A Decision Procedure for Separation Logic in SMT

Andrew Reynolds, Radu Iosif, Cristina Serban, Tim King

A Decision Procedure for Separation Logic in SMT

Andrew Reynolds[1], Radu Iosif[2], Cristina Serban[2], and Tim King[3]

[1] The University of Iowa
[2] Université de Grenoble Alpes, CNRS, VERIMAG
[3] Google Inc.

Abstract. This paper presents a complete decision procedure for the entire quantifier-free fragment of Separation Logic (SL) interpreted over heaplets with data elements ranging over a parametric multi-sorted (possibly infinite) domain. The algorithm uses a combination of theories and is used as a specialized solver inside a DPLL(T) architecture. A prototype was implemented within the CVC4 SMT solver. Preliminary evaluation suggests the possibility of using this procedure as a building block of a more elaborate theorem prover for SL with inductive predicates, or as back-end of a bounded model checker for programs with low-level pointer and data manipulations.

1 Introduction

Separation Logic (SL) [21] is a logical framework for describing dynamically allocated mutable data structures generated by programs that use pointers and low-level memory allocation primitives. The logics in this framework are used by a number of academic (Space Invader [4]), and industrial (Infer [7]) tools for program verification. The main reason for choosing to work within the SL framework is its ability to provide compositional proofs of programs, based on the principle of *local reasoning*: analyzing different parts of the program (e.g. functions, threads), that work on *disjoint parts of the heap*, and combining the analysis results a posteriori.

The main ingredients of SL are: (i) the *separating conjunction* $\phi * \psi$, which asserts that ϕ and ψ hold for separate portions of the memory (heap), and (ii) the *magic wand* $\varphi \mathrel{-\!\!*} \psi$, which asserts that any extension of the heap by a disjoint heap that satisfies φ must satisfy ψ. Consider, for instance, a memory configuration (heap), in which two cells are allocated, and pointed to by the program variables x and y, respectively, where the x cell has an outgoing selector field to the y cell, and vice versa. The heap can be split into two disjoint parts, each containing exactly one cell, and described by an atomic proposition $\mathsf{x} \mapsto \mathsf{y}$ and $\mathsf{y} \mapsto \mathsf{x}$, respectively. Then the entire heap is described by the formula $\mathsf{x} \mapsto \mathsf{y} * \mathsf{y} \mapsto \mathsf{x}$, which reads "$\mathsf{x}$ points to y and, separately, y points to x".

The expressive power of SL comes with the inherent difficulty of automatically reasoning about the satisfiability of its formulae, as required by push-button program analysis tools. Indeed, SL becomes undecidable in the presence of first-order quantification, even when the fragment uses only points-to predicates, without the separating conjunction or the magic wand [9]. Moreover, the quantifier-free fragment with no data constraints, using only points-to predicates $\mathsf{x} \mapsto (\mathsf{y}, \mathsf{z})$, where x, y and z are interpreted as

memory addresses, is PSPACE-complete, due to the implicit quantification over memory partitions, induced by the semantics of the separation logic connectives [9].

This paper presents a decision procedure for quantifier-free SL which is entirely parameterized by a base theory T of heap locations and data, i.e. the sorts of memory addresses and their contents can be chosen from a large variety of available theories handled by Satisfiability Modulo Theories (SMT) solvers, such as linear integer (real) arithmetic, strings, sets, uninterpreted functions, etc. Given a base theory T, we call $\mathsf{SL}(T)$ the set of separation logic formulae built on top of T, by considering points-to predicates and the separation logic connectives.

Contributions First, we show that the satisfiability problem for the quantifier-free fragment of $\mathsf{SL}(T)$ is PSPACE-complete, provided that the satisfiability of the quantifier-free fragment of the base theory T is in PSPACE. Our method is based on a semantics-preserving translation of $\mathsf{SL}(T)$ into second-order T formulae with quantifiers over a domain of sets and uninterpreted functions, whose cardinality is polynomially bound by the size of the input formula. For the fragment of T formulae produced by the translation from $\mathsf{SL}(T)$, we developed a lazy quantifier instantiation method, based on counterexample-driven refinement. We show that the quantifier instantiation algorithm is sound, complete and terminates on the fragment under consideration. We present our algorithm for the satisfiability of quantifier-free $\mathsf{SL}(T)$ logics as a component of a DPLL(T) architecture, which is widely used by modern SMT solvers. We have implemented the technique as a subsolver of the CVC4 SMT solver [2] and carried out experiments that handle non-trivial examples quite effectively. Applications of our procedure include:

1. Integration within theorem provers for SL with inductive predicates. Most inductive provers for SL use a high-level proof search strategy relying on a separate decision procedure for entailments in the non-inductive fragment, used to simplify the proof obligations, by discharging the non-inductive parts of both left- and right-hand sides, and attain an inductive hypothesis [6]. Due to the hard problem of proving entailments in the non-inductive fragment of SL, these predicates use very simple non-inductive formulae (a list of points-to propositions connected with separating conjunction), for which entailments are proved by syntactic substitutions and matching. Our work aims at extending the language of inductive SL solvers, by outsourcing entailments in a generic non-inductive fragment to a specialized procedure. To this end, we conducted experiments on several entailments corresponding to finite unfoldings of inductive predicates used in practice (Section 6).
2. Use as back-end of a bounded model checker for programs with pointer and data manipulations, based on a complete weakest precondition calculus that involves the magic wand connective [15]. To corroborate this hypothesis, we tested our procedure on verification conditions automatically generated by applying the weakest precondition calculus described in [15] to several program fragments (Section 6).

Related Work The study of the algorithmic properties of Separation Logic [21] has produced an extensive body of literature over time. We need to distinguish between SL with inductive predicates and restrictive non-inductive fragments, and SL without inductive predicates, which is the focus of this paper.

Regarding SL with fixed inductive predicates, Perez and Rybalchenko [16] define a theorem proving framework relying on a combination of SL inference rules dealing with singly-linked lists only, and a superposition calculus dealing with equalities and aliasing between variables. Concerning SL with generic user-provided inductive predicates, the theorem prover CYCLIST [6] builds entailment proofs using a sequent calculus. More recently, the tool SLIDE [14] reduces the entailment between inductive predicates to an inclusion between tree automata. The great majority of these inductive provers focus on applying induction strategies efficiently, and consider a very simple fragment of non-inductive SL formulae, typically conjunctions of equalities and disequalities between location variables and separated points-to predicates, without negations or the magic wand. On a more general note, the tool SPEN [10] considers also arithmetic constraints between the data elements in the memory cells, but fixes the shape of the user-defined predicates.

The idea of applying SMT techniques to decide satisfiability of SL formulae is not new. In their work, Piskac, Wies and Zufferey translate from SL with singly-linked list segments [17] and trees [18], respectively, into first-order logics (GRASS and GRIT) that are decidable in NP. The fragment handled in this paper is incomparable to the logics GRASS [17] and GRIT [18]. On one hand, we do not consider predicates defining recursive data structures, such as singly-linked lists. On the other hand, we deal with the entire quantifier-free fragment of SL, including arbitrary nesting of the magic wand, separating conjunction and classical boolean connectives. As a result, the decision problem we consider is PSPACE-complete, due to the possibility of arbitrary nesting of the boolean and SL connectives. To the best of our knowledge, our implementation is also the first to enable theory combination involving SL, in a fine-grained fashion, directly within the DPLL(T) loop.

The first theoretical results on decidability and complexity of SL without inductive predicates were given by Calcagno, Yang and O'Hearn [9]. They show that the quantifier-free fragment of SL without data constraints is PSPACE-complete by an argument that enumerates a finite (yet large) set of heap models. Their argument shows also the difficulty of the problem, however it cannot be directly turned into an effective decision procedure, because of the ineffectiveness of model enumeration. Building up on this small model property for the quantifier-free fragment of SL, a translation to first-order logic over uninterpreted sorts with empty signature is described in [8]. This translation is very similar to our translation to multi-sorted second-order logic, the main difference being using bounded tuples instead of sets of bounded cardinality. It also provides a decision procedure, though no implementation is available for comparison. A more elaborate tableau-based decision procedure is described by Méry and Galmiche [11]. This procedure generates verification conditions on-demand, but here no data constraints are considered, either.

Our procedure relies on a decision procedure for quantifier-free parametric theory of sets and on-demand techniques for quantifier instantiation. Decision procedures for the theory of sets in SMT are given in [23, 1]. Techniques for model-driven quantifier instantiation were introduced in the context of SMT in [13], and have been developed recently in [19, 5].

2 Preliminaries

We consider formulae in multi-sorted first-order logic, over a *signature* Σ consisting of a countable set of sort symbols and a set of function symbols. We assume that signatures always include a boolean sort Bool with constants $\top$ and $\bot$ denoting true and false respectively, and that each sort σ is implicitly equipped with an equality predicate $\approx$ over $\sigma \times \sigma$. Moreover, we may assume without loss of generality that equality is the only predicate belonging to Σ, since we can model other predicate symbols as function symbols with return sort Bool[4].

We consider a set Var of first-order variables, with associated sorts, and denote by $\varphi(\mathbf{x})$ the fact that the free variables of the formula φ belong to $\mathbf{x} \subseteq \mathsf{Var}$. Given a signature Σ, well-sorted terms, atoms, literals, and formulae are defined as usual, and referred to respectively as *Σ-terms*. We denote by $\phi[\varphi]$ the fact that φ is a subformula (subterm) of ϕ and by $\phi[\psi/\varphi]$ the result of replacing φ with ψ in ϕ. We write $\forall x.\varphi$ to denote universal quantification over variable x, where x occurs as a *free variable* in φ. If $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ is a tuple of variables, we write $\forall \mathbf{x}\, \varphi$ as an abbreviation of $\forall x_1 \cdots \forall x_n \varphi$. We say that a Σ-term is *ground* if it contains no free variables. We assume that Σ contains an if-then-else operator $\mathsf{ite}(b, \mathsf{t}, \mathsf{u})$, of sort $\mathsf{Bool} \times \sigma \times \sigma \to \sigma$, for each sort σ, that evaluates to t if b is true, and to u, otherwise.

A *Σ-interpretation* $\mathcal{I}$ maps: (i) each set sort symbol $\sigma \in \Sigma$ to a non-empty set $\sigma^{\mathcal{I}}$, the *domain* of σ in $\mathcal{I}$, (ii) each function symbol $f \in \Sigma$ of sort $\sigma_1 \times \ldots \times \sigma_n \to \sigma$ to a total function $f^{\mathcal{I}}$ of sort $\sigma_1^{\mathcal{I}} \times \ldots \times \sigma_n^{\mathcal{I}} \to \sigma^{\mathcal{I}}$ if $n > 0$, and to an element of $\sigma^{\mathcal{I}}$ if $n = 0$, and (iii) each variable $x \in \mathbf{x}$ to an element of $\sigma_x^{\mathcal{I}}$, where σ_x is the sort symbol associated with x. We denote by $\mathsf{t}^{\mathcal{I}}$ the interpretation of a term t induced by the mapping $\mathcal{I}$. The satisfiability relation between Σ-interpretations and Σ-formulae, written $\mathcal{I} \models \varphi$, is defined inductively, as usual. We say that $\mathcal{I}$ is *a model of* φ if $\mathcal{I} \models \varphi$.

A *first-order theory* is a pair $T = (\Sigma, \mathbf{I})$ where Σ is a signature and $\mathbf{I}$ is a non-empty set of Σ-interpretations, the *models* of T. For a formula φ, we denote by $[\![\varphi]\!]_T = \{\mathcal{I} \in \mathbf{I} \mid \mathcal{I} \models \varphi\}$ its set of T-models. A Σ-formula φ is *T-satisfiable* if $[\![\varphi]\!]_T \neq \emptyset$, and *$T$-unsatisfiable* otherwise. A Σ-formula φ is *T-valid* if $[\![\varphi]\!]_T = \mathbf{I}$, i.e. if $\neg\varphi$ is T-unsatisfiable. A formula φ *T-entails* a Σ-formula ψ, written $\varphi \models_T \psi$, if every model of T that satisfies φ also satisfies ψ. The formulae φ and ψ are *T-equivalent* if $\varphi \models_T \psi$ and $\psi \models_T \varphi$, and *equisatisfiable* (*in* T) if ψ is T-satisfiable if and only if φ is T-satisfiable. Furthermore, formulas φ and ψ are *equivalent (up to* $\mathbf{k}$*)* if they are satisfied by the same set of models (when restricted to the interpretation of variables $\mathbf{k}$). The T-satisfiability problem asks, given a Σ-formula φ, whether $[\![\varphi]\!]_T \neq \emptyset$, i.e. whether φ has a T-model.

2.1 Separation Logic

In the remainder of the paper we fix a theory $T = (\Sigma, \mathbf{I})$, such that the T-satisfiability for the language of quantifier-free boolean combinations of equalities and disequalties between Σ-terms is decidable. We fix two sorts Loc and Data from Σ, with no restriction other than the fact that Loc is always interpreted as a countably infinite set. We refer to

[4] For brevity, we may write $p(\mathbf{t})$ as shorthand for $p(\mathbf{t}) \approx \top$, where p is a function into Bool.

Separation Logic for T, written $\mathsf{SL}(T)$, as the set of formulae generated by the syntax:

$$\phi := \mathsf{t} \approx \mathsf{u} \mid \mathsf{t} \mapsto \mathsf{u} \mid \mathsf{emp} \mid \phi_1 * \phi_2 \mid \phi_1 \mathrel{-\!\!*} \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1$$

where t and u are well-sorted Σ-terms and that for any atomic proposition $\mathsf{t} \mapsto \mathsf{u}$, t is of sort Loc and u is of sort Data. Also, we consider that Σ has a constant nil of sort Loc, with the meaning that $\mathsf{t} \mapsto \mathsf{u}$ never holds when $\mathsf{t} \approx \mathsf{nil}$. In the following, we write $\phi \vee \psi$ for $\neg(\neg\phi \wedge \neg\psi)$ and $\phi \Rightarrow \psi$ for $\neg\phi \vee \psi$.

Given an interpretation $\mathcal{I}$, a *heap* is a finite partial mapping $h : \mathsf{Loc}^{\mathcal{I}} \rightharpoonup_{\text{fin}} \mathsf{Data}^{\mathcal{I}}$. For a heap h, we denote by $\text{dom}(h)$ its domain. For two heaps h_1 and h_2, we write $h_1 \# h_2$ for $\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$ and $h = h_1 \uplus h_2$ for $h_1 \# h_2$ and $h = h_1 \cup h_2$. For an interpretation $\mathcal{I}$, a heap $h : \mathsf{Loc}^{\mathcal{I}} \rightharpoonup_{\text{fin}} \mathsf{Data}^{\mathcal{I}}$ and a $\mathsf{SL}(T)$ formula ϕ, we define the satisfaction relation $\mathcal{I}, h \models_{\mathsf{SL}} \phi$ inductively, as follows:

$$\begin{array}{lll}
\mathcal{I}, h \models_{\mathsf{SL}} \mathsf{emp} & \iff & h = \emptyset \\
\mathcal{I}, h \models_{\mathsf{SL}} \mathsf{t} \mapsto \mathsf{u} & \iff & h = \{(\mathsf{t}^{\mathcal{I}}, \mathsf{u}^{\mathcal{I}})\} \text{ and } \mathsf{t}^{\mathcal{I}} \not\approx \mathsf{nil}^{\mathcal{I}} \\
\mathcal{I}, h \models_{\mathsf{SL}} \phi_1 * \phi_2 & \iff & \exists h_1, h_2 \,.\, h = h_1 \uplus h_2 \text{ and } \mathcal{I}, h_i \models_{\mathsf{SL}} \phi_i, \text{ for all } i = 1, 2 \\
\mathcal{I}, h \models_{\mathsf{SL}} \phi_1 \mathrel{-\!\!*} \phi_2 & \iff & \forall h' \text{ if } h' \# h \text{ and } \mathcal{I}, h' \models_{\mathsf{SL}} \phi_1 \text{ then } \mathcal{I}, h' \uplus h \models_{\mathsf{SL}} \phi_2
\end{array}$$

The satisfaction relation for the equality atoms $\mathsf{t} \approx \mathsf{u}$ and the Boolean connectives $\wedge$, $\neg$ are the classical ones from first-order logic. In particular $t \approx t$ is always true, denoted by $\top$, for any given heap. The (SL, T)-*satisfiability* problem asks, given an SL formula φ, if there is a T-model $\mathcal{I}$ such that $(\mathcal{I}, h) \models_{\mathsf{SL}} \varphi$ for some heap h.

In this paper we tackle the (SL, T)-satisfiability problem, under the assumption that the quantifier-free data theory $T = (\Sigma, \mathbf{I})$ has a decidable satisfiability problem for constraints involving Σ-terms. It has been proved [9] that the satisfiability problem is PSPACE-complete for the fragment of separation logic in which Data is interpreted as the set of pairs of sort Loc. We generalize this result to any theory whose satisfiability problem, for the quantifier-free fragment, is in PSPACE. This is, in general, the case of most SMT theories, which are typically in NP, such as the linear arithmetic of integers and reals, possibly with sets and uninterpreted functions, etc.

3 Reducing $\mathsf{SL}(T)$ to Multisorted Second-Order Logic

It is well-known [21] that separation logic cannot be formalized as a classical (unsorted) first-order theory, for instance, due to the behavior of the $*$ connective, that does not comply with the standard rules of contraction $\phi \Rightarrow \phi * \phi$ and weakening $\phi * \varphi \Rightarrow \phi$[5]. The basic reason is that $\phi * \varphi$ requires that ϕ and φ hold on *disjoint* heaps. Analogously, $\phi \mathrel{-\!\!*} \varphi$ holds on a heap whose extensions, by disjoint heaps satisfying ϕ, must satisfy φ. In the following, we leverage from the expressivity of multi-sorted first-order theories and translate $\mathsf{SL}(T)$ formulae into quantified formulae in the language of T, assuming that T subsumes a theory of sets and uninterpreted functions.

The integration of separation logic within the DPLL(T) framework [12] requires the input logic to be presented as a multi-sorted logic. To this end, we assume, without loss

[5] Take for instance ϕ as $x \mapsto 1$ and φ as $y \mapsto 2$.

of generality, the existence of a fixed theory $T = (\Sigma, \mathbf{I})$ that subsumes a theory of sets $\mathsf{Set}(\sigma)$ [1], for any sort σ of set elements, whose functions are the union $\cup$, intersection $\cap$ of sort $\mathsf{Set}(\sigma) \times \mathsf{Set}(\sigma) \rightarrow \mathsf{Set}(\sigma)$, singleton $\{.\}$ of sort $\sigma \rightarrow \mathsf{Set}(\sigma)$ and emptyset $\emptyset$ of sort $\mathsf{Set}(\sigma)$. We write $\ell \subseteq \ell'$ as a shorthand for $\ell \cup \ell' \approx \ell'$ and $\mathsf{t} \in \ell$ for $\{\mathsf{t}\} \subseteq \ell$, for any terms ℓ and ℓ' of sort $\mathsf{Set}(\sigma)$ and t of sort σ. The interpretation of the functions in the set theory is the classical (boolean) one.

Also, we assume that Σ contains infinitely many function symbols $\mathsf{pt}, \mathsf{pt}', \ldots \in \Sigma$ of sort $\mathsf{Loc} \rightarrow \mathsf{Data}$, where Loc and Data are two fixed sorts of T, such that for any interpretation $\mathcal{I} \in \mathbf{I}$, $\mathsf{Loc}^{\mathcal{I}}$ is an infinite countable set.

The main idea is to express the atoms and connectives of separation logic in multi-sorted second-order logic by means of a transformation, called *labeling*, which introduces (i) constraints over variables of sort $\mathsf{Set}(\mathsf{Loc})$ and (ii) terms over uninterpreted *points-to* functions of sort $\mathsf{Loc} \rightarrow \mathsf{Data}$. We describe the labeling transformation using judgements of the form $\phi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$, where ϕ is a $\mathsf{SL}(T)$ formula, $\overline{\ell} = \langle \ell_1, \ldots, \ell_n \rangle$ is a tuple of variables of sort $\mathsf{Set}(\mathsf{Loc})$ and $\overline{\mathsf{pt}} = \langle \mathsf{pt}_1, \ldots, \mathsf{pt}_n \rangle$ is a tuple of uninterpreted function symbols occurring under the scopes of universal quantifiers. To ease the notation, we write ℓ and pt instead of the singleton tuples $\langle \ell \rangle$ and $\langle \mathsf{pt} \rangle$. In the following, we also write $\bigcup \overline{\ell}$ for $\ell_1 \cup \ldots \cup \ell_n$, $\ell' \cap \overline{\ell}$ for $\langle \ell' \cap \ell_1, \ldots, \ell' \cap \ell_n \rangle$, $\ell' \cdot \overline{\ell}$ for $\langle \ell', \ell_1, \ldots, \ell_n \rangle$ and $\mathsf{ite}(\mathsf{t} \in \overline{\ell}, \overline{\mathsf{pt}}(\mathsf{t}) = \mathsf{u})$ for $\mathsf{ite}(\mathsf{t} \in \ell_1, \mathsf{pt}_1(\mathsf{t}) = \mathsf{u}, \mathsf{ite}(\mathsf{t} \in \ell_2, \mathsf{pt}_2(\mathsf{t}) = \mathsf{u}, \ldots, \mathsf{ite}(\mathsf{t} \in \ell_n, \mathsf{pt}_n(\mathsf{t}) = \mathsf{u}, \top) \ldots))$.

Intuitively, a labeled formula $\phi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$ says that it is possible to build, from any of its satisfying interpretations $\mathcal{I}$, a heap h such that $\mathcal{I}, h \models_{\mathsf{SL}} \phi$, where $\mathrm{dom}(h) = \ell_1^{\mathcal{I}} \cup \ldots \cup \ell_n^{\mathcal{I}}$ and $h = \mathsf{pt}_1^{\mathcal{I}}{\downarrow_{\ell_1^{\mathcal{I}}}} \cup \ldots \cup \mathsf{pt}_n^{\mathcal{I}}{\downarrow_{\ell_n^{\mathcal{I}}}}$[6]. More precisely, a variable ℓ_i defines a slice of the domain of the heap, whereas the restriction of pt_i to (the interpretation of) ℓ_i describes the heap relation on that slice. Observe that each interpretation of $\overline{\ell}$ and $\overline{\mathsf{pt}}$, such that $\ell_i^{\mathcal{I}} \cap \ell_j^{\mathcal{I}} = \emptyset$, for all $i \neq j$, defines a unique heap.

First, we translate an input $\mathsf{SL}(T)$ formula ϕ into a labeled second-order formula, with quantifiers over sets and uninterpreted functions, defined by the rewriting rules in Figure 1. A labeling step $\phi[\varphi] \Longrightarrow \phi[\psi/\varphi]$ applies if φ and ψ match the antecedent and consequent of one of the rules in Figure 1, respectively. It is not hard to show that this rewriting system is confluent, and we denote by $\phi{\Downarrow}$ the normal form of ϕ with respect to the application of labeling steps.

$$\frac{(\phi * \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]}{\neg \forall \ell_1 \forall \ell_2 \,.\, \neg(\ell_1 \cap \ell_2 \approx \emptyset \wedge \ell_1 \cup \ell_2 \approx \bigcup \overline{\ell} \wedge \phi \triangleleft [\ell_1 \cap \overline{\ell}, \overline{\mathsf{pt}}] \wedge \psi \triangleleft [\ell_2 \cap \overline{\ell}, \overline{\mathsf{pt}}])} \qquad \frac{(\phi \wedge \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]}{\phi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}] \wedge \psi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]}$$

$$\frac{(\phi \mathrel{-\!\!*} \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]}{\forall \ell' \forall \mathsf{pt}' \,.\, (\ell' \cap (\bigcup \overline{\ell}) \approx \emptyset \wedge \phi \triangleleft [\ell', \mathsf{pt}']) \Rightarrow \psi \triangleleft [\ell' \cdot \overline{\ell}, \mathsf{pt}' \cdot \overline{\mathsf{pt}}]} \qquad \frac{(\neg\phi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]}{\neg(\phi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}])}$$

$$\frac{\mathsf{t} \mapsto \mathsf{u} \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]}{\bigcup \overline{\ell} \approx \{\mathsf{t}\} \wedge \mathsf{ite}(\mathsf{t} \in \overline{\ell}, \overline{\mathsf{pt}}(\mathsf{t}) \approx \mathsf{u}) \wedge \mathsf{t} \not\approx \mathsf{nil}} \qquad \frac{\mathsf{emp} \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]}{\bigcup \overline{\ell} \approx \emptyset} \qquad \frac{\varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]}{\varphi} \ \varphi \text{ is pure}$$

Fig. 1. Labeling Rules

[6] We denote by $F{\downarrow_D}$ the restriction of the function F to the domain $D \subseteq \mathrm{dom}(F)$.

Example 1. Consider the $\mathsf{SL}(T)$ formula $(x \mapsto a \mathrel{-\!\!*} y \mapsto b) \wedge \mathsf{emp}$. The reduction to second-order logic is given below:

$$
\begin{array}{l}
((x \mapsto a \mathrel{-\!\!*} y \mapsto b) \wedge \mathsf{emp}) \triangleleft [\ell, \mathsf{pt}] \Longrightarrow^* \\
\ell \approx \emptyset \wedge \forall \ell' \forall \mathsf{pt}' \,.\, \ell' \cap \ell \approx \emptyset \wedge \ell' \approx \{x\} \wedge \mathsf{ite}(x \in \ell', \mathsf{pt}'(x) \approx a, \top) \wedge x \not\approx \mathsf{nil} \Rightarrow \\
\ell' \cup \ell \approx \{y\} \wedge \mathsf{ite}(y \in \ell', \mathsf{pt}'(y) \approx b, \mathsf{ite}(y \in \ell, \mathsf{pt}(y) \approx b, \top)) \wedge y \not\approx \mathsf{nil}
\end{array}
$$

■

The following lemma reduces the (SL, T)-satisfiability problem to the satisfiability of a quantified fragment of the multi-sorted second-order theory T, that contains sets and uninterpreted functions. For an interpretation $\mathcal{I}$, a variable x and a value $s \in \sigma_x^{\mathcal{I}}$, we denote by $\mathcal{I}[x \leftarrow s]$ the extension of $\mathcal{I}$ which maps x into s and behaves like $\mathcal{I}$ for all other symbols. We extend this notation to tuples $\overline{x} = \langle x_1, \ldots, x_n \rangle$ and $\overline{s} = \langle s_1, \ldots, s_n \rangle$ and write $\mathcal{I}[\overline{x} \leftarrow \overline{s}]$ for $\mathcal{I}[x_1 \leftarrow s_1] \ldots [x_n \leftarrow s_n]$. For a tuple of heaps $\overline{h} = \langle h_1, \ldots, h_n \rangle$ we write $\mathrm{dom}(\overline{h})$ for $\langle \mathrm{dom}(h_1), \ldots, \mathrm{dom}(h_n) \rangle$.

Lemma 1. *Given a* $\mathsf{SL}(T)$ *formula* φ *and tuples* $\overline{\ell} = \langle \ell_1, \ldots, \ell_n \rangle$ *and* $\overline{\mathsf{pt}} = \langle \mathsf{pt}_1, \ldots, \mathsf{pt}_n \rangle$ *for* $n > 0$*, for any interpretation* $\mathcal{I}$ *of* T *and any heap* h*:* $\mathcal{I}, h \models_{\mathsf{SL}} \varphi$ *if and only if*

1. *for all heaps* $\overline{h} = \langle h_1, \ldots, h_n \rangle$ *such that* $h = h_1 \uplus \ldots \uplus h_n$*,*
2. *for all heaps* $\overline{h}' = \left\langle h'_1, \ldots, h'_n \right\rangle$ *such that* $h_1 \subseteq h'_1, \ldots, h_n \subseteq h'_n$*,*

we have $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]\Downarrow$.

Although, in principle, satisfiability is undecidable in the presence of quantifiers and uninterpreted functions, the result of the next section strengthens this reduction, by adapting the labeling rules for $*$ and $\mathrel{-\!\!*}$ (Figure 1) to use bounded quantification over finite (set) domains.

4 A Reduction of $\mathsf{SL}(T)$ to Quantifiers Over Bounded Sets

In the previous section, we have reduced any instance of the (SL, T)-satisfiability problem to an instance of the T-satisfiability problem in the second-order multi-sorted theory T which subsumes the theory $\mathsf{Set}(\mathsf{Loc})$ and contains several quantified uninterpreted function symbols of sort $\mathsf{Loc} \mapsto \mathsf{Data}$. A crucial point in the translation is that the only quantifiers occurring in T are of the forms $\forall \ell$ and $\forall \mathsf{pt}$, where ℓ is a variable of sort $\mathsf{Set}(\mathsf{Loc})$ and pt is a function symbol of sort $\mathsf{Loc} \mapsto \mathsf{Data}$. Leveraging from a small model property for SL over the data domain $\mathsf{Data} = \mathsf{Loc} \times \mathsf{Loc}$ [9], we show that it is sufficient to consider only the case when the quantified variables range over a bounded domain of sets. In principle, this allows us to eliminate the universal quantifiers by replacing them with finite conjunctions and obtain a decidability result based on the fact that the quantifier-free theory T with sets and uninterpreted functions is decidable. Since the cost of a-priori quantifier elimination is, in general, prohibitive, in the next section we develop an efficient lazy quantifier instantiation procedure, based on counterexample-driven refinement.

For reasons of self-containment, we quote the following lemma [24] and stress the fact that its proof is oblivious of the assumption $\mathsf{Data} = \mathsf{Loc} \times \mathsf{Loc}$ on the range of heaps. Given a formula ϕ in the language $\mathsf{SL}(T)$, we first define the following measure:

$$
\begin{array}{llll}
|\phi * \psi| = |\phi| + |\psi| & |\phi \mathrel{-\!\!*} \psi| = |\psi| & |\phi \wedge \psi| = \max(|\phi|, |\psi|) & |\neg \phi| = |\phi| \\
|\mathsf{t} \mapsto \mathsf{u}| = 1 & |\mathsf{emp}| = 1 & |\phi| = 0 \text{ if } \phi \text{ is a } \Sigma\text{-formula} &
\end{array}
$$

Intuitively, $|\phi|$ gives the maximum number of *invisible* locations in the domain of a heap h, that are not in the range of $\mathcal{I}$ and which can be distinguished by ϕ. For instance, if $\mathcal{I}, h \models_{\mathsf{SL}} \neg\mathsf{emp} * \neg\mathsf{emp}$ and the domain of h contains more than two locations, then it is possible to restrict $\mathrm{dom}(h)$ to $|\neg\mathsf{emp} * \neg\mathsf{emp}| = 2$ locations only, to satisfy this formula.

Let $\mathrm{Pt}(\phi)$ be the set of terms (of sort $\mathsf{Loc} \cup \mathsf{Data}$) that occur on the left- or right-hand side of a points-to atomic proposition in ϕ. Formally, we have $\mathrm{Pt}(\mathsf{t} \mapsto \mathsf{u}) = \{\mathsf{t}, \mathsf{u}\}$, $\mathrm{Pt}(\phi * \psi) = \mathrm{Pt}(\phi \mathrel{-\!\!*} \psi) = \mathrm{Pt}(\phi) \cup \mathrm{Pt}(\psi)$, $\mathrm{Pt}(\neg\phi) = \mathrm{Pt}(\phi)$ and $\mathrm{Pt}(\mathsf{emp}) = \mathrm{Pt}(\phi) = \emptyset$, for a Σ-formula ϕ. The small model property is given by the next lemma:

Lemma 2. *[24, Proposition 96] Given a formula $\phi \in \mathsf{SL}(T)$, for any interpretation $\mathcal{I}$ of T, let $L \subseteq \mathsf{Loc}^{\mathcal{I}} \setminus \mathrm{Pt}(\phi)^{\mathcal{I}}$ be a set of locations, such that $\|L\| = |\phi|$ and $v \in \mathsf{Data}^{\mathcal{I}} \setminus \mathrm{Pt}(\phi)^{I}$. Then, for any heap h, we have $\mathcal{I}, h \models_{\mathsf{SL}} \phi$ iff $\mathcal{I}, h' \models_{\mathsf{SL}} \phi$, for any heap h' such that:*
- $\mathrm{dom}(h') \subseteq L \cup \mathrm{Pt}(\phi)^{\mathcal{I}}$,
- *for all $\ell \in \mathrm{dom}(h')$, $h'(\ell) \in \mathrm{Pt}(\phi)^{\mathcal{I}} \cup \{v\}$*

Based on the fact that the proof of Lemma 2 [24] does not involve reasoning about data values, other than equality checking, we refine our reduction from the previous section, by bounding the quantifiers to finite sets of constants of known size. To this end, we assume the existence of a total order on the (countable) set of constants in Σ of sort Loc, disjoint from any Σ-terms that occur in a given formula ϕ, and define $\mathrm{Bnd}(\phi, C) = \{c_{m+1}, \ldots, c_{m+|\phi|}\}$, where $m = \max\{i \mid c_i \in C\}$, and $m = 0$ if $C = \emptyset$. Clearly, we have $\mathrm{Pt}(\phi) \cap \mathrm{Bnd}(\phi, C) = \emptyset$ and also $C \cap \mathrm{Bnd}(\phi, C) = \emptyset$, for any C and any ϕ.

We now consider labeling judgements of the form $\varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C]$, where C is a finite set of constants of sort Loc, and modify all the rules in Figure 1, besides the ones with premises $(\phi * \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$ and $(\phi \mathrel{-\!\!*} \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$, by replacing any judgement $\varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$ with $\varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C]$. The two rules in Figure 2 are the bounded-quantifier equivalents of the $(\phi * \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$ and $(\phi \mathrel{-\!\!*} \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$ rules in Figure 1. As usual, we denote by $(\varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C])\Downarrow$ the formula obtained by exhaustively applying the new labeling rules to $\varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C]$.

Observe that the result of the labeling process is a formula in which all quantifiers are of the form $\forall \ell_1 \ldots \forall \ell_n \forall \mathsf{pt}_1 \ldots \forall \mathsf{pt}_n . \bigwedge_{i=1}^{n} \ell_i \subseteq L_i \wedge \bigwedge_{i=1}^{n} \mathsf{pt}_i \subseteq L_i \times D_i \Rightarrow \psi(\overline{\ell}, \overline{\mathsf{pt}})$, where L_i's and D_i's are finite sets of terms, none of which involves quantified variables, and ψ is a formula in the theory T with sets and uninterpreted functions. Moreover, the labeling rule for $\phi \mathrel{-\!\!*} \psi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C]$ uses a fresh constant d that does not occur in ϕ or ψ.

$$\frac{\phi * \psi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C]}{\begin{array}{l}\neg\forall \ell_1 \forall \ell_2 \ . \ \ell_1 \cup \ell_2 \subseteq C \cup \mathrm{Pt}(\phi * \psi) \Rightarrow \\ \qquad \neg(\ell_1 \cap \ell_2 \approx \emptyset \wedge \ell_1 \cup \ell_2 \approx \bigcup \overline{\ell} \wedge \phi \triangleleft [\ell_1 \cap \overline{\ell}, \overline{\mathsf{pt}}, C] \wedge \psi \triangleleft [\ell_2 \cap \overline{\ell}, \overline{\mathsf{pt}}, C])\end{array}}$$

$$\frac{\phi \mathrel{-\!\!*} \psi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C]}{\begin{array}{l}\forall \ell' \forall \mathsf{pt}' \ . \ \ell' \subseteq C' \cup \mathrm{Pt}(\phi \mathrel{-\!\!*} \psi) \wedge \\ \qquad \mathsf{pt}' \subseteq (C' \cup \mathrm{Pt}(\phi \mathrel{-\!\!*} \psi)) \times (\mathrm{Pt}(\phi \mathrel{-\!\!*} \psi) \cup \{d\}) \Rightarrow \\ \qquad (\ell' \cap (\bigcup \overline{\ell}) \approx \emptyset \ \wedge \ \phi \triangleleft [\ell', \mathsf{pt}', C']) \Rightarrow \psi \triangleleft [\ell' \cdot \overline{\ell}, \mathsf{pt}' \cdot \overline{\mathsf{pt}}, C]\end{array}} \quad \begin{array}{l} C' = \mathrm{Bnd}(\phi \wedge \psi, C) \\ d \notin \mathrm{Pt}(\phi \mathrel{-\!\!*} \psi)\end{array}$$

Fig. 2. Bounded Quantifier Labeling Rules

Example 2. We revisit below the labeling of the formula $(x \mapsto a \mathrel{-\!\!*} y \mapsto b) \wedge \mathsf{emp}$:

$$\begin{array}{c}((x \mapsto a \mathrel{-\!\!*} y \mapsto b) \wedge \mathsf{emp}) \triangleleft [\ell, \mathsf{pt}, C] \Longrightarrow^* \\ \ell \approx \emptyset \wedge \forall \ell' \subseteq \{x,y,a,b,c\}\, \forall \mathsf{pt}' \subseteq \{x,y,a,b,c\} \times \{x,y,a,b,d\})\ . \\ \ell' \cap \ell \approx \emptyset \wedge \ell' \approx \{x\} \wedge \mathsf{ite}(x \in \ell', \mathsf{pt}'(x) \approx a, \top) \wedge x \not\approx \mathsf{nil} \Rightarrow \\ \ell' \cup \ell \approx \{y\} \wedge \mathsf{ite}(y \in \ell', \mathsf{pt}'(y) \approx b, \mathsf{ite}(y \in \ell, \mathsf{pt}(y) \approx b, \top)) \wedge y \not\approx \mathsf{nil}\ .\end{array}$$

where $\mathrm{Pt}((x \mapsto a \mathrel{-\!\!*} y \mapsto b) \wedge \mathsf{emp}) = \{x,y,a,b\}$. Observe that the constant c was introduced by the bounded quantifier labeling of the term $x \mapsto a \mathrel{-\!\!*} y \mapsto b$. ■

The next lemma states the soundness of the translation of $\mathsf{SL}(T)$ formulae in a fragment of T that contains only bounded quantifiers, by means of the rules in Figure 2.

Lemma 3. *Given a formula φ in the language $\mathsf{SL}(T)$, for any interpretation $\mathcal{I}$ of T, let $L \subseteq \mathsf{Loc}^{\mathcal{I}} \setminus \mathrm{Pt}(\varphi)^{\mathcal{I}}$ be a set of locations such that $\|L\| = |\varphi|$ and $v \in \mathsf{Data}^{\mathcal{I}} \setminus \mathrm{Pt}(\varphi)^{\mathcal{I}}$ be a data value. Then there exists a heap h such that $\mathcal{I}, h \models_{\mathsf{SL}} \varphi$ iff there exist heaps $\overline{h}' = \left\langle h'_1, \ldots, h'_n \right\rangle$ and $\overline{h}'' = \left\langle h''_1, \ldots, h''_n \right\rangle$ such that:*

1. *for all $1 \leq i < j \leq n$, we have $h'_i \# h'_j$,*
2. *for all $1 \leq i \leq n$, we have $h'_i \subseteq h''_i$ and*
3. *$\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h}')][\overline{\mathsf{pt}} \leftarrow \overline{h}''][C \leftarrow L][d \leftarrow v] \models_T \varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C]\Downarrow$.*

5 A Counterexample-Guided Approach for Solving $\mathsf{SL}(T)$ Inputs

This section presents a novel decision procedure for the (SL, T)-satisfiability of the set of quantifier-free $\mathsf{SL}(T)$ formulae φ. To this end, we present an efficient decision procedure for the T-satisfiability of $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$, obtained as the result of the transformation described in Section 4. The main challenge in doing so is treating the universal quantification occurring in $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$. As mentioned, the key to decidability is that all quantified formulae in $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$ are equivalent to formulas of the form $\forall \mathbf{x}.(\bigwedge \mathbf{x} \subseteq \mathbf{s}) \Rightarrow \varphi$, where each term in the tuple $\mathbf{s}$ is a finite set (or product of sets) of ground Σ-terms. For brevity, we write $\forall \mathbf{x} \subseteq \mathbf{s}.\varphi$ to denote a quantified formula of this form. While such formulae are clearly equivalent to a finite conjunction of instances, the cost of constructing these instances is in practice prohibitively expensive. Following recent approaches for handling universal quantification [13, 19, 5, 20], we use a counterexample-guided approach for choosing instances of quantified formulae that are relevant to the satisfiability of our input. The approach is based on an iterative procedure maintaining an evolving set of quantifier-free Σ-formulae Γ, which is initially a set of formulae obtained from φ by a purification step, described next.

We associate with each closed quantified formula a boolean variable A, called the *guard* of $\forall \mathbf{x}.\varphi$, and a (unique) set of Skolem symbols $\mathbf{k}$ of the same sort as $\mathbf{x}$. We write $(A, \mathbf{k}) \leftrightharpoons \forall \mathbf{x}.\varphi$ to denote that A and $\mathbf{k}$ are associated with $\forall \mathbf{x}.\varphi$. For a set of formulae Γ, we write $\mathrm{Q}(\Gamma)$ to denote the set of quantified formulae whose guard occurs within a formula in Γ. We write $\lfloor \psi \rfloor$ for the result of replacing in ψ all closed quantified formulae (not occurring beneath other quantifiers in ψ) with their corresponding guards. Conversely, we write $\lceil \Gamma \rceil$ to denote the result of replacing all guards in Γ by the quantified formulae they are associated with. Then $\lfloor \psi \rfloor^*$ denotes the smallest set of Σ-formulae:

$\mathsf{solve}_{\mathsf{SL(T)}}(\varphi)$:

Let C be a set of fresh constants of sort Loc such that $\|C\| = |\varphi|$.
Let ℓ and pt be a fresh symbols of sort $\mathsf{Set(Loc)}$ and $\mathsf{Loc} \Rightarrow \mathsf{Data}$ respectively.
Return $\mathsf{solve}_{\mathsf{T}}(\lfloor(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow\rfloor^*)$.

$\mathsf{solve}_{\mathsf{T}}(\Gamma)$:

1. If Γ is T-unsatisfiable,
   return "unsat",
   else let $\mathcal{I}$ be a T-model of Γ.
2. If $\Gamma, A \models_T \lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$ for all $\forall\mathbf{x}.\psi \in \mathrm{Q}(\Gamma)$, where $(A, \mathbf{k}) \leftrightharpoons \forall\mathbf{x}.\psi$ and $A^{\mathcal{I}} = \top$,
   return "sat",
   else let $\mathcal{J}$ be a T-model of $\Gamma \cup \{A, \neg\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor\}$ for some $\prec_{\Gamma,\mathcal{I}}$-minimal $\forall\mathbf{x} \subseteq \mathbf{s}.\psi$,
   where $(A, \mathbf{k}) \leftrightharpoons \forall\mathbf{x} \subseteq \mathbf{s}.\psi$.
3. Let $\mathbf{t}$ be a vector of terms, such that $\mathbf{t} \subseteq \mathbf{s}$, and $\mathbf{t}^{\mathcal{J}} = \mathbf{k}^{\mathcal{J}}$.
   Return $\mathsf{solve}_{\mathsf{T}}(\ \Gamma \cup \lfloor A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]\rfloor^*)$.

Fig. 3. Procedure $\mathsf{solve}_{\mathsf{SL(T)}}$ for deciding (SL, T)-satisfiability of $\mathsf{SL}(T)$ formula φ.

$$\lfloor\psi\rfloor \in \lfloor\psi\rfloor^*$$
$$(\neg A \Rightarrow \lfloor\neg\varphi[\mathbf{k}/\mathbf{x}]\rfloor) \in \lfloor\psi\rfloor^* \quad \text{if } \forall\mathbf{x}.\varphi \in \mathrm{Q}(\lfloor\psi\rfloor^*) \text{ where } (A, \mathbf{k}) \leftrightharpoons \forall\mathbf{x}.\varphi.$$

In other words, $\lfloor\psi\rfloor^*$ contains clauses that witness the negation of each universally quantified formula occurring in ψ. It is easy to see that if ψ is a Σ-formula possibly containing quantifiers, then $\lfloor\psi\rfloor^*$ is a set of quantifier-free Σ-formulae, and if all quantified formulas in ψ are of the form $\forall\mathbf{x} \subseteq \mathbf{s}.\varphi$ mentioned above, then all quantified formulas in $\mathrm{Q}(\lfloor\psi\rfloor^*)$ are also of this form.

Example 3. If ψ is the formula $\forall x.(P(x) \Rightarrow \neg\forall y.R(x, y))$, then $\lfloor\psi\rfloor^*$ is the set:

$$\{A_1, \neg A_1 \Rightarrow \neg(P(k_1) \Rightarrow A_2), \neg A_2 \Rightarrow \neg R(k_1, k_2)\}$$

where $(A_1, k_1) \leftrightharpoons \forall x.(P(x) \Rightarrow \neg\forall y.R(x, y))$ and $(A_2, k_2) \leftrightharpoons \forall y.R(k_1, y)$. ■

Our algorithm $\mathsf{solve}_{\mathsf{SL(T)}}$ for determining the (SL, T)-satisfiability of input φ is given in Figure 3. It first constructs the set C based on the value of $|\varphi|$, which it computes by traversing the structure of φ. It then invokes the subprocedure $\mathsf{solve}_{\mathsf{T}}$ on the set $\lfloor(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow\rfloor^*$ where ℓ and pt are fresh free symbols.

At a high level, the recursive procedure $\mathsf{solve}_{\mathsf{T}}$ takes as input a (quantifier-free) set of T-formulae Γ, where Γ is T-unsatisfiable if and only if $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$ is. On each invocation, $\mathsf{solve}_{\mathsf{T}}$ will either (i) terminate with "unsat", in which case φ is T-unsatisfiable, (ii) terminate with "sat", in which case φ is T-satisfiable, or (iii) add the set corresponding to the purification of the instance $\lfloor A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]\rfloor^*$ to Γ and repeats.

In more detail, in Step 1 of the procedure, we determine the T-satisfiability of Γ using a combination of a satisfiability solver and a decision procedure for T [7]. If Γ is

[7] Non-constant Skolem symbols k introduced by the procedure may be treated as uninterpreted functions. Constraints of the form $k \subseteq S_1 \times S_2$ are translated to $\bigwedge_{c \in S_1} k(c) \in S_2$. Furthermore,

T-unsatisfiable, since Γ is T-entailed by $\lceil\Gamma\rceil$, we may terminate with "unsat". Otherwise, there is a T-model $\mathcal{I}$ for Γ and T. In Step 2 of the procedure, for each A that is interpreted to be true by $\mathcal{I}$, we check whether $\Gamma \cup \{A\}$ T-entails $\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$ for fresh free constants $\mathbf{k}$, which can be accomplished by determining whether $\Gamma \cup \{A, \neg\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor\}$ is T-unsatisfiable. If this check succeeds for a quantified formula $\forall\mathbf{x}.\psi$, the algorithm has established that $\forall\mathbf{x}.\psi$ is entailed by Γ. If this check succeeds for all such quantified formulae, then Γ is equivalent to $\lceil\Gamma\rceil$, and we may terminate with "sat". Otherwise, let $Q^+_{\mathcal{I}}(\Gamma)$ be the subset of $Q(\Gamma)$ for which this check did not succeed. We call this the set of *active quantified formulae* for $(\mathcal{I}, \Gamma)$. We consider an active quantified formula that is minimal with respect to the relation $\prec_{\Gamma,\mathcal{I}}$ over $\mathrm{Q}(\Gamma)$, where:

$$\varphi \prec_{\Gamma,\mathcal{I}} \psi \quad \text{if and only if } \varphi \in \mathrm{Q}(\lfloor\psi\rfloor^*) \cap Q^+_{\mathcal{I}}(\Gamma)$$

By this ordering, our approach considers innermost active quantified formulae first. Let $\forall\mathbf{x}.\psi$ be minimal with respect to $\prec_{\Gamma,\mathcal{I}}$, where $(A, \mathbf{k}) \leftrightharpoons \forall\mathbf{x}.\psi$. Since Γ, A does not T-entail $\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$, there must exist a model $\mathcal{J}$ for $\Gamma \cup \{\lfloor\neg\psi[\mathbf{k}/\mathbf{x}]\rfloor\}$ where $A^{\mathcal{J}} = \top$. In Step 3 of the procedure, we choose a tuple of terms $\mathbf{t} = (t_1, \ldots, t_n)$ based on the model $\mathcal{J}$, and add to Γ the set of formulae obtained by purifying $A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]$, where A is the guard of $\forall\mathbf{x} \subseteq \mathbf{s}.\psi$. Assume that $\mathbf{s} = (s_1, \ldots, s_n)$ and recall that each s_i is a finite union of ground Σ-terms. We choose each $\mathbf{t}$ such that t_i is a subset of s_i for each $i = 1, \ldots n$, and $\mathbf{t}^{\mathcal{J}} = \mathbf{k}^{\mathcal{J}}$. These two criteria are the key to the termination of the algorithm: the former ensures that only a finite number of possible instances can ever be added to Γ, and the latter ensures that we never add the same instance more than once.

Theorem 1. *For all* $\mathsf{SL}(T)$ *formulae* φ, $\mathsf{solve}_{\mathsf{SL}(T)}(\varphi)$*:*

1. *Answers "unsat" only if* φ *is* (SL, T)*-unsatisfiable.*
2. *Answers "sat" only if* φ *is* (SL, T)*-satisfiable.*
3. *Terminates.*

By Theorem 1, $\mathsf{solve}_{\mathsf{SL}(T)}$ is a decision procedure for the (SL, T)-satisfiability of the language of quantifier-free $\mathsf{SL}(T)$ formulae. The following corollary gives a tight complexity bound for the (SL, T)-satisfiability problem.

Corollary 1. *The* (SL, T)*-satisfiability problem is PSPACE-complete for any theory* T *whose satisfiability (for the quantifier-free fragment) is in PSPACE.*

In addition to being sound and complete, in practice, the approach $\mathsf{solve}_{\mathsf{SL}(T)}$ terminates in much less time that its theoretical worst-case complexity, given by the above corollary. This fact is corroborated by our evaluation of our prototype implementation of the algorithm, described in Section 6, and in the following example.

Example 4. Consider the $\mathsf{SL}(T)$ formula $\varphi \equiv \mathsf{emp} \wedge (y \mapsto 0 \mathrel{-\!\!*} y \mapsto 1) \wedge y \not\approx \mathsf{nil}$. When running $\mathsf{solve}_{\mathsf{SL}(T)}(\varphi)$, we first compute the set $C = \{c\}$, and introduce fresh symbols ℓ

---

the domain of k may be restricted to the set $\{c^{\mathcal{I}} \mid c \in S_1\}$ in models $\mathcal{I}$ found in Steps 1 and 2 of the procedure. This restriction comes with no loss of generality since, by construction of $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$, k is applied only to terms occurring in S_1.

and pt of sorts $\mathsf{Set}(\mathsf{Loc})$ and $\mathsf{Loc} \to \mathsf{Data}$ respectively. The formula $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$ is $\ell \approx \emptyset \wedge \forall \ell_4 \forall \mathsf{pt}'.\psi \wedge y \not\approx \mathsf{nil}$, where after simplification ψ is:

$$\begin{array}{rcl} \psi_4 & \equiv & (\ell_4 \subseteq \{y,0,1,c\} \wedge \mathsf{pt}' \subseteq \{y,0,1,c\} \times \{y,0,1,d\}) \Rightarrow \\ & & (\ell_4 \cap \ell \approx \emptyset \wedge \ell_4 \approx \{y\} \wedge \mathsf{pt}'(y) \approx 0 \wedge y \not\approx \mathsf{nil}) \Rightarrow \\ & & (\ell_4 \cup \ell \approx \{y\} \wedge \mathsf{ite}(y \in \ell_4, \mathsf{pt}'(y) \approx 1, \mathsf{pt}(y) \approx 1) \wedge y \not\approx \mathsf{nil}) \end{array}$$

Let $(A_4, (k_1, k_2)) \leftrightharpoons \forall \ell_4 \forall \mathsf{pt}'.\psi_4$. We call the subprocedure $\mathsf{solve}_\mathsf{T}$ on Γ_0, where:

$$\Gamma_0 \equiv \lfloor (\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow \rfloor^* \equiv \{\ell \approx \emptyset \wedge A_4 \wedge y \not\approx \mathsf{nil}, \neg A_4 \Rightarrow \neg \psi_4[k_1, k_2/\ell_4, \mathsf{pt}']\}.$$

The set Γ_0 is T-satisfiable with a model $\mathcal{I}_0$ where $A_4^{\mathcal{I}_0} = \top$. Step 2 of the procedure determines a model $\mathcal{J}$ for $\Gamma_0 \cup \{A_4, \neg\psi_4[k_1, k_2/\ell_4, \mathsf{pt}']\}$.

Let t_1 be $\{y\}$, where we know $t_1^\mathcal{J} = k_1^\mathcal{J}$ since $\mathcal{J}$ must satisfy $k_1 \approx \{y\}$ as a consequence of $\neg\psi_4[k_1, k_2/\ell_4, \mathsf{pt}']$. Let t_2 be a well-sorted subset of $\{y,0,1,c\} \times \{y,0,1,d\}$ such that $t_2^\mathcal{J} = k_2^\mathcal{J}$. Such a subset exists since $\mathcal{J}$ satisfies $k_2 \subseteq \{y,0,1,c\} \times \{y,0,1,d\}$. Notice that $t_2(y)^\mathcal{J} = 0^\mathcal{J}$ since $\mathcal{J}$ must satisfy $k_2(y) \approx 0$. Step 3 of the procedure recursively invokes $\mathsf{solve}_\mathsf{T}$ on Γ_1, where:

$$\begin{array}{l} \Gamma_1 \equiv \Gamma_0 \cup \lfloor A_4 \Rightarrow \psi_4[t_1, t_2/\ell_4, \mathsf{pt}'] \rfloor^* \\ \quad \equiv \Gamma_0 \cup \{A_4 \Rightarrow y \not\approx \mathsf{nil} \Rightarrow (\{y\} \approx \{y\} \wedge \mathsf{ite}(y \in \{y\}, 0 \approx 1, \mathsf{pt}(y) \approx 1) \wedge y \not\approx \mathsf{nil})\} \\ \quad \equiv \Gamma_0 \cup \{A_4 \Rightarrow y \not\approx \mathsf{nil} \Rightarrow \bot\} \end{array}$$

The set Γ_1 is T-unsatisfiable, since the added constraint contradicts $A_4 \wedge y \not\approx \mathsf{nil}$. ■

5.1 Integration in DPLL(T)

We have implemented the algorithm described in this section within the SMT solver CVC4 [2]. Our implementation accepts an extended syntax of SMT-LIB version 2 format [3] for specifying $\mathsf{SL}(T)$ formulae. In contrast to the presentation so far, our implementation does not explicitly introduce quantifiers, and instead treats SL atoms natively using an integrated subsolver that expands the semantics of these atoms in lazy fashion.

In more detail, given a $\mathsf{SL}(T)$ input φ, our implementation lazily computes the expansion of $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$ based on the translation rules in Figures 1 and 2 and the counterexample-guided instantiation procedure in Figure 3. This is accomplished by a module, which we refer to as the SL solver, that behaves analogously to a DPLL(T)-style *theory solver*, that is, a dedicated solver specialized for the T-satisfiability of a conjunction of T-constraints.

The DPLL(T) solving architecture [12] used by most modern SMT solvers, given as input a set of quantifier-free T-formulae Γ, incrementally constructs of set of literals over the atoms of Γ until either it finds a set M that entail Γ at the propositional level, or determines that such a set cannot be found. In the former case, we refer to M as a *satisfying assignment* for Γ. If T is a combination of theories $T_1 \cup \ldots \cup T_n$, then M is partitioned into $M_1 \cup \ldots \cup M_n$ where the atoms of M_i are either T_i-constraints or (dis)equalities shared over multiple theories. We use a theory solver (for T_i) to determine the T_i-satisfiability of the set M_i, interpreted as a conjunction. Given M_i, the solver will either add additional formulae to Γ, or otherwise report that M_i is T_i-satisfiable.

For $\mathsf{SL}(T)$ inputs, we extend our input syntax with a set of functions:

$$\begin{array}{lll} \mapsto : \mathsf{Loc} \times \mathsf{Data} \to \mathsf{Bool} & *^n : \mathsf{Bool}^n \to \mathsf{Bool} & \mathsf{emp} : \mathsf{Bool} \\ -\!\!* : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool} & \mathsf{lbl} : \mathsf{Bool} \times \mathsf{Set}(\mathsf{Loc}) \to \mathsf{Bool} & \end{array}$$

which we call *spatial functions* [8]. We refer to lbl as the *labeling predicate*, which can be understood as a placeholder for the $\triangleleft$ transformation in Figures 1 and 2. We refer to $p(\mathbf{t})$ as an *unlabeled spatial atom* if p is one of $\{\mathsf{emp}, \mapsto, *^n, -\!\!*\}$ and $\mathbf{t}$ is a vector of terms not containing lbl. If a is an unlabeled spatial atom, We refer to $\mathsf{lbl}(a, \ell)$ as a *labeled spatial atom*, and extend these terminologies to literals. We assume that all occurrences of spatial functions in our input φ occur only in unlabeled spatial atoms. Moreover, during execution, our implementation transforms all spatial atoms into a *normal form*, by applying associativity to flatten nested applications of $*$, and distributing Σ-formulae over spatial connectives, e.g. $((\mathsf{x} \mapsto \mathsf{y} \wedge \mathsf{t} \approx \mathsf{u}) * \mathsf{z} \mapsto \mathsf{w}) \iff \mathsf{t} \approx \mathsf{u} \wedge (\mathsf{x} \mapsto \mathsf{y} * \mathsf{z} \mapsto \mathsf{w})$.

When constructing satisfying assignments for φ, we relegate the set of all spatial literals M_k to the SL solver. For all unlabeled spatial literals $(\neg)a$, we add to Γ the formula $(a \Leftrightarrow \mathsf{lbl}(a, \ell_0))$, where ℓ_0 is a distinguished free constant of sort $\mathsf{Set}(\mathsf{Loc})$. Henceforth, it suffices for the SL solver to only consider the labeled spatial literals in M_k. To do so, firstly, it adds to Γ formulae based on the following criteria, which model one step of the reduction from Figure 1:

$$\begin{array}{ll} \mathsf{lbl}(\mathsf{emp}, \ell) \Leftrightarrow \ell \approx \emptyset & \text{if } (\neg)\mathsf{lbl}(\mathsf{emp}, \ell) \in M_k \\ \mathsf{lbl}(t \mapsto u, \ell) \Leftrightarrow \ell \approx \{t\} \wedge \mathsf{pt}(t) \approx u \wedge t \not\approx \mathsf{nil} & \text{if } (\neg)\mathsf{lbl}(t \mapsto u, \ell) \in M_k \\ \mathsf{lbl}((\varphi_1 * \ldots * \varphi_n), \ell) \Rightarrow (\varphi_1[\ell_1] \wedge \ldots \wedge \varphi_n[\ell_n]) & \text{if } \mathsf{lbl}((\varphi_1 * \ldots * \varphi_n), \ell) \in M_k \\ \neg\mathsf{lbl}((\varphi_1 -\!\!* \varphi_2), \ell) \Rightarrow (\varphi_1[\ell_1] \wedge \neg\varphi_2[\ell_2]) & \text{if } \neg\mathsf{lbl}((\varphi_1 -\!\!* \varphi_2), \ell) \in M_k \end{array}$$

where each ℓ_i is a fresh free constant, and $\varphi_i[\ell_i]$ denotes the result of replacing each top-level spatial atom a in φ_i with $\mathsf{lbl}(a, \ell_i)$. These formulae are added eagerly when such literals are added to M_k. To handle negated $*$-atoms and positive $-\!\!*$-atoms, the SL solver adds to Γ formulae based on the criteria:

$$\begin{array}{ll} \neg\mathsf{lbl}((\varphi_1 * \ldots * \varphi_n), \ell) \Rightarrow (\neg\varphi_1[t_1] \vee \ldots \vee \neg\varphi_n[t_n]) & \text{if } \neg\mathsf{lbl}((\varphi_1 * \ldots * \varphi_n), \ell) \in M_k \\ \mathsf{lbl}((\varphi_1 -\!\!* \varphi_2), \ell) \Rightarrow (\neg\varphi_1[t_1, f_1] \vee \varphi_2[t_2, f_2]) & \text{if } \mathsf{lbl}((\varphi_1 -\!\!* \varphi_2), \ell) \in M_k \end{array}$$

where each t_i and f_i is chosen based on the same criterion as described in Figure 3. For wand, we write $\varphi_i[t_i, f_i]$ to denote $\varphi_i'[t_i]$, where φ_i' is the result of replacing all atoms of the form $t \mapsto u$ where $t \in t_1$ in φ_i by $f_i(t) \approx u$.

CVC4 uses a scheme for incrementally checking the T-entailments required by $\mathsf{solve}_\mathsf{T}$, as well as constructing models $\mathcal{J}$ satisfying the negated form of the literals in literals in M_k before choosing such terms [20]. The formula of the above form are added to Γ lazily, that is, after all other solvers (for theories T_i) have determined their corresponding sets of literals M_i are T_i-satisfiable.

Partial Support for Quantifiers In many practical cases it is useful to check the validity of entailments between existentially quantified $\mathsf{SL}(T)$ formulae such as $\exists \mathbf{x} \,.\, \phi(\mathbf{x})$ and $\exists \mathbf{y} \,.\, \psi(\mathbf{y})$. Typically, this problem translates into a satisfiability query for an $\mathsf{SL}(T)$

[8] These functions are over the Bool sort. We refer to these functions as taking *formulae* as input, where formulae may be cast to terms of sort Bool through use of an if-then-else construct.

formula $\exists \mathbf{x} \forall \mathbf{y} \,.\, \phi(\mathbf{x}) \wedge \neg\psi(\mathbf{y})$, with one quantifier alternation. A partial solution to this problem is to first check the satisfiability of ϕ. If ϕ is not satisfiable, the entailment holds trivially, so let us assume that ϕ has a model. Second, we check satisfiability of $\phi \wedge \psi$. Again, if this is unsatisfiable, then the entailment cannot hold, because there exists a model of ϕ which is not a model of ψ. Else, if $\phi \wedge \psi$ has a model, we add an equality $x = y$ for each pair of variables $(x, y) \in \mathbf{x} \times \mathbf{y}$ that are mapped to the same term in this model, the result being a conjunction $E(\mathbf{x}, \mathbf{y})$ of equalities. Finally, we check the satisfiability of the formula $\phi \wedge \neg\psi \wedge E$. If this formula is unsatisfiable, the entailment is valid, otherwise, the test is inconclusive. In section 6, we applied this method manually, to test entailments between existentially quantified variables — general procedure for quantifier instantiation for $\mathsf{SL}(T)$ is envisaged in the near future.

6 Evaluation

We tested our implementation of the (SL, T)-satisfiability procedure in CVC4 (version 1.5 prerelease) [9] on two kinds of benchmarks: (i) finite unfoldings of inductive predicates with data constraints, mostly inspired by existing benchmarks, such as SL-COMP'14 [22], and (ii) verification conditions automatically generated by applying the weakest precondition calculus of [15] to the program loops in Figure 4 several times. All experiments were run on a 2.80GHz Intel(R) Core(TM) i7 CPU machine with with 8MB of cache [10]. For a majority of benchmarks, the runtime of CVC4 is quite low, with the exception of the $n = 4, 8$ cases of the entailments between tree_1^n and tree_2^n formulae, which resulted in a timeout after 300 seconds. For benchmarks where CVC4 times out, the performance bottleneck resides in its ground decision procedure for finite sets, indicating efficient support for this theory is important for our approach to separation logic.

```
1: while w ≠ nil do
2:     assert(w.data = 0)
3:     v := w;
4:     w := w.next;
5:     dispose(v);
6:     do
```

(z)disp

$\mathsf{ls}^0(x) \triangleq \mathsf{emp} \wedge x = \mathsf{nil}$

$\mathsf{ls}^n(x) \triangleq \exists y \,.\, x \mapsto y * \mathsf{ls}^{n-1}(y)$

```
1: while u ≠ nil do
2:     assert(u.data = 0)
3:     w := u.next;
4:     u.next := v;
5:     v := u;
6:     u := w;
7:     do
```

(z)rev

$\mathsf{zls}^0(x) \triangleq \mathsf{emp} \wedge x = \mathsf{nil}$

$\mathsf{zls}^n(x) \triangleq \exists y \,.\, x \mapsto (0, y) * \mathsf{zls}^{n-1}(y)$

Fig. 4. Program Loops

The first set of experiments is reported in Table 1. We have considered inductive predicates commonly used as verification benchmarks [22]. Here we check the validity

[9] Available at `http://cvc4.cs.nyu.edu/web/`.

[10] The CVC4 binary and examples used in these experiments are available at `http://cvc4.cs.nyu.edu/papers/ATVA2016-seplog/`.

lhs	rhs	$n=1$	$n=2$	$n=3$	$n=4$	$n=8$
Unfoldings of inductive predicates						
$\mathsf{lseg}_1(x,y,a) \triangleq \mathsf{emp} \wedge x=y \vee \exists z \exists b.\ x \mapsto (a,z) * \mathsf{lseg}_1(z,y,b) \wedge b=a+10$	$\mathsf{lseg}_2(x,y,a) \triangleq \mathsf{emp} \wedge x=y \vee \exists z \exists b.\ x \mapsto (a,z) * \mathsf{lseg}_2(z,y,b) \wedge a \leq b$	unsat < 0.01s	unsat < 0.01s	unsat < 0.01s	unsat 0.01s	unsat 0.01s
$\mathsf{tree}_1(x,a) \triangleq \mathsf{emp} \wedge x=\mathsf{nil} \vee \exists y \exists z \exists b \exists c.\ x \mapsto (a,y,z) * \mathsf{tree}_1(y,b) * \mathsf{tree}_1(z,c) \wedge b=a-10 \wedge c=a+10$	$\mathsf{tree}_2(x,a) \triangleq \mathsf{emp} \wedge x=\mathsf{nil} \vee \exists y \exists z \exists b \exists c.\ x \mapsto (a,y,z) * \mathsf{tree}_2(y,b) * \mathsf{tree}_2(z,c) \wedge b \leq a \wedge a \leq c$	unsat < 0.01s	unsat 0.06s	unsat 1.89s	timeout > 300s	timeout > 300s
$\mathsf{pos}_1(x,a) \triangleq x \mapsto a \vee \exists y \exists b.\ x \mapsto a * \mathsf{pos}_1(y,b)$	$\mathsf{neg}_1(x,a) \triangleq \neg x \mapsto a \vee \exists y \exists b.\ x \mapsto a * \mathsf{neg}_1(y,b)$	unsat 0.02s	unsat 0.04s	unsat 0.11s	unsat 0.25 s	unsat 3.01s
$\mathsf{pos}_1(x,a) \triangleq x \mapsto a \vee \exists y \exists b.\ x \mapsto a * \mathsf{pos}_1(y,b)$	$\mathsf{neg}_2(x,a) \triangleq x \mapsto a \vee \exists y \exists b.\ \neg x \mapsto a * \mathsf{neg}_2(y,b)$	unsat 0.01s	unsat 0.05s	unsat 0.11s	unsat 0.23s	unsat 2.10s
$\mathsf{pos}_1(x,a) \triangleq x \mapsto a \vee \exists y \exists b.\ x \mapsto a * \mathsf{pos}_1(y,b)$	$\mathsf{neg}_3(x,a) \triangleq x \mapsto a \vee \exists y \exists b.\ x \mapsto a * \neg \mathsf{neg}_3(y,b)$	unsat 0.02s	unsat 0.07s	unsat 0.24s	unsat 0.46s	unsat 4.05s
$\mathsf{pos}_1(x,a) \triangleq x \mapsto a \vee \exists y \exists b.\ x \mapsto a * \mathsf{pos}_1(y,b)$	$\mathsf{neg}_4(x,a) \triangleq x \mapsto a \vee \exists y \exists b.\ \neg x \mapsto a * \neg \mathsf{neg}_4(y,b)$	unsat 0.05s	sat 0.24s	unsat 0.33s	sat 2.77s	sat 24.72s
$\mathsf{pos}_2(x,a) \triangleq x \mapsto a \vee \exists y.\ x \mapsto a * \mathsf{pos}_2(a,y)$	$\mathsf{neg}_5(x,a) \triangleq \neg x \mapsto a \vee \exists y.\ x \mapsto a * \mathsf{neg}_5(a,y)$	unsat 0.02s	unsat 0.05s	unsat 0.14s	unsat 0.32s	unsat 3.69s
$\mathsf{pos}_2(x,a) \triangleq x \mapsto a \vee \exists y.\ x \mapsto a * \mathsf{pos}_2(a,y)$	$\mathsf{neg}_6(x,a) \triangleq x \mapsto a \vee \exists y.\ \neg x \mapsto a * \mathsf{neg}_6(a,y)$	sat 0.02s	unsat 0.04s	unsat 0.13s	unsat 0.27s	unsat 2.22s
Verification conditions						
$\mathsf{ls}^n(w)$	$\mathsf{wp}(\mathbf{disp}, \mathsf{ls}^{n-1}(w))$	< 0.01s	0.02s	0.05s	0.12s	1.97s
$\mathsf{ls}^n(w)$	$\mathsf{wp}^n(\mathbf{disp}, \mathsf{emp} \wedge w=\mathsf{nil})$	< 0.01s	0.02s	0.12s	0.41s	22.97s
$\mathsf{zls}^n(w)$	$\mathsf{wp}(\mathbf{zdisp}, \mathsf{zls}^{n-1}(w))$	0.01s	0.02s	0.05s	0.11s	1.34s
$\mathsf{zls}^n(w)$	$\mathsf{wp}^n(\mathbf{zdisp}, \mathsf{emp} \wedge w=\mathsf{nil})$	0.01s	0.02s	0.11s	0.43s	24.13s
$\mathsf{ls}^n(u) * \mathsf{ls}^0(v)$	$\mathsf{wp}(\mathbf{rev}, \mathsf{ls}^{n-1}(u) * \mathsf{ls}^1(v))$	0.06s	0.08s	0.14s	0.30s	2.83s
$\mathsf{ls}^n(u) * \mathsf{ls}^0(v)$	$\mathsf{wp}^n(\mathbf{rev}, u=\mathsf{nil} \wedge \mathsf{ls}^n(v))$	0.06s	0.12s	0.56s	1.75s	27.82s
$\mathsf{zls}^n(u) * \mathsf{zls}^0(v)$	$\mathsf{wp}(\mathbf{zrev}, \mathsf{zls}^{n-1}(u) * \mathsf{zls}^1(v))$	0.22s	0.04s	0.12s	0.25s	2.16s
$\mathsf{zls}^n(u) * \mathsf{zls}^0(v)$	$\mathsf{wp}^n(\mathbf{zrev}, u=\mathsf{nil} \wedge \mathsf{zls}^n(v))$	0.04s	0.10s	0.41s	1.27s	20.26s

Table 1. Experimental results

of the entailment between lhs and rhs, where both predicates are unfolded $n = 1,2,3,4,8$ times. The second set of experiments, reported in Table 1, considers the verification conditions of the forms $\varphi \Rightarrow \mathsf{wp}(\mathbf{l},\phi)$ and $\varphi \Rightarrow \mathsf{wp}^n(\mathbf{l},\phi)$, where $\mathsf{wp}(\mathbf{l},\phi)$ denotes the weakest precondition of the SL formula ϕ with respect to the sequence of statements $\mathbf{l}$, and $\mathsf{wp}^n(\mathbf{l},\phi) = \mathsf{wp}(\mathbf{l},\ldots\mathsf{wp}(\mathbf{l},\mathsf{wp}(\mathbf{l},\phi))\ldots)$ denotes the iterative application of the weakest precondition n times in a row. We consider the loops depicted in Figure 4, where, for each loop $\mathbf{l}$ we consider the variant $\mathbf{zl}$ as well, which tests that the data values contained within the memory cells are 0, by the assertions on line 2. The postconditions are specified by finite unfoldings of the inductive predicates ls and zls (Figure 4).

7 Conclusions

We have presented a decision procedure for quantifier-free $\mathsf{SL}(T)$ formulas that relies on a efficient, counterexample-guided approach for establishing the T-satisfiability of formulas having quantification over bounded sets. We have described an implementation of the approach as an integrated subsolver in the DPLL(T)-based SMT solver CVC4, showing the potential of the procedure as a backend for tools reasoning about low-level pointer and data manipulations.

References

1. Bansal, K.: Decision Procedures for Finite Sets with Cardinality and Local Theory Extensions. Ph.D. thesis, New York University (2016)
2. Barrett, C., Conway, C., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: Cvc4. In: CAV'11. pp. 171–177 (2011)
3. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB 2.5 standard. Tech. rep., The University of Iowa (2015), `http://smt-lib.org/`
4. Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P., Wies, T., Yang, H.: Shape analysis for composite data structures. In: CAV'07. LNCS, vol. 4590, pp. 178–192 (2007)
5. Bjørner, N., Janota, M.: Playing with quantified satisfaction. In: LPAR'15. EPIC, vol. 35, pp. 15–27 (2015)
6. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: APLAS'12. pp. 350–367 (2012)
7. Calcagno, C., Distefano, D.: Infer: An automatic program verifier for memory safety of c programs. In: Proc. of NASA Formal Methods'11. LNCS, vol. 6617 (2011)
8. Calcagno, C., Gardner, P., Hague, M.: From separation logic to first-order logic. In: FOSSACS'05. pp. 395–409 (2005)
9. Calcagno, C., Yang, H., Ohearn, P.W.: Computability and complexity results for a spatial assertion language for data structures. In: FSTTCS'01. pp. 108–119 (2001)
10. Enea, C., Sighireanu, M., Wu, Z.: On automated lemma generation for separation logic with inductive definitions. In: ATVA'15. pp. 80–96 (2015)
11. Galmiche, D., Méry, D.: Tableaux and resource graphs for separation logic. Journal of Logic and Computation 20(1), 189–231 (2010)
12. Ganzinger, H., Hagen, G., Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Dpll(t): Fast decision procedures. In: CAV'04, pp. 175–188 (2004)
13. Ge, Y., de Moura, L.: Complete instantiation for quantified formulas in satisfiability modulo theories. In: CAV'09. LNCS, vol. 5643 (2009)
14. Iosif, R., Rogalewicz, A., Vojnar, T.: Slide: Separation logic with inductive definitions, URL: `http://www.fit.vutbr.cz/research/groups/verifit/tools/slide/`
15. Ishtiaq, S.S., O'Hearn, P.W.: BI as an assertion language for mutable data structures. In: ACM SIGPLAN Notices. vol. 36, pp. 14–26 (2001)
16. Navarro Pérez, J.A., Rybalchenko, A.: Separation logic + superposition calculus = heap theorem prover. ACM SIGPLAN Notices 46(6), 556–566 (2011)
17. Piskac, R., Wies, T., Zufferey, D.: Automating separation logic using SMT. In: CAV'13. pp. 773–789 (2013)
18. Piskac, R., Wies, T., Zufferey, D.: Automating separation logic with trees and data. In: CAV'14. pp. 711–728 (2014)
19. Reynolds, A., Deters, M., Kuncak, V., Barrett, C.W., Tinelli, C.: Counterexample guided quantifier instantiation for synthesis in CVC4. In: CAV'15. pp. 198–216 (2015)
20. Reynolds, A., King, T., Kuncak, V.: An instantiation-based approach for solving quantified linear arithmetic. CoRR abs/1510.02642 (2015)
21. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: Logic in Computer Science. pp. 55–74. LICS'02 (2002)
22. Sighireanu, M., Cok, D.: Report on SL-COMP 2014. Journal on Satisfiability, Boolean Modeling and Computation 1, 173–186 (2014)
23. Suter, P., Steiger, R., Kuncak, V.: Sets with cardinality constraints in satisfiability modulo theories. In: VMCAI'11, pp. 403–418 (2011)
24. Yang, H.: Local Reasoning for Stateful Programs. Ph.D. thesis, University of Illinois at Urbana-Champaign (2001)