) \approx d$ and $\mathsf{tail}(\mathsf{nil}) \approx \mathsf{nil}$, terminates on $\mathsf{AX}_{\mathsf{LIST}}$ in a saturated state. Because the axioms satisfy a certain sufficient completeness requirement, this provides a proof of satisfiability. In particular, the list axioms are satisfied in the interpretation I_{LIST} with the domain $D_{\mathsf{LIST}} = \mathbf{LIST}$, the *finite length lists (over integers)*, which we assume to be freely generated by the constructors **nil** and $\mathbf{cons}(\cdot,\cdot)$, and the obvious interpretation for the Σ_{LIST}-operators.

We now turn to the templates for definitions.

Relations. Let Σ^+ be an expansion of Σ_{LIST} and $P \notin \Sigma^+$ a predicate symbol with arity $\mathbb{Z} \times \mathsf{LIST}$. Let Def_P a formula of the form

$$\begin{aligned} &\forall\, k_{\mathbb{Z}}\, l_{\mathsf{LIST}} \,.\, P(k,l) \Leftrightarrow \\ &\qquad l \approx \mathsf{nil} \wedge B[k] && \text{(P1)} \\ &\qquad \vee \exists\, h_{\mathbb{Z}}\, t_{\mathsf{LIST}} \,.\, l \approx \mathsf{cons}(h,t) \wedge C[k,h,t] && \text{(P2)} \\ &\qquad \vee \exists\, h_{\mathbb{Z}}\, t_{\mathsf{LIST}} \,.\, l \approx \mathsf{cons}(h,t) \wedge D[k,h,t] \wedge P(k,t) && \text{(P3)} \end{aligned}$$

where B is a Σ^+-formula of arity $\mathbb{Z}$, and C and D are Σ^+-formulas of arity $\mathbb{Z} \times \mathbb{Z} \times \mathsf{LIST}$.

Lemma 3.1. *Let D be a Σ^+-domain with $D_{\mathsf{LIST}} = \mathbf{LIST}$. Then* Def_P *is an admissible definition of P wrt. Σ^+ and D.*

Proof. Briefly, the proof proceeds by constructing a canonical (minimal) model of the $\Leftarrow$-direction of Def_P, which is also always a model of the $\Rightarrow$-direction. From a logic-programming angle, the user could as well give only the $\Leftarrow$-direction of Def_P, then the system can add the completion ($\Rightarrow$-direction) for disproving purposes.

We assume Interpretations include a valuation component for variables. We write $I_{[x \mapsto \boldsymbol{d}]}$ to indicate an update for the variable x to the domain element $\boldsymbol{d}$.

Let I be a Σ^+-interpretation with domain D. We have to show that I can be expanded to a $(\Sigma^+ \cup \{P\})$-interpretation $I' = I \cup I(P)$, such that $I' \models \mathrm{Def}_P$.

The definition of $I(P)$ utilizes transfinite induction, and we need several orderings for that. Let $\geq_{\mathbb{Z}}$ be a (any) well-ordering on the integers and $\geq$ its extension to the quasi-lexicographic ordering on **LIST**.[1] Because $\geq_{\mathbb{Z}}$ is well-founded and total, $\geq$ is well-founded and total, too (this is well-known). Let $>$ denote the strict subset of $\geq$.

Next, we define an ordering $\geq_P$ on pairs over integers and finite lists over integers as $(\boldsymbol{k_1}, \boldsymbol{l_1}) \geq_P (\boldsymbol{k_2}, \boldsymbol{l_2})$ iff $\boldsymbol{l_1} > \boldsymbol{l_2}$ or else $\boldsymbol{l_1} = \boldsymbol{l_2}$ and $\boldsymbol{k_1} \geq_{\mathbb{Z}} \boldsymbol{k_2}$. Notice that $\geq_P$ is also total and well-founded. Let $>_P$ denote the strict subset of $\geq_P$.

Let $(\boldsymbol{k}, \boldsymbol{l}) \in \mathbb{Z} \times \mathbf{LIST}$ be chosen arbitrarily. We need to decide whether to include $(\boldsymbol{k}, \boldsymbol{l})$ in $I'(P)$ or not, that is, whether to make $I'(P)(\boldsymbol{k}, \boldsymbol{l})$ true or false, respectively. We do this by evaluating the body of Def_P, which resorts to evaluating smaller elements only.

More formally, for a given pair $(\boldsymbol{k}, \boldsymbol{l})$ we define subsets $\epsilon_P(\boldsymbol{k}, \boldsymbol{l})$ and $I(P)_{(\boldsymbol{k},\boldsymbol{l})}$ of $\mathbb{Z} \times D_{\mathsf{LIST}}$. Assume that $\epsilon_P(\boldsymbol{k'}, \boldsymbol{l'})$ has already been defined for all $(\boldsymbol{k'}, \boldsymbol{l'}) \in \mathbb{Z} \times D_{\mathsf{LIST}}$ with $(\boldsymbol{k}, \boldsymbol{l}) >_P (\boldsymbol{k'}, \boldsymbol{l'})$. Where $I(P)_{(\boldsymbol{k},\boldsymbol{l})} = \bigcup_{(\boldsymbol{k},\boldsymbol{l}) >_P (\boldsymbol{k'},\boldsymbol{l'})} \epsilon_P(\boldsymbol{k'}, \boldsymbol{l'})$ define

$$\epsilon_P(\boldsymbol{k}, \boldsymbol{l}) = \{(\boldsymbol{k}, \boldsymbol{l})\} \text{ if } \begin{cases} \boldsymbol{l} = \mathbf{nil} \text{ and } I_{[k \mapsto \boldsymbol{k}]} \models B[k] & \text{or} \\ \boldsymbol{l} = \mathbf{cons}(\boldsymbol{h}, \boldsymbol{t}) \text{ and } I_{[k \mapsto \boldsymbol{k}, h \mapsto \boldsymbol{h}, t \mapsto \boldsymbol{t}]} \models C[k, h, t], & \\ \qquad\qquad \text{for some } \boldsymbol{h} \in \mathbb{Z} \text{ and } \boldsymbol{t} \in D_{\mathsf{LIST}} & \text{or} \\ \boldsymbol{l} = \mathbf{cons}(\boldsymbol{h}, \boldsymbol{t}),\ I_{[k \mapsto \boldsymbol{k}, h \mapsto \boldsymbol{h}, t \mapsto \boldsymbol{t}]} \models D[k, h, t] \text{ and} & \\ \qquad (I \cup I(P)_{(\boldsymbol{k},\boldsymbol{l})})_{[k \mapsto \boldsymbol{k}, t \mapsto \boldsymbol{t}]} \models P(k, t), & \\ \qquad\qquad \text{for some } \boldsymbol{h} \in \mathbb{Z} \text{ and } \boldsymbol{t} \in D_{\mathsf{LIST}} & \end{cases}$$

In all other cases define $\epsilon_P(\boldsymbol{k}, \boldsymbol{l}) = \emptyset$. Finally define $I(P) = \bigcup_{(\boldsymbol{k},\boldsymbol{l})} \epsilon_P(\boldsymbol{k}, \boldsymbol{l})$.

Notice that the conditions in the definition of $\epsilon_P(\boldsymbol{k}, \boldsymbol{l})$ are all well-defined. In particular, we have $(\boldsymbol{k}, \boldsymbol{l}) >_P (\boldsymbol{k}, \boldsymbol{t})$ in the last case. With the definition of $I(P)$ it is straightforward to show $(I \cup I(P)) \models \mathrm{Def}_P$ (assume a $>_P$-minimal pair $(\boldsymbol{k}, \boldsymbol{l})$ under which Def_P evaluates to false in $I \cup I(P)$ and lead this to a contradiction). $\square$

Example. Let $\mathsf{inRange} : \mathbb{Z} \times \mathsf{LIST}$ be a predicate symbol. Consider the extension of $\mathsf{Ax}_{\mathsf{LIST}}$ with the following (admissible) definition for P (the free variables are universally quantified with the obvious sorts).

$$\mathsf{inRange}(n, l) \Leftrightarrow l \approx \mathsf{nil} \vee \exists\, h_{\mathbb{Z}}\ t_{\mathsf{LIST}} \,.\, (l \approx \mathsf{cons}(h, t) \wedge 0 \leq h \wedge h < n \wedge \mathsf{inRange}(n, t))$$

[1] A quasi-lexicographic ordering, or shortlex ordering, compares firstly lists by their length, so that *nil* comes first, and then compares lists of the same length lexicographically.

This example comes from a case study with the first-order logic model checker from [1]. The $\mathsf{inRange}$ predicate is used there to specify lists of "ordered items" handled in a purchase order process, which must all be in a range $0..N-1$, for some $N \geq 0$. The other examples in this paper are contrived.

The following table lists some sample problems together with the runtimes (in seconds) needed to *disprove* them with the provers mentioned.[2]

Problem	Beagle	Spass+T	Z3
$\mathsf{inRange}(4, \mathsf{cons}(1, \mathsf{cons}(5, \mathsf{cons}(2, \mathsf{nil}))))$	6.2	0.3	0.2
$n > 4 \Rightarrow \mathsf{inRange}(n, \mathsf{cons}(1, \mathsf{cons}(5, \mathsf{cons}(2, \mathsf{nil}))))$	7.2	0.3	0.2
$\mathsf{inRange}(n, \mathsf{tail}(l)) \Rightarrow \mathsf{inRange}(n, l)$	3.9	0.3	0.2
$\exists\, n_{\mathbb{Z}}\ l_{\mathsf{LIST}} \,.\, l \not\approx \mathsf{nil} \wedge \mathsf{inRange}(n, l) \wedge n - \mathsf{head}(l) < 1$	2.7	0.3	0.2
$\mathsf{inRange}(n, l) \Rightarrow \mathsf{inRange}(n-1, l)$	8.2	0.3	>60
$l \not\approx \mathsf{nil} \wedge \mathsf{inRange}(n, l) \Rightarrow n - \mathsf{head}(l) > 2$	2.8	0.3	0.2
$n > 0 \wedge \mathsf{inRange}(n, l) \wedge l' = \mathsf{cons}(n-2, l) \Rightarrow \mathsf{inRange}(n, l')$	4.5	5.2	0.2

We remark that none of these problems are solvable by using any of the provers to directly establish consistency of the axioms, definitions and the conjecture. Even if only the $\Leftarrow$-direction is used, Z3 and Spass+T do not terminate. Because the universally quantified variables in the conjectures lead to Skolem constants, the resulting clause set is no longer sufficiently complete (see [3]), and a finite saturation obtained by Beagle does not allow one to conclude satisfiability.

Functions. Let $\Sigma^+ \supseteq \Sigma_{\mathsf{LIST}}$ be a signature, $s \in \mathit{sorts}(\Sigma)$ and $f \notin \Sigma^+$ a function symbol with arity $\mathbb{Z}\times\mathsf{LIST} \mapsto s$. Let Def_f be a set of (implicitly) universally quantified formulas of the form below, where k and h are $\mathbb{Z}$-sorted and t is LIST-sorted:

$$f(k, \mathsf{nil}) \approx b[k] \Leftarrow B[k] \tag{f$_0$}$$

$$f(k, \mathsf{cons}(h, t)) \approx c_1[k, h, t, f(k, t)] \Leftarrow C_1[k, h, t, f(k, t)] \tag{f$_1$}$$

$$\vdots$$

$$f(k, \mathsf{cons}(h, t)) \approx c_n[k, h, t, f(k, t)] \Leftarrow C_n[k, h, t, f(k, t)] \tag{f$_n$}$$

where B is a Σ^+-formula of arity $\mathbb{Z}$, each C_i is a Σ^+-formula of arity $\mathbb{Z}\times\mathbb{Z}\times\mathsf{LIST}\times s$, b is a Σ^+-term of arity $\mathbb{Z} \mapsto s$, and each c_i is a Σ^+-term with arity $\mathbb{Z}\times\mathbb{Z}\times\mathsf{LIST}\times s \mapsto s$.

Lemma 3.2. *Let D be a Σ^+-domain with $D_{\mathsf{LIST}} = \mathbf{LIST}$. If for all $1 \leq i < j \leq n$ the formula*

$$\forall\, k_{\mathbb{Z}}\ h_{\mathbb{Z}}\ t_{\mathsf{LIST}}\ x_s\ .\ C_i[k, h, t, x] \wedge C_j[k, h, t, x] \Rightarrow c_i[k, h, t, x] \approx c_j[k, h, t, x]$$

is valid in all Σ^+-interpretations with domain D then Def_f *is an admissible definition of f wrt. Σ^+ and D.*

[2] Here and below, Beagle has been run with "cautious simplification on" and "ordinary variables on"; Z3, version 4.3.1 with the options "pull-nested-quantifiers", "mbqi" and "macro-finder" on; SPASS+T used Yices as a theory solver. All timings obtained on reasonable recent computer hardware. The input problems are available on the Beagle website `http://users.cecs.anu.edu.au/~baumgart/systems/beagle/`.

Proof. The proof of Lemma 3.2 uses the same model construction technique as the proof of Lemma 3.1. Totality is obtained by interpreting f on an argument tuple such that none of the conditions f_0 to f_n holds true by an arbitrary domain element. The condition in the lemma statement enforces right-uniqueness (functionality). $\square$

The condition in the statement of Lemma 3.2 is needed to make sure that all cases (f_i) and (f_j) for $i \neq j$ are consistent. For example, for $\mathsf{f}(\mathsf{cons}(h,t)) \approx 1 \Leftarrow h \approx 1$ and $\mathsf{f}(\mathsf{cons}(h,t)) \approx \mathsf{a} \Leftarrow h \approx 1 + \mathsf{a}$ this is not the case. Indeed, $\forall h_{\mathbb{Z}} \ . h \approx 1 \wedge h \approx 1 + \mathsf{a} \Rightarrow 1 \approx \mathsf{a}$ is not valid. Notice that establishing the condition is a theorem proving task, which fits well with our method. In the examples below it is trivial.

Example. Let $\mathsf{length} : \mathsf{LIST} \mapsto \mathbb{Z}$, $\mathsf{count} : \mathbb{Z} \times \mathsf{LIST} \mapsto \mathbb{Z}$, $\mathsf{append} : \mathsf{LIST} \times \mathsf{LIST} \mapsto \mathsf{LIST}$ and $\mathsf{in} : \mathbb{Z} \times \mathsf{LIST}$ be operators. Consider the extension of $\mathsf{Ax}_{\mathsf{LIST}}$ with the following (admissible) definitions, in the given order.

$$\begin{array}{rcl}
\mathsf{length}(\mathsf{nil}) \approx 0 & \qquad & \mathsf{append}(\mathsf{nil}, l) \approx l \\
\mathsf{length}(\mathsf{cons}(h,t) \approx 1 + \mathsf{length}(t) & & \mathsf{append}(\mathsf{cons}(h,t), l) \approx \mathsf{cons}(h, \mathsf{append}(t,l)) \\
\mathsf{count}(k, \mathsf{nil}) \approx 0 & & \\
\mathsf{count}(k, \mathsf{cons}(h,t)) \approx \mathsf{count}(k,t) \Leftarrow k \not\approx h & & \mathsf{in}(k,l) \Leftrightarrow \mathsf{count}(k,l) > 0 \\
\mathsf{count}(k, \mathsf{cons}(h,t)) \approx \mathsf{count}(k,t) + 1 \Leftarrow k \approx h & &
\end{array}$$

Here are some sample conjectures together with the times for disproving them.[3]

Problem	Beagle	Spass+T	Z3
$\mathsf{length}(l_1) \approx \mathsf{length}(l_2) \Rightarrow l_1 \approx l_2$	4.3	9.0	0.2
$n \geq 3 \wedge \mathsf{length}(l) \geq 4 \Rightarrow \mathsf{inRange}(n,l)$	5.4	1.1	0.2
$\mathsf{count}(n,l) \approx \mathsf{count}(n, \mathsf{cons}(1,l))$	2.5	0.3	>60
$\mathsf{count}(n,l) \geq \mathsf{length}(l)$	2.7	0.3	>60
$l_1 \not\approx l_2 \Rightarrow \mathsf{count}(n,l_1) \not\approx \mathsf{count}(n,l_2)$	2.4	0.8	>60
$\mathsf{length}(\mathsf{append}(l_1,l_2)) \approx \mathsf{length}(l_1)$	2.1	0.3	0.2
$\mathsf{length}(l_1) > 1 \wedge \mathsf{length}(l_2) > 1 \Rightarrow \mathsf{length}(\mathsf{append}(k,l)) > 4$	37	>60	>60
$\mathsf{in}(n_1,l_1) \wedge \neg\mathsf{in}(n_2,l_2) \wedge l_3 \approx \mathsf{append}(l_1, \mathsf{cons}(n_2,l_2)) \Rightarrow \mathsf{count}(n,l_3) \approx \mathsf{count}(n,l_1)$	>60 (6.2)	9.1	>60

4 Arrays

The signature Σ_{ARRAY} consist of sorts ARRAY and $\mathbb{Z}$ and the operators $\mathsf{read} : \mathsf{ARRAY} \times \mathbb{Z} \mapsto \mathbb{Z}$, $\mathsf{write} : \mathsf{ARRAY} \times \mathbb{Z} \times \mathbb{Z} \mapsto \mathsf{ARRAY}$, and $\mathsf{init} : \mathbb{Z} \mapsto \mathsf{ARRAY}$. The *array axioms* $\mathsf{AX}_{\mathsf{ARRAY}}$ follow:

$$\begin{array}{rcl}
\mathsf{read}(\mathsf{write}(a,i,x),i) \approx x & \qquad & \mathsf{read}(a,i) \approx \mathsf{read}(b,i) \Rightarrow a \approx b \\
\mathsf{read}(\mathsf{write}(a,i,x),j) \approx \mathsf{read}(a,j) \vee i \approx j & & \mathsf{read}(\mathsf{init}(x),i) \approx x
\end{array}$$

[3] The time of 6.2 seconds for the last problem is with “ordinary variables off”.

With the axiom $\mathsf{read}(\mathsf{init}(x), i) \approx x$, a term $\mathsf{init}(t)$ represents an array that is initialized everywhere with t. As with the list axioms, the satisfiability of the array axioms can be established automatically with the Beagle prover by means of a finite saturation.

Relations. Let $\Sigma^+ \supseteq \Sigma_{\mathsf{ARRAY}}$ be a signature and $P \notin \Sigma^+$ a new predicate symbol with arity $\mathbb{Z} \times \mathsf{ARRAY}$. Let Def_P be a formula of the form $\forall\, k_{\mathbb{Z}}\ x_{\mathsf{ARRAY}}\,.\, P(k, x) \Leftrightarrow C[k, x]$, where C is a Σ^+-formula with arity $\mathbb{Z} \times \mathsf{ARRAY}$.

This is a simpler definition than that for LIST, as it does not admit recursion with the new operator P. Of course, this is balanced by the strength of the read operator for arrays. Using it we can easily define useful predicates without recursion. For example the sorted predicate defines arrays in which the first N elements are sorted in increasing order: $\mathsf{sorted}(a, n) \Leftrightarrow (0 \leq i \wedge i < j \wedge j < n) \Rightarrow \mathsf{read}(a, i) \leq \mathsf{read}(a, j)$.

Lemma 4.1. *Def_P is an admissible definition of P wrt. Σ^+ and D.*

Proof. This must be so, since for any Σ^+-interpretation I over D and any x, k, I provides an evaluation of $\phi[k, x]$ and so the obvious interpretation $I(P)$ for $\Sigma^+ \cup \{P\}$ can be defined. □

Functions. Let $\Sigma^+ \supseteq \Sigma_{\mathsf{ARRAY}}$ be a signature, $s \in sorts(\Sigma)$ and $f \notin \Sigma^+$ a function symbol with arity $\mathbb{Z} \times \mathsf{ARRAY} \mapsto s$. Let Def_f be a set of (implicitly) universally quantified formulas of the form below, where k is $\mathbb{Z}$-sorted, a is ARRAY-sorted and y is s-sorted:

$$f(a, k) \approx y \Leftarrow C_1[a, k, y] \tag{f$_1$}$$

$$\vdots$$

$$f(a, k) \approx y \Leftarrow C_n[a, k, y] \tag{f$_n$}$$

where each C_i is a Σ^+-formula of arity $\mathsf{ARRAY} \times \mathbb{Z} \times s$. Note the differences between the LIST version and this definition. Here we do not allow recursion- each C_i is strictly over the signature Σ^+ and, instead of a term c_i we have a universally quantified variable y as the evaluation of f. While some functions on arrays are difficult or impossible to express in this way (for example, the sum of the first N elements of an array), many other interesting functions fit this framework. Consider the function $\mathsf{rev} : \mathsf{ARRAY} \times \mathbb{Z} \mapsto \mathsf{ARRAY}$ that returns a copy of an array with the order of the first N elements reversed:

$$\begin{aligned}\mathsf{rev}(a, n) \approx b \Leftarrow \forall\, i_{\mathbb{Z}}\ .\, 0 \leq i \wedge i < n \wedge \mathsf{read}(b, i) \approx \mathsf{read}(a, n - (i + 1))\\ \vee\ ((0 > i \vee i \geq n) \wedge \mathsf{read}(b, i) \approx \mathsf{read}(a, i))\end{aligned}$$

Lemma 4.2. *Let D be a Σ^+-domain. If, for all $1 \leq i \leq j \leq n$ the formula*

$$C_i[a, k, y_1] \wedge C_j[a, k, y_2] \Rightarrow y_1 \approx y_2$$

is valid in all Σ^+-interpretations with domain D, then Def_f is an admissible definition of f wrt. Σ^+ and D.

Proof. Assume that the above condition is met and that I is a Σ^+ interpretation over D. For this particular $I(f)$, let f be a function which maps a tuple of domain elements

$\mathbf{x}$ to a domain element y of the correct sort such that $I \models C_i[\mathbf{x}, y]$ for some i or to some arbitrary $d \in D$ of the correct sort if no such i and y exist. Since each C_i is a Σ^+ formula, it has an evaluation in I and by assumption any satisfying y is unique up to sort equivalence. Where an arbitrary element is selected no contradiction arises since $I(f) \not\models f(\mathbf{x}) = d \Rightarrow C[\mathbf{x}, d]$. Thus, Def_f is an admissible definition for f. $\square$

Examples. Let the operators $\mathsf{inRange} : \mathsf{ARRAY} \times \mathbb{Z} \times \mathbb{Z}$, max, $\mathsf{distinct}$ be defined as follows (sorted and rev are as defined previously):

$$\mathsf{inRange}(a, r, n) \Leftrightarrow \forall i \,.\, (n \geq i \wedge i \geq 0) \Rightarrow (r \geq \mathsf{read}(a, i) \wedge \mathsf{read}(a, i) \geq 0)$$

$$\mathsf{distinct}(a, n) \Leftrightarrow \forall i, j \,.\, (n > i \wedge n > j \wedge j \geq 0 \wedge i \geq 0) \Rightarrow \mathsf{read}(a, i) \approx \mathsf{read}(a, j) \Rightarrow i \approx j)$$

$$\mathsf{max}(a, n) \approx w \Leftarrow \forall i \,.\, (n > i \wedge i \geq 0) \Rightarrow w \geq \mathsf{read}(a, i)) \wedge (\exists i \,.\, n > i \wedge i \geq 0 \wedge \mathsf{read}(a, i) \approx w)$$

Here are some sample conjectures together with the times for disproving them. [4] Note that u indicates termination with a status “unknown”.

Problem	Beagle	Spass+T	Z3
$n \geq 0 \Rightarrow \mathsf{inRange}(a, \mathsf{max}(a, n), n)$	1.40	0.16	u
$\mathsf{distinct}(\mathsf{init}(n), i)$	0.98	0.15	u
$\mathsf{read}(\mathsf{rev}(a, n+1), 0) = \mathsf{read}(a, n))$	>60	>60(0.27)	>60
$\mathsf{distinct}(a, n) \Rightarrow \mathsf{distinct}(\mathsf{rev}(a, n))$	>60	0.11	0.36
$\exists n_{\mathbb{Z}} \,.\, \neg\mathsf{sorted}(\mathsf{rev}(\mathsf{init}(n), m), m)$	>60	0.16	u
$\mathsf{sorted}(a, n) \wedge n > 0 \Rightarrow \mathsf{distinct}(a, n)$	2.40	0.17	0.01

In addition, SPASS+T, Beagle and Z3 were used to prove the functionality condition in Lemma 4.2 for the max and rev operators. All provers verified the condition for max but only SPASS+T and Z3 verified that for rev.

5 Conclusions

The aim of this work is to provide a reasonably expressive language (in practical terms) that allows one to specify properties of data structures under consideration, like lists and arrays, and that supports disproving by existing theorem provers. The main idea is to capitalize on the strengths of these systems in theorem *proving* and use these for solving (appropriately phrased)disproving problems, instead of relying on their model-building capabilities. The latter, direct approach does not work well in the context of (integer) background theories: both saturation based and SMT methods are inherently incomplete, and so non-provability does not entail non-validity. See [3] for further details under which complete theorem proving is possible.

We gave some example problems and tested them with the theorem provers SPASS+T, Beagle and Z3. These examples are all non-solvable with the direct approach and solvable with our approach. All of them could be solved, and in short time. In general, the

[4] SPASS+T used Yices as a theory solver. The time of 0.27s in the third problem is obtained by excluding the $\mathsf{inRange}$ definition.

first-order solvers Beagle and SPASS+T worked most reliably, possibly thanks to handling quantified formulas natively instead of relying solely on instantiation heuristics. On the other hand, it is easy to find examples where our method does not work. A simple example is the conjecture $\exists\, n_{\mathbb{Z}}\ l_{\mathsf{LIST}}\,.\,\mathsf{length}(\mathsf{cons}(n,l)) \approx 0$. (The direct approach does not work either, e.g., Beagle does not find a *finite* saturation.)

Acknowledgements. We thank the reviewers for their helpful comments.

References

1. M. D. Andreas Bauer, Peter Baumgartner and M. Norrish. Tableaux for verification of data-centric processes. In D. Galmiche and D. Larchey-Wendling, eds., *TABLEAUX*, 2013, *LNAI 8123*, pp. 28–43. Springer.
2. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1), 2009.
3. P. Baumgartner and U. Waldmann. Hierarchic superposition with weak abstraction. In M. P. Bonacina, ed., *CADE-24*, 2013, *LNAI 7898*, pp. 39–57. Springer.
4. A. R. Bradley, Z. Manna, and H. B. Sipma. Whats decidable about arrays? In *VMCAI*, 2006, pp. 427–442. Springer.
5. K. Claessen and A. Lillieström. Automated inference of finite unsatisfiability. *J. Autom. Reasoning*, 47(2):111–132, 2011.
6. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decision procedures for extensions of the theory of arrays. *Ann. Math. Artif. Intell.*, 50(3-4):231–254, 2007.
7. C. Ihlemann, S. Jacobs, and V. Sofronie-Stokkermans. On local reasoning in verification. In Ramakrishnan and Rehof [14], pp. 265–281.
8. D. Kapur and C. G. Zarba. A reduction approach to decision procedures, 2005.
9. C. Lynch and B. Morawska. Automatic decidability. In *LICS*, 2002, pp. 7–. IEEE Computer Society.
10. S. McPeak and G. C. Necula. Data structure specifications via local equality axioms. In K. Etessami and S. K. Rajamani, eds., *CAV*, 2005, *LNCS 3576*, pp. 476–490. Springer.
11. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In Ramakrishnan and Rehof [14], pp. 337–340.
12. L. M. de Moura and N. Bjørner. Generalized, efficient array decision procedures. In *FMCAD*, 2009, pp. 45–52. IEEE.
13. G. Nelson and D. C. Oppen. Fast decision procedures based on congruence closure. *Journal of Association for Computer Machinery*, 27(2), 1980.
14. C. R. Ramakrishnan and J. Rehof, eds. *TACAS*, 2008, *LNCS 4963*. Springer.
15. P. Rümmer and M. A. Shah. Proving programs incorrect using a sequent calculus for java dynamic logic. In Y. Gurevich and B. Meyer, eds., TAP, 2007, *LNCS 4454*, pp. 41–60. Springer.
16. G. S. S. Schulz, K. Claessen, and P. Baumgartner. The TPTP typed first-order form with arithmetic. In N. Bjoerner and A. Voronkov, eds., *LPAR-18*, 2012, *LNAI 7180*. Springer.
17. P. Suter, A. S. Köksal, and V. Kuncak. Satisfiability modulo recursive programs. In E. Yahav, ed., *SAS*, 2011, *LNCS 6887*, pp. 298–315. Springer.
18. U. Waldmann and V. Prevosto. Spass+t. In S. S. Geoff Sutcliffe, Renate Schmidt, ed., *ESCoR*, Seattle, WA, USA, 2006, CEUR Workshop Proceedings, pp. 18–33.