A Decision Procedure for Guarded Separation Logic: Complete Entailment Checking for Separation Logic with Inductive Definitions

Matheja, Christoph; Pagel, Jens; Zuleger, Florian

Published in:
ACM Transactions on Computational Logic

Link to article, DOI:
10.1145/3534927

Publication date:
2023

Document Version
Peer reviewed version

A Decision Procedure for Guarded Separation Logic

Complete Entailment Checking for Separation Logic with Inductive Definitions

CHRISTOPH MATHEJA, Technical University of Denmark, Denmark, and ETH Zurich, Switzerland
JENS PAGEL, TU Wien, Austria
FLORIAN ZULEGER, TU Wien, Austria

We develop a doubly-exponential decision procedure for the satisfiability problem of *guarded separation logic*—a novel fragment of separation logic featuring user-supplied inductive predicates, Boolean connectives, and separating connectives, including restricted (guarded) versions of negation, magic wand, and septraction. Moreover, we show that dropping the guards for any of the above connectives leads to an undecidable fragment.

We further apply our decision procedure to reason about *entailments* in the popular symbolic heap fragment of separation logic. In particular, we obtain a doubly-exponential decision procedure for entailments between (quantifier-free) symbolic heaps with inductive predicate definitions of bounded treewidth ($\mathbf{SL}_{\mathrm{btw}}$)—one of the most expressive decidable fragments of separation logic. Together with the recently shown 2ExpTime-hardness for entailments in said fragment, we conclude that the entailment problem for $\mathbf{SL}_{\mathrm{btw}}$ is 2ExpTime-complete—thereby closing a previously open complexity gap.

CCS Concepts: • **Theory of computation → Automated reasoning**; **Separation logic**; **Abstraction**; **Logic and verification**; **Problems, reductions and completeness**.

Additional Key Words and Phrases: decision procedures, entailment, magic wands, inductive predicates

1 INTRODUCTION

Separation Logic (SL) [Ishtiaq and O'Hearn 2001; Reynolds 2002] is a popular formalism for Hoare-style verification of imperative, heap-manipulating programs. At its core, SL extends first-order logic with two connectives—the *separating conjunction* $\star$ and the *separating implication* $\mathrel{-\!\!\star}$ (aka magic wand)—for concisely specifying how resources, such as program memory, can be split-up and extended, respectively. Based on these connectives, SL enables *local reasoning*, i.e., sound verification of program parts in isolation, about the resources employed by a program—a key property responsible for SL's broad adoption in static analysis [Berdine et al. 2007; Calcagno and Distefano 2011; Calcagno et al. 2015, 2011; Gotsman et al. 2007], automated verification [Berdine et al. 2005a, 2011; Chin et al. 2012; Jacobs et al. 2011; Müller et al. 2017; Piskac et al. 2014b; Ta et al. 2018], and interactive theorem proving [Appel 2014; Jung et al. 2018].

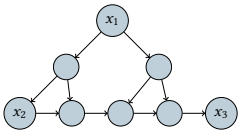
Regardless of the flavor of formal reasoning, any *automated* approach based on SL ultimately relies on a solver for discharging either the *satisfiability problem*—does the SL formula ϕ have a model?—or the *entailment problem*—is every model of ϕ also a model of ψ, or, equivalently, is $\phi \wedge \neg\psi$ unsatisfiable? While both problems are undecidable (and equivalent) in general [Calcagno et al. 2001], various decidable SL fragments, in which entailments cannot be reduced to the (un)satisfiability problem because negation is forbidden, have been proposed in the literature, e.g., [Berdine et al. 2004; Cook et al. 2011; Echenim et al. 2020a; Iosif et al. 2013].

Authors' addresses: Christoph Matheja, Technical University of Denmark, Denmark, and ETH Zurich, Switzerland, cmatheja@inf.ethz.ch; Jens Pagel, pagel@forsyte.at, TU Wien, Vienna, Austria; Florian Zuleger, TU Wien, Vienna, Austria, zuleger@forsyte.at.

1529-3785/2022/9-ART $15.00
https://doi.org/10.1145/3534927

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

(a) An SID specifying trees with linked leaves.



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

Fig. 1. The SID of Iosif et al. [2013] defining trees with linked lives and an illustration of a model.

In particular, the *symbolic heap* fragment—an idiomatic form of SL formulas with $\star$ but without $-\!\!\star$ that is often encountered when manually writing program proofs [Berdine et al. 2005b]—has received a lot of attention. Symbolic heaps appear, for instance, in the automated tools Infer [Calcagno and Distefano 2011], Sleek [Chin et al. 2012], Songbird [Ta et al. 2016], Grasshopper [Piskac et al. 2014b], Verifast [Jacobs et al. 2011], SLS [Ta et al. 2018], and Spen [Enea et al. 2017]. To support complex data structure specifications, symbolic heaps are often enriched with *systems of inductive predicate definitions* (SIDs). For example, Fig. 1 depicts an SID specifying trees with linked leaves as well as an illustration of a model (*nil*-pointers have been omitted for readability).

The precise form of permitted SIDs has a significant impact on the decidability and complexity of reasoning about symbolic heaps: Brotherston et al. [2014] showed that *satisfiability* is ExpTime-complete for symbolic heaps over arbitrary SIDs, whereas the *entailment problem* is undecidable in general (cf. [Antonopoulos et al. 2014; Iosif et al. 2014]). To deal with entailments, tools rely on specialized methods for fixed predicates [Berdine et al. 2004; Cook et al. 2011; Piskac et al. 2013, 2014a], decision procedures for restricted classes of SIDs [Iosif et al. 2013, 2014], or incomplete approaches, e.g., fold/unfold reasoning [Chin et al. 2012] or cyclic proofs [Brotherston et al. 2011].

Among the largest decidable classes of symbolic heaps with user-supplied SIDs is the fragment of *symbolic heaps with bounded treewidth* ($\mathbf{SL}_{\mathrm{btw}}$) developed by Iosif et al. [2013], which supports rich data structure definitions, such as the one in Fig. 1. Further examples include doubly-linked lists and binary trees with parent pointers. Decidability is achieved by imposing three syntactic conditions on SIDs, which allow reducing the entailment problem for $\mathbf{SL}_{\mathrm{btw}}$ to the (decidable) satisfiability problem for monadic second-order logic (MSO) over graphs of bounded treewidth (cf. [Courcelle and Engelfriet 2012]). This reduction yields an elementary decision procedure (by analyzing the resulting quantifier depth, it is in 4ExpTime). However, it is infeasible in practice. Furthermore, there is a "complexity gap" between the above decision procedure and a recent result proving that the entailment problem for $\mathbf{SL}_{\mathrm{btw}}$ is at least 2ExpTime-hard [Echenim et al. 2020b]

The goal of this article is twofold: First, we look beyond symbolic heaps and study *guarded separation logic* (**GSL**)—a novel SL fragment featuring both standard Boolean and separating connectives (including restricted forms of negation and magic wand) as well as SIDs supported by $\mathbf{SL}_{\mathrm{btw}}$. In particular, we develop a doubly-exponential decision procedure for the *satisfiability problem* of **GSL**. Second, we show that the *entailment problem* for $\mathbf{SL}_{\mathrm{btw}}$ can be reduced to the satisfiability problem for **GSL** because an $\mathbf{SL}_{\mathrm{btw}}$ formula ϕ entails an $\mathbf{SL}_{\mathrm{btw}}$ formula ψ iff the **GSL** formula $\phi \wedge \neg\psi$ is unsatisfiable. Consequently, we close the aforementioned complexity gap and conclude that the entailment problem for $\mathbf{SL}_{\mathrm{btw}}$ is *2ExpTime-complete*.

Guarded separation logic. Inspired by work on first-order logic with *guarded negation*, we propose the fragment **GSL** of *guarded separation logic*. **GSL** supports negation $\neg$, magic wand $-\!\!\star$ and septraction $\circledast$ [Brochenin et al. 2012], but requires each of these connectives to appear in conjunction with another **GSL** formula ϕ, i.e., $\phi \wedge \neg\psi$, $\phi \wedge (\psi -\!\!\star\, \vartheta)$, or $\phi \wedge (\psi \circledast \vartheta)$, acting as its *guard*; hence, the name. By construction, a guard is never equivalent to true and thus cannot be dropped.

While we consider the satisfiability problem of quantifier-free **GSL** formulas, we admit arbitrary inductive predicates as long as they can be defined in $\mathbf{SL}_{\mathrm{btw}}$, which supports existential quantifiers. Hence, the formulas

below belong to **GSL** and are thus covered by our decision procedure.

$$\mathsf{tll}(x, y, z) \wedge \neg x \mapsto \langle nil, nil, z\rangle \qquad \text{(a tree with linked leaves and at least three nodes)}$$
$$(x \mapsto \langle y, z\rangle \star \mathsf{tll}(y, \ell, r) \star \mathsf{tll}(z, r, nil)) \wedge \neg \mathsf{tll}(x, \ell, nil) \qquad \text{(encoding of an entailment in } \mathbf{SL}_{\mathrm{btw}})$$
$$(x \mapsto \langle y, z\rangle \star \mathsf{tll}(z, r, nil)) \wedge (\mathsf{tll}(y, \ell, r) \mathrel{-\!\!\star} \mathsf{tll}(x, \ell, nil)) \qquad \text{(the root's left subtree is missing)}$$

Abstraction-based satisfiability checking. Our decision procedure for **GSL** satisfiability—and thus also for $\mathbf{SL}_{\mathrm{btw}}$ entailments—is based on the *compositional computation* of an *abstraction* of program states, i.e., the universe of potential models, that *refines* the satisfaction relation $\models$ of **GSL**.[1] That is, we will develop an abstract domain $\mathbb{A}$ and an abstraction function $\mathsf{abst} : \mathbf{States} \to \mathbb{A}$ with the following three key properties:

(1) *Refinement.* Whenever $\mathsf{abst}(\sigma) = \mathsf{abst}(\sigma')$ holds for two states σ and σ', then σ and σ' satisfy the same **GSL** formulas, i.e., the equivalence relation induced by our abstraction function,
$$\sigma \equiv_{\mathsf{abst}} \sigma' \quad \text{iff} \quad \mathsf{abst}(\sigma) = \mathsf{abst}(\sigma'),$$
refines the satisfaction relation $\models$ of **GSL**.
(2) *Compositionality.* For each logical connective supported by **GSL**, the abstraction function abst can be computed compositionally from already known abstractions. For example, for the separating conjunction $\star$, this means that there exists an effectively computable operation $\bullet : \mathbb{A} \times \mathbb{A} \to \mathbb{A}$ such that, for all states σ and σ',
$$\mathsf{abst}(\sigma \uplus \sigma') = \mathsf{abst}(\sigma) \bullet \mathsf{abst}(\sigma'),$$
where $\sigma \uplus \sigma'$ denotes the "disjoint union" of two states used to assign semantics to the separating conjunction.
(3) *Finiteness.* The abstract domain $\mathbb{A}$ has only finitely many elements after fixing the number of free variables.

Put together, refinement and compositionality allow lifting the abstraction function over states

$$\mathsf{abst} : \mathbf{States} \to \mathbb{A}$$

to a function over models of **GSL** formulas

$$\mathsf{abst}_{\mathrm{GSL}} : \mathbf{GSL} \to \mathbb{A}, \qquad \phi \mapsto \{\mathsf{abst}(\sigma) \mid \sigma \in \mathbf{States}, \sigma \models \phi\},$$

Provided we can compute the abstraction $\mathsf{abst}_{\mathrm{GSL}}$ of every atomic **GSL** formula, we can then use $\mathsf{abst}_{\mathrm{GSL}}$ for satisfiability checking: The **GSL** formula ϕ is satisfiable iff $\mathsf{abst}_{\mathrm{GSL}}(\phi) \neq \emptyset$. Finiteness then ensures that the set $\mathsf{abst}_{\mathrm{GSL}}(\phi)$ is finite; it can thus be computed and checked for emptiness.

We will provide a more detailed overview in Section 6 of our abstraction once we have precisely defined the semantics of guarded separation logic formulas.

Contributions. The main contributions of this article can be summarized as follows:

- We study the decidability of (quantifier-free) *guarded* separation logic (**GSL**)—a novel separation logic fragment that goes beyond symbolic heaps with user-defined inductive definitions by featuring restricted (guarded) versions of the magic wand, septraction, and negation.
- We show that omitting the guards for *any* of the three operators — magic wand, septraction, and negation — leads to an undecidable logic. Together with our decidability results, this yields an almost tight decidability delineation between separation logics that admit user-defined inductive predicate definitions.
- We present a decision procedure for the satisfiability problem of **GSL** based on the compositional computation of finite abstractions, called Φ-types, of potential models.
- We analyze the complexity of the above decision procedure and show that satisfiability of **GSL** is decidable in 2ExpTime.

[1] We will properly formalize all notions mentioned in this section in the remainder of this article.

- We apply our decision procedure for **GSL** to decide, again in 2ExpTime, the *entailment* problem for (quantifier-free) symbolic heaps with user-defined inductive definitions of bounded-tree; in light of the recently shown 2ExpTime-hardness by Echenim et al. [2020b], we obtain that said entailment problem is 2ExpTime-*complete*—thereby closing an existing complexity gap.

This article unifies and revises the results of two conference papers [Katelaan et al. 2019; Katelaan and Zuleger 2020]. We note that both papers only sketch the main ideas and most of the proofs were omitted. In this article, we dedicate a whole section to the careful motivation of our abstraction (see Section 6) and present all the proofs (an early version of this article was put on arXiv [Pagel et al. 2020] in order to convince the reviewers of [Katelaan and Zuleger 2020] of the correctness of our results). We remark that we have significantly improved the presentation and reworked all the technical details in comparison to our earlier technical report [Pagel et al. 2020].

Organization of the article. After agreeing on basic notational conventions in Section 2, we briefly recap separation logic in Section 3. In particular, we consider user-defined inductive definitions and the bounded treewidth fragment upon which our own SL fragments are based.

We introduce the novel fragment of guarded separation logic in Section 4. Section 5 shows that even small extensions of guarded separation logic lead to an unsatisfiable satisfiability problem. The remainder of this article is concerned with developing a decision procedure for guarded separation logic and, by extension, the entailment problem for symbolic heaps with inductive definitions of bounded treewidth. Section 6 informally discusses the main ideas underlying our decision procedure. The formal details are worked out in Sections 7 to 9. In particular, in Section 9, we present the decision procedure itself and analyze its complexity. Finally, we conclude in Section 10. To improve readability, some technical proofs have been moved to the Appendix at the end of this article.

Acknowledgments. We thank Mnacho Echenim, Radu Iosif, and Nicolas Peltier for their outstandingly thorough study of [Katelaan et al. 2019], which presented the originally proposed abstraction-based decision procedure, and their help in discovering an incompleteness issue, which we were able to fix in our follow-up work [Katelaan and Zuleger 2020; Pagel et al. 2020].

2 NOTATION

Throughout this article, we adhere to the following basic notational conventions.

Sequences. Finite sequences are denoted either in boldface, e.g., $\mathbf{x}$, or by explicitly listing their elements, e.g., $\langle x_1, \dots, x_k \rangle$; the *empty sequence* is $\langle\rangle$. The *length* of the sequence $\mathbf{x}$ is $|\mathbf{x}|$. We call $\mathbf{x}$ *repetition-free* if its elements are pairwise different. To reduce notational clutter, we often omit the brackets around sequences of length one, i.e., we write x instead of $\langle x \rangle$. The sequence $\mathbf{x} \cdot \mathbf{y}$ is obtained from *concatenating* the sequences $\mathbf{x}$ and $\mathbf{y}$. A^* is the *set of all finite sequences* over some set A; A^+ is the set of all *non-empty* finite sequences over A.

Sets from sequences. We frequently treat sequences as sets if the ordering of elements is irrelevant. For example, $x \in \mathbf{x}$ states that the sequence $\mathbf{x}$ contains the element x, $\mathbf{x} \cup \mathbf{y}$ is the set consisting of all elements of the sequences $\mathbf{x}$ and $\mathbf{y}$, etc.

Partial functions. We denote by $f : A \rightharpoonup B$ a *(partial) function* with *domain* $\mathrm{dom}(f) \triangleq A$ and *image* $\mathrm{img}(f) \triangleq B$. If f is undefined on x, i.e., $x \notin \mathrm{dom}(f)$, we write $f(x) = \bot$. Moreover, $f \circ g$ is the *composition* of the functions f and g mapping every x to $f(g(x))$. We interpret the *size* $|f|$ of a partial function f as the cardinality of its domain, i.e., $|f| \triangleq |\mathrm{dom}(f)|$; f is *finite* if $|f|$ is finite.

We often describe finite partial functions as sets of mappings. The set $\{x_1 \mapsto y_1, \dots, x_k \mapsto y_k\}$, for example, represents the partial function that, for every $i \in [1, k]$, maps x_i to y_i; it is undefined for all other values. Furthermore, $f \cup g$ denotes the (not necessarily disjoint) union of f and g; it is defined iff $f(x) = g(x)$ holds for

$$
\begin{array}{lll}
 & & x \in \mathbf{Var},\ u, v \in \mathbf{Var} \cup \mathbf{Loc},\ \mathbf{w} \in (\mathbf{Var} \cup \mathbf{Loc})^+,\ \mathrm{pred} \in \mathbf{Preds} \\
\phi_{\mathrm{atom}} & ::= & \mathbf{emp} \mid u \approx v \mid u \not\approx v \mid u \mapsto \mathbf{w} \mid \mathrm{pred}(\mathbf{w}) \\
\phi & ::= & \phi_{\mathrm{atom}} \mid \phi \star \phi \mid \phi \mathrel{-\!\!\star} \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \exists x.\, \phi \mid \forall x.\, \phi
\end{array}
$$

Fig. 2. The syntax of a first-order separation logic with user-defined predicates (**SL**)

all $x \in \mathrm{dom}(f) \cap \mathrm{dom}(g)$. Formally:

$$
(f \cup g)(x) \triangleq \begin{cases} f(x), & \text{if } x \in \mathrm{dom}(f), \\ g(x), & \text{if } x \in \mathrm{dom}(g) \setminus \mathrm{dom}(f), \\ \bot, & \text{otherwise.} \end{cases}
$$

We write $f \uplus g$ instead of $f \cup g$ whenever we additionally require that $\mathrm{dom}(f) \cap \mathrm{dom}(g) = \emptyset$. Furthermore, we denote by $f[x/v]$ the *updated* partial function in which x maps to v, i.e.,

$$
f[x/v](y) \triangleq \begin{cases} v & \text{if } y = x, \\ f(y), & \text{otherwise.} \end{cases}
$$

In particular, if $f(x)$ is undefined, $f[x/v]$ adds x to the domain of the resulting function. By slight abuse of notation, we write $f[x/\bot]$ to denote the function in which x is removed from the domain of f. To compare partial functions, we say that function g *subsumes* function f, written $f \subseteq g$, if (1) g is at least as defined as g and (2) g agrees with f on their common domain. Formally:

$$
f \subseteq g \quad \text{iff} \quad \mathrm{dom}(f) \subseteq \mathrm{dom}(g) \quad \text{and} \quad \forall x \in \mathrm{dom}(f) \colon f(x) = g(x).
$$

Whenever f and g map to sets or functions, we also use a weaker ordering. The relation $f \sqsubseteq g$ is defined as $f \subseteq g$ but only requires that $g(x)$ subsumes $f(x)$ if both are defined on x, i.e.,

$$
f \sqsubseteq g \quad \text{iff} \quad \mathrm{dom}(f) \subseteq \mathrm{dom}(g) \quad \text{and} \quad \forall x \in \mathrm{dom}(f) \colon f(x) \subseteq g(x).
$$

Functions over sequences. We implicitly lift partial functions $f \colon A \rightharpoonup B$ to functions $f \colon A^* \rightharpoonup B^*$ over sequences by pointwise application. That is, for a sequence $\langle a_1, \ldots, a_k \rangle \in A^*$, we define

$$
f(\langle a_1, \ldots, a_k \rangle) \triangleq f(a_1, \ldots, a_k) \triangleq \langle f(a_1), \ldots, f(a_k) \rangle,
$$

where, as indicated above, we omit the brackets indicating sequences to improve readability.

Finally, we lift the update $f[x/v]$ of a single value to sequences of values $\mathbf{x} = \langle x_1, \ldots, x_k \rangle$ and $\mathbf{v} = \langle v_1, \ldots, v_k \rangle$ by setting $f[\mathbf{x}/\mathbf{v}] \triangleq f[x_1/v_1][x_2/v_2] \ldots [x_k/v_k]$.

3 SEPARATION LOGIC WITH INDUCTIVE DEFINITIONS

We briefly recapitulate the basics of first-order separation logic with user-defined predicates. That is, we introduce the syntax and semantics of both separation logic and systems of inductive definitions, the symbolic heap fragment, and the bounded treewidth fragment originally studied by Iosif et al. [2013]. Most of the presented material is fairly standard (cf., among others, [Brotherston et al. 2014; Iosif et al. 2014; Ishtiaq and O'Hearn 2001; Reynolds 2002]) with the notable exception that *our semantics of pure formulas enforces the heap to be empty*. A reader familiar with separation logic may skim over this section to familiarize herself with our notation.

3.1 The Syntax of Separation Logic

Figure 2 defines the syntax of first-order separation logic with user-defined predicates (**SL** for short), where x is drawn from a countably infinite set **Var** of *variables*, u and v are either variables or *locations* drawn from the countably infinite set **Loc**, and $\mathbf{w}$ is a finite sequence whose elements can be both variables and locations.

In particular, notice that any location in **Loc** may appear as a constant in formulas. Moreover, pred is taken from a finite set **Preds** of *predicate identifiers*; each predicate pred is equipped with an *arity* $\mathrm{ar}(\mathrm{pred}) \in \mathbb{N}$ that determines its number of parameters.

Informally, the meaning of the atomic formulas is as follows:

- The *empty-heap predicate* **emp** denotes the empty heap.
- The *equality* $u \approx v$ and the *disequality* $u \not\approx v$ express that u and v alias and that they do not alias in the current program state (whose heap needs to be empty), respectively.
- The *points-to assertion* $u \mapsto \mathbf{w}$ states that the address u points to a heap-allocated object consisting of $|\mathbf{w}| > 0$ fields, where the i-th field stores the i-th location of the sequence $\mathbf{w}$.
- The *predicate call* $\mathrm{pred}(\mathbf{w})$ allows to refer to user-defined data structures, e.g., lists and trees.

The **SL** formulas **emp** and $u \mapsto v$ are called *spatial* atoms because they describe the spatial layout of the heap, whereas (dis-)equalities are called *pure* atoms [Ishtiaq and O'Hearn 2001] because they do not depend on the heap. Apart from atoms, **SL** formulas are built from

- classical propositional connectives, i.e., conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$),
- existential ($\exists$) and universal ($\forall$) quantifiers, and
- *separating connectives*, i.e., *separating conjunction* $\star$ and *implication* (or magic wand) $\mathrel{-\!\!\star}$.

As usual, one can derive additional operators such as standard implication $\phi \Rightarrow \psi \triangleq \neg\phi \vee \psi$ and *septraction* $\phi \circledast \psi \triangleq \neg(\phi \mathrel{-\!\!\star} \neg\psi)$ (cf. [Brochenin et al. 2012; Thakur et al. 2014]).

The semantics of the classical connectives is standard. Let us briefly compare their meaning with the intuition underlying the separating connectives. While $\phi \wedge \psi$ means that the program state satisfies both ϕ and ψ simultaneously, $\phi \star \psi$ denotes that (the heap component of) the program state can be split into two disjoint parts which separately satisfy ϕ and ψ. Similarly, while $\phi \Rightarrow \psi$ means that every program state satisfying ϕ also satisfies ψ, $\phi \mathrel{-\!\!\star} \psi$ means that the *extension* of the program state with any program state that satisfies ϕ yields a program state that satisfies ψ.

The magic wand is useful for weakest-precondition reasoning, e.g., to express memory allocation [Batz et al. 2019; Ishtiaq and O'Hearn 2001; Reynolds 2002]. However, automated verification tools often do not or only partially support the magic wand, because its inclusion quickly leads to undecidability [Appel 2014; Blom and Huisman 2015; Schwerhoff and Summers 2015].

3.1.1 Substitution. Various constructions in this article involve syntactically replacing variables and locations—we thus give a generic definition that allows performing multiple substitutions at once. Let $\mathbf{y}, \mathbf{z} \in (\mathbf{Var} \cup \mathbf{Loc})^*$ be sequences of the same length, where $\mathbf{y}$ is repetition-free. We denote by $\phi[\mathbf{y}/\mathbf{z}]$ the formula obtained from ϕ through (simultaneous) *substitution* of each element in $\mathbf{y}$ by the element in $\mathbf{z}$ at the same position; Appendix A.1 provides a formal definition. E.g.,

$$(\exists x.\, x \mapsto \langle y, 7\rangle \star \mathsf{ls}(y, x))[\langle y, 7\rangle / \langle 3, x\rangle] \;=\; \exists x.\, (x \mapsto \langle 3, x\rangle) \star \mathsf{ls}(3, x).$$

Moreover, we write $\phi(\mathbf{z})$ as a shortcut for the substitution $\phi[\mathrm{fvars}(\phi)/\mathbf{z}]$.

3.2 The Stack-Heap Model

We interpret **SL** in terms of the widely-used *stack-heap model*, which already appears in the seminal papers of Ishtiaq and O'Hearn [2001] and Reynolds [2002]. A stack-heap pair $\langle \mathfrak{s}, \mathfrak{h} \rangle$ consists of a stack $\mathfrak{s}$ assigning *values* in **Val** to variables and a heap $\mathfrak{h}$ assigning values in **Val** to allocated memory *locations* taken from $\mathbf{Loc} \subseteq \mathbf{Val}$.

We fix the set of locations $\mathbf{Loc} \triangleq \mathbb{N}_{>0}$. To simplify the technical development, we will work with the sets of values $\mathbf{Val} = \mathbf{Loc}$ and $\mathbf{Val} = \mathbf{Loc} \cup \{nil\}$, where the latter extends the former with the null pointer *nil*.

We will mostly work with $\mathbf{Val} = \mathbf{Loc}$ (and generally prefer the term locations over values), which simplifies the presentation of our decision procedure. However, we will use $\mathbf{Val} \triangleq \mathbf{Loc} \cup \{nil\}$ in examples and in the

undecidability proofs (in these examples and proofs one could also replace *nil* by a fresh variable that refers to a location which is never allocated; hence, the use of *nil* in these places has been a decision of taste and presentational convenience). We later present a reduction that extends our decision procedure to the extended set of values that includes the null pointer (see Cor. 9.4). We note, however, that it would not be difficult to extend our decision procedure to directly deal with $\mathbf{Val} \triangleq \mathbf{Loc} \cup \{nil\}$. We summarize that everywhere, where *nil* is not explicitly mentioned, the reader should identify values with locations, that is, assume $\mathbf{Val} = \mathbf{Loc} \triangleq \mathbb{N}_{>0}$; in such cases, we will prefer the term locations over values. The set **Stacks** of *stacks* then consists of all finite partial functions mapping variables to locations, i.e.,

$$\mathbf{Stacks} \triangleq \{\mathfrak{s} \mid \mathfrak{s} \colon V \rightharpoonup \mathbf{Val},\ V \subseteq \mathbf{Var},\ |V| < \infty, \}\,,$$

In order to treat both evaluations of variables and constant locations uniformly, we slightly abuse notation and set $\mathfrak{s}(v) \triangleq v$ for all values $v \in \mathbf{Val}$.[2] The set **Heaps** of *heaps* consists of all finite partial functions mapping allocated memory locations to sequences of locations, i.e.,

$$\mathbf{Heaps} \triangleq \left\{\mathfrak{h} \mid \mathfrak{h} \colon L \rightharpoonup \mathbf{Val}^+,\ L \subseteq \mathbf{Loc},\ |L| < \infty\right\}.$$

By mapping locations to sequences rather than single locations, the heap assigns every allocated memory location to the entire structure allocated at this location. This is a fairly standard—but far from ubiquitous [Calcagno et al. 2006; Reynolds 2002]—abstraction of the actual memory layout; it simplifies the memory model without losing precision as long as we do not use pointer arithmetic.

We frequently refer to stack-heap pairs $\langle \mathfrak{s}, \mathfrak{h} \rangle$ as (program) *states*.

3.2.1 Location terminology. We denote by $\mathsf{locs}(\phi)$ the set of all locations in **Loc** that explicitly appear as constants symbols in **SL** formula ϕ. Similarly, the set of all locations appearing in heap $\mathfrak{h}$ is $\mathsf{locs}(\mathfrak{h}) \triangleq \mathsf{dom}(\mathfrak{h}) \cup \bigcup_{\mathbf{v} \in \mathsf{img}(\mathfrak{h})} \mathbf{v}$; we lift this set to states $\langle \mathfrak{s}, \mathfrak{h} \rangle$ by setting $\mathsf{locs}(\langle \mathfrak{s}, \mathfrak{h} \rangle) \triangleq \mathsf{img}(\mathfrak{s}) \cup \mathsf{locs}(\mathfrak{h})$.

We often distinguish between allocated, referenced, and dangling locations $\ell \in \mathbf{Loc}$: ℓ is *allocated* in heap $\mathfrak{h}$ if $\ell \in \mathsf{dom}(\mathfrak{h})$; it is *referenced* if $\ell \in \mathsf{img}(\mathfrak{h})$. Finally, ℓ is *dangling* if it appears in $\mathfrak{h}$ but is not allocated.

We call a variable $x \in \mathbf{Var}$ allocated, referenced, or dangling if the location $\mathfrak{s}(x)$ is allocated, referenced, or dangling, respectively. We collect all allocated variables and all referenced variables in state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ in the sets $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) \triangleq \{x \mid \mathfrak{s}(x) \in \mathsf{dom}(\mathfrak{h})\}$ and $\mathsf{refed}(\mathfrak{s}, \mathfrak{h}) \triangleq \{x \mid \mathfrak{s}(x) \in \mathsf{img}(\mathfrak{h})\}$.

3.3 The Semantics of Separation Logic

Figure 3 defines the semantics of **SL** in terms of a satisfaction relation $\models_\Phi$, where the sole purpose of Φ—explained in Section 3.3.1 below—is to assign semantics to user-defined predicate calls. A state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ that satisfies an **SL** formula ϕ, i.e., $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$, is called a *model* of ϕ.

The empty-heap predicate **emp** holds iff the heap is empty; equalities and disequalities between variables hold iff the stack maps the variables to identical and different locations, respectively. For (dis-)equalities, we additionally require that the heap is empty. This is non-standard, but not unprecedented [Piskac et al. 2013], and will simplify the technical development.

A points-to assertion $x \mapsto \langle y_1, \ldots, y_k \rangle$ holds in the singleton heap that allocates exactly the location $\mathfrak{s}(x)$ and stores the locations $\mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k)$ at this location. This interpretation of points-to assertions is often called a *precise* [Calcagno et al. 2007; Yang 2001] semantics, because the heap contains precisely the object described by the points-to assertion, and nothing else.

For the separating conjunction, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models \psi \star \theta$ holds if and only if there exist domain-disjoint heaps $\mathfrak{h}_1$, $\mathfrak{h}_2$ such that their union ($\uplus$, see Section 2) is $\mathfrak{h}$ and both $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \psi$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \theta$ hold. While the separating

[2]This convention does not affect the formal definition of stacks; in particular, their domain and image remains unchanged.

ϕ	$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$ iff
emp	$\mathfrak{h} = \emptyset$
$u \approx v$	$\mathfrak{s}(u) = \mathfrak{s}(v)$ and $\mathfrak{h} = \emptyset$
$u \not\approx v$	$\mathfrak{s}(u) \neq \mathfrak{s}(v)$ and $\mathfrak{h} = \emptyset$
$u \mapsto \mathbf{y}$	$\mathfrak{h} = \{\mathfrak{s}(u) \mapsto \mathfrak{s}(\mathbf{y})\}$
$\mathrm{pred}(\mathbf{y})$	$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi(\mathbf{y})$ for some $(\mathrm{pred}(\mathbf{x}) \Leftarrow \psi) \in \Phi$
$\psi \star \theta$	exists $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \psi$, and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \theta$
$\psi \mathrel{-\!\!\star} \theta$	for all $\mathfrak{h}_0$, if $\mathrm{dom}(\mathfrak{h}_0) \cap \mathrm{dom}(\mathfrak{h}) = \emptyset$ and $\langle \mathfrak{s}, \mathfrak{h}_0 \rangle \models_\Phi \psi$ then $\langle \mathfrak{s}, \mathfrak{h}_0 \uplus \mathfrak{h} \rangle \models_\Phi \theta$
$\psi \wedge \theta$	$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi$ and $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \theta$
$\psi \vee \theta$	$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi$ or $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \theta$
$\neg\psi$	not $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi$
$\exists x.\, \psi$	exists $v \in \mathbf{Loc}$ such that $\langle \mathfrak{s}[x/v], \mathfrak{h} \rangle \models_\Phi \psi$
$\forall x.\, \psi$	for all $v \in \mathbf{Loc}$, $\langle \mathfrak{s}[x/v], \mathfrak{h} \rangle \models_\Phi \psi$

Fig. 3. Semantics of **SL**

conjunction is about splitting the heap, the magic wand is about extending it: $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi \mathrel{-\!\!\star} \psi$ holds iff all ways to extend $\mathfrak{h}$ with a disjoint model of ϕ yields a model of ψ.

The semantics of the Boolean connectives and the quantifiers is standard. In particular, as justified by the lemma below, the semantics of quantifiers can also be interpreted in terms of syntactic substitution rather than updating the stack (which is formally defined in Section 2).

Lemma 3.1 (Substitution Lemma). *For all* **SL** *formulas* ϕ*, states* $\langle \mathfrak{s}, \mathfrak{h} \rangle$*, variables* $x \in \mathrm{fvars}(\phi)$*, and locations* $\ell \in \mathbf{Loc}$*, we have* $(\mathfrak{s}[x/v], \mathfrak{h}) \models_\Phi \phi$ *iff* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi[x/v]$.

Proof. By induction on the structure of **SL** formulas. □

Example 3.2. (1) $(x \mapsto y) \star (y \mapsto nil)$ *states that the heap consists of exactly two objects, one pointed to by* x*, the other pointed to by* y*; that the object pointed to by* x *contains a pointer to the object pointed to by* y*; and that the object pointed to by* y *contains a null pointer. Put less precisely but more concisely,* x *points to* y*,* y *points to nil, and* x *and* y *are separate objects on the heap. The precise semantics of assertions guarantees that there are no other objects in the heap.*

(2) $(x \mapsto y) \wedge (z \mapsto y)$ *states that (a) the heap consists of a single object* x *that points to* y *and that simultaneously (b) the heap consists of a single object* z *that points to* y*. This formula is only satisfiable for stacks* $\mathfrak{s}$ *with* $\mathfrak{s}(x) = \mathfrak{s}(z)$.

(3) $(x \mapsto y) \mathrel{-\!\!\star} (z \mapsto y)$ *states that after adding a pointer from* x *to* y *to the heap, we obtain a heap that contains a single pointer from* z *to* y*. This formula is only satisfiable for the empty heap and for stacks* $\mathfrak{s}$ *with* $\mathfrak{s}(x) = \mathfrak{s}(z)$.

(4) $\forall x.\, (x \mapsto nil) \mathrel{-\!\!\star} ((\neg\mathbf{emp}) \star (\neg\mathbf{emp}))$ *states that the heap contains at least one pointer: no matter which variable we additionally allocate, the resulting heap can be split into two nonempty parts, so the original heap must itself have been nonempty—the formula is equivalent to* $\neg\mathbf{emp}$.

3.3.1 Systems of inductive definitions. Predicates are interpreted in terms of a user-supplied *system of inductive definitions* (SID). An SID is a finite set Φ of *rules* of the form $\mathrm{pred}(\mathbf{x}) \Leftarrow \phi(\mathbf{x})$, where $\mathrm{pred} \in \mathbf{Preds}$ is a predicate symbol, $\mathrm{fvars}(\mathrm{pred}) = \mathbf{x} \in \mathbf{Var}^*$ are the formal *parameters* of pred with $|\mathbf{x}| = \mathrm{ar}(\mathrm{pred})$, and ϕ is an **SL** formula with free variables $\mathbf{x}$; the *size* $|\Phi|$ of Φ is the sum of the sizes of the formulas in its rules. We collect all predicates that occur in Φ in the set $\mathbf{Preds}(\Phi)$. Moreover, we assume that all rules with the same predicate pred on the left-hand side have the same (repetition-free) sequence of parameters $\mathbf{x}$.

A stack-heap pair $\langle \mathfrak{s}, \mathfrak{h} \rangle$ satisfies the predicate call $\mathsf{pred}(\mathbf{y})$ with respect to SID Φ iff Φ contains a rule $\mathsf{pred}(\mathbf{x}) \Leftarrow \phi$ such that $\langle \mathfrak{s}, \mathfrak{h} \rangle$ satisfies the rule's right-hand side once we instantiate its formal parameters with the arguments passed to the predicate call, i.e., $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi(\mathbf{y})$.

Notice that rules involving arbitrary **SL** formulas—e.g., $\mathsf{pred}(x) \Leftarrow \neg\mathsf{pred}(x)$—do *not* necessarily lead to a well-defined semantics of predicate calls. We will restrict the formulas allowed to appear in SIDs in Section 3.4.2 to ensure that our semantics is always well-defined.

Example 3.3 (Inductive Definitions). (1) *Let Φ_{ls} be the SID given by the following rules:*

$$\begin{array}{lcl@{\qquad}lcl} \mathsf{lseg}(x_1, x_2) & \Leftarrow & x_1 \mapsto x_2 & \mathsf{ls}(x_1) & \Leftarrow & x_1 \mapsto \mathit{nil} \\ \mathsf{lseg}(x_1, x_2) & \Leftarrow & \exists y.\, x_1 \mapsto y \star \mathsf{lseg}(y, x_2) & \mathsf{ls}(x_1) & \Leftarrow & \exists y.\, (x_1 \mapsto y) \star \mathsf{ls}(y) \end{array}$$

The predicate $\mathsf{lseg}(x_1, x_2)$ describes non-empty singly-linked list segments with head x_1 and tail x_2; the predicate $\mathsf{ls}(x_1)$ describes those list segments that are terminated by a null pointer. Hence, the formulas $\mathsf{lseg}(x_1, \mathit{nil})$ and $\mathsf{ls}(x_1)$ are equivalent with respect to the SID Φ_{ls}.

(2) *The SID $\Phi_{\mathsf{odd/even}}$ below defines all non-empty list segments of odd and even length, respectively.*

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

(3) *The SID Φ_{tree} below defines null-terminated binary trees with root x_1.*

$$\mathsf{tree}(x_1) \Leftarrow x_1 \mapsto \langle \mathit{nil}, \mathit{nil} \rangle \qquad \mathsf{tree}(x_1) \Leftarrow \exists \langle l, r \rangle .\, (x_1 \mapsto \langle l, r \rangle) \star \mathsf{tree}(l) \star \mathsf{tree}(r)$$

3.3.2 Satisfiability and entailment. An **SL** formula ϕ is *satisfiable* with respect to Φ iff there exists a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ such that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$. Moreover, the **SL** formula ϕ *entails* the **SL** formula ψ given SID Φ, written $\phi \models_\Phi \psi$, iff for all states $\langle \mathfrak{s}, \mathfrak{h} \rangle$, we have $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$ implies $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi$.

3.3.3 Isomorphic states. Our decision procedure will exploit that **SL** formulas cannot distinguish between individual locations—as long as formulas do not explicitly use constant locations. More formally, formulas cannot distinguish isomorphic states:

Definition 3.4 (Isomorphic States). *Two states $\langle \mathfrak{s}, \mathfrak{h} \rangle$ and $\langle \mathfrak{s}', \mathfrak{h}' \rangle$ are* isomorphic*, written $\langle \mathfrak{s}, \mathfrak{h} \rangle \cong \langle \mathfrak{s}', \mathfrak{h}' \rangle$, iff there exists a bijection $\sigma \colon \mathsf{locs}(\langle \mathfrak{s}, \mathfrak{h} \rangle) \to \mathsf{locs}(\langle \mathfrak{s}', \mathfrak{h}' \rangle)$ 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 3.5. *Let ϕ be an **SL** formula with $\mathsf{locs}(\phi) = \emptyset$. Then, for all states $\langle \mathfrak{s}, \mathfrak{h} \rangle$ and $\langle \mathfrak{s}', \mathfrak{h}' \rangle$,*

$$\langle \mathfrak{s}, \mathfrak{h} \rangle \cong \langle \mathfrak{s}', \mathfrak{h}' \rangle \quad \textit{implies} \quad \langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi \textit{ iff } \langle \mathfrak{s}', \mathfrak{h}' \rangle \models_\Phi \phi.$$

Proof. By induction on the structure of **SL** formulas. □

3.4 The Bounded Treewidth Fragment

Our main goal is to develop a decision procedure for entailments in an **SL** fragment that extends the so-called bounded treewidth fragment ($\mathbf{SL}_{\mathrm{btw}}$) of Iosif et al. [2013]. In this section, we briefly recapitulate that fragment as we will rely on similar restrictions for SIDs.

3.4.1 Symbolic heaps. Formulas in $\mathbf{SL}_{\mathrm{btw}}$ are restricted to symbolic heaps with user-supplied predicates—a popular fragment of **SL** that is both expressive enough for specifying complex heap shapes and suitable to serve

as an abstract domain for program analyses (cf. [Berdine et al. 2007, 2005b; Calcagno et al. 2011]). A *symbolic heap* is a formula of the form

$$\exists \underbrace{x_1, \ldots, x_k}_{k \geq 0}. \underbrace{\phi_{\text{atom}} \star \cdots \star \phi_{\text{atom}}}_{\text{1 or more atoms}}.$$

Notice that negation, disjunction, universal quantifiers, and magic wands are *not* allowed in symbolic heaps. In particular, this means—since pure formulas are evaluated in the empty heap—that there is *no* symbolic heap that is always satisfied, i.e., equivalent to true.

When working with symbolic heaps, it is convenient to group the atoms into (1) a spatial part collecting all points-to assertions, (2) a part collecting all predicate calls, and (3) a pure part collecting all equalities and disequalities (in that order). Hence, the set $\mathbf{SH}^{\exists}$ of *symbolic heaps* ϕ_{sh} is given by

$$\phi_{\text{sh}} ::= \underbrace{\exists \mathbf{e}.}_{\mathbf{e} \in \mathbf{Var}^*} \underbrace{(u_1 \mapsto \mathbf{v}_1) \star \cdots \star (u_k \mapsto \mathbf{v}_k)}_{\text{spatial part, } \mathbf{emp} \text{ for } k=0} \star \underbrace{\text{pred}_1(\mathbf{w}_1) \star \cdots \star \text{pred}_l(\mathbf{w}_l)}_{\text{predicate calls, } \mathbf{emp} \text{ for } l=0}$$
$$\star \underbrace{u_1 \approx v_1 \star \cdots \star u_m \approx v_m \star u'_1 \not\approx v'_1 \star \cdots \star u'_n \not\approx v'_n}_{\text{pure part, } \mathbf{emp} \text{ for } m=0, n=0 \text{ respectively}}.$$

3.4.2 Symbolic heap SIDs. Our semantics of predicate calls (see Figure 3) is well-defined as long as all formulas appearing in the underlying SID are symbolic heaps—a requirement that we impose throughout the remainder of this article. For instance, all SIDs in Example 3.3 only use symbolic heaps in their rules. The restriction of SID rules to symbolic heaps is standard. In fact, our semantics coincides with other semantics from the separation logic literature that—instead of replacing predicates by rules step by step—are based on least fixed points [Brotherston 2007; Brotherston et al. 2014] or derivation trees [Iosif et al. 2013, 2014; Jansen et al. 2017; Matheja 2020].

3.4.3 The bounded treewidth fragment. Since negation is not available, the entailment problem for symbolic heaps is genuinely different from the satisfiability problem: it is impossible to solve an entailment $\phi \models_{\Phi} \psi$ by checking the unsatisfiability of $\phi \wedge \neg\psi$, because the latter formula is not a symbolic heap. In fact, the satisfiability problem for symbolic heaps is decidable in general [Brotherston et al. 2014], whereas the entailment problem is not [Antonopoulos et al. 2014; Iosif et al. 2014]. However, various subclasses of symbolic heaps with a decidable—and even tractable [Cook et al. 2011]—entailment problem have been studied in the literature (e.g., [Berdine et al. 2004; Iosif et al. 2013, 2014; Le et al. 2017]); as such, the symbolic heap fragment has been the main focus of a recent competition of entailment solvers (SL-COMP) [et al. 2019]. The largest of these fragments has been developed by Iosif et al. [2013]; it achieves decidability by imposing three restrictions—progress, connectivity, and establishment—on SIDs to ensure that all models of predicates are of *bounded treewidth*.[3]

Local allocation and references. To formalize the above three assumptions for SIDs, we need two auxiliary definitions: we collect all variables and locations that appear on the left-hand side of points-to assertions in formula ϕ in the *local allocation* set $\mathsf{lalloc}(\phi)$; analogously, the *local references* set $\mathsf{lref}(\phi)$ collects all variables and locations appearing on the right-hand side of points-to assertions.

We now present the three aforementioned conditions imposed on SIDs Φ to ensure decidability of the entailment problem for symbolic heaps.

Progress. A predicate pred satisfies *progress* iff there exists a free variable $x \in \mathsf{fvars}(\text{pred})$ such that, for all rules $(\text{pred}(\mathbf{x}) \Leftarrow \phi) \in \Phi$, (1) ϕ contains exactly one point-to assertion, and (2) x is allocated in ϕ, i.e., $\mathsf{lalloc}(\phi) = \{x\}$.

[3] More precisely, when viewed as graphs, all models of the SIDs satisfying the three aforementioned restrictions have bounded treewidth. For a formal definition of treewidth, we refer to [Diestel 2016].

In this case, we call x the *root* of pred. Moreover, if the i-th parameter of pred, say x_i, is its root, then we set $\mathsf{predroot}(\mathsf{pred}(\mathbf{x})) \triangleq x_i$.

Connectivity. A predicate pred satisfies *connectivity* iff for all rules of pred, all variables that are allocated in the recursive calls of the rule are also referenced in the rule. Formally, for all rules $(\mathsf{pred} \Leftarrow \phi) \in \Phi$ and for all calls $\mathsf{pred}'(\mathbf{y})$ appearing in ϕ, we have $\mathsf{predroot}(\mathsf{pred}'(\mathbf{y})) \subseteq \mathsf{lref}(\phi)$.

Establishment. A predicate pred is *established* iff all existentially quantified variables across all rules of pred are eventually allocated, or equal to a parameter. Formally, for all rules $(\mathsf{pred}(\mathbf{x}) \Leftarrow \exists \mathbf{y}.\, \phi) \in \Phi$ and for all states $\langle \mathfrak{s}, \mathfrak{h} \rangle$, if $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$ then $\mathfrak{s}(\mathbf{y}) \subseteq \mathrm{dom}(\mathfrak{h}) \cup \mathfrak{s}(\mathbf{x})$.

3.4.4 SIDs of bounded treewidth. We denote by $\mathbf{ID}_{\mathrm{btw}}$ the set of all SIDs in which all predicates satisfy *progress*, *connectivity*, and *establishment*. E.g., the SIDs given in Example 3.3 belong to $\mathbf{ID}_{\mathrm{btw}}$.

Theorem 3.6 ([Echenim et al. 2020b; Iosif et al. 2013]). *The entailment problem for symbolic heaps over SIDs in* $\mathbf{ID}_{\mathrm{btw}}$ *is decidable, of elementary complexity, and 2-EXPTIME hard.*

In the remainder of this article, we strengthen the above theorem in two ways: First, we give a larger decidable **SL** fragment, and, second, we develop a 2-EXPTIME decision procedure which, by the above lower bounds, is of optimal asymptotic complexity. Moreover, we show that even small extensions of our fragments lead to an undecidable entailment problem.

3.4.5 Global Assumptions about SIDs. Unless stated otherwise, we assume that all SIDs considered in this article belong to $\mathbf{ID}_{\mathrm{btw}}$. Moreover, to avoid notational clutter, Φ always refers to an arbitrary, but fixed SID in $\mathbf{ID}_{\mathrm{btw}}$ unless it is explicitly given. Without loss of generality, we make two further assumptions about the rules in SIDs to simplify the technical development.

First, we assume that non-recursive rules do *not* contain existential quantifiers because they can always be eliminated: due to progress and establishment, all existentially-quantified variables in a non-recursive rule must be provably equal to either a constant or a parameter of the predicate.

Second, to avoid dedicated reasoning about points-to assertions, we may add dedicated predicates simulating points-to assertions to every SID; we call the resulting SIDs *pointer-closed*:

Definition 3.7 (Pointer-closed SID). *An SID* Φ *is* pointer-closed *w.r.t.* ϕ *iff it contains a predicate* ptr_k *and a single rule* $\mathrm{ptr}_k(\langle x_1, \ldots, x_{k+1} \rangle) \Leftarrow x_1 \mapsto \langle x_2, \ldots, x_{k+1} \rangle$ *for all points-to assertions mapping to structures of length* k *in* ϕ.

Since all predicates introduced by transforming an SID into a pointer-closed one satisfy progress, connectivity, and establishment, we can safely assume that SIDs in $\mathbf{ID}_{\mathrm{btw}}$ are pointer-closed. As a consequence of this assumption, we consider the number of formal parameters of predicates to be at least as large as the number of fields of points-to assertions whenever we analyze complexities.

4 THE GUARDED FRAGMENT OF SEPARATION LOGIC

To obtain fragments of **SL** with both support for complex data structure predicates and a decidable entailment problem, we rely on the same restrictions on user-defined predicates as Iosif et al. [2013]: the semantics of predicate calls needs to be determined by SIDs taken from the bounded treewidth fragment $\mathbf{ID}_{\mathrm{btw}}$. In contrast to Iosif et al. [2013], our work is, however, not limited to entailments between symbolic heaps over the predicates at hand. Rather, we additionally consider reasoning about a novel (quantifier-free) *guarded* fragment of separation logic (**GSL**) featuring restricted variants of negation $\neg$, magic wand $\mathrel{-\!\!\star}$, and septraction $\circledast$.

Intuitively, the guarded fragment enforces that the aforementioned connectives $\neg$, $\mathrel{-\!\!\star}$, and $\circledast$ only appear in conjunction with another formula restricting the possible shapes of the heap. We will show in Section 5 that this

restriction is crucial: lifting it for any of these connectives yields an undecidable entailment problem—even if the remaining connectives are removed.

4.1 Guarded Formulas

The set **GSL** of formulas in (quantifier-free) *guarded separation logic* is given by the grammar

$$\begin{aligned} \phi ::= \phi_{\text{atom}} \quad & (::= \ \mathbf{emp} \mid u \approx v \mid u \not\approx v \mid u \mapsto \mathbf{w} \mid \mathsf{pred}(\mathbf{w})) && \text{(same atoms as SL)} \\ & \mid \phi \star \phi \mid \phi \wedge \phi \mid \phi \vee \phi && \text{(standard connectives)} \\ & \mid \phi \wedge \neg\phi \mid \phi \wedge (\phi \mathrel{-\!\!\star} \phi) \mid \phi \wedge (\phi \circledast \phi). && \text{(\textit{guarded} connectives)} \end{aligned}$$

The atoms as well as the connectives $\star$, $\wedge$, and $\vee$ are the same as for the full logic **SL** introduced in Section 3.1. Moreover, negation $\neg$, magic wand $\mathrel{-\!\!\star}$, and septraction $\circledast$ may only appear in *guarded* form, i.e., in conjunction with another guarded formula ϕ. Since **GSL** is a syntactic fragment of **SL**, the semantics of **GSL** is given by the semantics of **SL** presented in Section 3.3.

Example 4.1. *Assume the predicate* $\mathsf{lseg}(x_1, x_2)$ *represents all non-empty list segments from* x_1 *to* x_2*; a formal definition is found in Example 3.3. Moreover, consider the following guarded formulas:*

(1) $\mathsf{lseg}(x, y) \wedge \neg x \mapsto y$ *states that the heap consists of a list of length at least two.*
(2) $\mathsf{lseg}(x, y) \wedge (\mathsf{lseg}(y, z) \circledast \mathsf{lseg}(x, x))$ *states that the heap consists of a list segment from* x *to* y *that can be extended to a cyclic list by adding a list from* y *to* z*; it entails that* x *and* z *are aliases.*

In contrast to variants of separation logic in the literature (cf. [Calcagno et al. 2011; Reynolds 2002]), our separation logic **SL** does not contain an atom true, which is always satisfied. While true is, of course, definable in **SL**, e.g., $\mathbf{emp} \vee \neg\mathbf{emp}$, it is *not* definable in **GSL**. In particular, $x \approx x$ is not equivalent to true as our semantics of equalities and disequalities requires the heap to be empty. This is crucial: If true were definable, **GSL** would coincide with the set of all quantifier-free **SL** formulas, because we could choose true for all guards.

4.2 Guarded States and Dangling Pointers

The decision procedure developed in this article exploits that all models of guarded formulas are themselves guarded in the sense that they have only a limited amount of dangling pointers. We recall from Section 3.2.1 that a dangling pointer is a location that is not allocated; the set of all dangling locations in heap $\mathfrak{h}$ is thus given by $\mathsf{dangling}(\mathfrak{h}) \triangleq \mathsf{locs}(\mathfrak{h}) \setminus \mathsf{dom}(\mathfrak{h})$.

In the following, we first define guarded states, then show that establishment implies a models of atomic predicates to be guarded, and finally lift this result to arbitrary guarded formulas.

Definition 4.2 (Guarded State). *The set* **GStates** *of* guarded states *is given by*

$$\mathbf{GStates} \ \triangleq \ \{\langle \mathfrak{s}, \mathfrak{h} \rangle \mid \mathfrak{s} \in \mathbf{Stacks}, \mathfrak{h} \in \mathbf{Heaps}, \mathsf{dangling}(\mathfrak{h}) \subseteq \mathsf{img}(\mathfrak{s})\} \ .$$

Guarded states are well-behaved with regard to taking the union of heaps:

Lemma 4.3. *Let* $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle \in \mathbf{GStates}$ *with* $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \neq \bot$*. Then,* $\langle \mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2 \rangle \in \mathbf{GStates}$.

Proof. We observe that $\mathsf{dangling}(\mathfrak{h}_1 \uplus \mathfrak{h}_2) \subseteq \mathsf{dangling}(\mathfrak{h}_1) \cup \mathsf{dangling}(\mathfrak{h}_2) \subseteq \mathsf{img}(\mathfrak{s})$ □

Furthermore, due to establishment (cf. Section 3.4.5), models of predicate calls are guarded:

Lemma 4.4. *For all predicates* $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ *and all states* $\langle \mathfrak{s}, \mathfrak{h} \rangle$*, we have*

$$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(\mathbf{x}) \quad \textit{implies} \quad \langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}.$$

Proof. By induction on the number of rule applications needed to establish $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(\mathbf{x})$; a detailed proof is found in Appendix A.2. □

We now lift the result from Lemma 4.4 from atomic predicates to arbitrary guarded formulas. We will use the following result that every model of a guarded formula satisfies a finite number of predicates conjoined by the separating conjunction:

Lemma 4.5. *Let $\phi \in$ **GSL** be a guarded formula with $\mathsf{fvars}(\phi) = \mathbf{x}$. Then, for every state $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$, there are predicates $\mathsf{pred}_i \in \mathbf{Preds}(\Phi)$ and variables $\mathbf{z_i} \subseteq \mathbf{x}$ such that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \bigstar_{1 \le i \le k} \mathsf{pred}_i(\mathbf{z_i})$.*

Proof. By structural induction on ϕ; see Appendix A.4 for details. □

Corollary 4.6. *For all $\phi \in$ **GSL** and all states $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$, we have $\langle \mathfrak{s}, \mathfrak{h} \rangle \in$ **GStates**.*

Proof. Immediate from Lemma 4.4 and Lemma 4.5. □

On the importance of guardedness. The fact that all appearances of negation, magic wand, and septraction in **GSL** are guarded by a conjunction with another **GSL** formula is crucial for limiting the number of dangling pointers in the above lemma.

For the negation and the magic wand this is straightforward: without guards, both can be used to define true, e.g., $\mathbf{emp} \vee \neg\mathbf{emp}$ and $((x \mapsto nil) \star (x \mapsto nil)) \mathrel{-\!\!\star} \mathbf{emp}$). Since true is satisfied by all states, the number of dangling locations is unbounded. For the septraction $\circledast$, consider the following SID:

$$\mathsf{tll}(r, l, t) \Leftarrow l \mapsto t \star r \approx l \qquad \mathsf{lseg}(l, t) \Leftarrow \exists n.\ (l \mapsto n) \star \mathsf{lseg}(n, t)$$
$$\mathsf{tll}(r, l, t) \Leftarrow \exists \langle u, v, m \rangle .\ (r \mapsto \langle u, v \rangle) \star \mathsf{tll}(u, l, m) \star \mathsf{tll}(s_2, m, r) \qquad \mathsf{lseg}(l, t) \Leftarrow l \mapsto t$$

The tll predicate encodes a binary tree with root r and leftmost leaf l overlaid with a singly-linked list segment from l to t whose nodes are the leaves of the tree. Now, assume a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ satisfying the unguarded formula $\mathsf{lseg}(l, t) \circledast \mathsf{tll}(r, l, t)$. In other words, there exists a heap $\mathfrak{h}_1$ with $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathsf{lseg}(l, t)$ and $\langle \mathfrak{s}, \mathfrak{h} \cup \mathfrak{h}_1 \rangle \models_\Phi \mathsf{tll}(r, l, t)$. Since each list element is a leaf of the tree in heap $\mathfrak{h}$, we have $\mathsf{dangling}(\mathfrak{h}) = \mathsf{dom}(\mathfrak{h}_1)$—a finite, but unbounded set of dangling pointers.

5 BEYOND GUARDED SEPARATION LOGIC: UNDECIDABILITY PROOFS

Before we develop our decision procedure for the fragment **GSL** of guarded separation logic with inductive definitions of bounded treewidth, we further justify the need for guarding negation, magic wand, and septraction. More precisely, we show in this section that omitting the guards for *any* of the above three operators leads to an undecidable logic. Together with our decidability results presented afterwards, this yields an almost tight delineation between undecidability of separation logics that allow arbitrary SIDs in $\mathbf{ID}_{\mathrm{btw}}$.

5.1 Encoding Context-Free Language in SIDs

All of our undecidability results, which are presented in Section 5.2, rely on a novel encoding of the language-intersection problem for context-free grammars—a well-known undecidable problem.

Definition 5.1 (Context-free grammar). *A* context-free grammar *(CFG) in Chomsky normal form is a 4-tuple $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$, where $\mathbf{N}$ is a finite set of* nonterminals*; $\mathbf{T}$ is a finite set of terminals, which is disjoint from $\mathbf{N}$; $\mathbf{R} \subseteq \mathbf{N} \times (\mathbf{N}^2 \cup \mathbf{T})$ is a finite set of* production rules *mapping nonterminals to two nonterminals or a single terminal; and $\mathbf{S} \in \mathbf{N}$ is the* start symbol*. **CFG** is the set of all CFGs.*

We often denote production rules $\langle a, b \rangle$ by $a \to b$ to improve readability. Since we assume all CFGs to be in Chomsky normal form, all rules are either of the form $N \to AB$ or $N \to a$, where N, A, B are nonterminals in $\mathbf{N}$ and a is a terminal in $\mathbf{T}$.

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

Fig. 4. The SID Φ encoding derivations of the CFGs $\mathbf{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle$ and $\mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$.

Definition 5.2. *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle \in \mathbf{CFG}$ *and let* $v, w \in (\mathbf{N} \cup \mathbf{T})^*$. *We write* $v \Rightarrow w$ *if there exist strings* $u_1, u_2 \in (\mathbf{N} \cup \mathbf{T})^*$ *and a rule* $a \to b$ *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* $\mathbf{G}$ *is given by* $\mathcal{L}(\mathbf{G}) \triangleq \{w \in \mathbf{T}^* \mid \mathbf{S} \Rightarrow^+ w\}$.

In the following, we exploit the following classic undecidability result (cf. [Bar-Hillel et al. 1961]):

Theorem 5.3 (Undecidability of language intersection). *Given two CFGs* $\mathbf{G}_1$ *and* $\mathbf{G}_2$, *it is undecidable whether* $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$ *holds—even if neither* $\mathbf{G}_1$ *nor* $\mathbf{G}_2$ *accept the empty string* $\langle\rangle$.

Encoding CFGs as SIDs. Throughout the remainder of this section, we fix a set $\mathbf{T} = \{a_1, \ldots, a_n\}$ of terminals and two CFGs $\mathbf{G}_1$ and $\mathbf{G}_2$, where we assume that their sets of nonterminals do not overlap, i.e., $\mathbf{N}_1 \cap \mathbf{N}_2 = \emptyset$. Fig. 4 depicts the SID Φ encoding $\mathbf{G}_1$ and $\mathbf{G}_2$: For each terminal symbol a_i, we introduce a predicate $\mathsf{letter}_i(a)$.[4] Moreover, for each nonterminal $N \in \mathbf{N}_1 \cup \mathbf{N}_2$, there is a corresponding predicate encoding the derivations of $\mathbf{G}_1$ and $\mathbf{G}_2$ as trees with linked leaves (TLL), similar to the SID in Fig. 1 on p. 2. The predicate word overapproximates the possible *front*, i.e., the list of linked leaves of the TLL; we will need it later to prove undecidability of individual fragments.

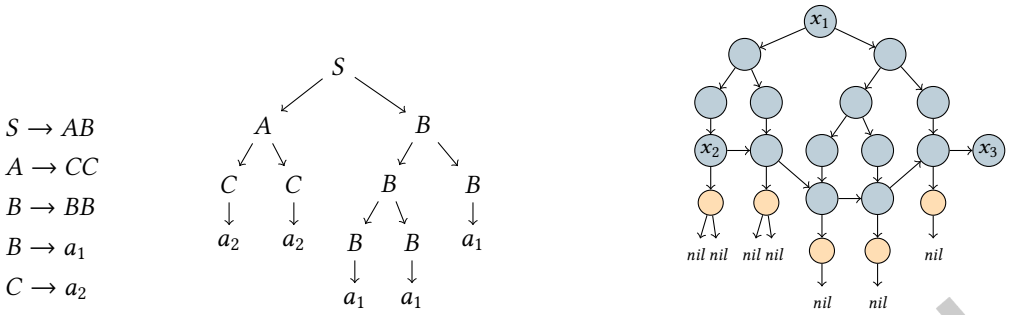
By construction, every word in $\mathcal{L}(\mathbf{G}_i)$ corresponds to at least one state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ with $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathbf{S}_i(x_1, x_2, x_3)$. Furthermore, every model $\langle \mathfrak{s}, \mathfrak{h} \rangle$ of $\mathbf{S}_i(x_1, x_2, x_3)$ corresponds to both a *derivation tree*[5] and a word in $\mathcal{L}(\mathbf{G}_i)$, where the inner nodes of the TLL correspond to the derivation tree and its front corresponds to the word in $\mathcal{L}(\mathbf{G}_i)$.

Example 5.4. *Fig. 5 illustrates both a derivation tree (Figure 5b) and a model of our encoding (Fig. 5c) for the CFG* $\mathbf{G} = \langle \{S, A, B, C\}, \{a_1, a_2\}, \mathbf{R}, S \rangle$ *whose rules are provided in Figure 5a. We observe that the depicted model encodes the aforementioned derivation tree: Every nonterminal is translated to a node in a 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 to represent terminal* a_k. *The list of linked leaves and orange nodes together form the* induced word, *i.e.,* $a_2 a_2 a_1 a_1 a_1 a_1$.

To show that our encoding is correct, i.e., it adequately captures the language of a given CFG, we need to refer to the word induced by a given model. We first define the terminals of such a word, which are given by the letter predicates in the models' list of linked leaves.

[4]While it is convenient to model each terminal a_i through a single points-to assertion mapping a to i null pointers, it is noteworthy, that, in principle, points-to assertions mapping to at most two locations suffice.

[5]We do not formally define derivation trees for CFGs as they are not required for the formal development; we refer to [Hopcroft et al. 2007] for a thorough introduction of CFGs.



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

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

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

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

Definition 5.5 (Induced letters). *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be the corresponding SID encoding. Let* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{word}(x, y)$ *and let* $j_1, \ldots, j_m \in \{1, \ldots, n\}$ *be such that*

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

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

Every model of the predicate $N(x_1, x_2, x_3)$ contains a sub-heap satisfying the word predicate:

Lemma 5.6. *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be its SID encoding. Let* $x_1, x_2, x_3 \in \mathbf{Var}$, $N \in \mathbf{N}$ *and let* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi N(x_1, x_2, x_3) \star \mathsf{t}$. *Then there exists a unique heap* $\mathfrak{h}_w \subseteq \mathfrak{h}$ *with* $\langle \mathfrak{s}, \mathfrak{h}_w \rangle \models_\Phi \mathsf{word}(x_2, x_3)$.

Proof. A straightforward induction shows that the models of the predicate call $N(x_1, x_2, x_3)$ are trees with linked leaves with root $\mathfrak{s}(x_1)$, leftmost leaf $\mathfrak{s}(x_2)$, and successor of rightmost leaf $\mathfrak{s}(x_3)$. We pick as $\mathfrak{h}_w$ the heap that contains $\mathfrak{s}(x_2)$ as well as all locations reachable from $\mathfrak{s}(x_2)$ in $\mathfrak{h}$. This gives us precisely the list from $\mathfrak{s}(x_2)$ to $\mathfrak{s}(x_3)$. Moreover, every leaf satisfies a formula of the form $\exists a.\ (y \mapsto \langle z, a \rangle) \star \mathsf{letter}_k(a)$. Consequently, $\langle \mathfrak{s}, \mathfrak{h}_w \rangle \models_\Phi \mathsf{word}(x_2, x_3)$. □

Lemma 5.6 ensures that models of our encoding of CFGs induce a word over the given alphabet.

Definition 5.7 (Induced word). *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be its SID encoding. Let* $x_1, x_2, x_3 \in \mathbf{Var}$, $N \in \mathbf{N}$ *and let* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi N(x_1, x_2, x_3)$. *Let* $\mathfrak{h}_w \subseteq \mathfrak{h}$ *be the unique heap with* $\langle \mathfrak{s}, \mathfrak{h}_w \rangle \models_\Phi \mathsf{word}(x_2, x_3)$. *The* induced word *of* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *and* N *is* $\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) \triangleq \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_w, x_2, x_3)$.

Every word $w \in \mathcal{L}(\mathbf{G})$ is then the induced word of a model of the corresponding SID encoding.

Lemma 5.8 (Completeness of the encoding). *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be the corresponding SID encoding. Let* $x_1, x_2, x_3 \in \mathbf{Var}$ *and let* $w \in \mathcal{L}(\mathbf{G})$. *Then there exists a model* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *of* $\mathsf{S}(x_1, x_2, x_3)$ *with* $\mathsf{wordof}_\mathsf{S}(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = w$.

Proof. By mathematical induction on the number of $\Rightarrow$ steps; see Appendix A.5. □

Likewise, every induced word of a model of the corresponding SID encoding is in $\mathcal{L}(\mathbf{G})$.

Lemma 5.9 (Soundness of the encoding). *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be its SID encoding. Let* $x_1, x_2, x_3 \in \mathbf{Var}$ *and let* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{S}(x_1, x_2, x_3)$. *Then* $\mathsf{wordof}_\mathsf{S}(\mathfrak{s}, \mathfrak{h}, x_2, x_3) \in \mathcal{L}(\mathbf{G})$.

Proof. By induction on the height h of the tree contained in $\mathfrak{h}$; see Appendix A.6. □

5.2 Undecidability of Unguarded Fragments

We are now ready to prove that omitting guards leads to undecidable SL fragments. To conveniently describe these fragments, we write $\mathbf{SL}_{\mathrm{btw}}(\cdot_1, \ldots, \cdot_k)$ for the restriction of quantifier-free formulas in $\mathbf{SL}_{\mathrm{btw}}$ to formulas built from all atoms as well as the additional symbols and connectives $\cdot_1, \ldots, \cdot_k$. For example, formulas in $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrm{t})$ are built from atomic predicates, the predicate t (true), and the binary connectives $\star$, $\wedge$. As usual, t holds in all models, i.e., $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathrm{t}$ for all states $\langle \mathfrak{s}, \mathfrak{h} \rangle$.

First, we show that allowing t as well as both standard and separating conjunction $\wedge$ and $\star$ (i.e., GSL without disjunction or any of the guarded connectives) immediately leads to undecidability.

Theorem 5.10. *The satisfiability problem for the fragment* $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrm{t})$ *is undecidable.*

Proof. Let Φ be the encoding of the CFGs $\mathbf{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle$ and $\mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$ as described in Section 5.1. Moreover, consider the $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrm{t})$ formula $\phi \triangleq (\mathbf{S}_1(a, x, y) \star \mathrm{t}) \wedge (\mathbf{S}_2(b, x, y) \star \mathrm{t})$. Then ϕ is satisfiable iff $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$; see Appendix A.7 for details. □

Corollary 5.11. *The satisfiability problem of* $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \neg)$ *is undecidable.*

Proof. This follows directly from the undecidability of $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrm{t})$ (Theorem 5.10), because t is definable in $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \neg)$; for example, $\mathrm{t} \triangleq \neg(\mathbf{emp} \wedge \neg\mathbf{emp})$. □

Corollary 5.12. *The satisfiability problem of* $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrel{-\!\!\star})$ *is undecidable.*

Proof. Follows directly from the undecidability of $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrm{t})$ (Theorem 5.10), because t is definable in $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrel{-\!\!\star})$; for example, $\mathrm{t} \triangleq (x \not\approx x) \mathrel{-\!\!\star} \mathbf{emp}$. □

Our final undecidability proof concerns unguarded septractions. We need one more auxiliary result before we can prove this result.

Lemma 5.13. *Let* $\mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$ *be the CFG fixed in Section 5.1. Moreover, let* Φ *be the corresponding SID encoding,* $\mathsf{word}_2(x, y) \triangleq (\mathsf{word}(x, y) \circledast \mathbf{S}_2(a, x, y)) \circledast \mathbf{S}_2(a, x, y)$*, and let* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *be a state. Then* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{word}_2(x, y)$ *iff* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{word}(x, y)$ *and* $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) \in \mathcal{L}(\mathbf{G}_2)$.

Proof. See Appendix A.8. □

To prove the undecidability of separation logic in the presence of unguarded septractions, we show that $\psi \triangleq \mathsf{word}_2(x, y) \circledast \mathbf{S}_1(a, x, y)$ is satisfiable iff $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$. Intuitively, this holds because ψ is satisfiable iff it is possible to replace the "word part" of models of $\mathbf{S}_1(a, x, y)$ with the "word part" of models of $\mathbf{S}_2(b, x, y)$.

Theorem 5.14. *The satisfiability problem of* $\mathbf{SL}_{\mathrm{btw}}(\circledast)$ *is undecidable.*

Proof. $\psi \triangleq \mathsf{word}_2(x, y) \circledast \mathbf{S}_1(a, x, y)$ is satisfiable iff $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$; see Appendix A.9. □

We have shown that all extensions of the guarded fragment **GSL**, in which one of the guards is dropped, lead to an undecidable satisfiability problem. In the remainder of this paper, we develop a decision procedure for **GSL**; keeping all guards thus indeed ensures decidability.

6 TOWARDS A COMPOSITIONAL ABSTRACTION FOR GSL

In Section 1, we sketched our goal of using a finite *compositional abstraction* that *refines* the satisfaction relation in order to decide the satisfiability problem for the separation logic fragment **GSL**. The same procedure then also allows deciding entailments between (quantifier-free) symbolic heaps in $\mathbf{SH}^{\exists}$ with user-defined predicates (defined by rules that may, of course, contain quantifiers). The key challenge is to develop an abstraction mechanism that can deal with arbitrary user-defined predicates from the $\mathbf{ID}_{\text{btw}}$ fragment. To get an abstraction that satisfies refinement, we need to be able to deduce from the abstraction which predicate calls hold in the underlying model. To this end, we will abstract every state by a set of formulas that relates the state to predicates of the SID.

In the following we introduce our abstraction, starting with a simple but insufficient idea, and then incrementally improve on it.

Purpose of this section. This section serves as a roadmap; it outlines the main concepts underlying our decision procedure and explains them informally by means of examples. We will formalize all of these concepts in follow-up sections—references to the formal details are provided where appropriate. Similarly, the remaining sections will frequently refer back to this section to either give further details on the examples, or to pin-point the progress of our formalization.

6.1 First Attempt: Abstracting States by Symbolic Heaps

Our first idea is to abstract a state by the quantifier-free symbolic heaps that it satisfies:

$$\text{abst}_1(\mathfrak{s}, \mathfrak{h}) \triangleq \left\{\phi \in \mathbf{SH}^{\exists} \mid \phi \text{ quantifier-free}, \langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \phi\right\}.$$

Let us analyze the properties of this abstraction function. (For the moment, we ignore whether we can actually compute this abstraction.)

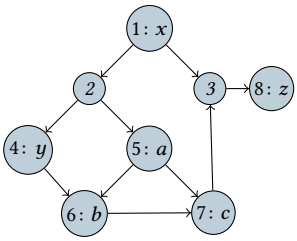
6.1.1 Finiteness. The abstraction abst_1 is finite (for a fixed number of variables, given by the domain of the stack $\mathfrak{s}$), because there are only finitely many quantifier-free symbolic heaps up to logical equivalence: Since $\Phi \in \mathbf{ID}_{\text{btw}}$, every predicate call in a symbolic heap ϕ has to allocate at least one free variable due to the *progress* property (cf. Section 3.4.3); the same holds for every points-to assertion. Consequently, every satisfiable quantifier-free formula can contain at most $|\text{fvars}(\phi)|$ many predicate calls and points-to assertions. In principle, we can, of course, "blow up" a satisfiable formula ϕ to arbitrary size by adding **emp** atoms and (dis-)equalities, but any fixed *aliasing constraint* over $\text{fvars}(\phi)$, i.e., any fixed relationship between the free variables of formula ϕ, can be expressed with at most $|\text{fvars}(\phi)|^2$) such atoms. Hence, we are able to obtain a bound on the size of the symbolic heaps that need to be considered for constructing the abstraction.

6.1.2 Refinement. Recall that our abstraction satisfies refinement iff states leading to the same abstraction satisfy the same formulas. This immediately holds for abst_1—at least on the quantifier-free symbolic-heap fragment of **GSL**.

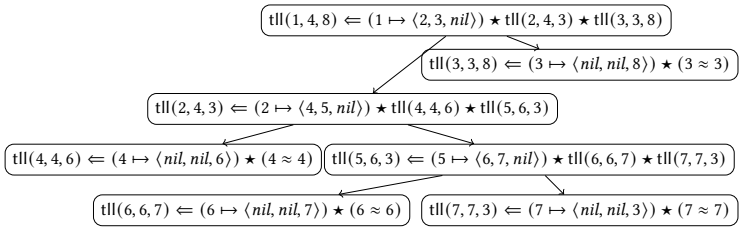
6.1.3 Compositionality. Can we also *compose* abstractions, i.e., can we find a (computable) operator $\bullet$ with $\text{abst}_1(\mathfrak{s}, \mathfrak{h}_1) \bullet \text{abst}_1(\mathfrak{s}, \mathfrak{h}_2) = \text{abst}_1(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$? Unfortunately, the example below demonstrates that finding such an operator is quite challenging. Assume that Φ defines the list-segment predicate lseg (cf. Example 3.3). Moreover, consider a state $(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$ such that

- $\mathfrak{s}(u) \neq \mathfrak{s}(v)$ for all $u, v \in \{x, y, z\}$ with $u \neq v$,
- $\text{abst}_1(\mathfrak{s}, \mathfrak{h}_1) = \{\text{lseg}(x, y)\}$, $\text{abst}_1(\mathfrak{s}, \mathfrak{h}_2) = \{\text{lseg}(y, z)\}$, and $\text{abst}_1(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \{\text{lseg}(x, z)\}$,

where we omit pure constraints in the sets $\text{abst}_1(\cdots)$ for readability. We highlight that it is a-priori unclear how to infer that $\text{lseg}(x, z)$ holds in the composed state, i.e., that $\text{abst}_1(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \{\text{lseg}(x, z)\}$—at least by relying solely on the assumptions $\text{abst}_1(\mathfrak{s}, \mathfrak{h}_1) = \{\text{lseg}(x, y)\}$ and $\text{abst}_1(\mathfrak{s}, \mathfrak{h}_2) = \{\text{lseg}(y, z)\}$. In particular, it is unclear



(a) A model $\langle \mathfrak{s}, \mathfrak{h} \rangle$ of the predicate $\mathsf{tll}(x, y, z)$.



(b) A corresponding Φ-tree $\mathfrak{t}$.

Fig. 6. A model $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{tll}(x, y, z)$ and the Φ-tree $\mathfrak{t}$ corresponding to this model.

how to derive this fact by a syntactic argument. One might turn towards an argument based on the semantics, i.e., considering the definition of lseg in Φ. However, then the above composition operation $\bullet$ boils down to an entailment check $\mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z) \models_\Phi \mathsf{lseg}(x, z)$. Hence, we end up with a chicken-and-egg problem: we need an entailment checker to implement the composition operator $\bullet$ that we would like to use in the implementation of our abstraction-based (satisfiability and) entailment checker.

6.2 Second Attempt: Unfolding Predicates into Forests

Next, we attempt to extend the abstraction abst_1 to get a "more syntactic" composition operation. To this end, we need to take a step back and reflect on the semantics of SIDs.

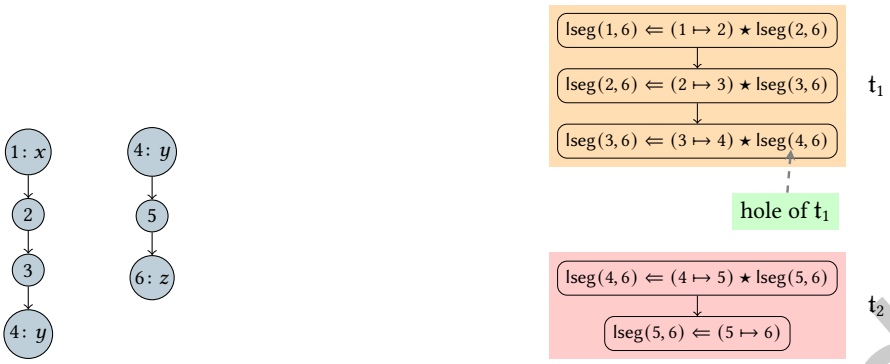
6.2.1 Unfolding predicate calls. According to the SL semantics (Section 3.3), $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(\mathbf{z})$ holds iff there exists a rule $\mathsf{pred}(\mathbf{x}) \Leftarrow \psi(\mathbf{x}) \in \Phi$ such that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi(\mathbf{z})$. We say that we have *unfolded* the predicate pred by the above rule. In general, ψ may itself contain predicate calls. To prove $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi(\mathbf{z})$, we must continue unfolding the remaining predicate calls according to rules of the respective predicates until no predicate calls remain.

It is natural to visualize an unfolding process as a tree. In fact, defining the semantics of inductive predicates based on such *unfolding trees* is a common approach in the literature (cf. [Iosif et al. 2013, 2014; Jansen et al. 2017; Matheja 2020]). In this article, we use a variant of unfolding trees, called Φ-*trees*, which we will formally introduce in Definition 7.1.

Example 6.1 (Φ-tree). *Recall the SID Φ from Fig. 1, which defines the predicate* tll. *Figure 6a depicts a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ with $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{tll}(x, y, z)$. Each node is labeled with a location and the stack variable evaluating to the location (if any). The depicted state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ thus corresponds to*

$$\begin{aligned} \mathfrak{s} \;&=\; \{x \mapsto 1, y \mapsto 4, z \mapsto 8, a \mapsto 5, b \mapsto 6, c \mapsto 7\}\,, \text{ and} \\ \mathfrak{h} \;&=\; \{1 \mapsto \langle 2, 3, nil\rangle\,, 2 \mapsto \langle 4, 5, nil\rangle\,, 5 \mapsto \langle 6, 7, nil\rangle\,, 4 \mapsto \langle nil, nil, 6\rangle\,, \\ &\qquad 6 \mapsto \langle nil, nil, 7\rangle\,, 7 \mapsto \langle nil, nil, 3\rangle\,, 3 \mapsto \langle nil, nil, 8\rangle\}. \end{aligned}$$

Figure 6b shows a Φ-tree $\mathfrak{t}$ corresponding to this state. Each node of $\mathfrak{t}$ is labeled with a rule instance*, i.e., a rule of the SID in which all variables—both formal parameters and existentially-quantified variables—have been instantiated with the locations of the state. This is different from other notions of unfolding trees, e.g., [Iosif et al. 2013], in which nodes are labeled by rules, not rule instances. Note that $\mathfrak{t}$ induces the heap $\mathfrak{h}$: $\mathfrak{h}$ is the union of all the points-to assertions that occur in the node labels of $\mathfrak{t}$. We denote this heap by* $\mathsf{heap}(\mathfrak{t})$.



(a) The states $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$, $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$.

(b) Φ-trees corresponding to the states.

Fig. 7. States $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathsf{lseg}(x, y)$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \mathsf{lseg}(y, z)$ and Φ-trees $\mathfrak{t}_1, \mathfrak{t}_2$ corresponding to the states. The tree $\mathfrak{t}_1$ contains one predicate call that is not unfolded, $\mathsf{lseg}(4, 6)$. We say that 4, the root of this folded predicate call, is a hole of the tree.

Φ-trees enable an alternative reading of the entailment $\mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z) \models_\Phi \mathsf{lseg}(x, z)$: The entailment is valid iff whenever $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z)$ holds, it is possible to find a Φ-tree $\mathfrak{t}$ with root $\mathsf{lseg}(\mathfrak{s}(x), \mathfrak{s}(z))$ such that $\mathsf{heap}(\mathfrak{t}) = \mathfrak{h}$.

6.2.2 Abstracting states by forests. Our next abstraction attempt is to encode the existence of a suitable Φ-tree. More precisely, we encode that the models of $\mathsf{lseg}(x, y)$ and $\mathsf{lseg}(y, z)$ each correspond to *partial* Φ-trees of $\mathsf{lseg}(x, z)$ that can be combined into an unfolding tree of $\mathsf{lseg}(x, z)$. A partial Φ-tree is obtained by prematurely stopping the unfolding process. Consequently, such a tree may contain *holes*—predicate calls that have not been unfolded.

Example 6.2 (Φ-trees with holes). *Recall the entailment* $\mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z) \models_\Phi \mathsf{lseg}(x, z)$ *from above. Figure 7a shows states* $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ *with* $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathsf{lseg}(x, y)$ *and* $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \mathsf{lseg}(y, z)$. *By the semantics of* $\star$, *it holds for* $\mathfrak{h} \triangleq \mathfrak{h}_1 \uplus \mathfrak{h}_2$ *that* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z)$.

How can Φ-trees be used to argue that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{lseg}(x, z)$*? Fig. 7b shows a* Φ-forest*—a set of Φ-trees—consisting of the partial Φ-trees* $\mathfrak{t}_1$ *and* $\mathfrak{t}_2$ *with* $\mathsf{heap}(\mathfrak{t}_1) = \mathfrak{h}_1$ *and* $\mathsf{heap}(\mathfrak{t}_2) = \mathfrak{h}_2$, *respectively. Notice that* $\mathfrak{t}_1$ *contains a* hole*: Since the predicate call* $\mathsf{lseg}(4, 6)$ *is not unfolded, the hole, location* 4, *is not allocated in the tree, even though it is the root parameter of the* hole predicate $\mathsf{lseg}(4, 6)$. *We can* merge $\mathfrak{t}_1$ *and* $\mathfrak{t}_2$ *into a larger tree by plugging* $\mathfrak{t}_2$ *into the hole of* $\mathfrak{t}_1$, *i.e., we add an edge from the hole of* $\mathfrak{t}_1$ *to the root of* $\mathfrak{t}_2$. *This is possible because the root of* $\mathfrak{t}_2$ *is labeled with the aforementioned hole predicate,* $\mathsf{lseg}(4, 6)$. *The resulting tree is a Φ-tree for* $\mathsf{lseg}(x, z)$. *That is, it is a tree without holes whose root is labeled with a rule instance of the predicate call* $\mathsf{lseg}(\mathfrak{s}(x), \mathfrak{s}(z))$. *By merging the two trees, we verified the above entailment: every model of* $\mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z)$ *is also a model of* $\mathsf{lseg}(x, z)$.

In fact, we go one step further and consider Φ-*forests* (cf. Definition 7.4), i.e., sets of partial Φ-trees. For example, the set $\{\mathfrak{t}_1, \mathfrak{t}_2\}$ illustrated in Fig. 7 is a Φ-forest.

Example 6.3 (Φ-forest). *Continuing Example 6.1, Fig. 8 depicts a* Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3, \mathfrak{t}_4\}$ *that encodes one way to obtain the state* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *through iterative unfolding of predicate calls. Both* $\mathfrak{t}_1$ *and* $\mathfrak{t}_2$ *only* partially *unfold the predicates at their roots, leaving locations* 5 *resp.* 6 *and* 7 *as holes. By merging the four trees, we get the tree* $\mathfrak{t}$ *from Example 6.1.*

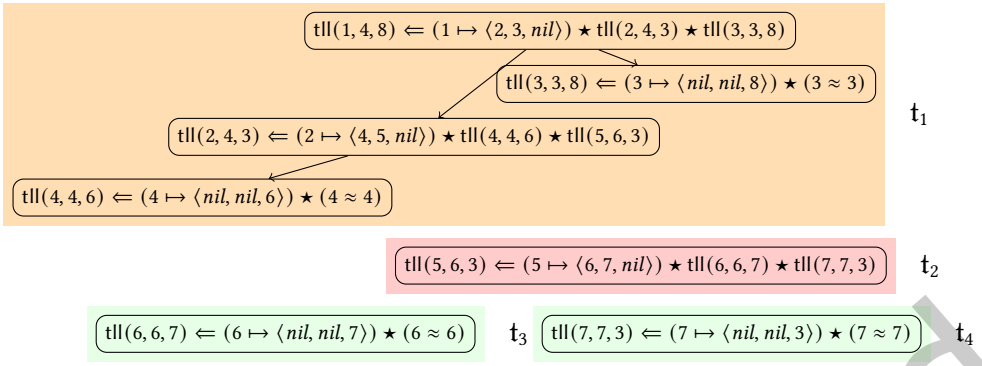


Fig. 8. A Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3, \mathfrak{t}_4\}$ for the state from Example 6.1, used in Example 6.3.

Our second idea towards a suitable abstraction is to abstract a state by computing all Φ-forests consisting of trees whose combined heap matches the heap of the state:

$$\mathrm{abst}_2(\mathfrak{s}, \mathfrak{h}) \triangleq \left\{ \mathfrak{f} \;\middle|\; \mathfrak{f} \text{ is a } \Phi\text{ -forest of } (\mathfrak{s}, \mathfrak{h}) \text{ with } \mathfrak{h} = \bigcup_{\mathfrak{t} \in \mathfrak{f}} \mathrm{heap}(\mathfrak{t}) \right\}.$$

6.2.3 Compositionality. We can define a suitable composition operation $\mathrm{abst}_2(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathrm{abst}_2(\mathfrak{s}, \mathfrak{h}_2)$ by computing all ways to merge the Φ-forests of $\mathrm{abst}_2(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathrm{abst}_2(\mathfrak{s}, \mathfrak{h}_2)$. This approach yields precisely the set of all Φ-forests of $\langle \mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2 \rangle$, i.e., the set $\mathrm{abst}_2(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$ from above, as required.

6.2.4 Finiteness. Unfortunately, abst_2 results in an *infinite* abstraction due to two main issues:

Issue 1 The tree nodes are labeled with concrete locations, so the result of abst_2 differs even if the states are isomorphic, i.e., identical up to renaming of locations. Apart from leading to an infinite abstraction, distinguishing such states is undesirable as they satisfy the same **GSL** formulas as long as we do not explicitly use constant locations.

Issue 2 If we keep track of all Φ-forests, the size of $\mathrm{abst}_2(\mathfrak{s}, \mathfrak{h})$ grows with the size of $\mathfrak{h}$—it is unbounded. For example, the abstraction of a list-segment of size n contains the forest that consist of a single tree with n nodes, the forest that consists of n one-node trees as well as all possibilities in between.

6.3 Third Attempt: Forest Projections

Our first attempt yields a finite abstraction that is not compositional, whereas our second attempt is compositional but not finite. We now construct a finite *and* compositional abstraction by considering an additional abstraction—called the *projection*— on top of Φ-forests with holes. To this end, we denote by $\mathrm{rootpred}(\mathfrak{t})$ the root and by $\mathrm{allholepreds}(\mathfrak{t})$ the hole predicates of a Φ-tree $\mathfrak{t}$.

Example 6.4. (1) *Let* $\mathfrak{t}_1, \mathfrak{t}_2$ *be the* Φ*-trees from Example 6.2, which are illustrated in Fig. 7b. Then,*
- $\mathrm{rootpred}(\mathfrak{t}_1) = \mathrm{lseg}(1, 6)$, $\mathrm{allholepreds}(\mathfrak{t}_1) = \{\mathrm{lseg}(4, 6)\}$, *and*
- $\mathrm{rootpred}(\mathfrak{t}_2) = \mathrm{lseg}(4, 6)$, $\mathrm{allholepreds}(\mathfrak{t}_2) = \emptyset$.

(2) *Let* $\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3, \mathfrak{t}_4$ *be the* Φ*-trees from Example 6.3, which are illustrated in Fig. 8. Then,*
- $\mathrm{rootpred}(\mathfrak{t}_1) = \mathrm{tll}(1, 4, 8)$, $\mathrm{allholepreds}(\mathfrak{t}_1) = \{\mathrm{tll}(5, 6, 3)\}$,
- $\mathrm{rootpred}(\mathfrak{t}_2) = \mathrm{tll}(5, 6, 3)$, $\mathrm{allholepreds}(\mathfrak{t}_2) = \{\mathrm{tll}(6, 6, 7), \mathrm{tll}(7, 7, 3)\}$,
- $\mathrm{rootpred}(\mathfrak{t}_3) = \mathrm{tll}(6, 6, 7)$, $\mathrm{allholepreds}(\mathfrak{t}_3) = \emptyset$, *and*

- $\mathsf{rootpred}(\mathsf{t}_4) = \mathsf{tll}(7,7,3)$, $\mathsf{allholepreds}(\mathsf{t}_4) = \emptyset$.

The *projection* of a Φ-forest $\mathfrak{f}$ can be viewed as a **GSL** formula encoding, for each tree $\mathsf{t} \in \mathfrak{f}$, a model of $\mathsf{rootpred}(\mathsf{t})$ from which models of $\mathsf{allholepreds}(\mathsf{t})$ have been subtracted, i.e.,[6]

$$\bigstar_{\mathsf{t}\in\mathfrak{f}} \left[(\bigstar \mathsf{allholepreds}(\mathsf{t})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathsf{t}) \right]. \tag{$\dagger$}$$

The goal of the projection operation is to combat Issue 2 identified above: to restore finiteness, we keep only limited information about each unfolding tree, and remember only its root predicate and its hole predicates. The magic wand introduced by the projection operation in the formula ($\dagger$) allows us to maintain the compositionality of the abstraction.

Example 6.5 (Forest projection—with locations). *Recall from Example 6.2, the states* $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathsf{lseg}(x,y)$, $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \mathsf{lseg}(y,z)$, *and the corresponding* Φ*-trees* $\mathsf{t}_1, \mathsf{t}_2$. *The projection of stack* $\mathfrak{s}$ *and* Φ*-forest* $\{\mathsf{t}_1, \mathsf{t}_2\}$ *is then the formula* $(\mathsf{lseg}(4,6) \mathrel{-\!\!\star} \mathsf{lseg}(1,6)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{lseg}(4,6))$.

6.3.1 Abstracting from locations. Our goal, which we will soon complete, has been to define a compositional abstraction over states. To this end, we introduced (partial) unfolding trees. These trees are naturally defined through the instantiation of SID rules with locations. Unfortunately, locations present an obstacle towards obtaining a finite abstraction (Issue 1): we get a different abstraction even for Φ-forests that encode the same model up to isomorphism! However, after having projected unfolding trees to formulas, we are able to reverse the instantiation of variables with locations. We in fact define the projection operation ($\dagger$) to output variables instead of locations: Say t is a Φ-tree of the state $\langle \mathfrak{s}, \mathfrak{h} \rangle$. Then we replace the locations in the formula ($\dagger$) as follows:

(1) Every location $v \in \mathsf{img}(\mathfrak{s})$ is replaced by a variable x satisfying $\mathfrak{s}(x) = v$.
(2) Every location in $\mathsf{dom}(\mathfrak{h}) \setminus \mathsf{img}(\mathfrak{s})$ is replaced by an *existentially-quantified* variable, because there exists a location in the heap $\mathfrak{h}$ that corresponds to the location in the formula ($\dagger$).
(3) All other locations are replaced by *universally-quantified* variables, because these locations do not occur in the heap $\mathfrak{h}$ (this holds because we will always assume $\langle \mathfrak{s}, \mathsf{heap}(\mathsf{t}) \rangle \in \mathbf{GStates}$) and can thus be picked in an arbitrary way.

We remark that the formal definition of projection uses non-standard quantifiers $\mathbb{\exists}$ and $\mathbb{\forall}$ in projections (further discussed below). For the moment, this difference does not matter; it is safe to replace them with the usual first-order quantifiers $\exists$ and $\forall$ for intuition.

Example 6.6 (Forest projection—with variables (without quantifiers)). *Continuing Example 6.5, the projection of* $\mathfrak{s}$ *and* Φ*-forest* $\{\mathsf{t}_1, \mathsf{t}_2\}$ *using the above replacement is*

$$(\mathsf{lseg}(y,z) \mathrel{-\!\!\star} \mathsf{lseg}(x,z)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{lseg}(y,z)).$$

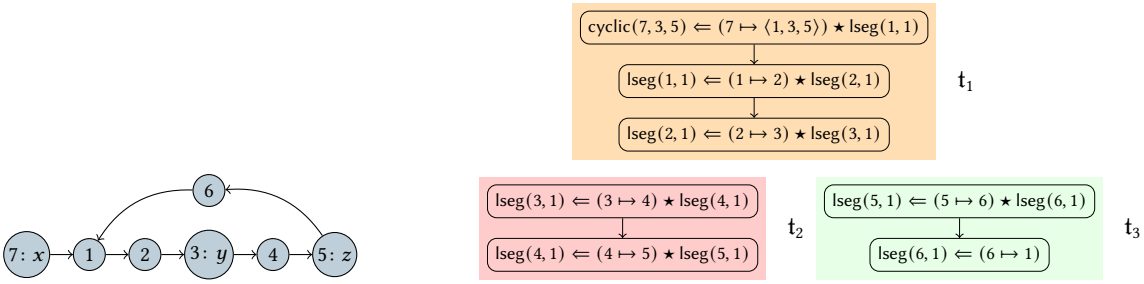
We will later prove that the projection operation is sound *(Lemma 7.25), i.e., for the given example,*

$$(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) \models_\Phi (\mathsf{lseg}(y,z) \mathrel{-\!\!\star} \mathsf{lseg}(x,z)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{lseg}(y,z)).$$

We further note that the idea of connecting holes with corresponding roots in Φ*-trees is mirrored on the level of formulas: since the magic wand* $\mathrel{-\!\!\star}$ *is the left-adjoint of the separating conjunction* $\star$, *an application of* modus ponens[7] *(formalized in Lemma 7.18) suffices to establish that*

$$(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) \models_\Phi \mathbf{emp} \mathrel{-\!\!\star} \mathsf{lseg}(x,z).$$

[6]Recall that $\bigstar\{\phi_1, \ldots \phi_n\}$ is a shortcut for $\phi_1 \star \ldots \star \phi_n$.
[7]i.e., $\phi \star (\phi \mathrel{-\!\!\star} \psi) \Rightarrow \psi$



(a) A model $\langle \mathfrak{s}, \mathfrak{h} \rangle$ of cyclic(x, y, z) (b) Φ-trees corresponding to the state $\langle \mathfrak{s}, \mathfrak{h} \rangle$.

Fig. 9. A state $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \text{cyclic}(x, y, z)$ and Φ-trees corresponding to this state.

Example 6.7 (Forest projection—with variables (and quantifiers)). *We consider a Φ, consisting of the list segment predicate* lseg *(see Example 3.3) and the following additional predicate:*

$$\text{cyclic}(x, y, z) \quad \Leftarrow \quad \exists a.\, x \mapsto \langle a, y, z \rangle \star \text{ls}(a, a)$$

Fig. 9b depicts a model $\langle \mathfrak{s}, \mathfrak{h} \rangle$ of cyclic(x, y, z) *and Φ-trees $\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3$ with $\mathfrak{h} = \text{heap}(\mathsf{t}_1) \cup \text{heap}(\mathsf{t}_2) \cup \text{heap}(\mathsf{t}_3)$. The projection of stack $\mathfrak{s}$ and Φ-forest $\{\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3\}$ is*

$$\exists a.\, (\text{lseg}(y, a) \mathrel{-\!\!\star} \text{cyclic}(x, y, z)) \star (\text{lseg}(z, a) \mathrel{-\!\!\star} \text{lseg}(y, a)) \star \text{lseg}(z, a).$$

We will later prove that the projection operation is sound *(Lemma 7.25), i.e., for the given example,*

$$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \exists a.\, (\text{lseg}(y, a) \mathrel{-\!\!\star} \text{cyclic}(x, y, z)) \star (\text{lseg}(z, a) \mathrel{-\!\!\star} \text{lseg}(y, a)) \star \text{lseg}(z, a).$$

Equipped with this extended projection operation, we are now in a position to specify the third (and almost final) abstraction function:

$$\text{abst}_3(\mathfrak{s}, \mathfrak{h}) \triangleq \left\{ \phi \;\middle|\; \phi \text{ is the projection of a } \Phi \text{ -forest } \mathsf{f} \text{ of } (\mathfrak{s}, \mathfrak{h}) \text{ with } \mathfrak{h} = \bigcup_{\mathsf{t} \in \mathsf{f}} \text{heap}(\mathsf{t}) \right\}.$$

6.3.2 Compositionality. As already hinted at in Example 6.6, the projection of formulas allows us to define a (computable) operator $\bullet$ with $\text{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \text{abst}_3(\mathfrak{s}, \mathfrak{h}_2) = \text{abst}_3(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$ such that:

(1) We have $\phi \star \psi \in \text{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \text{abst}_3(\mathfrak{s}, \mathfrak{h}_2)$ for all $\phi \in \text{abst}_3(\mathfrak{s}, \mathfrak{h}_1)$ and $\psi \in \text{abst}_3(\mathfrak{s}, \mathfrak{h}_2)$.
(2) The set of formulas $\text{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \text{abst}_3(\mathfrak{s}, \mathfrak{h}_2)$ is closed under application of modus ponens.
(3) The set $\text{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \text{abst}_3(\mathfrak{s}, \mathfrak{h}_2)$ is closed under certain rules for manipulating quantifiers.

Example 6.8 (Composition Operation on Projections). (1) *We recall the states $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \text{lseg}(x, y)$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \text{lseg}(y, z)$ from Example 6.2, and the Φ-trees $\mathsf{t}_1, \mathsf{t}_2$. The projection of $\mathfrak{s}$ and $\{\mathsf{t}_1\}$ is* $\text{lseg}(y, z) \mathrel{-\!\!\star} \text{lseg}(x, z)$*, and the projection of $\mathfrak{s}$ and $\{\mathsf{t}_2\}$ is* $\mathbf{emp} \mathrel{-\!\!\star} \text{lseg}(y, z)$*. Hence,*

$$(\text{lseg}(y, z) \mathrel{-\!\!\star} \text{lseg}(x, z)) \star (\mathbf{emp} \mathrel{-\!\!\star} \text{lseg}(y, z)) \in \text{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \text{abst}_3(\mathfrak{s}, \mathfrak{h}_2).$$

By applying modus ponens, we get $\mathbf{emp} \mathrel{-\!\!\star} \text{lseg}(x, z) \in \text{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \text{abst}_3(\mathfrak{s}, \mathfrak{h}_2)$*. The above reasoning approach will lead to a compositional proof of the entailment*

$$\text{lseg}(x, y) \star \text{lseg}(y, z) \models_\Phi \text{lseg}(x, z).$$

(2) *Let* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *be the model and let* $\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3$ *be the* Φ*-trees from Example 6.7. We set* $\mathfrak{h}_1 = \mathsf{heap}(\mathsf{t}_1) \cup \mathsf{heap}(\mathsf{t}_3)$ *and* $\mathfrak{h}_2 = \mathsf{heap}(\mathsf{t}_2)$*. The projection of* $\mathfrak{s}$ *and* $\{\mathsf{t}_1, \mathsf{t}_3\}$ *is*

$$\exists a.\, (\mathsf{lseg}(y, a) \mathrel{-\!\!\star} \mathsf{cyclic}(x, y, z)) \star \mathsf{lseg}(z, a),$$

and the projection of $\mathfrak{s}$ *and* $\{\mathsf{t}_2\}$ *is* $\forall a'.\, \mathsf{lseg}(z, a') \mathrel{-\!\!\star} \mathsf{lseg}(y, a')$*. Hence, we have*

$$[\exists a.\, (\mathsf{lseg}(y, a) \mathrel{-\!\!\star} \mathsf{cyclic}(x, y, z)) \star \mathsf{lseg}(z, a)] \star [\forall a'.\, \mathsf{lseg}(z, a') \mathrel{-\!\!\star} \mathsf{lseg}(y, a')] \in \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2).$$

By instantiating a' *with* a *and moving* $\exists a.$ *to the front of the formula, we get that*

$$\exists a.\, (\mathsf{lseg}(y, a) \mathrel{-\!\!\star} \mathsf{cyclic}(x, y, z)) \star \mathsf{lseg}(z, a) \star (\mathsf{lseg}(z, a) \mathrel{-\!\!\star} \mathsf{lseg}(y, a)) \in \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2).$$

By applying modus ponens (twice), we get $\exists a.\, \mathsf{cyclic}(x, y, z) \in \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2)$*. As the variable* a *does not appear free anymore, the quantifier can be dropped, and we get*

$$\mathsf{cyclic}(x, y, z) \in \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2).$$

The above reasoning approach will lead to a compositional proof of the entailment

$$\mathsf{fork}(x, y, z) \star \mathsf{lseg}(y, z) \models_\Phi \mathsf{cyclic}(x, y, z),$$

where Φ *extends the SID from Example 6.7 by the predicate*

$$\mathsf{fork}(x, y, z) \quad \Leftarrow \quad \exists a.\, (x \mapsto \langle a, y, z \rangle) \star \mathsf{ls}(a, y) \star \mathsf{ls}(z, a).$$

(3) *Let* $\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3, \mathsf{t}_4$ *be the* Φ*-trees from Example 6.3 for the state* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *of Example 6.1. We set* $\mathfrak{h}_1 = \mathsf{heap}(\mathsf{t}_1) \cup \mathsf{heap}(\mathsf{t}_3) \cup \mathsf{heap}(\mathsf{t}_4)$ *and* $\mathfrak{h}_2 = \mathsf{heap}(\mathsf{t}_2)$*. The projection of* $\mathfrak{s}$ *and* $\{\mathsf{t}_1, \mathsf{t}_3, \mathsf{t}_4\}$ *is*

$$\exists r.\, (\mathsf{tll}(a, b, c) \mathrel{-\!\!\star} \mathsf{tll}(x, y, z)) \star (b \mapsto \langle nil, nil, c \rangle) \star (c \mapsto \langle nil, nil, r \rangle),$$

and the projection of $\mathfrak{s}$ *and* $\{\mathsf{t}_2\}$ *is* $\forall r'.\, ((b \mapsto \langle nil, nil, c \rangle) \star (c \mapsto \langle nil, nil, r' \rangle)) \mathrel{-\!\!\star} \mathsf{tll}(a, b, c)$*. Hence,*

$$[\exists r.\, (\mathsf{tll}(a, b, c) \mathrel{-\!\!\star} \mathsf{tll}(x, y, z)) \star (b \mapsto \langle nil, nil, c \rangle) \star (c \mapsto \langle nil, nil, r \rangle)] \star [\forall r'.\, ((b \mapsto \langle nil, nil, c \rangle) \star (c \mapsto \langle nil, nil, r' \rangle)) \mathrel{-\!\!\star} \mathsf{tll}(a, b, c)] \in \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2).$$

By instantiating r' *with* r *and moving* $\exists r.$ *to the front of the formula, we get that*

$$\exists r.\, (\mathsf{tll}(a, b, c) \mathrel{-\!\!\star} \mathsf{tll}(x, y, z)) \star (b \mapsto \langle nil, nil, c \rangle) \star (c \mapsto \langle nil, nil, r \rangle) \star (((b \mapsto \langle nil, nil, c \rangle) \star (c \mapsto \langle nil, nil, r \rangle)) \mathrel{-\!\!\star} \mathsf{tll}(a, b, c)) \in \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2).$$

By applying modus ponens for the magic wand (twice), we get that

$$\exists r.\, \mathsf{tll}(x, y, z) \in \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2).$$

As the variable r *does not appear free anymore, the quantifier can be dropped and we get*

$$\mathsf{tll}(x, y, z) \in \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2).$$

The above reasoning approach will lead to a compositional proof of the entailment

$$\mathsf{tllHole}(x, y, z, a, b, c) \star (a \mapsto \langle b, c, nil \rangle) \models_\Phi \mathsf{tll}(x, y, z),$$

where Φ extends the TLL SID from Fig. 1 by the predicates

$$\begin{array}{lcl}
\mathsf{tllHole}(x,y,z,a,b,c) & \Leftarrow & \exists l,r.\,(x \mapsto \langle l,r,nil\rangle) \star \mathsf{helper}(l,r,y,z,a,b,c) \\
\mathsf{helper}(l,r,y,z,a,b,c) & \Leftarrow & (l \mapsto \langle y,a,nil\rangle) \star \mathsf{list4}(y,b,c,r,z) \\
\mathsf{list4}(y,b,c,r,z) & \Leftarrow & (y \mapsto \langle nil,nil,b\rangle) \star \mathsf{list3}(b,c,r,z) \\
\mathsf{list3}(b,c,r,z) & \Leftarrow & (b \mapsto \langle nil,nil,c\rangle) \star \mathsf{list2}(c,r,z) \\
\mathsf{list2}(c,r,z) & \Leftarrow & (c \mapsto \langle nil,nil,r\rangle) \star \mathsf{ptr}(r,z).
\end{array}$$

6.3.3 Guarded Quantifiers. We now discuss the semantics of the special quantifiers $\mathbb{\exists}$ and $\mathbb{\forall}$ used in the projection operation. We rely on a non-standard semantics because we want our approach to directly support SIDs with (dis-)equalities. If one would disallow (dis-)equalities in SIDs, one could use the usual $\exists$ and $\forall$ quantifiers instead of $\mathbb{\exists}$ and $\mathbb{\forall}$ (which is sufficient for the SIDs in Example 6.8). We motivate our non-standard semantics with the SID Φ given by the following predicates:

$$p(x,a,b) \quad \Leftarrow \quad \exists y.\,(x \mapsto y) \star q(y,a) \star x \neq a \star a \neq b \qquad q(y,a) \quad \Leftarrow \quad (y \mapsto nil) \star y \neq a$$

We consider the stack $\mathfrak{s} = \{x \mapsto 1\}$ and the heap $\mathfrak{h} = \{1 \mapsto 2, 2 \mapsto nil\}$. We further consider the unfolding tree $\mathfrak{t}$ consisting of the root $p(1,4,5) \Leftarrow (1 \mapsto 2) \star q(2,4) \star 1 \neq 4 \star 4 \neq 5$ with the single child $q(2,4) \Leftarrow (2 \mapsto nil) \star 2 \neq 4$; note that $\mathsf{heap}(\mathfrak{t}) = \mathfrak{h}$. The projection of this unfolding tree is the formula $\mathbb{\forall} a,b.\,p(x,a,b)$. As discussed earlier, we want that the projection operation is sound, i.e., $\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi \mathbb{\forall} a,b.\,p(x,a,b)$. However, using the standard quantifier $\forall$ instead of $\mathbb{\forall}$ does not work:

$$(1)\ \langle \mathfrak{s},\mathfrak{h}\rangle \not\models_\Phi p(1,5,5) \quad (2)\ \langle \mathfrak{s},\mathfrak{h}\rangle \not\models_\Phi p(1,1,5) \quad (3)\ \langle \mathfrak{s},\mathfrak{h}\rangle \not\models_\Phi p(1,2,5)$$

The above example shows that we need to prevent instantiating universally quantified variables with

(1) identical locations, see $\langle \mathfrak{s},\mathfrak{h}\rangle \not\models_\Phi p(1,5,5)$,
(2) locations that are in the image of the stack, see $\langle \mathfrak{s},\mathfrak{h}\rangle \not\models_\Phi p(1,1,5)$, and
(3) locations that are existentially quantified, see $\langle \mathfrak{s},\mathfrak{h}\rangle \not\models_\Phi p(1,2,5)$.

For the semantics of $\mathbb{\forall}$ we use that in $\mathbf{SL}_{\mathrm{btw}}$ all existentially quantified variables (that are not equal to a parameter) are allocated because of the establishment requirement, and set

$\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi \mathbb{\forall} \langle a_1,\dots,a_k\rangle.\,\phi$ iff for all pairwise different locations
$v_1,\dots,v_k \in \mathbf{Loc} \setminus (\mathrm{dom}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s}))$ it holds that $\langle \mathfrak{s} \cup \{a_1 \mapsto v_1,\dots,a_k \mapsto v_k\},\mathfrak{h}\rangle \models_\Phi \phi$

Our main requirement for giving semantics to the $\mathbb{\exists}$ quantifier is the correctness of the following entailment, which we already used in Example 6.8: $(\mathbb{\exists} e.\,\phi) \star (\mathbb{\forall} a.\,\psi) \models_\Phi \mathbb{\exists} e.\,\phi \star \psi[a/e]$ (†)

This is ensured by the following semantics for $\mathbb{\exists}$:

$\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi \mathbb{\exists} \langle e_1,\dots,e_k\rangle.\,\phi$ iff for all pairwise different locations
$v_1,\dots,v_k \in \mathrm{dom}(\mathfrak{h}) \setminus \mathrm{img}(\mathfrak{s})$ such that $\langle \mathfrak{s} \cup \{e_1 \mapsto v_1,\dots,e_k \mapsto v_k\},\mathfrak{h}\rangle \models_\Phi \phi$.

We call our quantifiers $\mathbb{\exists}$ and $\mathbb{\forall}$ *guarded* because they exclude the instantiation of variables with certain locations. We note that our quantifiers are *not* dual, i.e., $\mathbb{\exists}\mathbf{x}.\,\phi$ is *not* equivalent to $\neg\mathbb{\forall}\mathbf{x}.\,\neg\phi$. However, we believe that our semantics is sufficiently motivated by our considerations on the soundness of the projection and the entailment (†).

6.3.4 Finiteness. Did we solve Issues 1 and 2 from the second attempt? Unfortunately not completely. However, one additional restriction on unfolding forests will be sufficient to guarantee the finiteness of the abstraction. We first explain the issue by means of an example:

Example 6.9. *Let Φ be an SID that defines the list-segment predicate* lseg. *Let $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{lseg}(x, nil)$ with $|\mathfrak{h}| > n$. Then, there exists a forest $\mathfrak{f}$ such that $\mathfrak{h} = \bigcup_{\mathfrak{t} \in \mathfrak{f}} \mathsf{heap}(\mathfrak{t})$ and whose projection is*

$$\exists y_1, \ldots, y_n . \mathsf{lseg}(y_n, nil) \mathbin{\star} (\mathsf{lseg}(y_n, nil) \mathbin{-\!\!\star} \mathsf{lseg}(y_{n-1}, nil)) \\ \mathbin{\star} \cdots \mathbin{\star} (\mathsf{lseg}(y_2, nil) \mathbin{-\!\!\star} \mathsf{lseg}(y_1, nil)) \mathbin{\star} (\mathsf{lseg}(y_1, nil) \mathbin{-\!\!\star} \mathsf{lseg}(x, nil))$$

As there exist such models $\langle \mathfrak{s}, \mathfrak{h} \rangle$ for arbitrary $n \in \mathbb{N}$, there are infinitely many (non-equivalent) formulas resulting from projections of unfolding forests.

Fortunately, we do not need to consider all unfolding forests for deciding the satisfiability of the considered separation logic **GSL**. We recall that our goal is to define a compositional abstraction:

$$\mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2) = \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$$

Hence, we need to ensure that every unfolding tree of $\langle \mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2 \rangle$ can be composed via $\bullet$ from unfolding trees of $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$. In our approach we will have the guarantee that $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ are guarded (cf. Corollary 8.19). With this in mind, let us consider an unfolding tree of $\langle \mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2 \rangle$ that is composed of some trees of $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$, i.e., without loss of generality there is a pointer that is allocated in $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and points to a location in $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$. Then, this pointer is dangling for the state $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and the target of this pointer is in the image of the stack. Now, we recall from the definition of composition that the target of the pointer is a hole of an unfolding tree of $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and the root of an unfolding tree of $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$. Thus, we can restrict our attention to unfolding trees whose roots and holes are in the image of the stack! This motivates the following definition:

An unfolding tree $\mathfrak{t}$ is $\mathfrak{s}$-*delimited*, if the root and holes of $\mathfrak{t}$ are in the image of the stack $\mathfrak{s}$.

Equipped with this definition (which we will formalize in Definition 8.10), we restrict the abstraction function abst_3 to forests of delimited unfolding trees. This guarantees the finiteness of the abstraction: The formulas resulting from the projection of such forests have the property that (1) all root parameters of predicate calls are free variables and every variable occurs at most once as a root parameter, and (2) all root parameters of predicates calls on the left-hand side of a magic wand are free variables and every variable occurs at most once as a root parameter. Because the number of free variables is bounded, finiteness easily follows.

6.4 Summary of Overview

To sum up, we propose abstracting the state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ in the following way:

(1) We compute all $\mathfrak{s}$-delimited Φ-forests of $\langle \mathfrak{s}, \mathfrak{h} \rangle$.
(2) We project these forests onto formulas.
(3) The abstraction of $\langle \mathfrak{s}, \mathfrak{h} \rangle$ is the set of all these formulas; we call this set the *type* of $\langle \mathfrak{s}, \mathfrak{h} \rangle$.

The resulting abstraction is (1) finite (the set of types is finite), (2) compositional (we have $\mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_2) = \mathsf{abst}_3(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$), and (3) computable (we only need to apply rules for modus ponens and for manipulating quantifiers as illustrated in Example 6.8).

Outline of the following sections. In the remainder of this article, we give the technical details for the material overviewed in this section. In Section 7, we formalize Φ-forests (Section 7.1), their projections (Section 7.2), and how to compose forest projections (Section 7.3). The type abstraction is introduced in Section 8. We discuss how the satisfiability problem for guarded **SL** formulas can be reduced to computing types in Section 8.1. We formalize $\mathfrak{s}$-delimited forests in Section 8.3 and discuss how types can be computed compositionally in Sections 8.4 and 8.5. Finally, in Section 9, we present algorithms for computing the types of **GSL** formulas, summarize our overall decision procedure, and discuss our decidability and complexity results.

7 FORESTS AND THEIR PROJECTIONS

We now start formalizing the concepts that have been informally introduced in Section 6: Φ-*forests* (Section 7.1), their projection onto formulas (Section 7.2), and how to compose them (Section 7.3).

7.1 Forests

Our main objects of study in this section are Φ-*forests* (Definition 7.4) made up of Φ-*trees* (Definition 7.1). As motivated in Section 6, a Φ-tree encodes one fixed way to unfold a predicate call by means of the rules of the SID Φ. The differences between the unfolding trees of Iosif et al. [2013, 2014]; Jansen et al. [2017] and our Φ-trees are that (1) we instantiate variables with locations, and (2) Φ-trees can have *holes*, i.e., we allow that one or more of the predicate calls introduced (by means of recursive rules) in the unfolding process remain folded.

7.1.1 Rule instances. We annotate every node of a Φ-tree with a *rule instance* of the SID Φ, i.e., a formula obtained from a rule of the SID by instantiating both the formal arguments of the predicates and the existentially quantified variables of the rule with locations:

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

In the above definition, we refer only to those (dis-)equalities that occur explicitly in the formula, not those implied by recursive calls or by the separating conjunction. Validity of these (dis-)equalities is straightforward to check because all variables have been instantiated with concrete locations.

The notion of a rule instance is motivated as follows: whenever $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \mathsf{pred}(\mathbf{v})$, there is at least one rule instance $(\mathsf{pred}(\mathbf{v}) \Leftarrow \psi) \in \mathbf{RuleInst}(\Phi)$ such that $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \psi$.

7.1.2 Φ-trees. We represent a Φ-tree as a partial function $\mathfrak{t} \colon \mathbf{Loc} \rightharpoonup \left(2^{\mathbf{Loc}} \times \mathbf{RuleInst}(\Phi)\right)$, where the set **Loc** of locations serves as the nodes of the tree; every node is mapped to its successors in the (directed) tree and to its label, a rule instance. Moreover, for $\mathfrak{t}$ to be a Φ-tree, it must satisfy additional consistency criteria. To formalize these criteria, we fix some SID Φ and a node

$$\mathfrak{t}(l) = \left\langle \mathbf{v}, (\mathsf{pred}(\mathbf{v}) \Leftarrow (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{v_1}) \star \cdots \star \mathsf{pred}_m(\mathbf{v_m}) \star \Pi) \right\rangle,$$

where Π is a set of equalities and disequalities. Notice that all rule instances are of the above form because—by our global assumptions in Section 3.4.5—Φ satisfies the progress property. We introduce the following shortcuts for the node at location l to simplify working with Φ-trees:

$$\begin{aligned}
\mathsf{succ}_\mathfrak{t}(l) &\triangleq \mathbf{v} && \text{(locations corresponding to the successors of node } l\text{)}\\
\mathsf{head}_\mathfrak{t}(l) &\triangleq \mathsf{pred}(\mathbf{v}) && \text{(the predicate on the lhs of the rule instance)}\\
\mathsf{heap}_\mathfrak{t}(l) &\triangleq \{a \mapsto \mathbf{b}\} && \text{(the unique heap satisfying the points-to assertion in the rule instance)}\\
\mathsf{calls}_\mathfrak{t}(l) &\triangleq \left\{\mathsf{pred}_1(\mathbf{v_1}), \ldots, \mathsf{pred}_m(\mathbf{v_m})\right\} && \text{(the predicate calls in the rule instance)}\\
\mathsf{rule}_\mathfrak{t}(l) &\triangleq \mathsf{pred}(\mathbf{v}) \Leftarrow (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{v_1}) \star \cdots \star \mathsf{pred}_m(\mathbf{v_m}) \star \Pi && \text{(the rule instance)}
\end{aligned}$$

Moreover, we define the *hole predicates* of l as those predicate calls in $\mathsf{calls}_\mathfrak{t}(l)$ whose root does not occur in $\mathsf{succ}_\mathfrak{t}(l)$; the *holes* of l are the corresponding locations:

- $\mathsf{holepreds}_\mathfrak{t}(l) \triangleq \{\mathsf{pred}'(\mathbf{z}') \in \mathsf{calls}_\mathfrak{t}(l) \mid \forall c \in \mathsf{succ}_\mathfrak{t}(l).\, \mathsf{head}_\mathfrak{t}(c) \neq \mathsf{pred}'(\mathbf{z}')\}$, and
- $\mathsf{holes}_\mathfrak{t}(l) \triangleq \left\{\mathsf{predroot}(\mathsf{pred}'(\mathbf{z}')) \mid \mathsf{pred}'(\mathbf{z}') \in \mathsf{holepreds}_\mathfrak{t}(l)\right\}$.

We lift some of the above definitions from individual locations to entire trees t:

$$\mathsf{heap}(\mathsf{t}) \triangleq \bigcup_{c \in \mathrm{dom}(\mathsf{t})} \mathsf{heap}_{\mathsf{t}}(c) \qquad \text{(the heap satisfying exactly the points-to assertions in } \mathsf{t})$$

$$\mathsf{ptrlocs}(\mathsf{t}) \triangleq \bigcup_{(c \mapsto \mathbf{d}) \in \mathsf{heap}(\mathsf{t})} \{c\} \cup \mathbf{d} \qquad \text{(all locations that appear in points-to assertions in } \mathsf{t})$$

$$\mathsf{allholepreds}(\mathsf{t}) \triangleq \bigcup_{c \in \mathrm{dom}(\mathsf{t})} \mathsf{holepreds}_{\mathsf{t}}(c) \qquad \text{(all hole predicates in } \mathsf{t})$$

$$\mathsf{allholes}(\mathsf{t}) \triangleq \bigcup_{l \in \mathrm{dom}(\mathsf{t})} \mathsf{holes}_{\mathsf{t}}(l) \qquad \text{(all holes in } \mathsf{t})$$

We denote by $\mathsf{graph}(\mathsf{t})$ the directed graph induced by the successors of locations in t. That is,

$$\mathsf{graph}(\mathsf{t}) \triangleq \langle \mathrm{dom}(\mathsf{t}), \{\langle x, y\rangle \mid x \in \mathrm{dom}(\mathsf{t}), y \in \mathsf{succ}_{\mathsf{t}}(x)\}\rangle\,.$$

The *height* of t is the length of the longest path in the directed graph $\mathsf{graph}(\mathsf{t})$.

Definition 7.1 (Φ-Tree). *A partial function* $\mathsf{t} \colon \mathbf{Loc} \rightharpoonup \left(2^{\mathbf{Loc}} \times \mathbf{RuleInst}(\Phi)\right)$ *is a* Φ-tree *iff*

(1) Φ *is in the fragment of SIDs of bounded treewidth, i.e.,* $\Phi \in \mathbf{ID}_{\mathrm{btw}}$,
(2) $\mathsf{graph}(\mathsf{t})$ *is a directed tree, and*
(3) t *is* Φ-consistent*, i.e., for all locations* $l \in \mathrm{dom}(\mathsf{t})$*, we have:*
   - l *is the single allocated location in its rule instance, i.e.,* $\mathsf{heap}_{\mathsf{t}}(l) = \{l \mapsto \ldots\}$,
   - l *points to its successors in* t*, i.e.,* $\mathsf{heap}_{\mathsf{t}}(l) = \{l \mapsto \mathbf{b}\}$ *implies* $\mathsf{succ}_{\mathsf{t}}(l) \subseteq \mathbf{b}$*, and*
   - *the predicate calls associated with the successors* $\mathsf{succ}_{\mathsf{t}}(l) = \langle v_1, \ldots, v_k\rangle$*, of* l *appear in the rule instance at location* l*, i.e.,* $\{\mathsf{head}_{\mathsf{t}}(v_1), \ldots, \mathsf{head}_{\mathsf{t}}(v_k)\} \subseteq \mathsf{calls}_{\mathsf{t}}(l)$.

Since every Φ*-tree* t *is a directed tree, it has a root, which we denote by* $\mathsf{root}(\mathsf{t})$*; the corresponding predicate call is* $\mathsf{rootpred}(\mathsf{t}) \triangleq \mathsf{head}_{\mathsf{t}}(\mathsf{root}(\mathsf{t}))$.

Example 7.2 (Φ-Tree). (1) *A* Φ*-tree over the SID* $\Phi_{\mathsf{odd/even}}$ *(cf. Example 3.3) is given by*

$$\mathsf{t}(l) \triangleq \begin{cases} \langle b, \mathsf{even}(l_1, a) \Leftarrow (l_1 \mapsto b) \star \mathsf{odd}(b, a)\rangle & \text{if } l = l_1 \\ \langle \emptyset, \mathsf{odd}(b, a) \Leftarrow (b \mapsto l_2) \star \mathsf{even}(l_2, a)\rangle & \text{if } l = b \\ \bot & \text{otherwise.} \end{cases}$$

Formally, t *is defined over the locations* $\mathrm{dom}(\mathsf{t}) = \{l_1, b\}$*. Moreover, we have* $\mathsf{succ}_{\mathsf{t}}(l_1) = \{b\}$, $\mathsf{head}_{\mathsf{t}}(l_1) = \mathsf{even}(l_1, a)$, $\mathsf{calls}_{\mathsf{t}}(l_1) = \{\mathsf{odd}(b, a)\}$, $\mathsf{heap}(\mathsf{t}) = \{l_1 \mapsto b, b \mapsto l_2\}$, $\mathsf{heap}_{\mathsf{t}}(l_1) = \{l_1 \mapsto b\}$, $\mathsf{ptrlocs}(\mathsf{t}) = \{l_1, b, l_2\}$, $\mathsf{allholes}(\mathsf{t}) = \{l_2\}$*, and* $\mathsf{allholepreds}(\mathsf{t}) = \{\mathsf{even}(l_2, a)\}$.

(2) *All of the trees considered in Section 6 are* Φ*-trees.*

We remark that the above definition of Φ-trees does not account for rule instances in which the same predicate call appears multiple times. Similarly, we do not account for multiple predicate calls with the same root parameter. As we will see in Section 8.3, such cases do not need to be considered. We can thus ignore these cases in favor of a simpler formalization.

Our main motivation for considering Φ-trees is that they give a more structured view on models of predicate calls. In particular, every such model corresponds to (at least one) Φ-tree without holes:

Lemma 7.3. *Let* $\langle \mathfrak{s}, \mathfrak{h}\rangle$ *be a state and* $\mathsf{pred} \in \mathbf{Preds}(\Phi)$*. Then,* $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \mathsf{pred}(z_1, \ldots, z_k)$ *iff there exists a* Φ*-tree* t *with* $\mathsf{rootpred}(\mathsf{t}) = \mathsf{pred}(\mathfrak{s}(z_1), \ldots, \mathfrak{s}(z_k))$, $\mathsf{allholepreds}(\mathsf{t}) = \emptyset$*, and* $\mathsf{heap}(\{\mathsf{t}\}) = \mathfrak{h}$.

Proof. The statement directly follows by induction on the number of rules applied to derive $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \mathsf{pred}(z_1, \ldots, z_k)$ resp. the height of the tree t. □

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

Definition 7.4 (Φ-Forest). *A* Φ-forest f *is a finite set of* Φ-*trees* $\mathsf{f} = \{\mathsf{t}_1, \ldots, \mathsf{t}_k\}$ *with pairwise disjoint locations, i.e.,* $\mathrm{dom}(\mathsf{t}_i) \cap \mathrm{dom}(\mathsf{t}_j) = \emptyset$ *for* $i \neq j$.

We assume that all definitions are lifted from Φ-trees to Φ-forests, i.e., for $\mathsf{f} = \{\mathsf{t}_1, \ldots, \mathsf{t}_k\}$, we define

- the *induced heap of* f as $\mathsf{heap}(\mathsf{f}) \triangleq \bigcup_{\mathsf{t}\in\mathsf{f}} \mathsf{heap}(\mathsf{t})$; if $l \in \mathrm{dom}(\mathsf{t}_i)$ then $\mathsf{rule}_{\mathsf{f}}(l) = \mathsf{rule}_{\mathsf{t}_i}(l)$;
- $\mathsf{graph}(\mathsf{f}) \triangleq \big\langle \mathrm{dom}(\mathsf{f}), \{\langle x, y\rangle \mid 1 \leq i \leq k, x \in \mathrm{dom}(\mathsf{t}_i), y \in \mathsf{succ}_{\mathsf{t}_i}(x)\}\big\rangle$; and
- $\mathrm{dom}(\mathsf{f}) \triangleq \bigcup_i \mathrm{dom}(\mathsf{t}_i)$; $\mathsf{roots}(\mathsf{f}) \triangleq \{\mathsf{root}(\mathsf{t}_i) \mid 1 \leq i \leq k\}$; $\mathsf{allholes}(\mathsf{f}) \triangleq \bigcup_{1\leq i\leq k} \mathsf{allholes}(\mathsf{t}_i)$.

Example 7.5 (Φ-Forest). *Both Example 6.3 and Example 6.6 define a* Φ-*forest.*

7.1.4 Composing Forests. As motivated in Section 6.2.3, Φ-forests are composed by (1) taking their disjoint union and (2) optionally merging pairs of trees of the resulting forest by identifying the root of one tree with a hole of another tree.

Disjoint union of forests. The union of two Φ-forests corresponds to ordinary set union, provided no location is in the domain of both forests; otherwise, it is undefined.

Definition 7.6 (Union of Φ-forests). *Let* $\mathsf{f}_1, \mathsf{f}_2$ *be* Φ-*forests. The* union *of* $\mathsf{f}_1, \mathsf{f}_2$ *is given by*

$$\mathsf{f}_1 \uplus \mathsf{f}_2 \triangleq \begin{cases} \mathsf{f}_1 \cup \mathsf{f}_2 & \text{if } \mathrm{dom}(\mathsf{f}_1) \cap \mathrm{dom}(\mathsf{f}_2) = \emptyset, \\ \bot, & \text{otherwise.} \end{cases}$$

Lemma 7.7. *Let* $\mathsf{f} = \mathsf{f}_1 \uplus \mathsf{f}_2$. *Then* $\mathsf{heap}(\mathsf{f}) = \mathsf{heap}(\mathsf{f}_1) \uplus \mathsf{heap}(\mathsf{f}_2)$.

Proof. $\mathsf{heap}(\mathsf{f}) = \bigcup_{\mathsf{t}\in\mathsf{f}} \mathsf{heap}(\mathsf{t}) = (\bigcup_{\mathsf{t}\in\mathsf{f}_1} \mathsf{heap}(\mathsf{t})) \cup (\bigcup_{\mathsf{t}\in\mathsf{f}_2} \mathsf{heap}(\mathsf{t})) = \mathsf{heap}(\mathsf{f}_1) \uplus \mathsf{heap}(\mathsf{f}_2)$. (Where we have $\uplus$ rather than $\cup$ because $\mathsf{f}_1 \uplus \mathsf{f}_2$ is defined.) □

Splitting forests. We formalize the process of merging Φ-trees in a roundabout way: we first define a way to *split* the trees of a forest into sub-trees at a fixed set of locations—the inverse of merging forests. This may seem like an arbitrary choice, but will simplify the technical development in follow-up sections. We first consider two examples of splitting before formalizing it in Definition 7.9.

Example 7.8 (Splitting forests).
(1) *Let* t *be the* Φ-*tree from Example 7.2. The* $\{b\}$-*split of* $\{\mathsf{t}\}$ *is given by* $\{\mathsf{t}_1, \mathsf{t}_2\}$, *for the trees* $\mathsf{t}_1 = \{l_1 \mapsto \langle \emptyset, \mathsf{even}(l_1, a) \Leftarrow (l_1 \mapsto b) \star \mathsf{odd}(b, a)\rangle\}$ *and* $\mathsf{t}_2 = \{b \mapsto \langle \emptyset, \mathsf{odd}(b, a) \Leftarrow (b \mapsto l_2) \star \mathsf{even}(l_2, a)\rangle\}$ $\{\mathsf{t}_1, \mathsf{t}_2\}$ *is the* $\mathbf{l}$-*split of* $\{\mathsf{t}\}$ *for all* $\mathbf{l} \supseteq \{b\}$*: in our definition of* $\mathbf{l}$-*split we will not require for the locations in* $\mathbf{l}$ *to actually occur in the forest.*
(2) *Recall the forest* $\mathsf{f} = \{\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3\}$ *from Example 6.3 and the tree* t *from Example 6.1. Then* f *is the* $\{2, 4\}$-*split of* $\{\mathsf{t}\}$. *Likewise,* f *is the* $\{1, 2, 4, 7\}$-*split of* $\{\mathsf{t}\}$, *because* 1 *is the root of a tree and* 7 *does not occur in the forest. In contrast,* f *is* not *the* $\{1, 2, 5\}$-*split of* $\{\mathsf{t}\}$, *because* $5 \in \mathrm{dom}(\mathsf{f}) \setminus \mathsf{roots}(\mathsf{f})$.

Definition 7.9 ($\mathbf{l}$-split). *Let* $\mathsf{f}, \bar{\mathsf{f}}$ *be* Φ-*forests and* $\mathbf{l} \subseteq \mathbf{Loc}$. *Then* $\bar{\mathsf{f}}$ *is an* $\mathbf{l}$-split *of* f *if*

(1) *both forests cover the same locations, i.e.,* $\mathrm{dom}(\mathsf{f}) = \mathrm{dom}(\bar{\mathsf{f}})$,
(2) *both forests contain the same rule instances, i.e.,* $\mathsf{rule}_{\mathsf{f}}(d) = \mathsf{rule}_{\bar{\mathsf{f}}}(d)$ *for all* $d \in \mathrm{dom}(\mathsf{f})$, *and*
(3) *the graph of* $\bar{\mathsf{f}}$ *is obtained from the graph of* f *by removing edges leading to locations in* $\mathbf{l}$, *i.e.,* $\mathsf{graph}(\bar{\mathsf{f}}) = \mathsf{graph}(\mathsf{f}) \setminus \{(a, b) \mid a \in \mathbf{Loc}, b \in \mathbf{l}\}$.

Lemma 7.10 (Uniqueness of $\mathbf{l}$-split). *For all* $\mathbf{l} \subseteq \mathbf{Loc}$, *every* Φ-*forest has a unique* $\mathbf{l}$-*split* $\mathsf{split}(\mathsf{f}, \mathbf{l})$.

Proof. See Appendix A.10. □

To formalize how we merge trees, we define a derivation relation $\blacktriangleright^*$ between forests in which we iteratively split trees at suitable locations. Intuitively, $\mathsf{f}_1 \blacktriangleright^* \mathsf{f}_2$ holds if splitting the trees in f_2 at zero or more locations yields f_1; or, equivalently, if "merging" zero or more trees of f_1 yields f_2.

Definition 7.11 (Forest derivation). *The forest f_2 is* one-step derivable *from the forest f_1, denoted $\mathsf{f}_1 \blacktriangleright \mathsf{f}_2$ iff there exists a location $l \in \mathrm{dom}(\mathsf{f})$ such that $\mathsf{f}_1 = \mathsf{split}(\mathsf{f}_2, \{l\})$.*
The reflexive-transitive closure of $\blacktriangleright$ is denoted by $\blacktriangleright^*$.

Example 7.12. *Let $\bot$ denote the everywhere undefined partial function. Then, consider the forests $\mathsf{f} \triangleq \{\mathsf{t}_1, \mathsf{t}_2\}$ and $\bar{\mathsf{f}} \triangleq \{\bar{\mathsf{t}}\}$ given by the trees below. Then $\mathsf{f} \blacktriangleright \bar{\mathsf{f}}$ because $\mathsf{f} = \mathsf{split}(\bar{\mathsf{f}}, l_2)$.*

$$\begin{aligned}
\mathsf{t}_1 &\triangleq \{l_1 \mapsto \langle \bot, \mathsf{odd}(l_1, l_4) \Leftarrow (l_1 \mapsto l_2) \star \mathsf{even}(l_2, l_4)\rangle\} \\
\mathsf{t}_2 &\triangleq \{l_2 \mapsto \langle l_3, \mathsf{even}(l_2, l_4) \Leftarrow (l_2 \mapsto l_3) \star \mathsf{odd}(l_3, l_4)\rangle, \\
&\qquad l_3 \mapsto \langle \bot, \mathsf{odd}(l_3, l_4) \Leftarrow (l_3 \mapsto l_4)\rangle\} \\
\bar{\mathsf{t}} &\triangleq \{l_1 \mapsto \langle l_2, \mathsf{odd}(l_1, l_4) \Leftarrow (l_1 \mapsto l_2) \star \mathsf{even}(l_2, l_4)\rangle, \\
&\qquad l_2 \mapsto \langle l_3, \mathsf{even}(l_2, l_4) \Leftarrow (l_2 \mapsto l_3) \star \mathsf{odd}(l_3, l_4)\rangle, \\
&\qquad l_3 \mapsto \langle \bot, \mathsf{odd}(l_3, l_4) \Leftarrow (l_3 \mapsto l_4)\rangle\}.
\end{aligned}$$

We note that multiple steps of $\blacktriangleright$ correspond to splitting at multiple locations, because

$$\mathsf{split}(\mathsf{f}, \{l_1, \ldots, l_k\}) = \mathsf{split}(\ldots \mathsf{split}(\mathsf{split}(\mathsf{f}, \{l_1\}), \{l_2\}), \ldots, \{l_k\}).$$

Lemma 7.13. *$\mathsf{f}_1 \blacktriangleright^* \mathsf{f}_2$ iff there exists a set of locations $\mathbf{l}$ with $\mathsf{f}_1 = \mathsf{split}(\mathsf{f}_2, \mathbf{l})$.*

Moreover, forests in the $\blacktriangleright^*$ relation describe the same states:

Lemma 7.14. *Let f be a Φ-forest and $\bar{\mathsf{f}} \blacktriangleright^* \mathsf{f}$. Then $\mathsf{heap}(\bar{\mathsf{f}}) = \mathsf{heap}(\mathsf{f})$.*

Proof. Since $\bar{\mathsf{f}} \blacktriangleright^* \mathsf{f}$, there exists—by Lemma 7.13—a set of locations $\mathbf{l}$ with $\bar{\mathsf{f}} = \mathsf{split}(\mathsf{f}, \mathbf{l})$. By definition of $\mathbf{l}$-splits, we have (1) $\mathrm{dom}(\bar{\mathsf{f}}) = \mathrm{dom}(\mathsf{f})$ and (2) $\mathsf{rule}_{\bar{\mathsf{f}}}(l) = \mathsf{rule}_{\mathsf{f}}(l)$ for every location $l \in \mathrm{dom}(\bar{\mathsf{f}})$. Consequently, $\mathsf{heap}(\bar{\mathsf{f}}) = \mathsf{heap}(\mathsf{f})$. □

Based on the $\blacktriangleright^*$, we define the *composition* operation on pairs of forests as motivated in Section 6.2:

Definition 7.15 (Forest composition). *The* composition *of f_1 and f_2 is $\mathsf{f}_1 \bullet_{\mathrm{F}} \mathsf{f}_2 \triangleq \{\mathsf{f} \mid \mathsf{f}_1 \uplus \mathsf{f}_2 \blacktriangleright^* \mathsf{f}\}$.*

7.2 Forest Projections

In Section 6.3, we informally presented the *projection* of Φ-forests onto **GSL** formulas, and discussed the need for using *guarded quantifiers*. As a reminder, we repeat here the informal definition of the projection: Given a stack s and a Φ-forest $\mathsf{f} = \{\mathsf{t}_1, \ldots, \mathsf{t}_k\}$,

(1) we compute the formula $\phi \triangleq \bigstar_{1 \le i \le k} (\bigstar \mathsf{allholepreds}(\mathsf{t}_i)) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathsf{t}_i)$, in which all parameters of all predicate calls are locations;
(2) we replace in ϕ every location $v \in \mathrm{img}(\mathsf{s})$ by an arbitrary but fixed variable x with $\mathsf{s}(x) = v$ holds;
(3) we replace every location $v \in \mathrm{dom}(\mathsf{t}) \setminus \mathrm{img}(\mathsf{s})$ by a *guarded existential*;
(4) we replace every other location by a *guarded universal*.

(We point out that any occurrence of *nil* in ϕ would not be replaced during step (2)-(4) by the projection operation because it is not a location.) Now we make the above definitions precise. First, we introduce the *projection* of trees and forests (Section 7.2.1). Then, we state the definition of *guarded quantifiers* (in Section 7.2.2); Finally, we introduce the *stack-projection* (in Section 7.2.3).

7.2.1 Tree and Forest Projections. We are now ready to define the forest projection outlined in Section 6.3. We begin with defining the projection of a tree:

Definition 7.16 (Projection of a Tree). *The* projection $\mathsf{project}^{\mathbf{Loc}}(\mathfrak{t})$ *of a* Φ*-tree* $\mathfrak{t}$ *is given by*

$$\mathsf{project}^{\mathbf{Loc}}(\mathfrak{t}) \triangleq (\bigstar \mathsf{allholepreds}(\mathfrak{t})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}).$$

Example 7.17. *Recall from Example 7.2 the* Φ*-tree* $\mathfrak{t}$ *over an SID describing lists of even and odd length. This tree admits the tree projection* $\mathsf{project}^{\mathbf{Loc}}(\mathfrak{t}) = \mathsf{even}(l_2, a) \mathrel{-\!\!\star} \mathsf{even}(l_1, a)$.

Tree projections are sound in the sense that the induced heap of a tree satisfies its tree projection. To prove this result, we need the following variant of modus ponens (cf. [Reynolds 2002]):

Lemma 7.18 (Generalized modus ponens).

$$((\mathsf{pred}_2(\mathbf{x}_2) \star \psi) \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x}_1)) \star (\psi' \mathrel{-\!\!\star} \mathsf{pred}_2(\mathbf{x}_2)) \quad \textit{implies} \quad (\psi \star \psi') \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x}_1).$$

Lemma 7.19 (Soundness of Tree Projections). *Let* $\mathfrak{t}$ *be a* Φ*-tree. Then, we have* $\langle \_, \mathsf{heap}(\mathfrak{t}) \rangle \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathfrak{t})$ *(where* _ *denotes an arbitrary stack).*

Proof. By mathematical induction on the height of $\mathfrak{t}$; see Appendix A.12 for details. □

7.2.2 Guarded Quantifiers. As motivated in Section 6.3.3, we introduce *guarded* quantifiers, which we denote by $\circledS\exists$ and $\circledS\forall$, respectively. Specifically, we consider formulas $\circledS\exists \mathbf{e}.\ (\circledS\forall \mathbf{a}.\ (\phi_{\mathrm{qf}} \star \cdots \star \phi_{\mathrm{qf}}))$, where ϕ_{qf} denotes *quantifier-free* **SL** formulas (cf. Section 3.1). We collect all formulas of the above form in the set $\mathbf{SL}^{\circledS\forall}_{\mathrm{btw}}$. Our guarded quantifiers have the following semantics:

- $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \circledS\exists \langle e_1, \ldots, e_k \rangle . \phi$ iff there exist pairwise different locations
$$v_1, \ldots, v_k \in \mathrm{dom}(\mathfrak{h}) \setminus \mathrm{img}(\mathfrak{s}) \quad \text{such that} \quad \langle \mathfrak{s} \cup \{e_1 \mapsto v_1, \ldots, e_k \mapsto v_k\}, \mathfrak{h} \rangle \models_\Phi \phi.$$
- $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \circledS\forall \langle a_1, \ldots, a_k \rangle . \phi$ iff for all pairwise different locations
$$v_1, \ldots, v_k \in \mathbf{Loc} \setminus (\mathrm{dom}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})), \quad \text{we have} \quad \langle \mathfrak{s} \cup \{a_1 \mapsto v_1, \ldots, a_k \mapsto v_k\}, \mathfrak{h} \rangle \models_\Phi \phi.$$

Notice that our guarded quantifiers differ from the standard ones in three aspects: first, guarded quantifiers cannot be instantiated with locations that are already known, i.e., not with any location that is already in the stack. Second, we require that the quantified locations are pairwise different. Third, our quantifiers are *not* dual, i.e., $\circledS\exists \mathbf{x}.\ \phi$ is *not* equivalent to $\neg \circledS\forall \mathbf{x}.\ \neg\phi$. For guarded states, location terms in a formula can be replaced by a guarded universal quantifier as long as they do not appear in the state in question:

Lemma 7.20. *Let* $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$ *and* ϕ *be a quantifier free* **SL** *formula with* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$*. Moreover, let* $\mathbf{v} \in (\mathbf{Loc} \setminus (\mathrm{dom}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})))^*$ *be a repetition-free sequence of locations. Then, for every set* $\mathbf{a} \triangleq \{a_1, \ldots, a_{|\mathbf{v}|}\}$ *of fresh variables (i.e.,* $\mathbf{a} \cap \mathrm{dom}(\mathfrak{s}) = \emptyset$*), we have* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \circledS\forall \mathbf{a}.\ \phi[\mathbf{v}/\mathbf{a}]$.

Proof. See Appendix A.11. □

Many standard equivalences of separation logic continue to hold for formulas with guarded quantifiers; we list corresponding *rewriting rules* in Fig. 10. These rules establish the *rewriting equivalence* $\equiv$, which preserves logical equivalence—we will only consider formulas up to $\equiv$.

Lemma 7.21 (Soundness of rewriting equivalence). *If* $\phi_1 \equiv \phi_2$ *then* $\phi_1 \models_\Phi \phi_2$.

7.2.3 Stack-Projection. We now abstract from locations in projections (cf. Section 6.3.1), replacing every location l in the projection of a forest $\mathfrak{f}$ by a variable: a stack variable, if l is in the image of the stack, an existentially-quantified one if $l \in \mathrm{dom}(\mathfrak{f})$, and a universally-quantified one otherwise.

$$\frac{\phi_1 \equiv \phi_2}{\phi_1 \star \psi \equiv \phi_2 \star \psi}\ (\text{mono}) \quad \frac{}{\phi_1 \star \mathbf{emp} \equiv \phi_1}\ (\text{emp}) \quad \frac{\phi_1 \equiv \phi_2}{\phi_2 \mathrel{-\!\!\star} \psi \equiv \phi_1 \mathrel{-\!\!\star} \psi}\ (\text{anti}) \quad \frac{}{\phi_1 \equiv \phi_1}\ (\text{id})$$

$$\frac{}{\phi_1 \star (\phi_2 \star \phi_3) \equiv (\phi_1 \star \phi_2) \star \phi_3}\ (\text{assoc}) \quad \frac{}{\phi_1 \star \phi_2 \equiv \phi_2 \star \phi_1}\ (\text{comm}) \quad \frac{\phi_1 \equiv \phi_2}{\phi_2 \equiv \phi_1}\ (\text{sym})$$

$$\frac{\mathbb{Q} \in \{\forall, \exists\} \qquad z \notin \mathsf{vars}(\phi)}{\mathbb{Q}y.\, \phi \equiv \mathbb{Q}z.\, \phi[y/z]}\ (\text{ren}) \quad \frac{\phi_1 \equiv \phi_2}{\exists \mathbf{y}.\, \phi_1 \equiv \exists \mathbf{y}.\, \phi_2}\ (\exists\text{-intro}) \quad \frac{\phi_1 \equiv \phi_2}{\forall \mathbf{y}.\, \phi_1 \equiv \forall \mathbf{y}.\, \phi_2}\ (\forall\text{-intro})$$

$$\frac{\phi_1 \equiv \phi_3 \qquad \phi_3 \equiv \phi_2}{\phi_1 \equiv \phi_2}\ (\text{trans}) \quad \frac{\mathbb{Q} \in \{\forall, \exists\} \qquad z \notin \mathsf{vars}(\phi)}{\mathbb{Q}z.\, \phi \equiv \phi}\ (\text{drop})$$

Fig. 10. A set of rules for rewriting $\mathbf{SL}^{\exists\forall}_{\mathrm{btw}}$ formulas into equivalent formulas.

Aliasing and Variable Order. In case of aliasing, i.e., if there are multiple variables that are mapped to the same location l, there are multiple choices for replacing l by a stack variable x with $\mathfrak{s}(x) = l$. This has the consequence that the projection would not be unique. In order to avoid this problem, we assume an arbitrary, but fixed, linear ordering of the variables **Var**. We then choose the variable among all the aliases of a variable that is maximal according to this variable ordering. Formally:

Definition 7.22 (stack-choice function $\mathfrak{s}^{-1}_{\max}$). *Let $\mathfrak{s}$ be a stack. Then, the stack-choice function of $\mathfrak{s}$ maps a location $l \in \mathrm{img}(\mathfrak{s})$ to $\mathfrak{s}^{-1}_{\max}(l) = \max\{x \in \mathrm{dom}(\mathfrak{s}) \mid \mathfrak{s}(x) = l\}$.*

Quantified Variables. We (mostly) maintain the convention that we denote (guarded) universally resp. existentially quantified variables by $a_1, a_2, \ldots$ resp. $e_1, e_2, \ldots$. We will always assume that $\{a_1, a_2, \ldots\} \cap \{e_1, e_2, \ldots\} = \emptyset$ and that $\mathrm{dom}(\mathfrak{s}) \cap (\{a_1, a_2, \ldots\} \cup \{e_1, e_2, \ldots\}) = \emptyset$ for any stack $\mathfrak{s}$. We are now ready to give the main definition of this subsection:

Definition 7.23 (Stack-projection). *Let $\mathfrak{f} = \{\mathsf{t}_1, \ldots, \mathsf{t}_k\}$ be a Φ-forest, $\mathfrak{s}$ be a stack, and*

- *let $\phi = \bigstar_{1 \le i \le k} \mathsf{project}^{\mathbf{Loc}}(\mathsf{t}_i)$ be the projection of trees of $\mathfrak{f}$ conjoined by $\star$,*
- *let $\mathbf{w} = \mathsf{locs}(\phi) \cap (\mathrm{dom}(\mathfrak{f}) \setminus \mathrm{img}(\mathfrak{s}))$ be some (arbitrarily ordered) sequence of locations that occur in the formula ϕ and are allocated in $\mathsf{heap}(\mathfrak{f})$ but are not the value of any stack variable,*
- *and let $\mathbf{v} = \mathsf{locs}(\phi) \setminus (\mathrm{img}(\mathfrak{s}) \cup \mathrm{dom}(\mathfrak{f}))$ be some (arbitrarily ordered) sequence of locations that occur in the formula ϕ and are neither allocated nor the value of any stack variable.*

Then, we define the stack-projection *of $\mathfrak{s}$ and $\mathfrak{f}$ as*

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \ \triangleq\ \exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi[\mathrm{dom}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{v} \cdot \mathbf{w} / \mathrm{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{a} \cdot \mathbf{e}],$$

where $\mathbf{e} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{w}|} \rangle$ and $\mathbf{a} \triangleq \langle a_1, a_2, \ldots, a_{|\mathbf{v}|} \rangle$ denote disjoint sets of fresh variables.

The stack-projection is well-defined because $\mathrm{dom}(\mathfrak{s}^{-1}_{\max})$, $\mathbf{w}$ and $\mathbf{v}$ form a partitioning of $\mathsf{locs}(\phi)$. Notice that the stack-projection is *unique* (w.r.t. the rewriting equivalence $\equiv$ defined in Fig. 10): while the stack-projection involves picking an (arbitrary) order on the trees $\mathsf{t}_1, \ldots, \mathsf{t}_k$ and a choice of the fresh variables $\mathbf{e}$ and $\mathbf{a}$, this does not matter because of the commutativity and associativity of $\star$ and the possibility to rename quantified variables, which is allowed for by the rules of the rewriting equivalence $\equiv$.

Example 7.24 (Stack-projection). *We consider three examples of stack-projections:*

(1) *Let t be the Φ-tree from Example 7.2. Then, for $\mathfrak{f} = \{\mathsf{t}\}$ and $\mathfrak{s} = \{x_1 \mapsto l_1, x_2 \mapsto l_2\}$, we have*

$$\mathsf{heap}(\mathfrak{f}) = \{l_1 \mapsto b, b \mapsto l_2\} \quad \text{and} \quad \mathsf{project}^{\mathbf{Loc}}(\mathfrak{f}) = \forall a_1.\, \mathsf{even}(l_2, a_1) \mathrel{-\!\!\star} \mathsf{odd}(l_1, a_1).$$

As all locations in this formula are in the image of the stack, we have

$$\mathsf{project}(\mathfrak{s},\mathfrak{f}) = \mathsf{project}^{\mathbf{Loc}}(\mathfrak{f})[\mathrm{dom}(\mathfrak{s}_{\max}^{-1})/\mathrm{img}(\mathfrak{s}_{\max}^{-1})] = \forall a_1.\, \mathsf{even}(x_2,a_1) \mathrel{-\!\!\star} \mathsf{odd}(x_1,a_1).$$

(2) *Let* $\langle \mathfrak{s},\mathfrak{h}\rangle$ *be the model and let* $\mathsf{t}_1,\mathsf{t}_2,\mathsf{t}_3$ *be the* Φ*-trees from Example 6.7. Then,*

$$\mathsf{project}(\mathfrak{s},\{\mathsf{t}_1,\mathsf{t}_3\}) = \exists a.\, (\mathsf{lseg}(y,a) \mathrel{-\!\!\star} \mathsf{cyclic}(x,y,z)) \star \mathsf{lseg}(z,a), \text{ and}$$
$$\mathsf{project}(\mathfrak{s},\{\mathsf{t}_2\}) = \forall a'.\, \mathsf{lseg}(z,a') \mathrel{-\!\!\star} \mathsf{lseg}(y,a').$$

(3) *Let* $\mathsf{t}_1,\mathsf{t}_2,\mathsf{t}_3,\mathsf{t}_4$ *be the* Φ*-trees from Example 6.3 for the state* $\langle \mathfrak{s},\mathfrak{h}\rangle$ *of Example 6.1. Then,*

$$\mathsf{project}(\mathfrak{s},\{\mathsf{t}_1,\mathsf{t}_3,\mathsf{t}_4\}) = \exists r.\, (\mathsf{tll}(a,b,c) \mathrel{-\!\!\star} \mathsf{tll}(x,y,z)) \star (b \mapsto \langle nil, nil, c\rangle) \star (c \mapsto \langle nil, nil, r\rangle), \text{ and}$$
$$\mathsf{project}(\mathfrak{s},\{\mathsf{t}_2\}) = \forall r'.\, ((b \mapsto \langle nil, nil, c\rangle) \star (c \mapsto \langle nil, nil, r'\rangle)) \mathrel{-\!\!\star} \mathsf{tll}(a,b,c).$$

In each of the above examples, we observe that $\langle \mathfrak{s}, \mathrm{heap}(\mathfrak{f})\rangle \models_\Phi \mathsf{project}(\mathfrak{s},\mathfrak{f})$. This is not a coincidence as eliminating locations preserves the soundness of forest projections:

Lemma 7.25 (Soundness of stack-projection). *Let* $\langle \mathfrak{s},\mathfrak{h}\rangle \in \mathbf{GStates}$. *Moreover, let* $\mathfrak{f}$ *be a* Φ*-forest with* $\mathrm{heap}(\mathfrak{f}) = \mathfrak{h}$. *Then, we have* $\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi \mathsf{project}(\mathfrak{s},\mathfrak{f})$.

Proof. See Appendix A.13. □

Example 7.26 (Why we need guarded quantifiers). *We now have the machinery available to discuss why guarded quantifiers are needed. To this end, let us revisit the motivating example in Section 6.3.3: We consider the state* $\langle \mathfrak{s},\mathfrak{h}\rangle$*, given by the stack* $\mathfrak{s} = \{x \mapsto 1\}$ *and the heap* $\mathfrak{h} = \{1 \mapsto 2, 2 \mapsto nil\}$*, and the SID* Φ *given by the following predicates:*

$$p(x,a,b) \iff \exists y.\, (x \mapsto y) \star q(y,a) \star x \neq a \star a \neq b \qquad q(y,a) \iff (y \mapsto nil) \star y \neq a$$

We further consider the unfolding tree t *consisting of the rule instance* $p(1,4,5) \Leftarrow (1 \mapsto 2) \star q(2,4) \star 1 \neq 4 \star 4 \neq 5$ *at the root with a single child for the rule instance* $q(2,4) \Leftarrow (2 \mapsto nil) \star 2 \neq 4$*. Hence, we have* $\mathrm{rootpred}(\mathsf{t}) = p(1,4,5)$, $\mathsf{allholepreds}(\mathsf{t}) = \emptyset$*, and* $\mathsf{locs}(p(1,4,5)) \setminus (\mathrm{dom}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})) = \{4,5\}$*. By Definition 7.16, we then obtain the tree projection*

$$\begin{aligned}\mathsf{project}(\mathfrak{s},\mathsf{t}) &= \forall a,b.\, \big((\bigstar \mathsf{allholepreds}(p(1,a,b))) \mathrel{-\!\!\star} \mathrm{rootpred}(\mathsf{t})\big)[\langle 1,4,5\rangle / \langle x,a,b\rangle] \\ &= \forall a,b.\, \mathbf{emp} \mathrel{-\!\!\star} p(x,4,5)[\langle 4,5\rangle / \langle a,b\rangle] \;=\; \forall a,b.\, p(x,a,b).\end{aligned}$$

By Lemma 7.19, we have $\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi \forall a,b.\, p(x,a,b)$*. In particular, the semantics of* $\forall$ *guarantees that* a *and* b *refer to distinct locations that are neither allocated and nor the value of any stack variable. This is crucial to ensure soundness of the stack-projection: if we would use a standard universal quantifier instead of a guarded one,* $\langle \mathfrak{s},\mathfrak{h}\rangle$ *would not be a model of* $\mathsf{project}^{\mathbf{Loc}}(\mathsf{t})$ *as neither* $\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi p(x,1,5)$*,* $\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi p(x,2,5)$ *nor* $\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi p(x,5,5)$ *hold.*

7.3 Composing Projections

7.3.1 Motivation. Recall from Section 6.1.3 that our goal is the definition of a *composition* operator for the projections of forests. This operation should collect exactly those projections of forests $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$ (see Definition 7.15) that can be derived from $\mathsf{project}(\mathfrak{s},\mathfrak{f}_1)$ and $\mathsf{project}(\mathfrak{s},\mathfrak{f}_2)$, i.e.,

$$\mathsf{project}(\mathfrak{s},\mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s},\mathfrak{f}_2) \stackrel{?}{=} \{\mathsf{project}(\mathfrak{s},\mathfrak{f}) \mid \mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2\}\,.$$

Put differently, we are looking for an operation $\bullet_{\mathbf{P}}$ such that $\mathsf{project}(\mathfrak{s},\cdot)$ is a homomorphism from the set of Φ-forests and $\bullet_{\mathbf{F}}$ to the set of projections and $\bullet_{\mathbf{P}}$.

How can we define such an operation $\bullet_{\mathbf{P}}$? Intuitively, we need to conjoin the projections via $\star$ in order to simulate the operation $\mathfrak{f}_1 \uplus \mathfrak{f}_2$, and apply the generalized modus ponens rule (see Lemma 7.18) in order to simulate

the operation ▸ on trees. There is, however, one complication: our forest projections contain quantifiers. In particular, $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \star \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ is of the form $(\exists \mathbf{e_1}.\, \forall \mathbf{a_1}.\, \phi_1) \star (\exists \mathbf{e_2}.\, \forall \mathbf{a_2}.\, \phi_2)$, whereas $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1 \uplus \mathfrak{f}_2)$ is of the form $\exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi$, where ϕ_1, ϕ_2, and ϕ do not contain guarded quantifiers. In other words, $\bullet_{\mathbf{P}}$ has to push the guarded quantifiers to the front before the modus ponens rule can be applied.

7.3.2 Definition of the Composition Operation. We will define $\bullet_{\mathbf{P}}$ in terms of two operations: (1) An operator $\bar{\star}$ that captures all sound ways to move the guarded quantifiers to the front of the formula $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \star \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ (i.e., "re-scopes the guarded quantifiers"). (2) A derivation operator ▷ that rewrites formulas based on the generalized modus ponens rule (Lemma 7.18).

Definition 7.27 (Re-scoping). *We say χ is a* re-scoping *of $\exists \mathbf{e_1}.\, \forall \mathbf{a_1}.\, \phi_1$ and $\exists \mathbf{e_2}.\, \forall \mathbf{a_2}.\, \phi_2$, in signs $\chi \in (\exists \mathbf{e_1}.\, \forall \mathbf{a_1}.\, \phi_1) \bar{\star} (\exists \mathbf{e_2}.\, \forall \mathbf{a_2}.\, \phi_2)$, if there are repetition-free sequences of variables $\mathbf{a}$, $\mathbf{d_i}$ and $\mathbf{u_i} \subseteq \mathbf{a} \cup \mathbf{d_{3-i}}$, for $i = 1, 2$, such that (1) $\mathbf{a}$, $\mathbf{d_1}$ and $\mathbf{d_2}$ are pairwise disjoint, and*
(2) $\chi \equiv \exists \mathbf{d_1} \cdot \mathbf{d_2}.\, \forall \mathbf{a}.\, \phi_1[\mathbf{e_1} \cdot \mathbf{a_1}/\mathbf{d_1} \cdot \mathbf{u_1}] \star \phi_2[\mathbf{e_2} \cdot \mathbf{a_2}/\mathbf{d_2} \cdot \mathbf{u_2}]$.

The re-scoping operation is sound with regard to the semantics of separation logic:

Lemma 7.28 (Soundness of Re-scoping). *Let $\exists \mathbf{e_1}.\, \forall \mathbf{a_1}.\, \phi_1$ and $\exists \mathbf{e_2}.\, \forall \mathbf{a_2}.\, \phi_2$ be some formulas whose predicates are defined by some SID Φ. Then,*

$$\chi \in (\exists \mathbf{e_1}.\, \forall \mathbf{a_1}.\, \phi_1) \bar{\star} (\exists \mathbf{e_2}.\, \forall \mathbf{a_2}.\, \phi_2) \quad \textit{implies} \quad (\exists \mathbf{e_1}.\, \forall \mathbf{a_1}.\, \phi_1) \star (\exists \mathbf{e_2}.\, \forall \mathbf{a_2}.\, \phi_2) \models_\Phi \chi.$$

Proof. Follows directly from the semantics of the guarded quantifiers $\exists$ and $\forall$. □

Definition 7.29 (Derivability). *We say χ can be* derived *from $\exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi$, in signs $\exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi \triangleright \chi$, if χ can be obtained from $\exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi$ by applying Lemma 7.18 and the rewriting equivalence $\equiv$ (see Fig. 10), formally, if there are predicates $\mathsf{pred}_1(\mathbf{x_1}), \mathsf{pred}_2(\mathbf{x_2})$, and formulas ψ, ψ', ζ such that*
(1) $\phi \equiv (\mathsf{pred}_2(\mathbf{x_2}) \star \psi) \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1}))) \star (\psi' \mathrel{-\!\!\star} \mathsf{pred}_2(\mathbf{x_2})) \star \zeta$, *and*
(2) $\chi \equiv \exists \mathbf{e}.\, \forall \mathbf{a}.\, (\psi \star \psi') \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1}) \star \zeta$.

The derivability relation is sound with regard to the semantics of separation logic:

Lemma 7.30 (Soundness of Derivability). *Let $\exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi_1$ be some formula whose predicates are defined by some SID Φ. Then, $\exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi_1 \triangleright \chi$ implies $\exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi_1 \models_\Phi \chi$.*

Proof. Follows directly from the soundness of the generalized modus ponens rule (see Lemma 7.18) and the soundness of the rewriting equivalence $\equiv$. □

We now define composition based on the re-scoping and derivation operations:

Definition 7.31 (Composition Operation). *We define the* composition *of ϕ_1 and ϕ_2 by*

$$\phi_1 \bullet_{\mathbf{P}} \phi_2 \triangleq \{\phi \mid \zeta \triangleright^* \phi \textit{ for some } \zeta \in \phi_1 \bar{\star} \phi_2\}.$$

Corollary 7.32 (Soundness of $\bullet_{\mathbf{P}}$). *$\phi \in \phi_1 \bullet_{\mathbf{P}} \phi_2$ implies $\phi_1 \star \phi_2 \models_\Phi \phi$.*

Proof. Follows immediately from Lemmas 7.28 and 7.30. □

Example 7.33.
- *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 $\phi_1 \star \phi_2 \triangleright \mathbf{emp} \mathrel{-\!\!\star} \mathsf{ls}(x_1, x_3)$. Hence, $(\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ls}(x_1, x_3)) \in \phi_1 \bullet_{\mathbf{P}} \phi_2$.*
- *For $\phi_1 = \forall a.\, \mathsf{ls}(x_2, a) \mathrel{-\!\!\star} \mathsf{ls}(x_1, a)$ and $\phi_2 = \forall b.\, \mathsf{ls}(x_3, b) \mathrel{-\!\!\star} \mathsf{ls}(x_2, b)$, we have $\forall c.\, (\mathsf{ls}(x_2, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)) \star (\mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_2, c)) \in \phi_1 \bar{\star} \phi_2$. With $\forall c.\, (\mathsf{ls}(x_2, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)) \star (\mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_2, c)) \triangleright \forall c.\, (\mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c))$, we have $\forall c.\, (\mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)) \in \phi_1 \bullet_{\mathbf{P}} \phi_2$.*

Let us also revisit our informal exposition in Example 6.8 and make it precise:

Example 7.34 (Composition Operation on Projections).

- *Let* $\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3$ *be the* Φ*-trees from Example 6.7. We set* $\mathsf{f}_1 = \{\mathsf{t}_1, \mathsf{t}_3\}$ *and* $\mathsf{f}_1 = \{\mathsf{t}_2\}$*. We then have:*

  $\mathsf{project}(\mathfrak{s}, \mathsf{f}_1) = \exists a.\, (\mathsf{lseg}(y, a) \mathrel{-\!\!\star} \mathsf{cyclic}(x, y, z)) \star \mathsf{lseg}(z, a)$, *and*

  $\mathsf{project}(\mathfrak{s}, \mathsf{f}_2) = \forall a'.\, \mathsf{lseg}(z, a') \mathrel{-\!\!\star} \mathsf{lseg}(y, a')$*. Then,*

  $\exists a.\, (\mathsf{lseg}(y, a) \mathrel{-\!\!\star} \mathsf{cyclic}(x, y, z)) \star \mathsf{lseg}(z, a) \star (\mathsf{lseg}(z, a) \mathrel{-\!\!\star} \mathsf{lseg}(y, a))$

  $\quad \in \mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \bar{\star} \mathsf{project}(\mathfrak{s}, \mathsf{f}_2)$ *and further*

  $\exists a.\, (\mathsf{lseg}(y, a) \mathrel{-\!\!\star} \mathsf{cyclic}(x, y, z)) \star \mathsf{lseg}(z, a) \star (\mathsf{lseg}(z, a) \mathrel{-\!\!\star} \mathsf{lseg}(y, a)) \rhd^* \mathsf{cyclic}(x, y, z)$.

  Hence, we have $\mathsf{cyclic}(x, y, z) \in \mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathsf{f}_2)$.

- *Let* $\mathsf{t}_1, \mathsf{t}_2, \mathsf{t}_3, \mathsf{t}_4$ *be the* Φ*-trees from Example 6.3. We set* $\mathsf{f}_1 = \{\mathsf{t}_1, \mathsf{t}_3, \mathsf{t}_4\}$ *and* $\mathsf{f}_2 = \{\mathsf{t}_2\}$*. We have*

  $\mathsf{project}(\mathfrak{s}, \mathsf{f}_1) = \exists r.\, (\mathsf{tll}(a, b, c) \mathrel{-\!\!\star} \mathsf{tll}(x, y, z)) \star (b \mapsto \langle nil, nil, c\rangle) \star (c \mapsto \langle nil, nil, r\rangle)$, *and*

  $\mathsf{project}(\mathfrak{s}, \mathsf{f}_2) = \forall r'.\, ((b \mapsto \langle nil, nil, c\rangle) \star (c \mapsto \langle nil, nil, r'\rangle)) \mathrel{-\!\!\star} \mathsf{tll}(a, b, c)$*. Then,*

  $\exists r.\, (\mathsf{tll}(a, b, c) \mathrel{-\!\!\star} \mathsf{tll}(x, y, z)) \star (b \mapsto \langle nil, nil, c\rangle) \star (c \mapsto \langle nil, nil, r\rangle) \star$

  $\quad (((b \mapsto \langle nil, nil, c\rangle) \star (c \mapsto \langle nil, nil, r\rangle)) \mathrel{-\!\!\star} \mathsf{tll}(a, b, c)) \in \mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \bar{\star} \mathsf{project}(\mathfrak{s}, \mathsf{f}_2)$*. Further,*

  $\exists r.\, (\mathsf{tll}(a, b, c) \mathrel{-\!\!\star} \mathsf{tll}(x, y, z)) \star (b \mapsto \langle nil, nil, c\rangle) \star (c \mapsto \langle nil, nil, r\rangle) \star$

  $\quad (((b \mapsto \langle nil, nil, c\rangle) \star (c \mapsto \langle nil, nil, r\rangle)) \mathrel{-\!\!\star} \mathsf{tll}(a, b, c)) \rhd^* \mathsf{tll}(x, y, z)$.

  Hence, we have $\mathsf{tll}(x, y, z) \in \mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathsf{f}_2)$.

7.3.3 Relating the Composition of Forests and of Projections. Recall from Section 7.3.1 our design goal that the projection function $\mathsf{project}(\mathfrak{s}, \cdot)$ should be a homomorphism from forests and forest composition $\bullet_{\mathbf{F}}$ (Definition 7.15) to projections and projection composition $\bullet_{\mathbf{P}}$ (Definition 7.31), i.e.,

$$\mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathsf{f}_2) \stackrel{?}{=} \{\mathsf{project}(\mathfrak{s}, \mathsf{f}) \mid \mathsf{f} \in \mathsf{f}_1 \bullet_{\mathbf{F}} \mathsf{f}_2\}\,.$$

Indeed, in one direction our composition operation achieves this:

Lemma 7.35. *Let* $\mathfrak{s}$ *be a stack and let* $\mathsf{f}_1, \mathsf{f}_2$ *be* Φ*-forests such that* $\mathsf{f}_1 \uplus \mathsf{f}_2 \neq \bot$*. Then,*

$$\mathsf{f} \in \mathsf{f}_1 \bullet_{\mathbf{F}} \mathsf{f}_2 \quad \textit{implies} \quad \mathsf{project}(\mathfrak{s}, \mathsf{f}) \in \mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathsf{f}_2).$$

Proof. See Appendix A.14. □

Unfortunately, as demonstrated below, the homomorphism breaks in the other direction:

Example 7.36 (Projection is not homomorphic). *Consider the* Φ*-forests* $\mathsf{f}_1 = \{\mathsf{t}_1\}$ *and* $\mathsf{f}_2 = \{\mathsf{t}_2\}$ *and the stack* $\mathfrak{s} \triangleq \{x_1 \mapsto l_1, x_2 \mapsto l_2, x_3 \mapsto l_3\}$*, where*

$$\mathsf{t}_1 = \{l_1 \mapsto \langle \emptyset, (\mathsf{odd}(l_1, m_1) \Leftarrow (l_1 \mapsto l_2) \star \mathsf{even}(l_2, m_1))\rangle\}\,, \text{ and}$$
$$\mathsf{t}_2 = \{l_2 \mapsto \langle \emptyset, (\mathsf{even}(l_2, m_2) \Leftarrow (l_2 \mapsto l_3) \star \mathsf{odd}(l_3, m_2))\rangle\}\,.$$

The corresponding projections are

$$\mathsf{project}(\mathfrak{s}, \mathsf{f}_1) = \forall a.\, \mathsf{even}(x_2, a) \mathrel{-\!\!\star} \mathsf{odd}(x_1, a) \textit{ and } \mathsf{project}(\mathfrak{s}, \mathsf{f}_2) = \forall a.\, \mathsf{odd}(x_3, a) \mathrel{-\!\!\star} \mathsf{even}(x_2, a).$$

Moreover, we have $\forall a.\, \mathsf{odd}(x_3, a) \mathrel{-\!\!\star} \mathsf{odd}(x_1, a) \in \mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathsf{f}_2)$.

However, since different locations, namely m_1 *and* m_2*, are unused in the two forests, there is only one forest in* $\mathsf{f}_1 \bullet_{\mathbf{F}} \mathsf{f}_2$*:* $\{\mathsf{t}_1, \mathsf{t}_2\}$*. It is not possible to merge the trees, because the hole predicate of the first tree,* $\mathsf{even}(l_2, m_1)$*, is different from the root of the second tree,* $\mathsf{even}(l_2, m_2)$*. In particular, there does* not *exist a forest* f *with* $\mathsf{f} \in \mathsf{f}_1 \bullet_{\mathbf{F}} \mathsf{f}_2$ *and* $\mathsf{project}(\mathfrak{s}, \mathsf{f}) \equiv \forall a.\, \mathsf{odd}(x_3, a) \mathrel{-\!\!\star} \mathsf{odd}(x_1, a)$.

The essence of Example 7.36 is that while $\bullet_{\mathbf{P}}$ allows renaming quantified universals, $\bullet_{\mathbf{F}}$ does not allow renaming locations, breaking the homomorphism. To get a correspondence between the two notions of composition, we therefore allow renaming all locations that do *not* occur as the value of any stack variable. We capture this in the notion of $\mathfrak{s}$-*equivalence*:

Definition 7.37 ($\mathfrak{s}$-equivalence). *Two* Φ*-forests* $\mathfrak{f}_1, \mathfrak{f}_2$ *are* $\mathfrak{s}$-equivalent, *denoted* $\mathfrak{f}_1 \equiv_{\mathfrak{s}} \mathfrak{f}_2$, *iff there is a bijective function* $\sigma\colon \mathbf{Loc} \to \mathbf{Loc}$ *such that* $\sigma(l) = l$ *for all* $l \in \mathrm{img}(\mathfrak{s})$, *and* $\sigma(\mathfrak{f}_1) = \mathfrak{f}_2$, *where*

- $\sigma(\{\mathtt{t}_1, \ldots, \mathtt{t}_k\}) \triangleq \{\sigma(\mathtt{t}_1), \ldots, \sigma(\mathtt{t}_k)\}$,
- $\sigma(\mathtt{t}) \triangleq \left\{\sigma(l) \mapsto \left\langle \sigma(\mathsf{succ}_{\mathtt{t}_1}(l)), \mathsf{rule}_{\mathtt{t}_1}(l)[\mathrm{dom}(\sigma)/\mathrm{img}(\sigma)] \mid l \in \mathrm{dom}(\mathtt{t}) \right\rangle\right\}$, *and*
- $(\mathsf{pred}(\mathbf{l}) \Leftarrow \phi)[\mathbf{v}/\mathbf{w}] \triangleq \mathsf{pred}(\mathbf{l}[\mathbf{v}/\mathbf{w}]) \Leftarrow \phi[\mathbf{v}/\mathbf{w}]$ *for sequences of locations* $\mathbf{v}$ *and* $\mathbf{w}$.

Note that $\mathfrak{f}_1 \equiv_{\mathfrak{s}} \mathfrak{f}_2$ implies that $\langle \mathfrak{s}, \mathsf{heap}(\mathfrak{f}_1)\rangle$ and $\langle \mathfrak{s}, \mathsf{heap}(\mathfrak{f}_2)\rangle$ are isomorphic. In fact,

Lemma 7.38. *If* $\mathfrak{f}_1$ *and* $\mathfrak{f}_2$ *are* Φ*-forests with* $\mathfrak{f}_1 \equiv_{\mathfrak{s}} \mathfrak{f}_2$, *then* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \equiv \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$.

Proof. Direct from the definition of $\mathfrak{s}$-equivalence and the stack-projection. □

With the definition of $\mathfrak{s}$-equivalence in place, we indeed obtain the desired composition:

Theorem 7.39. *If* $\mathfrak{f}_1, \mathfrak{f}_2$ *be* Φ*-forests with* $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \neq \bot$. *Then,*

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \;=\; \left\{\mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}}) \mid \bar{\mathfrak{f}} \in \bar{\mathfrak{f}}_1 \bullet_{\mathbf{F}} \bar{\mathfrak{f}}_2,\ \bar{\mathfrak{f}}_1 \equiv_{\mathfrak{s}} \mathfrak{f}_1,\ \bar{\mathfrak{f}}_2 \equiv_{\mathfrak{s}} \mathfrak{f}_2\right\}.$$

Proof. See Appendix A.15. □

8 THE TYPE ABSTRACTION

We now formally introduce the abstraction on which our decision procedure for **GSL** will be built. As motivated in Section 6.4, we abstract every (guarded) state to a Φ-*type*, which is a *set* of stack-forest projections. In order to ensure the *finiteness* of the abstraction we need to restrict Φ-types to certain kinds of stack-forest projections. Let us denote by $\mathrm{forests}_\Phi(\mathfrak{h}) \triangleq \{\bar{\mathfrak{f}} \mid \mathsf{heap}(\bar{\mathfrak{f}}) = \mathfrak{h}\}$ the set of all Φ-forests whose induced heap is $\mathfrak{h}$. We will then abstract a state $\langle \mathfrak{s}, \mathfrak{h}\rangle$ to a *subset* of the stack-forest projections whose induced heap is $\mathfrak{h}$, i.e., $\{\mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}}) \mid \bar{\mathfrak{f}} \in \mathrm{forests}_\Phi(\mathfrak{h})\}$.

We call the formulas $\mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}})$ *unfolded symbolic heaps* (USHs) with respect to SID Φ because any such stack-forest projection can be obtained by "partially unfolding" a symbolic heap (which might require adding appropriate (guarded) quantifiers). Intuitively, the USHs satisfied by a state $\langle \mathfrak{s}, \mathfrak{h}\rangle$ capture all ways in which $\langle \mathfrak{s}, \mathfrak{h}\rangle$ relates to the predicates in SID Φ. While the entire set of USHs is finite for every fixed state $\langle \mathfrak{s}, \mathfrak{h}\rangle$, the set of all USHs w.r.t. an SID Φ is infinite in general:

Example 8.1. *Assume the SID* Φ *defines the list-segment predicate* lseg *(see Example 3.3). Moreover, let* $\langle \mathfrak{s}, \mathfrak{h}\rangle$ *be a state with* $|\mathfrak{h}| > n \in \mathbb{N}$ *such that* $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \mathsf{lseg}(x, nil)$. *Then there exists a forest* $\bar{\mathfrak{f}}$ *with* $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathfrak{h}$ *whose projection consists of n components, i.e.,*

$$\mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}}) = \exists y_1, \ldots, y_n.\mathsf{lseg}(y_n, nil) \star (\mathsf{lseg}(y_n, nil) \mathrel{-\!\!\star} \mathsf{lseg}(y_{n-1}, nil)) \\ \star \cdots \star (\mathsf{lseg}(y_2, nil) \mathrel{-\!\!\star} \mathsf{lseg}(y_1, nil)) \star (\mathsf{lseg}(y_1, nil) \mathrel{-\!\!\star} \mathsf{lseg}(x, nil)).$$

As there exist such states $\langle \mathfrak{s}, \mathfrak{h}\rangle$ *for arbitrary natural numbers n, there are infinitely many USHs w.r.t.* Φ.

To obtain a *finite* abstraction, we restrict ourselves to *delimited USHs* (DUSHs), in which (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 8.2. *An unfolded symbolic heap* ϕ *is* delimited *iff*

(1) *for all* $\mathsf{pred}(\mathbf{z}) \in \phi$, $\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \in \mathrm{fvars}(\phi)$, *and*

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

The notion of delimited unfolded symbolic heaps is motivated as follows: (1) For every guarded state, the targets of dangling pointers are in the image of the stack. (2) That every variable occurs at most once as the root of a predicate on the left-hand side of a magic wand is a prerequisite for "eliminating" the magic wand through the generalized modus ponens rule.

Example 8.3. *Recall from Example 3.3 the SID* Φ_{tree} *that defines binary trees. We consider the stack* $\mathfrak{s} = \{x \mapsto l_1, y \mapsto l_2, z \mapsto l_3\}$.

(1) *We consider the following trees and corresponding forest-projections:*

$$\mathfrak{t}_1 \triangleq \{l_1 \mapsto \langle \emptyset, \mathsf{tree}(l_1) \Leftarrow (l_1 \mapsto \langle l_2, l_3\rangle) \star \mathsf{tree}(l_2) \star \mathsf{tree}(l_3)\rangle\}$$
$$\mathfrak{t}_2 \triangleq \{l_2 \mapsto \langle \emptyset, \mathsf{tree}(l_2) \Leftarrow l_2 \mapsto \langle \mathit{nil}, \mathit{nil}\rangle\rangle\}, \qquad \mathfrak{t}_3 \triangleq \{l_3 \mapsto \langle \emptyset, \mathsf{tree}(l_3) \Leftarrow l_3 \mapsto \langle \mathit{nil}, \mathit{nil}\rangle\rangle\},$$
$$\mathsf{project}(\mathfrak{s}, \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\}) = ((\mathsf{tree}(y) \star \mathsf{tree}(z)) \mathrel{-\!\!\star} \mathsf{tree}(x)) \star \mathsf{tree}(y) \star \mathsf{tree}(z)$$
$$\bar{\mathfrak{t}} \triangleq \{l_1 \mapsto \langle \langle l_2, l_3\rangle, \mathfrak{t}_1(l_1)\rangle\} \cup \mathfrak{t}_2 \cup \mathfrak{t}_3 \qquad \mathsf{project}(\mathfrak{s}, \{\bar{\mathfrak{t}}\}) = \mathsf{tree}(x)$$

We observe that $\mathsf{project}(\mathfrak{s}, \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\})$ *and* $\mathsf{project}(\mathfrak{s}, \{\bar{\mathfrak{t}}\})$ *are delimited.* $\mathsf{project}(\mathfrak{s}, \{\bar{\mathfrak{t}}\})$ *can be obtained from* $\mathsf{project}(\mathfrak{s}, \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\})$ *by two applications of modus ponens.*

(2) *We consider the following trees and the projection of the corresponding forest:*

$$\mathfrak{t}_1 \triangleq \{l_1 \mapsto \langle \emptyset, \mathsf{tree}(l_1) \Leftarrow (l_1 \mapsto \langle l_2, l_2\rangle) \star \mathsf{tree}(l_2) \star \mathsf{tree}(l_2)\rangle\}$$
$$\mathfrak{t}_2 \triangleq \{l_2 \mapsto \langle \emptyset, \mathsf{tree}(l_2) \Leftarrow l_2 \mapsto \langle \mathit{nil}, \mathit{nil}\rangle\rangle\}$$
$$\mathsf{project}(\mathfrak{s}, \{\mathfrak{t}_1, \mathfrak{t}_2\}) = ((\mathsf{tree}(y) \star \mathsf{tree}(y)) \mathrel{-\!\!\star} \mathsf{tree}(x)) \star \mathsf{tree}(y)$$

We note that $\mathsf{project}(\mathfrak{s}, \{\mathfrak{t}_1, \mathfrak{t}_2\})$ *is not delimited because the variable y appears twice on the LHS of a magic wand; at most one occurrence of y can be eliminated using modus ponens.*

We collect the set of all delimited unfolded symbolic heaps (DUSH) over the SID Φ in

$$\mathbf{DUSH}_\Phi \triangleq \{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \mathfrak{s} \in \mathbf{Stacks}, \mathfrak{f} \text{ is a } \Phi\text{-forest}, \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \text{ is delimited}\}.$$

We are now ready to introduce the type abstraction. Given a state $\langle \mathfrak{s}, \mathfrak{h}\rangle$ and an SID Φ, we call the set of all projections of Φ-forests capturing the heap $\mathfrak{h}$ in the DUSH fragment the Φ-*type* of $\langle \mathfrak{s}, \mathfrak{h}\rangle$:

Definition 8.4 (Φ-Type). *The* Φ-type *(type for short) of a state* $\langle \mathfrak{s}, \mathfrak{h}\rangle$ *and an SID* Φ *is given by*

$$\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \triangleq \{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})\} \cap \mathbf{DUSH}_\Phi.$$

In the remainder of this section, we discuss the main results and building blocks required for turning the type abstraction into a decision procedure for guarded separation logic (**GSL**).

8.1 Understanding Satisfiability as Computing Types

The main idea underlying our decision procedure is to reduce the satisfiability problem for **GSL**—"given a **GSL** formula ϕ, does ϕ have a model $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi$?"—to the question of whether some type $\mathcal{T}$ can be computed from a model of ϕ, i.e., $\mathcal{T} = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ should hold for some $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi$; the set of all such types will be formally defined further below.

8.1.1 Aliasing Constraints. To conveniently reason about sets of types, we require that types in the same set have the same free variables and the same aliases, i.e., we will group types by *aliasing constraint*—an equivalence relation $\mathbf{ac} \subseteq \mathbf{Var} \times \mathbf{Var}$ representing all aliases under consideration. More formally, every stack $\mathfrak{s}$ induces an aliasing constraint $\mathsf{aliasing}(\mathfrak{s})$ given by

$$\mathsf{aliasing}(\mathfrak{s}) \triangleq \{(x, y) \mid x, y \in \mathrm{dom}(\mathfrak{s}) \text{ and } \mathfrak{s}(x) = \mathfrak{s}(y)\}\,.$$

We denote the *domain* of an aliasing constraint $\mathbf{ac}$ by $\mathrm{dom}(\mathbf{ac}) \triangleq \{x \mid (x, x) \in \mathbf{ac}\}$. Furthermore, we write $\mathbf{ac}(x)$ for the set of aliases of x given by the aliasing constraint $\mathbf{ac}$, i.e., the equivalence class $\mathbf{ac}(x) \triangleq \{y \mid (x, y) \in \mathbf{ac}\}$ of $\mathbf{ac}$ that contains x. To obtain a canonical formalization, we frequently represent the equivalence class of x by its largest[8] member; formally, $[x]^{\mathbf{ac}}_{=} \triangleq \max \mathbf{ac}(x)$.

8.1.2 From **GSL** *satisfiability to types.* As outlined at the beginning of Section 8.1, our decision procedure will be based on computing sets of types of the following form:

Definition 8.5 (**ac**-Types). *Let* $\mathbf{ac}$ *be an aliasing constraint (cf. Section 8.1.1). Then the set* $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ *of* **ac**-types *of* **GSL** *formula* ϕ *is defined as*

$$\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi) \triangleq \left\{\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) \mid \mathfrak{s} \in \mathbf{Stacks},\ \mathfrak{h} \in \mathbf{Heaps},\ \mathsf{aliasing}(\mathfrak{s}) = \mathbf{ac},\ \langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \phi\right\}.$$

By the above definition, a **GSL** formula ϕ with at least one non-empty set of **ac**-types is satisfiable: some type coincides with $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$, where $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \phi$. Conversely, if ϕ is satisfiable, then there exists a model $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \phi$ and the set $\mathbf{Types}^{\mathsf{aliasing}(\mathfrak{s})}_{\Phi}(\phi)$ is non-empty. In summary:

$$\phi \text{ is satisfiable} \quad \text{iff} \quad \exists \mathbf{ac}.\ \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi) \neq \emptyset.$$

On a first glance, finding a suitable aliasing constraint $\mathbf{ac}$ and proving non-emptiness of $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ might appear as difficult as finding a state $\langle \mathfrak{s}, \mathfrak{h}\rangle$ such that $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \phi$ holds due to three concerns:

(1) There are, in general, both infinitely many aliasing constraints and infinitely many Φ-types, because the size of stacks—and thus the number of free variables to consider—is unbounded.
(2) Even if the set $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ is finite, effectively computing it is non-trivial.
(3) Deciding whether a type $\mathcal{T}$ belongs to $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ is non-trivial: assume that $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \phi$, $\langle \mathfrak{s}', \mathfrak{h}'\rangle \not\models_{\Phi} \phi$, and both states yield the same type, i.e., $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) = \mathsf{type}_{\Phi}(\mathfrak{s}', \mathfrak{h}')$. Determining that $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ would then require us to know that $\mathcal{T}$ can be computed from a specific state, namely $\langle \mathfrak{s}, \mathfrak{h}\rangle$.

As informally motivated in Section 6, our type abstraction can deal with each of the above concerns; we provide the formal details addressing each concern in the remainder of this section:

Regarding (1), we discuss in Section 8.2 how both aliasing constraints and types can safely be restricted to finite subsets. Determining whether $\exists \mathbf{ac}.\ \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi) \neq \emptyset$ holds thus amounts to computing finitely many finite sets. This corresponds to achieving *finiteness* in Section 6.

Regarding (2), we introduce operations for effectively computing Φ-types from existing ones in Sections 8.3 and 8.4; they will be the building blocks of our decision procedure. This corresponds to achieving *compositionality* in Section 6.

Regarding (3), we show in Section 8.5 that one can decide whether a type $\mathcal{T}$ belongs to $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ *without reverting to any state underlying* $\mathcal{T}$. In particular, we will show that states yielding the same Φ-type satisfy the same **GSL** formulas. This corresponds to achieving *refinement* in Section 6.

[8] w.r.t. the linear ordering over variables we assume throughout this article; notice that the maximum is well-defined as long as the set of aliases of a variable is finite.

8.2 Finiteness

To ensure finiteness of the type abstraction, we only consider stacks with variables taken from some arbitrary, but fixed, finite set $\mathbf{x}$ of variables. In particular, we denote by $\mathbf{DUSH}^{\mathbf{x}}_{\Phi}$ the restriction of delimited unfolded symbolic heaps ($\mathbf{DUSH}_{\Phi}$) to formulas ϕ with free variables in $\mathbf{x}$, i.e., $\mathsf{fvars}(\phi) \subseteq \mathbf{x}$. With this restriction in place, we are immediately able to argue the finiteness of the DUSH Fragment based on the following observation: Every variable can appear at most twice (once as the projection of a hole and once as the projection of a tree).

Lemma 8.6. *Let $n \triangleq |\Phi| + |\mathbf{x}|$, where $\mathbf{x}$ is a finite set of variables. Then $\left|\mathbf{DUSH}^{\mathbf{x}}_{\Phi}\right| \in 2^{O(n^2 \log(n))}$.*

Proof. See Appendix A.16. □

Analogously to $\mathbf{DUSH}^{\mathbf{x}}_{\Phi}$, we only consider aliasing constraints and types over the finite set $\mathbf{x}$, i.e., we consider the finite set of aliasing constraints $\mathbf{AC}^{\mathbf{x}} \triangleq \{\mathsf{aliasing}(\mathfrak{s}) \mid \mathfrak{s} \in \mathbf{Stacks}, \mathsf{dom}(\mathfrak{s}) = \mathbf{x}\}$. We note that the number of aliasing constraints in $\mathbf{AC}^{\mathbf{x}}$ equals the $|\mathbf{x}|$-th Bell number, bounded by $n^n \in O(2^{n \log(n)})$, where $n = |\mathbf{x}|$. Furthermore, we collect in $\mathbf{Types}^{\mathbf{x}}_{\Phi}$ all $\mathbf{ac}$-types over Φ and $\mathbf{x}$, i.e.,

$$\mathbf{Types}^{\mathbf{x}}_{\Phi} \triangleq \bigcup_{\mathbf{ac} \in \mathbf{AC}^{\mathbf{x}}} \bigcup_{\phi \in \mathbf{GSL}} \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi).$$

The above restriction of types to variables in $\mathbf{x}$ indeed ensures finiteness:

Theorem 8.7. *Let $\mathbf{x} \subseteq \mathbf{Var}$ be finite and $n \triangleq |\Phi| + |\mathbf{x}|$. Then $\left|\mathbf{Types}^{\mathbf{x}}_{\Phi}\right| \in 2^{2^{O(n^2 \log(n))}}$.*

Proof. Recall from Lemma 8.6 that the set $\mathbf{DUSH}^{\mathbf{x}}_{\Phi}$ of DUSHs over Φ with free variables taken from $\mathbf{x}$ is of size $2^{O(n^2 \log(n))}$. We show below that every type $\mathcal{T} \in \mathbf{Types}^{\mathbf{x}}_{\Phi}$ is a subset of $\mathbf{DUSH}^{\mathbf{x}}_{\Phi}$. Hence, the size of $\mathbf{Types}^{\mathbf{x}}_{\Phi}$ is bounded by the number of subsets of $\mathbf{DUSH}^{\mathbf{x}}_{\Phi}$, i.e., $\left|\mathbf{Types}^{\mathbf{x}}_{\Phi}\right| \in 2^{2^{O(n^2 \log(n))}}$.

It remains to show that, for every $\mathcal{T} \in \mathbf{Types}^{\mathbf{x}}_{\Phi}$, we have $\mathcal{T} \subseteq \mathbf{DUSH}^{\mathbf{x}}_{\Phi}$: By definition of $\mathbf{Types}^{\mathbf{x}}_{\Phi}$, there exists an aliasing constraint $\mathbf{ac} \in \mathbf{AC}^{\mathbf{x}}$ and a $\mathbf{GSL}$ formula ϕ such that $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$. By Definition 8.5, there exists a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ such that $\mathsf{dom}(\mathfrak{s}) = \mathsf{dom}(\mathbf{ac}) \subseteq \mathbf{x}$, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_{\Phi} \phi$, and $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$. By Definition 8.4, $\mathcal{T} = \{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \mathfrak{f} \in \mathsf{forests}_{\Phi}(\mathfrak{h})\} \cap \mathbf{DUSH}_{\Phi} \subseteq \mathbf{DUSH}^{\mathbf{x}}_{\Phi}$. □

8.3 $\mathfrak{s}$-Delimited Forests

To introduce the forests that correspond to DUSHs we make use of the notions of an *interface* of a Φ-forest, which is the set of locations that appear in some tree either as the root or as a hole:

Definition 8.8 (Interface). *The* interface *of a Φ-forest $\mathfrak{f} = \{\mathsf{t}_1, \ldots, \mathsf{t}_k\}$ is given by*

$$\mathsf{interface}(\mathfrak{f}) \triangleq \bigcup_{1 \leq i \leq k} (\{\mathsf{root}(\mathsf{t}_i)\} \cup \mathsf{allholes}(\mathsf{t}_i)).$$

Example 8.9 (Interface). *Recall the forest $\mathfrak{f}$ from Example 7.5. We have* $\mathsf{interface}(\mathfrak{f}) = \{l_1, l_2, l_3\}$*: the locations l_1, l_2, l_3 all occur as the roots of a tree; l_2 and l_3 additionally occur as holes (of t_3 and t_1, respectively); l_4 occurs neither as root nor as hole of a tree and is thus not part of the interface.*

An *$\mathfrak{s}$-delimited forest* is a Φ-forest whose interface consists only of locations covered by some stack variable and which does not have any duplicate holes:

Definition 8.10 ($\mathfrak{s}$-delimited Φ-Forest). *A Φ-forest $\mathfrak{f}$ is* $\mathfrak{s}$-delimited *iff (1)* $\mathsf{interface}(\mathfrak{f}) \subseteq \mathsf{img}(\mathfrak{s})$*, and (2) for every $l \in \mathsf{allholes}(\mathfrak{f})$ in some tree $\mathsf{t} \in \mathfrak{f}$, there is exactly one rule instance (for $l' \in \mathsf{dom}(\mathsf{t})$)*

$$\mathsf{rule}_{\mathsf{t}}(l') = \mathsf{pred}(\mathbf{v}) \Leftarrow (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{v_1}) \star \cdots \star \mathsf{pred}_m(\mathbf{v_m}) \star \Pi$$

and exactly one index $i \in [1, m]$ such that $\mathsf{predroot}(\mathsf{pred}_i(\mathbf{v_i})) = l$.

Example 8.11. *We consider the forests from Example 8.3: We note that* $\{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\}$ *from (1) is* $\mathfrak{s}$*-delimited, while* $\{\mathfrak{t}_1, \mathfrak{t}_2\}$ *from (2) is not.*

A Φ-forest is $\mathfrak{s}$-delimited precisely when its projection is delimited (see Appendix A.17 for a proof):

Lemma 8.12. *Let* $\mathfrak{f}$ *be a forest and let* $\mathfrak{s}$ *be a stack. Then* $\mathfrak{f}$ *is* $\mathfrak{s}$*-delimited iff* $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$ *is delimited.*

We now state that the $\mathfrak{s}$-delimitedness of forests is preserved under decomposition; this result will allow us to lift the composition of DUSH formulas (resp. $\mathfrak{s}$-delimited forests) to types.

Theorem 8.13. *Let* $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle \in \mathbf{GStates}$ *be guarded states, and let* $\mathfrak{f}$ *be an* $\mathfrak{s}$*-delimited forest with* $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$*. Then, there exist* $\mathfrak{s}$*-delimited forests* $\mathfrak{f}_1, \mathfrak{f}_2$ *with* $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$ *and* $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$*.*

Proof. See Appendix A.18. □

8.4 Operations on Types

type composition, *renaming of variables*, *forgetting variables*, and *type extension*. These operations will be the building blocks of our decision procedure for **GSL**.

8.4.1 Type Composition. We define an operation $\bullet$ on the level of Φ-types such that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$, i.e., $\mathsf{type}_\Phi(\mathfrak{s}, \cdot)$ is a homomorphism w.r.t. to the operation $\uplus$ on heaps and the operation $\bullet$ on types. As justified below, we can define $\bullet$ by applying our composition operation for forest projections, $\bullet_{\mathbf{P}}$ (cf. Definition 7.31), to all elements of the involved types.

Theorem 8.14 (Compositionality of Φ-types). *For all guarded states* $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ *and* $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ *with* $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \neq \bot$*,* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$ *can be computed from* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ *and* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$ *as follows:*

$$\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \{\phi \in \mathbf{DUSH}_\Phi \mid \textit{ex. } \psi_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1), \psi_2 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) \textit{ such that } \phi \in \psi_1 \bullet_{\mathbf{P}} \psi_2\}.$$

Proof. See Appendix A.19. □

Our second consideration for defining the composition operation $\bullet$ on Φ-types is that the operation $\uplus$ is only defined on disjoint heaps. In order to be able to express a corresponding condition on the level of types, we will make use of the following notion:

Definition 8.15 (Allocated variables of a type). *The set of* allocated variables *of* Φ*-type* $\mathcal{T}$ *is*

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

The above notion is motivated by the fact that, for each non-empty type, the allocated variables of the type agree with the allocated variables of every state having that type.

Lemma 8.16. *Let* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *be a state with* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \neq \emptyset$*. Then,* $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))$*.*

Proof. See Appendix A.20. □

We note that, for every model $\langle \mathfrak{s}, \mathfrak{h} \rangle$ of some predicate call $\mathsf{pred}(z_1, \ldots, z_k)$, there is at least one tree $\mathfrak{t}$ with $\mathsf{heap}(\{\mathfrak{t}\}) = \mathfrak{h}$ (see Lemma 7.3); hence, $\mathsf{project}(\mathfrak{s}, \{\mathfrak{t}\}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ and the non-emptiness requirement of Lemma 8.16 is fulfilled—a fact that generalizes to all models of guarded formulas:

Lemma 8.17. *Let* $\phi \in \mathbf{GSL}$ *and let* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *be a state with* $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$*. Then,* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \neq \emptyset$*.*

Proof. See Appendix A.21. □

We are now ready to state our composition operation $\bullet$ on Φ-types:

Definition 8.18 (Type composition). *The* composition $\mathcal{T}_1 \bullet \mathcal{T}_2$ *of* Φ*-types* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *is given by*

$$\mathcal{T}_1 \bullet \mathcal{T}_2 \triangleq \begin{cases} \bot, & \text{if } \mathsf{alloced}(\mathcal{T}_1) \cap \mathsf{alloced}(\mathcal{T}_2) \neq \emptyset, \\ \left(\bigcup_{\phi_1 \in \mathcal{T}_1, \phi_2 \in \mathcal{T}_2} \phi_1 \bullet_{\mathbf{P}} \phi_2\right) \cap \mathbf{DUSH}_\Phi, & \text{otherwise.} \end{cases}$$

We now state two results that $\bullet$ has the desired properties, i.e., that $\mathsf{type}_\Phi(\mathfrak{s}, \cdot)$ is a homomorphism w.r.t. to the operation $\uplus$ on heaps and the operation $\bullet$ on types (cf. Appendices A.22 and A.23):

Corollary 8.19 (Compositionality of type abstraction). *For guarded states* $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ *and* $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ *with* $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \neq \bot$*, we have* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$.

Lemma 8.20. *For* $i \in \{1, 2\}$*, let* $\langle \mathfrak{s}, \mathfrak{h}_i \rangle$ *be states with* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i) = \mathcal{T}_i \neq \emptyset$ *and* $\mathcal{T}_1 \bullet \mathcal{T}_2 \neq \bot$*. Then, there are states* $\langle \mathfrak{s}, \mathfrak{h}_i' \rangle$ *such that* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i') = \mathcal{T}_i$ *and* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1' \uplus \mathfrak{h}_2') = \mathcal{T}_1 \bullet \mathcal{T}_2$.

8.4.2 Renaming Variables. To compute the types of predicate calls $\mathsf{pred}(\mathbf{y})$ compositionally, we need a mechanism to rename variables in Φ-types: Once we know the types of a predicate call $\mathsf{pred}(\mathbf{x})$ over the formal arguments $\mathbf{x} = \mathsf{fvars}(\mathsf{pred})$, we can compute the types of $\mathsf{pred}(\mathbf{y})$ by renaming $\mathbf{x}$ to $\mathbf{y}$. Such a renaming amounts to a simple variable substitution:

Definition 8.21 (Variable Renaming). *Let* $\mathbf{x}$ *be a sequence of pairwise distinct variables, let* $\mathbf{y}$ *be an arbitrary sequence of variables with* $|\mathbf{y}| = |\mathbf{x}|$*, and let* $\mathbf{ac}$ *be an aliasing constraint with* $\mathbf{y} \subseteq \mathrm{dom}(\mathbf{ac})$*. Moreover, let* $\mathbf{y}'$ *be the sequence obtained by replacing every variable in* $y \in \mathbf{y}$ *by the maximal variable in its equivalence class, i.e., by* $[y]^{\mathbf{ac}}_{=}$*. Then, the* $[\mathbf{x}/\mathbf{y}]$-renaming *of type* $\mathcal{T}$ *w.r.t. aliasing constraint* $\mathbf{ac}$ *is given by* $\mathcal{T}[\mathbf{ac} : \mathbf{x}/\mathbf{y}] \triangleq \{\phi[\mathbf{x}/\mathbf{y}'] \mid \phi \in \mathcal{T}\}$.

Variable renaming is compositional as it corresponds to first renaming variables at the level of stacks and then computing the type of the resulting state. More formally, assume a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$, where we already renamed $\mathbf{x}$ to $\mathbf{y}$ in stack $\mathfrak{s}$; in particular, $\mathbf{x} \cap \mathrm{dom}(\mathfrak{s}) = \emptyset$. Computing $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ then coincides with the $[\mathbf{x}/\mathbf{y}]$-renaming of $\mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h})$, where $\mathfrak{s}' = \mathfrak{s}[\mathbf{x}/\mathbf{y}] \triangleq \mathfrak{s}[\mathbf{x}/\mathfrak{s}(\mathbf{y})]$ is the stack $\mathfrak{s}$ in which the variables $\mathbf{x}$ have not been renamed to $\mathbf{y}$ yet (cf. Appendix A.24).[9]

Lemma 8.22. *For* $\mathbf{x}$*,* $\mathbf{y}$ *as above and a stack* $\mathfrak{s}$ *with* $\mathbf{y} \subseteq \mathrm{dom}(\mathfrak{s})$ *and* $\mathbf{x} \cap \mathrm{dom}(\mathfrak{s}) = \emptyset$*, we have*

$$\mathsf{type}_\Phi(\mathfrak{s}[\mathbf{x}/\mathbf{y}], \mathfrak{h})[\mathsf{aliasing}(\mathfrak{s}) : \mathbf{x}/\mathbf{y}] = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}).$$

8.4.3 Forgetting Variables. Our third operation on types removes a free variable x from a type $\mathcal{T}$. Intuitively, for every formula $\phi \in \mathcal{T}$, there are two cases:

(1) If x aliases with some free variable, then we replace x by its largest alias.
(2) If x does not alias with any free variable, then we remove it from the set of free variables by introducing a (guarded) existential quantifier.

Formally, we fix an aliasing constraint $\mathbf{ac}$ (cf. Section 8.1.1) characterizing which free variables are aliases. Forgetting a variable x in a formula ϕ with respect to $\mathbf{ac}$ is then defined as follows:

$$\mathsf{forget}_{\mathbf{ac},x}(\phi) \triangleq \begin{cases} \phi[x/\max(\mathbf{ac}(x) \setminus \{x\})], & \text{if } x \in \mathsf{fvars}(\phi) \text{ and } \mathbf{ac}(x) \neq \{x\}, \\ \exists x.\, \phi, & \text{if } x \in \mathsf{fvars}(\phi) \text{ and } \mathbf{ac}(x) = \{x\}, \\ \phi, & \text{if } x \notin \mathsf{fvars}(\phi)\ . \end{cases}$$

Forgetting a variable in a *type* $\mathcal{T}$ corresponds to applying the above operation to all $\phi \in \mathcal{T}$. However, $\mathsf{forget}_{\mathbf{ac},x}(\phi)$ does—in general—not belong to the fragment $\mathbf{DUSH}_\Phi$ because we might existentially quantify over a root variable of ϕ. Hence, we additionally intersect the result with $\mathbf{DUSH}_\Phi$:

[9] Recall that $\mathfrak{s}[\mathbf{u}/\mathbf{v}]$ denotes a stack *update* in which variables in $\mathbf{u}$ are added to the domain of stack $\mathfrak{s}$ if necessary.

Definition 8.23 (Forgetting a variable). *The Φ-type obtained from* forgetting *variable x in Φ-type $\mathcal{T}$ w.r.t. aliasing constraint* $\mathbf{ac}$ *is defined by* $\mathsf{forget}_{\mathbf{ac},x}(\mathcal{T}) \triangleq \left\{\mathsf{forget}_{\mathbf{ac},x}(\phi) \mid \phi \in \mathcal{T}\right\} \cap \mathbf{DUSH}_\Phi$.

The above operation is compositional as forgetting an allocated variable in the type of a guarded state coincides with first removing the variable from the state and then computing its type:

Lemma 8.24. *Let $\langle \mathfrak{s}, \mathfrak{h}\rangle$ be a guarded state such that $\mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{h})$ holds for some variable x. Then,*

$$\mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),x}(\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})) = \mathsf{type}_\Phi(\mathfrak{s}[x/\bot],\mathfrak{h}) \ .$$

Proof. See Appendix A.25. □

8.4.4 Type Extension. Our fourth and final operation is concerned with extending types to stacks over larger domains. To this end, we instantiate universally quantified variables with free variables that do not appear in the type so far. Formally, let $\phi = \exists \mathbf{e}.\, \forall(\mathbf{a} \cdot u \cdot \mathbf{b}).\, \psi$ be a formula and let x be a fresh variable, i.e., $x \notin \mathrm{fvars}(\phi)$. We call the formula $\exists \mathbf{e}.\, \forall(\mathbf{a} \cdot \mathbf{b}).\, \psi[u/x]$ an *x-instantiation* of ϕ. Extending a type by variable x then corresponds to adding all x-instantiations of its members:

Definition 8.25 (x-extension of a Typ). *The* x-extension *of a Φ-type $\mathcal{T}$ by a fresh variable x is*

$$\mathsf{extend}_x(\mathcal{T}) \triangleq \mathcal{T} \cup \{\phi' \textit{ is an } x\textit{-instantiation of } \phi \mid \phi \in \mathcal{T}\} \ .$$

As for the other operations, the x-extension of a type is compositional in the sense that it coincides with computing the type of a state with an already extended stack (cf. Appendix A.26):

Lemma 8.26. *For every state $\langle \mathfrak{s}, \mathfrak{h}\rangle$, variable x with $\mathfrak{s}(x) \notin \mathsf{locs}(\mathfrak{h})$ and $\mathsf{aliasing}(\mathfrak{s})(x) = \{x\}$,*

$$\mathsf{extend}_x(\mathsf{type}_\Phi(\mathfrak{s}[x/\bot],\mathfrak{h})) = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}).$$

Rather than extending a type by a single variable, it will be convenient to extend it by all variables in an *aliasing constraint* that are not aliases of an existing variable.

Definition 8.27 (Extension of a type with regard to an aliasing constraint). *Let $\mathbf{ac} \subseteq \mathbf{ac}'$ be aliasing constraints. Let $\mathbf{y}$ be a repetition-free sequence of all maximal variables in* $\mathrm{dom}(\mathbf{ac})$*, and let $\mathbf{y}'$ be the sequence obtained by replacing every variable in $y \in \mathbf{y}$ by the corresponding maximal variable in $\mathbf{ac}'$, i.e., by $[y]^{\mathbf{ac}'}_{=}$.* [10] *Moreover, let $\mathbf{z} = \langle z_1, \ldots, z_n\rangle$, $n \geq 0$, be a repetition-free sequence of all maximal variables in* $\mathrm{dom}(\mathbf{ac}')$ *that are not aliases of variables in* $\mathrm{dom}(\mathbf{ac})$*.* [11] *Then the* $\mathbf{ac}'$-extension *of a Φ-type $\mathcal{T}$ w.r.t. aliasing constraint $\mathbf{ac}$ is defined as* $\mathsf{extend}_{\mathbf{ac}'}(\mathcal{T}) \triangleq \mathcal{T}_n$*, where*

$$\mathcal{T}_k = \begin{cases} \{\phi[\mathbf{y}/\mathbf{y}'] \mid \phi \in \mathcal{T}\}, & \textit{if } k = 0 \\ \mathsf{extend}_{z_k}(\mathcal{T}_{k-1}), & \textit{if } 0 < k \leq n \ . \end{cases}$$

The above operation preserves compositionality as it boils down to multiple type extensions:

Lemma 8.28. *Let $\langle \mathfrak{s}, \mathfrak{h}\rangle$ be a state and $\mathbf{ac}$ be an aliasing constraint with $\mathbf{ac} \subseteq \mathsf{aliasing}(\mathfrak{s})$. Let $\mathfrak{s}'$ be the restriction of $\mathfrak{s}$ to the domain* $\mathrm{dom}(\mathbf{ac})$*. If $\mathfrak{s}(x) \notin \mathsf{locs}(\mathfrak{h})$ for every variable $x \in \mathrm{dom}(\mathfrak{s})$ that is not an alias of a variable in* $\mathrm{dom}(\mathbf{ac})$*, then* $\mathsf{extend}_{\mathsf{aliasing}(\mathfrak{s})}(\mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h})) = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$.

Proof. Let $k \geq 0$ be the number of variables in $\mathrm{dom}(\mathfrak{s})$ that are no aliases of variables in $\mathrm{dom}(\mathbf{ac})$. By Definition 8.27, we have $\mathsf{extend}_{\mathsf{aliasing}(\mathfrak{s})}(\mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h})) = \mathcal{T}_k$, i.e., we need to apply k type extensions. The claim then follows from Lemma 8.26 by induction on k. □

[10] We recall that we need maximal variables for maintaining canonic projections, i.e., type representations.

[11] I.e., $z \in \mathbf{z}$ iff $z \in \mathrm{dom}(\mathbf{ac}')$, $z = [z]^{\mathbf{ac}'}_{=}$ and $z \notin \mathbf{ac}'(y)$ for all $y \in \mathrm{dom}(\mathbf{ac})$.

8.5 Type Refinement

The main insight required for effectively deciding whether a type $\mathcal{T}$ belongs to $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ is that states with identical Φ-types satisfy the same **GSL** formulas—a statement we formalize below. This property is perhaps surprising, as types only contain formulas from the DUSH fragment, which is largely orthogonal to **GSL**. For example, **GSL** formulas allow guarded negation and guarded septraction, but neither quantifiers nor unguarded magic wands, whereas DUSHs allow limited use of guarded quantifiers and unguarded magic wands, but neither Boolean structure nor septraction.

Theorem 8.29 (Refinement theorem). *For all stacks* $\mathfrak{s}$*, heaps* $\mathfrak{h}_1$*,* $\mathfrak{h}_2$*, and* **GSL** *formulas* ϕ*,*

$$\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}_2) \qquad \textit{implies} \qquad \langle \mathfrak{s},\mathfrak{h}_1\rangle \models_\Phi \phi \;\; \textit{iff} \;\; \langle \mathfrak{s},\mathfrak{h}_2\rangle \models_\Phi \phi\ .$$

Proof. See Appendix A.27. □

Theorem 8.29 immediately implies that, if the type of a state $\langle \mathfrak{s},\mathfrak{h}\rangle$ is equal to some already-known type of some other state $\langle \mathfrak{s}',\mathfrak{h}'\rangle$ satisfying formula ϕ, then $\langle \mathfrak{s},\mathfrak{h}\rangle$ satisfies ϕ.

Corollary 8.30. *If there is a type* $\mathcal{T} \in \mathbf{Types}^{\mathsf{aliasing}(\mathfrak{s})}_{\Phi}(\phi)$ *with* $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) = \mathcal{T}$*, then* $\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi \phi$*.*

Moreover, recall that **GSL** formulas are quantifier-free (although quantifiers may appear in predicate definitions). As demonstrated below, this limitation is crucial for upholding Theorem 8.29.

Example 8.31. *Recall* Φ_{ls} *from Example 3.3. Moreover, let* $\langle \mathfrak{s},\mathfrak{h}_k\rangle$*,* $k \in \mathbb{N}$*, be a list of length* k *from* x_1 *to* x_2*. It then holds for all* $i, j \geq 2$ *that* $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}_i) = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}_j)$*. However,*

$$\langle \mathfrak{s},\mathfrak{h}_2\rangle \not\models_\Phi \exists\, \langle y_1, y_2\rangle \,.\, \mathsf{lseg}(x_1,y_1) \star \mathsf{lseg}(y_1,y_2) \star \mathsf{lseg}(y_2,x_2), \;\; \textit{whereas, for all } j \geq 3,$$
$$\langle \mathfrak{s},\mathfrak{h}_j\rangle \models_\Phi \exists\, \langle y_1, y_2\rangle \,.\, \mathsf{lseg}(x_1,y_1) \star \mathsf{lseg}(y_1,y_2) \star \mathsf{lseg}(y_2,x_2).$$

Hence, the refinement theorem does not hold if we admit quantifiers in **GSL** *formulas.*

9 ALGORITHMS FOR COMPUTING TYPES

As discussed in Section 8.1, deciding whether a **GSL** formula ϕ is decidable boils down to computing finite sets of types $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ for suitable aliasing constraints $\mathbf{ac}$. We now present two algorithms for effectively computing $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$: Section 9.1 deals with computing types of predicate calls defined by SIDs and Section 9.2 shows how to compute types of **GSL** formulas, respectively.[12]

9.1 Computing the Types of Predicate Calls

We first compute, for every predicate $\mathsf{pred} \in \mathbf{Preds}(\Phi)$, the set of all $\mathbf{ac}$-types of pred. Specifically, for every aliasing constraint $\mathbf{ac} \in \mathbf{AC}^{\mathbf{x}\cup\mathsf{fvars}(\mathsf{pred})}$, where $\mathbf{x} \subseteq \mathbf{Var}$ finite, we will compute

$$\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\mathsf{pred}) \triangleq \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))).$$

Once we have a way to compute these types, we can also compute types for any **GSL** formula with free variables $\mathbf{x}$, as we will see in Section 9.2.

9.1.1 Assumptions. Throughout this section, we fix a pointer-closed SID $\Phi \in \mathbf{ID}_{\mathrm{btw}}$ and a finite set of variables $\mathbf{x}$; we assume w.l.o.g. that $\mathbf{x} \cap \mathsf{fvars}(\mathsf{pred}) = \emptyset$ for all predicates $\mathsf{pred} \in \mathbf{Preds}(\Phi)$.

$$
\begin{array}{lcl}
\mathrm{ptypes}^{\mathbf{x}}_p(x \approx y, \mathbf{ac}) & \triangleq & \text{if } \langle x, y\rangle \in \mathbf{ac} \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset \\
\mathrm{ptypes}^{\mathbf{x}}_p(x \not\approx y, \mathbf{ac}) & \triangleq & \text{if } \langle x, y\rangle \notin \mathbf{ac} \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset \\
\mathrm{ptypes}^{\mathbf{x}}_p(a \mapsto \mathbf{b}, \mathbf{ac}) & \triangleq & \left\{\mathrm{type}_\Phi(\mathrm{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b}))\right\} \\
\mathrm{ptypes}^{\mathbf{x}}_p(\mathrm{pred}(\mathbf{y}), \mathbf{ac}) & \triangleq & \text{let } \mathbf{z} \triangleq \mathrm{fvars}(\mathrm{pred}) \text{ in} \\
 & & \mathrm{extend}_{\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}}(p(\mathrm{pred}, \mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}\big|_{\mathbf{x}\cup\mathbf{z}})[\mathbf{ac} : \mathbf{z}/\mathbf{y}]) \\
\mathrm{ptypes}^{\mathbf{x}}_p(\phi_1 \star \phi_2, \mathbf{ac}) & \triangleq & \mathrm{ptypes}^{\mathbf{x}}_p(\phi_1, \mathbf{ac}) \bullet \mathrm{ptypes}^{\mathbf{x}}_p(\phi_2, \mathbf{ac}) \\
\mathrm{ptypes}^{\mathbf{x}}_p(\exists y.\, \phi, \mathbf{ac}) & \triangleq & \bigcup_{\mathbf{ac}' \in \mathbf{AC}^{\mathrm{dom}(\mathbf{ac})\cup\{y\}} \text{ with } \mathbf{ac}'|_{\mathrm{dom}(\mathbf{ac})} = \mathbf{ac}} \mathrm{forget}_{\mathbf{ac}', y}(\mathrm{ptypes}^{\mathbf{x}}_p(\phi, \mathbf{ac}'))
\end{array}
$$

Fig. 11. Computing (a subset of) the Φ-types of existentially-quantified symbolic heap $\phi \in \mathbf{SH}^\exists$ for stacks with aliasing constraint $\mathbf{ac}$ under the assumption that p maps every predicate symbol pred and every aliasing constraint to (a subset of) the types $\mathbf{Types}^{\mathbf{ac}}_\Phi(\mathrm{pred})$. Here, $\mathbf{ac}[\mathbf{u}/\mathbf{v}]^{-1}$ denotes the addition of the variables $\mathbf{u}$ into the aliasing constraint $\mathbf{ac}$ such that the variables $\mathbf{u}$ are aliases of the variables $\mathbf{v}$ respectively; see Definition 9.1. We denote by $\mathbf{ac}|_{\mathbf{y}}$ the restriction of $\mathbf{ac}$ to the variables in $\mathbf{y}$, i.e., $\mathbf{ac}|_{\mathbf{y}} \triangleq \mathbf{ac} \cap (\mathbf{y} \times \mathbf{y})$.

9.1.2 A Fixed-Point Algorithm for Computing the Types of Predicates. We compute $\mathbf{Types}^{\mathbf{ac}}_\Phi(\mathrm{pred})$ for all choices of $\mathbf{ac}$ and pred by a simultaneous fixed-point computation. Specifically, our goal is to compute a (partial) function $p\colon \mathbf{Preds}(\Phi) \times \mathbf{AC} \to 2^{\mathbf{Types}_\Phi}$ that maps every predicate pred and every aliasing constraint $\mathbf{ac} \in \mathbf{AC}^{\mathbf{x}\cup\mathrm{fvars}(\mathrm{pred})}$ to the set of types $\mathbf{Types}^{\mathbf{ac}}_\Phi(\mathrm{pred})$. We start off the fixed-point computation with $p(\mathrm{pred}, \mathbf{ac}) = \emptyset$ for all pred and $\mathbf{ac}$; each iteration adds to p some more types such that $p(\mathrm{pred}, \mathbf{ac}) \subseteq \mathbf{Types}^{\mathbf{ac}}_\Phi(\mathrm{pred})$; and when we reach the fixed point, $p(\mathrm{pred}, \mathbf{ac}) = \mathbf{Types}^{\mathbf{ac}}_\Phi(\mathrm{pred})$ will hold for all pred and $\mathbf{ac}$. Each iteration amounts to applying the function $\mathrm{ptypes}^{\mathbf{x}}_p(\phi, \mathbf{ac})$ defined in Fig. 11 to all rule bodies $\phi \in \mathbf{SH}^\exists$ of the SID Φ and all aliasing constraints $\mathbf{ac}$. Here, p is the pre-fixed point from the previous iteration. The function ptypes operates on sets of types. Hence, we need to lift $\bullet$, $\cdot[\cdot : \cdot/\cdot]$, forget and extend from types to sets of types in a point-wise manner, i.e.,

$$
\begin{array}{l}
\{\mathcal{T}_1, \ldots, \mathcal{T}_m\} \bullet \left\{\mathcal{T}_1', \ldots, \mathcal{T}_n'\right\} \triangleq \left\{\mathcal{T}_i \bullet \mathcal{T}_j' \mid 1 \le i \le m, 1 \le j \le n, \mathcal{T}_i \bullet \mathcal{T}_j' \neq \bot\right\}, \\
\{\mathcal{T}_1, \ldots, \mathcal{T}_m\}[\mathbf{ac} : \mathbf{x}/\mathbf{y}] \triangleq \{\mathcal{T}_1[\mathbf{ac} : \mathbf{x}/\mathbf{y}], \ldots, \mathcal{T}_m[\mathbf{ac} : \mathbf{x}/\mathbf{y}]\}, \\
\mathrm{forget}_{\mathbf{ac}, y}(\{\mathcal{T}_1, \ldots, \mathcal{T}_m\}) \triangleq \left\{\mathrm{forget}_{\mathbf{ac}, y}(\mathcal{T}_1), \ldots, \mathrm{forget}_{\mathbf{ac}, y}(\mathcal{T}_m)\right\}, \text{ and} \\
\mathrm{extend}_{\mathbf{ac}}(\{\mathcal{T}_1, \ldots, \mathcal{T}_m\}) \triangleq \{\mathrm{extend}_{\mathbf{ac}}(\mathcal{T}_1), \ldots, \mathrm{extend}_{\mathbf{ac}}(\mathcal{T}_m)\}.
\end{array}
$$

Further, ptypes uses the following operation on aliasing constraints:

Definition 9.1 (Reverse renaming of aliasing constraints). *Let* $\mathbf{x}$ *be a sequence of pairwise distinct variables and let* $\mathbf{y}$ *be a sequence of (not necessarily pairwise distinct) variables with* $|\mathbf{y}| = |\mathbf{x}|$. *Moreover, let* $\mathbf{ac}$ *be an aliasing constraint with* $\mathbf{x} \cap \mathrm{dom}(\mathbf{ac}) = \emptyset$ *and* $\mathbf{y} \subseteq \mathrm{dom}(\mathbf{ac})$. *Then, the* reverse renaming $\mathbf{x}$ *to* $\mathbf{y}$ *in* $\mathbf{ac}$ *is given by the aliasing constraint* $\mathbf{ac}[\mathbf{x}/\mathbf{y}]^{-1} \in \mathbf{AC}^{\mathrm{dom}(\mathbf{ac})\cup\mathbf{x}}$ *defined by*

$$\mathbf{ac}[\mathbf{x}/\mathbf{y}]^{-1} \triangleq \{(a_1, a_2) \mid \textit{ there is } (b_1, b_2) \in \mathbf{ac} \textit{ with } b_1 = a_1[\mathbf{x}/\mathbf{y}] \textit{ and } b_2 = a_2[\mathbf{x}/\mathbf{y}]\}.$$

Informally, the function $\mathrm{ptypes}^{\mathbf{x}}_p(\phi, \mathbf{ac})$ works as follows:

- If $\phi = x \approx y$ or $\phi = x \not\approx y$, we use the aliasing constraint $\mathbf{ac}$ to check whether the (dis)equality ϕ holds and then return either the type of the empty model or no type. This is justified because our semantics enforces that (dis)equalities only hold in the empty heap.
- If $\phi = a \mapsto \mathbf{b}$, there is—up to isomorphism—only one state with aliasing constraint $\mathbf{ac}$ that satisfies ϕ. We denote this state by $\mathrm{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b})$ and return its type.

[12]Recall that the formulas in SIDs are symbolic heaps and *not* GSL formulas; for example, they may contain quantifiers.

- If $\phi = \mathsf{pred}(\mathbf{y})$, we look up the types of $\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$ in the pre-fixed point p and then appropriately rename the formal parameters $\mathsf{fvars}(\mathsf{pred})$ to the actual arguments $\mathbf{y}$:
  - For the look-up we use the aliasing constraint $\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}$, which is obtained from the aliasing constraint $\mathbf{ac}$ by adding the formal parameters $\mathbf{z} \triangleq \mathsf{fvars}(\mathsf{pred})$ to $\mathbf{ac}$ such that the $\mathbf{z}$ are aliases of the variables $\mathbf{y}$ respectively; see Definition 9.1 for details.
  - Crucially, we restrict $\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}$ to the variables $\mathbf{x} \cup \mathbf{z}$ before we look up the types of $\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$. This restriction guarantees that the computation of $\mathsf{ptypes}_p^{\mathbf{x}}(\phi, \mathbf{ac})$ does not diverge by considering larger and larger aliasing constraints in recursive calls. (An illustration of the problem as well as an argument why our solution does not lead to divergence can be found in Appendix A.29.2).
  - After the loop-up we extend the types over aliasing constraint $\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}\big|_{\mathbf{x}\cup\mathbf{z}}$ to types over aliasing constraint $\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}$, undoing the earlier restriction.
  - Finally, we rename the formal parameters $\mathbf{z}$ of the recursive call with the actual parameters $\mathbf{y}$ and obtain types over aliasing constraint $\mathbf{ac}$.
- If $\phi = \phi_1 \star \phi_2$, we apply the type composition operator developed in previous sections.
- If $\phi = \exists y.\, \phi'$, we consider all ways to extend the aliasing constraint $\mathbf{ac}$ with y and recurse. Our treatment of predicate calls outlined above guarantees that this does not lead to divergence.

Fixed Point Computation. We use the following wrapper for ptypes:

$$\mathsf{unfold}_{\mathbf{x}} \colon (\mathbf{Preds}(\Phi) \times \mathbf{AC} \to 2^{\mathbf{Types}_\Phi}) \to (\mathbf{Preds}(\Phi) \times \mathbf{AC} \to 2^{\mathbf{Types}_\Phi}),$$

$$\mathsf{unfold}_{\mathbf{x}}(p) = \lambda(\mathsf{pred}, \mathbf{ac}).\, p(\mathsf{pred}, \mathbf{ac}) \cup \bigcup_{(\mathsf{pred}(\mathbf{y}) \Leftarrow \phi) \in \Phi} \mathsf{ptypes}_p^{\mathbf{x}}(\mathbf{ac}, \phi).$$

We observe that $\mathsf{unfold}_{\mathbf{x}}$ is a monotone function defined over a finite complete lattice:

(1) The considered order $\sqsubseteq$ of $\mathbf{Preds}(\Phi) \times \mathbf{AC} \to 2^{\mathbf{Types}_\Phi}$ is the point-wise comparison of functions:
$$f \sqsubseteq g \triangleq \forall \mathsf{pred} \forall \mathbf{ac}.\, f(\mathsf{pred}, \mathbf{ac}) \subseteq g(\mathsf{pred}, \mathbf{ac}).$$
(2) $\mathbf{Preds}(\Phi) \times \mathbf{AC} \to 2^{\mathbf{Types}_\Phi}$ is a finite lattice because the image $2^{\mathbf{Types}_\Phi}$ and the domain $\left\{\langle \mathsf{pred}, \mathbf{ac}\rangle \mid \mathsf{pred} \in \mathbf{Preds}(\Phi), \mathbf{ac} \in \mathbf{AC}^{\mathbf{x}\cup\mathsf{fvars}(\mathsf{pred})}\right\}$ of the considered functions are finite.
(3) $\mathbf{Preds}(\Phi) \times \mathbf{AC} \to 2^{\mathbf{Types}_\Phi}$ is complete because the image $2^{\mathbf{Types}_\Phi}$ of the considered functions is a complete lattice (the subset lattice over $\mathbf{Types}_\Phi$).

By Tarski's and Knaster's fixed point theorem the least fixed point of $\mathsf{unfold}_{\mathbf{x}}$ exists. This fixed point can be obtained in finitely many steps by Kleene iteration[13]:

$$\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}}) \triangleq \lim_{n\in\mathbb{N}} \mathsf{unfold}_{\mathbf{x}}^n(\lambda(\mathsf{pred}', \mathbf{ac}').\, \emptyset)$$

Moreover, since the lattice is finite, finitely many iterations suffice to reach the least fixed point.

Correctness and Complexity. We analyze the correctness of our construction, i.e.,

$$\text{for all } \mathsf{pred} \in \mathbf{Preds}(\Phi) \text{ and } \mathbf{ac} \in \mathbf{AC}^{\mathbf{x}\cup\mathsf{fvars}(\mathsf{pred})}.\quad \mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \mathbf{ac}) = \mathbf{Types}_\Phi^{\mathbf{ac}}(\mathsf{pred}),$$

as well as its complexity in three steps, which can be found in A.29:

(1) We show $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \mathbf{ac}) \subseteq \mathbf{Types}_\Phi^{\mathbf{ac}}(\mathsf{pred})$.
(2) We show $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \mathbf{ac}) \supseteq \mathbf{Types}_\Phi^{\mathbf{ac}}(\mathsf{pred})$.
(3) We show that $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})$ is computable in $2^{2^{O(n^2 \log(n))}}$, where $n \triangleq |\Phi| + |\mathbf{x}|$.

[13]To be precise we invoke a constructive version of Tarski's and Knaster's fixed point theorem [Cousot and Cousot 1979], which supports the computation of the least fixed point by transfinite induction; the finiteness of the lattice, however, ensures that the fixed point is already reached after finitely many steps.

$$
\begin{array}{lcl}
\mathsf{types}(\mathbf{emp}, \mathbf{ac}) & \triangleq & \{\{\mathbf{emp}\}\} \\
\mathsf{types}(x \approx y, \mathbf{ac}) & \triangleq & \text{if } \langle x, y\rangle \in \mathbf{ac} \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset \\
\mathsf{types}(x \not\approx y, \mathbf{ac}) & \triangleq & \text{if } \langle x, y\rangle \notin \mathbf{ac} \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset \\
\mathsf{types}(a \mapsto \mathbf{b}, \mathbf{ac}) & \triangleq & \left\{\mathsf{type}_\Phi(\mathsf{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b}))\right\} \\
\mathsf{types}(\mathsf{pred}(\mathbf{y}), \mathbf{ac}) & \triangleq & \mathsf{lfp}(\mathsf{unfold}_{\mathrm{dom}(\mathbf{ac})})(\mathsf{pred}, \mathbf{ac}[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]^{-1})[\mathbf{ac} : \mathsf{fvars}(\mathsf{pred})/\mathbf{y}] \\
\mathsf{types}(\phi_1 \star \phi_2, \mathbf{ac}) & \triangleq & \mathsf{types}(\phi_1, \mathbf{ac}) \bullet \mathsf{types}(\phi_2, \mathbf{ac}) \\
\mathsf{types}(\phi_1 \wedge \phi_2, \mathbf{ac}) & \triangleq & \mathsf{types}(\phi_1, \mathbf{ac}) \cap \mathsf{types}(\phi_2, \mathbf{ac}) \\
\mathsf{types}(\phi_1 \vee \phi_2, \mathbf{ac}) & \triangleq & \mathsf{types}(\phi_1, \mathbf{ac}) \cup \mathsf{types}(\phi_2, \mathbf{ac}) \\
\mathsf{types}(\phi_1 \wedge \neg\phi_2, \mathbf{ac}) & \triangleq & \mathsf{types}(\phi_1, \mathbf{ac}) \setminus \mathsf{types}(\phi_2, \mathbf{ac}) \\
\mathsf{types}(\phi_0 \wedge (\phi_1 \circledast \phi_2), \mathbf{ac}) & \triangleq & \{\mathcal{T} \in \mathsf{types}(\phi_0, \mathbf{ac}) \mid \exists \mathcal{T}' \in \mathsf{types}(\phi_1, \mathbf{ac}).\, \mathcal{T} \bullet \mathcal{T}' \in \mathsf{types}(\phi_2, \mathbf{ac})\} \\
\mathsf{types}(\phi_0 \wedge (\phi_1 \mathrel{-\!\!\star} \phi_2), \mathbf{ac}) & \triangleq & \{\mathcal{T} \in \mathsf{types}(\phi_0, \mathbf{ac}) \mid \forall \mathcal{T}' \in \mathsf{types}(\phi_1, \mathbf{ac}).\, \mathcal{T} \bullet \mathcal{T}' \in \mathsf{types}(\phi_2, \mathbf{ac})\}
\end{array}
$$

Fig. 12. Computation of Φ-types for quantifier-free GSL formula ϕ and stacks with aliasing constraint $\mathbf{ac}$.

9.2 Computing the Types of Guarded Formulas

After we have established how to compute the types of predicate calls, we are now ready to define a function $\mathsf{types}(\phi, \mathbf{ac})$ that computes the types of arbitrary GSL formulas ϕ—i.e., quantifier-free guarded formulas—for some fixed stack-aliasing constraint $\mathbf{ac}$; the function is defined in Fig. 12.

Theorem 9.2 (Correctness and Complexity of the Type Computation). *Let $\phi \in \mathbf{GSL}$ with $\mathsf{fvars}(\phi) = \mathbf{x}$ and $\mathsf{locs}(\phi) = \emptyset$. Further, let $\mathbf{ac} \in \mathbf{AC}^{\mathbf{x}}$. Then, $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi) = \mathsf{types}(\phi, \mathbf{ac})$. Moreover, $\mathsf{types}(\phi, \mathbf{ac})$ can be computed in $2^{2^{O(n^2 \log(n))}}$, where $n \triangleq |\Phi| + |\phi|$.*

We now state the main result of this article:

Theorem 9.3 (Decidability of GSL). *Let $\phi \in \mathbf{GSL}$ and $n \triangleq |\Phi| + |\phi|$. It is decidable in time $2^{2^{O(n^2 \log(n))}}$ whether ϕ is satisfiable.*

Proof. Let $\mathbf{x} \triangleq \mathsf{fvars}(\phi)$. Note that $|\mathbf{x}| \leq n$. The formula ϕ is satisfiable iff there exists a state $\langle \mathfrak{s}, \mathfrak{h}\rangle$ with $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi$. By Lemma 8.17, $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \neq \emptyset$. Hence, it is sufficient to compute $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi)$ for all aliasing constraints $\mathbf{ac}$ with $\mathrm{dom}(\mathbf{ac}) = \mathbf{x}$ and check whether $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi) \neq \emptyset$.

By Theorem 9.2 we can compute $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi) = \mathsf{types}(\phi, \mathbf{ac})$ in $2^{2^{O(n^2 \log(n))}}$ for a fixed aliasing constraints $\mathbf{ac}$. Since there are at most $n^n \in O(2^{n \log(n)})$ stack-aliasing constraints, we can conclude that we can perform the satisfiability check in time $O(2^{n \log(n)}) \cdot 2^{2^{O(n^2 \log(n))}} = 2^{2^{O(n^2 \log(n))}}$. □

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 9.4 (Decidability of entailment for GSL). *Let $\phi, \psi \in \mathbf{GSL}$ and $n \triangleq |\Phi| + |\phi| + |\psi|$. The entailment problem $\phi \models_\Phi \psi$ is decidable in time $2^{2^{O(n^2 \log(n))}}$.*

Proof. If $\phi, \psi \in \mathbf{GSL}$, then $\phi \wedge \neg\psi \in \mathbf{GSL}$. The entailment $\phi \models_\Phi \psi$ is valid iff $\phi \wedge \neg\psi$ is unsatisfiable. Since 2ExpTime is closed under complement, the claim follows from Theorem 9.3. □

Example 9.5. *The entailments in Example 6.8 can be proven using our decision procedure.*

Finally, our decision procedure is also applicable to (quantifier-free) symbolic heaps over inductive predicate definitions of bounded treewidth, because these formulas are always guarded.[14] Hence, we also obtain a tighter complexity bound for the original decidability result of Iosif et al. [2013]:

Corollary 9.6 (Decidability of entailment for $\mathbf{SL}_{\mathrm{btw}}$). *Let $\phi, \psi \in \mathbf{SL}_{\mathrm{btw}}$ be quantifier-free and $n \triangleq |\Phi| + |\phi| + |\psi|$. The entailment problem $\phi \models_\Phi \psi$ is decidable in time $2^{2^{O(n^2 \log(n))}}$.*

Proof. Follows from Corollary 9.4, since every quantifier-free $\mathbf{SL}_{\mathrm{btw}}$ formula is in **GSL**. □

Corollary 9.7 (Decidability over Values with Null-Pointer). *Let $\phi, \psi \in \mathbf{GSL}$ with $\mathrm{locs}(\phi) \cup \mathrm{locs}(\phi) \subseteq \{nil\}$. Then, the satisfiability of ϕ resp. the entailment $\phi \models_\Phi \psi$ over $\mathbf{Val} \triangleq \mathbf{Loc} \cup \{nil\}$ is decidable in $2^{2^{O(n^2 \log(n))}}$ for $n \triangleq |\Phi| + |\phi|$ resp. $n \triangleq |\Phi| + |\phi| + |\psi|$.*

10 CONCLUSION

We have given a unified and revised presentation of the decision procedures developed in [Katelaan et al. 2019; Katelaan and Zuleger 2020] covering (1) the satisfiability of quantifier-free guarded separation logic and (2) the entailment problem of (quantifier-free) symbolic heaps over SIDs of bounded treewidth. In particular, we have established a 2ExpTime upper bound for both problems. A corresponding lower bound has been proven recently [Echenim et al. 2020b]. Hence, we can conclude that our decision procedures have optimal computational complexity.

To the best of our knowledge, our decision procedure for **GSL** is the first decision procedure to support an SL fragment combining user-supplied inductive definitions, Boolean structure, and magic wands. We obtained an almost tight delineation between decidability and undecidability: We showed that any extension of **GSL** in which one of the guards is dropped, leads to undecidability.

In this article, we considered the quantifier-free fragment of **GSL**; quantifiers can only appear in SID rules. An interesting question for future research is to what extent quantifiers can also be admitted in **GSL** formulas without sacrificing decidability. This question is also of practical interest as quantifiers naturally appear in entailments obtained from verification condition generators.

We mention that recent follow-up work by Echenim et al. [2021] generalizes the decidability of the entailment problem for $\mathbf{SL}_{\mathrm{btw}}$ by weakening the establishment requirement. The result employs an abstraction inspired by the type abstraction presented in this article. It is an interesting question whether this result can be lifted to guarded separation logic as well. Further, our undecidability results require an unbounded number of dangling pointers. While [Echenim et al. 2021] supports classes of structures with unbounded treewidth, the entailment needs only to be checked for so-called normal structures of bounded treewidth. It would be interesting to interpret this result in terms of the number of dangling pointers that need to be considered in order to understand whether a bounded number of dangling pointers is fundamental for decidability.

Finally, apart from implementing our decision procedure, it would be interesting whether one can extract a proof certificate from our type abstraction that can also be checked independently by other proof systems based on separation logic.

REFERENCES

Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. 2014. Foundations for Decision Problems in Separation Logic with General Inductive Predicates. In *FOSSACS*. 411–425.

Andrew W. Appel. 2014. *Program Logics - for Certified Compilers.* Cambridge University Press.

[14] Notice that only the formulas in the entailment query need to be quantifier-free; quantifiers are permitted in inductive definitions. Since we can use an arbitrary number of free variables at the top-level, this is a mild restriction.

Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. 1961. On formal properties of simple phrase structure grammars. *Sprachtypologie und Universalienforschung* 14 (1961), 143–172.

Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Thomas Noll. 2019. Quantitative separation logic: a logic for reasoning about probabilistic pointer programs. *PACMPL* 3, POPL (2019), 34:1–34:29.

Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, Peter W. O'Hearn, Thomas Wies, and Hongseok Yang. 2007. Shape Analysis for Composite Data Structures. In *CAV*. 178–192.

Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. 2004. A Decidable Fragment of Separation Logic. In *FSTTCS*. 97–109.

Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. 2005a. Smallfoot: Modular Automatic Assertion Checking with Separation Logic. In *FMCO*. 115–137.

Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. 2005b. Symbolic Execution with Separation Logic. In *APLAS*. 52–68.

Josh Berdine, Byron Cook, and Samin Ishtiaq. 2011. SLAyer: Memory Safety for Systems-Level Code. In *CAV*. 178–183.

Stefan Blom and Marieke Huisman. 2015. Witnessing the elimination of magic wands. *Int. J. Softw. Tools Technol. Transf.* 17, 6 (2015), 757–781.

Rémi Brochenin, Stéphane Demri, and Étienne Lozes. 2012. On the almighty wand. *Inf. Comput.* 211 (2012), 106–137.

James Brotherston. 2007. Formalised Inductive Reasoning in the Logic of Bunched Implications. In *SAS (LNCS, Vol. 4634)*, Hanne Riis Nielson and Gilberto Filé (Eds.). Springer, 87–103.

James Brotherston, Dino Distefano, and Rasmus Lerche dahl Petersen. 2011. Automated Cyclic Entailment Proofs in Separation Logic. In *CADE-23*. 131–146.

James Brotherston, Carsten Fuhs, Juan Antonio Navarro Pérez, and Nikos Gorogiannis. 2014. A decision procedure for satisfiability in separation logic with inductive predicates. In *CSL-LICS*. 25:1–25:10.

Cristiano Calcagno and Dino Distefano. 2011. Infer: An Automatic Program Verifier for Memory Safety of C Programs. In *NFM*. 459–465.

Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dul ma Rodriguez. 2015. Moving Fast with Software Verification. In *NFM*. 3–11.

Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. 2006. Beyond Reachability: Shape Abstraction in the Presence of Pointer Arithmetic. In *SAS*. 182–203.

Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. 2011. Compositional Shape Analysis by Means of Bi-Abduction. *J. ACM* 58, 6 (2011), 26:1–26:66.

Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. 2007. Local Action and Abstract Separation Logic. In *LICS*. 366–378.

Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. 2001. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *APLAS*. 289–300.

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

Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. 2011. Tractable Reasoning in a Fragment of Separation Logic. In *CONCUR*. 235–249.

Bruno Courcelle and Joost Engelfriet. 2012. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*. Encyclopedia of mathematics and its applications, Vol. 138. Cambridge University Press.

Patrick Cousot and Radhia Cousot. 1979. Constructive versions of Tarski's fixed point theorems. *Pacific journal of Mathematics* 82, 1 (1979), 43–57.

Reinhard Diestel. 2016. *Graph Theory, 5th Edition*. Graduate texts in mathematics, Vol. 173. Springer.

Mnacho Echenim, Radu Iosif, and Nicolas Peltier. 2020a. The Bernays-Schönfinkel-Ramsey Class of Separation Logic with Uninterpreted Predicates. *ACM Trans. Comput. Log.* 21, 3 (2020), 19:1–19:46.

Mnacho Echenim, Radu Iosif, and Nicolas Peltier. 2020b. Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME hard. 73 (2020), 191–211.

Mnacho Echenim, Radu Iosif, and Nicolas Peltier. 2021. Decidable Entailments in Separation Logic with Inductive Definitions: Beyond Establishment. In *CSL (LIPICS)*.

Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. 2017. SPEN: A Solver for Separation Logic. In *NFM*. 302–309.

Mihaela Sighireanu et al. 2019. SL-COMP: Competition of Solvers for Separation Logic. In *TACAS*. 116–132.

Alexey Gotsman, Josh Berdine, Byron Cook, and Mooly Sagiv. 2007. Thread-modular shape analysis. In *PLDI*. 266–277.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2007. *Introduction to automata theory, languages, and computation, 3rd Edition*. Addison-Wesley.

Radu Iosif, Adam Rogalewicz, and Jirí Simácek. 2013. The Tree Width of Separation Logic with Recursive Definitions. In *CADE-24*. 21–38.

Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. 2014. Deciding Entailments in Inductive Separation Logic with Tree Automata. In *ATVA*. 201–218.

Samin S. Ishtiaq and Peter W. O'Hearn. 2001. BI as an Assertion Language for Mutable Data Structures. In *POPL*. 14–26.

Bart Jacobs, Jan Smans, Pieter Philippaerts, Frédéric Vogels, Willem Penninckx, and Frank Piessens. 2011. VeriFast: A Powerful, Sound, Predictable, Fast Verifier for C and Java. In *NFM*. 41–55.

Christina Jansen, Jens Katelaan, Christoph Matheja, Tho mas Noll, and Florian Zuleger. 2017. Unified Reasoning About Robustness Properties of Symbolic-Heap Separation Logic. In *ESOP*. 611–638.

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

Jens Katelaan, Christoph Matheja, and Florian Zuleger. 2019. Effective Entailment Checking for Separation Logic with Inductive Definitions. In *TACAS*. 319–336.

Jens Katelaan and Florian Zuleger. 2020. Beyond Symbolic Heaps: Deciding Separation Logic With Inductive Definitions. In *LPAR (EPiC Series in Computing, Vol. 73)*. EasyChair, 390–408.

Quang Loc Le, Makoto Tatsuta, Jun Sun, and Wei-Ngan Chin. 2017. A Decidable Fragment in Separation Logic with Inductive Predicates and Arithmetic. In *CAV*. 495–517.

Christoph Matheja. 2020. *Automated reasoning and randomization in separation logic.* Dissertation. RWTH Aachen University.

Peter Müller, Malte Schwerhoff, and Alexander J. Summers. 2017. Viper: A Verification Infrastructure for Permission-Based Reasoning. In *Dependable Software Systems Engineering*. 104–125.

Jens Pagel, Christoph Matheja, and Florian Zuleger. 2020. Complete Entailment Checking for Separation Logic with Inductive Definitions. *CoRR* abs/2002.01202 (2020). arXiv:2002.01202

Ruzica Piskac, Thomas Wies, and Damien Zufferey. 2013. Automating Separation Logic Using SMT. In *CAV*. 773–789.

Ruzica Piskac, Thomas Wies, and Damien Zufferey. 2014a. Automating Separation Logic with Trees and Data. In *CAV*. 711–728.

Ruzica Piskac, Thomas Wies, and Damien Zufferey. 2014b. GRASShopper - Complete Heap Verification with Mixed Specifications. In *TACAS*. 124–139.

John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *LICS*. 55–74.

Malte Schwerhoff and Alexander J. Summers. 2015. Lightweight Support for Magic Wands in an Automatic Verifier. In *ECOOP*. 614–638.

Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. 2016. Automated Mutual Explicit Induction Proof in Separation Logic. In *FM (LNCS, Vol. 9995)*. 659–676.

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

Aditya V. Thakur, Jason Breck, and Thomas W. Reps. 2014. Satisfiability modulo abstraction for separation logic with linked lists. In *SPIN*. 58–67.

Hongseok Yang. 2001. *Local Reasoning for Stateful Programs.* Ph. D. Dissertation. University of Illinois at Urbana-Champaign, Champaign, IL, USA. Advisor(s) Reddy, Uday S. AAI3023240.

A ELECTRONIC APPENDIX

A.1 Formal definition of substitution

For $\mathbf{y} = \langle y_1, \ldots, y_k \rangle$ and $\mathbf{z} = \langle z_1, \ldots, z_k \rangle$, the substitution $\phi[\mathbf{y}/\mathbf{z}]$ is defined by the table below. Since quantified variables can be renamed before performing a substitution, we assume w.l.o.g. that $\mathbf{y}$ contains no variables that are bound by a quantifier in ϕ.

ϕ	$\phi[\mathbf{y}/\mathbf{z}]$	
y_i	z_i	$1 \leq i \leq k$
u	u	$u \notin \mathbf{y}$
$\langle v_1, \ldots, v_n \rangle$	$\langle v_1[\mathbf{y}/\mathbf{z}], \ldots, v_n[\mathbf{y}/\mathbf{z}] \rangle$	
emp	**emp**	
$u \approx v$	$u[\mathbf{y}/\mathbf{z}] \approx v[\mathbf{y}/\mathbf{z}]$	
$u \not\approx v$	$u[\mathbf{y}/\mathbf{z}] \not\approx v[\mathbf{y}/\mathbf{z}]$	
$u \mapsto \mathbf{v}$	$u[\mathbf{y}/\mathbf{z}] \mapsto \mathbf{v}[\mathbf{y}/\mathbf{z}]$	
$\mathsf{pred}(\mathbf{x})$	$\mathsf{pred}(\mathbf{x}[\mathbf{y}/\mathbf{z}])$	
$\neg\psi$	$\neg(\psi[\mathbf{y}/\mathbf{z}])$	
$\psi \oplus \theta$	$\psi[\mathbf{y}/\mathbf{z}] \oplus \theta[\mathbf{y}/\mathbf{z}]$	$\oplus \in \{\wedge, \vee, \star, \mathrel{-\!\!\star}\}$
$\exists x.\, \psi$	$\exists x.\, \psi[\mathbf{y}/\mathbf{z}]$	$x \notin \mathbf{y}$
$\forall x.\, \psi$	$\forall x.\, \psi[\mathbf{y}/\mathbf{z}]$	$x \notin \mathbf{y}$

A.2 Proof of Lemma 4.4

Claim. For all predicates $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ and all states $\langle \mathfrak{s}, \mathfrak{h} \rangle$, we have

$$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(\mathbf{x}) \quad \text{implies} \quad \langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}.$$

Proof. Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state such that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(\mathbf{x})$ for some predicate $\mathsf{pred} \in \mathbf{Preds}(\Phi)$. The proof proceeds by strong mathematical induction on the number of rule applications needed to establish $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(\mathbf{x})$:

According to the semantics, there is a rule $(\mathsf{pred}(\mathbf{x}) \Leftarrow \phi) \in \Phi$, for some $\phi = \exists \mathbf{e}.\, \phi'$ with $\phi' = (y \mapsto \mathbf{z}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k}) \star \Pi$, Π pure (note that for $k = 0$ there are no recursive rule applications). Because of $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(\mathbf{x})$, there exists a stack $\mathfrak{s}' = \mathfrak{s}[\mathbf{e}/\mathbf{v}]$ for some locations $\mathbf{v}$ such that $\langle \mathfrak{s}', \mathfrak{h} \rangle \models \phi'$. Thus, there are heaps $\mathfrak{h}_0, \mathfrak{h}_1, \ldots, \mathfrak{h}_k$ with $\mathfrak{h} = \mathfrak{h}_0 \uplus \mathfrak{h}_1 \uplus \cdots \uplus \mathfrak{h}_k$, $\mathfrak{h}_0 \models (y \mapsto \mathbf{z})$ and $\langle \mathfrak{s}', \mathfrak{h}_i \rangle \models \mathsf{pred}_i(\mathbf{z_i})$ for all $1 \leq i \leq k$. By I.H., we have $\langle \mathfrak{s}', \mathfrak{h}_i \rangle \in \mathbf{GStates}$. Hence, $\mathsf{dangling}(\mathfrak{h}_0) \subseteq \mathsf{img}(\mathfrak{s}')$ for all $0 \leq i \leq k$. By the definition of establishment (see Section 3.4.3), we have $\mathfrak{s}'(\mathbf{e}) \subseteq \mathsf{dom}(\mathfrak{h}) \cup \mathfrak{s}(\mathbf{x}) \cup \{0\}$. Because of $\mathfrak{h}_i \subseteq \mathfrak{h}$ and $\mathsf{dangling}(\mathfrak{h}) = \mathsf{locs}(\mathfrak{h}) \setminus \mathsf{dom}(\mathfrak{h})$ we get that $\mathsf{dangling}(\mathfrak{h}_i) \subseteq \mathfrak{s}(\mathbf{x})$. Hence, $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$ because

$$\begin{aligned} \mathsf{dangling}(\mathfrak{h}) &= \mathsf{dangling}(\mathfrak{h}_0 \uplus \mathfrak{h}_1 \uplus \cdots \uplus \mathfrak{h}_k) \\ &\subseteq \mathsf{dangling}(\mathfrak{h}_0) \cup \mathsf{dangling}(\mathfrak{h}_1) \cup \cdots \cup \mathsf{dangling}(\mathfrak{h}_k) \\ &\subseteq \mathfrak{s}(\mathbf{x}) = \mathsf{img}(\mathfrak{s}) \,. \end{aligned}$$

□

A.3 Proof of Corollary 4.6

Claim. For all $\phi \in \mathbf{GSL}$ and all states $\langle \mathfrak{s}, \mathfrak{h} \rangle$, we have

$$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi \quad \text{implies} \quad \langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}.$$

Proof. Let $\phi \in \mathbf{GSL}$ be a guarded formula and let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state with $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$. The proof proceeds by structural induction on ϕ:

Case $\phi = \mathbf{emp}, \phi = x \approx y, x \not\approx y$**:** Clearly, there are no dangling pointers in the empty heap.

Case $\phi = x \mapsto \mathbf{y}$**.** Immediate because of $\mathsf{dangling}(\mathfrak{h}) \subseteq \mathfrak{s}(\mathbf{y}) \subseteq \mathsf{img}(\mathfrak{s})$.

Case $\phi = \mathsf{pred}(\mathbf{x})$**.** By Lemma 4.4.

Case $\phi = \phi_1 \star \phi_2$**.** Since $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$, there exist heaps $\mathfrak{h}_1, \mathfrak{h}_2$ with $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ and $\langle \mathfrak{s}, \mathfrak{h}_i \rangle \models_\Phi \phi_i$. By I.H., we have $\langle \mathfrak{s}, \mathfrak{h}_i \rangle \in \mathbf{GStates}$, i.e., $\mathsf{dangling}(\mathfrak{h}_i) \subseteq \mathsf{img}(\mathfrak{s})$. Thus, $\mathsf{dangling}(\mathfrak{h}) = \mathsf{dangling}(\mathfrak{h}_1 \uplus \mathfrak{h}_2) \subseteq \mathsf{dangling}(\mathfrak{h}_1) \cup \mathsf{dangling}(\mathfrak{h}_2) \subseteq \mathsf{img}(\mathfrak{s})$. Hence, $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$.

Case $\phi = \phi_1 \wedge \phi_2$**.** By the semantics of $\wedge$, this in particular means $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_1$. By I.H., it then follows that $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$. Notice that this case covers all standard conjunctions including the guarded negation, the guarded magic wand, and the guarded septraction.

Case $\phi = \phi_1 \vee \phi_2$**.** Assume w.l.o.g. that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_1$. By I.H., we have $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$. □

A.4 Proof of Lemma 4.5

Claim. Let $\phi \in \mathbf{GSL}$ be a guarded formula with $\mathsf{fvars}(\phi) = \mathbf{x}$. Then, for every state $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$, there are predicates $\mathsf{pred}_i \in \mathbf{Preds}(\Phi)$ and variables $\mathbf{z_i} \subseteq \mathbf{x}$ such that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \bigstar_{1 \le i \le k} \mathsf{pred}_i(\mathbf{z_i})$.

Proof. By structural induction on ϕ.

Case $\phi = \mathbf{emp}$**.** This case is immediate because, for $k = 0$, **emp** coincides with $\bigstar_{1 \le i \le k} \ldots$.

Case $\phi = x \approx y$**.** Since, in our semantics, the equality $x \approx y$ entails **emp**, we immediately obtain $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$. The case for disequalities $x \not\approx y$ is analogous.

Case $\phi = x \mapsto \mathbf{y}$**.** Since Φ is pointer-closed, there exists a predicate $\mathsf{pred} \in \Phi$ such that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(x \cdot \mathbf{y})$.

Case $\phi = \mathsf{pred}(\mathbf{x})$**.** Clearly, the claim holds.

Case $\phi = \phi_1 \star \phi_2$**.** Since $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$, there exist domain-disjoint heaps $\mathfrak{h}_1, \mathfrak{h}_2$ with $\mathfrak{h} = \mathfrak{h}_1 \cup \mathfrak{h}_2$ such that $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \phi_1$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi_2$. By the induction hypothesis, there exist predicate calls $\mathsf{pred}_{1,1}(\mathbf{x_1}), \ldots, \mathsf{pred}_{1,m}(\mathbf{x_m})$ and $\mathsf{pred}_{2,1}(\mathbf{y_1}), \ldots, \mathsf{pred}_{2,n}(\mathbf{y_n})$ such that

$$\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \bigstar_{1 \le i \le m} \mathsf{pred}_{1,i}(\mathbf{x_i}) \quad \text{and} \quad \langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \bigstar_{1 \le j \le n} \mathsf{pred}_{2,j}(\mathbf{y_j}).$$

The semantics of the separating conjunction $\star$ and the fact $\mathfrak{h} = \mathfrak{h}_1 \cup \mathfrak{h}_2$ then yield

$$\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \bigstar_{1 \le i \le m} \mathsf{pred}_{1,i}(\mathbf{x_i}) \star \bigstar_{1 \le j \le n} \mathsf{pred}_{2,j}(\mathbf{y_j}).$$

Case $\phi = \phi_1 \wedge \phi_2$**.** By the semantics of $\wedge$, this in particular means $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_1$. By the induction hypothesis, the claim then holds for $\langle \mathfrak{s}, \mathfrak{h} \rangle$ and ϕ_1. Notice that this case covers all standard conjunctions including the guarded negation, the guarded magic wand, and the guarded septraction.

Case $\phi = \phi_1 \vee \phi_2$**.** Assume w.l.o.g. that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_1$. By the induction hypothesis, the claim then holds for $\langle \mathfrak{s}, \mathfrak{h} \rangle$ and ϕ_1. □

A.5 Proof of Lemma 5.8

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

Proof. We show the stronger claim that, for all $x_1, x_2, x_3 \in \mathbf{Var}$, $w \in \mathcal{L}(\mathbf{G})$, and $N \in \mathbf{N}$, if $N \Rightarrow^+ w$, then there exists a model $\langle \mathfrak{s}, \mathfrak{h} \rangle$ of $N(x_1, x_2, x_3)$ with $\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = w$. We proceed by mathematical induction on the number m of $\Rightarrow$ steps in a (minimal-length) derivation $N \Rightarrow^+ w$.

If $m = 1$, $w = a_i$ for some $1 \le i \le n$ and there exists a rule $N \to a_i$. Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a model of $\exists a.\, (x_1 \mapsto \langle x_3, a \rangle) \star \mathsf{letter}_i(a) \star x_1 \approx x_2$. Note that this is a rule of the predicate N, so it holds that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi N(x_1, x_2, x_3)$. Moreover, $\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = a_i$.

If $m > 1$, there exists a rule $N \to AB$ such that $N \Rightarrow AB \Rightarrow^+ w$. Then, there exist words w_A, w_B with $w = w_A \cdot w_B$, $A \Rightarrow^+ w_A$, and $B \Rightarrow^+ w_B$.

Observe that both of the above derivations consist of strictly fewer than m steps. Now, fix some variables $l, m, r \in \mathbf{Var}$. By I.H., there exist states $(\mathfrak{s}_1, \mathfrak{h}_1)$ and $(\mathfrak{s}_2, \mathfrak{h}_2)$ such that

- $\langle \mathfrak{s}_1, \mathfrak{h}_1 \rangle \models_\Phi A(l, x_2, m)$ and $\mathrm{wordof}_A(\mathfrak{s}_1, \mathfrak{h}_1, x_2, m) = w_A$ as well as
- $\langle \mathfrak{s}_2, \mathfrak{h}_2 \rangle \models_\Phi B(r, m, x_3)$ and $\mathrm{wordof}_B(\mathfrak{s}_2, \mathfrak{h}_2, m, x_3) = w_B$.

Assume w.l.o.g. that (1) $\mathrm{dom}(\mathfrak{s}_1) \cap \mathrm{dom}(\mathfrak{s}_2) = m$, (2) $\mathfrak{s}_1(m) = \mathfrak{s}_2(m)$, (3) and $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \neq \bot$; if this is not the case, simply replace $\langle \mathfrak{s}_1, \mathfrak{h}_1 \rangle$ and $\langle \mathfrak{s}_2, \mathfrak{h}_2 \rangle$ with appropriate isomorphic models. We choose some location $k \in \mathbf{Loc}$ such that $k \notin \mathrm{locs}(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$.

Let $\mathfrak{s} \triangleq \mathfrak{s}_1 \cup \mathfrak{s}_2 \cup \{x_1 \mapsto k\}$ and $\mathfrak{h} \triangleq \mathfrak{h}_1 \cup \mathfrak{h}_2 \cup \{k \mapsto \langle \mathfrak{s}(l), \mathfrak{s}(r) \rangle\}$. We obtain $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi (x_1 \mapsto \langle l, r \rangle) \star A(l, x_2, m) \star B(r, m, x_3)$ and thus also $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \exists \langle l, r, m \rangle \,.\, (x_1 \mapsto \langle l, r \rangle) \star A(l, x_2, m) \star B(r, m, x_3)$. By definition of Φ, we conclude $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi N(x_1, x_2, x_3)$. Furthermore, observe that

$$\begin{aligned} \mathrm{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) &= \mathrm{wordof}_A(\mathfrak{s}, \mathfrak{h}_1, x_2, m) \cdot \mathrm{wordof}_N(\mathfrak{s}, \mathfrak{h}_2, m, x_3) \\ &= w_A \cdot w_B = w. \end{aligned}$$

□

A.6 Proof of Lemma 5.9

Claim (Soundness of the encoding). Let $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ and let Φ be the corresponding SID encoding. Let $x_1, x_2, x_3 \in \mathbf{Var}$ and let $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathbf{S}(x_1, x_2, x_3)$. Then $\mathrm{wordof}_{\mathbf{S}}(\mathfrak{s}, \mathfrak{h}, x_2, x_3) \in \mathcal{L}(\mathbf{G})$.

Proof. We show the stronger claim that for all $x_1, x_2, x_3 \in \mathbf{Var}$, all models $\langle \mathfrak{s}, \mathfrak{h} \rangle$, and all $N \in \mathbf{N}$, if $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi N(x_1, x_2, x_3)$ then $N \Rightarrow^+ \mathrm{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3)$. Observe that $\mathfrak{h}$ is a tree overlaid with a linked list. We proceed by mathematical induction on the height h of the tree in $\mathfrak{h}$.

If $h = 0$, then Φ contains a rule

$$N(x_1, x_2, x_3) \Leftarrow \exists a. \, (x_1 \mapsto \langle x_3, a \rangle) \star \mathrm{letter}_k(a) \star x_1 \approx x_2$$

whose right-hand side is satisfied by $\langle \mathfrak{s}, \mathfrak{h} \rangle$. Then $\mathrm{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = a_k$. By definition of Φ, this implies $N \to a_k \in \mathbf{R}_1 \cup \mathbf{R}_2$ and, consequently, $N \Rightarrow a_k$. Hence, $N \Rightarrow^+ a_k = \mathrm{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3)$.

If $h > 0$, there exists a rule $(N(x_1, x_2, x_3) \Leftarrow \psi) \in \Phi$ such that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi$, where ψ is of the form

$$\exists l, r, m. \, (x_1 \mapsto \langle l, r \rangle) \star A(l, x_2, m) \star B(r, m, x_3).$$

Recall that by definition of Φ, we have $N \to AB \in \mathbf{R}_1 \cup \mathbf{R}_2 \qquad (\dagger)$.

By the semantics of $\exists$ and $\star$ there then are a stack $\mathfrak{s}'$ with $\mathrm{dom}(\mathfrak{s}') = \mathrm{dom}(\mathfrak{s}) \cup \{l, r, m\}$ and heaps $\mathfrak{h}_0, \mathfrak{h}_A, \mathfrak{h}_B$ such that $\mathfrak{h} = \mathfrak{h}_0 \uplus \mathfrak{h}_A \uplus \mathfrak{h}_B$, $\langle \mathfrak{s}', \mathfrak{h}_0 \rangle \models_\Phi (x_1 \mapsto \langle l, r \rangle)$, $\langle \mathfrak{s}', \mathfrak{h}_A \rangle \models_\Phi A(l, x_2, m)$, and $\langle \mathfrak{s}', \mathfrak{h}_B \rangle \models_\Phi B(r, m, x_3)$. Note that the height of the trees in $\mathfrak{h}_A$ and $\mathfrak{h}_B$ is at most $h - 1$, so we can apply the induction hypotheses for these models to obtain

$$A \Rightarrow^+ \mathrm{wordof}_A(\mathfrak{s}', \mathfrak{h}_1, x_2, m) \qquad \text{and} \qquad B \Rightarrow^+ \mathrm{wordof}_B(\mathfrak{s}', \mathfrak{h}_2, m, x_3).$$

Together with $(\dagger)$, we derive

$$\begin{aligned} N &\Rightarrow AB \\ &\Rightarrow^+ \mathrm{wordof}_A(\mathfrak{s}', \mathfrak{h}_1, x_2, m) \cdot \mathrm{wordof}_B(\mathfrak{s}', \mathfrak{h}_2, m, x_3) \\ &= \mathrm{wordof}_N(\mathfrak{s}', \mathfrak{h}, x_2, x_3) \\ &= \mathrm{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3). \end{aligned}$$

□

A.7 Proof of Theorem 5.10

Claim. The satisfiability problem for the fragment $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrm{t})$ is undecidable.

Proof. Let Φ be the encoding of the CFGs $\mathbf{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle$ and $\mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$ as described in Section 5.1. Moreover, consider the $\mathbf{SL}_{\mathrm{btw}}(\wedge, \star, \mathrm{t})$ formula

$$\phi \triangleq (\mathrm{S}_1(a,x,y) \star \mathrm{t}) \wedge (\mathrm{S}_2(b,x,y) \star \mathrm{t}).$$

We claim that ϕ is satisfiable iff $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$; both implications are proven separately:

If ϕ is satisfiable, there exists a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ with $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$. By Lemma 5.6, there exist heaps $\mathfrak{h}_{w_1}, \mathfrak{h}_{w_2} \subseteq \mathfrak{h}$ such that $\mathsf{wordof}_{\mathrm{S}_i}(\mathfrak{s}, \mathfrak{h}, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_i}, x, y)$ for $i \in \{1, 2\}$.

Observe that both $\langle \mathfrak{s}, \mathfrak{h}_{w_1} \rangle \models_\Phi \mathsf{word}(x,y)$ and $\langle \mathfrak{s}, \mathfrak{h}_{w_2} \rangle \models_\Phi \mathsf{word}(x,y)$. Hence, $\mathfrak{h}_{w_1} = \mathfrak{h}_{w_2}$ and thus

$$w \triangleq \mathsf{wordof}_{\mathrm{S}_2}(\mathfrak{s}, \mathfrak{h}, x, y) = \mathsf{wordof}_{\mathrm{S}_1}(\mathfrak{s}, \mathfrak{h}, x, y).$$

By Lemma 5.9, we have $w \in \mathcal{L}(\mathbf{G}_1)$ and $w \in \mathcal{L}(\mathbf{G}_2)$, i.e., $w \in \mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2)$.

Conversely, assume $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$. Then there exists a word $w \in \mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2)$. By Lemma 5.8, there exist states $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ with $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathrm{S}_1(a,x,y)$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \mathrm{S}_2(b,x,y)$. Let $\mathfrak{h}_{w_1} \subseteq \mathfrak{h}_1, \mathfrak{h}_{w_2} \subseteq \mathfrak{h}_2$ be the unique heaps with $\mathsf{wordof}_{\mathrm{S}_1}(\mathfrak{s}, \mathfrak{h}_1, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_1}, x, y) = w = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_2}, x, y) = \mathsf{wordof}_{\mathrm{S}_2}(\mathfrak{s}, \mathfrak{h}_2, x, y)$.

Observe that $\langle \mathfrak{s}, \mathfrak{h}_{w_1} \rangle \cong \langle \mathfrak{s}, \mathfrak{h}_{w_2} \rangle$ (see Definition 3.4). Consequently, we can reason about an isomorphic state in which we replace $\mathfrak{h}_2$ by a heap that contains $\mathfrak{h}_{w_1}$ (as opposed to $\mathfrak{h}_{w_2}$) as sub-heap and is otherwise disjoint from $\mathfrak{h}_1$. That is, there exists a heap $\mathfrak{h}_2'$ such that $(\mathfrak{s}, \mathfrak{h}_2) \cong (\mathfrak{s}, \mathfrak{h}_2')$, $\mathsf{locs}(\mathfrak{h}_2') \cap \mathsf{locs}(\mathfrak{h}_1) = \mathsf{locs}(\mathfrak{h}_{w_1})$, and $\mathsf{wordof}_{\mathrm{S}_2}(\mathfrak{s}, \mathfrak{h}_2', x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_1}, x, y) = w$. In particular, $\langle \mathfrak{s}, \mathfrak{h}_2' \rangle \models_\Phi \mathrm{S}_2(b,x,y)$, because isomorphic states satisfy the same formulas (Lemma 3.5). Now let $\mathfrak{h} \triangleq \mathfrak{h}_1 \cup \mathfrak{h}_2'$ be the (non-disjoint) union of $\mathfrak{h}_1$ and $\mathfrak{h}_2'$. Since $\mathfrak{h}_1 \subseteq \mathfrak{h}$ and $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathrm{S}_1(a,x,y)$, we have $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathrm{S}_1(a,x,y) \star \mathrm{t}$; and similarly, since $\mathfrak{h}_2' \subseteq \mathfrak{h}$ and $\langle \mathfrak{s}, \mathfrak{h}_2' \rangle \models_\Phi \mathrm{S}_2(a,x,y)$, we have that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathrm{S}_2(b,x,y)$. Consequently, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$. □

A.8 Proof of Lemma 5.13

Claim. Let $\mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$ be the CFG fixed in Section 5.1. Moreover, let Φ be the corresponding SID encoding, $\mathsf{word}_2(x,y) \triangleq (\mathsf{word}(x,y) \circledast \mathrm{S}_2(a,x,y)) \circledast \mathrm{S}_2(a,x,y)$, and let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state. Then $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{word}_2(x,y)$ iff $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{word}(x,y)$ and $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) \in \mathcal{L}(\mathbf{G}_2)$.

Proof. Assume $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{word}_2(x,y)$. By the semantics of $\circledast$, there exists a heap $\mathfrak{h}_1$ with $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathsf{word}(x,y) \circledast \mathrm{S}_2(a,x,y)$ such that $\langle \mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1 \rangle \models_\Phi \mathrm{S}_2(a,x,y)$. Observe that $\mathfrak{h}_1$ contains precisely the inner nodes of $\langle \mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1 \rangle$, i.e., everything *except* the part of the state that induces the word. Consequently, $\mathfrak{h}$ is the part of the state that induces the word, i.e., $\mathsf{wordof}_{\mathrm{S}_2}(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y)$ and $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{word}(x,y)$. Lemma 5.9 then yields $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) \in \mathcal{L}(\mathbf{G}_2)$.

Conversely, assume a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be such that $w \triangleq \mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) \in \mathcal{L}(\mathbf{G}_2)$. As a consequence of Lemma 5.8, there exists a heap $\mathfrak{h}_1$ with $\langle \mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1 \rangle \models_\Phi \mathrm{S}_2(a,x,y)$. Because $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{word}(x,y)$ by assumption, the semantics of $\circledast$ yields that $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathsf{word}(x,y) \circledast \mathrm{S}_2(a,x,y)$. Because $\langle \mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1 \rangle \models_\Phi \mathrm{S}_2(a,x,y)$, we obtain by the semantics of $\circledast$ that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi (\mathsf{word}(x,y) \circledast \mathrm{S}_2(a,x,y)) \circledast \mathrm{S}_2(a,x,y)$. □

A.9 Proof of Theorem 5.14

Claim. The satisfiability problem of $\mathbf{SL}_{\mathrm{btw}}(\circledast)$ is undecidable.

Proof. We claim that $\psi \triangleq \mathsf{word}_2(x,y) \circledast \mathrm{S}_1(a,x,y)$ is satisfiable iff $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$.

Assume ψ is satisfiable, i.e., there exists a state $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \psi$. By the semantics of $\circledast$, there exists a heap $\mathfrak{h}_0 \subseteq \mathfrak{h}$ with $\mathfrak{h}_0 \models_\Phi \mathsf{word}_2(x,y)$ and $\langle \mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_0 \rangle \models_\Phi \mathrm{S}_1(a,x,y)$. As $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y) \in \mathcal{L}(\mathbf{G}_2)$, by Lemma 5.13, we have that $\langle \mathfrak{s}, \mathfrak{h}_0 \rangle \models_\Phi \mathsf{word}(x,y)$. It follows that $\mathfrak{h}_0$ is the unique sub-heap of $\mathfrak{h} \uplus \mathfrak{h}_0$ with $\mathsf{wordof}_{\mathrm{S}_1}(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_0) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y)$. By Lemma 5.9, $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y) \in \mathcal{L}(\mathbf{G}_1)$. Together with Lemma 5.13, we thus have that $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}_0, x, y) \in \mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2)$.

Conversely, assume there exists a word $w \in \mathcal{L}(\mathrm{G}_1) \cap \mathcal{L}(\mathrm{G}_2)$. As shown in the proof of Theorem 5.10, there exist states $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle, \langle \mathfrak{s}, \mathfrak{h} \rangle$ with $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathrm{S}_1(a, x, y)$, $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \mathrm{S}_2(b, x, y)$ and $\mathrm{locs}(\mathfrak{h}_1) \cap \mathrm{locs}(\mathfrak{h}_2) = \mathrm{locs}(\mathfrak{h})$ such that

$$\mathrm{wordof}_{\mathrm{S}_1}(\mathfrak{s}, \mathfrak{h}_1, x, y) = \mathrm{wordof}_{\mathrm{S}_2}(\mathfrak{s}, \mathfrak{h}_2, x, y) = \mathrm{letters}(\mathfrak{s}, \mathfrak{h}, x, y) = w \,.$$

Let $\mathfrak{h}_0 \subseteq \mathfrak{h}_1$ be the sub-heap of $\mathfrak{h}_1$ with $\mathfrak{h} \uplus \mathfrak{h}_0 = \mathfrak{h}_1$. By Lemma 5.13, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathrm{word}_2(x, y)$. Consequently, $\langle \mathfrak{s}, \mathfrak{h}_0 \rangle \models_\Phi \psi$, i.e., ψ is satisfiable. □

A.10 Proof of Lemma 7.10

Claim. For all $\mathbf{l} \subseteq \mathbf{Loc}$, every Φ-forest has a unique $\mathbf{l}$-split $\mathrm{split}(\mathfrak{f}, \mathbf{l})$.

Proof. Let $\mathfrak{f}$ be a Φ-forest with $\mathrm{graph}(\mathfrak{f}) = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$. Moreover, consider the graph

$$\mathcal{G} \triangleq \langle V_\mathcal{G}, E_\mathcal{G} \setminus \{\langle a, b \rangle \mid a \in \mathbf{Loc}, b \in \mathbf{l}\} \rangle \,.$$

Since $\mathrm{graph}(\mathfrak{f})$ is a forest and $\mathcal{G} \subseteq \mathrm{graph}(\mathfrak{f})$, $\mathcal{G}$ is a forest, i.e., all connected components $C_1, \ldots, C_k$ of $\mathcal{G}$ are trees. Formally, let $\mathrm{locs}(C_i)$ be all locations in C_i and let $\mathrm{succ}_{C_i}(a)$ be the largest set of locations such that every edge in $\{a\} \times \mathrm{succ}_{C_i}(a)$ appears in C_i. We then define:

$$\mathfrak{t}_i \triangleq \left\{ a \mapsto \left\langle \mathrm{succ}_{C_i}(a), \mathrm{rule}_\mathfrak{f}(a) \right\rangle \mid a \in \mathrm{locs}(C_i) \right\}, \qquad \text{(tree induced by component } C_i\text{)}$$

$$\bar{\mathfrak{f}} \triangleq \{\mathfrak{t}_1, \ldots, \mathfrak{t}_n\} \,. \qquad \text{(}\Phi\text{-forest induced by the connected components)}$$

By construction, the forest $\bar{\mathfrak{f}}$ is an $\mathbf{l}$-split of $\mathfrak{f}$. Moreover, since every $\mathbf{l}$-split must have the same domain and the same rule instances as $\mathfrak{f}$ and because every connected component gives rise to a single Φ-tree, the $\mathbf{l}$-split $\mathrm{split}(\mathfrak{f}, \{l\}) = \bar{\mathfrak{f}}$ is unique. □

A.11 Proof of Lemma 7.20

Let $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$ and ϕ be a quantifier free $\mathbf{SL}$ formula with $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$. Moreover, let $\mathbf{v} \in (\mathbf{Loc} \setminus (\mathrm{dom}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})))^*$ be a repetition-free sequence of locations. Then, for every set $\mathbf{a} \triangleq \{a_1, \ldots, a_{|\mathbf{v}|}\}$ of fresh variables (i.e., $\mathbf{a} \cap \mathrm{dom}(\mathfrak{s}) = \emptyset$), we have $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \forall \mathbf{a}.\, \phi[\mathbf{v}/\mathbf{a}]$.

Proof sketch. Let $\mathbf{w} \in (\mathbf{Loc} \setminus (\mathrm{dom}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})))^*$ be a repetition-free sequence of locations with $|\mathbf{v}| = |\mathbf{w}|$. We note that $\mathrm{dangling}(\mathfrak{h}) \subseteq \mathrm{img}(\mathfrak{s})$ because of $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$. Hence, neither $\mathbf{v}$ nor $\mathbf{w}$ intersect with $\mathrm{locs}(\mathfrak{h})$ or $\mathrm{img}(\mathfrak{s})$. Thus, it follows that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi[\mathbf{v}/\mathbf{w}]$. Since $\mathbf{w}$ was arbitrary, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \forall \mathbf{a}.\, \phi[\mathbf{v}/\mathbf{a}]$ by the semantics of $\forall$. □

A.12 Proof of Lemma 7.19

Claim. Let $\mathfrak{t}$ be a Φ-tree. Then, $\langle \_, \mathrm{heap}(\mathfrak{t}) \rangle \models_\Phi \mathrm{project}^{\mathbf{Loc}}(\mathfrak{t})$ (where _ denotes an arbitrary stack).

Proof. We prove the claim by mathematical induction on the height of $\mathfrak{t}$.

By construction, $\mathfrak{t}$ has a root $r = \mathrm{root}(\mathfrak{t})$ with $m \geq 0$ successors that are the root of subtrees $\mathfrak{t}_1, \ldots, \mathfrak{t}_m$. Hence, there is a rule instance $\mathrm{rule}_\mathfrak{t}(r)$ (up to applying commutativity of $\star$) of the form[15]

$$\mathrm{rootpred}(\mathfrak{t}) \Leftarrow (a \mapsto \mathbf{b}) \star \left(\bigstar_{1 \leq i \leq m} \mathrm{rootpred}(\mathfrak{t}_i)\right) \star \bigstar \mathrm{holepreds}_\mathfrak{t}(r),$$

By the semantics of $\star$ and $\mathrel{-\!\!\star}$, we have

$$\langle \_, \{a \mapsto \mathbf{b}\} \rangle \models_\Phi \left(\left(\bigstar_{1 \leq i \leq m} \mathrm{rootpred}(\mathfrak{t}_i)\right) \star \bigstar \mathrm{holepreds}_\mathfrak{t}(r)\right) \mathrel{-\!\!\star} \mathrm{rootpred}(\mathfrak{t})$$

[15] For $\mathrm{height}(\mathfrak{t}) = 0$, there are no successors, i.e., $\bigstar_{1 \leq i \leq m} \mathrm{rootpred}(\mathfrak{t}_i)$ is equivalent to $\mathbf{emp}$.

We apply the I.H. for each tree t_i and obtain

$$\langle \_, \mathsf{heap}(\mathsf{t}_i)\rangle \models_\Phi (\bigstar \mathsf{allholepreds}(\mathsf{t}_i)) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathsf{t}_i).$$

On the level of heaps, we have $\mathsf{heap}(\mathsf{t}) = \{a \mapsto \mathbf{b}\} \uplus \mathsf{heap}(\mathsf{t}_1) \uplus \ldots \uplus \mathsf{heap}(\mathsf{t}_m)$. Applying the semantics of $\star$ and the definition of ψ_i then yields

$$\langle \_, \mathsf{heap}(\mathsf{t})\rangle \models_\Phi \Big(\big((\bigstar_{1\le i\le m}\mathsf{rootpred}(\mathsf{t}_i)) \star \bigstar \mathsf{holepreds}_{\mathsf{t}}(r)\big) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathsf{t})\Big) \star \bigstar_{1\le i\le m}((\bigstar \mathsf{allholepreds}(\mathsf{t}_i)) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathsf{t}_i)).$$

Applying Lemma 7.18 m times, we then obtain

$$\begin{aligned}\langle \_, \mathsf{heap}(\mathsf{t})\rangle &\models_\Phi \big((\bigstar_{1\le i\le m}(\bigstar \mathsf{allholepreds}(\mathsf{t}_i))) \star \bigstar \mathsf{holepreds}_{\mathsf{t}}(r)\big) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathsf{t})\\ &= \big(\bigstar \mathsf{allholepreds}(\mathsf{t})\big) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathsf{t}). && \text{(Def. of allholepreds(t))}\end{aligned}$$

□

A.13 Proof of Lemma 7.25

Claim (Soundness of stack-projection). Let $\langle \mathfrak{s}, \mathfrak{h}\rangle \in \mathbf{GStates}$. Moreover, let $\mathfrak{f}$ be a Φ-forest with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$. Then, we have $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f})$.

Proof. Let $\mathfrak{f} = \{\mathsf{t}_1, \ldots, \mathsf{t}_k\}$ and $\phi = \bigstar_{1\le i\le k} \mathsf{project}^{\mathbf{Loc}}(\mathsf{t}_i)$. By Lemma 7.19, we know for each i that $\langle \mathfrak{s}, \mathsf{heap}(\mathsf{t}_i)\rangle \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathsf{t}_i)$. By definition, $\mathsf{heap}(\mathfrak{f}) = \mathsf{heap}(\mathsf{t}_1) \uplus \cdots \uplus \mathsf{heap}(\mathsf{t}_k)$. Applying the semantics of $\star$ then yields $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi$ (†).

Let $\mathbf{w} = \mathsf{locs}(\phi) \cap (\mathsf{dom}(\mathfrak{f}) \setminus \mathsf{img}(\mathfrak{s}))$ be the locations that occur in ϕ and are allocated in $\mathsf{heap}(\mathfrak{f})$ but are not the value of any stack variable, and let $\mathbf{v} = \mathsf{locs}(\phi) \setminus (\mathsf{img}(\mathfrak{s}) \cup \mathsf{dom}(\mathfrak{f}))$ be the locations that occur in the formula ϕ and are neither allocated nor the value of any stack variable.

Then, we have

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi[\mathsf{dom}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{v} \cdot \mathbf{w}/\mathsf{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{a} \cdot \mathbf{e}],$$

where $\mathbf{e} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{w}|}\rangle$ and $\mathbf{a} \triangleq \langle a_1, a_2, \ldots, a_{|\mathbf{v}|}\rangle$ denote some disjoint sets of fresh variables.

The claim then follows by the implications below:

$$\begin{aligned}
&\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi && \text{(by (†))}\\
\Longrightarrow\ &\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi[\mathsf{dom}(\mathfrak{s}^{-1}_{\max})/\mathsf{img}(\mathfrak{s}^{-1}_{\max})] && \text{(stack–heap semantics)}\\
\Longrightarrow\ &\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi[\mathsf{dom}(\mathfrak{s}^{-1}_{\max})/\mathsf{img}(\mathfrak{s}^{-1}_{\max})][\mathbf{v} \cdot \mathbf{w}/\mathbf{a} \cdot \mathbf{e}][\mathbf{a} \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{w}] && (\mathbf{a} \text{ and } \mathbf{e} \text{ are disjoint sets of fresh variables})\\
\Longrightarrow\ &\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi[\mathsf{dom}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{v} \cdot \mathbf{w}/\mathsf{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{a} \cdot \mathbf{e}][\mathbf{a} \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{w}] && (\mathbf{v} \cap \mathsf{img}(\mathfrak{s}) = \emptyset \text{ and } \mathbf{w} \cap \mathsf{img}(\mathfrak{s}) = \emptyset)\\
\Longrightarrow\ &\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \forall \mathbf{a}.\, \phi[\mathsf{dom}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{v} \cdot \mathbf{w}/\mathsf{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{a} \cdot \mathbf{e}][\mathbf{e}/\mathbf{w}] && \text{(by Lemma 7.20)}\\
\Longrightarrow\ &\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi[\mathsf{dom}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{v} \cdot \mathbf{w}/\mathsf{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{a} \cdot \mathbf{e}] && \text{(semantics of } \exists)\\
\Longrightarrow\ &\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f}) && \square
\end{aligned}$$

A.14 Proof of Lemma 7.35

Before we prove Lemma 7.35, we need two auxiliary results.

Lemma A.1. *Let $\mathfrak{s}$ be a stack and let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests with $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \neq \bot$. Then,* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1 \uplus \mathfrak{f}_2) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$.

Proof. We set $\mathfrak{f}_0 \triangleq \mathfrak{f}_1 \uplus \mathfrak{f}_2$. For $i \in \{0,1,2\}$, let $\phi_i = \bigstar_{\mathfrak{t} \in \mathfrak{f}_i} \mathsf{project}^{\mathbf{Loc}}(\mathfrak{t})$, let $\mathbf{w_i} = \mathsf{locs}(\phi_i) \cap (\mathsf{dom}(\mathfrak{f}_i) \setminus \mathsf{img}(\mathfrak{s}))$ be the locations that occur in the formula ϕ and are allocated in $\mathsf{heap}(\mathfrak{f}_i)$ but are not the value of any stack variable, and let $\mathbf{v_i} = \mathsf{locs}(\phi_i) \setminus (\mathsf{img}(\mathfrak{s}) \cup \mathsf{dom}(\mathfrak{f}_i))$ be the locations that occur in the formula ϕ_i and are neither allocated nor the value of any stack variable. Then, we have

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) = \exists \mathbf{e_i}.\, \forall \mathbf{a_i}.\, \phi_i[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_i} \cdot \mathbf{w_i} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_i} \cdot \mathbf{e_i}],$$

where $\mathbf{e_i} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{w_i}|} \rangle$ and $\mathbf{a_i} \triangleq \langle a_1, a_2, \ldots, a_{|\mathbf{v_i}|} \rangle$ denote some disjoint sets of fresh variables. Because of $\mathfrak{f}_0 = \mathfrak{f}_1 \uplus \mathfrak{f}_2$ we have $\mathbf{w_0} = \mathbf{w_1} \cdot \mathbf{w_2}$ and hence can choose $\mathbf{e_0}$ such that $\mathbf{e_0} = \mathbf{e_1} \cdot \mathbf{e_2}$.

We now argue that we can find sequences of variables $\mathbf{u_i} \subseteq \mathbf{a_0} \cup \mathbf{e_{3-i}}$, for $i = 1, 2$, such that

$$\begin{aligned}\phi_0[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_0} \cdot \mathbf{w_0} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_0} \cdot \mathbf{e_0}] \equiv \\ \phi_1[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_1} \cdot \mathbf{w_2} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_1} \cdot \mathbf{e_1}][\mathbf{a_1}/\mathbf{u_1}] \star \\ \phi_2[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_2} \cdot \mathbf{w_2} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_2} \cdot \mathbf{e_2}][\mathbf{a_2}/\mathbf{u_2}] \quad (*)\end{aligned}$$

We consider a location $l \in \mathbf{v_i}$ for $i \in \{1,2\}$. If $l \in \mathsf{dom}(\mathfrak{f}_{3-i})$, then there is a variable $e \in \mathbf{e_{3-i}}$ which replaces l in the projection $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_0)$. If $l \notin \mathsf{dom}(\mathsf{heap}(\mathfrak{f}_{3-i}))$, then there is a variable $a \in \mathbf{a_0}$ which replaces l in the projection $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_0)$. Hence, we can choose sequences of variables $\mathbf{u_i} \subseteq \mathbf{a_0} \cup \mathbf{e_{3-i}}$, for $i \in \{1,2\}$, such that the following holds for all $l \in \mathbf{v_i}$ and $i \in \{1,2\}$:

$$l[\mathbf{v_0} \cdot \mathbf{w_0} / \mathbf{a_0} \cdot \mathbf{e_0}] = l[\mathbf{v_i}/\mathbf{a_i}][\mathbf{a_i}/\mathbf{u_i}]$$

The above then implies (*). □

Lemma A.2. *Let $\mathfrak{s}$ be a stack and let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests with $\mathfrak{f}_1 \blacktriangleright \mathfrak{f}_2$. Then,* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \rhd \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$.

Proof. Since $\mathfrak{f}_1 \blacktriangleright \mathfrak{f}_2$, there exists a forest $\mathfrak{f}$ and trees $\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}$ such that

(1) $\mathfrak{f}_1 = \mathfrak{f} \cup \{\mathfrak{t}_1, \mathfrak{t}_2\}$,
(2) $\mathfrak{f}_2 = \mathfrak{f} \cup \{\mathfrak{t}\}$,
(3) $\mathsf{rootpred}(\mathfrak{t}_1) \in \mathsf{allholepreds}(\mathfrak{t}_2)$,
(4) $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{rootpred}(\mathfrak{t}_2)$, and
(5) $\mathsf{allholepreds}(\mathfrak{t}) = \mathsf{allholepreds}(\mathfrak{t}_1) \cup (\mathsf{allholepreds}(\mathfrak{t}_2) \setminus \{\mathsf{rootpred}(\mathfrak{t}_1)\})$.

Intuitively, this implies that the projections of $\mathfrak{t}_1$ and $\mathfrak{t}_2$ can be merged into the projection of $\mathfrak{t}$ via the generalized modus ponens (see Lemma 7.18). In the following we make this claim formal.

For $i \in \{1,2\}$, let $\phi_i = \bigstar_{\mathfrak{t} \in \mathfrak{f}_i} \mathsf{project}^{\mathbf{Loc}}(\mathfrak{t})$, let $\mathbf{w_i} = \mathsf{locs}(\phi_i) \cap \mathsf{dom}(\mathfrak{f}_i) \setminus \mathsf{img}(\mathfrak{s})$ be the locations that occur in the formula ϕ and are allocated in $\mathsf{heap}(\mathfrak{f}_i)$ but are not the value of any stack variable, and let $\mathbf{v_i} = \mathsf{locs}(\phi_i) \setminus (\mathsf{img}(\mathfrak{s}) \cup \mathsf{dom}(\mathfrak{f}_i))$ be the locations that occur in the formula ϕ_i and are neither allocated nor the value of any stack variable. Then, we have

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) = \exists \mathbf{e_i}.\, \forall \mathbf{a_i}.\, \phi_i[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_i} \cdot \mathbf{w_i} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_i} \cdot \mathbf{e_i}],$$

where $\mathbf{e_i} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{w_i}|} \rangle$ and $\mathbf{a_i} \triangleq \langle a_1, a_2, \ldots, a_{|\mathbf{v_i}|} \rangle$ denote some disjoint sets of fresh variables. We note that

$$\phi_1 \equiv \bigstar_{\mathfrak{t}' \in \mathfrak{f}} \mathsf{project}^{\mathbf{Loc}}(\mathfrak{t}') \star (\bigstar \mathsf{allholepreds}(\mathfrak{t}_1)) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}_1) \star (\bigstar \mathsf{allholepreds}(\mathfrak{t}_2)) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}_2)$$

and

$$\phi_2 \equiv \bigstar_{\mathfrak{t}' \in \mathfrak{f}} \mathsf{project}^{\mathbf{Loc}}(\mathfrak{t}') \star (\bigstar \mathsf{allholepreds}(\mathfrak{t}_1) \star \bigstar (\mathsf{allholepreds}(\mathfrak{t}_2) \setminus \{\mathsf{rootpred}(\mathfrak{t}_1)\})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}_2).$$

In particular, we have $\mathrm{locs}(\phi_2) \subseteq \mathrm{locs}(\phi_1)$. Hence, we can assume without loss of generality that $\mathbf{e_2} \subseteq \mathbf{e_1}$ and $\mathbf{a_2} \subseteq \mathbf{a_1}$. By (*), we have

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \equiv \exists \mathbf{e_1}.\, \forall \mathbf{a_1}.\, \bigstar_{t' \in \mathfrak{f}} \mathsf{project}^{\mathbf{Loc}}(t') \star (\bigstar \mathsf{allholepreds}(t_1)) \mathrel{-\!\!\star} \mathsf{rootpred}(t_1) \star (\bigstar \mathsf{allholepreds}(t_2)) \mathrel{-\!\!\star} \mathsf{rootpred}(t_2)[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_i} \cdot \mathbf{w_i} / \mathrm{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_i} \cdot \mathbf{e_i}],$$

and

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \equiv \exists \mathbf{e_1}.\, \forall \mathbf{a_1}.\, \bigstar_{t' \in \mathfrak{f}} \mathsf{project}^{\mathbf{Loc}}(t') \star (\bigstar \mathsf{allholepreds}(t_1) \star \bigstar (\mathsf{allholepreds}(t_2) \setminus \{\mathsf{rootpred}(t_1)\})) \mathrel{-\!\!\star} \mathsf{rootpred}(t_2)[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_i} \cdot \mathbf{w_i} / \mathrm{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_i} \cdot \mathbf{e_i}],$$

We now recognize that $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ can be obtained from $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1)$ by applying the generalized modus ponens rule and dropping the quantified variables $\mathbf{e_1} \setminus \mathbf{e_2}$ and $\mathbf{a_1} \setminus \mathbf{a_2}$, which is supported by our rewriting rules (see Fig. 10) because these variables do not appear in ϕ_2. □

Claim (Lemma 7.35). Let $\mathfrak{s}$ be a stack and let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests such that $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \neq \bot$. Then,

$$\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2 \quad \text{implies} \quad \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2).$$

Proof. The claim is an immediate consequence of Lemmas A.1 and A.2. □

A.15 Proof of Theorem 7.39

Before we prove Theorem 7.39, we need two auxiliary results.

Lemma A.3. *Let $\mathfrak{s}$ be a stack, let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests with $\mathrm{dom}(\mathfrak{f}_1) \cap \mathrm{dom}(\mathfrak{f}_2) \cap \mathrm{img}(\mathfrak{s}) = \emptyset$, and let $\chi \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$. Then, there exist forests $\mathfrak{f}_1', \mathfrak{f}_2'$ with $\mathfrak{f}_1 \equiv_{\mathfrak{s}} \mathfrak{f}_1'$, $\mathfrak{f}_2 \equiv_{\mathfrak{s}} \mathfrak{f}_2'$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1' \uplus \mathfrak{f}_2') \equiv \chi$.*

Proof. For $i \in \{1, 2\}$, let $\phi_i = \bigstar_{t \in \mathfrak{f}_i} \mathsf{project}^{\mathbf{Loc}}(t)$, let $\mathbf{w_i} = \mathrm{locs}(\phi_i) \cap \mathrm{dom}(\mathfrak{f}_i) \setminus \mathrm{img}(\mathfrak{s})$ be the locations that occur in the formula ϕ and are allocated in $\mathrm{heap}(\mathfrak{f}_i)$ but are not the value of any stack variable, and let $\mathbf{v_i} = \mathrm{locs}(\phi_i) \setminus (\mathrm{img}(\mathfrak{s}) \cup \mathrm{dom}(\mathfrak{f}_i))$ be the locations that occur in the formula ϕ_i and are neither allocated nor the value of any stack variable. Then, we have

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) = \exists \mathbf{e_i}.\, \forall \mathbf{a_i}.\, \phi_i[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_i} \cdot \mathbf{w_i} / \mathrm{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_i} \cdot \mathbf{e_i}],$$

where $\mathbf{e_i} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{w_i}|} \rangle$ and $\mathbf{a_i} \triangleq \langle a_1, a_2, \ldots, a_{|\mathbf{v_i}|} \rangle$ denote some disjoint sets of fresh variables.

By the definition of the re-scoping operation, we have $\chi = \exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi$, where

(1) $\mathbf{e} = \mathbf{e_1} \cdot \mathbf{e_2}$, and
(2) $\phi = \phi_1[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_1} \cdot \mathbf{w_1} / \mathrm{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_1} \cdot \mathbf{e_1}][\mathbf{a_1}/\mathbf{u_1}] \star \phi_2[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_2} \cdot \mathbf{w_2} / \mathrm{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_2} \cdot \mathbf{e_2}][\mathbf{a_2}/\mathbf{u_2}]$ for some sequences $\mathbf{u_i} \subseteq \mathbf{a} \cup \mathbf{e_{3-i}}$.

We can now choose bijective functions $\sigma_1 \colon \mathbf{Loc} \to \mathbf{Loc}$ and $\sigma_2 \colon \mathbf{Loc} \to \mathbf{Loc}$ such that

- $\sigma_1(l) = \sigma_2(l) = l$ for all $l \in \mathrm{img}(\mathfrak{s})$,
- $\sigma_1(l) = \sigma_2(k)$ if and only if $l[\mathbf{v_1} \cdot \mathbf{w_1} / \mathbf{a_1} \cdot \mathbf{e_1}][\mathbf{a_1}/\mathbf{u_1}] = k[\mathbf{v_2} \cdot \mathbf{w_2} / \mathbf{a_2} \cdot \mathbf{e_2}][\mathbf{a_2}/\mathbf{u_2}]$ for all $l \in \mathbf{v_1} \cdot \mathbf{w_1}, k \in \mathbf{v_2} \cdot \mathbf{w_2}$, and
- $\mathrm{dom}(\sigma_1(\mathfrak{f}_1)) \cap \mathrm{dom}(\sigma_1(\mathfrak{f}_2)) = \emptyset$.

We set $\mathfrak{f}_1' \triangleq \sigma(\mathfrak{f}_1)$ and $\mathfrak{f}_2' \triangleq \sigma(\mathfrak{f}_2)$. By the above we have $\mathfrak{f}_1 \equiv_{\mathfrak{s}} \mathfrak{f}_1'$, $\mathfrak{f}_2 \equiv_{\mathfrak{s}} \mathfrak{f}_2'$ and $\mathfrak{f}_1' \uplus \mathfrak{f}_2' \neq \bot$. Further, we get that $\mathsf{project}^{\mathbf{Loc}}(\mathfrak{f}_1' \uplus \mathfrak{f}_2') \equiv \mathsf{project}^{\mathbf{Loc}}(\mathfrak{f}_1)[\mathrm{dom}(\sigma_1)/\mathrm{img}(\sigma_1)] \star \mathsf{project}^{\mathbf{Loc}}(\mathfrak{f}_2)[\mathrm{dom}(\sigma_2)/\mathrm{img}(\sigma_2)]$. Finally, we get that $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1' \uplus \mathfrak{f}_2') \equiv \exists \mathbf{e}.\, \forall \mathbf{a}.\, \phi$ because we can appropriately rename the quantified variables by rewrite equivalence $\equiv$. □

We note that the below lemma does not require the notion of $\mathfrak{s}$-equivalence:

Lemma A.4. *Let f_1 be a Φ-forest and let χ be a formula such that* $\mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \rhd \chi$. *Then, there exist a forest f_2 with $\mathsf{f}_1 \blacktriangleright \mathsf{f}_2$ and* $\mathsf{project}(\mathfrak{s}, \mathsf{f}_2) \equiv \chi$.

Proof. Let $\phi_1 = \bigstar_{\mathsf{t} \in \mathsf{f}_1} \mathsf{project}^{\mathbf{Loc}}(\mathsf{t})$, let $\mathbf{w_1} = \mathsf{locs}(\phi_1) \cap (\mathsf{dom}(\mathsf{f}_1) \setminus \mathsf{img}(\mathfrak{s}))$ be the locations that occur in the formula ϕ and are allocated in $\mathsf{heap}(\mathsf{f}_1)$ but are not the value of any stack variable, and let $\mathbf{v_1} = \mathsf{locs}(\phi_1) \setminus (\mathsf{img}(\mathfrak{s}) \cup \mathsf{dom}(\mathsf{f}_1))$ be the locations that occur in the formula ϕ_1 and are neither allocated nor the value of any stack variable. Then, we have

$$\mathsf{project}(\mathfrak{s}, \mathsf{f}_1) = \exists \mathbf{e_1}. \forall \mathbf{a_1}. \phi_1[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_1} \cdot \mathbf{w_1} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_1} \cdot \mathbf{e_1}],$$

where $\mathbf{e_1} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{w_1}|} \rangle$ and $\mathbf{a_1} \triangleq \langle a_1, a_2, \ldots, a_{|\mathbf{v_1}|} \rangle$ denote some disjoint sets of fresh variables. By definition of the projection of forests, we have

$$\phi_1 \equiv \bigstar_{\mathsf{t} \in \mathsf{f}} ((\bigstar \mathsf{allholepreds}(\mathsf{t})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathsf{t})) .$$

By the definition of $\rhd$ we have that there are predicates $\mathsf{pred}_1(\mathbf{x_1}), \mathsf{pred}_2(\mathbf{x_2})$, and formulae ψ, ψ', ζ such that

(1) $\phi_1 \equiv (\mathsf{pred}_2(\mathbf{x_2}) \star \psi) \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1}))) \star ((\psi' \mathrel{-\!\!\star} \mathsf{pred}_2(\mathbf{x_2})) \star \zeta$, and
(2) $\chi \equiv \exists \mathbf{e_1}. \forall \mathbf{a_1}. (\psi \star \psi') \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1}) \star \zeta$.

Hence, there must be a forest f and trees $\mathsf{t}_1, \mathsf{t}_2$ such that

(1) $\mathsf{f}_1 = \mathsf{f} \cup \{\mathsf{t}_1, \mathsf{t}_2\}$,
(2) $\mathsf{rootpred}(\mathsf{t}_2) \in \mathsf{allholepreds}(\mathsf{t}_1)$, and
(3) $\mathsf{rootpred}(\mathsf{t}_2)[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_1} \cdot \mathbf{w_1} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_1} \cdot \mathbf{e_1}] = \mathsf{pred}_2(\mathbf{x_2})$.

Let $l = \mathsf{root}(\mathsf{t}_2)$. Then, there is a tree t with $\{\mathsf{t}_1, \mathsf{t}_2\} = \mathsf{split}(\{\mathsf{t}\}, \{l\})$. We set $\mathsf{f}_2 = \mathsf{f} \cup \{\mathsf{t}\}$. We note that $\mathsf{f}_1 \blacktriangleright \mathsf{f}_2$. It remains to argue that $\mathsf{project}(\mathfrak{s}, \mathsf{f}_2) \equiv \chi$.

Let $\phi_2 = \bigstar_{\mathsf{t} \in \mathsf{f}_2} \mathsf{project}^{\mathbf{Loc}}(\mathsf{t})$, let $\mathbf{w_2} = \mathsf{locs}(\phi_2) \cap (\mathsf{dom}(\mathsf{f}_2) \setminus \mathsf{img}(\mathfrak{s}))$ be the locations that occur in the formula ϕ and are allocated in $\mathsf{heap}(\mathsf{f}_2)$ but are not the value of any stack variable, and let $\mathbf{v_2} = \mathsf{locs}(\phi_2) \setminus (\mathsf{img}(\mathfrak{s}) \cup \mathsf{dom}(\mathsf{f}_2))$ be the locations that occur in the formula ϕ_2 and are neither allocated nor the value of any stack variable. Then, we have

$$\mathsf{project}(\mathfrak{s}, \mathsf{f}_2) = \exists \mathbf{e_2}. \forall \mathbf{a_2}. \phi_2[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_2} \cdot \mathbf{w_2} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_2} \cdot \mathbf{e_2}],$$

where $\mathbf{e_2} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{w_2}|} \rangle$ and $\mathbf{a_2} \triangleq \langle a_1, a_2, \ldots, a_{|\mathbf{v_2}|} \rangle$ denote some disjoint sets of fresh variables. We now note that $\mathsf{locs}(\phi_2) \cup \{l\} = \mathsf{locs}(\phi_1)$. Hence, we can assume without loss of generality that $\mathbf{e_2} \subseteq \mathbf{e_1}$ and $\mathbf{a_2} \subseteq \mathbf{a_1}$. Thus, we have $(\psi \star \psi') \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1}) \star \zeta \equiv \phi_2[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_2} \cdot \mathbf{w_2} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_2} \cdot \mathbf{e_2}]$. Finally, we note that

$$\chi \equiv \exists \mathbf{e_1}. \forall \mathbf{a_1}. \phi_2[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_2} \cdot \mathbf{w_2} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_2} \cdot \mathbf{e_2}] \equiv \exists \mathbf{e_2}. \forall \mathbf{a_2}. \phi_2[\mathsf{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v_2} \cdot \mathbf{w_2} / \mathsf{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{a_2} \cdot \mathbf{e_2}],$$

because we can drop the quantified variables $\mathbf{e_1} \setminus \mathbf{e_2}$ and $\mathbf{a_1} \setminus \mathbf{a_2}$, which is supported by our rewriting rules (see Fig. 10) because these variables do not appear in ϕ_2. □

Claim (Theorem 7.39). If $\mathsf{f}_1, \mathsf{f}_2$ be Φ-forests with $\mathsf{f}_1 \equiv_{\mathfrak{s}} \mathsf{f}_2$, then

$$\mathsf{project}(\mathfrak{s}, \mathsf{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathsf{f}_2) \;=\; \left\{ \mathsf{project}(\mathfrak{s}, \mathsf{f}) \mid \mathsf{f} \in \bar{\mathsf{f}}_1 \bullet_{\mathbf{F}} \bar{\mathsf{f}}_2,\ \bar{\mathsf{f}}_1 \equiv_{\mathfrak{s}} \mathsf{f}_1,\ \bar{\mathsf{f}}_2 \equiv_{\mathfrak{s}} \mathsf{f}_2 \right\}.$$

Proof. The claim is an immediate consequence of Lemmas 7.35, A.3 and A.4. □

A.16 Proof of Lemma 8.6

Claim. Let $n \triangleq |\Phi| + |\mathbf{x}|$, where $\mathbf{x}$ is a finite set of variables. Then $\left|\mathbf{DUSH}^{\mathbf{x}}_{\Phi}\right| \in 2^{O(n^2 \log(n))}$.

Proof. We first show the following claim (†): every element of $\mathbf{DUSH}^{\mathbf{x}}_{\Phi}$ can be encoded as a string of length $O(n^2)$ over the alphabet $Z \triangleq \mathbf{Preds}(\Phi) \cup \mathbf{x} \cup \{e_1, \dots, e_{n^2}\} \cup \{a_1, \dots, a_{n^2}\} \cup \{\mathbf{emp}, \star, \mathrel{-\!\!\star}, (,)\}$ where $e_1, \dots, e_{n^2}$ and $a_1, \dots, a_{n^2}$ are fresh variables.

By definition, every DUSH $\phi \in \mathbf{DUSH}^{\mathbf{x}}_{\Phi}$ is of the form

$$\phi = \exists \mathbf{e}.\ \forall \mathbf{a}.\ \psi_1 \star \dots \star \psi_m,$$
$$\psi_i = \zeta_i \mathrel{-\!\!\star} \mathsf{pred}_i(\mathbf{z_i}) \text{ for } 1 \leq i \leq m.$$

Since ϕ is delimited, $\mathsf{predroot}(\mathsf{pred}_i(\mathbf{z_i})) \in \mathbf{x}$. Moreover, ϕ is the projection of a Φ-forest $\mathfrak{f}$. Hence, every variable $x \in \mathbf{x}$ can appear as a root parameter in at most one subformula ψ_i—otherwise, the value corresponding to x would be in the domain of two trees in $\mathfrak{f}$, which contradicts the fact that $\mathfrak{f}$ is a Φ-forest. Consequently, the number m of subformulas ψ_i is bounded by $|\mathbf{x}| \leq n$.

Next, consider the subformulas ζ_i appearing on the left-hand side of magic wands. For every predicate call $\mathsf{pred}'(\mathbf{z}')$ in ζ_i, $\mathsf{predroot}(\mathsf{pred}'(\mathbf{z}'))$ is a hole. Since the forest $\mathfrak{f}$ is delimited, it follows that $\mathsf{predroot}(\mathsf{pred}'(\mathbf{z}')) \in \mathbf{x}$. Since no hole may occur more than once in a delimited USH, the *total* number of predicate calls across all ζ_i is also bounded by $|\mathbf{x}| \leq n$.

Overall, ϕ thus contains at most $2n \in O(n)$ predicate calls. Each predicate call takes at most $|\Phi| \leq n$ parameters. Since there are no superfluous quantified variables, this means that ϕ contains at most $n^2 - |\mathbf{x}| \leq n^2$ different variables. We can thus assume w.l.o.g. that all existentially-quantified variables in ϕ are among the variables $e_1, \dots, e_{n^2}$ and all universally-quantified variables are among $a_1, \dots, a_{n^2}$. There then is no need to include the quantifiers explicitly in the string encoding. After dropping the quantifiers, we obtain a formula ϕ' that consists exclusively of letters from the alphabet Z. Moreover, this formula consists of at most $O(n^2)$ letters. This concludes the proof of (†).

Now observe that $|Z| \in O(n^2)$. Consequently, every letter of Z can be encoded by $O(\log(n^2)) = O(\log(n))$ bits. Therefore, every $\phi \in \mathbf{DUSH}^{\mathbf{x}}_{\Phi}$ can be encoded by a bit string of length $O(n^2 \log(n))$. Since there are $2^{O(n^2 \log(n))}$ such strings, the claim follows. □

A.17 Proof of Lemma 8.12

Claim. Let $\mathfrak{f}$ be a forest and let $\mathfrak{s}$ be a stack. Then $\mathfrak{f}$ is $\mathfrak{s}$-delimited iff $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$ is delimited.

Proof. Recall that the projection contains predicate calls corresponding to the roots and holes of the forest. It thus holds for all forests that

$$\mathsf{interface}(\mathfrak{f}) = \{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z}) \in \mathsf{project}(\mathfrak{f})\}. \tag{†}$$

We show that if $\mathfrak{f}$ is $\mathfrak{s}$-delimited then $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$ is delimited. The proof of the other direction is completely analogous.

If $\mathfrak{f}$ is $\mathfrak{s}$-delimited then $\mathsf{interface}(\mathfrak{f}) \subseteq \mathsf{img}(\mathfrak{s})$ and thus, by (†),

$$\{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z}) \in \mathsf{project}(\mathfrak{f})\} \subseteq \mathsf{img}(\mathfrak{s}).$$

Trivially, the set of root locations in the projection is a subset of the set of *all* locations in the projection.

$$\{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z}) \in \mathsf{project}(\mathfrak{f})\} \subseteq \mathsf{locs}(\mathsf{project}(\mathfrak{f})).$$

Combining the above two observations, we conclude

$$\begin{aligned}&\{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z}) \in \mathsf{project}(\mathfrak{f})\}\\ \subseteq\ &\mathsf{img}(\mathfrak{s}) \cap \mathsf{locs}(\mathsf{project}(\mathfrak{f})).\end{aligned}$$

We apply $\mathfrak{s}^{-1}_{\max}$ on both sides to obtain that

$$\begin{aligned}&\mathfrak{s}^{-1}_{\max}(\{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z} \in \mathsf{project}(\mathfrak{f}))\})\\ \subseteq\ &\mathsf{dom}(\mathfrak{s}) \cap \underbrace{(\mathsf{fvars}(\mathsf{project}(\mathfrak{s}, \mathfrak{f}))}_{\left\{\mathfrak{s}^{-1}_{\max}(l) \mid l \in \mathsf{img}(\mathfrak{s}) \cap \mathsf{locs}(\mathsf{project}(\mathfrak{f}))\right\}} \subseteq \mathsf{fvars}(\mathsf{project}(\mathfrak{s}, \mathfrak{f})).\end{aligned}$$

Moreover, since there are no duplicate holes in $\mathfrak{f}$, and the holes of $\mathfrak{f}$ are mapped to the predicate calls on the left-hand side of magic wands in $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$, no variable can occur twice as root parameter on the left-hand side of magic wands in $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$.

Consequently, $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$ is delimited. □

A.18 Proof of Theorem 8.13

We first some auxiliary definitions and results. Recall that we described how Φ-forests are merged in terms of splitting them at suitable locations (cf. Definition 7.9). Every split adds these locations to the interface of the resulting forest—provided they did not appear in the forest to begin with.

Lemma A.5. *Let* $\mathfrak{f}$ *be a forest and* $\mathbf{l} \subseteq \mathbf{Loc}$*. Then,* $\mathsf{interface}(\mathsf{split}(\mathfrak{f}, \mathbf{l})) = \mathsf{interface}(\mathfrak{f}) \cup (\mathbf{l} \cap \mathsf{dom}(\mathfrak{f}))$.

Proof. In the following, let $\mathsf{locs}(\mathsf{graph}(\mathfrak{f}))$ denote all those locations that occur in the relation $\mathsf{graph}(\mathfrak{f})$.

$$\begin{aligned}&\mathsf{roots}(\mathsf{split}(\mathfrak{f}, \mathbf{l}))\\ =&\mathsf{roots}(\mathfrak{f}) \cup \{b \in \mathbf{l} \mid \exists a.\, (a, b) \in \mathsf{graph}(\mathfrak{f})\}\\ =&\mathsf{roots}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathsf{dom}(\mathfrak{f}) \mid \exists a.\, (a, b) \in \mathsf{graph}(\mathfrak{f})\} && (\mathsf{locs}(\mathsf{graph}(\mathfrak{f})) \subseteq \mathsf{dom}(\mathfrak{f}))\\ =&\{b \in \mathsf{dom}(\mathfrak{f}) \mid \forall a.\, (a, b) \notin \mathsf{graph}(\mathfrak{f})\}\\ &\cup \{b \in \mathbf{l} \cap \mathsf{dom}(\mathfrak{f}) \mid \exists a.\, (a, b) \in \mathsf{graph}(\mathfrak{f})\} && (\text{all and only roots have no predecessor})\\ =&\{b \in \mathsf{dom}(\mathfrak{f}) \mid \forall a.\, (a, b) \notin \mathsf{graph}(\mathfrak{f})\}\\ &\cup \{b \in \mathbf{l} \cap \mathsf{dom}(\mathfrak{f}) \mid \forall a.\, (a, b) \notin \mathsf{graph}(\mathfrak{f})\}\\ &\cup \{b \in \mathbf{l} \cap \mathsf{dom}(\mathfrak{f}) \mid \exists a.\, (a, b) \in \mathsf{graph}(\mathfrak{f})\} && (\text{second set subset of first set})\\ =&\mathsf{roots}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathsf{dom}(\mathfrak{f}) \mid \forall a.\, (a, b) \notin \mathsf{graph}(\mathfrak{f})\}\\ &\cup \{b \in \mathbf{l} \cap \mathsf{dom}(\mathfrak{f}) \mid \exists a.\, (a, b) \in \mathsf{graph}(\mathfrak{f})\}\\ =&\mathsf{roots}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathsf{dom}(\mathfrak{f})\}\end{aligned}$$

Similarly,

$$\begin{aligned}\mathsf{allholes}(\mathsf{split}(\mathfrak{f}, \mathbf{l})) =&\mathsf{allholes}(\mathfrak{f}) \cup \{b \in \mathbf{l} \mid \exists a.\, (a, b) \in \mathsf{graph}(\mathfrak{f})\}\\ =&\mathsf{allholes}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathsf{dom}(\mathfrak{f})\}\,.\end{aligned}$$

By definition of interfaces, we thus obtain

$$\begin{aligned}\mathsf{interface}(\mathsf{split}(\mathfrak{f},\mathfrak{l})) =& \mathsf{roots}(\mathsf{split}(\mathfrak{f},\mathfrak{l})) \cup \mathsf{allholes}(\mathsf{split}(\mathfrak{f},\mathfrak{l}))\\ =& \mathsf{roots}(\mathfrak{f}) \cup \{t \in \mathfrak{l} \cap \mathrm{dom}(\mathfrak{f})\}\\ & \cup \mathsf{allholes}(\mathfrak{f}) \cup \{b \in \mathfrak{l} \cap \mathrm{dom}(\mathfrak{f})\}\\ =& \mathsf{interface}(\mathfrak{f}) \cup \{b \in \mathfrak{l} \cap \mathrm{dom}(\mathfrak{f})\}\,. \end{aligned}$$ □

Definition A.6. *Let $\mathfrak{f}$ be an $\mathfrak{s}$-delimited forest. We call* $\mathsf{split}(\mathfrak{f}, \mathrm{img}(\mathfrak{s}))$ *the* $\mathfrak{s}$-decomposition *of* $\mathfrak{f}$.

Lemma A.7. *The $\mathfrak{s}$-decomposition of an $\mathfrak{s}$-delimited forest is $\mathfrak{s}$-delimited.*

Proof. Let $\bar{\mathfrak{f}}$ be the $\mathfrak{s}$-decomposition of an $\mathfrak{s}$-delimited forest $\mathfrak{f}$. By definition, $\bar{\mathfrak{f}} = \mathsf{split}(\mathfrak{f}, \mathrm{img}(\mathfrak{s}))$. By Lemma A.5, we have $\mathsf{interface}(\bar{\mathfrak{f}}) \subseteq \mathsf{interface}(\mathfrak{f}) \cup \mathrm{img}(\mathfrak{s})$. Since $\mathfrak{f}$ is $\mathfrak{s}$-delimited, $\mathsf{interface}(\mathfrak{f}) \subseteq \mathrm{img}(\mathfrak{s})$. Overall, we thus obtain $\mathsf{interface}(\bar{\mathfrak{f}}) \subseteq \mathrm{img}(\mathfrak{s})$, i.e., $\bar{\mathfrak{f}}$ is $\mathfrak{s}$-delimited. □

We observe that, since the $\mathfrak{s}$-decomposition of a forest is obtained by splitting the trees of the forest at all locations in $\mathrm{img}(\mathfrak{s})$, only the roots of the trees in an $\mathfrak{s}$-decomposition of forest $\mathfrak{f}$ can be locations in $\mathrm{img}(\mathfrak{s})$:

Lemma A.8. *For every Φ-tree $\bar{\mathfrak{t}}$ in an $\mathfrak{s}$-decomposition, we have* $\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{t}}) = \{\mathrm{root}(\bar{\mathfrak{t}})\}$.

Proof. Let $\bar{\mathfrak{f}}$ be an $\mathfrak{s}$-decomposition and let $\bar{\mathfrak{t}} \in \bar{\mathfrak{f}}$. Since $\bar{\mathfrak{f}}$ is $\mathfrak{s}$-delimited by Lemma A.7, we have $\{\mathrm{root}(\bar{\mathfrak{t}})\} \subseteq \mathrm{img}(\mathfrak{s})$. Since $\mathrm{root}(\bar{\mathfrak{t}}) \in \mathrm{dom}(\bar{\mathfrak{t}})$, $\{\mathrm{root}(\bar{\mathfrak{t}})\} \subseteq \mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{t}})$.

Conversely, since $\bar{\mathfrak{f}} = \mathsf{split}(\mathfrak{f}, \mathrm{img}(\mathfrak{s}))$, we have $\mathsf{roots}(\bar{\mathfrak{f}}) = \mathsf{roots}(\mathfrak{f}) \cup (\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\mathfrak{f}))$, i.e., every location in $\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{f}})$ is a root of $\bar{\mathfrak{f}}$. Consequently, $\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{t}}) \subseteq \{\mathrm{root}(\bar{\mathfrak{t}})\}$. □

Lemma A.9. *Let $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle \in$ **GStates** be guarded states, and let $\mathfrak{f}$ be a $\mathfrak{s}$-delimited forest with $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$. Then, there exist forests $\mathfrak{f}_1, \mathfrak{f}_2$ with $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \bar{\mathfrak{f}}$ and $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$, where $\bar{\mathfrak{f}}$ is the $\mathfrak{s}$-decomposition of $\mathfrak{f}$.*

Proof. We let $\mathfrak{f}_i \triangleq \{\bar{\mathfrak{t}} \in \bar{\mathfrak{f}} \mid \mathrm{root}(\bar{\mathfrak{t}}) \in \mathrm{dom}(\mathfrak{h}_i)\}$. Since $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \mathfrak{f}$ and thus $\mathsf{heap}(\mathfrak{f}_1) \uplus \mathsf{heap}(\mathfrak{f}_2) = \mathsf{heap}(\mathfrak{f})$ by Lemma 7.7, it suffices to show that for every tree $\bar{\mathfrak{t}}$ in $\mathfrak{f}_i$ that $\mathsf{heap}(\bar{\mathfrak{t}}) \subseteq \mathfrak{h}_i$.

To this end, let $\bar{\mathfrak{t}} \in \mathfrak{f}_i$. Assume towards a contradiction that $\mathrm{dom}(\bar{\mathfrak{t}}) \cap \mathrm{dom}(\mathfrak{h}_{3-i}) \neq \emptyset$. Then there exist locations $l_1 \in \mathrm{dom}(\bar{\mathfrak{t}}) \cap \mathrm{dom}(\mathfrak{h}_i)$ and $l_2 \in \mathrm{dom}(\bar{\mathfrak{t}}) \cap \mathrm{dom}(\mathfrak{h}_{3-i})$ with $l_2 \in \mathrm{succ}_{\bar{\mathfrak{t}}}(l_1)$. In particular, $l_2 \in \mathrm{img}(\mathfrak{h}_i)$ and $l_2 \in \mathrm{dom}(\mathfrak{h}_{3-i})$, implying that $l_2 \in \mathsf{dangling}(\mathfrak{h}_i)$. However, since $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle \in$ **GStates**, we have that $l_2 \in \mathrm{img}(\mathfrak{s})$. Since $l_2 \neq \mathrm{root}(\mathfrak{t})$, this contradicts Lemma A.8. □

We restate the claim of Theorem 8.13: Let $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle \in$ **GStates** be guarded states, and let $\mathfrak{f}$ be a $\mathfrak{s}$-delimited forest with $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$. Then there exist $\mathfrak{s}$-delimited forests $\mathfrak{f}_1, \mathfrak{f}_2$ with $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$ and $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$.

Proof. Let $\bar{\mathfrak{f}}$ be the $\mathfrak{s}$-decomposition of $\mathfrak{f}$. In particular, we then have $\bar{\mathfrak{f}} \blacktriangleright^* \mathfrak{f}$ by definition of $\blacktriangleright^*$. Let $\mathfrak{f}_1, \mathfrak{f}_2$ be such that $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \bar{\mathfrak{f}}$ and $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$. Such forests exist by Lemma A.9. Then $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \bar{\mathfrak{f}} \blacktriangleright^* \mathfrak{f}$, i.e., $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$. Since $\bar{\mathfrak{f}}$ is $\mathfrak{s}$-delimited (by Lemma A.7), so are $\mathfrak{f}_1$ and $\mathfrak{f}_2$. □

A.19 Proof of Theorem 8.14

Claim. For all guarded states $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ with $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \neq \bot$, $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$ can be computed from $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$ as follows:

$$\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \{\phi \in \mathbf{DUSH}_\Phi \mid \text{ex. } \psi_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1), \psi_2 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) \text{ such that } \phi \in \psi_1 \bullet_{\mathbf{P}} \psi_2\}.$$

Proof. $\subseteq$ Let $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$. By Definition 8.4, we know that (1) $\phi = \mathsf{project}(\mathfrak{s}, \mathfrak{f})$ for some forest $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$ and (2) ϕ is delimited. By (2) and Lemma 8.12, $\mathfrak{f}$ is delimited as well. Moreover, by Theorem 8.13, there exist $\mathfrak{s}$-delimited forests $\mathfrak{f}_1$ and $\mathfrak{f}_2$ with $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$ and, for $i \in \{1, 2\}$, $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$. By

Lemma 8.12, both $\psi_1 \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1)$ and $\psi_2 \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ are delimited—hence, $\psi_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ and $\psi_2 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. Furthermore, by Theorem 7.39, we have $\phi = \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \psi_1 \bullet_{\mathbf{P}} \psi_2$.

$\supseteq$ Assume there exist formulas $\phi \in \mathbf{DUSH}_\Phi$, $\psi_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$, and $\psi_2 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$ such that $\phi \in \psi_1 \bullet_{\mathbf{P}} \psi_2$. By Definition 8.4, there exist forests $\mathfrak{f}, \mathfrak{f}_1, \mathfrak{f}_2$ such that, for $i \in \{1, 2\}$, we have $\psi_i = \mathsf{project}(\mathfrak{s}, \mathfrak{f}_i)$ and $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$. Then, by Theorem 7.39, there exist forests $\bar{\mathfrak{f}}_1, \bar{\mathfrak{f}}_2$ such that $\mathfrak{f}_1 \equiv_{\mathfrak{s}} \bar{\mathfrak{f}}_1$, $\mathfrak{f}_2 \equiv_{\mathfrak{s}} \bar{\mathfrak{f}}_2$, $\mathfrak{f} \in \bar{\mathfrak{f}}_1 \bullet_{\mathbf{F}} \bar{\mathfrak{f}}_2$, and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \phi$. By Definition 7.37, we have, for $i \in \{1, 2\}$, $\mathsf{heap}(\bar{\mathfrak{f}}_i) = \mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$. Moreover, by Lemma 7.7, we have $\mathfrak{h}_1 \uplus \mathfrak{h}_2 = \mathsf{heap}(\bar{\mathfrak{f}}_1 \uplus \bar{\mathfrak{f}}_2)$. Since $\mathfrak{f} \in \bar{\mathfrak{f}}_1 \bullet_{\mathbf{F}} \bar{\mathfrak{f}}_2$, Lemma 7.14 and Definition 7.15 yield that $\mathfrak{h}_1 \uplus \mathfrak{h}_2 = \mathsf{heap}(\mathfrak{f})$. Hence, $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$ and thus also $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$. □

A.20 Proof of Lemma 8.16

We first show an auxiliary result, namely that the stack-allocated variables of a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ correspond precisely to the roots of the $\mathfrak{s}$-decomposed forests of $\mathfrak{h}$:

Lemma A.10. *Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state and let $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ an $\mathfrak{s}$-delimited forest. Then, we have* $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}$*, where $\bar{\mathfrak{f}}$ is the $\mathfrak{s}$-decomposition of $\mathfrak{f}$.*

Proof. By Lemma 7.14, $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathfrak{h}$ and thus, in particular, $\mathsf{dom}(\bar{\mathfrak{f}}) = \mathsf{dom}(\mathfrak{h})$. Consequently,

$$\mathfrak{s}(\mathsf{alloced}(\mathfrak{s}, \mathfrak{h})) = \mathsf{img}(\mathfrak{s}) \cap \mathsf{dom}(\bar{\mathfrak{f}}).$$

By Lemma A.8, we have $\mathsf{img}(\mathfrak{s}) \cap \mathsf{dom}(\bar{\mathfrak{t}}) = \{\mathsf{root}(\bar{\mathfrak{t}})\}$ for all $\bar{\mathfrak{t}} \in \bar{\mathfrak{f}}$. Hence,

$$\mathsf{img}(\mathfrak{s}) \cap \mathsf{dom}(\bar{\mathfrak{f}}) = \mathsf{roots}(\bar{\mathfrak{f}}).$$

Overall, we thus have $\mathfrak{s}(\mathsf{alloced}(\mathfrak{s}, \mathfrak{h})) = \mathsf{roots}(\bar{\mathfrak{f}})$. By taking the inverse $\mathfrak{s}^{-1}$ on both sides of the equation, we obtain that $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}$. □

We restate the claim of Lemma 8.16: Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state with $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \neq \emptyset$. Then, $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))$.

Proof. By definition of DUSHs, all root parameters of all DUSHs in $\mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))$ are in $\mathsf{img}(\mathfrak{s})$. Consequently, $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) \supseteq \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))$

For the other implication, let $\mathfrak{f}$ be a forest with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. Such a forest must exist, as $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \neq \emptyset$ by assumption. Let $\bar{\mathfrak{f}}$ be the $\mathfrak{s}$-decomposition of $\mathfrak{f}$. By Lemma A.7, $\bar{\mathfrak{f}}$ is delimited and by Lemma 7.14, $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathfrak{h}$, implying $\mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ and we can apply Lemma A.10 to obtain that

$$\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}.$$

Consequently, all variables in $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h})$ occur as root parameters on the right-hand side of magic wands in $\mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}})$. Therefore, $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) \subseteq \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))$. □

A.21 Proof of Lemma 8.17

Claim. Let $\phi \in \mathbf{GSL}$ and let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state with $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$. Then, $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \neq \emptyset$.

Proof. By Corollary 4.6, $\langle \mathfrak{s}, \mathfrak{h} \rangle$ is a guarded state. Lemma 4.5 then yields that there exist $k \geq 1$ predicate calls such that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \bigstar_{1 \leq i \leq k} \mathsf{pred}_i(\mathbf{x_i})$.

We split the heap $\mathfrak{h}$ into disjoint heaps $\mathfrak{h}_1 \uplus \cdots \uplus \mathfrak{h}_k$ such that $\langle \mathfrak{s}, \mathfrak{h}_i \rangle \models_\Phi \mathsf{pred}_i(\mathbf{x_i})$ for each $i \in [1, k]$. Next, consider the forest $\mathfrak{f} \triangleq \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$, where each $\mathfrak{t}_i$ is a Φ-tree with $\mathsf{heap}(\mathfrak{t}_i) = \mathfrak{h}_i$; such trees exist by Lemma 7.3. Observe further that each of these trees is delimited, because they do not have holes and their root is in $\mathfrak{s}(\mathbf{x_i})$. By Lemma 7.7, we have $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$. Finally, Lemma 7.25 yields $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f})$ and thus $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. Hence, $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \neq \emptyset$. □

A.22 Proof of Corollary 8.19

Claim. For guarded states $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ with $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \neq \bot$, we have $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$.

Proof. We need to show that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) \neq \bot$. Assume that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i) = \emptyset$ for $i = 1$ or $i = 2$. Then, $\mathsf{alloced}(\mathcal{T}_i) = \emptyset$ and we get that $\mathsf{alloced}(\mathcal{T}_1) \cap \mathsf{alloced}(\mathcal{T}_2) = \emptyset$, Otherwise, we have $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i) \neq \emptyset$ for $i = 1, 2$. Then, $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_i) = \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i))$. $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \neq \bot$ then implies that $\mathsf{alloced}(\mathcal{T}_1) \cap \mathsf{alloced}(\mathcal{T}_2) = \emptyset$. The claim then follows from Theorem 8.14. □

A.23 Proof of Lemma 8.20

Claim. For $i \in \{1, 2\}$, let $\langle \mathfrak{s}, \mathfrak{h}_i \rangle$ be states with $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i) = \mathcal{T}_i \neq \emptyset$ and $\mathcal{T}_1 \bullet \mathcal{T}_2 \neq \bot$. Then, there are states $\langle \mathfrak{s}, \mathfrak{h}'_i \rangle$ such that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}'_i) = \mathcal{T}_i$ and $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}'_1 \uplus \mathfrak{h}'_2) = \mathcal{T}_1 \bullet \mathcal{T}_2$.

Proof. We choose some states $\langle \mathfrak{s}, \mathfrak{h}'_i \rangle$ that are isomorphic to $\langle \mathfrak{s}, \mathfrak{h}_i \rangle$ such that $\mathsf{locs}(\mathfrak{h}'_1) \cap \mathsf{locs}(\mathfrak{h}'_2) \subseteq \mathsf{img}(\mathfrak{s})$. We have that $\langle \mathfrak{s}, \mathfrak{h}'_i \rangle = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i) = \mathcal{T}_i$ because isomorphic states have the same types (observe that the stack-projection replaces location that are not in the image of the stack by quantified variables). By Lemma 8.16, we have $\mathsf{alloced}(\mathcal{T}_i) = \mathsf{alloced}(\langle \mathfrak{s}, \mathfrak{h}'_i \rangle) = \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}'_i)$. Thus, we get $\mathfrak{h}'_1 \uplus \mathfrak{h}'_2 \neq \bot$ from $\mathcal{T}_1 \bullet \mathcal{T}_2 \neq \bot$ and $\mathsf{locs}(\mathfrak{h}'_1) \cap \mathsf{locs}(\mathfrak{h}'_2) \subseteq \mathsf{img}(\mathfrak{s})$. Then, Theorem 8.14 yields that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}'_1 \uplus \mathfrak{h}'_2) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}'_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}'_2)$. □

A.24 Proof of Lemma 8.22

Claim. For $\mathbf{x}$, $\mathbf{y}$ as above and a stack $\mathfrak{s}$ with $\mathbf{y} \subseteq \mathsf{dom}(\mathfrak{s})$ and $\mathbf{x} \cap \mathsf{dom}(\mathfrak{s}) = \emptyset$, we have

$$\mathsf{type}_\Phi(\mathfrak{s}[\mathbf{x}/\mathbf{y}], \mathfrak{h})[\mathsf{aliasing}(\mathfrak{s}) : \mathbf{x}/\mathbf{y}] = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}).$$

Proof. Let $\mathbf{y}'$ be the sequence obtained by replacing every variable in $y \in \mathbf{y}$ by the maximal variable y' with $\mathfrak{s}(y') = \mathfrak{s}(y)$. We consider some $\phi \in \mathsf{type}_\Phi(\mathfrak{s}[\mathbf{x}/\mathbf{y}], \mathfrak{h})$. Then, there exists a forest $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ such that $\phi = \mathsf{project}(\mathfrak{s}[\mathbf{x}/\mathbf{y}], \mathfrak{f})$. By construction of stack-projections (cf. Definition 7.23), we obtain that

$$\phi[\mathbf{x}/\mathbf{y}'] = \mathsf{project}(\mathfrak{s}[\mathbf{x}/\mathbf{y}], \mathfrak{f})[\mathbf{x}/\mathbf{y}'] = \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}).$$

The converse direction is analogous. □

A.25 Proof of Lemma 8.24

Claim. Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a guarded state such that $\mathfrak{s}(x) \in \mathsf{dom}(\mathfrak{h})$ holds for some variable x. Then,

$$\mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),x}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})) = \mathsf{type}_\Phi(\mathfrak{s}[x/\bot], \mathfrak{h}) \ .$$

Proof. We will use the following fact (†) based on the construction of projections (cf. Definition 7.23): If $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$, then $\mathfrak{s}(x)$ is replaced in $\mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f})$ by an existentially-quantified variable iff x is replaced in $\mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),x}(\mathsf{project}(\mathfrak{s}, \mathfrak{f}))$ by an existentially-quantified variable.

Now, consider some $\phi \in \mathsf{type}_\Phi(\mathfrak{s}[x/\bot], \mathfrak{h})$. Then, there is some $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ such that $\mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f}) = \phi \in \mathbf{DUSH}_\Phi$. Clearly, we have $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathbf{DUSH}_\Phi$ and hence $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. Applying (†), we conclude that

$$\phi = \mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),x}(\mathsf{project}(\mathfrak{s}, \mathfrak{f})) \in \mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),x}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})) \ .$$

Conversely, let $\phi \in \mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),x}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))$. By construction, $\phi \in \mathbf{DUSH}_\Phi$ and there is some $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ such that $\mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),x}(\mathsf{project}(\mathfrak{s}, \mathfrak{f})) = \phi$. Applying (†), we can conclude that $\phi = \mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}[x/\bot], \mathfrak{h})$. □

A.26 Proof of Lemma 8.26

Claim. For every state $\langle \mathfrak{s}, \mathfrak{h} \rangle$, variable x with $\mathfrak{s}(x) \notin \mathsf{locs}(\mathfrak{h})$ and $\mathsf{aliasing}(\mathfrak{s})(x) = \{x\}$,

$$\mathsf{extend}_x(\mathsf{type}_\Phi(\mathfrak{s}[x/\bot], \mathfrak{h})) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}).$$

Proof. Assume ϕ is the x-extension of some $\phi' \in \mathsf{type}_\Phi(\mathfrak{s}[x/\bot], \mathfrak{h})$. Then there exists some $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ such that $\phi' = \mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f}) \in \mathbf{DUSH}_\Phi$. We can now choose a forest $\mathfrak{f}'$ with $\mathfrak{f} \equiv_\mathfrak{s} \mathfrak{f}'$ such that $\phi = \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. Because of $\mathfrak{f} \equiv_\mathfrak{s} \mathfrak{f}'$ and $\mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f}) \in \mathbf{DUSH}_\Phi$ we get that $\mathsf{project}(\mathfrak{s}, \mathfrak{f}') \in \mathbf{DUSH}_\Phi$. Hence, $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$.

Conversely, let $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. Then there exists some $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ such that $\phi = \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathbf{DUSH}_\Phi$. We note that $\mathfrak{s}(x) \notin \mathsf{interface}(\mathfrak{f})$ since $\mathfrak{s}(x) \notin \mathsf{locs}(\mathfrak{h})$. Hence, $\mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}[x/\bot], \mathfrak{h})$. We distinguish two cases: First, if $x \notin \mathsf{fvars}(\phi)$, then $\phi = \mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}[x/\bot], \mathfrak{h})$. Second, if $x \in \mathsf{fvars}(\phi)$, then $\mathfrak{s}(x)$ corresponds to a universally quantified variable in $\mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f})$. Hence, ϕ is an x-instantiation of $\mathsf{project}(\mathfrak{s}[x/\bot], \mathfrak{f})$. By Definition 8.25, this means $\phi \in \mathsf{extend}_x(\mathsf{type}_\Phi(\mathfrak{s}[x/\bot], \mathfrak{h}))$. □

A.27 Proof of Theorem 8.29

For a concise formalization, we assume—in addition to our global assumptions stated in Section 3.4.5—that all formulas ϕ under consideration are **GSL** formulas without constant locations.

We will prove Theorem 8.29 by structural induction on the syntax of **GSL** formulas. For most base cases—those that involve the heap—we rely on the fact that a state satisfies all formulas in its type.

Lemma A.11. *If $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ for some state $\langle \mathfrak{s}, \mathfrak{h} \rangle$, then $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$.*

Proof. Since $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$, there exists a Φ-forest $\mathfrak{f}$ with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \phi$. By Lemma 7.25, we have $\langle \mathfrak{s}, \mathsf{heap}(\mathfrak{f}) \rangle \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f})$ and thus also $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi$. □

Finally, to deal with the separating conjunction, we need another auxiliary result. In Corollary 8.19, we showed how two types can be composed into a single one, i.e.,

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

To prove Theorem 8.29, we need the reverse: Given a composed type, say $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}_1 \bullet \mathcal{T}_2$, we need to *decompose* $\mathfrak{h}$ into two heaps whose types (in conjunction with stack $\mathfrak{s}$) are $\mathcal{T}_1$ and $\mathcal{T}_2$.

Lemma A.12 (Type decomposability). *Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state with $\emptyset \neq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}_1 \bullet \mathcal{T}_2$. Then, there exist $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathcal{T}_1 = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$, and $\mathcal{T}_2 = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$.*

Since proving type decomposability involves a bit more technical machinery, we refer the interested reader to Appendix A.28 for a detailed proof.

With the above three lemmas at hand, we can now prove the refinement theorem.

Claim (Refinement theorem). For all stacks $\mathfrak{s}$, heaps $\mathfrak{h}_1$, $\mathfrak{h}_2$, and **GSL** formulas ϕ,

$$\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) \qquad \text{implies} \qquad \langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \phi \ \text{ iff } \ \langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi\ .$$

Proof. We only show that if $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \phi$ then $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi$; the converse direction is symmetric. We proceed by induction on the structure of the **GSL** formula ϕ. Let us assume that $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \phi$.

Case $\phi = \mathbf{emp}$. By the semantics of **emp**, we have $\mathfrak{h}_1 = \emptyset$. Let $\mathfrak{f}$ be the empty forest. Then $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h}_1)$ and thus $\mathbf{emp} = \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. Hence, by Lemma A.11, we have $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \mathbf{emp}$.

Cases $\phi = x \approx y$, $\phi = x \not\approx y$. We observe that the states $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ have the same stack. Then we proceed as in the case for $\phi = \mathbf{emp}$.

Case $\phi = x \mapsto \langle y_1, \ldots, y_k \rangle$**.** By assumption, Φ is pointer-closed (see Definition 3.7), i.e., $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \mathsf{ptr}_k(x, y_1, \ldots, y_k)$. We define a Φ-forest $\mathfrak{f} = \{\mathfrak{t}\}$, where $\mathfrak{t}$ is

$$\mathfrak{t} = \{\mathfrak{s}(x) \mapsto \langle \emptyset, \mathsf{ptr}_k(\mathfrak{s}(x), \mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k)) \;\Leftarrow\; \mathfrak{s}(x) \mapsto \langle \mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k) \rangle \rangle\} \,.$$

Observe that $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h}_1)$ and

$$\mathsf{ptr}_k(x, y_1, \ldots, y_k) = \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2).$$

Hence, by Lemma A.11, we have $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \mathsf{ptr}_k(x, y_1, \ldots, y_k)$. By definition of the predicate ptr_k, we conclude that $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi x \mapsto \langle y_1, \ldots, y_k \rangle$.

Case $\phi = \mathsf{pred}(z_1, \ldots, z_k)$**.** By Lemma 7.3, there exists a Φ-tree $\mathfrak{t}$ such that $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(z_1), \ldots, \mathfrak{s}(z_k))$, $\mathsf{allholepreds}(\mathfrak{t}) = \emptyset$, and $\mathsf{heap}(\{\mathfrak{t}\}) = \mathfrak{h}_1$. Let

$$\psi \triangleq \mathsf{pred}(\mathfrak{s}(z_1), \ldots, \mathfrak{s}(z_k))[\mathrm{dom}(\mathfrak{s}_{\max}^{-1})/\mathrm{img}(\mathfrak{s}_{\max}^{-1})] = \mathsf{project}(\mathfrak{s}, \{\mathfrak{t}\}).$$

Then, $\psi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$ and, by Lemma A.11, $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \psi$. Observe that while $\psi \neq \mathsf{pred}(\mathbf{z})$ is possible, we have by definition of $\mathfrak{s}_{\max}^{-1}$ that the parameters of the predicate call in ψ evaluate to the same locations as the parameters $\mathbf{z}$. Hence, $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \mathsf{pred}(z_1, \ldots, z_k)$.

Case $\phi = \phi_1 \wedge \phi_2$**.** We then have $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \phi_1$ and $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \phi_2$. By I.H., $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi_1$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi_2$. Hence, $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi_1 \wedge \phi_2$.

Cases $\phi = \phi_1 \vee \phi_2$**,** $\phi = \phi_1 \wedge \neg\phi_2$**.** Analogous to previous case.

Case $\phi = \phi_1 \star \phi_2$**.** By the semantics of $\star$, there exist heaps $\mathfrak{h}_{1,1}$ and $\mathfrak{h}_{1,2}$ such that $\langle \mathfrak{s}, \mathfrak{h}_{1,i} \rangle \models_\Phi \phi_i$ for $1 \leq i \leq 2$. Let $\mathcal{T}_i \triangleq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_{1,i})$. By Corollary 4.6, we have that $\langle \mathfrak{s}, \mathfrak{h}_{1,i} \rangle \in \mathbf{GStates}$ for $1 \leq i \leq 2$. By Corollary 8.19 we have that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \mathcal{T}_1 \bullet \mathcal{T}_2$. By Lemma 8.17, we have that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \neq \emptyset$. We can then apply Lemma A.12 to $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$, $\mathcal{T}_1$ and $\mathcal{T}_2$ in order to obtain states $(\mathfrak{s}, \mathfrak{h}_{2,1})$ and $(\mathfrak{s}, \mathfrak{h}_{2,2})$ with $\mathfrak{h}_2 = \mathfrak{h}_{2,1} \uplus \mathfrak{h}_{2,2}$, $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_{2,1}) = \mathcal{T}_1$, and $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_{2,2}) = \mathcal{T}_2$.
We can thus apply the I.H. to both $\mathfrak{h}_{1,1}$, $\mathfrak{h}_{1,2}$, ϕ_1 and $\mathfrak{h}_{2,1}$, $\mathfrak{h}_{2,2}$, ϕ_2 to conclude that $\langle \mathfrak{s}, \mathfrak{h}_{2,1} \rangle \models_\Phi \phi_1$ and $\langle \mathfrak{s}, \mathfrak{h}_{2,2} \rangle \models_\Phi \phi_2$. Since $\mathfrak{h}_{2,1} \uplus \mathfrak{h}_{2,2} = \mathfrak{h}_2$, the semantics of $\star$ then yields $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi$.

Case $\phi = \phi_0 \wedge (\phi_1 \circledast \phi_2)$**.** Then there exists a heap $\mathfrak{h}_0$ with $\langle \mathfrak{s}, \mathfrak{h}_0 \rangle \models_\Phi \phi_1$ and $\langle \mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_0 \rangle \models_\Phi \phi_2$.
Since $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle$ have the same type, we have $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_2)$. We can therefore assume w.l.o.g. that $\mathfrak{h}_2 \uplus \mathfrak{h}_0$ is defined—if this is not the case, simply replace $\mathfrak{h}_0$ with a heap $\mathfrak{h}_0'$ such that $\langle \mathfrak{s}, \mathfrak{h}_0 \rangle \cong \langle \mathfrak{s}, \mathfrak{h}_0' \rangle$ and both $\mathfrak{h}_1 \uplus \mathfrak{h}_0'$ and $\mathfrak{h}_2 \uplus \mathfrak{h}_0'$ are defined. Then, by Lemma 3.5, we can conclude that $(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_0') \models_\Phi \phi$.
By Corollary 4.6 we have $\langle \mathfrak{s}, \mathfrak{h}_0 \rangle, \langle \mathfrak{s}, \mathfrak{h}_1 \rangle \in \mathbf{GStates}$. Corollary 8.19 then yields that

$$\begin{aligned} \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_0) &= \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_0) \\ &= \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_0) \\ &= \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2 \uplus \mathfrak{h}_0). \end{aligned}$$

Now, we apply the I.H. for ϕ_0, $\mathfrak{h}_1$ and $\mathfrak{h}_2$ to conclude that $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi_0$, as well as for ϕ_2, $\langle \mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_0 \rangle$ and $\langle \mathfrak{s}, \mathfrak{h}_2 \uplus \mathfrak{h}_0 \rangle$ to conclude that $\langle \mathfrak{s}, \mathfrak{h}_2 \uplus \mathfrak{h}_0 \rangle \models_\Phi \phi_2$. Hence, by the semantics of $\circledast$ and $\wedge$, we have $\langle \mathfrak{s}, \mathfrak{h}_2 \rangle \models_\Phi \phi_0 \wedge (\phi_1 \circledast \phi_2)$.

Case $\phi = \phi_0 \wedge (\phi_1 \mathrel{-\!\!\star} \phi_2)$**.** Analogous to the previous case for guarded septraction, except that we must consider *arbitrary* models $\mathfrak{h}_0$ with $\langle \mathfrak{s}, \mathfrak{h}_0 \rangle \models_\Phi \phi_1$ and $\langle \mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_0 \rangle \models_\Phi \phi_2$. □

A.28 Proof of Lemma A.12 (type decomposability)

We need a couple of auxiliary definitions and lemmata before we can show this result in Lemma A.12 at the end of this section.

Definition A.13. *We call* $\mathfrak{f}$ $\mathfrak{s}$-decomposed *iff* $\mathfrak{f} = \mathsf{split}(\mathfrak{f}, \mathsf{img}(\mathfrak{s}))$.

Lemma A.14. *Let* $\langle \mathfrak{s}, \mathfrak{h} \rangle$ *be a state with* $\emptyset \neq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ *and let* $\mathcal{T}_1, \mathcal{T}_2 \in \mathbf{Types}_\Phi^{\mathsf{aliasing}(\mathfrak{s})}$ *be types with* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}_1 \bullet \mathcal{T}_2$ *Then there exist* $\mathfrak{s}$*-decomposed,* $\mathfrak{s}$*-delimited* Φ*-forests* $\mathfrak{f}, \mathfrak{f}_1, \mathfrak{f}_2$ *such that*

(1) $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) \in \mathcal{T}_i$, $1 \leq i \leq 2$,
(2) $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \mathfrak{f}$, *and*
(3) $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$.

Proof. By assumption we have $\emptyset \neq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. Take an arbitrary forest $\bar{\mathfrak{f}}$ with $\mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. By definition, $\bar{\mathfrak{f}}$ is $\mathfrak{s}$-delimited. Let $\mathfrak{f} \triangleq \mathsf{split}(\bar{\mathfrak{f}}, \mathsf{img}(\mathfrak{s}))$ be the $\mathfrak{s}$-decomposition of $\bar{\mathfrak{f}}$. By Lemma A.7, $\mathfrak{f}$ is $\mathfrak{s}$-delimited. By Lemma 7.13, $\mathfrak{f} \blacktriangleright^* \bar{\mathfrak{f}}$. Lemma 7.14 thus gives us that $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$. Hence, $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$.

By definition of $\bullet$, there exist formulas $\psi_1 \in \mathcal{T}_1$, $\psi_2 \in \mathcal{T}_2$ with $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \psi_1 \bullet_{\mathbf{P}} \psi_2$. By definition, there exist Φ-forests $\mathfrak{g}_1, \mathfrak{g}_2$ with $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_i) = \psi_i$. Because $\mathcal{T}_1 \bullet \mathcal{T}_2$ is defined, we have $\mathsf{alloced}(\mathcal{T}_1) \cap \mathsf{alloced}(\mathcal{T}_2) = \emptyset$, allowing us to assume w.l.o.g. that $\mathfrak{g}_1 \uplus \mathfrak{g}_2 \neq \bot$. By Theorem 7.39, there then exist forests $\mathfrak{f}_1, \mathfrak{f}_2$ with $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) = \psi_i$ and $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$. Because $\mathfrak{f}$ is $\mathfrak{s}$-decomposed, this implies that zero $\blacktriangleright$-steps were taken by $\bullet_{\mathbf{F}}$, i.e., $\mathfrak{f} = \mathfrak{f}_1 \uplus \mathfrak{f}_2$. Moreover, because $\mathfrak{f}$ is $\mathfrak{s}$-decomposed and $\mathfrak{s}$-delimited, we get that $\mathfrak{f}_1$ and $\mathfrak{f}_2$ are $\mathfrak{s}$-decomposed and $\mathfrak{s}$-delimited as well. □

Definition A.15 (Roots of a DUSH). *Let* $\phi = \exists \mathbf{e}.\, \forall \mathbf{a}.\, \bigstar_{1 \leq i \leq k} (\zeta_i \mathrel{-\!\!\star} \mathsf{pred}_i(\mathbf{z_i}))$ *be a DUSH. The* roots *of* ψ *are the set*

$$\mathsf{dushroots}_{\mathfrak{s}}(\phi) \triangleq \bigcup_{1 \leq i \leq k} \mathsf{aliasing}(\mathfrak{s})(\mathsf{predroot}(\mathsf{pred}_i(\mathbf{z_i}))).$$

Clearly, the roots of a forest are connected to the roots of a DUSH via the stack:

Lemma A.16. *Let* $\mathfrak{f}$ *be a* Φ*-forest. Then,* $\mathsf{dushroots}_{\mathfrak{s}}(\mathsf{project}(\mathfrak{s}, \mathfrak{f})) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{f})\}$.

Proof. Let $\phi \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. By definition of DUSHs, $\mathsf{roots}(\mathfrak{f}) \subseteq \mathsf{img}(\mathfrak{s})$. Every root $l \in \mathsf{roots}(\mathfrak{f})$ is therefore replaced by a variable in $\mathfrak{s}^{-1}_{\max}(l)$ by stack–forest projection. Since $\mathsf{dushroots}_{\mathfrak{s}}(\phi)$ closes the set of roots under all $\mathsf{aliasing}(\mathfrak{s})(\cdot)$, we obtain that $\mathsf{dushroots}_{\mathfrak{s}}(\phi)$ contains *all* variables x with $\mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{f})$. □

Lemma A.17. *Let* $\mathcal{T}$ *be a* Φ*-type and* $\psi \in \mathcal{T}$*. There exists a formula* $\psi' \in \mathcal{T}$ *such that* $\psi' \rhd^* \psi$ *and* $\mathsf{alloced}(\mathcal{T}) = \mathsf{dushroots}_{\mathfrak{s}}(\psi')$.

Proof. Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be such that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}$. By Lemma 8.16 we then have that $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \mathsf{alloced}(\mathcal{T})$ (†). By definition of Φ-types, there then exists a Φ-forest $\mathfrak{f}$ with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \psi$. Let $\bar{\mathfrak{f}} \triangleq \mathsf{split}(\mathfrak{f}, \mathsf{img}(\mathfrak{s}))$ be the $\mathfrak{s}$-decomposition of $\mathfrak{f}$ and write $\psi' \triangleq \mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}})$. We show that ψ' has the desired properties.

First, $\bar{\mathfrak{f}} \blacktriangleright^* \mathfrak{f}$ by Lemma 7.13. Observe that $\mathfrak{f} \in \bar{\mathfrak{f}} \bullet_{\mathbf{F}} \emptyset$ (where $\emptyset$ is the empty forest). Lemma 7.35 therefore guarantees that $\psi \in \psi' \bullet_{\mathbf{P}} \mathbf{emp}$ and thus $\psi' \rhd^* \psi$.

Second, by Lemma 7.14, $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathsf{heap}(\mathfrak{f})$ and thus $\psi' \in \mathcal{T}$. Moreover, by Lemma A.10, $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}$. We combine the above with (†) to derive $\mathsf{alloced}(\mathcal{T}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}$. Lemma A.16 then yields that $\mathsf{alloced}(\mathcal{T}) = \mathsf{dushroots}_{\mathfrak{s}}(\psi')$. □

Claim (Lemma A.12). Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state with $\emptyset \neq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}_1 \bullet \mathcal{T}_2$. Then, there exist $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathcal{T}_1 = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$, and $\mathcal{T}_2 = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$.

Proof. Let $\mathcal{T} \triangleq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. By Lemma A.14, there exist $\mathfrak{s}$-decomposed, $\mathfrak{s}$-delimited forests $\mathfrak{f}, \mathfrak{f}_1, \mathfrak{f}_2$ with

(1) $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) \in \mathcal{T}_i$, $1 \leq i \leq 2$,
(2) $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \mathfrak{f}$, and
(3) $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$.

Define $\mathfrak{h}_1 \triangleq \mathsf{heap}(\mathfrak{f}_1), \mathfrak{h}_2 \triangleq \mathsf{heap}(\mathfrak{f}_2)$. Then, $\mathfrak{h}_1 \uplus \mathfrak{h}_2 = \mathfrak{h}$ because of $\mathsf{heap}(\mathfrak{f}_1 \uplus \mathfrak{f}_2) = \mathfrak{h}$. Because the $\mathfrak{f}_i$ are $\mathfrak{s}$-delimited, we have $\mathsf{dangling}(\mathfrak{h}_i) \subseteq \mathsf{img}(\mathfrak{s})$. Hence, $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle, \langle \mathfrak{s}, \mathfrak{h}_2 \rangle \in \mathbf{GStates}$. By Lemma A.10 and Lemma 8.16 we have

$$\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{f})\} = \mathsf{alloced}(\mathcal{T}). \tag{*}$$

Further, we have

$$\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_i) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{f}_i)\} = \mathsf{alloced}(\mathcal{T}_i), \tag{†}$$

where the first equality follows from Lemma A.10, and the second equality holds by (*) and because the $\mathfrak{f}_i$ are $\mathfrak{s}$-decomposed.

We will show that $\mathcal{T}_1 = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$; the argument for $\mathcal{T}_2 = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$ is symmetrical. We prove the inclusions $\mathcal{T}_1 \subseteq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathcal{T}_1 \supseteq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ separately.

"$\mathcal{T}_1 \subseteq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$:" Let $\psi_1 \in \mathcal{T}_1$. By Lemma A.17, there exists a formula $\psi_1' \in \mathcal{T}_1$ such that $\psi_1' \rhd^* \psi_1$, and $\mathsf{alloced}(\mathcal{T}_1) = \mathsf{dushroots}_\mathfrak{s}(\psi_1')$ (#). Let $\psi_2' \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \in \mathcal{T}_2$. By the definition of projections, we have $\psi_i' = \exists \mathbf{e_i}.\, \forall \mathbf{a_i}.\, \phi_i$ for some $\mathbf{e_i}, \mathbf{a_i}, \phi_i$, where we can assume w.l.o.g. that $\mathbf{e_1}, \mathbf{e_2}, \mathbf{a_1}, \mathbf{a_2}$ are pairwise disjoint. By definition of type composition, $\bullet$, it follows that $\psi \triangleq \exists \mathbf{e_1}, \mathbf{e_2}.\, \forall \mathbf{a_1}, \mathbf{a_2}.\, \phi_1 \star \phi_2 \in \mathcal{T}$. Then, there is an Φ-forest $\mathfrak{g} \in \mathsf{forests}_\Phi(\mathfrak{s}, \mathfrak{h})$ such that $\mathsf{project}(\mathfrak{s}, \mathfrak{g}) = \psi$. From (#), (*) and (†) we obtain that $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \mathsf{dushroots}_\mathfrak{s}(\psi)$. By Theorem 8.13 there exist Φ-forests $\mathfrak{g}_1, \mathfrak{g}_2$ with $\mathfrak{g} = \mathfrak{g}_1 \uplus \mathfrak{g}_2$, $\mathsf{heap}(\mathfrak{g}_i) = \mathfrak{h}_i$ and $\mathfrak{g} \in \mathfrak{g}_1 \bullet_\mathbf{F} \mathfrak{g}_2$. With $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_1 \uplus \mathfrak{g}_2) = \exists \mathbf{e_1}, \mathbf{e_2}.\, \forall \mathbf{a_1}, \mathbf{a_2}.\, \phi_1 \star \phi_2$ we then must have that $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_i) = \psi_i'$. Therefore, $\psi_1' \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$. Since $\psi_1' \rhd^* \psi_1$, Lemma A.4 then gives us a forest $\mathfrak{g}_1'$ s.t. $\mathfrak{g}_1 \blacktriangleright^* \mathfrak{g}_1'$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_1') = \psi_1$. Because also $\mathsf{heap}(\mathfrak{g}_1') = \mathsf{heap}(\mathfrak{g}_1)$ by Lemma 7.14, $\mathfrak{g}_1' \in \mathsf{forests}_\Phi(\mathfrak{h}_1)$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_1') \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$. Thus, $\psi_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$.

"$\mathcal{T}_1 \supseteq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$:" Let $\psi_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$. By Lemma A.17, there exists a formula $\psi_1' \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ such that $\psi_1' \rhd^* \psi_1$, and $\mathsf{alloced}(\mathcal{T}_1) = \mathsf{dushroots}_\mathfrak{s}(\psi_1')$ (#). Let $\psi_2' \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. By the definition of projections, we have $\psi_i' = \exists \mathbf{e_i}.\, \forall \mathbf{a_i}.\, \phi_i$ for some $\mathbf{e_i}, \mathbf{a_i}, \phi_i$, where we can assume w.l.o.g. that $\mathbf{e_1}, \mathbf{e_2}, \mathbf{a_1}, \mathbf{a_2}$ are pairwise disjoint. By Corollary 8.19 we then have that $\psi \triangleq \exists \mathbf{e_1}, \mathbf{e_2}.\, \forall \mathbf{a_1}, \mathbf{a_2}.\, \phi_1 \star \phi_2 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2) = \mathcal{T}$. From (#), (*) and (†) we obtain that $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \mathsf{dushroots}_\mathfrak{s}(\psi)$. Because of $\mathcal{T} = \mathcal{T}_1 \bullet \mathcal{T}_2$ there are some $\psi_i'' \in \mathcal{T}_i$ such that $\psi \in \psi_1'' \bullet_\mathbf{P} \psi_2''$. Because of $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_i) = \mathsf{alloced}(\mathcal{T}_i)$ we must have that $\psi_i'' = \psi_i'$. Therefore, $\psi_1' \in \mathcal{T}_1$. We now consider some forest $\mathfrak{g}_1$ with $\mathsf{type}_\Phi(\mathfrak{s}, \mathsf{heap}(\mathfrak{g}_1)) = \mathcal{T}_1$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_1') = \psi_1'$. Since $\psi_1' \rhd^* \psi_1$, Lemma A.4 then gives us a forest $\mathfrak{g}_1'$ s.t. $\mathfrak{g}_1 \blacktriangleright^* \mathfrak{g}_1'$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_1') = \psi_1$. Because also $\mathsf{heap}(\mathfrak{g}_1') = \mathsf{heap}(\mathfrak{g}_1)$ by Lemma 7.14, we get that $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_1) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathsf{heap}(\mathfrak{g}_1)) = \mathcal{T}_1$. Hence, $\psi_1 \in \mathcal{T}_1$. □

A.29 Correctness of the Fixed Point Algorithm For Computing Types of Predicate Calls

A.29.1 Soundness of the Type Computation. We organize the soundness proof into a sequence of simple lemmata about the base cases of the fixed point algorithm and about the operations $\bullet$, $\cdot[\cdot : \cdot/\cdot]$, forget and extend. The soundness of the overall algorithm is then be a direct consequence of these lemmata. We first characterize the types of atomic formulas.

Lemma A.18. *For all aliasing constraints* $\mathbf{ac}$, $\mathbf{Types}^{\mathbf{ac}}_\Phi(\mathbf{emp}) = \{\{\mathbf{emp}\}\}$.

Proof. Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state with $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\mathbf{emp})$ and $\mathsf{aliasing}(\mathfrak{s}) = \mathbf{ac}$. By definition, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathbf{emp}$ and thus $\mathfrak{h} = \emptyset$. We now argue that $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \{\mathbf{emp}\}$.

We note that $\emptyset \in \mathsf{forests}_\Phi(\mathfrak{h})$ ($\emptyset$ is the forest that does not contain any trees) and $\mathbf{emp} = \mathsf{project}(\mathfrak{s}, \{\emptyset\})$. Hence, $\mathbf{emp} \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$.

Conversely, let $\psi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. By definition, there is a Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$ with $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \psi$ and $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$. Since $\mathfrak{h} = \emptyset$, we have $\mathsf{heap}(\mathfrak{f}) = \emptyset$ and thus $\mathsf{heap}(\mathfrak{t}_i) = \emptyset$ for all $i \in [1, k]$. Hence, $k = 0$. By Definition 7.23 (stack projections), we then have $\psi = \mathbf{emp}$. □

Lemma A.19. *Let* $\mathbf{ac}$ *be a aliasing constraint and* $x, y \in \mathrm{dom}(\mathbf{ac})$.

- *If* $\langle x, y \rangle \in \mathbf{ac}$, *then* $\mathbf{Types}^{\mathbf{ac}}_\Phi(x \approx y) = \{\{\mathbf{emp}\}\}$ *and* $\mathbf{Types}^{\mathbf{ac}}_\Phi(x \not\approx y) = \emptyset$.

- *Otherwise,* $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(x \not\approx y) = \{\{\mathbf{emp}\}\}$ *and* $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(x \approx y) = \emptyset$.

Proof. We only consider the case $x \approx y$ as the argument for $x \not\approx y$ is completely analogous. If $\langle x, y\rangle \in \mathbf{ac}$, our semantics of equalities enforces $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\mathbf{emp}) = \mathbf{Types}^{\mathbf{ac}}_{\Phi}(x \approx y)$. The claim then follows from Lemma A.18. If $\langle x, y\rangle \notin \mathbf{ac}$, it holds for all $\mathfrak{s}$ with $\mathrm{aliasing}(\mathfrak{s}) = \mathbf{ac}$ that $\mathfrak{s}(x) \neq \mathfrak{s}(y)$. The semantics of $x \approx y$ then yields, for all heaps $\mathfrak{h}$, that $\langle \mathfrak{s}, \mathfrak{h}\rangle \not\models_{\Phi} x \approx y$. Hence, $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(x \approx y) = \emptyset$. □

Lemma A.20. *Let* $\mathbf{ac}$ *be an aliasing constraint, let* $a \in \mathrm{dom}(\mathbf{ac})$*, and let* $\mathbf{b} \in \mathbf{Var}^*$ *with* $\mathbf{b} \subseteq \mathrm{dom}(\mathbf{ac})$*. Then,* $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(a \mapsto \mathbf{b}) = \{\mathrm{type}_{\Phi}(\mathrm{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b}))\}$.

Proof. **"⊇"** Let $\langle \mathfrak{s}, \mathfrak{h}\rangle \triangleq \mathrm{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b})$. By definition, $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} a \mapsto \mathbf{b}$. Hence, $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(a \mapsto \mathbf{b})$.

"⊆" Let $\langle \mathfrak{s}, \mathfrak{h}\rangle$ be a state such that $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(a \mapsto \mathbf{b})$ and $\mathrm{aliasing}(\mathfrak{s}) = \mathbf{ac}$. By definition, $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} a \mapsto \mathbf{b}$ and thus, by the semantics of points-to assertions, $\mathfrak{h} = \{\mathfrak{s}(a) \mapsto \mathfrak{s}(\mathbf{b})\}$. Consequently, $\langle \mathfrak{s}, \mathfrak{h}\rangle \cong \mathrm{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b})$ and $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) = \mathrm{type}_{\Phi}(\mathrm{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b}))$. Since $\langle \mathfrak{s}, \mathfrak{h}\rangle$ was an arbitrary model of ϕ with $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(a \mapsto \mathbf{b})$ and $\mathrm{aliasing}(\mathfrak{s}) = \mathbf{ac}$, we have $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(a \mapsto \mathbf{b}) \subseteq \{\mathrm{type}_{\Phi}(\mathrm{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b}))\}$. □

We next consider the operations • (type composition), $\cdot[\cdot : \cdot/\cdot]$ (variable renaming), forget and extend (lifted to sets of types). The lemmata for these operations below are more general than what is needed for the soundness of our fixed point algorithm because we will also use them for proving the correctness of our algorithm dealing with guarded formulas, see Section 9.2.

Lemma A.21 (Type composition). *For* $\phi_1, \phi_2 \in \mathbf{GSL}$, $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_1 \star \phi_2) = \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_1) \bullet \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_2)$.

Proof. We show each inclusion separately.

- Let $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_1 \star \phi_2)$. Moreover, fix a state $\langle \mathfrak{s}, \mathfrak{h}\rangle$ be such that $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \phi_1 \star \phi_2$ and $\mathcal{T} = \mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$. Then, there exist heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\langle \mathfrak{s}, \mathfrak{h}_i\rangle \models_{\Phi} \phi_i$ and $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$. By Corollary 4.6, we have $\langle \mathfrak{s}, \mathfrak{h}_1\rangle, \langle \mathfrak{s}, \mathfrak{h}_2\rangle \in \mathbf{GStates}$. By Corollary 8.19, $\mathcal{T} = \mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_2)$. Since $\langle \mathfrak{s}, \mathfrak{h}_i\rangle \models_{\Phi} \phi_i$, we have $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_i) \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_i)$. Hence, $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_1) \bullet \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_2)$.
- Let $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_1) \bullet \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_2)$. Then, there are $\mathcal{T}_1 \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_1)$ and $\mathcal{T}_2 \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_2)$ such that $\mathcal{T} = \mathcal{T}_1 \bullet \mathcal{T}_2$. Moreover, there are states $\langle \mathfrak{s}, \mathfrak{h}_i\rangle$ such that $\mathrm{aliasing}(\mathfrak{s}) = \mathbf{ac}$, $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_i) = \mathcal{T}_i$ and $\langle \mathfrak{s}, \mathfrak{h}_i\rangle \models_{\Phi} \phi_i$. By Lemma 4.5 we have $\mathcal{T}_i \neq \emptyset$. By Lemma 8.20 there are states $\langle \mathfrak{s}, \mathfrak{h}'_i\rangle$ such that $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}'_i) = \mathcal{T}_i$ and $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}'_1 \uplus \mathfrak{h}'_2) = \mathcal{T}_1 \bullet \mathcal{T}_2$. By Corollary 8.30 we have $\langle \mathfrak{s}, \mathfrak{h}'_i\rangle \models_{\Phi} \phi_i$. By the semantics of $\star$, $\langle \mathfrak{s}, \mathfrak{h}'_1 \uplus \mathfrak{h}'_2\rangle \models_{\Phi} \phi_1 \star \phi_2$. Hence, $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi_1 \star \phi_2)$. □

Lemma A.22 (Renaming of type sets). *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be sequences of variables as in Definition 9.1 from above. Then, for every* GSL *formula* ϕ*, we have*

$$\mathbf{Types}^{\mathbf{ac}[\mathbf{x}/\mathbf{y}]^{-1}}_{\Phi}(\phi)[\mathbf{ac} : \mathbf{x}/\mathbf{y}] = \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi[\mathbf{x}/\mathbf{y}]).$$

Proof. Let $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi[\mathbf{x}/\mathbf{y}])$. By definition, this means $\mathrm{aliasing}(\mathfrak{s}) = \mathbf{ac}$. Moreover, we note that $\mathbf{ac}[\mathbf{x}/\mathbf{y}]^{-1} = \mathrm{aliasing}(\mathfrak{s}[\mathbf{x}/\mathbf{y}])$. By Lemma 8.22, we have $\mathrm{type}_{\Phi}(\mathfrak{s}[\mathbf{x}/\mathbf{y}], \mathfrak{h})[\mathbf{ac} : \mathbf{x}/\mathbf{y}] = \mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$, i.e., $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Types}^{\mathbf{ac}[\mathbf{x}/\mathbf{y}]^{-1}}_{\Phi}(\phi)[\mathbf{ac} : \mathbf{x}/\mathbf{y}]$. The converse direction is analogous. □

Lemma A.23 (Forgetting a variable in type sets). *Let* $\phi \in \mathbf{GSL}$ *be a formula with free variables* $\mathbf{x} \cup \{y\}$ *such that* $y \notin \mathbf{x}$*. Moreover, assume that, for every state* $\langle \mathfrak{s}, \mathfrak{h}\rangle$*,* $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_{\Phi} \phi$ *implies* $\mathfrak{s}(y) \in \mathrm{dom}(\mathfrak{h})$*. Then, for every aliasing constraint* $\mathbf{ac}$ *with* $\mathrm{dom}(\mathbf{ac}) = \mathbf{x}$*, we have*

$$\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\exists y.\, \phi) = \bigcup_{\mathbf{ac}' \in \mathbf{AC}^{\mathbf{x} \cup \{y\}} \text{ with } \mathbf{ac}'|_{\mathbf{x}} = \mathbf{ac}} \mathrm{forget}_{\mathbf{ac}', y}(\mathbf{Types}^{\mathbf{ac}'}_{\Phi}(\phi)).$$

Proof. Let $\mathcal{T} \in \mathsf{forget}_{\mathbf{ac}',y}(\mathbf{Types}_\Phi^{\mathbf{ac}'}(\phi))$, where $\mathbf{ac}' \in \mathbf{AC}^{\mathbf{x}\cup\{y\}}$ is an aliasing constraint satisfying $\mathbf{ac}'|_{\mathbf{x}} = \mathbf{ac}$. Then there exists a state $\langle \mathfrak{s}, \mathfrak{h}\rangle$ such that $\mathcal{T} = \mathsf{forget}_{\mathbf{ac}',y}(\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}))$, $\langle \mathfrak{s}, \mathfrak{h}\rangle \models_\Phi \phi$, $\mathrm{dom}(\mathfrak{s}) = \mathbf{x}$, and $\mathsf{aliasing}(\mathfrak{s})|_{\mathrm{dom}(\mathbf{ac})} = \mathbf{ac}$. By assumption, $\mathfrak{s}(y) \in \mathrm{dom}(\mathfrak{h})$. Lemma 8.24 then yields

$$\mathsf{type}_\Phi(\mathfrak{s}[y/\bot],\mathfrak{h}) = \mathsf{forget}_{\mathbf{ac}',y}(\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})) = \mathcal{T}.$$

By the semantics of existential quantifiers, we have $\langle \mathfrak{s}[y/\bot],\mathfrak{h}\rangle \models_\Phi \exists y.\,\phi$. Hence, $\mathcal{T} \in \mathbf{Types}_\Phi^{\mathbf{ac}}(\exists y.\,\phi)$.

Conversely, let $\mathcal{T} \in \mathbf{Types}_\Phi^{\mathbf{ac}}(\exists y.\,\phi)$. Then, there is a state $\langle \mathfrak{s},\mathfrak{h}\rangle$ such that $\langle \mathfrak{s},\mathfrak{h}\rangle \models_\Phi \exists y.\,\phi$, $\mathcal{T} = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$, $\mathrm{dom}(\mathfrak{s}) = \mathbf{x}\setminus\{y\}$ and $\mathsf{aliasing}(\mathfrak{s}) = \mathbf{ac}$. By the semantics of the existential quantifier, there is a location v such that $(\mathfrak{s}[y/v],\mathfrak{h}) \models_\Phi \phi$. By assumption we have $\mathfrak{s}(y) \in \mathrm{dom}(\mathfrak{h})$. Then, for $\mathbf{ac}' = \mathsf{aliasing}(\mathfrak{s}[y/v]) \in \mathbf{AC}^{\mathbf{x}\cup\{y\}}$, Lemma 8.24 yields

$$\mathcal{T} = \mathsf{forget}_{\mathbf{ac}',y}(\mathsf{type}_\Phi(\mathfrak{s}[y/v],\mathfrak{h})) = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \in \mathsf{forget}_{\mathbf{ac}',y}(\mathbf{Types}_\Phi^{\mathbf{ac}'}(\phi)). \qquad \square$$

Lemma A.24 (Extending type sets). *Let $\phi \in \mathbf{GSL}$ be a formula with free variables $\mathbf{x}$. Moreover, let $\mathbf{ac} \subseteq \mathbf{ac}'$ be alias constraints with $\mathrm{dom}(\mathbf{ac}) = \mathbf{x}$. Then, $\mathbf{Types}_\Phi^{\mathbf{ac}'}(\phi) \subseteq \mathsf{extend}_{\mathbf{ac}'}(\mathbf{Types}_\Phi^{\mathbf{ac}}(\phi))$.*

Proof. We consider some $\mathcal{T} \in \mathbf{Types}_\Phi^{\mathbf{ac}'}(\phi)$. Then, there is a state $\langle \mathfrak{s},\mathfrak{h}\rangle$ with $\mathrm{dom}(\mathfrak{s}) = \mathbf{x} = \mathrm{dom}(\mathbf{ac})$ and $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) = \mathcal{T}$. We now choose some extension $\mathfrak{s}'$ of $\mathfrak{s}$ to $\mathrm{dom}(\mathbf{ac}')$ such that $\mathfrak{s}(x) \notin \mathrm{locs}(\mathfrak{h})$ for every variable $x \in \mathrm{dom}(\mathfrak{s})$ that is not an alias of a variable in $\mathrm{dom}(\mathbf{ac})$. By Lemma 8.28 we then have $\mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h}) = \mathsf{extend}_{\mathbf{ac}'}(\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}))$. Hence, $\mathsf{extend}_{\mathbf{ac}'}(\mathcal{T}) \in \mathsf{extend}_{\mathbf{ac}'}(\mathbf{Types}_\Phi^{\mathbf{ac}}(\phi))$. $\square$

We are now ready to prove the soundness of the fixed point computation, i.e.,

$$\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred},\mathbf{ac}) \subseteq \mathbf{Types}_\Phi^{\mathbf{ac}}(\mathsf{pred}).$$

We first need to establish that $\mathsf{ptypes}_p^{\mathbf{x}}(\mathbf{ac},\phi)$ is sound when ϕ is a rule of predicate pred, i.e., that $\mathsf{ptypes}_p^{\mathbf{x}}(\mathbf{ac},\phi) \subseteq \mathbf{Types}_\Phi^{\mathbf{ac}}(\phi)$ holds, under the assumption that p maps every pair of predicate identifier and aliasing constraint to a subset of the corresponding types. As SID rules are guaranteed to be existentially-quantified symbolic heaps, it suffices to prove this result for arbitrary $\phi \in \mathbf{SH}^\exists$:

Lemma A.25. *Let $\phi \in \mathbf{SH}^\exists$ and $\mathbf{ac} \in \mathbf{AC}$ with $\mathrm{dom}(\mathbf{ac}) \supseteq \mathsf{fvars}(\phi)$. Moreover, let*

$$p\colon \mathbf{Preds}(\Phi) \times \mathbf{AC} \to 2^{\mathbf{Types}_\Phi}$$

be such that for all $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ and all $\mathbf{ac}' \in \mathbf{AC}^{\mathbf{x}\cup\mathsf{fvars}(\mathsf{pred})}$, it holds that $p(\mathsf{pred},\mathbf{ac}') \subseteq \mathbf{Types}_\Phi^{\mathbf{ac}'}(\mathsf{pred})$. Then $\mathsf{ptypes}_p^{\mathbf{x}}(\phi,\mathbf{ac}) \subseteq \mathbf{Types}_\Phi^{\mathbf{ac}}(\phi)$.

Proof. We proceed by induction on the structure of ϕ and apply the lemmata from above.

Cases $\phi = x \approx y$, $\phi = x \not\approx y$. The claim follows from Lemma A.19.

Case $\phi = a \mapsto \mathbf{b}$. The claim follows from Lemma A.20.

Case $\phi = \mathsf{pred}(\mathbf{y})$. Let $\mathbf{z} = \mathsf{fvars}(\mathsf{pred})$. By I.H., we have

$$p(\mathsf{pred},\ \mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}\big|_{\mathbf{z}\cup\mathbf{x}}) \subseteq \mathbf{Types}_\Phi^{\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}|_{\mathbf{z}\cup\mathbf{x}}}(\mathsf{pred}).$$

By Lemma A.24, we have

$$\mathsf{extend}_{\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}}(\mathbf{Types}_\Phi^{\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}|_{\mathbf{z}\cup\mathbf{x}}}(\mathsf{pred})) \subseteq \mathbf{Types}_\Phi^{\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}}(\mathsf{pred}).$$

Moreover, by Lemma A.22 we have

$$\mathbf{Types}_\Phi^{\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}}(\mathsf{pred})[\mathbf{ac}:\mathbf{z}/\mathbf{y}] = \mathbf{Types}_\Phi^{\mathbf{ac}}(\mathsf{pred}[\mathbf{z}/\mathbf{y}]).$$

Hence, we get

$$\mathsf{extend}_{\mathbf{ac}}(p(\mathsf{pred},\ \mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}\big|_{\mathbf{z}\cup\mathbf{x}}))[\mathbf{ac}:\mathbf{z}/\mathbf{y}] \subseteq \mathbf{Types}_\Phi^{\mathbf{ac}}(\mathsf{pred}[\mathbf{z}/\mathbf{y}]).$$

Case $\phi = \phi_1 \star \phi_2$**.** The claim follows from Lemma A.21 and the induction hypothesis.
Case $\phi = \exists y.\, \phi$**.** The claim follows from Lemma A.23 and the induction hypothesis.

□

Lemma A.26 (Soundness of type computation). $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \mathbf{ac}) \subseteq \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\mathsf{pred})$.

Proof. A straightforward induction on top of Lemma A.25. □

A.29.2 Completeness of the Type Computation. We now establish the completeness of the fixed point computation, i.e., $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \mathbf{ac}) \supseteq \mathbf{Types}^{\mathbf{ac}}_{\Phi}(\mathsf{pred})$. The main challenge is our treatment of predicate calls $\mathsf{ptypes}^{\mathbf{x}}_{p}(\mathsf{pred}(\mathbf{y}), \mathbf{ac})$, for which the recursive look-up $p(\mathsf{pred}, \mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}\big|_{\mathbf{x}\cup\mathbf{z}})$ restricts the stack-aliasing constraint $\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}$ to $\mathbf{x} \cup \mathbf{z}$, where $\mathbf{z} \triangleq \mathsf{fvars}(\mathsf{pred})$. This restriction of the variables is necessary in order to avoid having to consider larger and larger sequences of variables, which would lead to divergence as we illustrate below. Hence, our goal will be to establish that ptypes discovers all types even though we restrict the variables in the recursive look-up.

We now illustrate the need for restricting the variables in the recursive look-up: We assume a stack $\mathfrak{s}$ with $\mathsf{dom}(\mathfrak{s}) = \mathbf{x} \cup \mathbf{y}$ and pick a rule

$$\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \Leftarrow \exists \mathbf{e}.\, (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k}).$$

We extend $\mathfrak{s}$ to a stack $\mathfrak{s}'$ with $\mathsf{dom}(\mathfrak{s}') = \mathsf{dom}(\mathfrak{s}) \cup \mathbf{e}$ and are left with computing the types of

$$\big((a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k})\big)[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}].$$

At a first glance, this implies recursively computing the types of the calls, $\mathsf{pred}_i(\mathbf{z_i})$, w.r.t. the variables $\mathbf{x}' \triangleq \mathbf{x} \cup \mathbf{y} \cup \mathbf{e}$, i.e., we additionally have to consider the existentially quantified variables $\mathbf{e}$. We attempt to do so by picking a rule for each predicate, say we first pick a rule for predicate pred_i. Then, we need to consider an extension $\mathfrak{s}''$ of $\mathfrak{s}'$ with $\mathsf{dom}(\mathfrak{s}'') = \mathsf{dom}(\mathfrak{s}') \cup \mathbf{e_i}$ for the existentially quantified variables $\mathbf{e_i}$ on the right hand side of the picked rule. Continuing in this fashion, the computation diverges as we have to extend the set of considered variables $\mathbf{x}$ again and again.

However, a more careful analysis reveals that restricting the aliasing constraints to $\mathbf{ac}[\mathbf{z}/\mathbf{y}]^{-1}\big|_{\mathbf{x}\cup\mathbf{z}}$ for the recursive look-up (followed by extending and renaming the obtained set of types) is sufficient. This is a consequence of the *establishment* property, which we require for all SIDs in $\mathbf{ID}_{\mathrm{btw}}$. We formalize this insight in the notion of a *tree closure*, which restricts the locations a subtree can share with the variables appearing in the rule instance of its parent node:

Definition A.27 (Tree Closure). *Let* $\mathbf{u}$ *be a set of locations and let* t *be a* Φ*-tree. Moreover, let* $\mathsf{t}_{\mathsf{sub}}$ *be a proper subtree of* t *and let* $\mathsf{pred}(\mathbf{w}) = \mathsf{rootpred}(\mathsf{t}_{\mathsf{sub}})$*. Let* $\ell \in \mathsf{dom}(\mathsf{t})$ *be the parent location of the root of* $\mathsf{t}_{\mathsf{sub}}$ *and let* $\mathsf{rule}_{\mathsf{t}}(\ell) = \mathsf{pred}(\mathbf{v}) \Leftarrow \phi[\mathsf{fvars}(\mathsf{pred}) \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{m}]$ *be the rule instance at location* ℓ*. We say* t *is* $\mathbf{u}$*-closed for* $\mathsf{t}_{\mathsf{sub}}$*, if* $\mathsf{ptrlocs}(\mathsf{t}_{\mathsf{sub}}) \cap (\mathbf{v} \cup \mathbf{m}) \subseteq \mathbf{w} \cup \mathbf{u}$.

Furthermore, we say t *is* $\mathbf{u}$*-closed, if* t *is* $\mathbf{u}$*-closed for all proper subtrees of* t.

Example A.28. *The tree from Fig. 6b is* $\langle\rangle$*-closed, where* $\langle\rangle$ *is the empty sequence. If we replace location* 8 *everywhere in the tree with location* 7*, then the resulting tree is not* $\langle\rangle$*-closed anymore (consider the subtree rooted at location* 2*), but* 7*-closed.*

Lemma A.29. *Let* t *be some* Φ*-tree with* $\mathsf{allholepreds}(\mathsf{t}) = \emptyset$ *and* $\mathsf{pred}(\mathbf{u}) = \mathsf{rootpred}(\mathsf{t})$*. Let* $\ell \in \mathsf{dom}(\mathsf{t})$ *be some location and let* $\mathsf{rule}_{\mathsf{t}}(\ell) = \mathsf{pred}(\mathbf{v}) \Leftarrow \phi[\mathsf{fvars}(\mathsf{pred}) \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{m}]$ *be the rule instance at location* ℓ *of* t*. Then, we have* $\mathbf{v} \cup \mathbf{m} \subseteq \mathsf{dom}(\mathsf{t}) \cup \mathbf{u}$.

Proof. A direct consequence of establishment. □

Recall that, by Lemma 7.3, we have $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$ if and only if there exists a Φ-tree $\mathfrak{t}$ with $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(\mathsf{fvars}(\mathsf{pred})))$, $\mathsf{allholes}(\mathfrak{t}) = \emptyset$ and $\mathsf{heap}(\{\mathfrak{t}\}) = \mathfrak{h}$. The completeness of our fixed point algorithm relies on the observation that such trees $\mathfrak{t}$ are $\mathfrak{s}(\mathsf{fvars}(\mathsf{pred}))$-closed:

Lemma A.30. *Every Φ-tree $\mathfrak{t}$ with $\mathsf{allholepreds}(\mathfrak{t}) = \emptyset$ and $\mathsf{pred}(\mathbf{u}) = \mathsf{rootpred}(\mathfrak{t})$ is $\mathbf{u}$-closed.*

Proof. Let $\mathfrak{t}_{\mathsf{sub}}$ be a proper subtree of $\mathfrak{t}$ and let $\mathsf{pred}(\mathbf{w}) = \mathsf{rootpred}(\mathfrak{t}_{\mathsf{sub}})$. Let $\ell \in \mathsf{dom}(\mathfrak{t})$ be the parent location of the root of $\mathfrak{t}_{\mathsf{sub}}$ and let $\mathsf{rule}_{\mathfrak{t}}(\ell) = \mathsf{pred}(\mathbf{v}) \Leftarrow \phi[\mathsf{fvars}(\mathsf{pred}) \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{m}]$ be the rule instance at location ℓ. We consider some $k \in \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap (\mathbf{v} \cup \mathbf{m})$. Then k is either dangling or an allocated location in $\mathsf{heap}(\mathfrak{t}_{\mathsf{sub}})$.

Assume k is dangling in $\mathsf{heap}(\mathfrak{t}_{\mathsf{sub}})$: We define the stack $\mathfrak{s} : \mathsf{fvars}(\mathsf{pred}) \to \mathbf{Loc}$ by setting $\mathfrak{s}(\mathsf{fvars}(\mathsf{pred}_{\mathsf{sub}})) = \mathbf{w}$. By Lemma 7.3, we have $\langle \mathfrak{s}, \mathsf{heap}(\mathfrak{t}_{\mathsf{sub}}) \rangle \models_\Phi \mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$. By Lemma 4.4 we have $\langle \mathfrak{s}, \mathsf{heap}(\mathfrak{t}_{\mathsf{sub}}) \rangle \in \mathbf{GStates}$. Hence, $k \in \mathsf{img}(\mathfrak{s}) = \mathbf{w}$. In other words, dangling locations do not invalidate that $\mathfrak{t}$ is $\mathbf{u}$-closed.

Assume k is allocated in $\mathsf{heap}(\mathfrak{t}_{\mathsf{sub}})$, i.e., $k \in \mathsf{dom}(\mathfrak{t}_{\mathsf{sub}})$: To prove that $\mathfrak{t}$ is $\mathbf{u}$-closed, it is sufficient that $k \notin \mathbf{w}$ implies that $k \in \mathbf{u}$. Hence, let us assume that $k \notin \mathbf{w}$. Let $\mathfrak{t}_{\mathsf{rem}} = \mathfrak{t} \setminus \mathfrak{t}_{\mathsf{sub}}$ be the remainder of $\mathfrak{t}$ after splitting off $\mathfrak{t}_{\mathsf{sub}}$, i.e., $\mathsf{split}(\{\mathfrak{t}\}, \{\mathsf{root}(\mathfrak{t}_{\mathsf{sub}})\}) = \{\mathfrak{t}_{\mathsf{sub}}, \mathfrak{t}_{\mathsf{rem}}\}$. We note that $k \notin \mathsf{dom}(\mathfrak{t}_{\mathsf{rem}})$ because a location cannot be allocated in two subtrees. We choose a fresh location $k' \in \mathbf{Loc} \setminus \mathsf{ptrlocs}(\mathfrak{t})$ and then create a tree $\mathfrak{t}'_{\mathsf{sub}}$ as a copy of $\mathfrak{t}_{\mathsf{sub}}$ except that we replace every occurrence of location k in $\mathfrak{t}_{\mathsf{sub}}$ with k'. We note that $\mathsf{rootpred}(\mathfrak{t}'_{\mathsf{sub}}) = \mathsf{pred}_{\mathsf{sub}}(\mathbf{w}) = \mathsf{rootpred}(\mathfrak{t}_{\mathsf{sub}})$ because of $k \notin \mathbf{w}$. Hence, there is a tree $\mathfrak{t}'$ such that $\mathsf{split}(\{\mathfrak{t}'\}, \{\ell\}) = \{\mathfrak{t}'_{\mathsf{sub}}, \mathfrak{t}_{\mathsf{rem}}\}$. We note that, by construction of $\mathfrak{t}'$, we have that (1) $k' \notin \mathsf{dom}(\mathfrak{t}')$, (2) $\mathsf{rootpred}(\mathfrak{t}') = \mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathbf{u})$, (3) $\mathsf{allholepreds}(\mathfrak{t}') = \emptyset$, (4) ℓ is the parent of $\mathfrak{t}'_{\mathsf{sub}}$ in $\mathfrak{t}'$, and (5) $\mathsf{rule}_{\mathfrak{t}'}(\ell) = \mathsf{pred}(\mathbf{v}) \Leftarrow \phi[\mathsf{fvars}(\mathsf{pred}) \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{m}]$. Lemma A.29 then yields $\mathbf{v} \cup \mathbf{m} \subseteq \mathsf{dom}(\mathfrak{t}') \cup \mathbf{u}$. Since k is allocated, this means $k \in \mathsf{dom}(\mathfrak{t}') \cup \mathbf{u}$. With (1) we then obtain $k \in \mathbf{u}$. □

We are now ready to prove the completeness of our fixed-point algorithm for computing types. We will show that the fixed-point algorithm discovers, for all predicates pred and aliasing constraints $\mathbf{ac} \in \mathbf{AC}^{\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})}$, all types in the following set:

$$\{\mathsf{type}_\Phi(\mathfrak{s}, \mathsf{heap}(\mathfrak{t})) \mid \mathsf{aliasing}(\mathfrak{s}) = \mathbf{ac}, \mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(\mathsf{fvars}(\mathsf{pred}))), \mathsf{allholes}(\mathfrak{t}) = \emptyset\},$$

where—as shown above—we can rely on the assumption that the considered trees $\mathfrak{t}$ are $\mathfrak{s}(\mathbf{x})$-closed.

Lemma A.31. *Let $\mathfrak{s}$ be a stack with $\mathsf{dom}(\mathfrak{s}) = \mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})$ and $\mathfrak{s}(\mathsf{fvars}(\mathsf{pred})) = \mathbf{v}$. Let $\mathfrak{t}$ be an $\mathfrak{s}(\mathbf{x})$-closed Φ-tree with $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathbf{v})$ and $\mathsf{allholepreds}(\mathfrak{t}) = \emptyset$. Then,*

$$\mathsf{type}_\Phi(\mathfrak{s}, \mathsf{heap}(\mathfrak{t})) \in \mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})+1}(\lambda(\mathsf{pred}', \mathbf{ac}').\, \emptyset)(\mathsf{pred}, \mathsf{aliasing}(\mathfrak{s})).$$

Proof. We prove the claim by strong mathematical induction on $\mathsf{height}(\mathfrak{t})$:

Let $\mathfrak{h} \triangleq \mathsf{heap}(\mathfrak{t})$, $r \triangleq \mathsf{root}(\mathfrak{t})$, and $\mathbf{ac} \triangleq \mathsf{aliasing}(\mathfrak{s})$. Since $\mathfrak{t}$ is an Φ-tree with $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathbf{v})$, there is a rule $(\mathsf{pred}(\mathbf{x}) \Leftarrow \phi) \in \Phi$ with $\phi = \exists \mathbf{e}.\, \phi'$, $\phi' = (y \mapsto \mathbf{z}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k}) \star \Pi$, Π pure[16], such that $\mathsf{rule}_{\mathfrak{t}}(r) = \mathsf{pred}(\mathbf{v}) \Leftarrow \phi'[\mathsf{fvars}(\mathsf{pred}) \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{m}]$ for some $\mathbf{m} \in \mathbf{Loc}^*$ (i.e., the root r of $\mathfrak{t}$ is labeled with an instance of the rule).

Let $\mathfrak{s}' = \mathfrak{s}[\mathsf{fvars}(\mathsf{pred})/\mathbf{v}][\mathbf{e}/\mathbf{m}]$. Moreover, for $1 \leq i \leq k$, let $\mathfrak{t}_i$ be the subtree of $\mathfrak{t}$ such that $\mathsf{rootpred}(\mathfrak{t}_i) = \mathsf{pred}_i(\mathbf{z_i})[\mathsf{fvars}(\mathsf{pred}) \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{m}]$; let $\mathfrak{h}_i = \mathsf{heap}(\mathfrak{t}_i)$. By Lemma 7.3, we have $\langle \mathfrak{s}', \mathfrak{h}_i \rangle \models_\Phi \mathsf{pred}_i(\mathbf{z_i})$ and, by Lemma 4.4, we have $\langle \mathfrak{s}', \mathfrak{h}_i \rangle \in \mathbf{GStates}$. Finally, we denote by $\mathfrak{h}_0$ the unique heap such that $\langle \mathfrak{s}', \mathfrak{h}_0 \rangle \models_\Phi y \mapsto \mathbf{z}$. Clearly, $\langle \mathfrak{s}', \mathfrak{h}_0 \rangle \in \mathbf{GStates}$ and $\mathfrak{h} = \mathfrak{h}_0 \uplus \cdots \uplus \mathfrak{h}_k$.

[16] In case of $\mathsf{height}(\mathfrak{t}) = 0$, the rule $(\mathsf{pred}(\mathbf{x}) \Leftarrow \phi)$ is non-recursive, and we have $k = 0$ and there are no existentially quantified variables, i.e., $\mathbf{e} = \epsilon$.

We use the following abbreviations:

$$\begin{aligned}\mathbf{ac}' &\triangleq \mathsf{aliasing}(\mathfrak{s}')\\ \mathcal{T}_0 &\triangleq \mathsf{type}_\Phi(\mathsf{ptrmodel}_{\mathbf{ac}'}(y \mapsto \mathbf{z}))\\ \mathcal{T}_i &\triangleq \mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h}_i), i \geq 1\end{aligned}$$

Since $\mathfrak{t}$ is $\mathfrak{s}(\mathbf{x})$-closed we have that $\mathsf{ptrlocs}(\mathfrak{t}_i) \cap (\mathbf{v} \cup \mathbf{m}) \subseteq \mathfrak{s}'(\mathbf{z_i}) \cup \mathfrak{s}(\mathbf{x}) = \mathfrak{s}'(\mathbf{z_i}) \cup \mathfrak{s}'(\mathbf{x})$. Furthermore, due to $\mathsf{img}(\mathfrak{s}') = \mathbf{v} \cup \mathbf{m}$, we have $\mathfrak{s}'(x) \notin \mathsf{ptrlocs}(\mathfrak{t}_i) \supseteq \mathsf{locs}(\mathfrak{h}_i)$ for all $x \in \mathsf{dom}(\mathbf{ac}')$ for which there is no $y \in \mathbf{z_i} \cup \mathbf{x}$ with $\mathfrak{s}'(x) = \mathfrak{s}'(y)$. Let $\mathfrak{s}_i$ be the restriction of $\mathfrak{s}'$ to $\mathbf{x} \cup \mathbf{z_i}$. Lemma 8.28 then yields

$$\mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h}_i) = \mathsf{extend}_{\mathbf{ac}'}(\mathsf{type}_\Phi(\mathfrak{s}_i, \mathfrak{h}_i)). \tag{†}$$

We introduce some more abbreviations:

$$\begin{aligned}\mathfrak{s}_i' &\triangleq \mathfrak{s} \cup \left\{\mathsf{fvars}(\mathsf{pred}_i) \mapsto \mathfrak{s}'(\mathbf{z_i})\right\},\\ \mathcal{T}_i' &\triangleq \mathsf{type}_\Phi(\mathfrak{s}_i', \mathfrak{h}_i)\\ \mathbf{ac}_i &\triangleq \mathsf{aliasing}(\mathfrak{s}_i')\end{aligned}$$

Observe that $\mathfrak{s}_i = \mathfrak{s}_i'[\mathsf{fvars}(\mathsf{pred}_i)/\mathbf{z_i}]$. By Lemma 8.22, $\mathcal{T}_i'[\mathsf{fvars}(\mathsf{pred}_i)/\mathbf{z_i}] = \mathsf{type}_\Phi(\mathfrak{s}_i, \mathfrak{h}_i)$. We then apply (†) to obtain

$$\mathcal{T}_i = \mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h}_i) = \mathsf{extend}_{\mathbf{ac}'}(\mathsf{type}_\Phi(\mathfrak{s}_i, \mathfrak{h}_i)) = \mathsf{extend}_{\mathbf{ac}'}(\mathcal{T}_i'[\mathsf{fvars}(\mathsf{pred}_i)/\mathbf{z_i}]). \tag{‡}$$

Finally, we note that $\mathfrak{t}_i$ is $\mathfrak{s}_i'(\mathbf{x})$-closed, $\mathsf{rootpred}(\mathfrak{t}_i) = \mathsf{pred}(\mathfrak{s}_i'(\mathsf{fvars}(\mathsf{pred}_i)))$ and $\mathsf{allholepreds}(\mathfrak{t}_i) = \emptyset$. Since $\mathsf{height}(\mathfrak{t}_i) < \mathsf{height}(\mathfrak{t})$, we can then apply the induction hypothesis and conclude that

$$\begin{aligned}\mathcal{T}_i' &= \mathsf{type}_\Phi(\mathfrak{s}_i', \mathfrak{h}_i)\\ &\in \mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t}_i)+1}(\lambda(\mathsf{pred}', \mathbf{ac}').\,\emptyset)(\mathsf{pred}_i, \mathsf{aliasing}(\mathfrak{s}_i'))\\ &\sqsubseteq \mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})}(\lambda(\mathsf{pred}', \mathbf{ac}').\,\emptyset)(\mathsf{pred}_i, \mathbf{ac}_i).\end{aligned} \tag{♣}$$

To finish the proof, we set $p \triangleq \mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})}(\lambda(\mathsf{pred}', \mathbf{ac}').\,\emptyset)$ and proceed as follows:

$$\begin{aligned}&\mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})+1}(\lambda(\mathsf{pred}', \mathbf{ac}').\,\emptyset)(\mathsf{pred}, \mathbf{ac})\\ &= \bigcup_{(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \Leftarrow \phi) \in \Phi} \mathsf{ptypes}_p^{\mathbf{x}}(\mathbf{ac}, \phi) && \text{(by definition)}\\ &\supseteq \mathsf{ptypes}_p^{\mathbf{x}}(\mathbf{ac}, \exists \mathbf{e}.\,\phi') && (\phi = \exists \mathbf{e}.\phi')\\ &\supseteq \mathsf{forget}_{\mathbf{ac}',\mathbf{e}}(\mathsf{ptypes}_p^{\mathbf{x}}(\phi', \mathbf{ac}')) && \text{(Lemma A.23)}\\ &= \mathsf{forget}_{\mathbf{ac}',\mathbf{e}}(\mathsf{ptypes}_p^{\mathbf{x}}(y \mapsto \mathbf{z}, \mathbf{ac}') \bullet \mathsf{ptypes}_p^{\mathbf{x}}(\mathsf{pred}_1(\mathbf{z_1}), \mathbf{ac}') && \text{(Def. of } \phi', \text{ Lemma A.21)}\\ &\qquad \bullet \cdots \bullet \mathsf{ptypes}_p^{\mathbf{x}}(\mathsf{pred}_1(\mathbf{z_k}), \mathbf{ac}'))\\ &= \mathsf{forget}_{\mathbf{ac}',\mathbf{e}}(\{\mathcal{T}_0\} \bullet \mathsf{extend}_{\mathbf{ac}'}(p(\mathsf{pred}_1, \mathbf{ac}_1)[\mathsf{fvars}(\mathsf{pred}_1) : \mathbf{z_1}/)] && \text{(Def. of } \mathcal{T}_0, \text{ Lemma A.24)}\\ &\qquad \bullet \cdots \bullet \mathsf{extend}_{\mathbf{ac}'}(p(\mathsf{pred}_k, \mathbf{ac}_k)[\mathsf{fvars}(\mathsf{pred}_k) : \mathbf{z_k}/)])\\ &\supseteq \mathsf{forget}_{\mathbf{ac}',\mathbf{e}}(\{\mathcal{T}_0\} \bullet \mathsf{extend}_{\mathbf{ac}'}(\{\mathcal{T}_1'\}[\mathsf{fvars}(\mathsf{pred}_1) : \mathbf{z_1}/)] && \text{(by (♣))}\\ &\qquad \bullet \cdots \bullet \mathsf{extend}_{\mathbf{ac}'}(\{\mathcal{T}_k'\}[\mathsf{fvars}(\mathsf{pred}_k) : \mathbf{z_k}/)])\\ &= \mathsf{forget}_{\mathbf{ac}',\mathbf{e}}(\{\mathcal{T}_0 \bullet \mathcal{T}_1 \cdots \bullet \mathcal{T}_k\}) && \text{(by (‡))}\\ &= \left\{\mathsf{forget}_{\mathbf{ac}',\mathbf{e}}(\mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h}))\right\} && \text{(by Corollary 8.19)}\end{aligned}$$

$$= \left\{\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})\right\}. \qquad \text{(by Lemma 8.24, as } \mathfrak{s}(\mathbf{e}) \subseteq \mathrm{dom}(\mathfrak{h})\text{)}$$

Read from bottom to top, we have $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \in \mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})+1}(\lambda(\mathsf{pred}',\mathbf{ac}').\,\emptyset)(\mathsf{pred},\mathbf{ac})$. □

Completeness then follows by exploiting the one-to-one correspondence between $\mathfrak{s}(\mathbf{x})$-closed Φ-trees without holes and the models of a predicate:

Lemma A.32 (Completeness of type computation). *Let* $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ *such that* $\mathsf{fvars}(\mathsf{pred}) = \mathbf{z} = \langle z_1,\dots,z_k\rangle \subseteq \mathbf{x}$. *Moreover, let* $\langle\mathfrak{s},\mathfrak{h}\rangle \models_\Phi \mathsf{pred}(\mathbf{z})$ *for some state* $\langle\mathfrak{s},\mathfrak{h}\rangle$ *with* $\mathbf{x} = \mathrm{dom}(\mathfrak{s})$. *Then,*

$$\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \in \mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred},\mathsf{aliasing}(\mathfrak{s})).$$

Proof. By Lemma 7.3, there exists a Φ-tree $\mathfrak{t}$ such that $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(\mathbf{z}))$, $\mathsf{allholes}(\mathfrak{t}) = \emptyset$, and $\mathsf{heap}(\{\mathfrak{t}\}) = \mathfrak{h}$. By Lemma A.30, the tree $\mathfrak{t}$ is $\mathfrak{s}(\mathbf{z})$-closed. Since $\mathbf{z} \subseteq \mathbf{x}$, $\mathfrak{t}$ is also $\mathfrak{s}(\mathbf{x})$-closed. By Lemma A.31, we know that

$$\mathsf{type}_\Phi(\mathfrak{s},\mathsf{heap}(\mathfrak{t})) \in \mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})+1}(\lambda(\mathsf{pred}',\mathbf{ac}').\,\emptyset)(\mathsf{pred},\mathsf{aliasing}(\mathfrak{s})).$$

Recalling that $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}}) = \lim_{n\in\mathbb{N}} \mathsf{unfold}_{\mathbf{x}}^n(\lambda(\mathsf{pred}',\mathbf{ac}').\,\emptyset)$ and $\mathfrak{h} = \mathsf{heap}(\mathfrak{t})$, we conclude that

$$\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \in \mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred},\mathsf{aliasing}(\mathfrak{s})).$$ □

A.29.3 Complexity of the Fixed-Point Computation. We now establish that the types of predicates can be computed in *doubly-exponential time*. As a first step, we consider a special case: the complexity of computing the types of single points-to assertions $a \mapsto \mathbf{b}$. Intuitively, single points-to assertions correspond to Φ-trees of size one. To compute their types, we systematically enumerate all such trees and check for each tree whether the points-to assertion in the tree node coincides with $\mathfrak{s}(a) \mapsto \mathfrak{s}(\mathbf{b})$.

Lemma A.33. *Let* $\mathbf{ac}$ *be an aliasing constraint, let* $a \in \mathrm{dom}(\mathbf{ac})$, *and let* $\mathbf{b} \in \mathbf{Var}^*$ *with* $\mathbf{b} \subseteq \mathrm{dom}(\mathbf{ac})$. *Let* $n \triangleq \max\{|\Phi|, |\mathrm{dom}(\mathbf{ac})|\}$. *Then,* $\mathsf{type}_\Phi(\mathsf{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b}))$ *is computable in* $2^{O(n\log(n))}$.

Proof. Let $\langle\mathfrak{s},\mathfrak{h}\rangle = \mathsf{ptrmodel}_{\mathbf{ac}}(a \mapsto \mathbf{b})$. W.l.o.g. we can assume that $\mathrm{img}(\mathfrak{s}) \subseteq \{1,\dots,n\}$ (otherwise we can select an isomorphic model with this property). We observe that a single location is allocated in $\mathfrak{h}$. Hence, in order to compute the type of $\langle\mathfrak{s},\mathfrak{h}\rangle$ we need to considers exactly those forests that consists of a single tree with a single rule instance whose points-to assertion agrees with $a \mapsto \mathbf{b}$. We collect those rule instances in the set $\mathbf{R}$:

$$\mathbf{R} \triangleq \{\mathsf{pred}(\mathbf{l}) \Leftarrow ((v \mapsto \mathbf{w}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k}) \star \Pi)[\mathsf{fvars}(\mathsf{pred})\cdot\mathbf{e}/\mathbf{l}\cdot\mathbf{m}] \in \mathbf{RuleInst}(\Phi) \mid \mathbf{l}\cdot\mathbf{m} \in \mathcal{L}^*, v[\mathsf{fvars}(\mathsf{pred})\cdot\mathbf{e}/\mathbf{l}\cdot\mathbf{m}] = \mathfrak{s}(a), \mathbf{w}[\mathsf{fvars}(\mathsf{pred})\cdot\mathbf{e}/\mathbf{l}\cdot\mathbf{m}] = \mathfrak{s}(\mathbf{b}))\}$$

We note that $|\mathbf{l}\cdot\mathbf{m}| \le n$ for all rule instances. Then, $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$ is given by the projections of the forests that consists of a single tree with a rule instance from $\mathbf{R}$:

$$\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) = \{\mathsf{project}(\mathfrak{s},\{\{a \mapsto \langle\emptyset,\mathcal{R}\rangle\}\}) \mid \mathcal{R} \in \mathbf{R}\} \cap \mathbf{DUSH}_\Phi.$$

For computing $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$, we only need those rules instances in $\mathbf{R}$ such that $\mathbf{l}\cdot\mathbf{m} \subseteq \{1,\dots,n\}$: We have $\mathrm{img}(\mathfrak{s}) \subseteq \{1,\dots,n\}$ and we can rename locations not in $\{1,\dots,n\}$ to obtain a $\mathfrak{s}$-equivalent forest with the desired property; such forests have the same projections due to Lemma 7.38. Thus, we can compute $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$ by considering $n \cdot n^n \in 2^{O(n\log(n))}$ rule instances. □

Theorem A.34 (Complexity of type computation). *Let* $n \triangleq |\Phi|+|\mathbf{x}|$. *Then, one can compute the set* $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})$ *assigning sets of types to predicates in* $2^{2^{O(n^2\log(n))}}$.

Proof. Theorem 8.7 gives us a bound on the the size of all types over aliasing constraints in $\mathbf{AC}^{\mathbf{x}}$: $\left|\mathbf{Types}^{\mathbf{x}}_{\Phi}\right| \in 2^{2^{O(n^2 \log(n))}}$. Moreover, the number of predicates of Φ is bounded by n. Consequently, the number of functions with signature $\mathbf{Preds}(\Phi) \times \mathbf{AC} \to 2^{\mathbf{Types}_\Phi}$ is bounded by

$$n \cdot 2^{2^{O(n^2 \log(n))}} = 2^{2^{O(n^2 \log(n))+1}} = 2^{2^{O(n^2 \log(n))}}.$$

Since every iteration of the fixed-point computation discovers at least one new type, the computation terminates after at most $2^{2^{O(n^2 \log(n))}}$ many iterations. We will show that each iteration takes at most $2^{2^{O(n^2 \log(n))}}$ steps. This is sufficient to establish the claim because

$$\underbrace{2^{2^{O(n^2 \log(n))}}}_{\text{number of iterations}} \cdot \underbrace{2^{2^{O(n^2 \log(n))}}}_{\text{cost per iteration}} = 2^{2^{O(n^2 \log(n))}}.$$

We now study the time spent in each iteration: Given some predicate $\mathrm{pred} \in \mathbf{Preds}(\Phi)$ and aliasing constraint $\mathbf{ac} \in \mathbf{AC}^{\mathbf{x}}$, we need to compute

- the function $\mathrm{ptypes}^{\mathbf{x}}_p(\phi, \mathbf{ac})$ for each rule $\mathrm{pred}(\mathrm{fvars}(\mathrm{pred})) \Leftarrow \phi \in \Phi$, where p is the pre-fixed point from the previous iteration, and
- the union of the results of these function calls (note that we need to compute at most one union operation per rule $\phi \in \Phi$).

We argue below that each call $\mathrm{ptypes}^{\mathbf{x}}_p(\phi, \mathbf{ac})$ can be done in at most $2^{2^{O(n^2 \log(n))}}$ many steps. We further observe that the union over a set of types is linear in the number of types, i.e., linear in $2^{2^{O(n^2 \log(n))}}$. Hence, each iteration takes at most $2^{2^{O(n^2 \log(n))}}$ many steps because

$$\underbrace{n}_{\text{number of rules}} \cdot \underbrace{O(2^{n \log(n)})}_{\text{number of aliasing constraints}} \cdot \underbrace{2^{2^{O(n^2 \log(n))}}}_{\text{cost for a fixed rule and aliasing constraint}} = 2^{2^{O(n^2 \log(n))}}.$$

To conclude the proof, we consider the cost of evaluating $\mathrm{ptypes}^{\mathbf{x}}_p(\phi, \mathbf{ac})$ for a fixed rule body ϕ and aliasing constraint $\mathbf{ac}$. Since the type for each right-hand side of a rule is computed at most once, we note that the recursive calls of ptypes lead to at most $|\phi| \leq n$ evaluations of base cases, i.e., (dis-)equalities and points-to assertions, and operations $\bullet$, $\cdot[\cdot : \cdot/\cdot]$, forget and extend. It remains to establish the cost of these operations:

(1) Evaluating a (dis-)equality takes constant time.
(2) The evaluation of a points-to assertions can be done in time $O(2^{n \log(n)})$ by Lemma A.33 (observing that $|\operatorname{dom}(\mathbf{ac})| \leq |\Phi| + |\mathbf{x}| = n$).
(3) The evaluation of the operations $\bullet$, $\cdot[\cdot : \cdot/\cdot]$, forget and extend each takes time polynomial in the size of the types, i.e., $2^{O(n^2 \log(n))}$ (see Lemma 8.6). For $\cdot[\cdot : \cdot/\cdot]$, and forget this is trivial. For the composition operation, $\bullet$, the polynomial bound follows because (1) the number of formulas that can be obtained by re-scoping is bounded by the number of types, and (2) the number of formulas that can be obtained by $\triangleright$ steps is also bounded by the number of types. Similarly, the number of formulas that can be obtained by extend is bounded by the number of types. As the number of types to which each function is applied is bounded by $2^{2^{O(n^2 \log(n))}}$ we obtain the following cost for each $\bullet$, $\cdot[\cdot : \cdot/\cdot]$, forget and extend:

$$\underbrace{\mathit{poly}(2^{O(n^2 \log(n))})}_{\text{cost of operation for a single type}} \cdot \underbrace{2^{2^{O(n^2 \log(n))}}}_{\text{number of types}} = 2^{2^{O(n^2 \log(n))}}.$$

Hence, the cost of evaluating $\mathsf{ptypes}^{\mathbf{x}}_p(\phi, \mathbf{ac})$ for a fixed rule body ϕ and aliasing constraint $\mathbf{ac}$ is

$$\underbrace{n}_{\text{size of the rule } \phi} \cdot \underbrace{2^{2^{\mathcal{O}(n^2 \log(n))}}}_{\text{cost of each of the } n \text{ operations}} = 2^{2^{\mathcal{O}(n^2 \log(n))}}. \qquad \square$$

A.30 Correctness of the Algorithm For Computing the Types of Guarded Formulas

The correctness of types is almost immediate from our previous results established for computing types of predicates; we only need two additional lemmata which we state below:

Lemma A.35. *Let $\phi_1, \phi_2 \in \mathbf{GSL}$ be two formulas and let $\mathbf{ac}$ be a stack-aliasing constraint. Then,* $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1 \wedge \phi_2) = \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1) \cap \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)$, $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1 \vee \phi_2) = \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1) \cup \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)$ *and* $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1 \wedge \neg\phi_2) = \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1) \setminus \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)$

Proof. We only show the first claim, the other two claims are shown analogously.

By definition of types, the inclusion $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1 \wedge \phi_2) \subseteq \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1) \cap \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)$ is straightforward. For the converse direction, we consider some $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1) \cap \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)$. Because of $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1)$ there is a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ with $\mathcal{T} = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ and $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_1$. By Corollary 4.6, we have $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$. Thus, Corollary 8.30 yields $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_2$. Hence, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_1 \wedge \phi_2$ and we obtain that $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1 \wedge \phi_2)$. $\square$

Lemma A.36. *Let $\phi_0, \phi_1, \phi_2 \in \mathbf{GSL}$ be three formulas and let $\mathbf{ac}$ be a stack-aliasing constraint. Then,*

$$\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_0 \wedge (\phi_1 \circledast \phi_2)) = \{\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_0) \mid \exists \mathcal{T}' \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1).\, \mathcal{T} \bullet \mathcal{T}' \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)\},$$

$$\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_0 \wedge (\phi_1 \mathrel{-\!\!\circledast} \phi_2)) = \{\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_0) \mid \forall \mathcal{T}' \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1).\, \mathcal{T} \bullet \mathcal{T}' \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)\}.$$

Proof. We only show the first claim, the second claim is shown analogously.

Let $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_0 \wedge (\phi_1 \circledast \phi_2))$. Then, there is a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ with $\mathcal{T} = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_0$, and $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_1 \circledast \phi_2$. By the semantics of $\circledast$, there exists a heap $\mathfrak{h}_1$ with $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \phi_1$ and $\langle \mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1 \rangle \models_\Phi \phi_2$. Let $\mathcal{T}_1 \triangleq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathcal{T}_2 \triangleq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_2)$. By Corollary 8.19, $\mathcal{T}_2 = \mathcal{T} \bullet \mathcal{T}_1$. Hence, $\mathcal{T} \in \{\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_0) \mid \exists \mathcal{T}' \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1).\, \mathcal{T} \bullet \mathcal{T}' \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)\}$.

Conversely, let $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_0)$ such that there is an $\mathcal{T}' \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1)$ with $\mathcal{T} \bullet \mathcal{T}' \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)$. Then, there is a state $\langle \mathfrak{s}, \mathfrak{h} \rangle$ with $\mathcal{T} = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ and $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_0$. Further, there is a state $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle$ with $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathcal{T}'$ and $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \models_\Phi \phi_1$. We can assume w.l.o.g. that $\mathfrak{h} \uplus \mathfrak{h}_1 \neq \bot$—otherwise, replace $\mathfrak{h}_1$ with an isomorphic heap that has this property. Corollary 8.19 yields $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1) = \mathcal{T} \bullet \mathcal{T}_1 \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_2)$. Since $\phi_0, \phi_1 \in \mathbf{GSL}$, we have $\langle \mathfrak{s}, \mathfrak{h} \rangle \in \mathbf{GStates}$ and $\langle \mathfrak{s}, \mathfrak{h}_1 \rangle \in \mathbf{GStates}$ by Corollary 4.6. Thus, also $\langle \mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1 \rangle \in \mathbf{GStates}$. Corollary 8.30 then gives us that $\langle \mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1 \rangle \models_\Phi \phi_2$. Therefore, $\langle \mathfrak{s}, \mathfrak{h} \rangle \models_\Phi \phi_1 \circledast \phi_2$, which implies that $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_1 \circledast \phi_2)$. Hence, $\mathcal{T} \in \mathbf{Types}^{\mathbf{ac}}_\Phi(\phi_0 \wedge (\phi_1 \circledast \phi_2))$. $\square$

We restate the claim of Theorem 9.2: Let $\phi \in \mathbf{GSL}$ with $\mathsf{fvars}(\phi) = \mathbf{x}$ and $\mathsf{locs}(\phi) = \emptyset$. Further, let $\mathbf{ac} \in \mathbf{AC}^{\mathbf{x}}$. Then, $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi) = \mathsf{types}(\phi, \mathbf{ac})$. Moreover, $\mathsf{types}(\phi, \mathbf{ac})$ can be computed in $2^{2^{\mathcal{O}(n^2 \log(n))}}$, where $n \triangleq |\Phi| + |\phi|$.

Proof. We first prove that $\mathbf{Types}^{\mathbf{ac}}_\Phi(\phi) = \mathsf{types}(\phi, \mathbf{ac})$. The proof proceeds by induction on ϕ:

Case $\phi = \mathbf{emp}$. By Lemma A.18.
Case $\phi = x \approx y$, $\phi = x \not\approx y$.] By Lemma A.19.
Case $\phi = a \mapsto \mathbf{b}$. By Lemma A.20.
Case $\phi = \mathsf{pred}(\mathbf{y})$. By Lemma A.22, Lemma A.26 and Lemma A.32.
Case $\phi = \phi_1 \star \phi_2$. By Lemma A.21 and the I.H..
Case $\phi = \phi_1 \wedge \phi_2$, $\phi = \phi_1 \vee \phi_2$, $\phi = \phi_1 \wedge \neg\phi_2$. By Lemma A.35 and the I.H.
Case $\phi = \phi_0 \wedge (\phi_1 \circledast \phi_2)$, $\phi = \phi_0 \wedge (\phi_1 \mathrel{-\!\!\circledast} \phi_2)$. By Lemma A.36 and the I.H.

We now turn to the complexity claim:

We recall that the number of types in $\mathbf{Types}^{\mathbf{ac}}_{\Phi}(\phi)$ is bounded by $2^{2^{O(n^2 \log(n))}}$ (see Theorem 8.7). The evaluation of $\mathrm{types}(\phi, \mathbf{ac})$ consists of at most $|\phi| \leq n$ invocations of the form $\mathrm{types}(\cdot, \mathbf{ac})$. We will show that each of these invocations can be evaluated in time at most $2^{2^{O(n^2 \log(n))}}$; this is sufficient to establish the claim because of $n \cdot 2^{2^{O(n^2 \log(n))}} = 2^{2^{O(n^2 \log(n))}}$:

- For **emp** and (dis-)equalities, the evaluation time is constant.
- For points-to assertions, this follows from Lemma A.33.
- For predicate calls, this follows from Theorem A.34.
- For $\wedge$, $\vee$, and $\neg$, the bound follows because each of these operations can be implemented in linear time in terms of the number of types.
- For $\star$, this follows because (1) $\bullet$ is applied to at most $2^{2^{O(n^2 \log(n))}} \cdot 2^{2^{O(n^2 \log(n))}} = 2^{2^{O(n^2 \log(n))}}$ many types and (2) the composition $\mathcal{T}_1 \bullet \mathcal{T}_2$ takes time at most $\mathit{poly}(2^{O(n^2 \log(n))})$, as argued in the proof of Theorem A.34. Hence, the cost of $\bullet$ is $\mathit{poly}(2^{O(n^2 \log(n))}) \cdot 2^{2^{O(n^2 \log(n))}} = 2^{2^{O(n^2 \log(n))}}$.
- For septraction and the magic wand, this is analogously to the cases for $\wedge$ resp. $\vee$ and $\star$. □

A.31 Correctness of the Reduction over Values with the Null-Pointer to Values without the Null-Pointer (Cor. 9.7)

We only show the claim about satisfiability. The claim about entailment follows from the first claim as in the proof of Cor. 9.4.

The proof of the first claim proceeds by a reduction to the satisfiability of a formula over the set of values $\mathbf{Val} \triangleq \mathbf{Loc} = \mathbb{N}_{>0}$: Let x be a fresh variable that does not appear in ϕ and Φ. Let ϕ' be the formula $x \mapsto x \mathrel{-\!\!\star} (\phi^\circ \star x \mapsto x)$, where ϕ° is obtained from ϕ by replacing every occurrence of *nil* by x. Further, let Φ' be the SID obtained from Φ by replacing every occurrence of *nil* by x and adding x as an additional parameter to every predicate. We now claim that ϕ is satisfiable over $\mathbf{Val} \triangleq \mathbf{Loc} \cup \{\mathit{nil}\}$ wrt SID Φ iff ϕ' is satisfiable over $\mathbf{Val} \triangleq \mathbf{Loc}$ wrt SID Φ': Let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state with $\langle \mathfrak{s}, \mathfrak{h} \rangle \models \phi$. Let $\mathfrak{s}' = \mathfrak{s}[x/\ell]$ be the stack $\mathfrak{s}$ extended by the mapping of x to some fresh location $\ell \in \mathbf{Loc} \setminus (\mathrm{img}(\mathfrak{s}) \cup \mathrm{locs}(\mathfrak{h}))$ (in particular, ℓ does not appear as a constant in ϕ) and let $\mathfrak{h}'$ be the heap obtained from $\mathfrak{h}$ by mapping all locations that map to *nil* to ℓ. Then it is easy to verify that $\langle \mathfrak{s}', \mathfrak{h}' \rangle \models \phi^\circ$. Moreover we have that $\langle \mathfrak{s}', \mathfrak{h}' \rangle \models x \mapsto x \mathrel{-\!\!\star} (\phi^\circ \star x \mapsto x)$ because of $\ell \notin \mathrm{dom}(\mathfrak{h})$ (since ℓ has been chosen as a fresh location). For the other direction, let $\langle \mathfrak{s}, \mathfrak{h} \rangle$ be a state with $\langle \mathfrak{s}, \mathfrak{h} \rangle \models \phi'$. We observe that $\langle \mathfrak{s}, \mathfrak{h} \rangle \models \phi^\circ$ and $\mathfrak{s}(x) \notin \mathrm{dom}(\mathfrak{h})$ because of $\langle \mathfrak{s}, \mathfrak{h} \rangle \models x \mapsto x \mathrel{-\!\!\star} (\phi^\circ \star x \mapsto x)$. Let $\mathfrak{s}'$ be the stack obtained from $\mathfrak{s}$ by removing the mapping for x and let $\mathfrak{h}'$ be the heap obtained from $\mathfrak{h}$ by mapping all locations that map to $\mathfrak{s}(x)$ to *nil*. Then it is easy to verify that $\langle \mathfrak{s}', \mathfrak{h}' \rangle \models \phi$.

The claim then follows from Theorem 9.3 and the observation that $|\Phi'| + |\phi'| = O(|\Phi| + |\phi|)$.