# The Entailment Problem for Separation Logic with Overlaid Structures

**Lucas Bueri**
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France

**Nicolas Peltier**
Université Grenoble Alpes, CNRS, LIG, France

**Quentin Petitjean**
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France

**Mihaela Sighireanu**
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France

## Abstract

Separation Logic (SL) enables reasoning about programs that manipulate pointers. Its key feature is the separating conjunction $\star$, which asserts that two formulas hold on disjoint portions of memory. We consider an extension of SL, called Overlaid SL (OSL), that allows non-disjoint combinations of data structures defined over different fields, enriched with set constraints on the nodes of these structures. We prove that entailment is decidable for a broad class of data structures satisfying the so-called PCE conditions of [9], thus extending this result to OSL. Our decision procedure is nondeterministic with doubly exponential time complexity.

**2012 ACM Subject Classification** Theory of computation → Separation logic

**Keywords and phrases** Decision Procedure, Separation Logic, Inductive Definitions, Overlaid Data Structures

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2026.94

**Related Version** *Full Version*: `https://hal.science/view/index/docid/5671037`

**Funding** This work has been partially funded by the French National Research Agency project `ANR-21-CE48-0011`.

## 1 Introduction

Separation Logic (SL) [11, 16] is a widely used framework in program verification. It supports *local reasoning*, which improves scalability by enabling the verification of program properties using only the portions of the heap that the program interacts with. In essence, SL provides basic atoms such as $x \to (y_1, \ldots, y_d)$, expressing that the heap consists solely of the allocated location $x$, at which is stored the tuple $(y_1, \ldots, y_d)$, and `emp`, indicating the absence of any allocated location (empty heap). It also features the connective $\varphi_1 \star \varphi_2$ (called a *separating conjunction*), which asserts that the heap can be split into two disjoint parts satisfying $\varphi_1$ and $\varphi_2$, respectively. To describe recursive data structures, SL additionally allows predicate atoms defined by user-specified inductive rules with fixpoint semantics. A commonly studied fragment, due to its widespread use [5] in automated verifiers based on SL, is the *symbolic heap* fragment, where formulas are separating conjunctions of atoms. The central challenge in automated program verification using SL is the *entailment problem*[1]: given two formulas $\varphi$ and $\psi$, does $\varphi$ logically entail $\psi$ – that is, does every structure satisfying $\varphi$ also satisfy $\psi$? Even

[1] The satisfiability problem is strictly less general (since full negation is not permitted in the symbolic heap fragment) and corresponds to the special case where $\psi$ is false.


when restricted to symbolic heaps, the entailment problem remains undecidable; however, several decidable sub-fragments have been identified [1, 6, 9, 10]. The most expressive known decidable subclass is the PCE fragment (for **P**rogress, **C**onnected, and **E**stablished), which imposes structural constraints on the inductive rules defining predicate semantics [9]. This fragment captures a wide range of commonly used data structures, including lists, trees, and doubly linked lists. Later, the entailment problem was shown to be 2-ExpTime-complete for PCE formulas [14, 3].

In standard SL, the separating conjunction $\star$ models the disjoint union of heaps: two heaps can be combined only if they share no allocated locations. Although this operator provides an elegant and concise way to express disjointness, it is often too restrictive in practice. Many data structure specifications require more flexible forms of heap decomposition, in which certain locations may be shared across substructures [13, 8, 17]. For example, consider a program that maintains three lists of tasks (ongoing, waiting, and overall), with the additional constraint that the third list contains exactly the union of the elements of the first two (in arbitrary order). If list nodes are allocated locations, such a structure cannot be specified in standard SL. A proposed solution is the framework of *overlaid data structures* [13, 7, 2], in which each location may be allocated with multiple "fields". This leads to formulas such as $x.f \to (y_1, \ldots, y_d) \circledast x.g \to (z_1, \ldots, z_d)$ which asserts that the fields $f$ and $g$ of $x$ are both allocated and store the tuples $(y_1, \ldots, y_d)$ and $(z_1, \ldots, z_d)$, respectively. The connective $\star$ is replaced by a slightly weaker connective $\circledast$ (o-star), which permits the two heap components to share allocated locations (in this case, $x$), provided the fields they use are distinct. To express additional constraints on such structures, the framework also incorporates set constraints over variables denoting finite sets of locations. A formula of the form $\varphi$@X asserts that $\varphi$ holds and that the set variable X corresponds exactly to the domain of the heap. The example of overlaid lists of tasks defined above can then be described with the formula $(\mathtt{ls}_f(x_1, \mathrm{nil})@\mathrm{X}_1 \star \mathtt{ls}_f(x_2, \mathrm{nil})@\mathrm{X}_2) \circledast \mathtt{ls}_g(y, \mathrm{nil})@\mathrm{Y} \star \mathrm{X}_1 \cup \mathrm{X}_2 \approx \mathrm{Y}$, where $\mathtt{ls}_f(u, v)$ denotes a list segment from $u$ to $v$ whose pointers are stored in field $f$. Note that $\star$ may still be used when the heap components are disjoint. Distinct fields (e.g., $f$ for the lists rooted at $x_i$, and $g$ for the list rooted at $y$) allow the logic to maintain separation at the field level. Note that, in our framework, set constraints such as $\mathrm{X}_1 \cup \mathrm{X}_2 \approx \mathrm{Y}$ hold only on empty heaps. In [7], a fragment of the logic is introduced, for which satisfiability and entailment are proven Np-complete and co-Np-complete, respectively, and which can describe nested linked lists with shared nodes. However, the considered inductive rules are more restricted than those in the PCE fragment [9]. In a recent work [15], we introduced a decision procedure for the satisfiability problem which covers PCE rules.

**Contribution.** In the present paper, we show that the entailment problem is also decidable[2]. We reduce this problem to a satisfiability problem in the logic BAPA [12], which combines the Boolean algebra of sets of uninterpreted elements with Presburger arithmetic to capture cardinality constraints on sets. The decision procedure runs in nondeterministic doubly exponential time. In fact, we go beyond the symbolic heap fragment by considering entailments of the form $\varphi \models \psi$, where $\varphi$ contains no negation while $\psi$ is arbitrary (both satisfiability and entailment are undecidable for unrestricted formulas, even in standard SL [14]). Our results increase the expressive power of SL, allowing one to describe a substantially more general class of data structures. As in [7], our procedure relies on a reduction to BAPA. The

[2] The fragment considered in [15] is actually slightly more general than the one used here, since it allows inductive rules to compute sets of locations that need not coincide with the heap domain. For example, one may define in [15] a predicate $\mathtt{tree}(x, \mathrm{Y})$ representing a tree with root $x$ and set of leaves Y; in this paper, Y shall denote the full set of the tree's nodes. Although the fragment here is simpler, it is yet sufficient to capture the most common data structures.

key idea behind this reduction, however, differs significantly: instead of checking syntactic isomorphisms between formulas, we exploit a sound and complete abstract characterisation of their models. This allows us to handle a strictly more general fragment, both with respect to the admissible inductive rules and the formulas appearing in entailments, while retaining completeness of the decision procedure.

The remainder of the paper is organised as follows. We first formally introduce Overlaid Separation Logic (OSL) in Section 2. Then, we present in Section 3 the first step of our decision procedure, where the entailment problem is reduced to the search for a counter-example for simpler, normalised formulas. Some results related to the entailment problem in standard SL are recalled in Section 4, while the reduction to satisfiability in BAPA is presented in Section 5. We conclude by pointing out limitations and possible improvements.

## 2 Separation Logic with Overlaid Structures

For conciseness, $\vec{x}$ denotes a tuple $(x_1, \ldots, x_n)$ of size $n = |\vec{x}|$, and the $i$-th element ($i \geqslant 1$) of the tuple $\vec{x}$ is written $\vec{x}[i]$. We sometimes identify sequences with sets; for instance, we may write $x \in \vec{y}$ to state that $x = \vec{y}[i]$ for some $i \in [\![1, |\vec{y}|]\!]$. For any finite set S, card(S) denotes the cardinality of S.

▶ **Definition 1** (OSL formulas). *Let $\mathcal{F}$ be a finite set of* field *symbols. Let $\mathcal{V}$ be a (countably infinite) set of* location variables*, $\mathcal{S}$ a (countably infinite) set of* set variables*, and $\mathcal{P}$ a set of* spatial predicate symbols *(with associated arity $\# : \mathcal{P} \longrightarrow \mathbb{N}$). The set of* OSL formulas *$\Phi$ is inductively defined as follows:*

$$\Phi \ni \varphi := \mathtt{emp} \mid x.f \to (y_1, \ldots, y_d) \mid p(x_1, \ldots, x_{\#(p)}) \\ \mid \varphi_1 \star \varphi_2 \mid \varphi_1 \circledast \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi@\mathrm{T} \mid \pi$$

*with:*

$$\begin{aligned} \textit{pure formulas } \Pi \ni \pi &:= \mathrm{L} \mid \mathrm{B} \mid \mathrm{A} \mid \exists \mathrm{X}.\pi \mid \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid \sim\pi \\ \textit{equational atoms } \mathfrak{L} \ni \mathrm{L} &:= x \approx y \\ \textit{set atoms } \mathfrak{B} \ni \mathrm{B} &:= \mathrm{T}_1 \sqsubseteq \mathrm{T}_2 \\ \textit{set terms } \mathfrak{T} \ni \mathrm{T} &:= \{x\} \mid \mathrm{X} \mid \varnothing \mid \mathrm{T}_1 \sqcup \mathrm{T}_2 \mid \mathrm{T}_1 \sqcap \mathrm{T}_2 \\ \textit{arithmetic atoms } \mathfrak{A} \ni \mathrm{A} &:= t_1 \prec t_2 \mid \mathrm{K} \operatorname{div} t \\ \textit{arithmetic terms } \mathfrak{t} \ni t &:= \mathrm{K} \mid t_1 \oplus t_2 \mid \mathrm{K} \odot t \mid |\mathrm{T}| \end{aligned}$$

*where $d \in \mathbb{N}$, $\mathrm{K} \in \mathbb{Z}$, $f \in \mathcal{F}$, $x, y, x_1, \ldots, x_{\#(p)}, y_1, \ldots, y_d \in \mathcal{V}$, $\mathrm{X} \in \mathcal{S}$, and $p \in \mathcal{P}$.*

Atoms of the form $x.f \to (y_1, \ldots, y_d)$ and $p(x_1, \ldots, x_{\#(p)})$ are called *points-to atoms* and *predicate atoms*, respectively. We write $x \not\approx y$, $t_1 \preceq t_2$, and $\mathrm{T}_1 \not\sqsubseteq \mathrm{T}_2$ for $\sim(x \approx y)$, $\sim(t_2 \prec t_1)$, and $\sim(\mathrm{T}_1 \sqsubseteq \mathrm{T}_2)$, respectively, $\mathrm{T}_1 \approx \mathrm{T}_2$ for $(\mathrm{T}_1 \sqsubseteq \mathrm{T}_2) \wedge (\mathrm{T}_2 \sqsubseteq \mathrm{T}_1)$ if $\mathrm{T}_1, \mathrm{T}_2 \in \mathfrak{T}$, and $t_1 \approx t_2$ for $(t_1 \preceq t_2) \wedge (t_2 \preceq t_1)$ if $t_1, t_2 \in \mathfrak{t}$. To simplify notation, we do not include arithmetic variables in the syntax, but they can be encoded using terms $|\mathrm{X}|$ where X is a fresh set variable. There is no quantification over location variables, and quantification over sets occurs only in pure formulas. We write $\mathrm{fv}^{\mathrm{loc}}(\varphi)$ and $\mathrm{fv}^{\mathrm{set}}(\varphi)$ for the free location variables and set variables in $\varphi$, with $\mathrm{fv}(\varphi) \stackrel{\mathrm{def}}{=} \mathrm{fv}^{\mathrm{loc}}(\varphi) \uplus \mathrm{fv}^{\mathrm{set}}(\varphi)$. Note that we use two different negation symbols, $\sim$ for pure formulas and $\neg$ for OSL formulas. This distinction will be clarified later.

▶ **Definition 2** (Substitutions). *Let $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_n)$ be sequences of location variables. We write $[x_j/y_j]_{j \in [\![1,n]\!]}$, $[\vec{x}/\vec{y}]$, or $[x_1/y_1, \ldots, x_n/y_n]$ for the substitution $\theta$ that* simultaneously *replaces $x_j$ by $y_j$ for every $j \in [\![1, n]\!]$. The formula $\varphi\theta$ is obtained by substitution of every free occurrence of $x_j$ by $y_j$ in $\varphi$.*

The semantics of predicate symbols is defined using *inductive rules*. An *inductive rule associated with the predicate* $p \in \mathcal{P}$ is a pair of the form $p(x_1, \ldots, x_{\#(p)}) \Leftarrow \varphi$, where $\varphi$ is an OSL-formula. The variables freely occurring in the right-hand side of a rule but not in its left-hand side are called *existential variables*. A *set of inductive definitions* (SID) $\mathcal{R}$ contains finitely many rules associated with predicate symbols in $\mathcal{P}$.

Let $\mathcal{L}$ be a countably infinite set of *locations*. A *store* is a mapping $\mathfrak{s} : \mathcal{V} \to \mathcal{L}$ which associates each variable to a location. If $\mathrm{S} \subseteq \mathcal{V}$, we write $\mathfrak{s}(\mathrm{S})$ for $\{\mathfrak{s}(x) \mid x \in \mathrm{S}\}$. We write $\mathfrak{s}\,[x \mapsto \ell]$ for the store that maps $x$ to $\ell$ and agrees with $\mathfrak{s}$ on all variables other than $x$. Also, $\mathfrak{s}\,[x/y]$ denotes the updated store that maps $y$ to $\mathfrak{s}(x)$ and agrees with $\mathfrak{s}$ on everything else. A *set interpretation* is a mapping $\Sigma : \mathcal{S} \to \mathscr{P}(\mathcal{L})$ from set variables to finite subsets of locations. We denote by $[\![t]\!]_{(\mathfrak{s},\Sigma)}$ and $[\![\mathrm{T}]\!]_{(\mathfrak{s},\Sigma)}$ the interpretations of the arithmetic term $t$ and the set term T, respectively, in a pair $(\mathfrak{s}, \Sigma)$, defined inductively as usual. For any pure formula $\pi$, we write $(\mathfrak{s}, \Sigma) \models \pi$ to indicate that $\pi$ is true in $(\mathfrak{s}, \Sigma)$, where all the symbols and connectives are interpreted as usual.

A *heap* is a partial finite function mapping pairs in $\mathcal{L} \times \mathcal{F}$ to tuples of locations in $\mathcal{L}^*$ (thus we have $\mathrm{dom}(\mathfrak{h}) \subseteq \mathcal{L} \times \mathcal{F}$ for all heaps $\mathfrak{h}$). A heap is usually denoted as a set $\{(\ell_i, f_i) \mapsto (\vec{\ell_i}) \mid 1 \leqslant i \leqslant n\}$. We denote by $\mathrm{dom}_f(\mathfrak{h})$ the set $\{\ell \mid (\ell, f) \in \mathrm{dom}(\mathfrak{h})\}$, by $\mathrm{alloc}(\mathfrak{h})$ the set $\{\ell \mid \exists f \in \mathcal{F}.\, (\ell, f) \in \mathrm{dom}(\mathfrak{h})\}$ and by $\mathrm{loc}(\mathfrak{h})$ the set of locations occurring in $\mathfrak{h}$, i.e., the set: $\mathrm{loc}(\mathfrak{h}) = \{\ell_0, \ldots, \ell_n \mid \exists f \in \mathcal{F}.\, \mathfrak{h}(\ell_0, f) = (\ell_1, \ldots, \ell_n)\}$. For any pair $(\mathfrak{s}, \mathfrak{h})$, we write $\mathrm{loc}(\mathfrak{s}, \mathfrak{h})$ for the set $\mathrm{loc}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})$. A location $\ell$ is *allocated* in a heap $\mathfrak{h}$ if $\ell \in \mathrm{alloc}(\mathfrak{h})$. If $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \varnothing$, then $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ denotes the union of $\mathfrak{h}_1$ and $\mathfrak{h}_2$; otherwise $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ is not defined. An *OSL structure* is a tuple $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ where $\mathfrak{s}$ is a *store*, $\Sigma$ is a *set interpretation*, and $\mathfrak{h}$ is a heap. A variable $x$ is *allocated* in a structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ if $\mathfrak{s}(x)$ is allocated in $\mathfrak{h}$.

▶ **Definition 3** (Unfoldings). *Let $\mathcal{R}$ be an SID, and $\varphi, \psi$ two formulas. We write $\varphi \Leftarrow_{\mathcal{R}} \psi$ when $\psi$ is obtained by replacing a predicate atom $p(y_1, \ldots, y_{\#(p)})$ in $\varphi$ by a formula $\chi\,[x_j/y_j]_{j \in [\![1, \#(p)]\!]}$ where $p(x_1, \ldots, x_{\#(p)}) \Leftarrow \chi$ is a rule in $\mathcal{R}$. We assume that the variables occurring in the rules are renamed to avoid conflict with variables already occurring in $\varphi$. An* $\mathcal{R}$-unfolding *is a sequence of formulas $\varphi_1 \Leftarrow_{\mathcal{R}} \varphi_2 \Leftarrow_{\mathcal{R}} \ldots \Leftarrow_{\mathcal{R}} \varphi_k$. The $\mathcal{R}$-unfolding is* complete *when the last formula $\varphi_k$ is predicate-free.*

▶ **Definition 4** (OSL semantics). *Given a formula $\varphi$, an SID $\mathcal{R}$ and a structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ with $\mathrm{fv}(\varphi) \subseteq \mathrm{dom}(\mathfrak{s}) \cup \mathrm{dom}(\Sigma)$, we write $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi$ if and only if one of the following conditions holds:*

- *$\varphi = \mathtt{emp}$ and $\mathfrak{h} = \varnothing$; or $\varphi$ is a pure formula $\pi$, $\mathfrak{h} = \varnothing$ and $(\mathfrak{s}, \Sigma) \models \pi$;*
- *$\varphi = (x.f \to (y_1, \ldots, y_d))$ and $\mathfrak{h} = [(\mathfrak{s}(x), f) \mapsto (\mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_d))]$;*
- *$\varphi = p(z_1, \ldots, z_{\#(p)})$, $p \in \mathcal{P}$, and there exists a complete unfolding $\varphi \Leftarrow_{\mathcal{R}} \ldots \Leftarrow_{\mathcal{R}} \psi$ such that $(\mathfrak{s}', \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \psi$ for some $\mathfrak{s}'$ matching $\mathfrak{s}$ on $z_1, \ldots, z_{\#(p)}$;*
- *$\varphi = \psi@\mathrm{T}$, $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \psi$ and $\mathrm{alloc}(\mathfrak{h}) = [\![\mathrm{T}]\!]_{(\mathfrak{s},\Sigma)}$;*
- *$\varphi = \varphi_1 \star \varphi_2$ and there exist heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \cup \mathfrak{h}_2$, $\mathrm{alloc}(\mathfrak{h}_1) \cap \mathrm{alloc}(\mathfrak{h}_2) = \varnothing$, $(\mathfrak{s}, \mathfrak{h}_1, \Sigma) \models_{\mathcal{R}} \varphi_1$, and $(\mathfrak{s}, \mathfrak{h}_2, \Sigma) \models_{\mathcal{R}} \varphi_2$;*
- *$\varphi = \varphi_1 \circledast \varphi_2$ and there exist heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \cup \mathfrak{h}_2$, $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \varnothing$, $(\mathfrak{s}, \mathfrak{h}_1, \Sigma) \models_{\mathcal{R}} \varphi_1$, and $(\mathfrak{s}, \mathfrak{h}_2, \Sigma) \models_{\mathcal{R}} \varphi_2$;*
- *$\varphi = \varphi_1 \wedge \varphi_2$, $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi_1$, and $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi_2$;*
- *$\varphi = \varphi_1 \vee \varphi_2$ and $\big((\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi_1$ or $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi_2\big)$;*
- *$\varphi = \neg\psi$ and $(\mathfrak{s}, \mathfrak{h}, \Sigma) \not\models_{\mathcal{R}} \psi$.*

*We write $\varphi \models_{\mathcal{R}} \psi$ if for every structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ we have $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi \implies (\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \psi$. If both $\varphi \models_{\mathcal{R}} \psi$ and $\psi \models_{\mathcal{R}} \varphi$ hold, then we write $\varphi \equiv_{\mathcal{R}} \psi$.*

Formulas are taken modulo the usual properties of SL connectives: associativity and commutativity of $\star, \circledast$, $\wedge$ and $\vee$, neutrality of `emp` for $\star$ and $\circledast$, and commutativity of $\approx$. The pure formulas serve as an extension of `emp`, providing additional information on the store and set interpretation. Thus, a valid pure formula (such as $x \approx x$) behaves like `emp`. The operators $\wedge$ and $\vee$ appear twice in the syntax and the semantics when dealing with spatial or pure formulas. This is not a problem, as the definitions are compatible. On the contrary, the negation symbols $\neg$ and $\sim$ have different meanings, as the latter requires the heap to be empty: if $\pi$ is pure, then $\pi$ and $\sim\pi$ are both false in $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ if the heap $\mathfrak{h}$ is not empty. In contrast, $\neg$ denotes the usual negation. Notice that occurrences of $\neg$ are restricted here to the formulas in the right-hand side of the entailment to ensure decidability (see Section 3). Since $\mathrm{alloc}(\mathfrak{h}_1) \cap \mathrm{alloc}(\mathfrak{h}_2) = \varnothing \implies \mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \varnothing$, $\varphi_1 \circledast \varphi_2$ is a logical consequence of $\varphi_1 \star \varphi_2$. The interpretation of $\varphi_1 \circledast \varphi_2$ is weaker, as it allows allocations at the same locations in both $\varphi_1$ and $\varphi_2$, provided the considered fields are distinct. For instance, $x.f \to (y) \circledast x.g \to (y)$ is satisfiable, but not $x.f \to (y) \star x.g \to (y)$ or $x.f \to (y) \circledast x.f \to (y)$.

▶ **Definition 5** (OSL model). *An $\mathcal{R}$-model of a formula $\varphi$ is a structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ such that $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi$. A formula $\psi$ admitting an $\mathcal{R}$-model is* $\mathcal{R}$-satisfiable *(or simply* satisfiable *if $\mathcal{R}$ is clear from the context).*

We need additional restrictions on the rules in $\mathcal{R}$ in order to obtain decidable decision problems. To this purpose, we adapt the definition given in [9]. A formula $\varphi$ is an $\{f\}$-*formula* if, for every unfolding sequence $\varphi \equiv \varphi_1 \Leftarrow_{\mathcal{R}} \cdots \Leftarrow_{\mathcal{R}} \varphi_k$, the formula $\varphi_k$ contains no field symbol other than $f$; every pure formula is a $\{f\}$-formula, for any $f$. A predicate $p$ is an $\{f\}$-*predicate* if the right-hand side of each rule associated with $p$ is an $\{f\}$-formula. A predicate is a 1-predicate if it is an $\{f\}$-*predicate* for some $f \in \mathcal{F}$. These properties are decidable by a straightforward fixpoint computation over $\mathcal{R}$ and OSL formulas.

▶ **Definition 6** (PCE conditions). *An SID $\mathcal{R}$* satisfies the PCE conditions *if each rule $\rho \in \mathcal{R}$ is of the form $\rho:\ p(\vec{x}) \Leftarrow \pi \star \vec{x}[1].f \to (\vec{y}) \star \bigstar_{i=1}^{\ell} q_i(\vec{z_i})$ where:*

1. *the formula $\pi$ is a $\star$-conjunction of pure equational literals $u \approx v$ or $u \not\approx v$;*
2. *every predicate atom $q_i(\vec{z_i})$ has $\vec{z_i}[1] \in \vec{y}$, i.e., its first argument is a variable used by the tuple $\vec{y}$;*
3. *every existential variable $u \in \left(\vec{y} \cup \bigcup_{i=1}^{\ell} \vec{z_i}\right) \setminus \vec{x}$ is allocated by every structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ satisfying $\pi \star \vec{x}[1].f \to (\vec{y}) \star \bigstar_{i=1}^{\ell} q_i(\vec{z_i})$.*

The restrictions **2** and **3** above are equivalent, respectively, to the connectedness and establishment conditions of the PCE fragment[3] [9]. The syntax of rules also satisfies the progress condition of [9], as, by the definition above, they allocate exactly one location $\vec{x}[1]$. To these classic PCE restrictions, we add restriction **1**, which precludes set or arithmetic atoms in rules. In the following, we also assume that all predicates are 1-predicates.

A variable $x$ is called an $\{f\}$-*root* of a formula $\varphi$ if $\varphi$ contains an atom of the form $x.f \to (\vec{y})$ or $p(x, \vec{y})@\mathrm{T}$, where $p$ is an $\{f\}$-predicate. We denote by $\mathrm{roots}(\varphi, f)$ the set of $\{f\}$-roots of $\varphi$. By definition, for every PCE SID $\mathcal{R}$, the root of an atom $\varphi$ is allocated in every $\mathcal{R}$-model of $\varphi$ (since, by Definition 6, all the rules associated with $p$ allocate the first argument of $p$). Furthermore, for every model $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ of $\varphi$, we have $\mathrm{loc}(\mathfrak{h}) \subseteq \mathrm{alloc}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})$ (by the establishment condition 3, any location occurring in the heap is either allocated or associated with a predicate parameter, i.e., variable in $\varphi$).

[3] The establishment condition is sometimes relaxed by allowing non-allocated existential variables, provided that they are aliased to a variable in $\vec{x}$. For simplicity, we adopt the strongest definition, which entails no loss of expressivity.

▶ **Example 7.** Consider the following inductive rules.

$$\texttt{tree}(x) \Leftarrow x.t \to (y_1, y_2) \star \texttt{tree}(y_1) \star \texttt{tree}(y_2) \qquad \texttt{tree}(x) \Leftarrow x.t \to ()$$
$$\texttt{als}(x, y) \Leftarrow x.l \to (z) \star \texttt{als}(z, y) \star x \not\approx y \qquad \texttt{als}(x, y) \Leftarrow x.l \to (y) \star x \not\approx y$$

The predicate $\texttt{tree}(x)$ describes a full binary tree rooted at $x$, whose internal nodes use the field $t$ to point to their children; leaves point to an empty tuple of children. The predicate $\texttt{als}(x, y)$ describes a nonempty acyclic list segment from $x$ to $y$, using the field $l$. The OSL formula $\texttt{tree}(x)@\mathrm{X} \circledast \texttt{als}(y, z)@\mathrm{Y} \circledast \mathrm{Y} \sqsubseteq \mathrm{X}$ describes a heap that can be decomposed into two parts: a tree rooted at $x$, whose set of allocated nodes is recorded in the set variable X, and a list segment from $y$ to $z$, whose allocated nodes form the set Y. The constraint $\mathrm{Y} \sqsubseteq \mathrm{X}$ enforces that every list's cell also belongs to the tree. The heaps have disjoint domains, but share allocated locations. Intuitively, the formula specifies a data structure in which the list provides an alternative traversal of a part of the tree: all list locations appear in the tree, in arbitrary order, but the structural connections of the tree and the list remain independent. This independence is ensured by using distinct fields: $t$ for the tree and $l$ for the list.

## 3 Normalising the OSL Entailment Problem

In the remainder of the paper, we prove that the entailment problem is decidable if the left-hand side formula contains no occurrence of the negation operator $\neg$. The general problem, i.e., with no restrictions on the left-hand side, is undecidable for OSL because it covers the entailment problem between SL formulas with negation, a problem shown undecidable in [14], even if the inductive rules satisfy the conditions of [9].

▶ **Theorem 8.** *Let $\mathcal{R}$ be an SID with PCE 1-predicate rules, and $\varphi$, $\psi$ be two OSL formulas, where $\varphi$ contains no occurrence of $\neg$. Then, the entailment problem $\varphi \models_{\mathcal{R}} \psi$ is decidable.*

We provide a nondeterministic algorithm intended to generate a representation of a countermodel of $\varphi \models_{\mathcal{R}} \psi$, i.e., a model of $\varphi$ that does not satisfy $\psi$. In this section, we reduce the problem to a simpler instance of entailment, with additional hypotheses on the left-hand side formulas, as required by Lemma 25. We assume for simplicity that $\varphi$ contains at least one spatial atom; this can be enforced, if needed, by adding dummy predicates on both sides of the entailment.

### 3.1 Removing $\vee$, $\wedge$, and $\star$ from the Left-hand Side

▶ **Definition 9** ($\circledast$-formula). *An* $\circledast$-formula *is a formula $\varphi$ of the form $\pi \circledast \circledast_{i=1}^{n} \varphi_i$, where $\pi$ is pure, $n \geqslant 1$, and $\varphi_i$ (for $i \in [\![1, n]\!]$) is either a points-to atom $x_i.f_i \to (\vec{y_i})$ or an atom of the form $p_i(\vec{x_i})@\mathrm{T}_i$. The sequence of* set terms associated with $\varphi$ *is $(\mathrm{T}_i)_{i \in [\![1,n]\!]}$, where $\mathrm{T}_i$ is defined as the term $\{x_i\}$ when $\varphi_i$ is a points-to atom $x_i.f_i \to (\vec{y_i})$.*

The first step consists of ensuring that the left-hand side of the entailment is an $\circledast$-formula. To this purpose, we start by removing the connectives $\vee$, $\wedge$, and $\star$ from the left-hand side of the entailment. First, we remove disjunctions $\vee$ from $\varphi$ (except between pure formulas), by guessing, for every disjunction $\varphi_1 \vee \varphi_2$ in the formula $\varphi$, which one of $\varphi_1$ or $\varphi_2$ we keep to obtain the countermodel. Formally, for a given formula $\varphi$ with no negation $\neg$, we define the set of formulas $\mathrm{ND}(\varphi)$ inductively by:

- if $\varphi$ is an atom or a pure formula, then $\mathrm{ND}(\varphi) \stackrel{\text{def}}{=} \{\varphi\}$; otherwise:
- if $\varphi = \varphi_1@\mathrm{T}$, then $\mathrm{ND}(\varphi) \stackrel{\text{def}}{=} \{\varphi_1'@\mathrm{T} \mid \varphi_1' \in \mathrm{ND}(\varphi_1)\}$;
- if $\varphi = \varphi_1 \vee \varphi_2$, then $\mathrm{ND}(\varphi) \stackrel{\text{def}}{=} \mathrm{ND}(\varphi_1) \cup \mathrm{ND}(\varphi_2)$;
- if $\varphi = \varphi_1 \bullet \varphi_2$ for $\bullet \in \{\star, \circledast, \wedge\}$, then $\mathrm{ND}(\varphi) \stackrel{\text{def}}{=} \{\varphi_1' \bullet \varphi_2' \mid \varphi_1' \in \mathrm{ND}(\varphi_1), \varphi_2' \in \mathrm{ND}(\varphi_2)\}$.

▶ **Lemma 10.** *Let $\mathcal{R}$ be an SID and $\varphi$ an OSL formula with no negation $\neg$. Let $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ be an OSL structure. Then, $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi$ if and only if there exists a formula $\varphi' \in ND(\varphi)$ such that $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi'$.*

At this point, the entailment problem is reduced to a set of simpler instances in which the left-hand side $\varphi$ does not contain $\vee$ or $\neg$. We further simplify the entailments by computing a weakened form of the left-hand side $\varphi$. This weaker formula is obtained by removing $\wedge$ operators from $\varphi$ by keeping only the left-most branch for every conjunction in $\varphi$, by replacing all occurrences of $\star$ by $\circledast$ and by deleting all connectives @ operating on non-predicate atoms. We also remove pure formulas from the weakened form of $\varphi$. Formally, for any formula $\varphi$ with no occurrences of $\vee$ or $\neg$, we define the formula $\mathrm{WK}(\varphi) \in \Phi$ inductively by:

- if $\varphi = \varphi_1 \bullet \varphi_2$ for $\bullet \in \{\star, \circledast\}$, then $\mathrm{WK}(\varphi) \stackrel{\text{def}}{=} \mathrm{WK}(\varphi_1) \circledast \mathrm{WK}(\varphi_2)$;
- if $\varphi = \varphi_1 \wedge \varphi_2$, then $\mathrm{WK}(\varphi) \stackrel{\text{def}}{=} \mathrm{WK}(\varphi_1)$;
- if $\varphi = \varphi_1 @\mathrm{T}$ and $\varphi_1$ is not a predicate atom, then $\mathrm{WK}(\varphi) \stackrel{\text{def}}{=} \mathrm{WK}(\varphi_1)$;
- if $\varphi$ is pure, then $\mathrm{WK}(\varphi) \stackrel{\text{def}}{=} \mathtt{emp}$; and otherwise $\mathrm{WK}(\varphi) \stackrel{\text{def}}{=} \varphi$.

It is easy to see that $\varphi \models_{\mathcal{R}} \mathrm{WK}(\varphi)$, thus $\varphi \models_{\mathcal{R}} \psi \iff \varphi \wedge \mathrm{WK}(\varphi) \models_{\mathcal{R}} \psi \iff \mathrm{WK}(\varphi) \models_{\mathcal{R}} \psi \vee \neg\varphi$.

We immediately deduce the following:

▶ **Proposition 11.** *For all formulas $\varphi$ and $\psi$:*
$\varphi \models_{\mathcal{R}} \psi \iff \varphi \wedge WK(\varphi) \models_{\mathcal{R}} \psi \iff WK(\varphi) \models_{\mathcal{R}} \psi \vee \neg\varphi$.

By Lemma 10 and Proposition 11, we get an entailment whose antecedent $\big(\mathrm{WK}(\varphi)\big)$ does not contain occurrences of operators $\neg$, $\vee$, $\wedge$, and $\star$. Finally, we replace every predicate atom $p(\vec{x})$ in $\mathrm{WK}(\varphi)$ not already occurring in a pattern $p(\vec{x})@\mathrm{T}$, by an atom $p(\vec{x})@\mathrm{X}$ where $\mathrm{X} \in \mathcal{S}$ is a fresh set variable. We denote the resulting formula by $\mathrm{AS}(\mathrm{WK}(\varphi))$. This operation preserves the validity of the entailment, as any model of $\mathrm{WK}(\varphi) \wedge \neg(\psi \vee \neg\varphi)$ can be extended into a model of $\mathrm{AS}(\mathrm{WK}(\varphi))$ by mapping each fresh set variable X to the domain of the heap corresponding to the associated predicate atom $p(\vec{x})$. We get a reduction to a problem, $\mathrm{AS}(\mathrm{WK}(\varphi)) \models_{\mathcal{R}} \psi \vee \neg\varphi$, where the antecedent is an $\circledast$-formula.

▶ **Example 12.** Consider the entailment problem $\varphi \models_{\mathcal{R}} \psi$ with $\varphi = \big(p(x,y) \wedge q(x,y)\big) \vee \big(r(x) \star s(y)\big)$. We have $\mathrm{ND}(\varphi) = \{p(x,y) \wedge q(x,y), r(x) \star s(y)\}$ which yields two branches $p(x,y) \wedge q(x,y) \models_{\mathcal{R}} \psi$ and $r(x) \star s(y) \models_{\mathcal{R}} \psi$; any countermodel of one of these branches is a countermodel of $\varphi \models_{\mathcal{R}} \psi$. These problems are then normalised into $p(x,y) \models_{\mathcal{R}} \psi \vee \neg\big(p(x,y) \wedge q(x,y)\big)$ and $r(x) \circledast s(y) \models_{\mathcal{R}} \psi \vee \neg\big(r(x) \star s(y)\big)$, respectively. Finally, fresh set variables are introduced to denote the heap domains corresponding to the predicate atoms in the left-hand side, yielding: $p(x,y)@\mathrm{X} \models_{\mathcal{R}} \psi \vee \neg\big(p(x,y) \wedge q(x,y)\big)$ and $(r(x)@\mathrm{Y}) \circledast (s(y)@\mathrm{Z}) \models_{\mathcal{R}} \psi \vee \neg\big(r(x) \star s(y)\big)$.

## 3.2 Allocating Every Right-hand Side's Variable

The next step aims to ensure that all location variables in the entailment's consequent occur at root positions on the antecedent (possibly modulo aliasing). We first introduce a transformation that allows one to focus on countermodels in which all these variables are allocated. To this aim, we $\circledast$-conjunct dummy atoms $\xi.f \to ()$ to both sides of the entailment, where $\xi$ is a variable occurring in the consequent and $f \in \mathcal{F}$. As the allocated variables may depend on the model, we need to guess nondeterministically which couples $(\xi, f)$ need to be allocated. The following lemma asserts the correctness of the transformation.

▶ **Lemma 13.** *Let $\varphi$, $\psi$ be two OSL formulas over a SID $\mathcal{R}$, and let $\{\xi_1, \ldots, \xi_\eta\} \subseteq \mathcal{V}$ be a finite set of location variables. Let $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ be a structure. For every $\mathfrak{G} \subseteq [\![1, \eta]\!] \times \mathcal{F}$, we denote by $\theta_\mathfrak{G}$ the formula $\circledast_{(i,f)\in\mathfrak{G}} \xi_i.f \to ()$. A structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ satisfies $\varphi$ but not $\psi$ if and only if, for some guess $\mathfrak{G} \subseteq [\![1, \eta]\!] \times \mathcal{F}$, $\mathfrak{h}$ can be extended into a heap $\mathfrak{h}'$ of domain $\mathrm{dom}(\mathfrak{h}) \uplus \{(\mathfrak{s}(\xi_i), f) \mid (i, f) \in \mathfrak{G}\}$ such that $(\mathfrak{s}, \mathfrak{h}', \Sigma)$ satisfies $\varphi \circledast \theta_\mathfrak{G}$ but not $\psi \circledast \theta_\mathfrak{G}$, and moreover $\{\mathfrak{s}(\xi_i) \mid i \in [\![1, \eta]\!]\} \times \mathcal{F} \subseteq \mathrm{dom}(\mathfrak{h}')$.*

Next, we aim to further reduce the entailment problem to finding countermodels $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ of problems $\varphi \models_\mathcal{R} \psi$ satisfying $\mathfrak{s}(\mathrm{fv}^{\mathrm{loc}}(\psi)) \subseteq \mathfrak{s}(\mathrm{roots}(\varphi, f))$, for all $f \in \mathcal{F}$. By Definition 6, a location is allocated by a predicate atom if and only if it is associated with the root of some atom occurring in an unfolding of the predicate atom. The previous transformation ensures that every variable $\xi$ in $\mathrm{fv}^{\mathrm{loc}}(\psi)$ is allocated in $\varphi$. So we have to ensure that predicate atoms $p_{\mathrm{fin}}(\vec{\zeta})$ with $\vec{\zeta}[1]$ aliasing some $\xi \in \mathrm{fv}^{\mathrm{loc}}(\psi)$, appearing in some unfolding of $\varphi$, occur immediately in the antecedent. This requires (i) transforming the inductive rules in $\mathcal{R}$ to delete such $p_{\mathrm{fin}}(\vec{\zeta})$ atoms from the unfolding of some predicate atom in $\varphi$, and (ii) adding it into the formula $\varphi$[4]. To ensure that $\mathfrak{s}(\mathrm{fv}^{\mathrm{loc}}(\psi)) \subseteq \bigcap_{f\in\mathcal{F}} \mathfrak{s}(\mathrm{roots}(\varphi, f))$ we need to guess the set of couples $(\xi, f) \in \mathrm{fv}^{\mathrm{loc}}(\psi) \times \mathcal{F}$ such that $\mathfrak{s}(\xi) \notin \mathfrak{s}(\mathrm{roots}(\varphi, f))$ and apply the transformation described above on all such couples (as this set may depend on the countermodel).

More formally, let $\mathcal{R}$ be a SID and $\varphi = \pi \circledast \circledast_{i=1}^{n} \varphi_i$ an $\circledast$-formula. Let $(\xi, f) \in \mathrm{fv}^{\mathrm{loc}}(\psi) \times \mathcal{F}$. We transform $\mathcal{R}$ and $\varphi$ as follows. First, we make the following (nondeterministic) guesses: (i) an $\{f\}$-predicate atom $p_{\mathrm{init}}(\overrightarrow{x_{\mathrm{init}}})@\mathrm{T}$ occurring in $\varphi$, (ii) a rule $\rho_{\mathrm{fin}} \in \mathcal{R}$, and (iii) an $\{f\}$-predicate atom $p_{\mathrm{fin}}(\vec{z})$ occurring as $\ell$-th predicate atom in $\rho_{\mathrm{fin}}$. Then, we consider a tuple of $\#(p_{\mathrm{fin}})$ fresh variables $\vec{\zeta}$, and we introduce new predicates $\tilde{p}$ of arity $\#(\tilde{p}) = \#(p) + \#(p_{\mathrm{fin}})$ for every $\{f\}$-predicate $p$ occurring in $\mathcal{R}$. Rules for $\tilde{p}$ are added to $\mathcal{R}$ as follows: for every rule $\rho : p(\vec{x}) \Leftarrow \vec{x}[1].f \to (\vec{y}) \star \bigstar_{j=1}^{m} q_j(\vec{z_j}) \star \pi_\rho$, and for every $k \in [\![1, m]\!]$, we add a rule $\rho_{(k)} : \tilde{p}(\vec{x}, \vec{\zeta}) \Leftarrow \vec{x}[1].f \to (\vec{y}) \star \tilde{q}_k(\vec{z_k}, \vec{\zeta}) \star \bigstar_{j\in[\![1,m]\!]\setminus\{k\}} q_j(\vec{z_j}) \star \pi_\rho$. When $\rho = \rho_{\mathrm{fin}}$, we also add a rule (where $\vec{z_\ell} = (z_{\ell,1}, \ldots, z_{\ell,\#(p_{\mathrm{fin}})})$):

$$\rho_{\mathrm{end}} : \tilde{p}(\vec{x}, \vec{\zeta}) \Leftarrow \vec{x}[1].f \to (\vec{y}) \star \bigstar_{j=1}^{\#(p_{\mathrm{fin}})} \vec{\zeta}[j] \approx z_{\ell,j} \star \bigstar_{j\in[\![1,m]\!]\setminus\{\ell\}} q_j(\vec{z_j}) \star \pi_\rho .$$

We replace in $\varphi$ the atom $p_{\mathrm{init}}(\overrightarrow{x_{\mathrm{init}}})@\mathrm{T}$ by the formula $\tilde{p}_{\mathrm{init}}(\overrightarrow{x_{\mathrm{init}}}, \vec{\zeta})@\mathrm{Y} \circledast p_{\mathrm{fin}}(\vec{\zeta})@\mathrm{Z} \circledast \vec{\zeta}[1] \approx \xi \circledast \mathrm{T} \approx \mathrm{Y} \sqcup \mathrm{Z}$, where $\vec{\zeta}$ and $\mathrm{Y}, \mathrm{Z}$ are new free variables. We denote by $\mathcal{R}^{\ddagger}_{\xi,f,\mathfrak{G}}$ and $\varphi^{\ddagger}_{\xi,f,\mathfrak{G}}$ the results of the transformation, where $\mathfrak{G} \in \{p(\vec{x})@\mathrm{T} \text{ occurring in } \varphi\} \times \{\rho_{\mathrm{fin}} \in \mathcal{R}\} \times \{p(\vec{z}) \text{ occurring in } \rho_{\mathrm{fin}}\}$ is the tuple of guesses made. Lemma 14 asserts the soundness of the transformation.

▶ **Lemma 14.** *Let $\mathcal{R}$ be an SID with PCE 1-predicate rules and $\varphi$ an $\circledast$-formula. Let $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ be an OSL structure. Let $\xi$ be a variable and let $f \in \mathcal{F}$.*

1. *If $(\mathfrak{s}(\xi), f) \in \mathrm{dom}(\mathfrak{h})$, $\mathfrak{s}(\xi) \notin \mathfrak{s}(\mathit{roots}(\varphi, f))$, and $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_\mathcal{R} \varphi$, then there exists a guess $\mathfrak{G}$, a store $\mathfrak{s}'$, a set interpretation $\Sigma'$ and a variable $x$ such that $(\mathfrak{s}', \mathfrak{h}, \Sigma') \models_{\mathcal{R}^{\ddagger}_{\xi,f,\mathfrak{G}}} \varphi^{\ddagger}_{\xi,f,\mathfrak{G}} \circledast \xi \approx x$, $\mathit{roots}(\varphi^{\ddagger}_{\xi,f,\mathfrak{G}}, f) = \{x\} \uplus \mathit{roots}(\varphi, f)$, $\mathit{roots}(\varphi^{\ddagger}_{\xi,f,\mathfrak{G}}, g) = \mathit{roots}(\varphi, g)$ for all $g \in \mathcal{F} \setminus \{f\}$, $\mathfrak{s}(y) = \mathfrak{s}'(y)$ for all $y \in \mathrm{fv}^{\mathit{loc}}(\varphi)$, and $\Sigma(\mathrm{X}) = \Sigma'(\mathrm{X})$ for all $\mathrm{X} \in \mathrm{fv}^{\mathit{set}}(\varphi)$.*
2. *For every guess $\mathfrak{G}$, if $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}^{\ddagger}_{\xi,f,\mathfrak{G}}} \varphi^{\ddagger}_{\xi,f,\mathfrak{G}}$, then $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_\mathcal{R} \varphi$.*

[4] Note that new free variables may be introduced during this process, corresponding to the existential variables introduced by recursive calls occurring above $p_{\mathrm{fin}}(\vec{\zeta})$ in the unfolding. This does not lead to divergence, since these variables do not appear on the right-hand side.

▶ **Example 15.** Consider the entailment problem $p(x,y)@\mathrm{X} \models_{\mathcal{R}} \psi(x,y)$, with the following rules (generating a list of length strictly greater than 1):

$$p(u,v) \Leftarrow u.f \to (w) \star p(w,v)\,, \quad q(u,v) \Leftarrow u.f \to (v)\,, \quad p(u,v) \Leftarrow u.f \to (w) \star q(w,v)\,.$$

The variable $x$ already appears at an $f$-root position in $p(x,y)$, but this is not the case for $y$. We have to consider two possibilities nondeterministically.

1. The variable $y$ is not allocated. This is encoded by transforming the entailment into: $y.f \to () \circledast p(x,y)@\mathrm{X} \models_{\mathcal{R}} y.f \to () \circledast \psi(x,y)$.
2. The variable $y$ is allocated. In this case, there must be a call to $p$ or $q$ that allocates $y$. This yields two branches: $\tilde{p}(x,y,\zeta_1,\zeta_2)@\mathrm{Y} \circledast p(\zeta_1,\zeta_2)@\mathrm{Z} \circledast \zeta_1 \approx y \star \mathrm{X} \approx \mathrm{Y} \sqcup \mathrm{Z} \models_{\mathcal{R}} \psi(x,y)$ and $\tilde{p}(x,y,\zeta_1,\zeta_2)@\mathrm{Y} \circledast q(\zeta_1,\zeta_2)@\mathrm{Z} \circledast \zeta_1 \approx y \star \mathrm{X} \approx \mathrm{Y} \sqcup \mathrm{Z} \models_{\mathcal{R}} \psi(x,y)$, with the rules[5]: $\tilde{p}(u,v,\zeta_1,\zeta_2) \Leftarrow u.f \to (w) \star \tilde{p}(w,v,\zeta_1,\zeta_2)$ and $\tilde{p}(u,v,\zeta_1,\zeta_2) \Leftarrow u.f \to (w) \star \zeta_1 \approx w \star \zeta_2 \approx v$.

## 4 Abstractions on Standard Separation Logic

We briefly recall some results concerning the entailment problem in standard SL (i.e., when only a single field is present) and draw some easy consequences of them. For conciseness, we omit the full entailment-testing algorithm and recall only the definitions needed to understand our contribution. This choice highlights the modularity of our approach: the algorithm can incorporate different abstraction-based decision procedures [10, 14, 4], depending on the target fragment. An $\{f\}$-*heap* is a heap $\mathfrak{h}$ such that $\mathrm{dom}_g(\mathfrak{h}) = \varnothing$ for all fields $g \neq f$. A 1-*heap* is a heap $\mathfrak{h}$ such that there exists at most one field $f$ such that $\mathrm{dom}_f(\mathfrak{h}) \neq \varnothing$. If a formula $\varphi$ contains no set variables, then its truth value depends solely on the store and heap; we may thus omit the set interpretation from the structure and write $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \varphi$ rather than $(\mathfrak{s},\mathfrak{h},\Sigma) \models_{\mathcal{R}} \varphi$. Let $\mathrm{V_G}$ be a special set of location variables denoting the variables occurring in the entailment under consideration. In the following, we assume, w.l.o.g., that these variables occur in the domain of every store $\mathfrak{s}$ (so that the atoms occurring in the entailment can be evaluated in any structure $(\mathfrak{s},\mathfrak{h})$). If V is a set of variables, we thus denote by $\mathfrak{s}|_{\mathrm{V}}$ the restriction of $\mathfrak{s}$ to the set $\mathrm{V} \cup \mathrm{V_G}$.

▶ **Lemma 16.** *There exists a total function mapping all structures* $(\mathfrak{s},\mathfrak{h})$*, where* $\mathfrak{s}$ *is a store with* $\mathrm{dom}(\mathfrak{s}) \supseteq \mathrm{V_G}$*, and* $\mathfrak{h}$ *is a* $\{f\}$*-heap, to so-called* abstractions *of* $(\mathfrak{s},\mathfrak{h})$*, satisfying the following properties:*

1. *For any finite set of variables* V*, there are finitely many abstractions of structures* $(\mathfrak{s},\mathfrak{h})$ *such that* $\mathrm{dom}(\mathfrak{s}) \subseteq \mathrm{V} \cup \mathrm{V_G}$.
2. *An abstraction* $\mathcal{A}$ *provides access to the field* $\mathcal{A}.field = f$ *used, as well as to the domain of the store* $\mathcal{A}.V$*, to the aliasing relation* $\mathcal{A}.\sim$ *between free variables, and to the set of allocated free variables* $\mathcal{A}.alloc$*. More precisely, if* $\mathcal{A}$ *is the abstraction of* $(\mathfrak{s},\mathfrak{h})$*, then:* **(a)** $\mathrm{dom}(\mathfrak{s}) = \mathcal{A}.V$*;* **(b)** $\mathcal{A}.\sim$ *is the set of pairs* $(x,y)$ *such that* $\mathfrak{s}(x) = \mathfrak{s}(y)$*; and* **(c)** $\mathcal{A}.alloc$ *is the set of variables* $x \in \mathrm{dom}(\mathfrak{s})$ *such that* $\mathfrak{s}(x) \in \mathrm{dom}_f(\mathfrak{h})$*. A store* $\mathfrak{s}$ *is* compatible with an abstraction $\mathcal{A}$ *if* $\mathrm{dom}(\mathfrak{s}) = \mathcal{A}.V$ *and* $\{(x,y) \mid \mathfrak{s}(x) = \mathfrak{s}(y)\} = \mathcal{A}.\sim$*. Two stores* $\mathfrak{s}$ *and* $\mathfrak{s}'$ *are* compatible *if* $\mathrm{dom}(\mathfrak{s}) = \mathrm{dom}(\mathfrak{s}')$ *and* $\{(x,y) \mid \mathfrak{s}(x) = \mathfrak{s}(y)\} = \{(x,y) \mid \mathfrak{s}'(x) = \mathfrak{s}'(y)\}$.
3. *The abstraction provides enough information to test whether the spatial atoms built on* $\mathcal{A}.V$ *are satisfied. If* $\mathrm{fv}(\varphi) \subseteq \mathrm{dom}(\mathfrak{s})$ *and* $\mathcal{A}$ *is the abstraction of* $(\mathfrak{s},\mathfrak{h})$*, we write* $\mathcal{A} \models^{abst}_{\mathcal{R}} \varphi$ *(and say that* $\mathcal{A}$ *is* associated with $\varphi$*) if and only if* $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \varphi$*. Note that, by definition of* $\mathrm{V_G}$*,* $\mathrm{fv}(\varphi) \subseteq \mathrm{dom}(\mathfrak{s})$ *holds for all atoms* $\varphi$ *occurring in the entailment under consideration, hence the truth value of these atoms can be obtained from the abstraction.*

[5] In this case, the rules are identical across all branches. For conciseness, the rules containing $\tilde{q}$ are omitted, as there is no rule associated with $\tilde{q}$.

4. *Given two structures* $(\mathfrak{s}, \mathfrak{h}_1)$ *and* $(\mathfrak{s}, \mathfrak{h}_2)$*, the abstraction* $\mathcal{A}$ *of* $(\mathfrak{s}, \mathfrak{h}_1 \cup \mathfrak{h}_2)$ *depends only on the abstractions* $\mathcal{A}_i$ *of* $(\mathfrak{s}, \mathfrak{h}_i)$ *(for* $i \in \{1, 2\}$*). We write* $\mathcal{A} = \mathcal{A}_1 \star \mathcal{A}_2$*. Note that* $\mathcal{A}_1 \star \mathcal{A}_2$ *is defined exactly when* $\mathcal{A}_1.V = \mathcal{A}_2.V$*,* $\mathcal{A}_1.{\sim} = \mathcal{A}_2.{\sim}$ *and* $\mathcal{A}_1.alloc \cap \mathcal{A}_2.alloc = \varnothing$*.*

▶ **Example 17.** Consider the formula $\mathtt{ls}(x, y)$, denoting a nonempty list segment from $x$ to $y$. It is associated with several abstractions, each described by the set of formulas it satisfies. The most straightforward case is when the only additional satisfied formula is $\mathtt{ls}(x, y) \star x \not\approx y$, representing an acyclic list segment of length $> 1$ with distinct endpoints. In contrast, the abstraction satisfying $\mathtt{ls}(x, y) \star x \approx y$ denotes a cyclic list. If the segment length is 1, $x.f \to (y)$ also holds in both cases, yielding two other abstractions. A list looping on an inner node yields the formula $\mathtt{ls}(x, y) \star \mathtt{ls}(y, y)$ (implying $x \not\approx y$). Special cases where (at least) one of the two parts of the segment has length 1 must also be considered: $\mathtt{ls}(x, y) \star y.f \to (y)$, $x.f \to (y) \star \mathtt{ls}(y, y)$, and $x.f \to (y) \star y.f \to (y)$.

We say that two abstractions $\mathcal{A}_1$ and $\mathcal{A}_2$ are *combinable* if and only if ( $\mathcal{A}_1.V = \mathcal{A}_2.V$ , $\mathcal{A}_1.{\sim} = \mathcal{A}_2.{\sim}$ , and if $\mathcal{A}_1.field = \mathcal{A}_2.field$ then $\mathcal{A}_1.alloc \cap \mathcal{A}_2.alloc = \varnothing$ ). Additionally, we give a way to access the possible sizes of an abstraction:

▶ **Lemma 18.** *There exists an algorithm computing, for any abstraction* $\mathcal{A}$ *associated with an atom* $\varphi$*, a Presburger formula* $\mathcal{A}.card$*, with a single free variable*[6] $\kappa$*, describing the set of possible cardinalities of the abstracted structures, i.e., such that, for all* $k \in \mathbb{Z}$ *and all stores* $\mathfrak{s}$ *compatible with* $\mathcal{A}$*:* $\mathcal{A}.card[\kappa/k]$ *is valid if and only if there is a heap* $\mathfrak{h}$ *such that* $\mathcal{A}$ *is an abstraction of* $(\mathfrak{s}, \mathfrak{h})$*, and* $\mathrm{card}(\mathrm{dom}(\mathfrak{h})) = k$*.*

Informally, $\mathcal{A}.card$ is computed by first extracting, from the inductive rules used to compute abstractions, a context-free grammar that generates the possible cardinalities of abstractions (in unary), and then using existing results for computing the Parikh image of a context-free grammar with a single terminal [18].

## 5 Reduction to the Satisfiability Problem of a Pure Formula

In Section 3.1, we reduced the entailment problem to an instance where the antecedent is an $\circledast$-formula $\varphi = \pi \circledast \circledast_{i=1}^{n} \varphi_i$, associated with terms $\mathrm{T}_i$ (for $1 \leqslant i \leqslant n$). In Section 3.2, we reduced the entailment test $\varphi \models_{\mathcal{R}} \psi$ to the existence of a structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ such that $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi$, $(\mathfrak{s}, \mathfrak{h}, \Sigma) \not\models_{\mathcal{R}} \psi$ and $\mathfrak{s}(\mathrm{fv}^{\mathrm{loc}}(\psi)) \subseteq \mathfrak{s}(\mathrm{roots}(\varphi, f))$, for all $f \in \mathcal{F}$. In this section, we reduce both tests to the satisfiability of a pure formula.

### 5.1 Left-hand Side

First, we reduce the test $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi$ to the satisfiability of a pure formula using the abstractions described in Section 4. Let $\tilde{\varphi}_i \stackrel{\text{def}}{=} \varphi_i$ when $\varphi_i = x_i.f_i \to (\vec{y_i})$ (and $\mathrm{T}_i = \{x_i\}$), and $\tilde{\varphi}_i \stackrel{\text{def}}{=} p_i(\vec{x_i})$ when $\varphi_i = p_i(\vec{x_i})@\mathrm{T}_i$. For every $i \in [\![1, n]\!]$, we guess an abstraction $\mathcal{A}_i$ associated with $\tilde{\varphi}_i$ such that all $\mathcal{A}_i$ are pairwise combinable. The combination of these abstractions represents models for $\varphi$. Note that $\mathrm{fv}(\tilde{\varphi}_i) \subseteq \mathrm{V_G}$ by definition of $\mathrm{V_G}$ ; thus, $\mathcal{A}_i.V = \mathrm{V_G}$ necessarily holds for any abstraction $\mathcal{A}_i$ associated with $\tilde{\varphi}_i$. Moreover, since the abstractions $\mathcal{A}_i$ are combinable, $\mathcal{A}_i.{\sim}$ does not depend on $i$.

[6] In our framework, the variable $\kappa$ is encoded by a term $|\mathrm{K}|$, where K is a fresh set variable, as explained in Section 2.

Consider the pure formula $\pi_{\text{left}}$ defined as follows (assuming $n \geqslant 1$):

$$\pi \wedge \bigwedge_{(x,y)\in\mathcal{A}_1\bullet\sim} x \approx y \wedge \bigwedge_{(x,y)\notin\mathcal{A}_1\bullet\sim} (x \not\approx y) \wedge \bigwedge_{i=1}^{n} \left( \bigsqcup_{z\in\mathcal{A}_i\bullet alloc} \{z\} \approx \mathrm{T}_i \sqcap \bigsqcup_{z\in\mathcal{A}_i\bullet \mathtt{V}} \{z\} \right)$$
$$\wedge \bigwedge_{i<j \text{ such that } \mathcal{A}_i\bullet field=\mathcal{A}_j\bullet field} \mathrm{T}_i \sqcap \mathrm{T}_j \approx \varnothing \wedge \bigwedge_{i=1}^{n} \mathcal{A}_i\bullet card[\kappa/|\mathrm{T}_i|]$$

Note that $\pi_{\text{left}}$ depends on $\mathcal{A}_i$; we keep this dependence implicit for the sake of readability. The following lemma states that these abstractions give an alternative way to check the satisfiability of the formula $\varphi$:

▶ **Lemma 19.** *Let $\mathcal{R}$ be an SID with PCE 1-predicate rules and $\varphi = \pi \circledast \circledast_{i=1}^{n}\varphi_i$ an $\circledast$-formula. Let $\mathfrak{s}$ be a store and $\Sigma$ a set interpretation.*

1. *If $(\mathfrak{s},\mathfrak{h}_i,\Sigma) \models_{\mathcal{R}} \varphi_i$ for every $i \in [\![1,n]\!]$, $(\mathfrak{s},\Sigma) \models \pi$, and $(\mathrm{dom}(\mathfrak{h}_i))_{i\in[\![1,n]\!]}$ are pairwise disjoint, then the abstractions $\mathcal{A}_i$ of $(\mathfrak{s},\mathfrak{h}_i)$ for $i \in [\![1,n]\!]$ are pairwise combinable and satisfy $\mathcal{A}_i \models^{abst}_{\mathcal{R}} \tilde{\varphi}_i$ for every $i \in [\![1,n]\!]$ and $(\mathfrak{s},\Sigma) \models \pi_{left}$.*
2. *If pairwise combinable abstractions $(\mathcal{A}_i)_{i\in[\![1,n]\!]}$ satisfy $\mathcal{A}_i \models^{abst}_{\mathcal{R}} \tilde{\varphi}_i$ for all $i \in [\![1,n]\!]$ and $(\mathfrak{s},\Sigma) \models \pi_{left}$, then there are heaps $(\mathfrak{h}_i)_{i\in[\![1,n]\!]}$ with $(\mathrm{dom}(\mathfrak{h}_i))_{i\in[\![1,n]\!]}$ pairwise disjoint such that $(\mathfrak{s},\Sigma) \models \pi$, $(\mathfrak{s},\mathfrak{h}_i,\Sigma) \models_{\mathcal{R}} \varphi_i$, and $\mathcal{A}_i$ is the abstraction of $(\mathfrak{s},\mathfrak{h}_i)$ for every $i \in [\![1,n]\!]$.*

**Proof.** We denote by $(\mathrm{T}_i)_{i\in[\![1,n]\!]}$ the sequence of set terms associated with the $\circledast$-formula $\varphi = \pi \circledast \circledast_{i=1}^{n}\varphi_i$ (see Definition 9).

**(1)** Let $\mathfrak{h} = \biguplus_{i=1}^{n} \mathfrak{h}_i$ be a heap, and suppose $(\mathfrak{s},\mathfrak{h}_i,\Sigma) \models_{\mathcal{R}} \varphi_i$ for every $i \in [\![1,n]\!]$, $(\mathfrak{s},\Sigma) \models \pi$, and $(\mathrm{dom}(\mathfrak{h}_i))_{i\in[\![1,n]\!]}$ are pairwise disjoint.
For every $i \in [\![1,n]\!]$, $\varphi_i$ is a $\{f_i\}$-formula for some $f_i \in \mathcal{F}$, hence $\mathfrak{h}_i$ is a 1-heap, and we can define $\mathcal{A}_i$ as the abstraction of $(\mathfrak{s},\mathfrak{h}_i)$. We have $(\mathfrak{s},\mathfrak{h}_i) \models_{\mathcal{R}} \tilde{\varphi}_i$ hence $\mathcal{A}_i \models^{\text{abst}}_{\mathcal{R}} \tilde{\varphi}_i$ (as $\mathrm{fv}(\tilde{\varphi}_i) \subseteq \mathrm{V_G}$). Now $(\mathfrak{s},\Sigma) \models x \approx y$ if and only if $(x,y) \in \mathcal{A}_i\bullet\sim$ for all variables $x,y \in \mathrm{dom}(\mathfrak{s})$ and $i \in [\![1,n]\!]$, thus aliasing relations $(\mathcal{A}_i\bullet\sim)_{i\in[\![1,n]\!]}$ are identical. We recall that, by definition of abstractions, $\mathcal{A}_i\bullet alloc$ is the set of variables $x \in \mathrm{dom}(\mathfrak{s})$ such that $\mathfrak{s}(x) \in \mathrm{dom}_{f_i}(\mathfrak{h}_i)$ (for all $i \in [\![1,n]\!]$). Since $(\mathfrak{s},\mathfrak{h}_i,\Sigma) \models_{\mathcal{R}} \varphi_i$, we have $[\![\mathrm{T}_i]\!]_{(\mathfrak{s},\Sigma)} = \mathrm{alloc}(\mathfrak{h}_i) = \mathrm{dom}_{f_i}(\mathfrak{h}_i)$, hence the set $[\![\mathrm{T}_i \sqcap \bigsqcup_{z\in\mathcal{A}_i\bullet\mathtt{V}} \{z\}]\!]_{(\mathfrak{s},\Sigma)} = [\![\mathrm{T}_i]\!]_{(\mathfrak{s},\Sigma)} \cap \mathfrak{s}(\mathcal{A}_i\bullet V) = \mathrm{dom}_{f_i}(\mathfrak{h}_i) \cap \mathrm{img}(\mathfrak{s}) = \mathfrak{s}(\mathcal{A}_i\bullet alloc) = [\![\bigsqcup_{z\in\mathcal{A}_i\bullet alloc} \{z\}]\!]_{(\mathfrak{s},\Sigma)}$. Additionally, for all $i < j$ such that $\mathcal{A}_i\bullet field = \mathcal{A}_j\bullet field = f \in \mathcal{F}$, we have $[\![\mathrm{T}_i \sqcap \mathrm{T}_j]\!]_{(\mathfrak{s},\Sigma)} = \mathrm{dom}_f(\mathfrak{h}_i) \cap \mathrm{dom}_f(\mathfrak{h}_j) = \varnothing$, thus $\mathcal{A}_i\bullet alloc \cap \mathcal{A}_j\bullet alloc = \varnothing$. Finally, by Lemma 18, $[\![\,|\mathrm{T}_i|\,]\!]_{(\mathfrak{s},\Sigma)} = \mathrm{card}([\![\mathrm{T}_i]\!]_{(\mathfrak{s},\Sigma)}) = \mathrm{card}(\mathrm{dom}_{f_i}(\mathfrak{h}_i)) = k_i$ with $\mathcal{A}_i\bullet card[\kappa/k_i]$ for every $i \in [\![1,n]\!]$. Therefore the abstractions $(\mathcal{A}_i)_{i\in[\![1,n]\!]}$ are pairwise combinable, and $(\mathfrak{s},\Sigma) \models \pi_{\text{left}}$.

**(2)** Suppose there exist pairwise combinable abstractions $(\mathcal{A}_i)_{i\in[\![1,n]\!]}$ such that $\mathcal{A}_i \models^{\text{abst}}_{\mathcal{R}} \tilde{\varphi}_i$ for every $i \in [\![1,n]\!]$ and $(\mathfrak{s},\Sigma) \models \pi_{\text{left}}$.
For every $i \in [\![1,n]\!]$, since $\mathcal{A}_i \models^{\text{abst}}_{\mathcal{R}} \tilde{\varphi}_i$ and $\mathcal{A}_i\bullet card[\kappa/k_i]$ is valid for $k_i = [\![\,|\mathrm{T}_i|\,]\!]_{(\mathfrak{s},\Sigma)}$, we deduce by Lemma 18 that $\mathcal{A}_i$ is the abstraction of some structure $(\mathfrak{s}_i,\mathfrak{h}'_i) \models_{\mathcal{R}} \tilde{\varphi}_i$ of size $\mathrm{card}(\mathrm{dom}(\mathfrak{h}'_i)) = [\![\,|\mathrm{T}_i|\,]\!]_{(\mathfrak{s},\Sigma)}$. From $(\mathfrak{s},\Sigma) \models \pi_{\text{left}}$, we know that:

- $(\mathfrak{s},\Sigma) \models \pi$;
- for all $x,y \in \mathcal{A}_1\bullet V$, $\mathfrak{s}(x) = \mathfrak{s}(y)$ if and only if $(x,y) \in \mathcal{A}_1\bullet\sim$; since $\mathcal{A}_1$ and $\mathcal{A}_i$ are combinable for every $i \in [\![2,n]\!]$, the same result applies to any $\mathcal{A}_i$;
- $\mathfrak{s}(\mathcal{A}_i\bullet alloc) = [\![\mathrm{T}_i]\!]_{(\mathfrak{s},\Sigma)} \cap \mathfrak{s}(\mathcal{A}_i\bullet V)$ for all $i \in [\![1,n]\!]$;
- $[\![\mathrm{T}_i]\!]_{(\mathfrak{s},\Sigma)} \cap [\![\mathrm{T}_j]\!]_{(\mathfrak{s},\Sigma)} = \varnothing$ for all $i < j$ such that $\mathcal{A}_i\bullet field = \mathcal{A}_j\bullet field$.

From these results and for every $i \in [\![1,n]\!]$, we can rename locations used in $(\mathfrak{s}_i, \mathfrak{h}'_i)$ by mapping $\mathfrak{s}_i(x)$ to $\mathfrak{s}(x)$ for every $x \in \mathcal{A}_i{\bullet}\, V$ and every location in $\mathrm{dom}_{\mathcal{A}_i{\bullet}\mathit{field}}(\mathfrak{h}'_i) \setminus \mathrm{img}(\mathfrak{s}_i) = \mathrm{alloc}(\mathfrak{h}'_i) \setminus \mathfrak{s}_i(\mathcal{A}_i{\bullet}\mathit{alloc})$ to locations in $[\![\mathrm{T}_i]\!]_{(\mathfrak{s},\Sigma)} \setminus \mathrm{img}(\mathfrak{s})$, since both sets of locations have the same size. Let $\mathfrak{h}_i$ be the new heap obtained after renaming, and note that $\mathcal{A}_i$ is now an abstraction of $(\mathfrak{s}, \mathfrak{h}_i)$. We obtain $(\mathfrak{s}, \mathfrak{h}_i) \models_{\mathcal{R}} \tilde{\varphi}_i$ and $\mathrm{alloc}(\mathfrak{h}_i) = [\![\mathrm{T}_i]\!]_{(\mathfrak{s},\Sigma)}$, hence $(\mathfrak{s}, \mathfrak{h}_i, \Sigma) \models_{\mathcal{R}} \varphi_i$. For all $i < j$ such that $\mathcal{A}_i{\bullet}\mathit{field} = \mathcal{A}_j{\bullet}\mathit{field} = f \in \mathcal{F}$, we have $\mathrm{dom}_f(\mathfrak{h}_i) \cap \mathrm{dom}_f(\mathfrak{h}_j) = [\![\mathrm{T}_i]\!]_{(\mathfrak{s},\Sigma)} \cap [\![\mathrm{T}_j]\!]_{(\mathfrak{s},\Sigma)} = \varnothing$. Consequently, the sets $(\mathrm{dom}(\mathfrak{h}_i))_{i\in[\![1,n]\!]}$ are pairwise disjoint, and $(\mathfrak{s}, \biguplus_{i=1}^{n} \mathfrak{h}_i, \Sigma) \models_{\mathcal{R}} \varphi$. ◀

▶ **Example 20.** Consider the formula $\varphi_1 = p_f(x)@\mathrm{X} \circledast q_g(x)@\mathrm{X}$ together with the rules $p_f(u) \Leftarrow u.f \to ()$, $q_f(u) \Leftarrow u.f \to ()$, $q_f(u) \Leftarrow u.f \to (v) \star t_f(v)$, $t_f(u) \Leftarrow u.f \to ()$ and $t_f(u) \Leftarrow u.f \to (v, w) \star t_f(v) \star t_f(w)$. The atom $p_f(u)$ allocates a single memory cell $u$ containing an empty tuple, $t_f(u)$ denotes a binary tree rooted at $u$, and $q_f(u)$ denotes a memory cell $u$ containing either an empty tuple or a link to the root of a binary tree (all predicates are indexed by the field they use). Assume that $\mathrm{V_G} = \{x\}$. Then, $p_f(u)$ is associated with exactly one abstraction, where $u$ is allocated, and the heap is of cardinality 1. The atom $q_g(u)$ has two abstractions, both allocating $u$, one where the heap is of cardinality 1 and the other where the heap is of any cardinality $\geqslant 2$. In the first case, we get (after removing redundant subformulas) the following satisfiable formula: $\pi_{\text{left}} = \{x\} \approx \mathrm{X} \sqcap \{x\} \wedge |\mathrm{X}| \approx 1$. In the second case, this yields $\pi_{\text{left}} = \{x\} \approx \mathrm{X} \sqcap \{x\} \wedge |\mathrm{X}| \approx 1 \wedge 1 \prec |\mathrm{X}|$ (which is unsatisfiable).

Next, consider the formula $\varphi_2 = t_f(x)@\mathrm{X} \circledast t_f(y)@\mathrm{X}$. Assuming that $\mathrm{V_G} = \{x, y\}$, and considering abstractions such that $x$ and $y$ do not alias and each atom allocates both $x$ and $y$, we get the (unsatisfiable) formula: $\pi_{\text{left}} = x \not\approx y \wedge \{x, y\} \approx \mathrm{X} \sqcap \{x, y\} \wedge \mathrm{X} \sqcap \mathrm{X} \approx \varnothing \wedge 1 \prec |\mathrm{X}|$.

Finally, consider the formula $\varphi_3 = t_f(x)@\mathrm{X} \circledast t_g(y)@\mathrm{X}$ and $\mathrm{V_G} = \{x, y\}$. Using again abstractions allocating both $x$ and $y$, we get the following satisfiable formula: $\pi_{\text{left}} = x \not\approx y \wedge \{x, y\} \approx \mathrm{X} \sqcap \{x, y\} \wedge 1 \prec |\mathrm{X}|$. This corresponds to models in which $y$ and $x$ occur in the tree allocated by $t_f(x)$ and $t_g(y)$, respectively. In contrast, if we consider abstractions allocating no free variables other than the root of the atom, then the obtained formula is unsatisfiable: $\pi_{\text{left}} = x \not\approx y \wedge \{x\} \approx \mathrm{X} \sqcap \{x, y\} \wedge \{y\} \approx \mathrm{X} \sqcap \{x, y\} \wedge 0 \prec |\mathrm{X}|$.

## 5.2 Right-hand Side

We now focus on reducing the test $(\mathfrak{s}, \mathfrak{h}, \Sigma) \not\models_{\mathcal{R}} \psi$ to the satisfiability of a pure formula. Using Lemma 19, we try to match the combination of abstractions $(\mathcal{A}_i)_{i\in[\![1,n]\!]}$ (representing sets of structures $(\mathfrak{s}, \mathfrak{h}, \Sigma)$) against the formula $\psi$, and to express the possibility of a matching through a pure formula. For this, we use the following lemma, which ensures that the heaps corresponding to the abstractions $\mathcal{A}_i$ are distributed as a whole on each side of any separating conjunction of $\psi$, instead of being split.

▶ **Lemma 21.** *Let $\mathcal{R}$ be an SID with PCE 1-predicate rules and $\varphi = \pi \circledast \circledast_{i=1}^{n} \varphi_i$ an $\circledast$-formula. Let $(\mathfrak{s}, \biguplus_{i=1}^{n} \mathfrak{h}_i, \Sigma)$ be a model of $\varphi$, with $(\mathfrak{s}, \Sigma) \models \pi$, $(\mathfrak{s}, \mathfrak{h}_i, \Sigma) \models_{\mathcal{R}} \varphi_i$ for every $i \in [\![1,n]\!]$, and $(\mathrm{dom}(\mathfrak{h}_i))_{i\in[\![1,n]\!]}$ pairwise disjoint.*

*Let $\psi_1 \circledast \psi_2$ be an OSL-formula such that $\mathfrak{s}(\mathrm{fv}^{loc}(\psi_1 \circledast \psi_2)) \subseteq \mathfrak{s}(\mathit{roots}(\varphi, f))$ for all fields $f \in \mathcal{F}$, and $\mathrm{J} \subseteq [\![1,n]\!]$ such that $(\mathfrak{s}, \biguplus_{i\in\mathrm{J}} \mathfrak{h}_i, \Sigma) \models_{\mathcal{R}} \psi_1 \circledast \psi_2$. Then, there exists a decomposition $\mathrm{J} = \mathrm{J}_1 \uplus \mathrm{J}_2$ such that $(\mathfrak{s}, \biguplus_{i\in\mathrm{J}_1} \mathfrak{h}_i, \Sigma) \models_{\mathcal{R}} \psi_1$ and $(\mathfrak{s}, \biguplus_{i\in\mathrm{J}_2} \mathfrak{h}_i, \Sigma) \models_{\mathcal{R}} \psi_2$.*

The following definition builds a pure formula $\pi_{\text{right}}(\psi, [\![1,n]\!])$ expressing the match between $\psi$ and $(\mathcal{A}_i)_{i\in[\![1,n]\!]}$. The definition is done by induction on $\psi$. For each sub-formula $\psi'$ of $\psi$, the parameter $\mathrm{J} \subseteq [\![1,n]\!]$ tracks the abstractions to be matched by $\psi'$. At separating conjunctions, we guess how J is split over the two sub-formulas.

▶ **Definition 22.** *Given fixed abstractions* $(\mathcal{A}_i)_{i\in[\![1,n]\!]}$*, for any subformula* $\psi'$ *of* $\psi$ *and any set of natural numbers* $\mathrm{J}\subseteq[\![1,n]\!]$*, the formula* $\pi_{right}(\psi',\mathrm{J})$ *is defined by induction on* $\psi'$*:*

1. *for* $\psi'=\mathtt{emp}$*, if* $\mathrm{J}=\varnothing$*, then* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\mathrm{true}$*, otherwise* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\mathrm{false}$*;*
2. *for* $\psi'=x.f\to(\vec{y})$*, if* $\mathrm{J}=\{i\}$*,* $\mathcal{A}_i{\bullet}field=f$ *and* $\mathcal{A}_i\models^{abst}_{\mathcal{R}}\psi'$ *(for some* $i\in[\![1,n]\!]$*) then* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\mathrm{true}$*, otherwise* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\mathrm{false}$*;*
3. *for* $\psi'=p(\vec{x})$ *where* $p$ *is an* $\{f\}$*-predicate, if* $\mathcal{A}_i{\bullet}field=f$ *for every* $i\in\mathrm{J}$ *and* $\bigstar_{i\in\mathrm{J}}\mathcal{A}_i\models^{abst}_{\mathcal{R}}p(\vec{x})$ *then* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\mathrm{true}$*, otherwise* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\mathrm{false}$*;*
4. *for* $\psi'=\psi_1@\mathrm{U}$*,* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\pi_{right}(\psi_1,\mathrm{J})\ \wedge\ \bigsqcup_{i\in\mathrm{J}}\mathrm{T}_i\approx\mathrm{U}$*;*
5. *for* $\psi'=\psi_1\circledast\psi_2$*,* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\bigvee_{\mathrm{J}_1\uplus\mathrm{J}_2=\mathrm{J}}(\pi_{right}(\psi_1,\mathrm{J}_1)\wedge\pi_{right}(\psi_2,\mathrm{J}_2))$*;*
6. *for* $\psi'=\psi_1\star\psi_2$*,* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\bigvee_{\mathrm{J}_1\uplus\mathrm{J}_2=\mathrm{J}}(\pi_{right}(\psi_1,\mathrm{J}_1)\wedge\pi_{right}(\psi_2,\mathrm{J}_2)\wedge\bigsqcup_{i\in\mathrm{J}_1}\mathrm{T}_i\sqcap\bigsqcup_{i\in\mathrm{J}_2}\mathrm{T}_i\approx\varnothing)$*;*
7. *for* $\psi'=\psi_1\bullet\psi_2$ *(with* $\bullet\in\{\wedge,\vee\}$*),* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\pi_{right}(\psi_1,\mathrm{J})\bullet\pi_{right}(\psi_2,\mathrm{J})$*;*
8. *for* $\psi'=\neg\psi_1$*,* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\sim\pi_{right}(\psi_1,\mathrm{J})$*;*
9. *for* $\psi'$ *a pure formula, if* $\mathrm{J}=\varnothing$*, then* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\psi'$*, otherwise* $\pi_{right}(\psi',\mathrm{J})\stackrel{def}{=}\mathrm{false}$*.*

▶ **Lemma 23.** *Let* $\mathcal{R}$ *be an SID with PCE 1-predicate rules and* $\varphi=\pi\ \circledast\ \circledast_{i=1}^{n}\varphi_i$ *an* $\circledast$*-formula. Let* $(\mathfrak{s},\biguplus_{i=1}^{n}\mathfrak{h}_i,\Sigma)$ *be a model of* $\varphi$*, where* $(\mathfrak{s},\Sigma)\models\pi$*,* $(\mathfrak{s},\mathfrak{h}_i,\Sigma)\models_{\mathcal{R}}\varphi_i$ *for every* $i\in[\![1,n]\!]$*, and* $(\mathrm{dom}(\mathfrak{h}_i))_{i\in[\![1,n]\!]}$ *are pairwise disjoint, and let* $\mathcal{A}_i$ *be the abstraction of* $(\mathfrak{s},\mathfrak{h}_i)$*. Let* $\psi$ *be an OSL-formula such that* $\mathfrak{s}(\mathrm{fv}^{loc}(\psi))\subseteq\mathfrak{s}(roots(\varphi,f))$ *for all fields* $f\in\mathcal{F}$*. Then,* $(\mathfrak{s},\biguplus_{i=1}^{n}\mathfrak{h}_i,\Sigma)\models_{\mathcal{R}}\psi$ *if and only if* $(\mathfrak{s},\Sigma)\models\pi_{right}(\psi,[\![1,n]\!])$*.*

**Proof.** We show that $(\mathfrak{s},\biguplus_{j\in\mathrm{J}}\mathfrak{h}_j,\Sigma)\models_{\mathcal{R}}\psi'$ if and only if $(\mathfrak{s},\Sigma)\models\pi_{\mathrm{right}}(\psi',\mathrm{J})$, for all $\mathrm{J}\subseteq[\![1,n]\!]$ and $\psi'$ sub-formula of $\psi$, by induction on $\psi'$. Then, it suffices to take $\psi'=\psi$ and $\mathrm{J}=[\![1,n]\!]$ to get the result. We denote by $(\mathrm{T}_i)_{i\in[\![1,n]\!]}$ the sequence of set terms associated with the formula $\varphi=\pi\ \circledast\ \circledast_{i=1}^{n}\varphi_i$ in Definition 9. We recall that, by the progress condition, a predicate atom cannot hold if the heap is empty. Thus, the heaps $\mathfrak{h}_i$ (for $i\in[\![1,n]\!]$) are all nonempty. Moreover, $\mathfrak{h}_i$ is an $\{\mathcal{A}_i{\bullet}field\}$-heap. Note also that $\mathrm{fv}^{\mathrm{loc}}(\psi')\subseteq\mathrm{V_G}$ by definition of $\mathrm{V_G}$, so that $\mathrm{fv}^{\mathrm{loc}}(\psi')\subseteq\mathcal{A}{\bullet}V$, for all abstractions $\mathcal{A}$.

- For $\psi'=\mathtt{emp}$, if $\mathrm{J}=\varnothing$, then $\mathrm{dom}\left(\biguplus_{j\in\mathrm{J}}\mathfrak{h}_j\right)=\varnothing$, which implies $(\mathfrak{s},\varnothing,\Sigma)\models_{\mathcal{R}}\mathtt{emp}$ and $(\mathfrak{s},\Sigma)\models\mathrm{true}$, hence the equivalence. On the other hand, if $\mathrm{J}\neq\varnothing$, since no $\mathfrak{h}_i$ are empty, we have both $(\mathfrak{s},\biguplus_{j\in\mathrm{J}}\mathfrak{h}_j,\Sigma)\not\models_{\mathcal{R}}\mathtt{emp}$ and $(\mathfrak{s},\Sigma)\not\models\mathrm{false}$.
- For $\psi'=x.f\to(y_1,\ldots,y_d)$, if $(\mathfrak{s},\biguplus_{j\in\mathrm{J}}\mathfrak{h}_j,\Sigma)\models_{\mathcal{R}}\psi'$, in that case $\biguplus_{j\in\mathrm{J}}\mathfrak{h}_j=[(\mathfrak{s}(x),f)\mapsto(\mathfrak{s}(y_1),\ldots,\mathfrak{s}(y_d))]$ and therefore $\mathrm{J}=\{i\}$ for some $i\in[\![1,n]\!]$. $\mathcal{A}_i$ is the abstraction of $(\mathfrak{s},\mathfrak{h}_i)$, hence $\mathcal{A}_i{\bullet}field=f$ and $\mathcal{A}_i\models^{\mathrm{abst}}_{\mathcal{R}}\psi'$ (as $\mathrm{fv}(\psi')\subseteq\mathrm{V_G}$).
  Conversely, if $(\mathfrak{s},\Sigma)\models\pi_{\mathrm{right}}(\psi',\mathrm{J})$, then $\mathrm{J}=\{i\}$ and $\mathcal{A}_i\models^{\mathrm{abst}}_{\mathcal{R}}\psi'$ for some $i\in[\![1,n]\!]$ by definition, which implies that $(\mathfrak{s},\mathfrak{h}_i,\Sigma)\models_{\mathcal{R}}\psi'$.
- For $\psi'=p(\vec{x})$ for a $\{f\}$-predicate $p$, if $(\mathfrak{s},\biguplus_{j\in\mathrm{J}}\mathfrak{h}_j,\Sigma)\models_{\mathcal{R}}\psi'$, then $\mathrm{dom}_g(\mathfrak{h}_j)=\varnothing$ for all fields $g\neq f$ and for all $j\in\mathrm{J}$. Consequently, every $\varphi_j$ must be an $\{f\}$-formula and $\mathcal{A}_i{\bullet}field=f$ for $j\in\mathrm{J}$. By combining the abstractions, we obtain that $\bigstar_{j\in\mathrm{J}}\mathcal{A}_j$ is the abstraction of $(\mathfrak{s},\biguplus_{j\in\mathrm{J}}\mathfrak{h}_j)$, thus $\bigstar_{j\in\mathrm{J}}\mathcal{A}_j\models^{\mathrm{abst}}_{\mathcal{R}}\psi'$ (since $\mathrm{fv}(\psi')\subseteq\mathrm{V_G}$), which implies $(\mathfrak{s},\Sigma)\models\pi_{\mathrm{right}}(\psi',\mathrm{J})$.
  Conversely, if $(\mathfrak{s},\Sigma)\models\pi_{\mathrm{right}}(\psi',\mathrm{J})$, then $\bigstar_{j\in\mathrm{J}}\mathcal{A}_j\models^{\mathrm{abst}}_{\mathcal{R}}\psi'$ by definition, hence we obtain $(\mathfrak{s},\biguplus_{j\in\mathrm{J}}\mathfrak{h}_j)\models_{\mathcal{R}}\psi'$ by the same reasoning as before.

- For $\psi' = \psi_1@\mathrm{U}$, using $(\mathfrak{s}, \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \varphi_j$ for all $j \in \mathrm{J}$, we have $\mathrm{alloc}(\biguplus_{j\in\mathrm{J}} \mathfrak{h}_j) = \biguplus_{j\in\mathrm{J}} \mathrm{alloc}(\mathfrak{h}_j) = \biguplus_{j\in\mathrm{J}} [\![\mathrm{T}_j]\!]_{(\mathfrak{s},\Sigma)} = [\![\bigsqcup_{j\in\mathrm{J}} \mathrm{T}_j]\!]_{(\mathfrak{s},\Sigma)}$. Therefore, $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi'$ if and only if $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_1$ and $\mathrm{alloc}(\biguplus_{j\in\mathrm{J}} \mathfrak{h}_j) = [\![\mathrm{U}]\!]_{(\mathfrak{s},\Sigma)}$ if and only if $(\mathfrak{s}, \Sigma) \models \pi_{\mathrm{right}}(\psi_1, \mathrm{J})$ and $(\mathfrak{s}, \Sigma) \models \bigsqcup_{j\in\mathrm{J}} \mathrm{T}_j \approx \mathrm{U}$ if and only if $(\mathfrak{s}, \Sigma) \models \pi_{\mathrm{right}}(\psi', \mathrm{J})$.
- For operators $\wedge$, $\vee$, and $\neg$, the equivalence follows immediately from the induction hypothesis. For instance, $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \neg\psi_1$ if and only if $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}} \mathfrak{h}_j, \Sigma) \not\models_{\mathcal{R}} \psi_1$ if and only if $(\mathfrak{s}, \Sigma) \not\models \pi_{\mathrm{right}}(\psi_1, \mathrm{J})$ if and only if $(\mathfrak{s}, \Sigma) \models_{\sim} \pi_{\mathrm{right}}(\psi_1, \mathrm{J})$.
- For $\psi' = \psi_1 \circledast \psi_2$, suppose $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_1 \circledast \psi_2$. By Lemma 21, there exists a decomposition $\mathrm{J} = \mathrm{J}_1 \uplus \mathrm{J}_2$ that verifies $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}_1} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_1$ and $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}_2} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_2$. Therefore, we obtain, $(\mathfrak{s}, \Sigma) \models \pi_{\mathrm{right}}(\psi_1, \mathrm{J}_1)$ and $(\mathfrak{s}, \Sigma) \models \pi_{\mathrm{right}}(\psi_2, \mathrm{J}_2)$ using the induction hypothesis, which gives the result.
  Conversely, if $(\mathfrak{s}, \Sigma) \models \pi_{\mathrm{right}}(\psi', \mathrm{J})$, then there exist sets $\mathrm{J}_1, \mathrm{J}_2$ such that $\mathrm{J} = \mathrm{J}_1 \uplus \mathrm{J}_2$ and $(\mathfrak{s}, \Sigma) \models \pi_{\mathrm{right}}(\psi_1, \mathrm{J}_1) \wedge \pi_{\mathrm{right}}(\psi_2, \mathrm{J}_2)$. By induction hypothesis $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}_1} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_1$ and $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}_2} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_2$, hence the result.
- For $\psi' = \psi_1 \star \psi_2$, suppose $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}_1\uplus\mathrm{J}_2} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_1 \star \psi_2$. Then, $\psi_1 \star \psi_2$ implies $\psi_1 \circledast \psi_2$ thus, with Lemma 21, there exists $\mathrm{J} = \mathrm{J}_1 \uplus \mathrm{J}_2$ such that $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}_1} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_1$ and $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}_2} \mathfrak{h}_j, \Sigma) \models_{\mathcal{R}} \psi_2$. Both sub-structures are combinable, ensuring $\varnothing = \mathrm{alloc}(\biguplus_{j\in\mathrm{J}_1} \mathfrak{h}_j) \cap \mathrm{alloc}(\biguplus_{j\in\mathrm{J}_2} \mathfrak{h}_j) = \bigcup_{j\in\mathrm{J}_1} \mathrm{alloc}(\mathfrak{h}_j) \cap \bigcup_{j\in\mathrm{J}_2} \mathrm{alloc}(\mathfrak{h}_j)$
  $= \bigcup_{j\in\mathrm{J}_1} [\![\mathrm{T}_j]\!]_{(\mathfrak{s},\Sigma)} \cap \bigcup_{j\in\mathrm{J}_2} [\![\mathrm{T}_j]\!]_{(\mathfrak{s},\Sigma)} = [\![\bigsqcup_{j\in\mathrm{J}_1} \mathrm{T}_j \sqcap \bigsqcup_{j\in\mathrm{J}_2} \mathrm{T}_j]\!]_{(\mathfrak{s},\Sigma)}$.
  The reciprocal works exactly like $\psi_1 \circledast \psi_2$.
- Finally when $\psi'$ is pure, $(\mathfrak{s}, \biguplus_{j\in\mathrm{J}} \mathfrak{h}_j, \Sigma) \models \psi'$ implies $\mathrm{J} = \varnothing$ as in the case $\psi' = \mathtt{emp}$. ◀

▶ **Example 24.** Lemma 23 crucially depends on the fact that all location variables on the entailment's right-hand side occur as roots on the left-hand side (modulo aliasing). Consider, for instance, the formulas $\varphi = p(x, y)@\mathrm{X}$ and $\psi = y.f \to () \star x.f \to (y)$, and the entailment problem $\varphi \models_{\mathcal{R}} \psi$, with the rules $p(u, v) \Leftarrow u.f \to (v) \star q(v)$ and $q(v) \Leftarrow v.f \to ()$. Let $\mathfrak{s}$ be any store such that $\mathfrak{s}(x) \neq \mathfrak{s}(y)$, let $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ with $\mathfrak{h}_1 = \{(\mathfrak{s}(x), f) \mapsto (\mathfrak{s}(y))\}$ and $\mathfrak{h}_2 = \{(\mathfrak{s}(y), f) \mapsto ()\}$, and let $\Sigma(\mathrm{X}) = \{\mathfrak{s}(x), \mathfrak{s}(y)\}$. It is clear that $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi \wedge \psi$. Observe that $\mathfrak{s}(y) \notin \mathfrak{s}(\mathrm{roots}(p(x, y)@\mathrm{X}, f))$, hence the hypothesis of the lemma is not fulfilled. The left-hand side contains a unique atom $p(x, y)$. By Item 2 in Definition 22, $\pi_{\mathrm{right}}(y.f \to (), \varnothing) = \pi_{\mathrm{right}}(x.f \to (y), \varnothing) = \mathrm{false}$, thus $\pi_{\mathrm{right}}(\psi, \{1\}) = \mathrm{false}$ (by Item 6, since $\{1\}$ cannot be decomposed into a disjoint union of two nonempty sets). Hence, the equivalence stated in the lemma does not hold.

In contrast, when the reduction in Section 3.2 is applied, the left-hand side is replaced by $\tilde{p}(x, y, \zeta)@\mathrm{Y} \circledast \zeta \approx y \circledast q(\zeta)@\mathrm{Z} \circledast \mathrm{X} \approx \mathrm{Y} \sqcup \mathrm{Z}$ with $\tilde{p}(u, v, \zeta) \Leftarrow u.f \to (v) \star \zeta \approx v$ and $\mathfrak{s}(\zeta) = \mathfrak{s}(y)$. The left-hand side now contains two predicate atoms $\tilde{\varphi}_1 = \tilde{p}(x, y, \zeta)$ and $\tilde{\varphi}_2 = q(\zeta)$, associated with terms $\mathrm{T}_1 = \mathrm{Y}$ and $\mathrm{T}_2 = \mathrm{Z}$, respectively. Let $\mathcal{A}_i$ be the abstractions of $(\mathfrak{s}, \mathfrak{h}_i)$ (these abstractions may be computed from the formulas $\tilde{\varphi}_i$ using the algorithm in [4]). By Definition 22 (6), the formula $\pi_{\mathrm{right}}(\psi, \{1, 2\})$ is:
$\bigvee_{\mathrm{J}_1\uplus\mathrm{J}_2=\{1,2\}} \big(\pi_{\mathrm{right}}(y.f \to (), \mathrm{J}_1) \wedge \pi_{\mathrm{right}}(x.f \to (y), \mathrm{J}_2) \wedge \bigsqcup_{i\in\mathrm{J}_1} \mathrm{T}_i \sqcap \bigsqcup_{i\in\mathrm{J}_2} \mathrm{T}_i \approx \varnothing\big)$.
We consider all possible decompositions: If $\mathrm{J}_1 = \varnothing$ or $\mathrm{J}_2 = \varnothing$, then $\pi_{\mathrm{right}}(y.f \to (), \mathrm{J}_1)$ or $\pi_{\mathrm{right}}(x.f \to (y), \mathrm{J}_2)$ is false as shown previously. If $\mathrm{J}_1 = \{1\}$ and $\mathrm{J}_2 = \{2\}$, then both $\pi_{\mathrm{right}}(y.f \to (), \mathrm{J}_1)$ and $\pi_{\mathrm{right}}(x.f \to (y), \mathrm{J}_2)$ are false (since $\mathcal{A}_1 \not\models_{\mathcal{R}} y.f \to ()$ and $\mathcal{A}_2 \not\models_{\mathcal{R}} x.f \to (y)$). Finally, if $\mathrm{J}_1 = \{2\}$ and $\mathrm{J}_2 = \{1\}$, then $\pi_{\mathrm{right}}(y.f \to (), \mathrm{J}_1) = \pi_{\mathrm{right}}(x.f \to (y), \mathrm{J}_2) = \mathrm{true}$ (since $\mathcal{A}_2 \models_{\mathcal{R}} y.f \to ()$ and $\mathcal{A}_1 \models_{\mathcal{R}} x.f \to (y)$). We get $\pi_{\mathrm{right}}(\psi, \{1, 2\}) = \mathrm{Z} \sqcap \mathrm{Y} \approx \varnothing$ and $(\mathfrak{s}, \Sigma) \models \pi_{\mathrm{right}}(\psi, \{1, 2\})$, as expected.

Now consider the formula $\psi' = y.f \to () \star x.g \to (y)$. By the same reasoning, we get $\pi_{\mathrm{right}}(\psi', \{1, 2\}) = \pi_{\mathrm{right}}(y.f \to (), \{2\}) \wedge \pi_{\mathrm{right}}(x.g \to (y), \{1\}) \wedge \mathrm{Z} \sqcap \mathrm{Y} \approx \varnothing$. As $\mathcal{A}_1.\mathit{field} = f \neq g$, $\pi_{\mathrm{right}}(x.g \to (y), \{1\}) = \mathrm{false}$, so that $\pi_{\mathrm{right}}(\psi', \{1, 2\}) = \mathrm{false}$.

Consider the formula $\psi'' = \psi@\mathrm{X}$. By Definition 22 (4), we get: $\pi_{\text{right}}(\psi'', \{1,2\}) = \pi_{\text{right}}(\psi, \{1,2\}) \wedge \mathrm{Y} \sqcup \mathrm{Z} \approx \mathrm{X} = \mathrm{Z} \sqcap \mathrm{Y} \approx \varnothing \wedge \mathrm{Y} \sqcup \mathrm{Z} \approx \mathrm{X}$.

Finally, let $r(x)$ be a predicate atom denoting a list ending at (), defined by the rules: $r(u) \Leftarrow u.f \to (v) \star r(v)$ and $r(u) \Leftarrow u.f \to ()$. Using the same structures $(\mathfrak{s}, \mathfrak{h}_i)$ and abstractions $\mathcal{A}_i$ (for $i \in \{1,2\}$) as above, we get (by Definition 22 (3)): $\pi_{\text{right}}(r(x), \{1,2\}) =$ true, since $\mathcal{A}_1 \star \mathcal{A}_2 \models_{\mathcal{R}} r(x)$ (the latter condition can be tested by computing the abstraction $\mathcal{A}_1 \star \mathcal{A}_2$ as explained in [4]).

## 5.3 Combining both Formulas

Now we combine the two previous results, over the left-hand side $\varphi$ and the right-hand side $\psi$ of the entailment, to obtain the following lemma.

▶ **Lemma 25.** *Let $\mathcal{R}$ be an SID with PCE 1-predicate rules, $\varphi = \pi \circledast \circledast_{i=1}^{n} \varphi_i$ be an $\circledast$-formula, and $\psi$ be an OSL-formula.*

*There exists a structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ such that $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi \wedge \neg\psi$ and $\mathfrak{s}(\mathrm{fv}^{loc}(\psi)) \subseteq \mathfrak{s}(roots(\varphi, f))$ for all fields $f \in \mathcal{F}$ if and only if there exist pairwise combinable abstractions $(\mathcal{A}_i)_{i \in [\![1,n]\!]}$ such that: (i) $\mathcal{A}_i \models_{\mathcal{R}}^{abst} \tilde{\varphi}_i$ for all $i \in [\![1,n]\!]$; (ii) for all $f \in \mathcal{F}$ and $x \in \mathrm{fv}^{loc}(\psi)$, there exists $y \in roots(\varphi, f)$ such that $(x,y) \in \mathcal{A}_1 \bullet \sim$; and (iii) $\pi_{left} \wedge \sim \pi_{right}(\psi, [\![1,n]\!])$ is satisfiable.*

The existence of a structure satisfying the previous conditions is thus decidable:

▶ **Lemma 26.** *Given an SID with PCE 1-predicate rules $\mathcal{R}$, an $\circledast$-formula $\varphi$ and a formula $\psi$, the problem that asks whether there exists a structure $(\mathfrak{s}, \mathfrak{h}, \Sigma)$ such that $(\mathfrak{s}, \mathfrak{h}, \Sigma) \models_{\mathcal{R}} \varphi \wedge \neg\psi$ and for all $f \in \mathcal{F}$, $\mathfrak{s}(\mathrm{fv}^{loc}(\psi)) \subseteq \mathfrak{s}(roots(\varphi, f))$ is in* 2-NEXPTIME.

**Proof.** The decision problem exactly matches the first part of the equivalence in Lemma 25, thus we need to decide whether there exist abstractions $(\mathcal{A}_i)_{i \in [\![1,n]\!]}$ which satisfy points $(i)$ to $(iii)$. Given $\varphi_i$ for $i \in [\![1,n]\!]$, there exists a finite number of abstractions $\mathcal{A}_i$ which may be associated with $\varphi_i$, and the size of these abstractions is simply exponential w.r.t. the size of $\mathcal{R}$, $\varphi_i$ and $\mathrm{V_G}$. We may thus guess one abstraction $\mathcal{A}_i$ for every $i \in [\![1,n]\!]$ and check that $\mathcal{A}_i \models_{\mathcal{R}}^{\text{abst}} \varphi_i$. As abstractions are essentially defined as the sets of formulas they satisfy, this test can be performed in polynomial time[7] w.r.t. $\mathcal{A}_i$. Then, one can check whether the abstractions are pairwise combinable and if point $(ii)$ holds with straightforward polynomial algorithms.

It is clear that the size of $\pi_{\text{left}}$ is polynomial w.r.t. $\varphi$, $(\mathcal{A}_i)_{i \in [\![1,n]\!]}$ and $(\mathcal{A}_i \bullet card)_{i \in [\![1,n]\!]}$. The formulas $\mathcal{A}_i \bullet card$ are computed (using the polynomial time algorithm in [18]) from a context-free grammar which is polynomial w.r.t. the number of abstractions (as the abstractions correspond to non terminals of the grammar). Consequently, $\pi_{\text{left}}$ is of size double-exponential w.r.t. the size of $\mathcal{R}$, $\varphi$ and $\psi$. Meanwhile, the formula $\pi_{\text{right}}(\psi, [\![1,n]\!])$ has a construction linear in $\psi$ except for the cases $\star$ and $\circledast$, where all decompositions of J are considered, yielding an exponential blow-up. This leads to a global BAPA formula $\pi_{\text{left}} \wedge \sim \pi_{\text{right}}(\psi, [\![1,n]\!])$ of size at most double-exponential in the size of $\psi$. Therefore, we can check the satisfiability of the global formula in nondeterministic double-exponential time using the algorithm for BAPA in [12].

As a consequence, this entailment problem is decidable in at most nondeterministic double-exponential time. ◀

Theorem 8 is established by combining the reduction of Section 3 with the previous lemma.

[7] In practice, instead of using a generate-and-test algorithm, the abstraction associated with each formula $\varphi_i$ should be computed using a fixed-point algorithm.

## 6 Conclusion

In this paper, we design a decision procedure for the entailment problem in SL extended to support overlaid data structures strictly more general than those described in [13, 7, 2]. The algorithm runs in nondeterministic double-exponential time. Two interesting problems remain open. The first problem concerns the optimality of the procedure. It follows from the result of [3] that the entailment problem is 2-Exptime-hard; however, it is unclear whether the 2-Nexptime upper bound is tight. The second problem concerns extending the logic to allow less rigid ways of associating sets of locations with heaps. Currently, these sets always denote the domain of the heap corresponding to sub-formulas, due to construct $\varphi$@T, which states that the domain of the heap corresponding to $\varphi$ is T. In [15], more general associations are possible. They are encoded in the inductive rules, where set variables appear as parameters of predicate symbols and may denote specific subsets of the heap computed inductively. For instance, the framework [15] allows the definition of predicates such as $\texttt{ls}(x, y, \mathrm{Z})$, denoting a list segment from $x$ to $y$, where Z contains all locations occurring at, say, odd positions. The satisfiability problem for the logic in [15] is in Nexptime by means of a reduction to BAPA. However, it is unclear whether our entailment procedure can be extended to cover such definitions, or even whether the entailment is decidable in this setting. A long-term target is identifying decidable fragments in those proposed in [8, 17], where not all fields are allocated, a property incompatible with the establishment property of PCE.

## References


1 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2004. doi:10.1007/978-3-540-30538-5_9.

2 Cezara Dragoi, Constantin Enea, and Mihaela Sighireanu. Local shape analysis for overlaid data structures. In Francesco Logozzo and Manuel Fähndrich, editors, *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*, volume 7935 of *Lecture Notes in Computer Science*, pages 150–171. Springer, 2013. doi:10.1007/978-3-642-38856-9_10.

3 Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Entailment checking in separation logic with inductive definitions is 2-exptime hard. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 191–211. EasyChair, 2020. doi:10.29007/F5WH.

4 Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Decidable entailments in separation logic with inductive definitions: Beyond establishment. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CSL.2021.20.

5 Marco Eilers, Malte Schwerhoff, and Peter Müller. Verification algorithms for automated separation logic verifiers. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 362–386. Springer, 2024. doi:10.1007/978-3-031-65627-9_18.

6 Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. Compositional entailment checking for a fragment of separation logic. In Jacques Garrigue, editor, *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*, volume 8858 of *Lecture Notes in Computer Science*, pages 314–333. Springer, 2014. doi:10.1007/978-3-319-12736-1_17.

**7** Constantin Enea, Vlad Saveluc, and Mihaela Sighireanu. Compositional invariant checking for overlaid and nested linked lists. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7792 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2013. doi:10.1007/978-3-642-37036-6_9.

**8** Aquinas Hobor and Jules Villard. The ramifications of sharing in data structures. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 523–536. ACM, 2013. doi:10.1145/2429069.2429131.

**9** Radu Iosif, Adam Rogalewicz, and Jiri Simacek. The Tree Width of Separation Logic with Recursive Definitions. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38, Berlin, Heidelberg, 2013. Springer. ISSN: 1611-3349. doi:10.1007/978-3-642-38574-2_2.

**10** Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2014. doi:10.1007/978-3-319-11936-6_15.

**11** Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, pages 14–26, New York, NY, USA, January 2001. Association for Computing Machinery. doi:10.1145/360204.375719.

**12** Viktor Kuncak, Huu Hai Nguyen, and Martin Rinard. Deciding Boolean Algebra with Presburger Arithmetic. *Journal of Automated Reasoning*, 36(3):213–239, April 2006. doi:10.1007/s10817-006-9042-1.

**13** Oukseh Lee, Hongseok Yang, and Rasmus Petersen. Program Analysis for Overlaid Data Structures. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 592–608, Berlin, Heidelberg, 2011. Springer. doi:10.1007/978-3-642-22110-1_48.

**14** Christoph Matheja, Jens Pagel, and Florian Zuleger. A Decision Procedure for Guarded Separation Logic Complete Entailment Checking for Separation Logic with Inductive Definitions. *ACM Transactions on Computational Logic*, 24(1):1 : 1–76, January 2023. doi:10.1145/3534927.

**15** Nicolas Peltier, Quentin Petitjean, and Mihaela Sighireanu. Deciding satisfiability for overlaid symbolic heaps. In René Thiemann and Christoph Weidenbach, editors, *Frontiers of Combining Systems - 15th International Symposium, FroCoS 2025, Reykjavik, Iceland, September 29 - October 1, 2025, Proceedings*, volume 15979 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2025. doi:10.1007/978-3-032-04167-8_5.

**16** J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74, Copenhagen, Denmark, July 2002. ISSN: 1043-6871. doi:10.1109/LICS.2002.1029817.

**17** Asankhaya Sharma, Aquinas Hobor, and Wei-Ngan Chin. Specifying compatible sharing in data structures. In *Proceedings of the 17th International Conference on Formal Methods and Software Engineering (ICFEM)*, Lecture Notes in Computer Science, pages 349–365. Springer, 2015. doi:10.1007/978-3-319-25423-4_23.

**18** Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the Complexity of Equational Horn Clauses. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 337–352, Berlin, Heidelberg, 2005. Springer. doi:10.1007/11532231_25.