A Decision Procedure for Separation Logic in SMT

Andrew Reynolds[1], Radu Iosif[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 [5], Sleek [17]), and industrial (Infer [8]) tools for program verification. The main reason for chosing 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 global 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), (ii) the *magic wand* $\varphi \multimap \psi$, which asserts that any extension of the heap by a disjoint heap that satisfy ϕ must satisfy ψ, and (iii) the *frame rule*, that exploits separation to provide modular reasoning about programs. 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 viceversa. 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 x *points to* y *and separately* y *points to* x..

The expressive power of SL comes with an 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 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, which can, moreover, be arbitrarily nested [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. Applications of our procedure include:

- Integration with more sophisticated theorem provers for separation logic with inductive predicates. Currently, such solvers concentrate on aspects related to applying induction efficiently and apply heavy restrictions on the ground fragment of the logic considered (typically only separating conjunctions combined with very restricted data theories).
- Use as back-end of a bounded model checker for programs with pointer and data manipulations, based on a complete weakest pre-condition calculus that involves the magic wand connective [15].

Contributions As main contribution, we show that quantifier-free $\mathsf{SL}(T)$ is decidable, provided that the quantifier-free fragment of the base theory T is decidable. Our method is based on a semantics-preserving translation of $\mathsf{SL}(T)$ into first-order T formulae with quantifiers over a domain of sets, whose cardinality is bound by the size of the input formula. For the fragment of T formulae produced by the translation from $\mathsf{SL}(T)$, we developped 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 [12], which is widely used by modern SMT solvers. We have implemented a prototype solver as a branch of the CVC4 SMT solver [3] and carried out experiments that handle non-trivial examples quite effectively.

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.

Concerning SL with user-provided inductive predicates, we mention the theorem prover Sleek [17], which implements a semi-algorithmic entailment check, based on unfoldings and unifications. Along this line of work, the theorem prover Cyclist [7] 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. These tools focus on the induction strategies 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. In addition the tool Spen [10] considers arithmetic constraints between the data elements in the memory cells, but fixes the shape of the user-defined predicates.

Existing translations of SL with inductive definitions (mainly singly-linked lists) into SMT are described in [16, 18]. The goal of these translation is leveraging from SMT technology for deciding entailments between inductive predicates built on top of a restricted fragment of SL. Here we do not treat inductive predicates, but instead consider the entire quantifier-free SL, in combination with available SMT theories for locations (e.g. pointer arithmetic) and data.

The first theoretical results on decidability and complexity of SL without inductive predicates are 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. 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. However, combined, the results of [9] and [11] have inspired our decision procedure, that is parameterized by a decidable quantifier-free fragment of a base theory, and can be used in combination with any available SMT theory.

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 [22, 1]. Techniques for model-driven quantifier instantiation were introduced in the context of SMT in [13], and have been developed recently in [19, 6].

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 $\mathbf{x}$ 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 $\mathbf{x}$. 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 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 $b = \top$ 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

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

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. In particular, we have $\mathcal{I} \models \neg\varphi$ if and only if it is not the case that $\mathcal{I} \models \varphi$. 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. Observe that $[\![\varphi]\!]_T = \mathbf{I} \setminus [\![\neg\varphi]\!]_T$. 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 ϕ, written $\varphi \models_T \phi$, if every model of T that satisfies φ also satisfies ϕ. The formulae φ and ψ are *T-equivalent* if $\varphi \models_T \phi$ and $\phi \models_T \varphi$, 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 *Separation Logic* for T, written $\mathsf{SL}(T)$, as the set of formulae generated by the syntax:

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

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 intended 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 $h : \mathsf{Loc}^{\mathcal{I}} \rightharpoonup_{\mathrm{fin}} \mathsf{Data}^{\mathcal{I}}$. For a heap h, we denote by $\mathrm{dom}(h)$ its domain. For two heaps h_1 and h_2, we write $h_1 \# h_2$ for $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$ 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 $h : \mathsf{Loc}^{\mathcal{I}} \rightharpoonup_{\mathrm{fin}} \mathsf{Data}^{I}$ and a $\mathsf{SL}(T)$ formula ϕ, we define the satisfaction relation $\mathcal{I}, h \models_{\mathsf{SL}} \phi$ inductively, as follows:

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

The satisfaction relation for the equality atoms $\mathsf{t} \approx \mathsf{u}$ and the boolean connectives $\wedge$, $\neg$ is the one defined by T. The (SL, T)-*satisfiability* problem asks whether there is a T-model $\mathcal{I}$ such that $(\mathcal{I}, h) \models_{\mathsf{SL}}$ 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 $\mathsf{Data} = \mathsf{Loc} \times \mathsf{Loc}$. It is not hard, in principle, to extend their complexity argument, based on enumerating a finite set of heaps and checking whether each heap is a model, to more complex data theories, provided that their satisfiability problems are in PSPACE. This is, moreover, 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. However, since the purpose of this paper is providing a cost-effective decision procedure, we consider complexity bounds for the (SL, T)-satisfiability problem as future work.

3 Reducing $\mathsf{SL}(T)$ to Classical First-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, $\phi \multimap \varphi$ 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 first-order logic. To this end, we assume, without loss 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 $\mathsf{Set}(\sigma) \times \mathsf{Set}(\sigma) \to \mathsf{Set}(\sigma)$, singleton $\{.\}$ of sort $\sigma \to \mathsf{Set}(\sigma)$ 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 without loss of generality the existence of infinitely many function symbols $\mathsf{pt}, \mathsf{pt}', \ldots \in \Sigma$ of sort $\mathsf{Loc} \mapsto \mathsf{Data}$, 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[6]. We do not assume any particular interpretation of these symbols in the following and treat them as uninterpreted function symbols.

Observe that considering sets and uninterpreted functions to be part of the base theory T does not impact on the decidability of the satisfiability problem for the quantifier-free fragment of T-formulae. Moreover, the complexity of a set theory $\mathsf{Set}(\sigma)$ is in NP (PSPACE) whenever deciding the satisfiability (validity) of a set of constraints (equalities and disequalities) involving terms of sort σ is in NP (PSPACE) [22].

The main idea is to express the atoms and connectives of separation logic in multi-sorted first-order logic by means of a transformation, called *labeling*, which introduces

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

[6] The generalization of $\mathsf{SL}(T)$ to finite interpretations of Loc is considered as future work.

(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 $h = \mathsf{pt}_1^{\mathcal{I}}\downarrow_{\ell_1^{\mathcal{I}}} \cup \ldots \cup \mathsf{pt}_n^{\mathcal{I}}\downarrow_{\ell_n^{\mathcal{I}}}$[7]. More precisely, a variable ℓ_i defines a slice of the domain of the (global) 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.

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

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

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

Fig. 1. Labeling Rules

Example 1. We sketch the translation of $\mathsf{SL}(T)$ formulae into a first-order multi-sorted theory with sets and uninterpreted functions, by means of an example. Consider the $\mathsf{SL}(T)$ formula $(x \mapsto a * y \mapsto b) \wedge \neg(\top * x \mapsto a)$. A possible reduction using the rules in

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

Figure 1 is the following:

$$
\begin{aligned}
&((x \mapsto a * y \mapsto b) \wedge \neg(\top * x \mapsto a)) \triangleleft [\ell, \mathsf{pt}] && \stackrel{\wedge}{\Longrightarrow}\\
&(x \mapsto a * y \mapsto b) \triangleleft [\ell, \mathsf{pt}] \wedge (\neg(\top * x \mapsto a)) \triangleleft [\ell, \mathsf{pt}] && \stackrel{\neg}{\Longrightarrow}\\
&(x \mapsto a * y \mapsto b) \triangleleft [\ell, \mathsf{pt}] \wedge \neg((\top * x \mapsto a) \triangleleft [\ell, \mathsf{pt}]) && \stackrel{*}{\Longrightarrow}\\
&\neg \forall \ell_1 \forall \ell_2 \,.\, \neg(\ell_1 \cap \ell_2 \approx \emptyset \wedge \ell \approx \ell_1 \cup \ell_2 \wedge x \mapsto a \triangleleft [\ell_1 \cap \ell, \mathsf{pt}] \wedge y \mapsto b \triangleleft [\ell_2 \cap \ell, \mathsf{pt}]) \wedge \\
&\neg((\top * x \mapsto a) \triangleleft [\ell, \mathsf{pt}]) && \stackrel{*}{\Longrightarrow}\\
&\neg \forall \ell_1 \forall \ell_2 \,.\, \neg(\ell_1 \cap \ell_2 \approx \emptyset \wedge \ell \approx \ell_1 \cup \ell_2 \wedge x \mapsto a \triangleleft [\ell_1 \cap \ell, \mathsf{pt}] \wedge y \mapsto b \triangleleft [\ell_2 \cap \ell, \mathsf{pt}]) \wedge \\
&\neg(\neg \forall \ell_3 \,.\, \neg(x \mapsto a \triangleleft [\ell_3, \mathsf{pt}])) && \stackrel{\mapsto}{\Longrightarrow}\\
&\neg \forall \ell_1 \forall \ell_2 \,.\, \neg(\ell_1 \cap \ell_2 \approx \emptyset \wedge \ell \approx \ell_1 \cup \ell_2 \wedge \ell_1 \cap \ell \approx \{x\} \wedge \mathsf{ite}(x \in \ell_1 \cap \ell, \mathsf{pt}(x) \approx a, \top) \wedge x \not\approx \mathsf{nil} \wedge \\
&\ell_2 \cap \ell \approx \{y\} \wedge \mathsf{ite}(y \in \ell_2 \cap \ell, \mathsf{pt}(y) \approx b, \top) \wedge y \not\approx \mathsf{nil}) \wedge \\
&\neg(\neg \forall \ell_3 \,.\, \neg(\ell_3 \approx \{x\} \wedge \mathsf{ite}(x \in \ell_3, \mathsf{pt}(x) \approx a, \top) \wedge x \not\approx \mathsf{nil})) \ .
\end{aligned}
$$

∎

Example 2. Consider the $\mathsf{SL}(T)$ formula $(x \mapsto a \multimap y \mapsto b) \wedge \mathsf{emp}$. The reduction to first-order logic is given below:

$$
\begin{aligned}
&((x \mapsto a \multimap y \mapsto b) \wedge \mathsf{emp}) \triangleleft [\ell, \mathsf{pt}] && \stackrel{\wedge}{\Longrightarrow}\\
&(x \mapsto a \multimap y \mapsto b) \triangleleft [\ell, \mathsf{pt}] \wedge \mathsf{emp} \triangleleft [\ell, \mathsf{pt}] && \stackrel{\mathsf{emp}}{\Longrightarrow}\\
&(x \mapsto a \multimap y \mapsto b) \triangleleft [\ell, \mathsf{pt}] \wedge \ell \approx \emptyset && \stackrel{\multimap}{\Longrightarrow}\\
&\ell \approx \emptyset \wedge \forall \ell' \forall \mathsf{pt}' \,.\, \ell' \cap \ell \approx \emptyset \wedge (x \mapsto a \triangleleft [\ell', \mathsf{pt}']) \Rightarrow y \mapsto b \triangleleft [\langle \ell', \ell \rangle, \langle \mathsf{pt}', \mathsf{pt} \rangle] && \stackrel{\mapsto}{\Longrightarrow}\\
&\ell \approx \emptyset \wedge \forall \ell' \forall \mathsf{pt}' \,.\, \ell' \cap \ell \approx \emptyset \wedge \ell' \approx \{x\} \wedge \mathsf{ite}(x \in \ell', \mathsf{pt}'(x) \approx a, \top) \wedge x \not\approx \mathsf{nil} \Rightarrow \\
&\ell' \cup \ell \approx \{y\} \wedge \mathsf{ite}(y \in \ell', \mathsf{pt}'(y) \approx b, \mathsf{ite}(y \in \ell, \mathsf{pt}(y) \approx b, \top)) \wedge y \not\approx \mathsf{nil} \ .
\end{aligned}
$$

∎

The following lemma reduces the (SL, T)-satisfiability problem to the satisfiability of a quantified fragment of the multi-sorted first-order theory T, that contains sets and uninterpreted functions. Although, in principle, satisfiability is undecidable in the presence of quantifiers and uninterpreted functions, the result of the next section strenghtens this reduction, by adapting the labeling rules for $*$ and $\multimap$ (Figure 1) to use bounded quantification over finite (set) domains.

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

Proof. By induction on the structure of φ. We distinguish the following cases:

- $\varphi \equiv \phi * \psi$ and $\mathcal{I}, h \models_{\mathsf{SL}} \phi * \psi$ iff there exist heaps g_1, g_2 such that $h = g_1 \uplus g_2$ and $\mathcal{I}, g_1 \models_{\mathsf{SL}} \phi$, $\mathcal{I}, g_2 \models_{\mathsf{SL}} \psi$. "$\Rightarrow$" Let $\overline{h}$ and $\overline{h}'$ be tuples of heaps satisfying conditions (1) and (2). By the induction hypothesis we obtain:

$$\mathcal{I}[\ell_1 \leftarrow \mathrm{dom}(g_1)][\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \phi \triangleleft [\ell_1 \cap \overline{\ell}, \overline{\mathsf{pt}}] \Downarrow$$
$$\mathcal{I}[\ell_2 \leftarrow \mathrm{dom}(g_2)][\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \phi \triangleleft [\ell_2 \cap \overline{\ell}, \overline{\mathsf{pt}}] \Downarrow$$

because $g_j \cap h_i \subseteq h_i'$ for each $j = 1,2$ and $i = 1, \ldots, n$. Since, moreover, $\mathrm{dom}(g_1) \cap \mathrm{dom}(g_2) = \emptyset$ and $\mathrm{dom}(g_1) \cup \mathrm{dom}(g_2) = \bigcup_{i=1}^n \mathrm{dom}(h_i)$, we obtain $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \phi * \psi \triangleleft [\ell, \mathsf{pt}] \Downarrow$. "$\Leftarrow$" If $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \phi * \psi \triangleleft [\ell, \mathsf{pt}] \Downarrow$, there exists sets $L_1, L_2 \subseteq \mathsf{Loc}$ such that $L_1 \cap L_2 = \emptyset$ and $\mathrm{dom}(h) = L_1 \cup L_2$. Let $g_1 = h\downarrow_{L_1}$ and $g_2 = h\downarrow_{L_2}$. We have that $h = g_1 \uplus g_2$ and:

$$\mathcal{I}[\ell_1 \leftarrow \mathrm{dom}(g_1)][\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \phi \triangleleft [\ell_1 \cap \overline{\ell}, \overline{\mathsf{pt}}] \Downarrow$$
$$\mathcal{I}[\ell_2 \leftarrow \mathrm{dom}(g_2)][\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \psi \triangleleft [\ell_2 \cap \overline{\ell}, \overline{\mathsf{pt}}] \Downarrow$$

Since, moreover, $g_j = \biguplus_{i=1}^n g_j \cap h_i$ and $g_j \cap h_i \subseteq h_i'$, for $j = 1,2$ and $i = 1, \ldots, n$ we can apply the induction hypothesis to obtain that $\mathcal{I}, g_1 \models_{\mathsf{SL}} \phi$ and $\mathcal{I}, g_2 \models_{\mathsf{SL}} \psi$, thus $\mathcal{I}, h \models_{\mathsf{SL}} \phi * \psi$.
- $\varphi \equiv \phi \multimap \psi$. "$\Rightarrow$" Suppose that $\mathcal{I}, h \models_{\mathsf{SL}} \phi \multimap \psi$ and let $g \subseteq g'$ be heaps such that $g \# h$ and $\mathcal{I}[\ell' \leftarrow \mathrm{dom}(g)][\mathsf{pt}' \leftarrow g'] \models_T \phi \triangleleft [\ell', \mathsf{pt}'] \Downarrow$. By the induction hypothesis, we obtain $\mathcal{I}, g \models_{\mathsf{SL}} \phi$, thus $\mathcal{I}, g \uplus h \models_{\mathsf{SL}} \psi$. Since, moreover, $g \cdot \overline{h}$ and $g' \cdot \overline{h}'$ satisfy the conditions (1) and (2), by the induction hypothesis, we obtain $\mathcal{I}[\ell' \leftarrow \mathrm{dom}(g)][\mathsf{pt}' \leftarrow g'][\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\mathsf{pt} \leftarrow \overline{h}'] \models_T \psi \triangleleft [\ell' \cdot \overline{\ell}, \mathsf{pt}' \cdot \overline{\mathsf{pt}}] \Downarrow$. Since the choice of g and g' was arbitrary, we obtain that $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\mathsf{pt} \leftarrow \overline{h}'] \models_T \phi \multimap \psi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}] \Downarrow$. "$\Leftarrow$" Suppose that $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\mathsf{pt} \leftarrow \overline{h}'] \models_T \phi \multimap \psi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}] \Downarrow$ and let $g \subseteq g'$ be heaps such that $g \# h$ and $\mathcal{I}, g \models_{\mathsf{SL}} \phi$. By the induction hypothesis, we have that $\mathcal{I}[\ell' \leftarrow \mathrm{dom}(g)][\mathsf{pt}' \leftarrow g'] \models_T \phi \triangleleft [\ell', \mathsf{pt}'] \Downarrow$ and since $\mathrm{dom}(g) \cap (\bigcup_{i=1}^n \mathrm{dom}(h_i)) = \emptyset$, we have $\mathcal{I}[\ell' \leftarrow \mathrm{dom}(g)][\mathsf{pt}' \leftarrow g'][\overline{\ell} \leftarrow \overline{h}][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \psi \triangleleft [\ell' \cdot \overline{\ell}, \mathsf{pt}' \cdot \overline{\mathsf{pt}}] \Downarrow$. By the induction hypothesis, we obtain $\mathcal{I}, g \uplus h \models_{\mathsf{SL}} \psi$, thus $\mathcal{I}, h \models_{\mathsf{SL}} \phi \multimap \psi$.
- $\varphi \equiv \mathsf{t} \mapsto \mathsf{u}$. We have $\mathcal{I}, h \models_{\mathsf{SL}} \mathsf{t} \mapsto \mathsf{u}$ iff $h = \{(\mathsf{t}^{\mathcal{I}}, \mathsf{u}^{I})\}$ and $\mathsf{t}^{\mathcal{I}} \not\approx \mathsf{nil}$. "$\Rightarrow$" Let $\overline{h}$ and $\overline{h}'$ be tuples of heaps satisfying the conditions (1) and (2). Then there exists $1 \leq i \leq n$ such that $\mathrm{dom}(h_i) = \{\mathsf{t}^{I}\}$, $h_i'(\mathsf{t}^{I}) = \mathsf{u}^{I}$ and $\mathrm{dom}(h_j) = \emptyset$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$. We obtain, consequently that $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\mathsf{pt} \leftarrow \overline{h}'] \models_T \bigcup \overline{\ell} = \{\mathsf{t}\} \wedge \mathsf{ite}(\mathsf{t} \in \overline{\ell}, \overline{\mathsf{pt}}(\mathsf{t}) = \mathsf{u}) \wedge \mathsf{t} \not\approx \mathsf{nil}$. "$\Leftarrow$" If $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \mathsf{t} \mapsto \mathsf{u} \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}] \Downarrow$ for each $\overline{h}$ and $\overline{h}'$ satisfying conditions (1) and (2), we easily obtain that $\{\mathsf{t}^{\mathcal{I}}\} = \mathrm{dom}(h_i) \subseteq \mathrm{dom}(h_i')$ and $h_i'(\mathsf{t}^{\mathcal{I}}) = \mathsf{u}^{\mathcal{I}}$ for some $1 \leq i \leq n$, leading to $h = \{(\mathsf{t}^{\mathcal{I}}, \mathsf{u}^{\mathcal{I}})\}$. Moreover $\mathsf{t}^{\mathcal{I}} \not\approx \mathsf{nil}$, thus $\mathcal{I}, h \models_{\mathsf{SL}} \mathsf{t} \mapsto \mathsf{u}$.
- $\varphi \equiv \mathsf{emp}$. We have $\mathcal{I}, h \models_{\mathsf{SL}} \mathsf{emp}$ iff $\mathrm{dom}(h) = \emptyset$. "$\Rightarrow$" For any tuples $\overline{h}$ satisfying condition (1) we have $\mathrm{dom}(h_i) = \emptyset$ for all $1 \leq i \leq n$, thus $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \bigcup \overline{\ell} = \emptyset$ for all $\overline{h}'$ satisfying condition (2). "$\Leftarrow$ Let $\overline{h}$ and $\overline{h}'$ be tuples of heaps satisfying conditions (1) and (2), such that $\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h})][\overline{\mathsf{pt}} \leftarrow \overline{h}'] \models_T \mathsf{emp} \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}] \Downarrow$, then $\mathrm{dom}(h) = \emptyset$ and $\mathcal{I}, h \models_{\mathsf{SL}} \mathsf{emp}$.

The cases $\varphi \equiv \phi \wedge \psi$, $\varphi \equiv \neg\phi$ and φ is a Σ-formula are an easy exercise. □

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 first-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 decidability argument for separation logic 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 [23] 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:

$$|\phi * \psi| = |\phi| + |\psi| \qquad |\phi \multimap \psi| = |\psi| \qquad |\phi \wedge \psi| = \max(|\phi|, |\psi|) \qquad |\neg\phi| = |\phi|$$
$$|\mathsf{t} \mapsto \mathsf{u}| = 1 \qquad |\mathsf{emp}| = 1 \qquad |\phi| = 0 \text{ if } \phi \text{ is a } \Sigma\text{-formula}$$

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 two locations only, and obtain h' such that $\|\mathrm{dom}(h')\| = |(\neg\mathsf{emp}) * (\neg\mathsf{emp})| = 2$ and $\mathcal{I}, h' \models_{\mathsf{SL}} (\neg\mathsf{emp}) * (\neg\mathsf{emp})$.

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 $\mathrm{Pt}(\mathsf{t} \mapsto \mathsf{u}) = \{\mathsf{t}, \mathsf{u}\}$, $\mathrm{Pt}(\phi * \psi) = \mathrm{Pt}(\phi \multimap \psi) = \mathrm{Pt}(\phi) \cup \mathrm{Pt}(\psi)$, $\mathrm{Pt}(\neg\phi) = \mathrm{Pt}(\phi)$ and $\mathrm{Pt}(\mathsf{emp}) = \mathrm{Pt}(\phi) = \emptyset$, for a Σ-formula ϕ. The lemma is given below:

Lemma 2. *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 $\ell \in \mathrm{dom}(h')$, $h'(\ell) \in \mathrm{Pt}(\phi)^{\mathcal{I}} \cup \{v\}$*

Proof. See the proof of [23, Proposition 96]. □

The main argument for the decidability of separation logic when $\mathsf{Data} = \mathsf{Loc} \times \mathsf{Loc}$ is that, when an interpretation $\mathcal{I}$ of the free variables is fixed, in order to find a heap model h for a separation logic formula, it is enough to look within a finite set of heaps h', satisfying the above conditions. This argument also gives a PSPACE upper bound for the satisfiability problem in this fragment.

Based on the fact that the proof of Lemma 2 [23] 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 $(\phi \multimap \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$, 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 $(\phi \multimap \psi) \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}]$ 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 $\phi \multimap \psi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C]$ uses a fresh constant d that does not occur in ϕ or ψ.

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

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

Fig. 2. Bounded Quantifier Labeling Rules

Example 3. The outcome of the bounded quantifier labeling of the $\mathsf{SL}(T)$ formula $(x \mapsto a * y \mapsto b) \wedge \neg(\top * x \mapsto a)$ from Example 1 is the following:

$$\begin{array}{c}((x \mapsto a * y \mapsto b) \wedge \neg(\top * x \mapsto a)) \triangleleft [\ell, \mathsf{pt}, C] \\ \Longrightarrow^{*}\end{array}$$
$$\begin{array}{l}\neg \forall \ell_1 \forall \ell_2 \, . \, \ell_1 \cup \ell_2 \subseteq C \cup \{x, y, a, b\} \Rightarrow \neg(\ell_1 \cap \ell_2 \approx \emptyset \wedge \ell \approx \ell_1 \cup \ell_2 \wedge \\ (\ell_1 \cap \ell \approx \{x\} \wedge \mathsf{ite}(x \in \ell_1 \cap \ell, \mathsf{pt}(x) \approx a, \top) \wedge x \not\approx \mathsf{nil}) \wedge \\ (\ell_2 \cap \ell \approx \{y\} \wedge \mathsf{ite}(y \in \ell_2 \cap \ell, \mathsf{pt}(y) \approx b, \top) \wedge y \not\approx \mathsf{nil})) \wedge \\ \neg(\neg \forall \ell_3 \, . \, \ell_3 \subseteq C \cup \{x\} \Rightarrow \neg(\ell_3 \approx \{x\} \wedge \mathsf{ite}(x \in \ell_3, \mathsf{pt}(x) \approx a, \top) \wedge x \not\approx \mathsf{nil})) \quad .\end{array}$$

where $C = \{c_1, c_2\}$ is a set of fresh constants of sort Loc and $|(x \mapsto a * y \mapsto b) \wedge \neg(\top * x \mapsto a)| = 2$, $\mathrm{Pt}(x \mapsto a * y \mapsto b) = \{x, y, a, b\}$, $\mathrm{Pt}(\neg(\top * x \mapsto a)) = \{x, a\}$. ■

Example 4. We revisit below the labeling of the formula $(x \mapsto a \multimap y \mapsto b) \wedge \mathsf{emp}$ from Example 2:

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

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

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. Its proof relies on the following technical fact [23]:

Proposition 1. *Given two formulae $\phi, \psi \in \mathsf{SL}(T)$, for any interpretation $\mathcal{I}$ of T, let $L \subseteq \mathsf{Loc}^{\mathcal{I}} \setminus \mathrm{Pt}(\phi \multimap \psi)^{\mathcal{I}}$ be a set such that $\|L\| = \max(|\phi|, |\psi|)$ and $v \in \mathsf{Data}^{\mathcal{I}} \setminus \mathrm{Pt}(\phi \multimap \psi)^{\mathcal{I}}$ be a data value. Then, for any heap h we have $\mathcal{I}, h \models_{\mathsf{SL}} \phi \multimap \psi$ if and only if, for any heap h' such that $h \# h'$, $\mathcal{I}, h' \models_{\mathsf{SL}} \phi$, and:*
1. *$\mathrm{dom}(h') \subseteq L \cup \mathrm{Pt}(\phi \multimap \psi)^{\mathcal{I}}$,*
2. *$h'(\ell) \in \mathrm{Pt}(\phi \multimap \psi)^{\mathcal{I}} \cup \{v\}$, for all $\ell \in \mathrm{dom}(h')$,*

we have that $\mathcal{I}, h \uplus h' \models_{\mathsf{SL}} \psi$.

Proof. See [23, Proposition 89]. □

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 $\overline{h}' = \langle h'_1, \ldots, h'_n \rangle$ and $\overline{h}'' = \langle h''_1, \ldots, h''_n \rangle$ such that:*
1. *for all $1 \le i < j \le n$, we have $h'_i \# h'_j$,*
2. *for all $1 \le i \le n$, we have $h'_i \subseteq h''_i$ and*
3. *$\mathcal{I}[\overline{\ell} \leftarrow \mathrm{dom}(\overline{h}')][\overline{\mathsf{pt}} \leftarrow \overline{h}''][C \leftarrow L][d \leftarrow v] \models_T \varphi \triangleleft [\overline{\ell}, \overline{\mathsf{pt}}, C] \Downarrow$.*

Proof. By Lemma 2, there exists h such that $\mathcal{I}, h \models_{\mathsf{SL}} \varphi$ if and only if there exists h' such that $\mathrm{dom}(h') \subseteq L \cup \mathrm{Pt}(\varphi)^{\mathcal{I}}$ and for all $\ell \in \mathrm{dom}(h')$, $h'(\ell) \in \mathrm{Pt}(\varphi)^{\mathcal{I}} \cup \{v\}$, for a value $v \in \mathsf{Data}^{\mathcal{I}} \setminus \mathrm{Pt}(\varphi)^{\mathcal{I}}$. It is thus sufficient to prove the statement for these heaps only and assume from now on that h satisfies the conditions of Lemma 2. We prove the following stronger statement:

Fact 1 *For all heaps h such that:*
- *$\mathrm{dom}(h) \subseteq L \cup \mathrm{Pt}(\varphi)^{\mathcal{I}}$ and for all $\ell \in \mathrm{dom}(h)$, $h(\ell) \in \mathrm{Pt}(\varphi)^{\mathcal{I}} \cup \{v\}$,*

we have $\mathcal{I}, h \models_{\mathsf{SL}} \varphi$ iff for all tuples $\overline{h} = \langle h_1, \ldots, h_n \rangle$ and $\overline{h}' = \langle h'_1, \ldots, h'_n \rangle$ such that:
- *$h = h_1 \uplus \ldots \uplus h_n$ and $h_1 \subseteq h'_1, \ldots, h_n \subseteq h'_n$,*

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

The proof is by induction on the structure of φ, among the lines of the proof of Lemma 1. In particular, it can be easily verified that the quantified sets variables and uninterpreted functions belong to the domains required by the rules of Figure 2. □

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, 6, 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 where an evolving set of quantifier-free Σ-formulae Γ is maintained. We assume that Γ is initially a set of formulae obtained from φ by a purification step, described in the following.

We associate 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 $(A, \mathbf{k}) \leftrightarrows \forall \mathbf{x}.\varphi$ 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 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. We write $\lfloor \psi \rfloor^*$ denote the (smallest) set of Σ-formulae such that:

$$\begin{array}{rl} \lfloor \psi \rfloor \in \lfloor \psi \rfloor^* & \\ (\neg A \Rightarrow \lfloor \neg \varphi[\mathbf{k}/\mathbf{x}] \rfloor) \in \lfloor \psi \rfloor^* & \text{if } \forall \mathbf{x}.\varphi \in \mathrm{Q}(\lfloor \psi \rfloor^*) \text{ where } (A, \mathbf{k}) \leftrightarrows \forall \mathbf{x}.\varphi. \end{array}$$

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 5. 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 $(A_1, k_1) \leftrightarrows \forall x.(P(x) \Rightarrow \neg \forall y.R(x, y))$ and $(A_2, k_2) \leftrightarrows \forall y.R(k_1, y)$. $\square$

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, a set of formulae T-entailed by $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$. In some sense, the set Γ can be understood as an under-approximation of $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$. In other words, the

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

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.

models of Γ are a superset of those of $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$. On each invocation, $\mathsf{solve}_{\mathsf{T}}$ will either (i) terminate with "unsat", in which case our input φ is T-unsatisfiable, (ii) terminate with "sat", in which case our input 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 [8]. If Γ is 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

[8] 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, 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.

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.

Lemma 4. *For all T-formulae φ:*

1. *φ is T-satisfiable only if $\lfloor\varphi\rfloor^*$ is T-satisfiable, and*
2. *φ and $\lceil\lfloor\varphi\rfloor^*\rceil$ are T-equivalent up to their shared variables.*

Proof. (Sketch) To show Part (1), let $\mathcal{I}$ be a model of T and φ. Let $\mathcal{J}$ be an extension of $\mathcal{I}$ such that $A^{\mathcal{J}} = (\forall\mathbf{x}.\psi)^{\mathcal{I}}$ for each $\forall\mathbf{x}.\psi \in \mathrm{Q}(\lfloor\varphi\rfloor^*)$ where $(A, \mathbf{k}) \leftrightharpoons \forall\mathbf{x}.\psi$. The interpretation $\mathcal{J}$ satisfies T and $\lfloor\varphi\rfloor^*$ To show Part (2), notice that $\lceil\lfloor\varphi\rfloor^*\rceil$ is a set that can be constructed from an initial value $\{\varphi\}$, and updated by adding formulas of the form $(\neg\forall\mathbf{x}.\psi \Rightarrow \neg\psi[\mathbf{k}/\mathbf{x}])$ for fresh constants $\mathbf{k}$. For each step of this construction, it can be shown that the set of models are same when restricted to the interpretation of all variables apart from $\mathbf{k}$. Thus, by induction, φ and $\lceil\lfloor\varphi\rfloor^*\rceil$ are T-equivalent up to their shared variables. ◻

Lemma 5. *For every recursive call to $\mathsf{solve}_{\mathrm{T}}(\Gamma)$, if $\Gamma, A \models_T \lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$ where $(A, \mathbf{k}) \leftrightharpoons \forall\mathbf{x}.\psi$, then $\Gamma \models_T \forall\mathbf{x}.\psi$.*

Proof. It suffices to show there exists a subset Γ' of Γ such that Γ' does not contain $\mathbf{k}$ and $\Gamma' \models_T \lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$ (if such a Γ' exists, then since $\Gamma' \subseteq \Gamma$ and Γ' does not contain $\mathbf{k}$, we have that $\Gamma \models_T \forall\mathbf{x}.\psi$). By construction, Γ may be partitioned into sets Γ' and Γ'', where Γ' does not contain $\mathbf{k}$, and Γ'' contains only:

1. $\neg A \Rightarrow \lfloor\neg\psi[\mathbf{k}/\mathbf{x}]\rfloor$, and
2. Constraints of the form $\neg A_1 \Rightarrow \lfloor\neg\psi_1[\mathbf{j}/\mathbf{y}]\rfloor$ and $\lfloor A_1 \Rightarrow \psi_1[\mathbf{t}/\mathbf{y}]\rfloor^*$, where $(A_1, \mathbf{j}) \leftrightharpoons \forall\mathbf{y}.\psi_1$ and A_1 does not occur in Γ'.

Assume that $\Gamma \setminus \Gamma'', A, \lfloor\neg\psi[\mathbf{k}/\mathbf{x}]\rfloor$ has a model $\mathcal{I}$. Let $\mathcal{J}$ be an extension of $\mathcal{I}$ such that for each $(A_1, \mathbf{j}) \leftrightharpoons \forall\mathbf{x}.\psi_1$ occurring in $\mathrm{Q}(\Gamma'')$ but not in $\mathrm{Q}(\Gamma')$, we have $A_1^{\mathcal{J}} = (\forall\mathbf{x}.\psi_1)^{\mathcal{I}}$. The interpretation $\mathcal{J}$ satisfies $\Gamma, A, \lfloor\neg\psi[\mathbf{k}/\mathbf{x}]\rfloor$, noting that the constraint $\neg A \Rightarrow \lfloor\neg\psi[\mathbf{k}/\mathbf{x}]\rfloor$ holds since $A^{\mathcal{J}}$ must be $\top$. This contradicts the assumption that $\Gamma, A \models_T \lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$, and thus $\Gamma \setminus \Gamma'', A, \lfloor\neg\psi[\mathbf{k}/\mathbf{x}]\rfloor$ is T-unsatisfiable, in other words, $\Gamma', A \models_T \lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$. Since A does not occur in $\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$, we have that $\Gamma' \models_T \lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$, and hence the lemma holds. ◻

Lemma 6. *If $\Gamma = \{\lfloor\varphi\rfloor^* \mid \varphi \in S\}$ for some set S, and $\forall\mathbf{x}.\psi \in \mathrm{Q}(\Gamma)$ is $\prec_{\Gamma,\mathcal{I}}$-minimal, then $\Gamma \cup \{(\neg)\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor\}$ and $\Gamma \cup \{(\neg)\psi[\mathbf{k}/\mathbf{x}]\}$ are T-equivalent up to their shared variables.*

Proof. (Sketch) By definition of $\prec_{\Gamma,\mathcal{I}}$-minimal, we have that $\Gamma, A_0 \models_T \lfloor\psi_0[\mathbf{k}/\mathbf{x}]\rfloor$ for all $\forall\mathbf{x}_0.\psi_0 \in Q^+_{\mathcal{I}}(\lfloor\psi[\mathbf{k}_0/\mathbf{x}_0]\rfloor)$ where $(A_0, \mathbf{k}_0) \leftrightharpoons \forall\mathbf{x}_0.\psi_0$. For each such formula, by Lemma 5, $\Gamma \models_T \forall\mathbf{x}_0.\psi_0$. Thus, $\Gamma \cup \{(\neg)\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor\}$ and $\Gamma \cup \{(\neg)\lceil\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor\rceil\}$ are T-equivalent up to their shared variables, which by Lemma 4.2 implies that $\Gamma \cup \{(\neg)\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor\}$ and $\Gamma \cup \{(\neg)\psi[\mathbf{k}/\mathbf{x}]\}$ are T-equivalent up to their shared variables. ◻

Lemma 7. *For all* $\mathsf{SL}(T)$ *formulae* φ, $\mathsf{solve}_{\mathsf{T}}(\lfloor\psi\rfloor^*)$ *where* ψ *is* $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$*:*

1. *Answers "unsat" only if* ψ *is* T*-unsatisfiable.*
2. *Answers "sat" only if* ψ *is* T*-satisfiable.*
3. *Terminates.*

Proof. Assume $\mathsf{solve}_{\mathsf{T}}(\Gamma_i)$ calls $\mathsf{solve}_{\mathsf{T}}(\Gamma_{i+1})$ for $i = 0, \ldots, n-1$, where n is finite and $\Gamma_0 = \lfloor\psi\rfloor^*$. By definition of $\mathsf{solve}_{\mathsf{T}}$ and Lemma 4 (2), it can be shown that $\lceil\Gamma_i\rceil$ and $\lceil\Gamma_{i+1}\rceil$ are equisatisfiable in T, and thus by induction $\lceil\Gamma_j\rceil$ and $\lceil\Gamma_k\rceil$ are equisatisfiable in T for each $j, k \in \{1, \ldots, n\}$.

To show Part (1), assume without loss of generality, that $\mathsf{solve}_{\mathsf{T}}(\Gamma_n)$ answers "unsat". Then Γ_n is T-unsatisfiable, and by Lemma 4.1, we have that $\lceil\Gamma_n\rceil$ is T-unsatisfiable. Thus, $\lceil\Gamma_0\rceil$ is T-unsatisfiable, and thus by Lemma 4.2, ψ is T-unsatisfiable.

To show Part (2), without loss of generality, $\mathsf{solve}_{\mathsf{T}}(\Gamma_n)$ answers "sat". Then Γ_n is T-satisfiable with model $\mathcal{I}$. We argue that $\mathcal{I}$ satisfies $\lceil\Gamma_n\rceil$, which implies that $\lceil\Gamma_0\rceil$ is T-satisfiable, and thus by Lemma 4.2, ψ is T-satisfiable. Assume that $\mathcal{I}$ satisfies Γ_n but does not satisfy $\lceil\Gamma_n\rceil$. Thus, $A^{\mathcal{I}} \neq (\forall \mathbf{x}.\psi)^{\mathcal{I}}$ for some $\forall \mathbf{x}.\psi \in \mathrm{Q}(\Gamma_n)$ where $(A, \mathbf{k}) \leftrightharpoons \forall \mathbf{x}.\psi$. In the case that $A^{\mathcal{I}} = \bot$ and $(\forall \mathbf{x}.\psi)^{\mathcal{I}} = \top$, note that $\mathcal{I}$ does not satisfy the formula $(\neg A \Rightarrow \lfloor\neg\varphi[\mathbf{k}/\mathbf{x}]\rfloor) \in \Gamma_n$. In the case that $A^{\mathcal{I}} = \top$ and $(\forall \mathbf{x}.\psi)^{\mathcal{I}} = \bot$, by the definition of $\mathsf{solve}_{\mathsf{T}}$, we have $\Gamma_n, A \models_T \lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$. By Lemma 5, we have that $\Gamma_n \models_T \forall \mathbf{x}.\psi$. Since $\mathcal{I}$ is a model of Γ_n, it must be the case that $(\forall \mathbf{x}.\psi)^{\mathcal{I}} = \top$, contradicting the assumption that $(\forall \mathbf{x}.\psi)^{\mathcal{I}} = \bot$. This contradicts the assumption that $\mathcal{I}$ does not satisfy $\lceil\Gamma_n\rceil$.

To show Part (3), first note that the checks for T-satisfiability and T-entailment terminate, by assumption of a decision procedure for the T-satisfiability of quantifier-free formulae. Moreover, for each quantified formula $\forall \mathbf{x}.\psi$, only a finite number of instances $A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]$ exist for which the algorithm will add $\lfloor A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]\rfloor^*$ to Γ, which implies that the algorithm will consider only a finite number of quantified formulae, each for which only a finite number of instances will be added in this way. Thus, it suffices to show that the algorithm adds to Γ only sets $\lfloor A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]\rfloor^*$ that are not a subset of Γ. Assume this is not the case for some $\lfloor A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]\rfloor^*$, where $\forall \mathbf{x}.\psi \in \mathrm{Q}(\Gamma_n)$, $\forall \mathbf{x}.\psi$ is $\prec_{\Gamma,\mathcal{I}}$-minimal, and $(A, \mathbf{k}) \leftrightharpoons \forall \mathbf{x}.\psi$. The terms $\mathbf{t}$ meet the criteria in the procedure, in particular, for some model $\mathcal{J}$ of $\Gamma \cup \{\neg\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor\}$ where $A^{\mathcal{J}} = \top$, we have $\mathbf{t}^{\mathcal{J}} = \mathbf{k}^{\mathcal{J}}$. Since $A^{\mathcal{J}} = \top$ and $(A \Rightarrow \lfloor\psi[\mathbf{t}/\mathbf{x}]\rfloor) \in \lfloor A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]\rfloor^* \subseteq \Gamma$, $\mathcal{J}$ satisfies $\lfloor\psi[\mathbf{t}/\mathbf{x}]\rfloor$. Also, $\mathcal{J}$ satisfies $\neg\lfloor\psi[\mathbf{k}/\mathbf{x}]\rfloor$. By Lemma 6, since $\forall \mathbf{x}.\psi$ is $\prec_{\Gamma,\mathcal{I}}$-minimal, $\mathcal{J}$ satisfies $\psi[\mathbf{t}/\mathbf{x}]$ and $\neg\psi[\mathbf{k}/\mathbf{x}]$. Thus, $\mathbf{k}^{\mathcal{J}} \neq \mathbf{t}^{\mathcal{J}}$, contradicting our assumption. ◻

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.*

Proof. To show Part (1), by Lemma 7(1), it must be the case that $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$ is T-unsatisfiable. Thus, there cannot be heaps meeting the requirements of $\overline{h}'$ and $\overline{h}''$ in Lemma 3, and by that lemma means that φ is (SL, T)-unsatisfiable. To show Part (2), by Lemma 7(2), it must be the case that $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$ is T-satisfied by a model of T, call it $\mathcal{J}$. Let h' be the heap with domain $\ell^{\mathcal{J}}$ such that $h'(u) = \mathsf{pt}^{\mathcal{J}}(u)$ for all

$u \in \ell^{\mathcal{J}}$, and let $h'' = \mathsf{pt}^{\mathcal{J}}$. Since Loc has infinite cardinality, and due to the structure of $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$, we may assume that $C^{\mathcal{J}} \subseteq \mathsf{Loc}^{\mathcal{J}} \setminus \mathrm{Pt}(\varphi)^{\mathcal{J}}$, call this set L. Let $v = d^{\mathcal{J}}$. Let $\mathcal{I}$ be an interpretation such that $\mathcal{I}[\ell \leftarrow \mathrm{dom}(h')][\mathsf{pt} \hookleftarrow h''][C \leftarrow L][d \leftarrow v] = \mathcal{J}$. Since by assumption $\mathcal{J} \models_T (\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$, then by Lemma 3, there exists a heap h such that $\mathcal{I}, h \models_{\mathsf{SL}} \varphi$, and thus φ is (SL, T)-satisfiable. Part (3) is an immediate consequence of Lemma 7(3). $\square$

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.*

Proof. PSPACE-hardness is by the reduction from QSAT, which generalizes [9, Definition 7] to our case. Let $\phi \equiv \forall x_1 \exists y_1 \ldots \forall x_n \exists y_n \;.\; \psi$ be an instance of QSAT, where ψ is a boolean combination of the variables x_i and y_i of sort Bool. We encode ϕ in $\mathsf{SL}(T)$ using the translation function $\mathrm{Tr}(.)$, defined by induction on the structure of ϕ:

$$\begin{array}{rclrcl}
\mathrm{Tr}(x_i) & \equiv & (\mathsf{t}_i \mapsto d) * \top & \mathrm{Tr}(y_i) & \equiv & (\mathsf{u}_i \mapsto d) * \top \\
\mathrm{Tr}(\neg\psi) & \equiv & \neg\mathrm{Tr}(\psi) & \mathrm{Tr}(\psi_1 \bullet \psi_2) & \equiv & \mathrm{Tr}(\psi_1) \bullet \mathrm{Tr}(\psi_2) \\
\mathrm{Tr}(\exists y_i \;.\; \psi) & \equiv & ((\mathsf{u}_i \mapsto d) \vee \mathsf{emp}) * \mathrm{Tr}(\psi) & \mathrm{Tr}(\forall x_i \;.\; \psi) & \equiv & \neg(((\mathsf{t}_i \mapsto d) \vee \mathsf{emp}) * \neg\mathrm{Tr}(\psi))
\end{array}$$

where d is constant of sort Data and $\bullet \in \{\wedge, \vee\}$. It is not hard to check that $\phi = \top$ if and only if there exists a T-interpretation $\mathcal{I}$ and a heap h such that $\mathcal{I}, h \models_{\mathsf{SL}} \mathrm{Tr}(\phi)$.

To prove that (SL, T)-satisfiability is in PSPACE, we analyse the space complexity of the $\mathsf{solve}_{\mathsf{SL(T)}}$ algorithm. First, for any $\mathsf{SL}(T)$-formula φ, let $\mathrm{Size}(\varphi)$ be the size of the syntax tree of φ. Clearly $\|\lfloor\varphi\rfloor^*\| \leq \mathrm{Size}(\varphi)$, and moreover, $\mathrm{Size}(\psi) \leq \mathrm{Size}(\varphi)$ for each $\psi \in \lfloor\varphi\rfloor^*$. Then a representation of the set $\lfloor\varphi\rfloor^*$ by simply enumerating its elements will take space at most $\mathrm{Size}(\varphi)^2$. For a set of formulae Γ, let $\mathrm{Size}(\Gamma) = \sum_{\varphi \in \Gamma} \mathrm{Size}(\varphi)$ denote the size of its enumerative representation.

Second, it is not difficult to see that $|\varphi| \leq \mathrm{Size}(\varphi)$ and $\|\mathrm{Pt}(\varphi)\| \leq \mathrm{Size}(\varphi)$, for any $\mathsf{SL}(T)$-formula φ. Then, for each subformula $\forall x \subseteq s \;.\; \psi$ of $\varphi \triangleleft [\ell, \mathsf{pt}, C]\Downarrow$, we have $\|s\| \leq \mathrm{Size}(\varphi)^2$ – in fact $\|s\| \leq \mathrm{Size}(\varphi)$ if x is of sort $\mathsf{Set(Loc)}$ and $\|s\| \leq \mathrm{Size}(\varphi)^2$ if x is of sort $\mathsf{Loc} \mapsto \mathsf{Data}$. Then there are at most $\mathrm{Size}(\varphi)^2$ recursive calls on line 3 of the $\mathsf{solve}_{\mathsf{T}}$ procedure, that corresponds to an instance of the subformula $\forall x \subseteq s \;.\; \psi$ of $\varphi \triangleleft [\ell, \mathsf{pt}, C]\Downarrow$. Since there are at most $\mathrm{Size}(\varphi)$ such subformulae, there are at most $\mathrm{Size}(\varphi)^3$ recursive calls to $\mathsf{solve}_{\mathsf{T}}$ with arguments $\Gamma_0, \ldots, \Gamma_n$, respectively. Moreover, we have $\Gamma_0 = \{\varphi \triangleleft [\ell, \mathsf{pt}, C]\Downarrow\}$, thus $\mathrm{Size}(\Gamma_0) = \mathcal{O}(\mathrm{Size}(\varphi))$ and for each $i = 0, \ldots, n-1$, $\mathrm{Size}(\Gamma_{i+1}) \leq \mathrm{Size}(\Gamma_i) + \mathrm{Size}(\varphi)^2$, because a set of formulae $\lfloor A \Rightarrow \psi[\mathbf{t}/\mathbf{x}]\rfloor^*$ of size at most $\mathrm{Size}(\varphi)^2$ is added to Γ_i. Because T-satisfiability is in PSPACE, by the hypothesis, the checks at lines 1 and 2 can be done within space bounded by a polynomial in $\mathrm{Size}(\varphi)$, thus the space needed by $\mathsf{solve}_{\mathsf{SL(T)}}(\varphi)$ is also bounded by a polynomial in $\mathrm{Size}(\varphi)$. Hence the (SL, T)-satisfiability problem is in PSPACE. $\square$

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 examples.

Example 6. Consider the $\mathsf{SL}(T)$ formula $\varphi \equiv (y \mapsto a * z \mapsto b) \wedge \neg(\top * y \mapsto a)$ from Example 3. When running $\mathsf{solve}_{\mathsf{SL(T)}}(\varphi)$, we first compute the set $C = \{c_1, c_2\}$, and introduce fresh symbols ℓ and pt of sorts $\mathsf{Set(Loc)}$ and $\mathsf{Loc} \rightarrow \mathsf{Data}$ respectively. Let us define:

$$\begin{array}{rcll}\psi_1 & \equiv & \ell_{11} \cup \ell_{12} \subseteq C \cup \{y,z\} \Rightarrow \neg(& \ell_{11} \cap \ell_{12} \approx \emptyset \wedge \ell \approx \ell_{11} \cup \ell_{12} \wedge \\ & & & \ell_{11} \approx \{y\} \wedge \mathsf{pt}(y) \approx a \wedge y \not\approx \mathsf{nil} \\ & & & \ell_{12} \approx \{z\} \wedge \mathsf{pt}(z) \approx b \wedge z \not\approx \mathsf{nil}) \\ \psi_2 & \equiv & \ell_2 \subseteq C \cup \{y\} \Rightarrow \neg(& \ell_2 \approx \{y\} \wedge \mathsf{pt}(y) \approx a \wedge y \not\approx \mathsf{nil})\end{array}$$

The formula $(\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow$ is $\neg\forall \ell_{11}\ell_{12} .\psi_1 \wedge \forall \ell_2.\psi_2$. Let $(A_i, \mathbf{k}_i) \leftrightharpoons \forall \mathbf{x}.\psi_1$ for $i = 1,2$. We call the subprocedure $\mathsf{solve}_{\mathsf{T}}$ on Γ_0, where:

$$\Gamma_0 \equiv \lfloor (\varphi \triangleleft [\ell, \mathsf{pt}, C])\Downarrow \rfloor^* \equiv \{\neg A_1 \wedge A_2, \neg A_1 \Rightarrow \neg\psi_1[k_{11}, k_{12}/\ell_{11}, \ell_{12}], \neg A_2 \Rightarrow \neg\psi_2[k_2/\ell_2]\}.$$

The set Γ_0 is T-satisfiable with a model $\mathcal{I}_0$ where $A_1^{\mathcal{I}_0} = \bot$ and $A_2^{\mathcal{I}_0} = \top$. Step 2 of the procedure then checks whether $A_2, \Gamma_0 \models_T \phi$, where $\phi \equiv \lfloor \psi_2[k_2/\ell_2] \rfloor \equiv \psi_2[k_2/\ell_2]$. This is not the case, since a model $\mathcal{J}$ exists for $\Gamma_0 \cup \{A_2, \neg\phi\}$. Let $t = \{y\}$, where note that $t \subseteq C \cup \{y\}$, and $t^{\mathcal{J}} = k_2^{\mathcal{J}}$ since it must be the case that $\mathcal{J} \models_T k_2 \approx \{y\}$. Thus, Step 3 of the procedure recursively invokes $\mathsf{solve}_{\mathsf{T}}$ on Γ_1, where:

$$\Gamma_1 \equiv \Gamma_0 \cup \lfloor A_2 \Rightarrow \psi_2[\{y\}/\ell_2] \rfloor^* \equiv \Gamma_0 \cup \{A_2 \Rightarrow \neg(\mathsf{pt}(y) \approx a \wedge y \not\approx \mathsf{nil})\}$$

We have that Γ_1 is T-unsatisfiable by noting that since $\neg A_1 \wedge A_2 \in \Gamma_1$, we have $(\mathsf{pt}(x) \approx a \wedge y \not\approx \mathsf{nil})$ must be true due to $\neg A_1 \Rightarrow \neg\psi_1[k_{11}, k_{12}/\ell_{11}, \ell_{12}]$, and false due to $A_2 \Rightarrow \neg(\mathsf{pt}(y) \approx a \wedge y \not\approx \mathsf{nil})$. In other words, this establishes that the heap $\{y \mapsto a, z \mapsto b\}$ can be split into disjoint heaps h_1 and $h_2 = \{y \mapsto a\}$. Since this heap is the only possible model for $y \mapsto a * z \mapsto b$ and it is not a model for $\neg(\top * y \mapsto a)$, the conjunction φ is unsatisfiable. ■

Example 7. Modifying the previous example, let φ' be the $\mathsf{SL}(T)$ formula $a \not\approx b \wedge (y \mapsto a * z \mapsto b) \wedge \neg(\top * y \mapsto b)$, and let:

$$\psi_3 \;=\; \ell_3 \subseteq C \cup \{y\} \Rightarrow \;\neg(\ell_3 \approx \{y\} \wedge \mathsf{pt}(y) \approx b \wedge y \not\approx \mathsf{nil})$$

Let $(A_3, k_3) \leftrightharpoons \forall \mathbf{x}.\psi_3$. We call the subprocedure $\mathsf{solve}_{\mathsf{T}}$ on Γ_0, where in this case:

$$\Gamma_0 \equiv \lfloor (\varphi' \triangleleft [\ell, \mathsf{pt}, C])\Downarrow \rfloor^* \equiv \{a \not\approx b \wedge \neg A_1 \wedge A_3, \neg A_1 \Rightarrow \psi_1[k_{11}, k_{12}/\ell_{11}, \ell_{12}], \neg A_3 \Rightarrow \psi_3[k_3/\ell_3]\}.$$

The set Γ_0 is T-satisfiable with a model $\mathcal{I}_0$ where $A_1^{\mathcal{I}_0} = \bot$ and $A_3^{\mathcal{I}_0} = \top$. Step 2 of the procedure then checks whether $A_3, \Gamma_0 \models_T \phi$, where $\phi = \lfloor \psi_3[k_3/\ell_3] \rfloor = \psi_3[k_3/\ell_3]$. This is the case, since $A_3 \cup \Gamma_0 \cup \neg\phi$ is T-unsatisfiable due to the fact that they entail $\mathsf{pt}(y) \approx a \wedge \mathsf{pt}(y) \approx b \wedge a \not\approx b$. We conclude that φ' is (SL, T)-satisfiable. In other words, this establishes that the heap $\{y \mapsto a, z \mapsto b\}$ cannot be split into disjoint heaps h_1 and $h_2 = \{y \mapsto b\}$ when $a \not\approx b$. ■

Example 8. Consider the $\mathsf{SL}(T)$ formula $\varphi \equiv \mathsf{emp} \wedge (y \mapsto 0 \multimap y \mapsto 1) \wedge y \not\approx \mathsf{nil}$. When running $\mathsf{solve}_{\mathsf{SL(T)}}(\varphi)$, we first compute the set $C = \{c\}$, and introduce fresh symbols ℓ

and pt of sorts $\mathsf{Set(Loc)}$ 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:

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

Let $(A_4, (k_1, k_2)) \leftrightharpoons \forall \mathbf{x}.\psi_4$. 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 $\mathcal{J}$ exists 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:

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

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

Although not demonstrated in these examples, our approach is capable of handling $\mathsf{SL}(T)$ inputs involving T-constraints on both locations and data. For instance, modifying Example 7 slightly, assuming Data is Int, and T subsumes the theory of linear integer arithmetic LIA, our approach would by the same reasoning determine that $a = b + 1 \wedge (x \mapsto a * y \mapsto b) \wedge \neg(\top * x \mapsto b)$ is (SL, T)-satisfiable as a consequence of the fact that, all models of LIA must interpret a and b as distinct values, due to the constraint $a = b + 1$. In a similar vein, when considering Loc to be Int, the formula $-1 < x - y < 1 \wedge (x \mapsto a * y \mapsto b)$ is unsatisfiable, because the constraint $-1 < x - y < 1$ is only satisfied when x and y are interpreted by the same value, which conflicts with the separation constraint $x \mapsto a * y \mapsto b$.

5.1 Integration in DPLL(T)

We have implemented the algorithm described in this section within the SMT solver CVC4 [3]. Our implementation accepts an extended syntax of SMT lib version 2 format [4] 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 [9].

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:

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

which we call *spatial functions* [10]. 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 $\{\mathsf{emp}, \mapsto, *^n, -\!\!\circ\}$ 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*, where:

- Nested applications of $*$ are flattened, and
- Some of the subformulae of $*$ not containing spatial functions are *lifted*. For instance, we replace $((x \mapsto y \wedge t \approx u) * z \mapsto w)$ with $t \approx u \wedge (x \mapsto y * z \mapsto w)$.

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:

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

where each ℓ_i is a fresh free constant, and $\varphi_i[\ell_i]$ denotes the result of replacing each spatial atom a in φ_i with $\mathsf{lbl}(a, \ell_i)$. These formulae are added eagerly when such literals

[9] The SL functions $\mapsto, *, -\!\!\circ$ are not symbols belonging to a first-order theory, and thus this module is not a theory solver in the standard sense.

[10] 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.

are added to M_k. To handle negated $*$-atoms and positive $-\!\!\circledast$-atoms, the SL solver adds to Γ formulae based on the criteria:

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

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 [11]. 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.

6 Evaluation

Experimental evaluation was carried out on four types of benchmarks, reported in Figure 4. All benchmarks are instantiations of different patterns, resulting in formulae of increasing size. For instance, the unfold$n$-sat, nested$n$-sat and chain$n$-sat benchmarks (as well as their unsat versions) are entailment checks between n-times unfolding of the inductive predicates given below:

	$\mathsf{pos}(x,a)$	$\Rightarrow$ $\mathsf{neg}(x,a)$
unfold-sat	$x \mapsto a \vee \exists y \exists b\ .\ x \mapsto a * \mathsf{pos}(y,b)$	$\neg x \mapsto a \vee \exists y \exists b\ .\ x \mapsto a * \mathsf{neg}(y,b)$
unfold-unsat	$x \mapsto a \vee \exists y \exists b\ .\ x \mapsto a * \mathsf{pos}(y,b)$	$x \mapsto a \vee \exists y \exists b\ .\ \neg x \mapsto a * \mathsf{neg}(y,b)$
nested-sat	$x \mapsto a \vee \exists y \exists b\ .\ x \mapsto a * \mathsf{pos}(y,b)$	$x \mapsto a \vee \exists y \exists b\ .\ x \mapsto a * \neg\mathsf{neg}(y,b)$
nested-unsat	$x \mapsto a \vee \exists y \exists b\ .\ x \mapsto a * \mathsf{pos}(y,b)$	$x \mapsto a \vee \exists y \exists b\ .\ \neg x \mapsto a * \neg\mathsf{neg}(y,b)$
chain-sat	$x \mapsto a \vee \exists y\ .\ x \mapsto a * \mathsf{pos}(a,y)$	$\neg x \mapsto a \vee \exists y\ .\ x \mapsto a * \mathsf{neg}(a,y)$
chain-unsat	$x \mapsto a \vee \exists y\ .\ x \mapsto a * \mathsf{pos}(a,y)$	$x \mapsto a \vee \exists y\ .\ \neg x \mapsto a * \mathsf{neg}(a,y)$

The wand-n-k benchmarks are instantiations of entailments between $P_1 * P_2 \Rightarrow P_3$ and $P_1 \Rightarrow P_2 \multimap P_3$, where P_1 and P_3 are chains of $*$-separated points-to predicates of lengths n and k, respectively, and P_2's are points-to predicates.

We tested our implementation in CVC4 (version 1.5 prerelease) on these benchmarks. The runtimes of CVC4 are shown in Figure 4. All experiments were run on a 2.80GHz Intel(R) Core(TM) i7 CPU machine with with 8MB of cache.[12] For a majority of benchmarks, the runtime of CVC4 is quite low.

7 Conclusion

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

[11] For details, see Section 5 of [20].

[12] The examples and CVC4 binary can be found at http://cvc4.cs.nyu.edu/papers/CAV2016-seplog/.

benchmark	result	time	benchmark	result	time
chain1-sat	sat	0.07	nested1-unsat	sat	0.06
chain1-unsat	unsat	0.02	nested2-sat	sat	0.3
chain2-sat	sat	0.04	nested2-unsat	unsat	0.16
chain2-unsat	unsat	0.14	nested3-unsat	sat	0.54
chain4-sat	sat	0.25	nested4-sat	sat	31.38
chain4-unsat	unsat	0.57	nested4-unsat	unsat	1.18
chain8-sat	sat	2.72	nested8-sat	T/O	–
chain8-unsat	unsat	4.78	nested8-unsat	unsat	117.09
benchmark	**result**	**time**	**benchmark**	**result**	**time**
unfold1-sat	sat	0.03	wand-3-5	unsat	0.01
unfold1-unsat	unsat	0.05	wand-3-6	unsat	0.01
unfold2-sat	sat	0.07	wand-3-9	unsat	0.01
unfold2-unsat	unsat	0.11	wand-4-6	unsat	0.01
unfold4-sat	sat	0.32	wand-4-7	unsat	0.01
unfold4-unsat	unsat	0.41	wand-4-10	unsat	0.01
unfold8-sat	sat	2.61	wand-7-9	unsat	0.01
unfold8-unsat	unsat	3.55	wand-7-10	unsat	0.01
unfold10-sat	sat	6.15	wand-7-13	unsat	0.01

Fig. 4. Results of CVC4 for $\mathsf{SL}(T)$ benchmarks with a 300 second timeout.

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. For future work, we would like to extend the approach to be applicable to cases where Loc may have a finite interpretation, and improve the heuristics used by our decision procedure for choosing quantifier instantiations, in particular for cases with many quantifier alternations.

References

1. Bansal, K.: Decision Procedures for Finite Sets with Cardinality and Local Theory Extensions. Ph.D. thesis, New York University (2016)
2. Bansal, K., Reynolds, A., King, T., Barrett, C., Wies, T.: Deciding local theory extensions via e-matching. In: Computer Aided Verification, pp. 87–105. Springer International Publishing (2015)
3. Barrett, C., Conway, C., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Computer Aided Verification (CAV). Springer (2011)
4. Barrett, C., Fontaine, P., Tinelli, C.: The SMT-LIB standard—Version 2.5. Tech. rep., The University of Iowa (2015), available at http://smt-lib.org/
5. Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P., Wies, T., Yang, H.: Shape analysis for composite data structures. In: Proc. CAV'07. LNCS, vol. 4590. Springer (2007)
6. Bjørner, N., Janota, M.: Playing with quantified satisfaction
7. Brotherston, J., Gorogiannis, N., Petersen, R.L.: A generic cyclic theorem prover. In: APLAS. pp. 350–367 (2012)
8. 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. Springer (2011)

9. Calcagno, C., Yang, H., Ohearn, P.W.: Computability and complexity results for a spatial assertion language for data structures. In: FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science, pp. 108–119. Springer Berlin Heidelberg (2001)
10. Enea, C., Sighireanu, M., Wu, Z.: On automated lemma generation for separation logic with inductive definitions. In: Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings. 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: Computer aided verification, pp. 175–188. Springer (2004)
13. Ge, Y., de Moura, L.: Complete instantiation for quantified formulas in satisfiability modulo theories. In: Proceedings of CAV'09. LNCS, vol. 5643. Springer (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. ACM (2001)
16. Navarro Pérez, J.A., Rybalchenko, A.: Separation logic + superposition calculus = heap theorem prover. SIGPLAN Not. 46(6), 556–566 (Jun 2011), http://doi.acm.org/10.1145/1993316.1993563
17. Nguyen, H.H., Chin, W.N.: Enhancing program verification with lemmas. In: Proc of CAV'08. LNCS, vol. 5123. Springer (2008)
18. Piskac, R., Wies, T., Zufferey, D.: Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings, chap. Automating Separation Logic Using SMT, pp. 773–789. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
19. Reynolds, A., Deters, M., Kuncak, V., Barrett, C.W., Tinelli, C.: Counterexample guided quantifier instantiation for synthesis in CVC4. In: CAV. Springer (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.: Separation Logic: A Logic for Shared Mutable Data Structures. In: Proc. of LICS'02. IEEE CS Press (2002)
22. Suter, P., Steiger, R., Kuncak, V.: Sets with cardinality constraints in satisfiability modulo theories. In: Verification, Model Checking, and Abstract Interpretation, pp. 403–418. Springer Berlin Heidelberg (2011)
23. Yang, H.: Local Reasoning for Stateful Programs. Ph.D. thesis, University of Illinois at Urbana-Champaign (2001)