EPiC Series in Computing
Volume 73, 2020, Pages 390–408
LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Beyond Symbolic Heaps: Deciding Separation Logic with Inductive Definitions

Jens Pagel and Florian Zuleger

[1] TU Wien, Vienna, Austria
pagel@forsyte.at
[2] TU Wien, Vienna, Austria
zuleger@forsyte.at

Abstract

Symbolic-heap separation logic with inductive definitions is a popular formalism for reasoning about heap-manipulating programs. The fragment $\mathrm{SLID}_{\mathsf{btw}}$ introduced by Iosif, Rogalewicz and Simacek, is one of the most expressive fragments with a decidable entailment problem. In recent work, we improved on the original decidability proof by providing a direct model-theoretic construction, obtaining a 2-Exptime upper bound. In this paper, we investigate separation logics built on top of the inductive definitions from $\mathrm{SLID}_{\mathsf{btw}}$, i.e., logics that feature the standard Boolean and separation-logic operators. We give an almost tight delineation between decidability and undecidabilty. We establish the decidability of the satisfiability problem (in 2-Exptime) of a separation logic with conjunction, disjunction, separating conjunction and guarded forms of negation, magic wand, and septraction. We show that any further generalization leads to undecidabilty (under mild assumptions).

1 Introduction

Separation logics (SL) are a popular formalism for reasoning about the usage of *dynamic resources* [30, 25], and have been widely used in static analysis [11, 10], automated verification [5, 7, 13, 19, 29, 24, 31], and interactive verification [2, 21]. SLs extend first-order logic with so-called *separating connectives*, most importantly the *separating conjunction* $\star$, in order to be able to split resources and reason about program parts in isolation.

In this article, we study SLs for reasoning about the heap structures arising in heap-manipulating programs. Here, the SL formula $\phi_1 \star \phi_2$ specifies that a heap can be split into two disjoint sub-heaps that satisfy ϕ_1 and ϕ_2. In addition to the separating connectives, SLs for the heap usually feature points-to predicates and predicates for unbounded data-structures such as lists, trees, etc. While satisfiability and related problems for separation logic are undecidable in general, many decidable fragments have been proposed. Most decidability results have been obtained for the *symbolic-heap fragment* [6, 4, 16, 9, 17, 32, 20, 33]. Symbolic heaps are separation-logic formulas in which atomic predicates can only be combined with the separating conjunction; no other separating connectives or Boolean connectives are allowed. The decidability results fall into two categories: Logics with built-in predicates—usually for expressing linked lists and/or trees [4, 14, 27, 28, 22]; and logics with user-defined inductive definitions (SLID).

$$
\begin{array}{lcl}
\mathsf{odd}(x_1, x_2) & \Leftarrow & x_1 \mapsto x_2 \\
\mathsf{odd}(x_1, x_2) & \Leftarrow & \exists y.\, (x_1 \mapsto y) \star \mathsf{even}(y, x_2) \\
\mathsf{even}(x_1, x_2) & \Leftarrow & \exists y.\, (x_1 \mapsto y) \star \mathsf{odd}(y, x_2) \\
\mathsf{lseg}(x_1, x_2) & \Leftarrow & x_1 \mapsto x_2 \\
\mathsf{lseg}(x_1, x_2) & \Leftarrow & \exists y.\, (x_1 \mapsto y) \star \mathsf{lseg}(y, x_2)
\end{array}
$$

In SLID, users specify the shape of data structures by a set of recursive definitions. For example, the definitions on the right define segments of singly-linked lists of odd length, even length, and length at least one. Importantly, the formulae on the right-hand side of $\Leftarrow$ in the SLID definitions are restricted to be symbolic heaps, i.e., formulae built from atomic predicates (here $(\cdot \mapsto \cdot)$, odd, even, and lseg) and the separating conjunction $\star$. The satisfiability problem for SLID is decidable in exponential time [9]. The entailment problem, which is crucial for Hoare-style deductive verification, is, however, undecidable in general [1]. Consequently, restricted fragments of SLID definitions have been studied. While it is natural to consider restrictions to trees [17, 32, 33], it is possible to obtain decidability results for more expressive logics. Iosif et al. [16] proved the decidability of a particularly expressive fragment of SLID. The fragment, for example, supports the definition of binary trees whose leaves form a linked list (which might be used to implement a sorted set data structure). Following [16], we denote this logic by $\text{SLID}_{\mathsf{btw}}$, where the subscript alludes to the fact that that all models of $\text{SLID}_{\mathsf{btw}}$ formulas (when viewed as graphs) are of bounded treewidth (BTW). The original decidability result [16] was obtained by a reduction to the satisfiability problem of monadic second-order logic over graphs of BTW. Deciding the satisfiability via this reduction would involve a blowup of several exponentials and therefore seems impractical to implement [17]. In recent work [26], we improved on the original decidability proof by providing a direct model-theoretic construction, obtaining a 2-Exptime upper bound and promising experimental results. A matching lower bound has been announced in [15].

In this article, we study SLs that go beyond the symbolic-heap fragment. We consider SLs that feature the standard Boolean and separation-logic connectives in addition to atomic $\text{SLID}_{\mathsf{btw}}$ predicates. In these logics, the predicates (such as odd and even above) must still be defined using symbolic heaps, but the SL formulae that are built on top of atomic predicates can use other classical and separating connectives. For example, our decidability result allows discharging entailment queries such as $(\mathsf{lseg}(x, y) \wedge (\mathsf{even}(y, \mathsf{nil}) \mathrel{-\!\!\star} \mathsf{odd}(x, \mathsf{nil}))) \wedge \neg(x \mapsto y) \models_\Phi \mathsf{odd}(x, y)$. Such queries naturally arise in Hoare-style verification, where conditions require reasoning about the Boolean connectives ($\wedge$, $\vee$ and $\neg$) and weakest pre-condition computation requires reasoning about the magic-wand $\mathrel{-\!\!\star}$ [18, 30]. In this paper, we are concerned with the fundamental question what is decidable about an SL that is built on top of the inductive definitions from $\text{SLID}_{\mathsf{btw}}$.

Inspired by work on *guarded* first-order logic [3], we propose a guarded fragment of separation logic, $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$. Formulas in $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ may combine user-defined predicates from $\text{SLID}_{\mathsf{btw}}$ with the separating conjunction $\star$, classical conjunction $\wedge$ and classical $\vee$. Moreover, we allow negation $\neg$, magic wand $\mathrel{-\!\!\star}$ and septraction $\circledast$ [8] to appear in formulas of the form $\phi \wedge \neg\psi$, $\phi \wedge (\psi \mathrel{-\!\!\star} \zeta)$, and $\phi \wedge (\psi \mathbin{\circledast} \zeta)$. Here, ϕ acts as a *guard* on the operators; hence the name.

Contributions. Our two main technical results are as follows:

- We show that further generalizing $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ by allowing any one of the three operators $\neg$, $\mathrel{-\!\!\star}$ and $\circledast$ in an unguarded form yields an undecidable logic.

- We then prove that our approach for satisfiability- and entailment checking from [26] can be lifted to $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$, obtaining a 2-Exptime decision procedure for guarded separation logic. We thus obtain the first decidability result for a separation logic with support both for unbounded data structures and for the magic wand.

We thus obtain an (almost perfectly) tight delineation between what is decidable and undecidable in SLs with inductive definitions from $\mathrm{SLID}_{\mathsf{btw}}$; the only cases left open are whether the undecidability results can be tightened to proper subsets of $\{\wedge, *, \mathsf{t}\}$, $\{\wedge, *, \neg\}$ and $\{\wedge, *, -\!\!\star\}$.

Outline. We introduce SL with inductive definitions and its guarded fragment $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ in Section 2. We show that all extensions of $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ are undecidable in Section 3. In Section 4, we introduce the abstraction that underlies our decision procedure for $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$. We present the decidability result in Section 5 and conclude in Section 6. All proofs omitted in Sections 4 and 5 are in our technical report [26].

2 Separation Logic with Inductive Definitions

Preliminaries. We denote by $|X|$ the cardinality of the set X. Let f be a (partial) function. Then $\mathrm{dom}(f)$ and $\mathrm{img}(f)$ denote the domain and image of f, respectively. We frequently use set notation to define and reason about partial functions. For example, $f := \{x_1 \mapsto y_1, \ldots, x_k \mapsto y_k\}$ is the partial function that maps x_i to y_i, $1 \leq i \leq k$, and is undefined on all other values; $f \cup g$ is the union of partial functions f and g; and $f \subseteq g$ means $f(x) = g(x)$ for all $x \in \mathrm{dom}(f)$. Sets and ordered sequences are denoted in boldface, e.g., $\mathbf{x}$. To list the elements of a sequence, we write $\langle x_1, \ldots, x_k \rangle$. We shorten $\langle x \rangle$ to x to reduce clutter. The empty sequence is ε, the concatenation of $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{x} \cdot \mathbf{y}$. We lift functions to sequences, i.e, $f(\langle x_1, \ldots, x_k \rangle) := \langle f(x_1), \ldots, f(x_k) \rangle$.

Syntax of separation logic. Let $\mathbf{Var}$ denote an infinite set of *variables*, with $\mathsf{nil} \in \mathbf{Var}$. We assume a set $\mathbf{Preds}$ of *predicate identifiers*. Each predicate $\mathsf{pred} \in \mathbf{Preds}$ is equipped with an arity $\mathsf{ar}(\mathsf{pred}) \in \mathbb{N}$, representing the number of parameters to be passed to the predicate. The semantics of such predicates are defined by means of inductive definitions, introduced later.

The grammar in Fig. 1 defines three variants of separation logic (SL) with inductive definitions: *guarded* SL, formulas of the form ϕ_{g}, collected in the set $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$; quantifier-free SL, formulas of the form ϕ_{qf}, collected in $\mathbf{SL}^{\mathsf{qf}}_{\mathsf{btw}}$; and *existentially-quantified symbolic heaps*, collected in $\mathbf{SH}^{\exists}$.

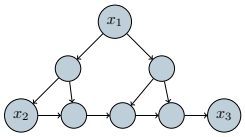
In Fig. 1, $\mathsf{pred} \in \mathbf{Preds}$ is a predicate identifier, $x, y \in \mathbf{Var}$ are variables, and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{Var}^*$ are (possibly empty) sequences of variables with $|\mathbf{x}| = \mathsf{ar}(\mathsf{pred})$. The first line of Fig. 1 defines the atomic formulas, τ, common to all SL variants studied in this article. $\mathbf{emp}$ is the *empty heap*, $x \mapsto \mathbf{y}$ asserts that x *points to* $\mathbf{y}$, $x \approx y$ asserts the equality between variables x and y, and $x \not\approx y$ asserts the disequality of x and y. Guarded formulas, ϕ_{g}, are built from atomic formulas using the *separating conjunction* $\star$, conjunction $\wedge$, disjunction $\vee$, *guarded negation* $\phi_{\mathsf{g}} \wedge \neg\phi_{\mathsf{g}}$, *guarded septraction* $\phi_{\mathsf{g}} \wedge (\phi_{\mathsf{g}} -\!\!\circledast \phi_{\mathsf{g}})$, and *guarded magic wands* $\phi_{\mathsf{g}} \wedge (\phi_{\mathsf{g}} -\!\!\star \phi_{\mathsf{g}})$. In quantifier-free formulas, ϕ_{qf}, all operators may occur unguarded and we include an additional atom t representing true. Finally, ϕ_{sh} formulas are existentially-quantified symbolic heaps. We use $\star$ instead of $\wedge$ in the pure constraint, Π, because in our semantics, (dis)equalities only hold in empty heaps.

$$
\begin{array}{lcl}
\tau & ::= & \mathbf{emp} \mid x \mapsto \mathbf{y} \mid \mathsf{pred}(\mathbf{x}) \mid x \approx y \mid x \not\approx y \\
\phi_{\mathsf{g}} & ::= & \tau \mid \phi_{\mathsf{g}} \star \phi_{\mathsf{g}} \mid \phi_{\mathsf{g}} \wedge \phi_{\mathsf{g}} \mid \phi_{\mathsf{g}} \vee \phi_{\mathsf{g}} \mid \phi_{\mathsf{g}} \wedge \neg\phi_{\mathsf{g}} \mid \phi_{\mathsf{g}} \wedge (\phi_{\mathsf{g}} -\!\!\circledast \phi_{\mathsf{g}}) \mid \phi_{\mathsf{g}} \wedge (\phi_{\mathsf{g}} -\!\!\star \phi_{\mathsf{g}}) \\
\phi_{\mathsf{qf}} & ::= & \tau \mid \mathsf{t} \mid \phi_{\mathsf{qf}} \star \phi_{\mathsf{qf}} \mid \phi_{\mathsf{qf}} -\!\!\star \phi_{\mathsf{qf}} \mid \phi_{\mathsf{qf}} -\!\!\circledast \phi_{\mathsf{qf}} \mid \phi_{\mathsf{qf}} \wedge \phi_{\mathsf{qf}} \mid \phi_{\mathsf{qf}} \vee \phi_{\mathsf{qf}} \mid \neg\phi_{\mathsf{qf}} \\
\phi_{\mathsf{sh}} & ::= & \exists \mathbf{e}.\, (x_1 \mapsto \mathbf{y}_1) \star \cdots \star (x_k \mapsto \mathbf{y}_k) \star \mathsf{pred}_1(\mathbf{z}_1) \star \cdots \star \mathsf{pred}_l(\mathbf{z}_l) \star \Pi, \\
 & & \quad \text{where } \Pi ::= a_1 \approx b_1 \star \cdots \star a_m \approx b_m \star c_1 \not\approx d_1 \star \cdots \star c_n \not\approx d_n
\end{array}
$$

Figure 1: The syntax of the separation-logic fragments studied in this article: Guarded formulas ϕ_{g}; quantifier-free formulas ϕ_{qf}; and existentially-quantified symbolic heaps, ϕ_{sh}.

$$\begin{aligned}\mathsf{tll}(x_1,x_2,x_3) &\Leftarrow x_1 \mapsto (\mathsf{nil},\mathsf{nil},x_3) \star x_1 = x_2\\ \mathsf{tll}(x_1,x_2,x_3) &\Leftarrow \exists l,r,m.\, x_1 \mapsto (l,r,\mathsf{nil})\\ &\qquad \star \mathsf{tll}(l,x_2,m) \star \mathsf{tll}(r,m,x_3)\end{aligned}$$

(a) The definition of the tll predicate representing trees with linked leaves.



(b) A model of $\mathsf{tll}(x_1,x_2,x_3)$.

Figure 2: A system of inductive definition (SID) defining trees with linked leaves.

We denote by $\mathbf{SL}(\cdot_1,\dots,\cdot_k)$ the restriction of $\mathbf{SL}^{\mathsf{qf}}_{\mathsf{btw}}$ to formulas built from τ and the additional symbols and operators $\cdot_1,\dots,\cdot_k$. For example, $\mathbf{SL}(\wedge,\star,\mathsf{t})$ is the SL in which formulas are built from atomic predicates τ, additional predicate t and binary operators $\star$, $\wedge$.

Additional notation. We write $\phi[\langle x_1,\dots,x_k\rangle / \langle y_1,\dots,y_k\rangle]$ for the formula obtained from ϕ by instantiating every occurrence of x_i with y_i. We sometimes write $\bigstar_{1\le i\le n}\phi_i$ to denote $\phi_1 \star \dots \star \phi_n$. For $n=0$, this expression evaluates to the neutral element of $\star$, **emp**. For an atomic formula τ and a formula ϕ, $\tau \in \phi$ denotes that τ occurs in ϕ. The size of a formula ϕ, $|\phi|$, is the sum of the number of atoms, the number of unary operators and binary operators, and the number of quantifiers in the formula. Finally, $\mathsf{fvars}(\phi)$ is the set of all free variables in ϕ.

Inductive definitions. Predicates are defined by a *system of inductive definitions* (SID). An SID is a finite set Φ of *rules* of the form $\mathsf{pred}(\mathbf{x}) \Leftarrow \phi$, where $\mathsf{pred} \in \mathbf{Preds}$ is a predicate symbol, $\mathbf{x}$ are the *parameters* of pred, and $\phi \in \mathbf{SH}^{\exists}$ is an existentially-quantified symbolic heap as defined in Fig. 1 [16, 1]. We assume that all rules of the same predicate pred have the same parameters. We collect these *free variables* of pred in the set $\mathsf{fvars}(\mathsf{pred})$. The size of an SID Φ, $|\Phi|$, is the sum of the sizes of the formulas in its rules.

Example 2.1. *The predicates* odd *and* even *in Section 1 define all lists of odd and even length, respectively. The predicate* tll *in Fig. 2a defines binary trees whose leaves are connected in a singly-linked list (TLL). The parameters correspond to the root, the left-most leaf (x_2) and the successor of the right-most leaf (x_3). Every node of a TLL contains three pointer fields: The left successor, the right successor (non-null at inner nodes, null at leaves), and the next leaf (null at inner nodes, non-null at the leaves). We show a graphical representation of a TLL in Fig. 2b.*

Semantics. We use the standard *stack–heap semantics* of separation logic [30]. Let **Loc** be an infinite set of *locations*. A *stack* is a finite partial function $\mathfrak{s}\colon \mathbf{Var} \rightharpoonup \mathbf{Loc}$. A *heap* is a finite partial function $\mathfrak{h}\colon \mathbf{Loc} \rightharpoonup \mathbf{Loc}^+$. A *model* is a stack–heap pair $(\mathfrak{s},\mathfrak{h})$ with $\mathfrak{s}(\mathsf{nil}) \notin \mathrm{dom}(\mathfrak{h})$.

We define the set of *allocated variables* of a model, $\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) := \{x \mid \mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{h})\}$. We denote by $\mathfrak{h}_1 + \mathfrak{h}_2$ the disjoint union of the heaps $\mathfrak{h}_1,\mathfrak{h}_2$. If $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) \neq \emptyset$, then $\mathfrak{h}_1+\mathfrak{h}_2$ is undefined. We let $\mathsf{locs}(\mathfrak{h}) := \mathrm{dom}(\mathfrak{h}) \cup \bigcup \mathrm{img}(\mathfrak{h})$ and $\mathsf{dangling}(\mathfrak{h}) := \{l \in \bigcup \mathrm{img}(\mathfrak{h}) \mid l \notin \mathrm{dom}(\mathfrak{h})\}$.

Figure 3 defines the semantics of separation logic formulas ϕ w.r.t. a fixed SID Φ. In the semantics of equalities and disequalities, we follow [27, 22] and require that the heap is empty. This ensures that t is not definable in guarded formulas (e.g., as $x \approx x$). Observe that we use a *precise* [12] semantics of the points-to assertion: $(\mathfrak{s},\mathfrak{h}) \models_\Phi x \mapsto \mathbf{y}$ holds only in single-pointer heaps. A heap is the model of a predicate call $\mathsf{pred}(\mathbf{y})$ iff it is the model of a rule of the predicate once the free variables of the rule, $\mathbf{x}$, have been replaced by the actual arguments, $\mathbf{y}$. Our semantics of predicates is equivalent to the least fixed-point semantics as used e.g. in [9].

As usual, $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1 \star \phi_2$ iff $\mathfrak{h}$ can be split into disjoint heaps that are models of ϕ_1 and ϕ_2; $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1 \mathrel{-\!\!\star} \phi_2$ iff extending $\mathfrak{h}$ with a model of ϕ_1 always yields a model of ϕ_2, provided the

$(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathbf{emp}$	iff	$\mathrm{dom}(\mathfrak{h}) = \emptyset$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi x \approx y$	iff	$\mathrm{dom}(\mathfrak{h}) = \emptyset$ and $\mathfrak{s}(x) = \mathfrak{s}(y)$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi x \not\approx y$	iff	$\mathrm{dom}(\mathfrak{h}) = \emptyset$ and $\mathfrak{s}(x) \neq \mathfrak{s}(y)$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi x \mapsto \langle y_1,\dots,y_k\rangle$	iff	$\mathfrak{h} = \{\mathfrak{s}(x) \mapsto \langle \mathfrak{s}(y_1),\dots,\mathfrak{s}(y_k)\rangle\}$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{pred}(\mathbf{y})$	iff	$(\mathfrak{s},\mathfrak{h}) \models_\Phi \psi[\mathbf{x}/\mathbf{y}]$ for some $(\mathsf{pred}(\mathbf{x}) \Leftarrow \psi) \in \Phi$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{t}$		always
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1 \star \phi_2$	iff	ex. $\mathfrak{h}_1,\mathfrak{h}_2$ s.t. $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2, (\mathfrak{s},\mathfrak{h}_1) \models_\Phi \phi_1$ and $(\mathfrak{s},\mathfrak{h}_2) \models_\Phi \phi_2$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1 \mathrel{-\!\!\star} \phi_2$	iff	for all $\mathfrak{h}_1,\mathfrak{h}_2$, if $\mathfrak{h}_2 = \mathfrak{h}_1 + \mathfrak{h}$ and $(\mathfrak{s},\mathfrak{h}_1) \models_\Phi \phi_1$ then $(\mathfrak{s},\mathfrak{h}_2) \models_\Phi \phi_2$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1 \circledast \phi_2$	iff	there ex. $\mathfrak{h}_1$ s.t. $(\mathfrak{s},\mathfrak{h}_1) \models_\Phi \phi_1$ and $(\mathfrak{s},\mathfrak{h}+\mathfrak{h}_1) \models_\Phi \phi_2$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1 \wedge \phi_2$	iff	$(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1$ and $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_2$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1 \vee \phi_2$	iff	$(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1$ or $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_2$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \neg\phi_1$	iff	$(\mathfrak{s},\mathfrak{h}) \not\models_\Phi \phi_1$
$(\mathfrak{s},\mathfrak{h}) \models_\Phi \exists e.\, \phi$	iff	exists $v \in \mathbf{Loc}$ s.t. $(\mathfrak{s} \cup \{e \mapsto v\}, \mathfrak{h}) \models_\Phi \phi$

Figure 3: The semantics of separation logic.

extension is defined; $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi_1 \circledast \phi_2$ if there exists a way to extend $\mathfrak{h}$ with a model of ϕ_1 and obtain a model of ϕ_2. The semantics of the Boolean connectives and quantifiers is standard.

Let ϕ, ψ be SL formulas. We say that ϕ is *satisfiable* w.r.t. SID Φ if there exists a model $(\mathfrak{s},\mathfrak{h})$ such that $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi$. We say that ϕ *entails* ψ w.r.t. Φ, denoted $\phi \models_\Phi \psi$, iff for all models $(\mathfrak{s},\mathfrak{h})$, if $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi$ then $(\mathfrak{s},\mathfrak{h}) \models_\Phi \psi$.

Formulas cannot distinguish between *isomorphic* models.

Definition 2.2. *Let $(\mathfrak{s},\mathfrak{h}), (\mathfrak{s}',\mathfrak{h}')$ be models. $(\mathfrak{s},\mathfrak{h})$ and $(\mathfrak{s}',\mathfrak{h}')$ are* isomorphic*, $(\mathfrak{s},\mathfrak{h}) \cong (\mathfrak{s}',\mathfrak{h}')$, if there exists a bijection $\sigma\colon (\mathsf{locs}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})) \to (\mathsf{locs}(\mathfrak{h}') \cup \mathrm{img}(\mathfrak{s}'))$ such that (1) for all x, $\mathfrak{s}'(x) = \sigma(\mathfrak{s}(x))$ and (2) $\mathfrak{h}' = \{\sigma(l) \mapsto \sigma(\mathfrak{h}(l)) \mid l \in \mathrm{dom}(\mathfrak{h})\}$.*

Lemma 2.3. *Let $\phi \in \mathbf{SL}^{\mathsf{qf}}_{\mathsf{btw}}$. Let $(\mathfrak{s},\mathfrak{h})$ and $(\mathfrak{s}',\mathfrak{h}')$ be models with $(\mathfrak{s},\mathfrak{h}) \cong (\mathfrak{s}',\mathfrak{h}')$. Then $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi$ iff $(\mathfrak{s}',\mathfrak{h}') \models_\Phi \phi$.*

The logic $\mathbf{SLID}_{\mathsf{btw}}$ [16]. In this article, we assume that all SIDs satisfy *progress*, *connectivity*, and *establishment* and denote the resulting restricted formalism by $\mathrm{SLID}_{\mathsf{btw}}$.

A predicate pred satisfies *progress* iff every rule of pred contains exactly one points-to assertion and there exists a variable $x \in \mathsf{fvars}(\mathsf{pred})$ such that for all rules $(\mathsf{pred} \Leftarrow \phi) \in \Phi$, x is the variable on the left-hand side of this points-to assertion. In this case, we call x the *root of the predicate*. Moreover, if the i-th parameter of pred is the root of pred, then $\mathsf{predroot}(\mathsf{pred}(z_1,\dots,z_k)) := z_i$.

A predicate pred satisfies *connectivity* iff for all rules $(\mathsf{pred}(\mathbf{x}) \Leftarrow (x \mapsto \mathbf{y}) * \psi) \in \Phi$, the root parameters of the recursive calls in ψ occur in $\mathbf{y}$. A predicate pred is *established* iff all existentially quantified variables across all rules of pred are eventually allocated. Formally, for all rules $(\mathsf{pred} \Leftarrow \exists \mathbf{y}.\, \phi) \in \Phi$ and for all models $(\mathfrak{s},\mathfrak{h})$, if $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi$ then $\mathfrak{s}(\mathbf{y}) \subseteq \mathrm{dom}(\mathfrak{h})$.

Example 2.4 (SID Assumptions)**.** *The SIDs in Ex. 2.1 are all in $\mathrm{SLID}_{\mathsf{btw}}$.*

Throughout this article, we often restrict our attention to models of guarded formulas, which we call *guarded models* and collect in $\mathbf{M}^{\mathsf{g}}_\Phi := \{(\mathfrak{s},\mathfrak{h}) \mid \text{ex. } \phi \in \mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}} \text{ s.t. } (\mathfrak{s},\mathfrak{h}) \models_\Phi \phi\}$.

In guarded models, only locations in $\mathrm{img}(\mathfrak{s})$ can be dangling.

Lemma 2.5. *Let $\phi \in \mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ and $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi$. Then $\mathsf{dangling}(\mathfrak{h}) \subseteq \mathsf{fvars}(\phi)$.*

$$
\begin{array}{lll}
\mathsf{letter}_i(a) & \Leftarrow a \mapsto \underbrace{\langle \mathsf{nil}, \ldots, \mathsf{nil} \rangle}_{\text{length } i}, & 1 \leq i \leq n \\
N(x_1, x_2, x_3) & \Leftarrow \exists l, r, m.\, (x_1 \mapsto \langle l, r \rangle) \star A(l, x_2, m) \star B(r, m, x_3), & j \in \{1, 2\}, N \to AB \in \mathbf{R}_j \\
N(x_1, x_2, x_3) & \Leftarrow \exists a.\, (x_1 \mapsto \langle x_3, a \rangle) \star \mathsf{letter}_k(a) \star x_1 \approx x_2, & j \in \{1, 2\}, N \to a_k \in \mathbf{R}_j \\
\mathsf{word}(x, y) & \Leftarrow \exists a.\, (x \mapsto \langle y, a \rangle) \star \mathsf{letter}_i(a), & 1 \leq i \leq n \\
\mathsf{word}(x, y) & \Leftarrow \exists n, a.\, (x \mapsto \langle n, a \rangle) \star \mathsf{letter}_i(a) \star \mathsf{word}(n, y), & 1 \leq i \leq n
\end{array}
$$

Figure 4: The SID Φ that encodes the derivations of the context-free grammars $\mathcal{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle$ and $\mathcal{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$.

3 Undecidability of Extensions

In this section, we justify the use of guarded negation, magic wand, and septraction in $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ by proving that allowing any of these three operators to be used unguarded leads to an undecidable logic. Together with the decidability result for $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ that we present later in the paper, this yields an almost tight delineation between decidability and undecidability.

Context-free grammars. Our undecidability results are based on an encoding of the language-intersection problem for context-free grammars.

Definition 3.1. *A* context-free grammar *(CFG) is a 4-tuple $\mathcal{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$, where $\mathbf{N}$ is a finite set of* nonterminals*; $\mathbf{T}$ is a finite set of terminals, disjoint from $\mathbf{N}$; $\mathbf{R} \subseteq \mathbf{N} \times (\mathbf{N}^2 \cup \mathbf{T})$ is a finite set of* production rules*; and $\mathbf{S} \in \mathbf{N}$ is the* start symbol*.* **CFG** *is the set of all CFGs.*

We often denote production rules (a, b) by $a \to b$ to improve readability. We assume w.l.o.g. that CFGs are in Chomsky normal form. Further, we only consider CFGs that do not accept the empty word. Under these assumptions, rules are either of the form $N \to AB$, $A, B \in \mathbf{N}$, or of the form $N \to a$, $a \in \mathbf{T}$.

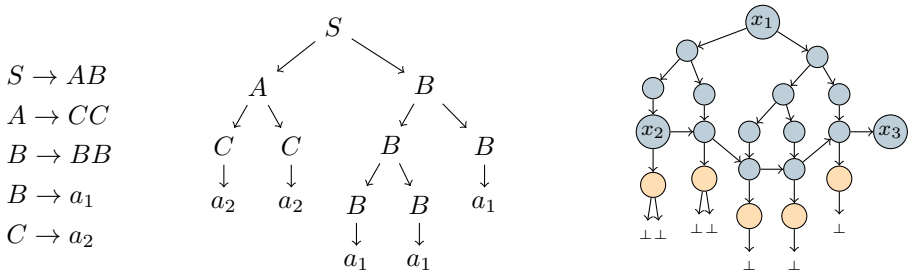
Definition 3.2. *Let $\mathcal{G} = (\mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S}) \in$ **CFG** and let $v, w \in \mathbf{N} \cup \mathbf{T}^*$. We write $v \Rightarrow w$ if there exist $u_1, u_2 \in \mathbf{N} \cup \mathbf{T}^*$, $(a, b) \in \mathbf{R}$ such that $v = u_1 \cdot a \cdot u_2$ and $w = u_1 \cdot b \cdot u_2$. We write $\Rightarrow^+$ for the transitive closure of $\Rightarrow$. The* language *of $\mathcal{G}$ is given by $\mathcal{L}(\mathcal{G}) := \{w \in \mathbf{T}^* \mid \mathbf{S} \Rightarrow^+ w\}$.*

In the following, we exploit the following classic result.

Theorem 3.3. *Let $\mathcal{G}_1, \mathcal{G}_2 \in$ **CFG**. It is undecidable whether $\mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2) \neq \emptyset$.*

Encoding CFGs as SIDs. Let $\mathbf{T} = \{a_1, \ldots, a_n\}$ and for $1 \leq i \leq 2$, let $\mathcal{G}_i = \langle \mathbf{N}_i, \mathbf{T}, \mathbf{R}_i, \mathbf{S}_i \rangle$ be a context-free grammar. Assume w.l.o.g. that $\mathbf{N}_1 \cap \mathbf{N}_2 = \emptyset$. Consider the SID Φ defined in Fig. 4. The predicates N, $N \in \mathbf{N}_1 \cup \mathbf{N}_2$, and letter_i, $1 \leq i \leq n$, encode the derivations of the grammars $\mathcal{G}_1, \mathcal{G}_2$ as trees with linked leaves (TLL), similar to the SID in Example 2.1. The predicate word is an auxiliary predicate that overapproximates the lists of linked leaves that the TLLs may contain; we will need it later. Every word in $\mathcal{L}(\mathcal{G}_i)$ corresponds to at least one model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}_i(x_1, x_2, x_3)$; and every model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}_i(x_1, x_2, x_3)$ corresponds to a *derivation tree* and a word in $\mathcal{L}(\mathcal{G}_i)$, where the inner nodes of the TLL correspond to the derivation tree and the linked list of leaves correspond to the word in $\mathcal{L}(\mathcal{G}_i)$.

Example 3.4. *We illustrate the encoding in Fig. 5. Figure 5a shows the rules $\mathbf{R}$ of a simple CFG $\mathcal{G} = \langle \{S, A, B, C\}, \{a_1, a_2\}, \mathbf{R}, S \rangle$. Figure 5b In Fig. 5c, we show the stack–heap model that encodes the aforementioned derivation tree. Every nonterminal is translated to a node in a*



(a) Production rules of a CFG $\mathcal{G}$.

(b) A derivation tree for the word $a_2a_2a_1a_1a_1 \in \mathcal{L}(\mathcal{G})$.

(c) The corresponding model of the predicate call $S(x_1, x_2, x_3)$.

Figure 5: Encoding a derivation of a context-free grammar as a stack–heap model.

binary tree (blue). The leaves of the tree are linked. They each have a successor that encodes a terminal symbol of the derivation (orange): The node contains k pointers to nil *(denoted as $\bot$ in the figure) to represent terminal a_k. The list of linked leaves and orange nodes together form the* induced word *of the model as defined later in Definition 3.7.*

To show the correctness of the encoding, we need the *induced words* of the models of Φ.

Definition 3.5. *Let $\mathcal{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ and let Φ be the corresponding SID encoding. Let $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{word}(x, y)$ and let $j_1, \ldots, j_m \in \{1, \ldots, n\}$ be such that*

$$(\mathfrak{s}, \mathfrak{h}) \models_\Phi \exists n_1, \ldots, n_{m-1}, b_1, \ldots, b_m. \, ((x \mapsto \langle n_1, b_1 \rangle) \star \mathsf{letter}_{j_1}(b_1)) \\ \star ((n_1 \mapsto \langle n_2, b_2 \rangle) \star \mathsf{letter}_{j_2}(b_2)) \star \cdots \star ((n_{m-1} \mapsto \langle y, b_m \rangle) \star \mathsf{letter}_{j_m}(b_m)).$$

We define the induced letters *of $(\mathfrak{s}, \mathfrak{h})$ and x, y as $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) := a_{j_1} a_{j_2} \cdots a_{j_m}$.*

Every model that satisfies $N(x_1, x_2, x_3)$ contains a sub-heap that satisfies the word predicate.

Lemma 3.6. *Let $\mathcal{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle$, $\mathcal{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$ and let Φ be the corresponding SID encoding. Let $x_1, x_2, x_3 \in \mathbf{Var}$, $N \in \mathbf{N}_1 \cup \mathbf{N}_2$ and let $(\mathfrak{s}, \mathfrak{h}) \models_\Phi N(x_1, x_2, x_3) \star \mathsf{t}$. Then there exists a unique heap $\mathfrak{h}_w \subseteq \mathfrak{h}$ with $(\mathfrak{s}, \mathfrak{h}_w) \models_\Phi \mathsf{word}(x_2, x_3)$.*

Lemma 3.6 ensures that the following is well defined.

Definition 3.7. *Let $\mathcal{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ and let Φ be the corresponding SID encoding. Let $x_1, x_2, x_3 \in \mathbf{Var}$, $N \in \mathbf{N}$ and let $(\mathfrak{s}, \mathfrak{h}) \models_\Phi N(x_1, x_2, x_3)$. Let $\mathfrak{h}_w \subseteq \mathfrak{h}$ be the unique heap with $(\mathfrak{s}, \mathfrak{h}_w) \models_\Phi \mathsf{word}(x_2, x_3)$. We define the* induced word *of $(\mathfrak{s}, \mathfrak{h})$ and N as $\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) := \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_w, x_2, x_3)$.*

Lemma 3.8. *Let $\mathcal{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ and let Φ be the corresponding SID encoding. Let $1 \leq i \leq 2$, $x_1, x_2, x_3 \in \mathbf{Var}$, and let $w \in \mathcal{L}(\mathcal{G})$. Then there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}(x_1, x_2, x_3)$ and $\mathsf{wordof}_{\mathbf{S}}(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = w$.*

Lemma 3.9. *Let $\mathcal{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ and let Φ be the corresponding SID encoding. Let $x_1, x_2, x_3 \in \mathbf{Var}$ and let $(\mathfrak{s}, \mathfrak{h})$ be a model with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}(x_1, x_2, x_3)$. Then $\mathsf{wordof}_{\mathbf{S}}(\mathfrak{s}, \mathfrak{h}, x_2, x_3) \in \mathcal{L}(\mathcal{G})$.*

We need an auxiliary result involving the formula

$$\mathsf{word}_2(x, y) := (\mathsf{word}(x, y) \circledast \mathbf{S}_2(a, x, y)) \circledast \mathbf{S}_2(a, x, y)$$

before we can prove the undecidability results.

Lemma 3.10. *Let $\mathcal{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$, let Φ be the corresponding SID encoding, and let $(\mathfrak{s}, \mathfrak{h})$ be a model. Then $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{word}_2(x, y)$ iff $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{word}(x, y)$ and $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) \in \mathcal{L}(\mathcal{G}_2)$.*

Proof. Let $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{word}_2(x, y)$. By the semantics of $\circledast$, there exists a heap $\mathfrak{h}_1$ with $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathsf{word}(x, y) \circledast \mathbf{S}_2(a, x, y)$ such that $(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1) \models_\Phi \mathbf{S}_2(a, x, y)$. Observe that $\mathfrak{h}_1$ contains precisely the inner nodes of the model $(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1)$, i.e., everything *except* the part of the model that induces the word. Consequently, $\mathfrak{h}$ is the part of the model that induces the word, i.e., $\mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y)$ and $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{word}(x, y)$. Lemma 3.9 then yields $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) \in \mathcal{L}(\mathcal{G}_2)$.

Conversely, let $(\mathfrak{s}, \mathfrak{h})$ be such that $w := \mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) \in \mathcal{L}(\mathcal{G}_2)$. As a consequence of Lemma 3.8, there exists a heap $\mathfrak{h}_1$ with $(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1) \models_\Phi \mathbf{S}_2(a, x, y)$. By the semantics of $\circledast$ and because $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{word}(x, y)$ by assumption, we then have $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathsf{word}(x, y) \circledast \mathbf{S}_2(a, x, y)$. Because $(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1) \models_\Phi \mathbf{S}_2(a, x, y)$, we obtain by the semantics of $\circledast$ that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi (\mathsf{word}(x, y) \circledast \mathbf{S}_2(a, x, y)) \circledast \mathbf{S}_2(a, x, y)$. ☐

We are ready to prove the undecidability results.

Theorem 3.11. *The satisfiability problems of the separation logics (1) $\mathbf{SL}(\wedge, \star, \mathtt{t})$, (2) $\mathbf{SL}(\wedge, \star, \neg)$, (3) $\mathbf{SL}(\wedge, \star, \mathrel{-\!\!\star})$, and (4) $\mathbf{SL}(\circledast)$ are undecidable.*

Proof. Throughout this proof, let $\mathcal{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle, \mathcal{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle \in \mathbf{CFG}$ and let Φ be the corresponding SID encoding.

Undecidability of $\mathbf{SL}(\wedge, \star, \mathtt{t})$: We claim that $\phi := (\mathbf{S}_1(a, x, y) \star \mathtt{t}) \wedge (\mathbf{S}_2(b, x, y) \star \mathtt{t})$ is satisfiable iff $\mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2) \neq \emptyset$. We prove the implications separately.

Assume ϕ is satisfiable. Then there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$. Let $\mathfrak{h}_{w_1}, \mathfrak{h}_{w_2} \subseteq \mathfrak{h}$ be such that $\mathsf{wordof}_{\mathbf{S}_i}(\mathfrak{s}, \mathfrak{h}, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_i}, x, y)$; such heaps exist by Lemma 3.6.

Observe that both $(\mathfrak{s}, \mathfrak{h}_{w_1}) \models_\Phi \mathsf{word}(x, y)$ and $(\mathfrak{s}, \mathfrak{h}_{w_2}) \models_\Phi \mathsf{word}(x, y)$. Consequently, $\mathfrak{h}_{w_1} = \mathfrak{h}_{w_2}$, which implies $\mathsf{wordof}_{\mathbf{S}_1}(\mathfrak{s}, \mathfrak{h}, x, y) = \mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s}, \mathfrak{h}, x, y) =: w$. By Lemma 3.9, it follows that $w \in \mathcal{L}(\mathcal{G}_1)$ and $w \in \mathcal{L}(\mathcal{G}_2)$, i.e., $w \in \mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2)$.

Conversely, assume $\mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2) \neq \emptyset$ and let $w \in \mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2)$. By Lemma 3.8, there exist models $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2)$ with $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathbf{S}_1(a, x, y)$ and $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \mathbf{S}_2(b, x, y)$. Let $\mathfrak{h}_{w_1} \subseteq \mathfrak{h}_1, \mathfrak{h}_{w_2} \subseteq \mathfrak{h}_2$ be the unique heaps with $\mathsf{wordof}_{\mathbf{S}_1}(\mathfrak{s}, \mathfrak{h}_1, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_1}, x, y) = w = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_2}, x, y) = \mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s}, \mathfrak{h}_2, x, y)$.

Observe that $(\mathfrak{s}, \mathfrak{h}_{w_1}) \cong (\mathfrak{s}, \mathfrak{h}_{w_2})$. Consequently, we can replace $\mathfrak{h}_2$ with an isomorphic heap that contains $\mathfrak{h}_{w_1}$ (as opposed to $\mathfrak{h}_{w_2}$) as sub-heap and is otherwise disjoint from $\mathfrak{h}_1$, i.e., there exists a heap $\mathfrak{h}_2'$ such that $\mathfrak{h}_2 \cong \mathfrak{h}_2'$, $\mathsf{locs}(\mathfrak{h}_2') \cap \mathsf{locs}(\mathfrak{h}_1) = \mathsf{locs}(\mathfrak{h}_{w_1})$, and $\mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s}, \mathfrak{h}_2', x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_1}, x, y) = w$. Note in particular that $(\mathfrak{s}, \mathfrak{h}_2') \models_\Phi \mathbf{S}_2(b, x, y)$, because isomorphic models satisfy the same formulas. Now let $\mathfrak{h} := \mathfrak{h}_1 \cup \mathfrak{h}_2'$ be the (non-disjoint) union of $\mathfrak{h}_1$ and $\mathfrak{h}_2'$. Since $\mathfrak{h}_1 \subseteq \mathfrak{h}$ and $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathbf{S}_1(a, x, y)$, we have $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}_1(a, x, y) \star \mathtt{t}$; and similarly, since $\mathfrak{h}_2' \subseteq \mathfrak{h}$ and $(\mathfrak{s}, \mathfrak{h}_2') \models_\Phi \mathbf{S}_2(a, x, y)$, we have that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}_2(b, x, y)$. Consequently, $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$.

Undecidability of $\mathbf{SL}(\wedge, \star, \neg)$: Follows directly from the undecidability of $\mathbf{SL}(\wedge, \star, \mathtt{t})$, because $\mathtt{t}$ is definable in $\mathbf{SL}(\wedge, \star, \neg)$; for example $\mathtt{t} := \neg(\mathbf{emp} \wedge \neg \mathbf{emp})$.

Undecidability of $\mathbf{SL}(\wedge, \star, \mathrel{-\!\!\star})$: Follows directly from the undecidability of $\mathbf{SL}(\wedge, \star, \mathtt{t})$, because $\mathtt{t}$ is definable in $\mathbf{SL}(\wedge, \star, \mathrel{-\!\!\star})$; for example $\mathtt{t} := (x \not\approx x) \mathrel{-\!\!\star} \mathbf{emp}$.

Undecidability of $\mathbf{SL}(\circledast)$: We claim that that $\psi := \mathsf{word}_2(x, y) \circledast \mathbf{S}_1(a, x, y)$, for word_2 as defined earlier in this section, is satisfiable iff $\mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2) \neq \emptyset$. Intuitively, this holds because ψ is satisfiable iff it is possible to replace the "word part" of a model of $\mathbf{S}_1(a, x, y)$ with the "word part" of a model of $\mathbf{S}_2(b, x, y)$.

We now formalize this intuition. Assume ψ is satisfiable and let $(\mathfrak{s}, \mathfrak{h})$ be such that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \psi$. By the semantics of $\circledast$, we have that there exists a heap $\mathfrak{h}_0 \subseteq \mathfrak{h}$ with $\mathfrak{h}_0 \models_\Phi \mathsf{word}_2(x, y)$

and $(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_0) \models_\Phi \mathbf{S}_1(a, x, y)$. As $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y) \in \mathcal{L}(\mathcal{G}_2)$ by Lemma 3.10, we have in particular that $(\mathfrak{s}, \mathfrak{h}_0) \models_\Phi \mathsf{word}(x, y)$. It follows that $\mathfrak{h}_0$ is the unique sub-heap of $\mathfrak{h} + \mathfrak{h}_0$ with $\mathsf{wordof}_{\mathbf{S}_1}(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_0) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y)$. By Lemma 3.9, $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y) \in \mathcal{L}(\mathcal{G}_1)$. Together with Lemma 3.10, we thus have that $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y) \in \mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2)$.

Conversely, assume there exists a word $w \in \mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2)$. By Lemma 3.8, there exist heaps $\mathfrak{h}, \mathfrak{h}_0, \mathfrak{h}', \mathfrak{h}'_0$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}_1(a, x, y)$, $\mathsf{wordof}_{\mathbf{S}_1}(\mathfrak{s}, \mathfrak{h}, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y)$, $(\mathfrak{s}, \mathfrak{h}') \models_\Phi \mathbf{S}_2(a, x, y)$, and $\mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s}, \mathfrak{h}', x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}'_0, x, y)$.

Because $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}'_0, x, y)$, it holds that $\mathfrak{h}_0 \cong \mathfrak{h}'_0$, so we can assume w.l.o.g. that $\mathfrak{h}_0 = \mathfrak{h}'_0$—if the models are not isomorphic, simply replace $\mathfrak{h}'$ with an appropriate isomorphic heap to establish this property. Let $\mathfrak{h}_2 \subseteq \mathfrak{h}'$ be the sub-heap of $\mathfrak{h}'$ with $\mathfrak{h}_2 + \mathfrak{h}_0 = \mathfrak{h}'$. By Lemma 3.10, $(\mathfrak{s}, \mathfrak{h}_0) \models_\Phi \mathsf{word}_2(x, y)$. Consequently, $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \psi$, i.e., ψ is satisfiable. □

The fundamental difference between the logics in Theorem 3.11 and $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ lies in the possibility to decompose the heap into parts with unboundedly many dangling pointers: The number of dangling pointers in the models of $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ formulas is always bounded by the number of free variables of the formula (cf. Lemma 2.5).

4 The Types Abstraction

Having established that all natural extensions of $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ are undecidable, we now turn to the decidability of $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$. In this section, we present the Φ-*type abstraction* that is at the heart of our decision procedure. The Φ-types abstraction and the associated results essentially follow our paper [23]; we only summarize here what is needed for proving the decidability results in the next section. We note, however, that since the publication of [23], we have tweaked the definition of the abstraction to fix an incompleteness issue and improve the presentation; we refer the reader to our technical report [26] for a full self-contained exposition.

The idea behind the Φ-type abstraction is as follows. Given a SID Φ and a model $(\mathfrak{s}, \mathfrak{h})$, we compute a set of formulas that encodes all the ways that one or more predicates of Φ can be *partially unfolded* such that $(\mathfrak{s}, \mathfrak{h})$ is a model of the partially-unfolded predicates.

Example 4.1. *Consider the following SID* Φ*.*

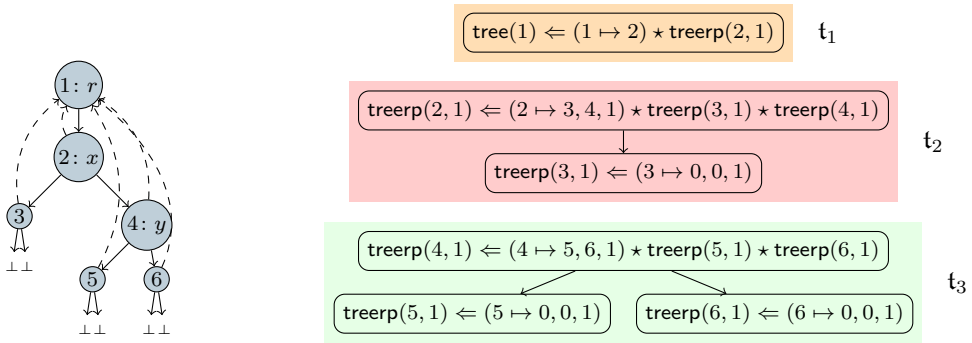
$$\begin{array}{lcl} \mathsf{tree}(r) & \Leftarrow & \exists x.\, (r \mapsto x) \star \mathsf{treerp}(x, r) \\ \mathsf{treerp}(x, r) & \Leftarrow & (x \mapsto \langle \mathsf{nil}, \mathsf{nil}, r \rangle) \\ \mathsf{treerp}(x, r) & \Leftarrow & \exists c_1, c_2.\, (x \mapsto \langle c_1, c_2, r \rangle) \star \mathsf{treerp}(c_1, r) \star \mathsf{treerp}(c_2, r) \end{array}$$

Let $\mathfrak{s} = \{r \mapsto 1, x \mapsto 2, y \mapsto 4\}$, $\mathfrak{h} = \{1 \mapsto 2, 2 \mapsto \langle 3, 4, 1 \rangle, 3 \mapsto \langle 0, 0, 1 \rangle, 4 \mapsto \langle 5, 6, 1 \rangle, 5 \mapsto \langle 0, 0, 1 \rangle, 6 \mapsto \langle 0, 0, 1 \rangle\}$. *We display* $(\mathfrak{s}, \mathfrak{h})$ *in Figure 6a. Each node is labeled with a location and the stack variable interpreted by the location (if any). The first two outgoing pointers of each node are displayed by solid edges, the third pointer by a dashed edge. Fig. 6b shows a* Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\}$ *that encodes one way to derive the model* $(\mathfrak{s}, \mathfrak{h})$ *by unfolding predicates of the SID. Both* $\mathfrak{t}_1$ *and* $\mathfrak{t}_2$ *only* partially *unfold the predicates at their roots. The* stack–forest projection *of* $\mathfrak{s}$ *and* $\mathfrak{f}$ *is the formula* $(\mathsf{treerp}(x, r) \mathrel{-\!\!\star} \mathsf{tree}(r)) \star (\mathsf{treerp}(y, r) \mathrel{-\!\!\star} \mathsf{treerp}(x, r)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, r))$.

In this section, we formalize the notion of Φ-forests; and we show how to systematically obtain the projections of all relevant forests of a model.

4.1 Φ-Forests

We begin by formalizing partial unfoldings of Φ-predicates. To this end, we introduce Φ-*forests* (Definition 4.3) made up of Φ-*trees* (Definition 4.2). Intuitively, a Φ-tree encodes one fixed way



(a) A model of $\mathsf{tree}(r)$. (b) A corresponding Φ-forest $\mathfrak{f}$ consisting of trees $\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3$.

Figure 6: A model of $\mathsf{tree}(r)$ and one of the Φ-forests corresponding to this model.

to unfold a predicate call by means of the rules of the SID Φ. Our notion of Φ-trees is related to the unfolding trees used, e.g., in [16, 17, 20], but there are two key differences. First, we instantiate variables with locations; second, we explicitly allow the unfolding process to stop at any point, i.e., we allow that one or more of the predicate calls introduced in the unfolding process remain folded. We call such folded predicate calls the *holes* of the Φ-tree.

The nodes of a Φ-tree are labeled with *rule instances*, which are obtained from the rules of the SID by instantiating both the formal arguments of the predicates and the existentially quantified variables of the rule with locations:

$$\begin{aligned}\mathbf{RuleInst}(\Phi) := \{\mathsf{pred}(\mathbf{l}) \Leftarrow \phi[\mathbf{x}\cdot\mathbf{y}/\mathbf{l}\cdot\mathbf{m}] \mid{}& (\mathsf{pred}(\mathbf{x}) \Leftarrow \exists\mathbf{y}.\,\phi) \in \Phi,\\ & \mathbf{l} \in \mathbf{Loc}^{\mathsf{ar}(\mathsf{pred})}, \mathbf{m} \in \mathbf{Loc}^{|\mathbf{y}|} \text{ and all}\\ & \text{(dis)equalities in } \phi[\mathbf{x}\cdot\mathbf{y}/\mathbf{l}\cdot\mathbf{m}] \text{ are valid}\}\end{aligned}$$

Rule instances are *not* SL formulas themselves, as the terms in rule instances are locations rather than variables. We will come back to this point later.

We model Φ-trees as functions $\mathfrak{t}\colon \mathbf{Loc} \rightharpoonup (\mathbf{Loc}^* \times \mathbf{RuleInst}(\Phi))$. The set of locations $\mathbf{Loc}$ serves as the nodes of the tree; and every node is mapped to its successors in the (directed) tree as well as to a rule instance that serves as a node label. For $\mathfrak{t}$ to be a Φ-tree, it must satisfy a certain set of consistency criteria. To make it easier to work with Φ-trees and formalize the consistency criteria, we first introduce some additional notation.

Let Φ be an SID and let $\mathfrak{t}(l) = \langle \mathbf{m}, (\mathsf{pred}(\mathbf{z}) \Leftarrow (a \mapsto \mathbf{b}) \star \phi\rangle$. Note that, by progress of the SID Φ, ϕ does not contain points-to assertions. We define:

$$\begin{aligned}&\mathsf{succ}_\mathfrak{t}(l) := \mathbf{m} \qquad \mathsf{ptr}_\mathfrak{t}(l) := a \mapsto \mathbf{b} \qquad \mathsf{head}_\mathfrak{t}(l) := \mathsf{pred}(\mathbf{z}) \qquad \mathsf{calls}_\mathfrak{t}(l) := \phi\\ &\mathsf{holepreds}_\mathfrak{t}(l) := \left\{\mathsf{pred}'(\mathbf{z}') \in \mathsf{calls}_\mathfrak{t}(l) \mid \forall c \in \mathsf{succ}_\mathfrak{t}(l).\, \mathsf{head}_\mathfrak{t}(c) \neq \mathsf{pred}'(\mathbf{z}')\right\}\end{aligned}$$

Informally, the *hole predicates* of l are those predicate calls in $\mathsf{calls}_\mathfrak{t}(l)$ whose root does not occur in $\mathsf{succ}_\mathfrak{t}(m)$. We collect all hole predicates of $\mathfrak{t}$ in $\mathsf{allholepreds}(\mathfrak{t}) := \bigcup_{c\in\mathrm{dom}(\mathfrak{t})} \mathsf{holepreds}_\mathfrak{t}(c)$. Finally, we define the projection of $\mathfrak{t}$ onto the directed graph, $\mathsf{graph}(\mathfrak{t}) \subseteq \mathbf{Loc} \times \mathbf{Loc}$, induced by its first component, $\mathsf{graph}(\mathfrak{t}) := \{(x, y) \mid x \in \mathrm{dom}(\mathfrak{t}), y \in \mathsf{succ}_\mathfrak{t}(x)\}$.

Definition 4.2 (Φ-Tree)**.** *Let Φ be an SID. A partial function $\mathfrak{t}\colon \mathbf{Loc} \rightharpoonup (\mathbf{Loc}^* \times \mathbf{RuleInst}(\Phi))$ is a* Φ-tree *iff*

1. $\mathsf{graph}(\mathfrak{t})$ *is a (directed) tree and*
2. $\mathfrak{t}$ *is* Φ-consistent*, i.e., for all* $l \in \mathrm{dom}(\mathfrak{t})$*, if* $\mathsf{succ}_{\mathfrak{t}}(l) = \langle y_1, \dots, y_k \rangle$*,* $\mathsf{head}_{\mathfrak{t}}(l) = \mathsf{pred}(\mathbf{z})$*,* $\mathsf{ptr}_{\mathfrak{t}}(l) = a \mapsto \mathbf{b}$*, and* $\mathsf{calls}_{\mathfrak{t}}(l) = \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \mathsf{pred}_m(\mathbf{z_m}) * \Pi$*,* Π *pure, then (1)* $l = a$*, (2)* $\mathsf{succ}_{\mathfrak{t}}(l) \sqsubseteq \mathbf{b}$*, and (3)* $\{\mathsf{head}_{\mathfrak{t}}(y_1), \dots, \mathsf{head}_{\mathfrak{t}}(y_k)\} \subseteq \{\mathsf{pred}_1(\mathbf{z_1}), \dots, \mathsf{pred}_m(\mathbf{z_m})\}$.

Let $\mathfrak{t}$ *be a* Φ*-tree. As* $\mathfrak{t}$ *is a directed tree, it has a root, which we denote by* $\mathsf{root}(\mathfrak{t})$*. We set* $\mathsf{rootpred}(\mathfrak{t}) := \mathsf{head}_{\mathfrak{t}}(\mathsf{root}(\mathfrak{t}))$.

We combine zero or more Φ-trees into Φ-*forests*.

Definition 4.3 (Φ-Forest)**.** *Let* Φ *be an SID. Let* $\mathfrak{t}_1, \dots, \mathfrak{t}_k$ *be* Φ*-trees. The set* $\mathfrak{f} = \{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}$ *is a* Φ-forest *iff* $\mathrm{dom}(\mathfrak{t}_i) \cap \mathrm{dom}(\mathfrak{t}_j) = \emptyset$ *for* $i \neq j$.

Example 4.4 (Φ-forest)**.** *Figure 6b shows the* Φ*-forest* $\mathfrak{f} = \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\}$ *with* $\mathfrak{t}_1 = \{1 \mapsto \langle \varepsilon, \mathsf{tree}(1) \Leftarrow (1 \mapsto 2) \star \mathsf{treerp}(2,1) \rangle\}$*,* $\mathfrak{t}_2 = \{2 \mapsto \langle 3, \mathsf{treerp}(2,1) \Leftarrow (2 \mapsto 3,4,1) \star \mathsf{treerp}(3,1) \star \mathsf{treerp}(4,1) \rangle, 3 \mapsto \langle \varepsilon, \mathsf{treerp}(3,1) \Leftarrow (3 \mapsto 0,0,1) \rangle\}$*, and* $\mathfrak{t}_3 = \{4 \mapsto \langle \langle 5,6 \rangle, \mathsf{treerp}(4,1) \Leftarrow (4 \mapsto 5,6,1) \star \mathsf{treerp}(5,1) \star \mathsf{treerp}(6,1) \rangle, 5 \mapsto \langle \varepsilon, \mathsf{treerp}(5,1) \Leftarrow (5 \mapsto 0,0,1) \rangle, 6 \mapsto \langle \varepsilon, \mathsf{treerp}(6,1) \Leftarrow (6 \mapsto 0,0,1) \rangle\}$

Definition 4.5 (Induced Heap)**.** *Let* $\mathfrak{f}$ *be a* Φ*-forest. The* induced heap of $\mathfrak{f}$ *is given by* $\mathsf{heap}(\mathfrak{f}) := \bigcup_{\mathfrak{t} \in \mathfrak{f}} \bigcup_{c \in \mathrm{dom}(\mathfrak{t})} \mathsf{ptr}_{\mathfrak{t}}(c)$.

Example 4.6. *For* $\mathfrak{h}$ *and* $\mathfrak{f}$ *as in Example 4.1,* $\mathfrak{h} = \mathsf{heap}(\mathfrak{f})$.

Our notion of Φ-trees generalizes "unfolding trees" [16] in the sense that every model of a predicate call corresponds to (at least one) Φ-tree without holes.

Lemma 4.7. *Let* Φ *be an SID,* $\mathsf{pred}(\mathbf{z})$ *a predicate call, and* $(\mathfrak{s}, \mathfrak{h})$ *a model. If* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{pred}(\mathbf{z})$ *there exists a* Φ*-tree* $\mathfrak{t}$ *with* $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(\mathbf{z}))$*,* $\mathsf{allholepreds}(\mathfrak{t}) = \emptyset$*, and* $\mathsf{heap}(\{\mathfrak{t}\}) = \mathfrak{h}$.

4.2 Projecting Φ-Forests onto Formulas

The main insight behind the *projection* of Φ-forests onto SL formulas is that every Φ-tree $\mathfrak{t}$ can be viewed as encoding a model of $\mathsf{rootpred}(\mathfrak{t})$ from which models of $\mathsf{allholepreds}(\mathfrak{t})$ have been subtracted. This can be naturally encoded by a magic wand, $(\bigstar \mathsf{allholepreds}(\mathfrak{t})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t})$.

Recall, however, that the above is *not* a formula, because it contains locations rather than variables. In our projection function, we thus need to replace the locations with variables. To this end, we need *guarded* versions of both existential and universal quantifiers, which we denote $\mathbb{I}$ and $\mathbb{V}$. They have the following semantics.

- $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbb{I} \langle e_1, \dots, e_k \rangle . \phi$ iff there exist pairwise different $v_1, \dots, v_k \in \mathrm{dom}(\mathfrak{h}) \setminus \mathrm{img}(\mathfrak{s})$ such that $(\mathfrak{s} \cup \{e_1 \mapsto v_1, \dots, e_k \mapsto v_k\}, \mathfrak{h}) \models_\Phi \phi$.
- $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbb{V} \langle a_1, \dots, a_k \rangle . \phi$ iff for all pairwise different $v_1, \dots, v_k \in \mathbf{Loc} \setminus (\mathsf{locs}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s}))$, it holds that $(\mathfrak{s} \cup \{a_1 \mapsto v_1, \dots, a_k \mapsto v_k\}, \mathfrak{h}) \models_\Phi \phi$.

Note that these quantifiers are not quite dual, i.e., $\mathbb{I}\mathbf{x}. \phi$ is *not* equivalent to $\neg \mathbb{V}\mathbf{x}. \neg\phi$. We take the following approach to translate a forest $\mathfrak{f}$ to a formula with guarded quantifiers:

1. Every location $v \in \mathrm{img}(\mathfrak{s})$ is replaced by an arbitrary variable in $\mathfrak{s}^{-1}(v)$.
2. Every location $v \in \mathsf{locs}(\mathsf{heap}(\mathfrak{f}))$ with $v \notin \mathrm{img}(\mathfrak{s})$ is replaced by a guarded existential. Note that by Lemma 2.5, $v \in \mathrm{dom}(\mathsf{heap}(\mathfrak{f}))$, as required by the above semantics of $\mathbb{I}$.

3. All other locations are replaced by a guarded universal.

Let us formalize this construction. In the following, we assume for all stacks that $\mathrm{dom}(\mathfrak{s}) \cap (\{a_1, a_2, \ldots\} \cup \{e_1, e_2, \ldots\}) = \emptyset$.

Definition 4.8 (Tree projection)**.** *Let* $\mathfrak{t}$ *be a* Φ*-tree and let* $\mathbf{v} \subseteq \mathbf{Loc}$*. Let* $\mathsf{locs}(\phi)$ *be the set of all locations that occur in* ϕ*. The* tree projection *of* $\mathfrak{t}$ *w.r.t.* $\mathbf{v}$*,* $\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t})$*, is then given by*

$$\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}) := \mathbb{\forall}\mathbf{a}.\, \psi[\mathbf{w}/\mathbf{a}] \text{ where } \psi := (\bigstar \mathsf{allholepreds}(\mathfrak{t})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t})$$
$$\mathbf{w} := \mathsf{locs}(\psi) \setminus (\mathbf{v} \cup \mathsf{locs}(\mathsf{heap}(\{\mathfrak{t}\})))$$
$$\mathbf{a} := \langle a_1, \ldots, a_{|\mathbf{w}|} \rangle .$$

Definition 4.9. *Let* $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$ *be a* Φ*-forest with* $\mathsf{heap}(\mathfrak{f}) \in \mathbf{M}^{\mathbf{g}}_{\Phi}$*. Let* $\mathbf{v} := \mathrm{img}(\mathfrak{s})$ *and let* $\mathbf{w} := \mathsf{locs}(\mathsf{heap}(\mathfrak{f})) \setminus \mathrm{img}(\mathfrak{s})$ *be the (arbitrarily ordered) sequence of locations that occur in a pointer in* $\mathsf{heap}(\mathfrak{f})$ *but are not the value of any stack variable. Let* $\mathbf{x} \subseteq \mathrm{dom}(\mathfrak{s})$ *be such that* $\mathfrak{s}(\mathbf{x}) = \mathbf{v}$ *and let* $\mathbf{e} := \langle e_1, e_2, \ldots, e_{|\mathbf{w}|} \rangle$*. Finally, let* $\psi := \bigstar_{1 \le i \le k} \mathsf{project}^{\mathbf{Loc}}(\mathbf{v} \cup \mathbf{w}, \mathfrak{t}_i)$*. The* stack–forest projection *of* $\mathfrak{s}$ *and* $\mathfrak{f}$*,* $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$*, is given by* $\mathbb{\exists}\mathbf{e}.\, \psi[\mathbf{v} \cdot \mathbf{w}/\mathbf{x} \cdot \mathbf{e}]$.

Example 4.10. *Let* Φ*,* $\mathfrak{s}$*,* $\mathfrak{t}_1$*,* $\mathfrak{t}_2$*,* $\mathfrak{t}_3$ *and* $\mathfrak{f}$ *be as in Example 4.1. (1) The formula* $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$ *is given in Example 4.1. (2) Let* $\mathfrak{s}' = \{x \mapsto 2, y \mapsto 4\}$*. Then* $\mathsf{project}(\mathfrak{s}', \{\mathfrak{t}_2, \mathfrak{t}_3\}) = \mathbb{\exists}e_1.\, (\mathsf{treerp}(y, e_1) \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, e_1))$*. (3) For* $\mathfrak{s}'' = \{x \mapsto 1, y \mapsto 2\}$ *and* $\mathfrak{t}'' = \{1 \mapsto \langle \varepsilon, \mathsf{even}(1, 3) \Leftarrow (1 \mapsto 2) \star \mathsf{odd}(2, 3) \rangle\}$*,* $\mathsf{project}(\mathfrak{s}'', \{\mathfrak{t}''\}) = \mathbb{\forall}a_1.\, \mathsf{odd}(y, a_1) \mathrel{-\!\!\star} \mathsf{even}(x, a_1)$.

Stack–forest projection is sound in the following sense.

Lemma 4.11. *Let* $\mathfrak{f}$ *be a* Φ*-forest with* $\mathsf{heap}(\mathfrak{f}) \in \mathbf{M}^{\mathbf{g}}_{\Phi}$ *and let* $\mathfrak{s}$ *be a stack. Then* $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_{\Phi} \mathsf{project}(\mathfrak{s}, \mathfrak{f})$.

The correctness of Lemma 4.11 hinges on the use of guarded quantifiers. For example, let $\Phi = \{\mathsf{pred}(x_1, x_2) \Leftarrow (x_1 \mapsto \mathsf{nil}) \star (x_1 \not\approx x_2)\}$ and $\mathfrak{t} = \{v_1 \mapsto \langle \varepsilon, \mathsf{pred}(v_1, v_2) \Leftarrow (v_1 \mapsto 0) \star (v_1 \not\approx v_2) \rangle\}$, $\mathfrak{f} = \{\mathfrak{t}\}$, and $\mathfrak{s} = \{y \mapsto v_1\}$. Then $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_{\Phi} \mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \mathbb{\forall}a_1.\, \mathbf{emp} \mathrel{-\!\!\star} \mathsf{pred}(y, a_1)$, but $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \not\models_{\Phi} \mathbf{emp} \mathrel{-\!\!\star} \mathsf{pred}(y, y)$, so $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \not\models_{\Phi} \mathbb{\forall}a_1.\, \mathbf{emp} \mathrel{-\!\!\star} \mathsf{pred}(y, a_1)$.

All stack–forest projections can be obtained by "partially unfolding" symbolic heaps (and adding appropriate quantifiers), so we call them *unfolded symbolic heaps* (USHs) w.r.t. Φ.

Delimited USHs. Clearly, there are infinitely many USHs w.r.t. any (nonempty) SID Φ. This makes USHs unsuitable for defining a finite abstraction. To obtain a finite abstraction, we only consider USHs where (1) all root parameters of predicate calls are free variables and (2) every variable occurs at most once as a root parameter on the left-hand side of a magic wand.

Definition 4.12. *An unfolded symbolic heap* ϕ *is* delimited *iff*

1. *for all* $\mathsf{pred}(\mathbf{z}) \in \phi$*,* $\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \in \mathsf{fvars}(\phi)$*, and*
2. *for all* z *there exists at most one predicate call* $\mathsf{pred}(\mathbf{z}) \in \phi$ *such that* $z = \mathsf{predroot}(\mathsf{pred}(\mathbf{z}))$ *and* $\mathsf{pred}(\mathbf{z})$ *occurs on the left-hand side of a magic wand.*

Example 4.13. *All projections in Example 4.10 are DUSHs. Let* $\mathfrak{t}''$ *be as in Example 4.10. Then* $\mathsf{project}(\{x \mapsto 1\}, \{\mathfrak{t}''\}) = \mathbb{\exists}e_1.\, \mathbb{\forall}a_1.\, \mathsf{odd}(e_1, a_1) \mathrel{-\!\!\star} \mathsf{even}(x, a_1)$ *is not a DUSH, because* $e_1 = \mathsf{predroot}(\mathsf{odd}(e_1, a_1))$*, but* $e_1 \notin \mathrm{dom}(\mathfrak{s})$.

We let $\mathbf{DUSH}_{\Phi} := \{\phi \mid \phi \text{ is USH w.r.t. } \Phi \text{ and delimited}\}$ be the set of delimited USHs (DUSHs) over SID Φ and $\mathbf{DUSH}^{\mathbf{x}}_{\Phi}$ the restriction of $\mathbf{DUSH}_{\Phi}$ to formulas ϕ with $\mathsf{fvars}(\phi) \subseteq \mathbf{x}$.

Lemma 4.14. *Let Φ be an SID and let $\mathbf{x} \in 2^{\mathbf{Var}}$ be a finite set of variables. Let $n := |\Phi| + |\mathbf{x}|$. Then $|\mathbf{DUSH}^{\mathbf{x}}_{\Phi}| \in 2^{\mathcal{O}(n^2 \log(n))}$.*

We will abstract a model by the set of all DUSHs that hold in the model. This makes sense because we can effectively *compose* such sets, as we will see next.

4.3 The Φ-Type Abstraction

We abstract a model $(\mathfrak{s}, \mathfrak{h})$ by its Φ-*type*, which consists of the set of DUSHs that hold in $(\mathfrak{s}, \mathfrak{h})$.

Definition 4.15 (Φ-Type)**.** *Let $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{M}^{\mathbf{g}}_{\Phi}$ be a guarded model and Φ an SID. The* Φ-type *of $(\mathfrak{s}, \mathfrak{h})$ is given by $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) := \{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \mathfrak{f} \text{ is a forest with } \mathsf{heap}(\mathfrak{f}) = \mathfrak{h}\} \cap \mathbf{DUSH}_{\Phi}$.*

In our decision procedure, we need to *compose* types, i.e., we need an operation $\bullet$ such that

$$\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2) = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_2). \tag{$\dagger$}$$

We now develop such an operation. We begin by defining an operation $\bullet_{\mathbf{P}}$ that satisfies the identity ($\dagger$), *provided* that we know that $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$. To define $\bullet_{\mathbf{P}}$, we need two ingredients: (1) a variant of $\star$ that captures all sound ways to move the existential quantifiers to the front of the formula $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \star \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ and (2) a derivation operation on projections, $\rhd$, that rewrites formulas based on the fact that

$$((\mathsf{pred}_2(\mathbf{x_2}) \star \psi) \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1})) \star (\psi' \mathrel{-\!\!\star} \mathsf{pred}_2(\mathbf{x_2})) \text{ implies } (\psi \star \psi') \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1}). \tag{$\ddagger$}$$

We need the following auxiliary notation. We write $\forall \mathbf{a}.\, \phi \xrightarrow{\exists \mathbf{e}/\forall \mathbf{u}} \forall \mathbf{b}.\, \phi'$ if there exist disjoint subsets $\bar{\mathbf{a}_1}, \bar{\mathbf{a}_2} \subseteq \mathbf{a}$ and subsets $\bar{\mathbf{u}} \subseteq \mathbf{u}$, $\bar{\mathbf{e}} \subseteq \mathbf{e}$ such that $\phi' = \phi[\bar{\mathbf{a}_1} \cdot \bar{\mathbf{a}_2} / \bar{\mathbf{u}} \cdot \bar{\mathbf{e}}]$ and $\mathbf{b} = \mathbf{a} \setminus (\bar{\mathbf{a}_1} \cup \bar{\mathbf{a}_2}) \cup \bar{\mathbf{u}}$.

Example 4.16. *Let $\phi = \forall \langle a_1, a_2 \rangle .\, (\mathsf{odd}(y, a_2) \mathrel{-\!\!\star} \mathsf{odd}(e_1, a_2)) \star (\mathsf{even}(e_1, a_1) \mathrel{-\!\!\star} \mathsf{odd}(x, a_1))$. Then*

1. *$\phi \xrightarrow{\exists e_2/\forall u_1} \forall u_1.\, (\mathsf{odd}(y, e_2) \mathrel{-\!\!\star} \mathsf{odd}(e_1, e_2)) \star (\mathsf{even}(e_1, u_1) \mathrel{-\!\!\star} \mathsf{odd}(x, u_1))$, but also*
2. *$\phi \xrightarrow{\exists e_2/\forall u_1} \forall u_1.\, (\mathsf{odd}(y, u_1) \mathrel{-\!\!\star} \mathsf{odd}(e_1, u_1)) \star (\mathsf{even}(e_1, e_2) \mathrel{-\!\!\star} \mathsf{odd}(x, e_2))$, and*
3. *$\phi \xrightarrow{\exists e_2/\forall u_1} \forall a_1.\, (\mathsf{odd}(y, e_2) \mathrel{-\!\!\star} \mathsf{odd}(e_1, e_2)) \star (\mathsf{even}(e_1, a_1) \mathrel{-\!\!\star} \mathsf{odd}(x, a_1))$, etc.*

The notions of *re-scoping* and *derivability* provide the two ingredients we need to define $\bullet_{\mathbf{P}}$.

Definition 4.17 (Re-scoping)**.** *Let $\phi_i = \exists \mathbf{e_i}.\, \bigstar_{1 \leq j \leq n_i} \psi_{i,j}$ for $1 \leq i \leq 2$ such that $\mathbf{e_1} \cap \mathbf{e_2} = \emptyset$. We say that ζ is a* re-scoping *of ϕ_1 and ϕ_2 if there exist formulas $\psi'_{1,1}, \ldots, \psi'_{2,n_2}$ such that (1) $\psi_{i,j} \xrightarrow{\exists \mathbf{e_{3-i}}/\forall \varepsilon} \psi'_{i,j}$ for all i, j; and (2) $\zeta = \exists \mathbf{e_1} \cdot \mathbf{e_2}.\, \bigstar_{1 \leq j \leq n_1} \psi'_{1,j} \star \bigstar_{1 \leq j \leq n_2} \psi'_{2,j}$.*

Definition 4.18 (Derivability)**.** *We say that ψ is derivable from $\zeta = \exists \mathbf{e}.\, \bigstar_{1 \leq i \leq n} \forall \mathbf{a_i}.\, \zeta_i$, written $\zeta \rhd \psi$, iff there exist indices m_1 and m_2, variables $\mathbf{u_1}, \mathbf{u_2}, \mathbf{b_1}, \mathbf{b_2}$, and formulas ϕ_1, ϕ_2, ϕ such that (1) $\forall \mathbf{a_{m_i}}.\, \zeta_{m_i} \xrightarrow{\exists \varepsilon/\forall \mathbf{u_i}} \forall \mathbf{b}_i.\, \phi_i$ for $1 \leq i \leq 2$, (2) $\phi_1 \star \phi_2 \implies \phi$ by ($\ddagger$) and (3) ψ is obtained from ζ by removing the subformulas $\forall \mathbf{a_{m_i}}.\, \zeta_{m_i}$ and adding the subformula $\forall (\mathbf{b_1} \cup \mathbf{b_2}).\, \phi$.*

We are finally ready to define a composition operation $\bullet_{\mathbf{P}}$ that satisfies ($\dagger$).

Definition 4.19 (Projection composition)**.** *Let $\phi_1, \phi_2 \in \mathbf{DUSH}_{\Phi}$. Then*

$$\phi_1 \bullet_{\mathbf{P}} \phi_2 := \{\phi \mid \text{there exist a } k \geq 1 \text{ and } \zeta_1, \ldots, \zeta_k \text{ s.t. } \zeta_1 \text{ is a re-scoping of } \phi_1 \text{ and } \phi_2, \\ \zeta_i \rhd \zeta_{i+1} \text{ for all } 1 \leq i < k, \text{ and } \zeta_k = \phi\}.$$

Example 4.20.
- *For* $\phi_1 = \mathsf{ls}(x_2, x_3) \mathrel{-\!\!\star} \mathsf{ls}(x_1, x_3)$ *and* $\phi_2 = \mathbf{emp} \mathrel{-\!\!\star} \mathsf{ls}(x_2, x_3)$*, it holds that* $\phi_1 \star \phi_2 \rhd \mathbf{emp} \mathrel{-\!\!\star} \mathsf{ls}(x_1, x_3)$*. Hence,* $(\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ls}(x_1, x_3)) \in \phi_1 \bullet_{\mathbf{P}} \phi_2$*.*
- *Let* $\phi_1 = \forall a.\, \mathsf{ls}(x_2, a) \mathrel{-\!\!\star} \mathsf{ls}(x_1, a)$ *and* $\phi_2 = \forall b.\, \mathsf{ls}(x_3, b) \mathrel{-\!\!\star} \mathsf{ls}(x_2, b)$*,* $\phi_1' = \forall c.\, \mathsf{ls}(x_2, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)$*, and* $\phi_2' = \forall c.\, \mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_2, c)$*. Observe that* $\phi_i \xrightarrow{\exists \varepsilon / \forall c} \phi_i'$ *and that* $(\mathsf{ls}(x_2, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)) \star (\mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_2, c)) \rhd \mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)$*. Hence,* $(\forall c.\, \mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)) \in \phi_1 \bullet_{\mathbf{P}} \phi_2$*.*
- *Let* $\phi_1 = \exists e_1.\, (\mathsf{treerp}(y, e_1) \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, e_1))$ *and* $\phi_2 = \mathbf{emp}$*. Then* $\psi = \exists e_1.\, (\mathsf{treerp}(y, e_1) \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, e_1))$ *is a re-scoping of* ϕ_1 *and* ϕ_2 *and* $(\mathsf{treerp}(y, e_1) \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, e_1)) \rhd \mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)$*, so* $(\exists e_1.\, \mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \in \phi_1 \bullet_{\mathbf{P}} \phi_2$*.*

Lemma 4.21 (Soundness)**.** *Let* $\mathfrak{s}$ *be a stack, let* $\mathfrak{h}_1, \mathfrak{h}_2$ *be heaps, and* $\phi_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1), \phi_2 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$*. If* $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$ *and* $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{M}^{\mathbf{g}}_\Phi$ *then* $(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2) \models_\Phi \psi$ *for all* $\psi \in \phi_1 \bullet_{\mathbf{P}} \phi_2$*.*

Moreover, $\bullet_{\mathbf{P}}$ satisfies (†) when applied to disjoint models. The only missing piece is to ensure that the composition is only defined if the underlying heaps can be composed via $+$. This is easy to ensure, as it is possible to infer from $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ the set of variables $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h})$: Let $\mathcal{T}$ be a Φ-type. We define the set of allocated variables of $\mathcal{T}$ as

$$\mathsf{alloced}(\mathcal{T}) := \{x \mid \text{there ex. } \phi \in \mathcal{T} \text{ and } (\psi \mathrel{-\!\!\star} \mathsf{pred}(\mathbf{z})) \in \phi \text{ s.t. } x = \mathsf{predroot}(\mathsf{pred}(\mathbf{z}))\}\}.$$

Lemma 4.22. *For all models* $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{M}^{\mathbf{g}}_\Phi$*, it holds that* $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))$*.*

This allows us to check for double allocation in the definition of composition.

Definition 4.23 (Type composition)**.** *Let* $\mathcal{T}_1, \mathcal{T}_2$ *be types. The* composition *of* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *is*

$$\mathcal{T}_1 \bullet \mathcal{T}_2 := \begin{cases} \bot, & \text{if } \mathsf{alloced}(\mathcal{T}_1) \cap \mathsf{alloced}(\mathcal{T}_2) \neq \emptyset \\ \phi_1 \bullet_{\mathbf{P}} \phi_2, & \text{otherwise.} \end{cases}$$

Furthermore, $\mathsf{type}_\Phi(\mathfrak{s}, \cdot)$ is the desired homomorphism (†) from heaps and $+$ to types and $\bullet$.

Theorem 4.24. *Let* $\mathfrak{s}$ *be a stack and Let* $\mathfrak{h}_1, \mathfrak{h}_2$ *be heaps with* $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$ *and* $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{M}^{\mathbf{g}}_\Phi$*. Then* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$*.*

We use Φ-types to define an abstraction of formulas.

Definition 4.25 ($\mathfrak{s}$-Types of a formula)**.** *Let* $\phi \in \mathbf{SL}^{\mathbf{g}}_{\mathsf{btw}}$*. The* $\mathfrak{s}$*-types of* ϕ *are given by* $\mathbf{Types}^{\mathfrak{s}}_\Phi(\phi) := \{\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \mid \mathfrak{h} \text{ heap}, (\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi\}$*.*

There are only finitely many Φ-types for every fixed set of variables.

Lemma 4.26. *For SID* Φ *and stack* $\mathfrak{s}$*, let* $n := |\Phi| + |\mathrm{dom}(\mathfrak{s})|$*. Then* $|\mathbf{Types}^{\mathfrak{s}}(\Phi)| \in 2^{2^{\mathcal{O}(n^2 \log(n))}}$*.*

Theorem 4.24 and Lemma 4.26 make it possible to effectively compute the set of all types of the predicates of an SID. Intuitively, this can be achieved by a fixed-point computation: In the first iteration, we compute the types of the models of non-recursive rules of the SID; in later iterations, we compute the types of models of recursive rules by combining the types discovered in previous iterations by means of the composition operation, $\bullet$. We showed in [26] that this approach yields the types of all predicates in double-exponential time.

Theorem 4.27 ([26])**.** *Let* Φ *be a SID, let* $\mathsf{pred}(\mathbf{y})$ *be a predicate call, and let* $\mathfrak{s}$ *be a stack with* $\mathbf{y} \subseteq \mathrm{dom}(\mathfrak{s})$*. Let* $n := |\Phi| + |\mathrm{dom}(\mathfrak{s})|$*. The set* $\mathbf{Types}^{\mathfrak{s}}_\Phi(\mathsf{pred}(\mathbf{y}))$ *can be computed in* $2^{2^{\mathcal{O}(n^2 \log(n))}}$*.*

5 Deciding Guarded Separation Logic

In this final technical section, we address two questions. First, how to compute the Φ-types of arbitrary $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ formulas. Second, how to use Φ-types to decide satisfiability and entailment.

In this section, we assume that Φ is a fixed SID that is *pointer-closed*, by which me mean that Φ contains predicates predicate $\mathsf{ptr}_1, \ldots, \mathsf{ptr}_k$ defined by the rule $\mathsf{ptr}_k(\mathbf{x}) \Leftarrow x_1 \mapsto \langle x_2, \ldots, x_{k+1} \rangle$, where k is the largest number of variables that occur on the right-hand side of points-to assertions in our satisfiability and entailment queries.

Our crucial insight is that Φ-types (of pointer-closed SIDs) *refine* the satisfaction relation of $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$: Models with the same types satisfy the same formulas. To show this for formulas with separating connectives, we need a way to reverse composition: we need to show that if $\mathcal{T} = \mathcal{T}_1 \bullet \mathcal{T}_2$, then the models of the type $\mathcal{T}$ are a composition of models of types $\mathcal{T}_1$ and $\mathcal{T}_2$.

Lemma 5.1 (Decomposition lemma)**.** *Let $\mathfrak{s}$ be a stack, let $\mathfrak{h}', \mathfrak{h}_1, \mathfrak{h}_2$ be heaps such that $(\mathfrak{s}, \mathfrak{h}'), (\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{M}^{\mathsf{g}}_{\Phi}$ and $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}') = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2)$. Then there exist $\mathfrak{h}'_1, \mathfrak{h}'_2$ such that $(\mathfrak{s}, \mathfrak{h}'_1), (\mathfrak{s}, \mathfrak{h}'_2) \in \mathbf{M}^{\mathsf{g}}_{\Phi}$, $\mathfrak{h}' = \mathfrak{h}'_1 + \mathfrak{h}'_2$ and $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}'_i)$ for $1 \leq i \leq 2$.*

Using the decomposition lemma for the separating connectives $\star$, $\mathrel{-\!\!\star}$ and $\circledast$, the *refinement theorem* can be proved by a straightforward induction.

Theorem 5.2 (Refinement theorem)**.** *Let $\mathfrak{s}$ be a stack and $\mathfrak{h}_1, \mathfrak{h}_2$ be heaps. Let $\phi \in \mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$. Moreover, assume $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. Then $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$ iff $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi$.*

We rely on Theorems 4.27 and 5.2 in our algorithm for computing the types of arbitrary $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ formulas, presented in Fig. 7. We explain why the algorithm is correct.

1. The formulas **emp**, $x \approx y$ and $x \not\approx y$ only hold in empty heaps and **emp** is the only DUSH that holds in the empty heap.

2. To compute $\mathsf{type}_\Phi(\mathfrak{s}, \{\mathfrak{s}(a) \mapsto \mathfrak{s}(\mathbf{b})\})$, we check for all rule instances of the SID whether they contain the pointer $a \mapsto \mathbf{b}$; if so, we take a single-tree Φ-forest that contains this rule instance and nothing else and project this forest onto a formula. All such formulas make up the type of $a \mapsto \mathbf{b}$. The details of this construction are in [26].

3. The set $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\mathsf{pred}(\mathbf{y}))$ is computable by Theorem 4.27. We present this computation in detail in [23, 26].

4. Theorem 4.24 guarantees that the types of $\phi_1 \star \phi_2$ can be computed via $\bullet$.

5. Theorem 5.2 guarantees that the Boolean operators can be implemented via set operations.

6. Magic wand and septraction are implemented by lifting their semantics from individual models to types in a straightforward way.

7. Guardedness guarantees that throughout the computation, we only need to consider guarded models. This is necessary for applying Theorems 4.24 and 5.2.

The asymptotic complexity of each operation is bounded by the size of the type abstraction. We thus obtain a double-exponential decision procedure for $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ for every *fixed* stack.

Theorem 5.3. *Let $\phi \in \mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ with $\mathsf{fvars}(\phi) = \mathbf{x}$. Let $\mathfrak{s}$ be a stack with $\mathrm{dom}(\mathfrak{s}) = \mathbf{x}$. Then (1) $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) = \mathsf{types}(\phi, \mathfrak{s})$ and (2) $\mathsf{types}(\phi, \mathfrak{s})$ can be computed in $2^{2^{\mathcal{O}(n^2 \log(n))}}$.*

$$
\begin{array}{lcl}
\mathsf{types}(\mathbf{emp}, \mathfrak{s}) & := & \{\{\mathbf{emp}\}\} \\
\mathsf{types}(x \approx y, \mathfrak{s}) & := & \text{if } \mathfrak{s}(x) = \mathfrak{s}(y) \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset \\
\mathsf{types}(x \not\approx y, \mathfrak{s}) & := & \text{if } \mathfrak{s}(x) \neq \mathfrak{s}(y) \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset \\
\mathsf{types}(a \mapsto \mathbf{b}, \mathfrak{s}) & := & \{\mathsf{type}_\Phi(\mathfrak{s}, \{\mathfrak{s}(a) \mapsto \mathfrak{s}(\mathbf{b})\})\} \\
\mathsf{types}(\mathsf{pred}(\mathbf{y}), \mathfrak{s}) & := & \mathbf{Types}^{\mathfrak{s}}_{\Phi}(\mathsf{pred}(\mathbf{y})) \\
\mathsf{types}(\phi_1 \star \phi_2, \mathfrak{s}) & := & \{\mathcal{T}_1 \bullet \mathcal{T}_2 \mid \mathcal{T}_1 \in \mathsf{types}(\phi_1, \mathfrak{s}), \mathcal{T}_2 \in \mathsf{types}(\phi_2, \mathfrak{s})\} \\
\mathsf{types}(\phi_1 \wedge \phi_2, \mathfrak{s}) & := & \mathsf{types}(\phi_1, \mathfrak{s}) \cap \mathsf{types}(\phi_2, \mathfrak{s}) \\
\mathsf{types}(\phi_1 \vee \phi_2, \mathfrak{s}) & := & \mathsf{types}(\phi_1, \mathfrak{s}) \cup \mathsf{types}(\phi_2, \mathfrak{s}) \\
\mathsf{types}(\phi_1 \wedge \neg\phi_2, \mathfrak{s}) & := & \mathsf{types}(\phi_1, \mathfrak{s}) \setminus \mathsf{types}(\phi_2, \mathfrak{s}) \\
\mathsf{types}(\phi_0 \wedge (\phi_1 \circledast \phi_2), \mathfrak{s}) & := & \{\mathcal{T} \in \mathsf{types}(\phi_0, \mathfrak{s}) \mid \exists \mathcal{T}' \in \mathsf{types}(\phi_1, \mathfrak{s}).\, \mathcal{T} \bullet \mathcal{T}' \in \mathsf{types}(\phi_2, \mathfrak{s})\} \\
\mathsf{types}(\phi_0 \wedge (\phi_1 \mathrel{-\!\!\star} \phi_2), \mathfrak{s}) & := & \{\mathcal{T} \in \mathsf{types}(\phi_0, \mathfrak{s}) \mid \forall \mathcal{T}' \in \mathsf{types}(\phi_1, \mathfrak{s}).\, \mathcal{T} \bullet \mathcal{T}' \in \mathsf{types}(\phi_2, \mathfrak{s})\}
\end{array}
$$

Figure 7: Computing the Φ-types of guarded formula $\phi \in \mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ for models with stack $\mathfrak{s}$.

Now that we know that $\mathsf{types}(\phi, \mathfrak{s})$ computes $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi)$, we are ready to prove that satisfiability of $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ formulas is decidable in double-exponential time.

Theorem 5.4. *Let Φ be an SID and $\phi \in \mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$. Let $n := |\Phi| + |\phi|$. It is decidable in time $2^{2^{\mathcal{O}(n^2 \log(n))}}$ whether ϕ is satisfiable.*

Proof. Consider the *aliasing constraint* of stack $\mathfrak{s}$, $\{(x, y) \mid x, y \in \mathrm{dom}(\mathfrak{s}) \text{ and } \mathfrak{s}(x) = \mathfrak{s}(y)\}$. If $\mathfrak{s}$ and $\mathfrak{s}'$ have the same aliasing constraint, then $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) = \mathbf{Types}^{\mathfrak{s}'}_{\Phi}(\phi)$. Since there are only exponentially many (in n) different aliasing constraints, the claim follows from Theorem 5.3. □

Since the entailment query $\phi \models_\Phi \psi$ is equivalent to checking the unsatisfiability of $\phi \wedge \neg\psi$, and the negation in $\phi \wedge \neg\psi$ is guarded, we obtain an entailment checker with the same complexity.

Corollary 5.5. *Let Φ be a SID, $\phi, \psi \in \mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ and $n := |\Phi| + |\phi| + |\psi|$. The entailment problem $\phi \models_\Phi \psi$ is decidable in time $2^{2^{\mathcal{O}(n^2 \log(n))}}$.*

6 Conclusion

We developed a 2-Exptime algorithm for deciding $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$, an SL in which formulas are built from user-defined predicates from $\mathrm{SLID}_{\mathsf{btw}}$ [16] using $\star$, $\wedge$, $\vee$, and guarded forms of $\neg$, $\mathrel{-\!\!\star}$, and $\circledast$. The only previously known decidability results for *any* SL with user-defined inductive definitions are limited to symbolic heaps, with a matching 2-Exptime bound for $\mathrm{SLID}_{\mathsf{btw}}$ [26]. Further, we showed that removing the guard of any of the guarded operators of $\mathbf{SL}^{\mathsf{g}}_{\mathsf{btw}}$ leads to undecidability, obtaining an almost perfect delineation between decidability and undecidability. We leave the integration of our decision procedure into a Hoare-style verification framework for future work.

References

[1] Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2014.

[2] Andrew W. Appel. *Program Logics - for Certified Compilers*. Cambridge University Press, 2014.

[3] Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded negation. *J. ACM*, 62(3):22:1–22:26, 2015.

[4] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2004.

[5] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*, volume 4111 of *Lecture Notes in Computer Science*, pages 115–137. Springer, 2005.

[6] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Symbolic execution with separation logic. In Kwangkeun Yi, editor, *Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings*, volume 3780 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2005.

[7] Josh Berdine, Byron Cook, and Samin Ishtiaq. Slayer: Memory safety for systems-level code. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 178–183. Springer, 2011.

[8] Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. *Inf. Comput.*, 211:106–137, 2012.

[9] James Brotherston, Carsten Fuhs, Juan Antonio Navarro Pérez, and Nikos Gorogiannis. A decision procedure for satisfiability in separation logic with inductive predicates. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 25:1–25:10. ACM, 2014.

[10] Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, volume 9058 of *Lecture Notes in Computer Science*, pages 3–11. Springer, 2015.

[11] Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. *J. ACM*, 58(6):26:1–26:66, 2011.

[12] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local action and abstract separation logic. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 366–378. IEEE Computer Society, 2007.

[13] Wei-Ngan Chin, Cristina David, Huu Hai Nguyen, and Shengchao Qin. Automated verification of shape, size and bag properties via user-defined predicates in separation logic. *Sci. Comput. Program.*, 77(9):1006–1036, 2012.

[14] Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 2011.

[15] Mnacho Echenim, Radu Iosif, and Nicolas Peltier. The lower bound of decidable entailments in separation logic with inductive definitions. *hal-02388028*, 2019.

[16] Radu Iosif, Adam Rogalewicz, and Jirí Simácek. The tree width of separation logic with recursive

definitions. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2013.

[17] Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2014.

[18] Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 14–26. ACM, 2001.

[19] Bart Jacobs, Jan Smans, Pieter Philippaerts, Frédéric Vogels, Willem Penninckx, and Frank Piessens. Verifast: A powerful, sound, predictable, fast verifier for C and java. In Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*, volume 6617 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2011.

[20] Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. Unified reasoning about robustness properties of symbolic-heap separation logic. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10201 of *Lecture Notes in Computer Science*, pages 611–638. Springer, 2017.

[21] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.*, 28:e20, 2018.

[22] Jens Katelaan, Dejan Jovanovic, and Georg Weissenbacher. A separation logic with data: Small models and automation. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 455–471. Springer, 2018.

[23] Jens Katelaan, Christoph Matheja, and Florian Zuleger. Effective entailment checking for separation logic with inductive definitions. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 319–336. Springer, 2019.

[24] Peter Müller, Malte Schwerhoff, and Alexander J. Summers. Viper: A verification infrastructure for permission-based reasoning. In Alexander Pretschner, Doron Peled, and Thomas Hutzelmann, editors, *Dependable Software Systems Engineering*, volume 50 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 104–125. IOS Press, 2017.

[25] Peter W. O'Hearn. Separation logic. *Commun. ACM*, 62(2):86–95, 2019.

[26] Jens Pagel, Christoph Matheja, and Florian Zuleger. Complete entailment checking for separation logic with inductive definitions. *CoRR*, abs/2002.01202, 2020.

[27] Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic using SMT. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 773–789. Springer, 2013.

[28] Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic with trees and data. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International*

Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings, volume 8559 of *Lecture Notes in Computer Science*, pages 711–728. Springer, 2014.

[29] Ruzica Piskac, Thomas Wies, and Damien Zufferey. Grasshopper - complete heap verification with mixed specifications. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2014.

[30] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 55–74. IEEE Computer Society, 2002.

[31] Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. Automated lemma synthesis in symbolic-heap separation logic. *PACMPL*, 2(POPL):9:1–9:29, 2018.

[32] Makoto Tatsuta and Daisuke Kimura. Separation logic with monadic inductive definitions and implicit existentials. In Xinyu Feng and Sungwoo Park, editors, *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*, volume 9458 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2015.

[33] Makoto Tatsuta, Koji Nakazawa, and Daisuke Kimura. Completeness of cyclic proofs for symbolic heaps with inductive definitions. In Anthony Widjaja Lin, editor, *Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*, volume 11893 of *Lecture Notes in Computer Science*, pages 367–387. Springer, 2019.