Unifying Decidable Entailments in Separation Logic with Inductive Definitions

Mnacho Echenim, Radu Iosif, Nicolas Peltier

Unifying Decidable Entailments in Separation Logic with Inductive Definitions

Mnacho Echenim[1] [0000−0001−5765−0758], Radu Iosif[2] [0000−0003−3204−3294], and Nicolas Peltier[1] [0000−0002−8943−7000]

[1] Univ. Grenoble Alpes, CNRS, LIG, F-38000 Grenoble France
[2] Univ. Grenoble Alpes, CNRS, VERIMAG, F-38000 Grenoble France

Abstract. The entailment problem $\varphi \models \psi$ in Separation Logic [12,15], between separated conjunctions of equational ($x \approx y$ and $x \not\approx y$), spatial ($x \mapsto (y_1, \ldots, y_\kappa)$) and predicate ($p(x_1, \ldots, x_n)$) atoms, interpreted by a finite set of inductive rules, is undecidable in general. Certain restrictions on the set of inductive definitions lead to decidable classes of entailment problems. Currently, there are two such decidable classes, based on two restrictions, called *establishment* [10,13,14] and *restrictedness* [8], respectively. Both classes are shown to be in 2EXPTIME by the independent proofs from [14] and [8], respectively, and a many-one reduction of established to restricted entailment problems has been given [8]. In this paper, we strictly generalize the restricted class, by distinguishing the conditions that apply only to the left- (φ) and the right- (ψ) hand side of entailments, respectively. We provide a many-one reduction of this generalized class, called *safe*, to the established class. Together with the reduction of established to restricted entailment problems, this new reduction closes the loop and shows that the three classes of entailment problems (respectively established, restricted and safe) form a single, unified, 2EXPTIME-complete class.

1 Introduction

Separation Logic [12,15] (SL) was primarily introduced for writing concise Hoare logic proofs of programs that handle pointer-linked recursive data structures (lists, trees, etc). Over time, SL has evolved into a powerful logical framework, that constitutes the basis of several industrial-scale static program analyzers [3,2,5], that perform scalable compositional analyses, based on the principle of *local reasoning*: describing the behavior of a program statement with respect only to the small (local) set of memory locations that are changed by that statement, with no concern for the rest of the program's state.

Given a set of memory locations (e.g., addresses), SL formulæ describe *heaps*, that are finite partial functions mapping finitely many locations to records of locations. A location ℓ is *allocated* if it occurs in the domain of the heap. An atom $x \mapsto (y_1, \ldots, y_\kappa)$ states that there is only one allocated location, associated with x, that moreover refers to the tuple of locations associated with $(y_1, \ldots, y_\kappa)$, respectively. The *separating conjunction* $\phi * \psi$ states that the heap can split into two parts, with disjoint domains, that make ϕ and ψ true, respectively. The separating conjunction is instrumental in supporting local reasoning, because the disjointness between the (domains of the) models of its arguments ensures that no update of one heap can actually affect the other.

Reasoning about recursive data structures of unbounded sizes (lists, trees, etc.) is possible via the use of predicate symbols, whose interpretation is specified by a user-provided *set of inductive definitions* (SID) of the form $p(x_1,\ldots,x_n) \Leftarrow \pi$, where p is a predicate symbol of arity n and the free variables of the formula π are among the parameters $x_1,\ldots,x_n$ of the rule. Here the separating conjunction ensures that each unfolding of the rules, which substitute some predicate atom $p(y_1,\ldots,y_n)$ by a formula $\pi[x_1/y_1,\ldots,x_n/y_n]$, corresponds to a way of building the recursive data structure. For instance, a list is either empty, in which case its head equals its tail pointer, or is built by first allocating the head, followed by all elements up to but not including the tail, as stated by the inductive definitions $\mathsf{ls}(x,y) \Leftarrow x \approx y$ and $\mathsf{ls}(x,y) \Leftarrow \exists z \,.\, x \mapsto (z) * \mathsf{ls}(z,y)$.

An important problem in program verification, arising during the construction of Hoare-style correctness proofs of programs, is the discharge of verification conditions of the form $\phi \models \psi$, where ϕ and ψ are SL formulæ, asking whether every model of ϕ is also a model of ψ. These problems, called *entailments*, are, in general, undecidable in the presence of inductively defined predicates [11,1].

A first decidable class of entailments, described in [10], involves three restrictions on the SID rules: *progress*, *connectivity* and *establishment*. Intuitively, the progress (P) condition states that every rule allocates exactly one location, the connectivity (C) condition states that the set of allocated locations has a tree-shaped structure, and the establishment (E) condition states that every existentially quantified variable from a rule defining a predicate is (eventually) allocated in every unfolding of that predicate. A 2EXPTIME algorithm was proposed for testing the validity of PCE entailments [13,14] and a matching 2EXPTIME-hardness lower bound was provided shortly after [6].

Later work relaxes the establishment condition, necessary for decidability [7], by proving that the entailment problem is still in 2EXPTIME if the establishment condition is replaced by the *restrictedness* (R) condition, which requires that every disequality ($x \not\approx y$) involves at least one free variable from the left-hand side of the entailment, propagated through the unfoldings of the inductive system [8]. Interestingly, the rules of a progressive, connected and restricted (PCR) entailment may generate data structures with "dangling" (i.e. existentially quantified but not allocated) pointers, which was not possible with PCE entailments.

In this paper, we generalize PCR entailments further, by showing that the connectivity and restrictedness conditions are needed only on the right-hand side of the entailment, whereas the only condition required on the left-hand side is progress (which can usually be enforced by folding or unfolding definitions). Our results thus allow for "asymetric" entailments, i.e., one can test whether the structures described by inductive rules that are (almost) arbitrary fulfill some restricted formula. Although the class of data structures that can be described is much larger, we show that this new class of entailments, called *safe*, is also 2EXPTIME-complete, by a many-one reduction of the validity of safe entailments to the validity of PCE entailments. A second contribution of the paper is the cross-certification of the two independent proofs of the 2EXPTIME upper bounds, for the PCE [6,14,8] and PCR [8] classes of entailments, respectively, by closing the loop. Namely, the reduction given in this paper enables the translation of any of the three entailment problems into an equivalent problem in any other class, while preserving the 2EXPTIME upper bound. This is because all the reductions are

polynomial in the overall size of the SID and singly-exponential in the maximum size of the rules in the SID. The theoretical interest of the reduction is that it makes the proof of decidability and of the complexity class much shorter and clearer. It also has some practical advantages, since it allows one to re-use existing implementations designed for established systems instead of having to develop entirely new automated reasoning systems. Due to space restrictions, some of the proofs are omitted. All proofs can be found in [9].

2 Definitions

For a (partial) function $f : A \to B$, we denote by $\mathrm{dom}(f)$ and $\mathrm{rng}(f)$ its domain and range, respectively. For a relation $R \subseteq A \times A$, we denote by R^* the reflexive and transitive closure of R.

Let κ be a fixed natural number throughout this paper and let P be a countably infinite set of *predicate symbols*. Each predicate symbol $p \in \mathsf{P}$ is associated a unique arity, denoted $ar(p)$. Let V be a countably infinite set of *variables*. For technical convenience, we also consider a special constant $\bot$, which will be used to denote "empty" record fields. Formulæ are built inductively, according to the following syntax:

$$\phi := x \not\approx x' \mid x \approx x' \mid x \mapsto (y_1, \ldots, y_\kappa) \mid p(x_1, \ldots, x_n) \mid \phi_1 * \phi_2 \mid \phi_1 \vee \phi_2 \mid \exists x \,.\, \phi_1$$

where $p \in \mathsf{P}$ is a predicate symbol of arity $n = ar(p)$, $x, x', x_1, \ldots, x_n \in \mathsf{V}$ are variables and $y_1, \ldots, y_\kappa \in \mathsf{V} \cup \{\bot\}$ are *terms*, i.e. either variables or $\bot$.

The set of variables freely occurring in a formula ϕ is denoted by $\mathrm{fv}(\phi)$, we assume by α-equivalence that the same variable cannot occur both free and bound in the same formula ϕ, and that distinct quantifiers bind distinct variables. The *size* $|\phi|$ of a formula ϕ is the number of occurrences of symbols in ϕ. A formula $x \approx x'$ or $x \not\approx x'$ is an *equational atom*, $x \mapsto (y_1, \ldots, y_\kappa)$ is a *points-to atom*, whereas $p(x_1, \ldots, x_n)$ is a *predicate atom*. Note that $\bot$ cannot occur in an equational or in a predicate atom. A formula is *predicate-less* if no predicate atom occurs in it. A *symbolic heap* is a formula of the form $\exists \boldsymbol{x} \,.\, *_{j=1}^{m} \alpha_i$, where each α_i is an atom and $\boldsymbol{x}$ is a possibly empty vector of variables.

Definition 1. *A variable x is* allocated by a symbolic heap ϕ *iff ϕ contains a sequence of equalities $x_1 \approx x_2 \approx \ldots \approx x_{n-1} \approx x_n$, for $n \geq 1$, such that $x = x_1$ and $x_n \mapsto (y_1, \ldots, y_\kappa)$ occurs in ϕ, for some variables $x_1, \ldots, x_n$ and some terms $y_1, \ldots, y_\kappa \in \mathsf{V} \cup \{\bot\}$.*

A *substitution* is a partial function mapping variables to variables. If σ is a substitution and ϕ is a formula, a variable or a tuple, then $\phi\sigma$ denotes the formula, the variable or the tuple obtained from ϕ by replacing every free occurrence of a variable $x \in \mathrm{dom}(\sigma)$ by $\sigma(x)$, respectively. We denote by $\{\langle x_i, y_i \rangle \mid i \in [\![1, n]\!]\}$ the substitution with domain $\{x_1, \ldots, x_n\}$ that maps x_i to y_i, for each $i \in [\![1, n]\!]$.

A *set of inductive definitions* (SID) $\mathcal{R}$ is a finite set of implications (or rules) of the form $p(x_1, \ldots, x_n) \Leftarrow \pi$, where $p \in \mathsf{P}$, $n = ar(p)$, $x_1, \ldots, x_n$ are pairwise distinct variables and π is a quantifier-free symbolic heap. The predicate atom $p(x_1, \ldots, x_n)$ is the *head* of the rule and $\mathcal{R}(p)$ denotes the subset of $\mathcal{R}$ consisting of rules with head $p(x_1, \ldots, x_n)$ (the choice of $x_1, \ldots, x_n$ is not important). The variables in $\mathrm{fv}(\pi) \setminus \{x_1, \ldots, x_n\}$ are called

the *existential variables of the rule*. Note that, by definition, these variables are not explicitly quantified inside π and that π is quantifier-free. For simplicity, we denote by $p(x_1,\ldots,x_n) \Leftarrow_{\mathcal{R}} \pi$ the fact that the rule $p(x_1,\ldots,x_n) \Leftarrow \pi$ belongs to $\mathcal{R}$. The *size* of $\mathcal{R}$ is defined as $|\mathcal{R}| \stackrel{\text{def}}{=} \sum_{p(x_1,\ldots,x_n)\Leftarrow_{\mathcal{R}}\pi} |\pi| + n$ and its *width* as $\mathrm{w}(\mathcal{R}) \stackrel{\text{def}}{=} \max_{p(x_1,\ldots,x_n)\Leftarrow_{\mathcal{R}}\pi} |\pi| + n$.

We write $p \succeq_{\mathcal{R}} q$, $p,q \in \mathsf{P}$ iff $\mathcal{R}$ contains a rule of the form $p(x_1,\ldots,x_n) \Leftarrow \pi$, and q occurs in π. We say that p *depends on* q if $p \succeq^*_{\mathcal{R}} q$. For a formula ϕ, we denote by $\mathcal{P}(\phi)$ the set of predicate symbols q, such that $p \succeq^*_{\mathcal{R}} q$ for some predicate p occurring in ϕ.

Given formulæ ϕ and ψ, we write $\phi \Leftarrow_{\mathcal{R}} \psi$ if ψ is obtained from ϕ by replacing an atom $p(u_1,\ldots,u_n)$ by $\pi\{\langle x_1,u_1\rangle,\ldots,\langle x_n,u_n\rangle\}$, where $\mathcal{R}$ contains a rule $p(x_1,\ldots,x_n) \Leftarrow \pi$. We assume, by a renaming of existential variables, that the set $(\mathrm{fv}(\pi) \setminus \{x_1,\ldots,x_n\}) \cap \mathrm{fv}(\phi)$ is empty. We call ψ an *unfolding* of ϕ iff $\phi \Leftarrow^*_{\mathcal{R}} \psi$.

We now define the semantics of SL. Let $\mathcal{L}$ be a countably infinite set of *locations* containing, in particular, a special location $\perp\!\!\!\perp$. A *structure* is a pair $(\mathfrak{s},\mathfrak{h})$, where:

- $\mathfrak{s}$ is a partial function from $\mathsf{V} \cup \{\bot\}$ to $\mathcal{L}$, called a *store*, such that $\bot \in \mathrm{dom}(\mathfrak{s})$ and $\mathfrak{s}(x) = \perp\!\!\!\perp \iff x = \bot$, for all $x \in \mathsf{V} \cup \{\bot\}$, and
- $\mathfrak{h} : \mathcal{L} \to \mathcal{L}^\kappa$ is a finite partial function, such that $\perp\!\!\!\perp \notin \mathrm{dom}(\mathfrak{h})$.

If $x_1,\ldots,x_n$ are pairwise distinct variables and $\ell_1,\ldots,\ell_n \in \mathcal{L}$ are locations, we denote by $\mathfrak{s}[x_i \leftarrow \ell_i \mid 1 \leq i \leq n]$ the store $\mathfrak{s}'$ defined by $\mathrm{dom}(\mathfrak{s}') = \mathrm{dom}(\mathfrak{s}) \cup \{x_1,\ldots,x_n\}$, $\mathfrak{s}'(y) = \ell_i$ if $y = x_i$ for some $i \in [\![1,n]\!]$, and $\mathfrak{s}'(y) = \mathfrak{s}(x)$ otherwise. If $x_1,\ldots,x_n \notin \mathrm{dom}(\mathfrak{s})$, then the store $\mathfrak{s}'$ is called an *extension* of $\mathfrak{s}$ to $\{x_1,\ldots,x_n\}$.

Given a heap $\mathfrak{h}$, we define $\mathrm{ref}(\mathfrak{h}) \stackrel{\text{def}}{=} \bigcup_{l \in \mathrm{dom}(\mathfrak{h})} \{\ell_i \mid \mathfrak{h}(\ell) = (\ell_1,\ldots,\ell_\kappa), i \in [\![1,\kappa]\!]\}$ and $\mathrm{loc}(\mathfrak{h}) \stackrel{\text{def}}{=} \mathrm{dom}(\mathfrak{h}) \cup \mathrm{ref}(\mathfrak{h})$. Two heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are *disjoint* iff $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \emptyset$, in which case $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ denotes the union of $\mathfrak{h}_1$ and $\mathfrak{h}_2$, undefined whenever $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are not disjoint.

Given an SID $\mathcal{R}$, $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \phi$ is the least relation between structures and formulæ such that whenever $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \phi$, we have $\mathrm{fv}(\phi) \subseteq \mathrm{dom}(\mathfrak{s})$ and the following hold:

$(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} x \approx x'$	if $\mathrm{dom}(\mathfrak{h}) = \emptyset$ and $\mathfrak{s}(x) = \mathfrak{s}(x')$
$(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} x \not\approx x'$	if $\mathrm{dom}(\mathfrak{h}) = \emptyset$ and $\mathfrak{s}(x) \neq \mathfrak{s}(x')$
$(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} x \mapsto (y_1,\ldots,y_\kappa)$	if $\mathrm{dom}(\mathfrak{h}) = \{\mathfrak{s}(x)\}$ and $\mathfrak{h}(\mathfrak{s}(x)) = \langle \mathfrak{s}(y_1),\ldots,\mathfrak{s}(y_\kappa)\rangle$
$(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \phi_1 * \phi_2$	if there exist disjoint heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ and $(\mathfrak{s},\mathfrak{h}_i) \models_{\mathcal{R}} \phi_i$, for both $i = 1,2$
$(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \phi_1 \vee \phi_2$	if $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \phi_i$, for some $i = 1,2$
$(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \exists x \,.\, \phi$	if there exists $\ell \in \mathcal{L}$ such that $(\mathfrak{s}[x \leftarrow \ell],\mathfrak{h}) \models \phi$
$(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} p(x_1,\ldots,x_n)$	if $p(x_1,\ldots,x_n) \Leftarrow_{\mathcal{R}} \phi$, and there exists a store $\mathfrak{s}_e$ coinciding with $\mathfrak{s}$ on $\{x_1,\ldots,x_n\}$, such that $(\mathfrak{s}_e,\mathfrak{h}) \models \phi$

Given formulæ ϕ and ψ, we write $\phi \models_{\mathcal{R}} \psi$ whenever $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \phi \Rightarrow (\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \psi$, for all structures $(\mathfrak{s},\mathfrak{h})$ and $\phi \equiv_{\mathcal{R}} \psi$ for ($\phi \models_{\mathcal{R}} \psi$ and $\psi \models_{\mathcal{R}} \phi$). We omit the subscript $\mathcal{R}$ whenever these relations hold for any SID. It is easy to check that, for all formulæ ϕ_1,ϕ_2,ψ, it is the case that $(\phi_1 \vee \phi_2) * \psi \equiv (\phi_1 * \psi) \vee (\phi_2 * \psi)$ and $(\exists x.\phi_1) * \phi_2 \equiv \exists x \,.\, \phi_1 * \phi_2$. Consequently, each formula can be transformed into an equivalent finite disjunction of symbolic heaps.

Definition 2. *An* entailment problem *is a triple* $\mathfrak{P} \stackrel{\text{def}}{=} \phi \vdash_{\mathcal{R}} \psi$*, where* ϕ *is a quantifier-free formula,* ψ *is a formula and* $\mathcal{R}$ *is an SID. The problem* $\mathfrak{P}$ *is* valid *iff* $\phi \models_{\mathcal{R}} \psi$*. The* size *of the problem* $\mathfrak{P}$ *is defined as* $|\mathfrak{P}| \stackrel{\text{def}}{=} |\phi| + |\psi| + |\mathcal{R}|$ *and its* width *is defined as* $\mathrm{w}(\mathfrak{P}) \stackrel{\text{def}}{=} \max(|\phi|, |\psi|, \mathrm{w}(\mathcal{R}))$.

Note that considering ϕ to be quantifier-free loses no generality, because $\exists x.\phi \models_{\mathcal{R}} \psi \iff \phi \models_{\mathcal{R}} \psi$.

3 Decidable Entailment Problems

The class of general entailment problems is undecidable, see Theorem 5 below for a refinement of the initial undecidability proofs [11,1]. A first attempt to define a natural decidable class of entailment problems is described in [10] and involves three restrictions on the SID rules, formally defined below:

Definition 3. *A rule* $p(x_1, \ldots, x_n) \Leftarrow \pi$ *is:*

1. progressing *(P) iff* $\pi = x_1 \mapsto (y_1, \ldots, y_\kappa) * \rho$ *and* ρ *contains no points-to atoms,*
2. connected *(C) iff it is progressing,* $\pi = x_1 \mapsto (y_1, \ldots, y_\kappa) * \rho$ *and every predicate atom in* ρ *is of the form* $q(y_i, \boldsymbol{u})$*, for some* $i \in [\![1, \kappa]\!]$,
3. established *(E) iff every existential variable* $x \in \mathrm{fv}(\pi) \setminus \{x_1, \ldots, x_n\}$ *is allocated by every predicate-less unfolding* $\pi \Leftarrow^*_{\mathcal{R}} \phi$.

An SID $\mathcal{R}$ *is* P *(resp.* C, E*)* for a formula ϕ *iff every rule in* $\bigcup_{p \in \mathcal{P}(\phi)} \mathcal{R}(p)$ *is P (resp. C,E). An entailment problem* $\phi \vdash_{\mathcal{R}} \psi$ *is* left- *(resp.* right-*)* P *(resp.* C, E*) iff* $\mathcal{R}$ *is P (resp. C, E) for* ϕ *(resp.* ψ*). An entailment problem is* P *(resp.* C, E*) iff it is both left- and right-P (resp. C, E).*

The decidability of progressing, connected and left-established entailment problems is an immediate consequence of the result of [10]. Moreover, an analysis of the proof [10] leads to an elementary recursive complexity upper bound, which has been recently tighten down to 2EXPTIME-complete [14,8,6]. In the following, we refer to Table 1 for a recap of the complexity results for the entailment problem. The last line is the main result of the paper and corresponds to the most general (known) decidable class of entailment problems (Definition 8).

Table 1. Decidability and Complexity Results for the Entailment Problem (✓ means that the corresponding condition holds on the left- and right-hand side of the entailment)

Reference	Progress	Connected	Established	Restricted	Complexity
Theorem 4	✓	✓	left	-	2EXP-co.
Theorem 5	✓	left	✓	-	undec.
[7, Theorem 6]	✓	✓	-	-	undec.
[8, Theorem 32]	✓	✓	-	✓	2EXP-co.
Theorem 31	✓	right	-	right	2EXP-co.

The following theorem is an easy consequence of previous results [6].

Theorem 4. *The progressing, connected and left-established entailment problem is* 2*EXPTIME-complete. Moreover, there exists a decision procedure that runs in time* $2^{2^{O(w(\mathfrak{P})^8 \cdot \log|\mathfrak{P}|)}}$ *for every instance* $\mathfrak{P}$ *of this problem.*

A natural question arises in this context: which of the restrictions from the above theorem can be relaxed and what is the price, in terms of computational complexity, of relaxing (some of) them? In the light of Theorem 5 below, the connectivity restriction cannot be completely dropped. Further, if we drop the establishment condition, the problem becomes undecidable [7, Theorem 6], even if both the left/right progress and connectivity conditions apply.

Theorem 5. *The progressing, left-connected and established entailment problem is undecidable.*

The second decidable class of entailment problems [8] relaxes the connectivity condition and replaces the establishment with a syntactic condition (that can be checked in polynomial time in the size of the SID), while remaining 2EXPTIME-complete. Informally, the definition forbids (dis)equations between existential variables in symbolic heaps or rules: the only allowed (dis)equations are of the form $x \bowtie y$ where x is a free variable (viewed as a constant in [8]). The definition given below is essentially equivalent to that of [8], but avoids any reference to constants; instead it uses a notion of $\mathcal{R}$-positional functions, which helps to identify existential variables that are always replaced by a free variable from the initial formula during unfolding.

An *$\mathcal{R}$-positional function* maps every n-ary predicate symbol p occurring in $\mathcal{R}$ to a subset of $[\![1,n]\!]$. Given an $\mathcal{R}$-positional function λ and a formula ϕ, we denote by $\mathsf{V}_\lambda(\phi)$ the set of variables x_i such that ϕ contains a predicate atom $p(x_1,\ldots,x_n)$ with $i \in \lambda(p)$. Note that V_λ is stable under substitutions, i.e. $\mathsf{V}_\lambda(\phi\sigma) = (\mathsf{V}_\lambda(\phi))\sigma$, for each formula ϕ and each substitution σ.

Definition 6. *Let* ψ *be a formula and* $\mathcal{R}$ *be an SID. The* fv-profile *of the pair* $(\psi,\mathcal{R})$ *is the* $\mathcal{R}$*-positional function* λ *such that the sets* $\lambda(p)$*, for* $p \in \mathsf{P}$*, are the maximal sets satisfying the following conditions:*

1. $\mathsf{V}_\lambda(\psi) \subseteq \mathrm{fv}(\psi)$.
2. *For all predicate symbols* $p \in \mathcal{P}(\psi)$*, all rules* $p(x_1,\ldots,x_n) \Leftarrow \pi$ *in* $\mathcal{R}$*, all predicate atoms* $q(y_1,\ldots,y_m)$ *in* π *and all* $i \in \lambda(q)$*, there exists* $j \in \lambda(p)$ *such that* $x_j = y_i$*.*

The fv-profile of $(\psi,\mathcal{R})$ *is denoted by* $\lambda^{\psi}_{\mathcal{R}}$.

Intuitively, given a predicate $p \in \mathsf{P}$, the set $\lambda^{\psi}_{\mathcal{R}}(p)$ denotes the formal parameters of p that, in every unfolding of ψ, will always be substituted by variables occurring freely in ψ. It is easy to check that $\lambda^{\psi}_{\mathcal{R}}$ can be computed in polynomial time w.r.t. $|\psi| + |\mathcal{R}|$, using a straightforward greatest fixpoint algorithm. The algorithm starts with a function mapping every predicate p of arity n to $[\![1,n]\!]$ and repeatedly removes elements from the sets $\lambda(p)$ to ensure that the above conditions hold. In the worst case, we may have eventually $\lambda(p) = \emptyset$ for all predicate symbols p.

Definition 7. *Let* λ *be an* $\mathcal{R}$*-positional function, and* V *be a set of variables. A formula* ϕ *is* λ-restricted *(*λ*-R) w.r.t.* V *iff the following hold:*

1. *for every disequation $y \not\approx z$ in ϕ, we have $\{y,z\} \cap V \neq \emptyset$, and*
2. $\mathsf{V}_\lambda(\phi) \subseteq V$.

A rule $p(x_1,\ldots,x_n) \Leftarrow x \mapsto (y_1,\ldots,y_\kappa) * \rho$ is:

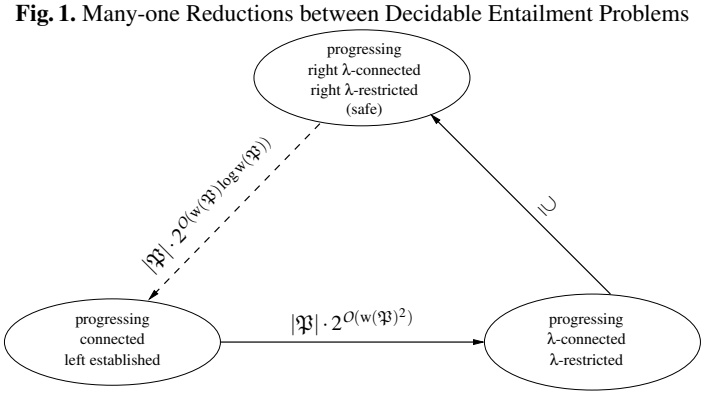
- λ-connected *(λ-C) iff for every atom $q(z_1,\ldots,z_m)$ occurring in ρ, we have $z_1 \in \mathsf{V}_\lambda(p(x_1,\ldots,x_n)) \cup \{y_1,\ldots,y_\kappa\}$,*
- λ-restricted *(λ-R) iff ρ is λ-restricted w.r.t. $\mathsf{V}_\lambda(p(x_1,\ldots,x_n))$.*

An SID $\mathcal{R}$ is P *(resp.* λ-C, λ-R*)* for a formula ϕ *iff every rule in $\bigcup_{p \in \mathcal{P}(\phi)} \mathcal{R}(p)$ is P (resp. λ-C, λ-R).*

An SID $\mathcal{R}$ is λ-C *(λ-R)* for a formula ϕ *iff every rule in $\bigcup_{p \in \mathcal{P}(\phi)} \mathcal{R}(p)$ is λ-C (λ-R).*
An entailment problem $\phi \vdash_\mathcal{R} \psi$ is left- *(*right-*)* λ-C, *(λ-R) iff $\mathcal{R}$ is λ-C (λ-R) for ϕ (ψ), where λ is considered to be $\lambda^\phi_\mathcal{R}$ ($\lambda^\psi_\mathcal{R}$). An entailment problem is* λ-C (λ-R*) iff it is both left- and right-λ-C (λ-R).*

The class of progressing, λ-connected and λ-restricted entailment problems has been shown to be a generalization of the class of progressing, connected and left-established problems, because the latter can be reduced to the former by a many-one reduction [8, Theorem 13] that runs in time $|\mathfrak{P}| \cdot 2^{O(\mathrm{w}(\mathfrak{P})^2)}$ on input $\mathfrak{P}$ (Figure 1) and preserves the problem's width asymptotically.

Fig. 1. Many-one Reductions between Decidable Entailment Problems



In the rest of this paper we close the loop by defining a syntactic extension of λ-progressing, λ-connected and λ-restricted entailment problems and by showing that this extension can be reduced to the class of progressing, connected and left-established entailment problems by a many-one reduction. The new fragment is defined as follows:

Definition 8. *An entailment problem $\phi \vdash_\mathcal{R} \psi$ is* safe *if, for $\lambda \stackrel{\mathsf{def}}{=} \lambda^\psi_\mathcal{R}$, the following hold:*

1. *every rule in $\mathcal{R}$ is progressing,*
2. *ψ is λ-restricted w.r.t.* $\mathrm{fv}(\phi)$,
3. *all the rules from $\bigcup_{p \in \mathcal{P}(\psi)} \mathcal{R}(p)$ are λ-connected and λ-restricted.*

Note that there is no condition on the formula ϕ, or on the rules defining the predicates occurring only in ϕ, other than the progress condition. The conditions in Definition 8 ensure that all the disequations occurring in any unfolding of ψ involve at least one

variable that is free in ϕ. Further, the heaps of the model of ψ must be *forests*, i.e. unions of trees, the roots of which are associated with the first argument of the predicate atoms in ψ or to free variables from ϕ.

A typical yet very simple example of such an entailment is the so-called "reversed list" problem that consists in checking that any list segment $\mathtt{revls}(z,y)$ defined in the reverse direction (from the tail to the head) is a list segment $\mathtt{ls}(x,y)$ in the usual sense (defined inductively from head to tail). This corresponds to the entailment problem $\mathtt{revls}(z,y) \vdash_{\mathcal{R}} \exists x.\mathtt{ls}(x,y)$ where $\mathcal{R}$ contains the following rules:

$$\begin{array}{ll} \mathtt{ls}(x,y) \Leftarrow x \mapsto (y) & \mathtt{revls}(z,y) \Leftarrow z \mapsto (y) \\ \mathtt{ls}(x,y) \Leftarrow x \mapsto (z) * \mathtt{ls}(z,y) & \mathtt{revls}(z,y) \Leftarrow z \mapsto (y) * \mathtt{revls}(u,z) \end{array}$$

This problem is considered as challenging for proof search-based automated reasoning procedures (see, e.g., [4,16]). The antecedent does not fulfill the connectivity condition, but the subsequent does, hence the entailment is safe. Similar, more complex examples can be defined, for instance a list can be constructed by interleaving elements at odd or even positions. Another example is the case of a data structure containing an unbounded number of acylic lists (e.g., a list of acyclic lists). Such a data structure does not fulfill the restricteness condition, since one needs to compare the pointers occurring along each list to the point at the end. Checking, for instance, that the concatenation of two lists of acyclic lists is again a list of (possibly cyclic) lists is a problem that fits into the safe class and can thus be effectively checked by our algorithm.

We refer the reader to Figure 1 for a general picture of the entailment problems considered so far and of the many-one reductions between them, where the reduction corresponding to the dashed arrow is the concern of the next section. Importantly, since all reductions are many-one, taking time polynomial in the size and exponential in the width of the input problem, while preserving its width asymptotically, the three classes from Figure 1 can be unified into a single (2EXPTIME-complete) class of entailments.

4 Reducing Safe to Established Entailments

In a model of a safe SID (Definition 8), the existential variables introduced by the replacement of predicate atoms with corresponding rule bodies are not required to be allocated. This is because safe SIDs are more liberal than established SIDs and allow heap structures with an unbounded number of dangling pointers. As observed in [8], checking the validity of an entailment (w.r.t a restricted SID) can be done by considering only those structures in which the dangling pointers point to pairwise distinct locations. The main idea of the hereby reduction of safe to established entailment problems is that any such structure can be extended by allocating all dangling pointers separately and, moreover, the extended structures can be defined by an established SID.

In what follows, we fix an arbitrary instance $\mathfrak{P} = \phi \vdash_{\mathcal{R}} \psi$ of the safe entailment problem (Definition 8) and denote by $\lambda \stackrel{\text{def}}{=} \lambda^{\psi}_{\mathcal{R}}$ the fv-profile of $(\psi, \mathcal{R})$ (Definition 6). Let $\boldsymbol{w} \stackrel{\text{def}}{=} (w_1, \ldots, w_\nu)$ be the vector of free variables from ϕ and ψ, where the order of variables is not important and assume w.l.o.g. that $\nu > 0$. Let $\mathcal{P}_l \stackrel{\text{def}}{=} \mathcal{P}(\phi)$ and

$\mathcal{P}_r \stackrel{\text{def}}{=} \mathcal{P}(\psi)$ be the sets of predicate symbols that depend on the predicate symbols occurring in the left- and right-hand side of the entailment, respectively. We assume that ϕ and ψ contain no points-to atoms and that $\mathcal{P}_l \cap \mathcal{P}_r = \emptyset$. Again, these assumptions lose no generality, because a points-to atom $u \mapsto (v_1, \ldots, v_\kappa)$ can be replaced by a predicate atom $p(u, v_1, \ldots, v_\kappa)$, where p is a fresh predicate symbol associated with the rule $p(x, y_1, \ldots, y_\kappa) \Leftarrow x \mapsto (y_1, \ldots, y_\kappa)$. Moreover the condition $\mathcal{P}_l \cap \mathcal{P}_r \neq \emptyset$ may be enforced by considering two copies of each predicate, for the left-hand side and for the right-hand side, respectively. Finally, we assume that every rule contains exactly μ existential variables, for some fixed $\mu \in \mathbb{N}$; this condition can be enforced by adding dummy literals $x \approx x$ if needed.

We describe a reduction of $\mathfrak{P}$ to an equivalent progressing, connected, and left-established entailment problem. The reduction will extend heaps, by adding $\nu + \mu$ record fields. We shall therefore often consider heaps and points-to atoms having $\kappa + \nu + \mu$ record fields, where the formal definitions are similar to those given previously. Usually such formulæ and heaps will be written with a prime. These additional record fields will be used to ensure that the constructed system is connected, by adding all the existential variables of a given rule (as well as the variables in $w_1, \ldots, w_\nu$) into the image of the location allocated by the considered rule. Furthermore, the left-establishment condition will be enforced by adding predicates and rules in order to allocate all the locations that correspond to existential quantifiers and that are not already allocated, making such locations point to a dummy vector $\bot \stackrel{\text{def}}{=} (\bot, \ldots, \bot)$, of length $\kappa + \nu + \mu$, where $\bot$ is the special constant denoting empty heap entries. To this aim, we shall use a predicate symbol $\underline{\bot}$ associated with the rule $\underline{\bot}(x) \Leftarrow x \mapsto \bot$. Note that allocating all these locations will entail (by definition of the separating conjunction) that they are distinct, thus the addition of such predicates and rules will reduce the number of satisfiable unfoldings. However, due to the restrictions on the use of disequations[3], we shall see that this does not change the status of the entailment problem.

Definition 9. *For any total function $\gamma : \mathcal{L} \to \mathcal{L}$ and any tuple $\boldsymbol{\ell} = \langle \ell_1, \ldots, \ell_n \rangle \in \mathcal{L}^n$, we denote by $\gamma(\boldsymbol{\ell})$ the tuple $\langle \gamma(\ell_1), \ldots, \gamma(\ell_n) \rangle$. If $\mathfrak{s}$ is a store, then $\gamma(\mathfrak{s})$ denotes the store with domain $\mathrm{dom}(\mathfrak{s})$, such that $\gamma(\mathfrak{s})(x) \stackrel{\text{def}}{=} \gamma(\mathfrak{s}(x))$, for all $x \in \mathrm{dom}(\mathfrak{s})$. Consider a heap $\mathfrak{h}$ such that for all $\ell \neq \ell' \in \mathrm{dom}(\mathfrak{h})$, we have $\gamma(\ell) \neq \gamma(\ell')$. Then $\gamma(\mathfrak{h})$ denotes the heap with domain $\mathrm{dom}(\gamma(\mathfrak{h})) = \{\gamma(\ell) \mid \ell \in \mathrm{dom}(\mathfrak{h})\}$, such that $\gamma(\mathfrak{h})(\gamma(\ell)) \stackrel{\text{def}}{=} \gamma(\mathfrak{h}(\ell))$, for all $\ell \in \mathrm{dom}(\mathfrak{h})$.*

The following lemma identifies conditions ensuring that the application of a mapping to a structure (Definition 9) preserves the truth value of a formula.

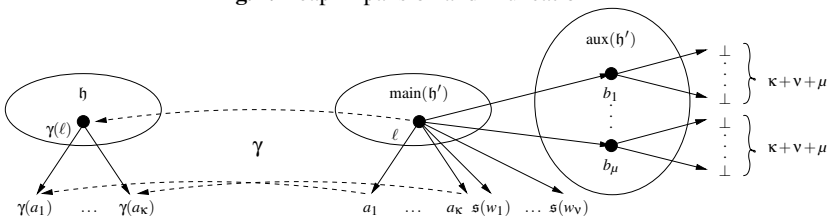
Lemma 10. *Given a set of variables V, let α be a formula that is λ-restricted w.r.t. V, such that $\mathcal{P}(\alpha) \subseteq \mathcal{P}_r$ and let $(\mathfrak{s}, \mathfrak{h})$ be an $\mathcal{R}$-model of α. For every mapping $\gamma : \mathcal{L} \to \mathcal{L}$ such that $\gamma(\ell) = \gamma(\ell') \Rightarrow \ell = \ell'$ holds whenever either $\{\ell, \ell'\} \subseteq \mathrm{dom}(\mathfrak{h})$ or $\{\ell, \ell'\} \cap \mathfrak{s}(V) \neq \emptyset$, we have $(\gamma(\mathfrak{s}), \gamma(\mathfrak{h})) \models_{\mathcal{R}} \alpha$.*

If γ is, moreover, injective, then the result of Lemma 10 holds for any formula:

Lemma 11. *Let α be a formula and let $(\mathfrak{s}, \mathfrak{h})$ be an $\mathcal{R}$-model of α. For every injective mapping $\gamma : \mathcal{L} \to \mathcal{L}$ we have $(\gamma(\mathfrak{s}), \gamma(\mathfrak{h})) \models_{\mathcal{R}} \alpha$.*

[3] Point (1) of Definition 7 in conjunction with point (2) of Definition 8.

Fig. 2. Heap Expansion and Truncation



4.1 Expansions and Truncations

We introduce a so-called *expansion* relation on structures, as well as a *truncation* operation on heaps. Intuitively, the expansion of a structure is a structure with the same store and whose heap is augmented with new allocated locations (each pointing to $\vec{\bot}$) and additional record fields, referring in particular to all the newly added allocated locations. These locations are introduced to accommodate all the existential variables of the predicate-less unfolding of the left-hand side of the entailment (to ensure that the obtained entailment is left-established). Conversely, the truncation of a heap is the heap obtained by removing these extra locations. We also introduce the notion of a γ-expansion which is a structure whose image by γ is an expansion.

We recall that, throughout this and the next sections, $\boldsymbol{w} = (w_1, \ldots, w_\nu)$ denotes the vector of free variables occurring in the problem, which is assumed to be fixed throughout this section and that $\{w_1, \ldots, w_\nu, \bot\} \subseteq \mathrm{dom}(\mathfrak{s})$, for every store $\mathfrak{s}$ considered here. Moreover, we assume w.l.o.g. that $w_1, \ldots, w_\nu$ do not occur in the considered SID $\mathcal{R}$ and denote by μ the number of existential variables in each rule of $\mathcal{R}$. We refer to Figure 2 for an illustration of the definition below:

Definition 12. *Let $\gamma : \mathcal{L} \to \mathcal{L}$ be a total mapping. A structure $(\mathfrak{s}, \mathfrak{h}')$ is a* γ-expansion *(or simply an* expansion *if $\gamma = id$) of some structure $(\mathfrak{s}, \mathfrak{h})$, denoted by $(\mathfrak{s}, \mathfrak{h}') \rhd_\gamma (\mathfrak{s}, \mathfrak{h})$, if $\mathfrak{h} : \mathcal{L} \to \mathcal{L}^\kappa$, $\mathfrak{h}' : \mathcal{L} \to \mathcal{L}^{\kappa+\mu+\nu}$ and there exist two disjoint heaps,* $\mathrm{main}(\mathfrak{h}')$ *and* $\mathrm{aux}(\mathfrak{h}')$*, such that $\mathfrak{h}' = \mathrm{main}(\mathfrak{h}') \uplus \mathrm{aux}(\mathfrak{h}')$ and the following hold:*

1. *for all $\ell_1, \ell_2 \in \mathrm{dom}(\mathrm{main}(\mathfrak{h}'))$, if $\gamma(\ell_1) = \gamma(\ell_2)$ then $\ell_1 = \ell_2$,*
2. *$\gamma(\mathrm{dom}(\mathrm{main}(\mathfrak{h}'))) = \mathrm{dom}(\mathfrak{h})$,*
3. *for each $\ell \in \mathrm{dom}(\mathrm{main}(\mathfrak{h}'))$, we have $\mathfrak{h}'(\ell) = \langle \boldsymbol{a}, \mathfrak{s}(\boldsymbol{w}), b_1^\ell, \ldots, b_\mu^\ell \rangle$, for some locations $b_1^\ell, \ldots, b_\mu^\ell \in \mathcal{L}$ and $\gamma(\boldsymbol{a}) = \mathfrak{h}(\gamma(\ell))$,*
4. *for each $\ell \in \mathrm{dom}(\mathrm{aux}(\mathfrak{h}'))$, we have $\mathfrak{h}'(\ell) = \perp\!\!\!\perp$ and there exists a location $\ell' \in \mathrm{dom}(\mathrm{main}(\mathfrak{h}'))$ such that $\mathrm{main}(\mathfrak{h}')(\ell')$ is of the form $\langle \boldsymbol{a}, \boldsymbol{\ell}, b_1^{\ell'}, \ldots, b_\mu^{\ell'} \rangle$ where $\vec{\ell}$ is a tuple of locations and $\ell = b_i^{\ell'}$, for some $i \in [\![1, \mu]\!]$. The element ℓ' is called the* connection of ℓ in $\mathfrak{h}'$ *and is denoted by $\mathrm{C}_{\mathfrak{h}'}(\ell)$.*[4]

Let $(\mathfrak{s}, \mathfrak{h}')$ be a γ-expansion of $(\mathfrak{s}, \mathfrak{h})$ and let $\ell \in \mathrm{dom}(\mathrm{main}(\mathfrak{h}'))$ be a location. Since $\nu > 0$ and for all $i \in [\![1, \nu]\!]$, $\mathfrak{s}(w_i)$ occurs in $\mathfrak{h}'(\ell)$, and since we assume that $\mathfrak{s}(w_i) \neq \perp\!\!\!\perp = \mathfrak{s}(\bot)$ for every $i \in [\![1, \nu]\!]$, necessarily $\mathrm{main}(\mathfrak{h}')(\ell) \neq \perp\!\!\!\perp$. This entails that the decomposition

[4] Note that ℓ' does not depend on γ, and if several such locations exist, then one is chosen arbitrarily.

$\mathfrak{h}' = \mathrm{main}(\mathfrak{h}') \uplus \mathrm{aux}(\mathfrak{h}')$ is unique: $\mathrm{main}(\mathfrak{h}')$ and $\mathrm{aux}(\mathfrak{h}')$ are the restrictions of $\mathfrak{h}'$ to the locations ℓ in $\mathrm{dom}(\mathfrak{h}')$ such that $\mathfrak{h}'(\ell) \neq \mathbb{1}$ and $\mathfrak{h}'(\ell) = \mathbb{1}$, respectively. In the following, we shall thus freely use the notations $\mathrm{aux}(\mathfrak{h}')$ and $\mathrm{main}(\mathfrak{h}')$, for arbitrary heaps $\mathfrak{h}'$.

Definition 13. *Given a heap $\mathfrak{h}'$, we denote by* $\mathrm{trunc}(\mathfrak{h}')$ *the heap* $\mathfrak{h}$ *defined as follows:* $\mathrm{dom}(\mathfrak{h}) \stackrel{\mathsf{def}}{=} \mathrm{dom}(\mathfrak{h}') \setminus \{\ell \in \mathrm{dom}(\mathfrak{h}') \mid \mathfrak{h}'(\ell) = \mathbb{1}\}$ *and for all* $\ell \in \mathrm{dom}(\mathfrak{h})$*, if* $\mathfrak{h}'(\ell) = (\ell_1, \ldots, \ell_{\kappa+\nu+\mu})$*, then* $\mathfrak{h}(\ell) \stackrel{\mathsf{def}}{=} (\ell_1, \ldots, \ell_\kappa)$.

Note that, if $\mathfrak{h} = \mathrm{trunc}(\mathfrak{h}')$ then $\mathfrak{h} : \mathcal{L} \to \mathcal{L}^\kappa$ and $\mathfrak{h}' : \mathcal{L} \to \mathcal{L}^{\kappa+\mu+\nu}$ are heaps of different out-degrees. In the following, we silently assume this fact, to avoid cluttering the notation by explicitly specifying the out-degree of a heap.

Example 14. Assume that $\mathcal{L} = \mathbb{N}$, $\nu = \mu = 1$. Let $\mathfrak{s}$ be a store such that $\mathfrak{s}(w_1) = 0$. We consider:

$$\begin{aligned}
\mathfrak{h} &\stackrel{\mathsf{def}}{=} \{\langle 1,2\rangle, \langle 2,2\rangle\},\\
\mathfrak{h}'_1 &\stackrel{\mathsf{def}}{=} \{\langle 1,(2,0,1)\rangle, \langle 2,(2,0,3)\rangle, \langle 3,(\bot,\bot,\bot)\rangle\},\\
\mathfrak{h}'_2 &\stackrel{\mathsf{def}}{=} \{\langle 1,(3,0,1)\rangle, \langle 2,(4,0,3)\rangle, \langle 3,(\bot,\bot,\bot)\rangle\}.
\end{aligned}$$

We have $(\mathfrak{s},\mathfrak{h}'_1) \rhd_{id} (\mathfrak{s},\mathfrak{h})$ and $(\mathfrak{s},\mathfrak{h}'_2) \rhd_\gamma (\mathfrak{s},\mathfrak{h})$, with $\gamma \stackrel{\mathsf{def}}{=} \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,2\rangle, \langle 4,2\rangle\}$. Also, $\mathrm{trunc}(\mathfrak{h}'_1) = \{\langle 1,2\rangle, \langle 2,2\rangle\} = \mathfrak{h}$ and $\mathrm{trunc}(\mathfrak{h}'_2) = \{\langle 1,3\rangle, \langle 2,4\rangle\}$. Note that $\mathfrak{h}$ has out-degree $\kappa = 1$, whereas $\mathfrak{h}'_1$ and $\mathfrak{h}'_2$ have out-degree 3. ■

Lemma 15. *If* $(\mathfrak{s},\mathfrak{h}') \rhd_\gamma (\mathfrak{s},\mathfrak{h})$ *then* $\mathfrak{h} = \gamma(\mathrm{trunc}(\mathfrak{h}'))$*, hence* $(\mathfrak{s},\mathfrak{h}') \rhd_{id} (\mathfrak{s},\mathrm{trunc}(\mathfrak{h}'))$.

The converse of Lemma 15 does not hold in general, but it holds under some additional conditions:

Lemma 16. *Consider a store $\mathfrak{s}$, let $\mathfrak{h}'$ be a heap and let* $\mathfrak{h} \stackrel{\mathsf{def}}{=} \mathrm{trunc}(\mathfrak{h}')$*. Let* $D_2 \stackrel{\mathsf{def}}{=} \{\ell \in \mathrm{dom}(\mathfrak{h}') \mid \mathfrak{h}'(\ell) = \mathbb{1}\}$ *and* $D_1 \stackrel{\mathsf{def}}{=} \mathrm{dom}(\mathfrak{h}') \setminus D_2$*. Assume that:*

1. *for every location $\ell \in D_1$, $\mathfrak{h}(\ell)$ is of the form $(\ell_1, \ldots, \ell_\kappa)$ and $\mathfrak{h}'(\ell)$ is of the form $(\ell_1, \ldots, \ell_\kappa, \mathfrak{s}(\boldsymbol{w}), \ell'_1, \ldots, \ell'_\mu)$;*
2. *every location $\ell \in D_2$ has a connection in $\mathfrak{h}'$.*

Then $(\mathfrak{s},\mathfrak{h}') \rhd_{id} (\mathfrak{s},\mathfrak{h})$.

4.2 Transforming the Consequent

We first describe the transformation for the right-hand side of the entailment problem, as this transformation is simpler.

Definition 17. *We associate each n-ary predicate $p \in \mathcal{P}_r$ with a new predicate $\widehat{p}$ of arity $n+\nu$. We denote by $\widehat{\alpha}$ the formula obtained from α by replacing every predicate atom $p(x_1, \ldots, x_n)$ by $\widehat{p}(x_1, \ldots, x_n, \boldsymbol{w})$, where $\boldsymbol{w} = (w_1, \ldots, w_\nu)$.*

Definition 18. *We denote by $\widehat{\mathcal{R}}$ the set of rules of the form:*

$$\widehat{p}(x_1, \ldots, x_n, \boldsymbol{w}) \Leftarrow x_1 \mapsto (y_1, \ldots, y_\kappa, \boldsymbol{w}, z_1, \ldots, z_\mu)\sigma * \widehat{\rho}\sigma * \xi_I * \chi_\sigma$$

where:

- $p(x_1,\ldots,x_n) \Leftarrow x_1 \mapsto (y_1,\ldots,y_\kappa) * \rho$ *is a rule in* $\mathcal{R}$ *with* $p \in \mathcal{P}_r$,
- $z_1,\ldots,z_\mu$ *are variables not occurring in* $\mathrm{fv}(\rho) \cup \{x_1,\ldots,x_n,y_1,\ldots,y_\kappa,w_1,\ldots,w_\nu\}$,
- σ *is a substitution with* $\mathrm{dom}(\sigma) \subseteq \mathrm{fv}(\rho) \setminus \{x_1\}$ *and* $\mathrm{rng}(\sigma) \subseteq \{w_1,\ldots,w_\nu\}$,
- $\xi_I \stackrel{\text{def}}{=} *_{i\in I} \underline{\bot}(z_i)$, *with* $I \subseteq \{1,\ldots,\mu\}$,
- $\chi_\sigma \stackrel{\text{def}}{=} *_{x\in\mathrm{dom}(\sigma)} x \approx x\sigma$.

We denote by $\mathcal{R}_c$ *the set of rules in* $\widehat{\mathcal{R}}$ *that are connected*[5].

Note that the free variables $\boldsymbol{w}$ are added as parameters in the rules above, instead of some arbitrary tuple of fresh variables $\boldsymbol{\omega}$, of the same length as $\boldsymbol{w}$. This is for the sake of conciseness, since these parameters $\boldsymbol{\omega}$ will be systematically mapped to $\boldsymbol{w}$.

Example 19. Assume that $\psi = \exists x\,.\, p(x,w_1)$, with $\nu = 1, \mu = 1$ and $\lambda(p) = \{2\}$. Assume also that p is associated with the rule: $p(u_1,u_2) \Leftarrow u_1 \mapsto u_1 * q(u_2)$. Observe that the rule is λ-connected, but not connected. Then $\mathrm{dom}(\sigma) \subseteq \{u_2\}$, $\mathrm{rng}(\sigma) \subseteq \{w_1\}$ and $I \subseteq \{1\}$, so that $\widehat{\mathcal{R}}$ contains the following rules:

$$
\begin{array}{l}
(1)\ p(u_1,u_2,w_1) \Leftarrow u_1 \mapsto (u_1,w_1,z_1) * q(u_2) \\
(2)\ p(u_1,u_2,w_1) \Leftarrow u_1 \mapsto (u_1,w_1,z_1) * q(u_2) * \underline{\bot}(z_1) \\
(3)\ p(u_1,u_2,w_1) \Leftarrow u_1 \mapsto (u_1,w_1,z_1) * q(w_1) * u_2 \approx w_1 \\
(4)\ p(u_1,u_2,w_1) \Leftarrow u_1 \mapsto (u_1,w_1,z_1) * q(w_1) * \underline{\bot}(z_1) * u_2 \approx w_1
\end{array}
$$

Rules (1) and (2) are not connected, hence do not occur in $\mathcal{R}_c$. Rules (3) and (4) are connected, hence occur in $\mathcal{R}_c$. Note that (4) is established, but (3) is not. ■

We now relate the SIDs $\mathcal{R}$ and $\mathcal{R}_c$ by the following result:

Lemma 20. *Let* α *be a formula that is* λ*-restricted w.r.t.* $\{w_1,\ldots,w_\nu\}$ *and contains no points-to atoms, with* $\mathcal{P}(\alpha) \subseteq \mathcal{P}_r$. *Given a store* $\mathfrak{s}$ *and two heaps* $\mathfrak{h}$ *and* $\mathfrak{h}'$, *such that* $(\mathfrak{s},\mathfrak{h}') \triangleright_{id} (\mathfrak{s},\mathfrak{h})$, *we have* $(\mathfrak{s},\mathfrak{h}') \models_{\mathcal{R}_c} \widehat{\alpha}$ *if and only if* $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \alpha$.

4.3 Transforming the Antecedent

We now describe the transformation operating on the left-hand side of the entailment problem. For technical convenience, we make the following assumption:

Assumption 21. *We assume that, for every predicate* $p \in \mathcal{P}_l$, *every rule of the form* $p(x_1,\ldots,x_n) \Leftarrow \pi$ *in* $\mathcal{R}$ *and every atom* $q(x'_1,\ldots,x'_m)$ *occurring in* π, $x'_1 \notin \{x_1,\ldots,x_n\}$.

This is without loss of generality, because every variable $x'_1 \in \{x_1,\ldots,x_n\}$ can be replaced by a fresh variable z, while conjoining the equational atom $z \approx x'_1$ to π. Note that the obtained SID may no longer be connected, but this is not problematic, because the left-hand side of the entailment is not required to be connected anyway.

Definition 22. *We associate each pair* (p,X), *where* $p \in \mathcal{P}_l$, $ar(p) = n$ *and* $X \subseteq [\![1,n]\!]$, *with a fresh predicate symbol* p_X, *such that* $ar(p_X) = n+\nu$. *A* decoration *of a formula* α *containing no points-to atoms, such that* $\mathcal{P}(\alpha) \subseteq \mathcal{P}_l$, *is a formula obtained by replacing each predicate atom* $\beta \stackrel{\text{def}}{=} q(y_1,\ldots,y_m)$ *in* α *by an atom of the form* $q_{X_\beta}(y_1,\ldots,y_m,\boldsymbol{w})$, *with* $X_\beta \subseteq [\![1,m]\!]$. *The set of decorations of a formula* α *is denoted by* $D(\alpha)$.

[5] Note that all the rules in $\widehat{\mathcal{R}}$ are progressing.

The role of the set X in a predicate atom $p_X(x_1,\ldots,x_n,\boldsymbol{w})$ will be explained below. Note that the set of decorations of an atom α is always finite.

Definition 23. *We denote by $D(\mathcal{R})$ the set of rules of the form*

$$p_X(x_1,\ldots,x_n,\boldsymbol{w}) \Leftarrow x_1 \mapsto (y_1,\ldots,y_\kappa,\boldsymbol{w},z_1,\ldots,z_\mu)\sigma * \rho' * *_{i\in I}\underline{\perp}(z_i),$$

where:

- $p(x_1,\ldots,x_n) \Leftarrow x_1 \mapsto (y_1,\ldots,y_\kappa) * \rho$ *is a rule in $\mathcal{R}$ and $X \subseteq [\![1,n]\!]$;*
- $\{z_1,\ldots,z_\mu\} = (\mathrm{fv}(\rho) \cup \{y_1,\ldots,y_\kappa\}) \setminus \{x_1,\ldots,x_n\}$,
- σ *is a substitution, with* $\mathrm{dom}(\sigma) \subseteq \{z_1,\ldots,z_\mu\}$ *and* $\mathrm{rng}(\sigma) \subseteq \{x_1,\ldots,x_n,w_1,\ldots,w_\nu, z_1,\ldots,z_\mu\}$*;*
- ρ' *is a decoration of* $\rho\sigma$*;*
- $I \subseteq \{1,\ldots,\mu\}$ *and* $z_i \notin \mathrm{dom}(\sigma)$*, for all* $i \in I$.

Lemma 24. *Let α be a formula containing no points-to atom, with $\mathcal{P}(\alpha) \subseteq \mathcal{P}_l$, and let α' be a decoration of α. If $(\mathfrak{s},\mathfrak{h}') \models_{D(\mathcal{R})} \alpha'$ and $(\mathfrak{s},\mathfrak{h}') \triangleright_{id} (\mathfrak{s},\mathfrak{h})$, then $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{R}} \alpha$.*

At this point, the set X for predicate symbol p_X is of little interest: atoms are simply decorated with arbitrary sets. However, we shall restrict the considered rules in such a way that for every model $(\mathfrak{s},\mathfrak{h})$ of an atom $p_X(x_1,\ldots,x_{n+\nu})$, with $n = ar(p)$, the set X denotes a set of indices $i \in [\![1,n]\!]$ such that $\mathfrak{s}(x_i) \in \mathrm{dom}(\mathfrak{h})$. In other words, X will denote a set of formal parameters of p_X that are allocated in every model of p_X.

Definition 25. *Given a formula α, we define the set $Alloc(\alpha)$ as follows: $x \in Alloc(\alpha)$ iff α contains either a points-to atom of the form $x \mapsto (y_1,\ldots,y_{\kappa+\mu+\nu})$, or a predicate atom $q_X(x'_1,\ldots,x'_{m+\nu})$ with $x'_i = x$ for some $i \in X$.*

Note that, in contrast with Definition 1, we do not consider that $x \in Alloc(\alpha)$, for those variables x related to a variable from $Alloc(\alpha)$ by equalities.

Definition 26. *A rule $p_X(x_1,\ldots,x_{n+\nu}) \Leftarrow \pi$ in $D(\mathcal{R})$ with $n = ar(p)$ with $\rho = x_1 \mapsto (y_1,\ldots,y_k,\boldsymbol{w},z_1,\ldots,z_\mu) * \rho'$ is* well-defined *if the following conditions hold:*

1. $\{x_1\} \subseteq Alloc(p_X(x_1,\ldots,x_{n+\nu})) \subseteq Alloc(\pi)$*;*
2. $\mathrm{fv}(\pi) \subseteq Alloc(\pi) \cup \{x_1,\ldots,x_{n+\nu}\}$.

We denote by $\mathcal{R}_l$ the set of well-defined rules in $D(\mathcal{R})$.

We first state an important properties of $\mathcal{R}_l$.

Lemma 27. *Every rule in $\mathcal{R}_l$ is progressing, connected and established.*

We now relate the systems $\mathcal{R}$ and $\mathcal{R}_l$ by the following result:

Definition 28. *A store $\mathfrak{s}$ is* quasi-injective *if, for all $x,y \in \mathrm{dom}(\mathfrak{s})$, the implication $\mathfrak{s}(x) = \mathfrak{s}(y) \Rightarrow x = y$ holds whenever $\{x,y\} \not\subseteq \{w_1,\ldots,w_\nu\}$.*

Lemma 29. *Let L be an infinite subset of $\mathcal{L}$. Consider a formula α containing no points-to atom, with $\mathcal{P}(\alpha) \subseteq \mathcal{P}_l$, and let $(\mathfrak{s},\mathfrak{h})$ be an $\mathcal{R}$-model of α, where $\mathfrak{s}$ is quasi-injective, and $(\mathrm{rng}(\mathfrak{s}) \cup \mathrm{loc}(\mathfrak{h})) \cap L = \emptyset$. There exists a decoration α' of α, a heap $\mathfrak{h}'$ and a mapping $\gamma : \mathcal{L} \to \mathcal{L}$ such that:*

– $(\mathfrak{s},\mathfrak{h}') \rhd_\gamma (\mathfrak{s},\mathfrak{h})$,
– *if* $\ell \notin L$ *then* $\gamma(\ell) = \ell$,
– $\mathrm{loc}(\mathfrak{h}') \setminus \mathrm{rng}(\mathfrak{s}) \subseteq L$,
– $\mathrm{dom}(\mathrm{aux}(\mathfrak{h}')) \subseteq L$ *and*
– $(\mathfrak{s},\mathfrak{h}') \models_{\mathcal{R}_\mathcal{U}} \alpha'$.

Furthermore, if $\mathfrak{s}(u) \in \mathrm{dom}(\mathfrak{h}') \setminus \{\mathfrak{s}(w_i) \mid 1 \leq i \leq \nu\}$ *then* $u \in \mathit{Alloc}(\alpha')$.

4.4 Transforming Entailments

We define $\widehat{\mathcal{R}} \stackrel{\text{def}}{=} \mathcal{R}_\mathcal{U} \cup \mathcal{R}_\mathcal{L}$. We show that the instance $\phi \vdash_\mathcal{R} \psi$ of the safe entailment problem can be solved by considering an entailment problem on $\widehat{\mathcal{R}}$ involving the elements of $D(\phi)$ (see Definition 22). Note that the rules from $\mathcal{R}_\mathcal{U}$ are progressing, connected and established, by Lemma 27, whereas the rules from $\mathcal{R}_\mathcal{L}$ are progressing and connected, by Definition 18. Hence, each entailment problem $\phi' \vdash_{\widehat{\mathcal{R}}} \widehat{\psi}$, where $\phi' \in D(\phi)$, is progressing, connected and left-established.

Lemma 30. $\phi \models_\mathcal{R} \psi$ *if and only if* $\bigvee_{\phi' \in D(\phi)} \phi' \models_{\widehat{\mathcal{R}}} \widehat{\psi}$.

Proof. "$\Rightarrow$" Assume that $\phi \models_\mathcal{R} \psi$ and let $\phi' \in D(\phi)$ be a formula, $(\mathfrak{s},\mathfrak{h}')$ be an $\widehat{\mathcal{R}}$-model of ϕ' and $\mathfrak{h} \stackrel{\text{def}}{=} \mathrm{trunc}(\mathfrak{h}')$. By construction, $(\mathfrak{s},\mathfrak{h}')$ is an $\mathcal{R}_\mathcal{U}$-model of ϕ'. By definition of $D(\phi)$, ϕ' is a decoration of ϕ. Let $D_2 \stackrel{\text{def}}{=} \{\ell \in \mathrm{dom}(\mathfrak{h}') \mid \mathfrak{h}'(\ell) = \mathbb{1}\!\!\mathbb{1}\}$, $D_1 \stackrel{\text{def}}{=} \mathrm{dom}(\mathfrak{h}') \setminus D_2$, and consider a location $\ell \in \mathrm{dom}(\mathfrak{h}')$. By definition, ℓ must be allocated by some rule in $\mathcal{R}_\mathcal{U}$. If ℓ is allocated by a rule of the form given in Definition 23, then necessarily $\mathfrak{h}'(\ell)$ is of the form $(\ell_1,\ldots,\ell_\kappa,\mathfrak{s}(w),\ell'_1,\ldots,\ell'_\mu)$ and $\ell \in D_1$. Otherwise, ℓ is allocated by the predicate $\bot\!\!\!\bot$ and we must have $\ell \in D_2$ by definition of the only rule for $\bot\!\!\!\bot$. Since this predicate must occur within a rule of the form given in Definition 23, ℓ necessarily occurs in the μ last components of the image of a location in D_1, hence admits a connection in $\mathfrak{h}'$. Consequently, by Lemma 16 $(\mathfrak{s},\mathfrak{h}') \rhd_{id} (\mathfrak{s},\mathfrak{h})$, and by Lemma 24, $(\mathfrak{s},\mathfrak{h}) \models_\mathcal{R} \phi$. Thus $(\mathfrak{s},\mathfrak{h}) \models_\mathcal{R} \psi$, and by Lemma 20, $(\mathfrak{s},\mathfrak{h}') \models_{\mathcal{R}_\mathcal{L}} \widehat{\psi}$, thus $(\mathfrak{s},\mathfrak{h}') \models_{\widehat{\mathcal{R}}} \widehat{\psi}$.

"$\Leftarrow$" Assume that $\bigvee_{\phi' \in D(\phi)} \phi' \models_{\widehat{\mathcal{R}}} \widehat{\psi}$ and let $(\mathfrak{s},\mathfrak{h})$ be a $\mathcal{R}$-model of ϕ. Since the truth values of ϕ and ψ depend only on the variables in $\mathrm{fv}(\phi) \cup \mathrm{fv}(\psi)$, we may assume, w.l.o.g., that $\mathfrak{s}$ is quasi-injective. Consider an infinite set $L \subseteq \mathcal{L}$ such that $(\mathrm{rng}(\mathfrak{s}) \cup \mathrm{loc}(\mathfrak{h})) \cap L = \emptyset$. By Lemma 29, there exist a heap $\mathfrak{h}'$, a mapping $\gamma : \mathcal{L} \to \mathcal{L}$ and a decoration ϕ' of ϕ such that $\gamma(\ell) = \ell$ for all $\ell \notin L$, $(\mathfrak{s},\mathfrak{h}') \rhd_\gamma (\mathfrak{s},\mathfrak{h})$ and $(\mathfrak{s},\mathfrak{h}') \models \phi'$. Since $\mathrm{rng}(\mathfrak{s}) \cap L = \emptyset$, we also have $\gamma(\mathfrak{s}) = \mathfrak{s}$. Then $(\mathfrak{s},\mathfrak{h}') \models \widehat{\psi}$. Let $\mathfrak{h}_1 \stackrel{\text{def}}{=} \mathrm{trunc}(\mathfrak{h}')$. Since $(\mathfrak{s},\mathfrak{h}') \rhd_\gamma (\mathfrak{s},\mathfrak{h})$, by Lemma 15 we have $(\mathfrak{s},\mathfrak{h}') \rhd_{id} (\mathfrak{s},\mathfrak{h}_1)$, and by Lemma 20, $(\mathfrak{s},\mathfrak{h}_1) \models \psi$. By Lemma 15 we have $\mathfrak{h} = \gamma(\mathfrak{h}_1)$. Since ψ is λ-restricted w.r.t. $\{w_1,\ldots,w_n\}$, we deduce by Lemma 10 that $(\mathfrak{s},\mathfrak{h}) \models \psi$. □

This leads to the main result of this paper:

Theorem 31. *The safe entailment problem is 2EXPTIME-complete.*

Proof. The 2EXPTIME-hard lower bound follows from [8, Theorem 32], as the class of progressing, λ-connected and λ-restricted entailment problems is a subset of the safe

entailment class. For the 2EXPTIME membership, Lemma 30 describes a many-one reduction to the progressing, connected and established class, shown to be in 2EXPTIME, by Theorem 4. Considering an instance $\mathfrak{P} = \phi \vdash_{\mathcal{R}} \psi$ of the safe class, Lemma 30 reduces this to checking the validity of $|D(\phi)|$ instances of the form $\phi' \vdash_{\widehat{\mathcal{R}}} \widehat{\psi}$, that are all progressing, connected and established, by Lemma 27. Since a formula $\phi' \in D(\phi)$ is obtained by replacing each predicate atom $p(x_1,\ldots,x_n)$ of ϕ by $p_X(x_1,\ldots,x_n,\boldsymbol{w})$ and there are at most 2^n such predicate atoms, it follows that $|D(\phi)| = 2^{O(\mathrm{w}(\mathfrak{P}))}$. To obtain 2EXPTIME-membership of the problem, it is sufficient to show that each of the progressing, connected and established instances $\phi' \vdash_{\widehat{\mathcal{R}}} \widehat{\psi}$ can be built in time $|\mathfrak{P}| \cdot 2^{O(\mathrm{w}(\mathfrak{P}) \cdot \log \mathrm{w}(\mathfrak{P}))}$. First, for each $\phi' \in D(\phi)$, by Definition 22, we have $|\phi'| \leq |\phi| \cdot (1+\nu) \leq |\phi| \cdot (1+\mathrm{w}(\mathfrak{P})) = |\phi| \cdot 2^{O(\log \mathrm{w}(\mathfrak{P}))}$. By Definition 17, we have $|\widehat{\phi}| \leq |\phi| \cdot (1+\nu) = |\phi| \cdot 2^{O(\log \mathrm{w}(\mathfrak{P}))}$. By Definition 23, $D(\mathcal{R})$ can be obtained by enumeration in time that depends linearly of

$$|D(\mathcal{R})| \leq |\mathcal{R}| \cdot 2^{\mu} \cdot (n+\nu+\mu)^{\nu} \leq |\mathcal{R}| \cdot 2^{\mathrm{w}(\mathfrak{P}) + \mathrm{w}(\mathfrak{P}) \cdot \log \mathrm{w}(\mathfrak{P})} = |\mathfrak{P}| \cdot 2^{O(\mathrm{w}(\mathfrak{P}))}$$

This is because the number of intervals I is bounded by 2^{μ} and the number of substitutions σ by $(n+\nu+\mu)^{\nu}$, in Definition 23. By Definition 25, checking whether a rule is well-defined can be done in polynomial time in the size of the rule, hence in $2^{O(\mathrm{w}(\mathfrak{P}))}$, so the construction of $\mathcal{R}_{\mathcal{U}}$ takes time $|\mathfrak{P}| \cdot 2^{O(\mathrm{w}(\mathfrak{P}) \log \mathrm{w}(\mathfrak{P}))}$. Similarly, by Definition 23, the set $\widehat{\mathcal{R}}$ is constructed in time

$$|\widehat{\mathcal{R}}| \leq |\mathcal{R}| \cdot 2^{\mu} \cdot \mathrm{w}(\mathfrak{P})^{\nu} \leq |\mathcal{R}| \cdot 2^{\mathrm{w}}(\mathfrak{P}) \cdot 2^{\mathrm{w}(\mathfrak{P}) \cdot \log \mathrm{w}(\mathfrak{P})} = |\mathfrak{P}| \cdot 2^{O(\mathrm{w}(\mathfrak{P}))}$$

Moreover, checking that a rule in $\widehat{\mathcal{R}}$ is connected can be done in time polynomial in the size of the rule, hence the construction of $\mathcal{R}_{\mathcal{C}}$ takes time $2^{O(\mathrm{w}(\mathfrak{P}) \log \mathrm{w}(\mathfrak{P}))}$. Then the entire reduction takes time $2^{O(\mathrm{w}(\mathfrak{P}) \log \mathrm{w}(\mathfrak{P}))}$, which proves the 2EXPTIME upper bound for the safe class of entailments. □

5 Conclusion and Future Work

Together with the results of [10,14,6,8], Theorem 31 draws a clear and complete picture concerning the decidability and complexity of the entailment problem in Separation Logic with inductive definitions. The room for improvement in this direction is probably very limited, since Theorem 31 pushes the frontier quite far. Moreover, virtually any further relaxation of the conditions leads to undecidability.

A possible line of future research which could be relevant for applications would be to consider inductive rules constructing simultaneously several data structures, which could be useful for instance to handle predicates comparing two structures, but it is clear that very strong conditions would be required to ensure decidability. We are also interested in defining effective, goal-directed, proof procedures (i.e., sequent or tableaux calculi) for testing the validity of entailment problems. Thanks to the reduction devised in the present paper, it is sufficient to focus on systems that are progressing, connected and left-established. We are also trying to extend the results to entailments with formulæ involving data with infinite domains, either by considering a theory of locations (e.g., arithmetic on addresses), or, more realistically, by considering additional sorts for data.

References

1. Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *FOSSACS 2014, ETAPS 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 411–425, 2014.
2. Josh Berdine, Byron Cook, and Samin Ishtiaq. Slayer: Memory safety for systems-level code. In Ganesh Gopalakrishnan andShaz Qadeer, editor, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *LNCS*, pages 178–183. Springer, 2011.
3. Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, volume 9058 of *LNCS*, pages 3–11. Springer, 2015.
4. Duc-Hiep Chu, Joxan Jaffar, and Minh-Thai Trinh. Automatic induction proofs of data-structures in imperative programs. In David Grove and Stephen M. Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 457–466. ACM, 2015. doi:10.1145/2737924.2737984.
5. Kamil Dudka, Petr Peringer, and Tomás Vojnar. Predator: A practical tool for checking manipulation of dynamic data structures using separation logic. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *LNCS*, pages 372–378. Springer, 2011.
6. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Entailment checking in separation logic with inductive definitions is 2-exptime hard. In *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.
7. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Entailment is Undecidable for Symbolic Heap Separation Logic Formulae with Non-Established Inductive Rules. working paper or preprint, September 2020. URL: https://hal.archives-ouvertes.fr/hal-02951630.
8. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Decidable entailments in separation logic with inductive definitions: Beyond establishment. In *CSL 2021: 29th International Conference on Computer Science Logic*, EPiC Series in Computing. EasyChair, 2021.
9. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Unifying decidable entailments in separation logic with inductive definitions, 2021. arXiv:2012.14361.
10. Radu Iosif, Adam Rogalewicz, and Jiri Simacek. The tree width of separation logic with recursive definitions. In *Proc. of CADE-24*, volume 7898 of *LNCS*, 2013.
11. Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In Franck Cassez and Jean-François Raskin, editors, *ATVA 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 201–218. Springer, 2014.
12. Samin S Ishtiaq and Peter W O'Hearn. Bi as an assertion language for mutable data structures. In *ACM SIGPLAN Notices*, volume 36, pages 14–26, 2001.
13. Jens Katelaan, Christoph Matheja, and Florian Zuleger. Effective entailment checking for separation logic with inductive definitions. In Tomás Vojnar and Lijun Zhang, editors, *TACAS 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 319–336. Springer, 2019.

14. Jens Pagel and Florian Zuleger. Beyond symbolic heaps: Deciding separation logic with inductive definitions. In *LPAR-23*, volume 73 of *EPiC Series in Computing*, pages 390–408. EasyChair, 2020.
15. J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*, 2002.
16. Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. Automated lemma synthesis in symbolic-heap separation logic. *Proc. ACM Program. Lang.*, 2(POPL):9:1–9:29, 2018. doi:10.1145/3158097.