Decidable Entailments in Separation Logic with Inductive Definitions: Beyond Establishment

Mnacho Echenim
Université Grenoble Alpes, CNRS, LIG, F-38000 Grenoble, France

Radu Iosif
Université Grenoble Alpes, CNRS, VERIMAG, F-38000 Grenoble, France

Nicolas Peltier
Université Grenoble Alpes, CNRS, LIG, F-38000 Grenoble, France

Abstract

We define a class of Separation Logic [10, 16] formulæ, whose entailment problem *given formulæ* $\phi, \psi_1, \ldots, \psi_n$*, is every model of* ϕ *a model of some* ψ_i*?* is 2-EXPTIME-complete. The formulæ in this class are existentially quantified separating conjunctions involving predicate atoms, interpreted by the least sets of store-heap structures that satisfy a set of inductive rules, which is also part of the input to the entailment problem. Previous work [8, 12, 15] consider *established* sets of rules, meaning that every existentially quantified variable in a rule must eventually be bound to an *allocated* location, i.e. from the domain of the heap. In particular, this guarantees that each structure has treewidth bounded by the size of the largest rule in the set. In contrast, here we show that establishment, although sufficient for decidability (alongside two other natural conditions), is not necessary, by providing a condition, called *equational restrictedness*, which applies syntactically to (dis-)equalities. The entailment problem is more general in this case, because equationally restricted rules define richer classes of structures, of unbounded treewidth. In this paper we show that

(1) every established set of rules can be converted into an equationally restricted one and
(2) the entailment problem is 2-EXPTIME-complete in the latter case, thus matching the complexity of entailments for established sets of rules [12, 15].

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Separation logic, Induction definitions, Inductive theorem proving, Entailments, Complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.20

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.00502`.

1 Introduction

Separation Logic (SL) [10, 16] is widely used to reason about programs manipulating recursively linked data structures, being at the core of several industrial-scale static program analysis techniques [3, 2, 5]. Given an integer $\mathfrak{K} \geq 1$, denoting the number of fields in a record datatype, and an infinite set $\mathbb{L}$ of memory locations (addresses), the assertions in this logic describe *heaps*, that are finite partial functions mapping locations to records, i.e., $\mathfrak{K}$-tuples of locations. A location ℓ in the domain of the heap is said to be *allocated* and the *points-to* atom $x \mapsto (y_1, \ldots, y_{\mathfrak{K}})$ states that the location associated with x refers to the tuple of locations associated with $(y_1, \ldots, y_{\mathfrak{K}})$. The *separating conjunction* $\phi * \psi$ states that the formulæ ϕ and ψ hold in non-overlapping parts of the heap, that have disjoint domains. This connective allows for modular program analyses, because the formulæ specifying the behaviour of a program statement refer only to the small (local) set of locations that are manipulated by that statement, with no concern for the rest of the program's state.

Formulæ consisting of points-to atoms connected with separating conjunctions describe heaps of bounded size only. To reason about recursive data structures of unbounded sizes (lists, trees, etc.), the base logic is enriched by predicate symbols, with a semantics specified

by user-defined inductive rules. For instance, the rules: $\mathsf{excls}(x, y) \Leftarrow \exists z \ . \ x \mapsto (z, y) * z \neq \mathsf{c}$ and $\mathsf{excls}(x, y) \Leftarrow \exists z \exists v \ . \ x \mapsto (z, v) * \mathsf{excls}(v, y) * z \neq \mathsf{c}$ describe a non-empty list segment, whose elements are records with two fields: the first is a data field, that keeps a list of locations, which excludes the location assigned to the global constant c, and the second is used to link the records in a list whose head and tail are pointed to by x and y, respectively.

An important problem in program verification, arising during construction of Hoare-style correctness proofs, is the discharge of verification conditions, that are entailments of the form $\phi \vdash \psi_1, \ldots, \psi_n$, where ϕ and $\psi_1, \ldots, \psi_n$ are separating conjunctions of points-to, predicates and (dis-)equalities, also known as *symbolic heaps*. The *entailment problem* then asks if *every model of ϕ is a model of some ψ_i?* In general, the entailment problem is undecidable and becomes decidable when the inductive rules used to interpret the predicates satisfy three restrictions [8]:

(1) *progress*, stating that each rule allocates *exactly* one memory cell,
(2) *connectivity*, ensuring that the allocated memory cells form a tree-shaped structure, and
(3) *establishment*, stating that all existentially quantified variables introduced by an inductive rule must be assigned to some allocated memory cell, in every structure defined by that rule.

For instance, the above rules are progressing and connected but not established, because the $\exists z$ variables are not explicitly assigned an allocated location, unlike the $\exists v$ variables, passed as first parameter of the $\mathsf{excls}(x, y)$ predicate, and thus always allocated by the points-to atoms $x \mapsto (z, y)$ or $x \mapsto (z, v)$, from the first and second rule defining $\mathsf{excls}(x, y)$, respectively.

The argument behind the decidability of a progressing, connected and established entailment problem is that every model of the left-hand side is encoded by a graph whose treewidth[1] is bounded by the size of the largest symbolic heap that occurs in the problem [8]. Moreover, the progress and connectivity conditions ensure that the set of models of a symbolic heap can be represented by a Monadic Second Order (MSO) logic formula interpreted over graphs, that can be effectively built from the symbolic heap and the set of rules of the problem. The decidability of entailments follows then from the decidability of the satisfiability problem for MSO over graphs of bounded treewidth (Courcelle's Theorem) [4]. Initially, no upper bound better than elementary recursive was known to exist. Recently, a 2-EXPTIME algorithm was proposed [12, 14] for sets of rules satisfying these three conditions, and, moreover, this bound was shown to be tight [6].

Several natural questions arise: are the progress, connectivity and establishment conditions really necessary for the decidability of entailments? How much can these restriction be relaxed, without jeopardizing the complexity of the problem? Can one decide entailments that involve sets of heaps of unbounded treewidth? In this paper, we answer these questions by showing that entailments are still 2-EXPTIME-complete when the establishment condition is replaced by a condition on the (dis-)equations occurring in the symbolic heaps of the problem. Informally, such (dis-)equations must be of the form $x \doteq \mathsf{c}$ ($x \neq \mathsf{c}$), where c ranges over some finite and fixed set of globally visible constants (including special symbols such as nil, that denotes a non-allocated address, but also any free variable occurring on the left-hand side of the entailment). We also relax slightly the progress and connectivity conditions, by allowing forest-like heap structures (instead of just trees), provided that every root is mapped to a constant symbol. These entailment problems are called *equationally restricted* (*e-restricted*, for short). For instance, the entailment problem $\mathsf{excls}(x, y) * \mathsf{excls}(y, z) \vdash \mathsf{excls}(x, z)$, with the above rules, falls in this category.

[1] The treewidth of a graph is a parameter measuring how close the graph is to a tree, see [7, Ch. 11] for a definition.

We prove that the e-restricted condition loses no generality compared to establishment, because any established entailment problem can be transformed into an equivalent e-restricted entailment problem. E-restricted problems allow reasoning about structures that contain dangling pointers, which frequently occur in practice, especially in the context of modular program analysis. Moreover, the set of structures considered in an e-restricted entailment problem may contain infinite sequences of heaps of strictly increasing treewidths, that are out of the scope of established problems [8].

The decision procedure for e-restricted problems proposed in this paper is based on a similar idea as the one given, for established problems, in [14, 15]. We build a suitable abstraction of the set of structures satisfying the left-hand side of the entailment bottom-up, starting from points-to and predicate atoms, using abstract operators to compose disjoint structures, to add and remove variables, and to unfold the inductive rules associated with the predicates. The abstraction is precise enough to allow checking that all the models of the left-hand side fulfill the right-hand side of the entailment and also general enough to ensure termination of the entailment checking algorithm.

Although both procedures are similar, there are essential differences between our work and [14, 15]. First, we show that instead of using a specific language for describing those abstractions, the considered set of structures can themselves be defined in SL, by means of formulæ of some specific pattern called *core formulæ*. Second, the fact that the systems are not established makes the definition of the procedure much more difficult, due to the fact that the considered structures can have an unbounded treewidth. This is problematic because, informally, this boundedness property is essential to ensure that the abstractions can be described using a finite set of variables, denoting the *frontier* of the considered structures, namely the locations that can be shared with other structures. In particular, the fact that disjoint heaps may share unallocated (or "unnamed") locations complexifies the definition of the composition operator. This problem is overcome by considering a specific class of structures, called *normal structures*, of bounded treewidth, and proving that the validity of an entailment can be decided by considering only normal structures.

In terms of complexity, we show that the running time of our algorithm is doubly exponential w.r.t. the maximal size among the symbolic heaps occurring in the input entailment problem (including those in the rules) and simply exponential w.r.t. the number of such symbolic heaps (hence w.r.t. the number of rules). This means that the 2-EXPTIME upper bound is preserved by any reduction increasing exponentially the number of rules, but increasing only polynomially the size of the rules. On the other hand, the 2-EXPTIME-hard lower bound is proved by a reduction from the membership problem for exponential-space bounded Alternating Turing Machines [6].

2 Separation Logic with Inductive Definitions

Let $\mathbb{N}$ denote the set of natural numbers. For a countable set S, we denote by $||S|| \in \mathbb{N} \cup \{\infty\}$ its cardinality. For a partial mapping $f : A \rightharpoonup B$, let $\mathrm{dom}(f) \stackrel{\text{def}}{=} \{x \in A \mid f(x) \in B\}$ and $\mathrm{rng}(f) \stackrel{\text{def}}{=} \{f(x) \mid x \in \mathrm{dom}(f)\}$ be its domain and range, respectively. We say that f is *total* if $\mathrm{dom}(f) = A$, written $f : A \to B$ and *finite*, written $f : A \rightharpoonup_{fin} B$ if $||\mathrm{dom}(f)|| < \infty$. Given integers n and m, we denote by $[\![n \,..\, m]\!]$ the set $\{n, n+1, \ldots, m\}$, so that $[\![n \,..\, m]\!] = \emptyset$ if $n > m$. For a relation $\lhd \subseteq A \times A$, we denote by $\lhd^*$ its reflexive and transitive closure.

For an integer $n \geq 0$, let A^n be the set of n-tuples with elements from A. Given a tuple $\mathbf{a} = (a_1, \ldots, a_n)$ and $i \in [\![1 \,..\, n]\!]$, we denote by $\mathbf{a}_i$ the i-th element of $\mathbf{a}$ and by $|\mathbf{a}| \stackrel{\text{def}}{=} n$ its length. By $f(\mathbf{a})$ we denote the tuple obtained by the pointwise application of f to the

elements of $\mathbf{a}$. If multiplicity and order of the elements are not important, we blur the distinction between tuples and sets, using the set-theoretic notations $x \in \mathbf{a}$, $\mathbf{a} \cup \mathbf{b}$, $\mathbf{a} \cap \mathbf{b}$ and $\mathbf{a} \setminus \mathbf{b}$.

Let $\mathbb{V} = \{x, y, \ldots\}$ be an infinite countable set of logical first-order variables and $\mathbb{P} = \{p, q, \ldots\}$ be an infinite countable set (disjoint from $\mathbb{V}$) of relation symbols, called *predicates*, where each predicate p has arity $\#p \geq 0$. We also consider a finite set $\mathbb{C}$ of *constants*, of known bounded cardinality, disjoint from both $\mathbb{V}$ and $\mathbb{P}$. Constants will play a special rôle in the upcoming developments and the fact that $\mathbb{C}$ is bounded is of a particular importance. A *term* is either a variable or a constant and we denote by $\mathbb{T} \stackrel{\text{def}}{=} \mathbb{V} \cup \mathbb{C}$ the set of terms.

Throughout this paper we consider an integer $\mathfrak{K} \geq 1$ that, intuitively, denotes the number of fields in a record datatype. Although we do not assume $\mathfrak{K}$ to be a constant in any of the algorithms presented in the following, considering that every datatype has exactly $\mathfrak{K}$ records simplifies the definition. The logic $\mathsf{SL}^{\mathfrak{K}}$ is the set of formulæ generated inductively by the syntax:

$$\phi \quad := \quad \mathsf{emp} \mid t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) \mid p(t_1, \ldots, t_{\#p}) \mid t_1 \approx t_2 \mid \phi_1 * \phi_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi_1 \mid \exists x \,.\, \phi_1$$

where $p \in \mathbb{P}$, $t_i \in \mathbb{T}$ and $x \in \mathbb{V}$. Atomic propositions of the form $t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})$ are called *points-to atoms* and those of the form $p(t_1, \ldots, t_{\#p})$ are *predicate atoms*. If $\mathfrak{K} = 1$, we write $t_0 \mapsto t_1$ for $t_0 \mapsto (t_1)$.

The connective $*$ is called *separating conjunction*, in contrast with the classical conjunction $\wedge$. The *size* of a formula ϕ, denoted by $\mathrm{size}(\phi)$, is the number of occurrences of symbols in it. We write $\mathsf{fv}(\phi)$ for the set of *free* variables in ϕ and $\mathsf{trm}(\phi) \stackrel{\text{def}}{=} \mathsf{fv}(\phi) \cup \mathbb{C}$. A formula is *predicate-free* if it has no predicate atoms. As usual, $\phi_1 \vee \phi_2 \stackrel{\text{def}}{=} \neg(\neg\phi_1 \wedge \neg\phi_2)$ and $\forall x \,.\, \phi \stackrel{\text{def}}{=} \neg\exists x \,.\, \neg\phi$. For a set of variables $\mathbf{x} = \{x_1, \ldots, x_n\}$ and a quantifier $Q \in \{\exists, \forall\}$, we write $Q\mathbf{x} \,.\, \phi \stackrel{\text{def}}{=} Qx_1 \ldots Qx_n \,.\, \phi$. By writing $t_1 = t_2$ ($\phi_1 = \phi_2$) we mean that the terms (formulæ) t_1 and t_2 (ϕ_1 and ϕ_2) are syntactically the same.

A *substitution* is a partial mapping $\sigma : \mathbb{V} \rightharpoonup \mathbb{T}$ that maps variables to terms. We denote by $[t_1/x_1, \ldots, t_n/x_n]$ the substitution that maps the variable x_i to t_i, for each $i \in [\![1 \,..\, n]\!]$ and is undefined elsewhere. By $\phi\sigma$ we denote the formula obtained from ϕ by substituting each variable $x \in \mathsf{fv}(\phi)$ by $\sigma(x)$ (we assume that bound variables are renamed to avoid collisions if needed). By abuse of notation, we sometimes write $\sigma(x)$ for x, when $x \notin \mathrm{dom}(\sigma)$.

To interpret $\mathsf{SL}^{\mathfrak{K}}$ formulæ, we consider an infinite countable set $\mathbb{L}$ of *locations*. The semantics of $\mathsf{SL}^{\mathfrak{K}}$ formulæ is defined in terms of *structures* $(\mathfrak{s}, \mathfrak{h})$, where:

- $\mathfrak{s} : \mathbb{T} \rightharpoonup \mathbb{L}$ is a partial mapping of terms into locations, called a *store*, that interprets at least all the constants, i.e. $\mathbb{C} \subseteq \mathrm{dom}(\mathfrak{s})$ for every store $\mathfrak{s}$, and
- $\mathfrak{h} : \mathbb{L} \rightharpoonup_{fin} \mathbb{L}^{\mathfrak{K}}$ is a finite partial mapping of locations into $\mathfrak{K}$-tuples of locations, called a *heap*.

Given a heap $\mathfrak{h}$, let $\mathrm{loc}(\mathfrak{h}) \stackrel{\text{def}}{=} \{\ell_0, \ldots, \ell_{\mathfrak{K}} \mid \ell_0 \in \mathrm{dom}(\mathfrak{h}),\ \mathfrak{h}(\ell_0) = (\ell_1, \ldots, \ell_{\mathfrak{K}})\}$ be the set of locations that occur in the heap $\mathfrak{h}$. 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 their *disjoint union* is denoted by $\mathfrak{h}_1 \uplus \mathfrak{h}_2$, otherwise undefined. The *frontier between* $\mathfrak{h}_1$ *and* $\mathfrak{h}_2$ is the set of common locations $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \stackrel{\text{def}}{=} \mathrm{loc}(\mathfrak{h}_1) \cap \mathrm{loc}(\mathfrak{h}_2)$. Note that disjoint heaps may have nonempty frontier. The *satisfaction relation* $\models$ between structures $(\mathfrak{s}, \mathfrak{h})$ and predicate-free $\mathsf{SL}^{\mathfrak{K}}$ formulæ ϕ is defined recursively on the structure of formulæ:

$$\begin{array}{ll}
(\mathfrak{s}, \mathfrak{h}) \models t_1 \approx t_2 & \Leftrightarrow t_1, t_2 \in \mathrm{dom}(\mathfrak{s}) \text{ and } \mathfrak{s}(t_1) = \mathfrak{s}(t_2) \\
(\mathfrak{s}, \mathfrak{h}) \models \mathsf{emp} & \Leftrightarrow \mathfrak{h} = \emptyset \\
(\mathfrak{s}, \mathfrak{h}) \models t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) & \Leftrightarrow t_0, \ldots, t_{\mathfrak{K}} \in \mathrm{dom}(\mathfrak{s}),\ \mathrm{dom}(\mathfrak{h}) = \{\mathfrak{s}(t_0)\} \text{ and } \mathfrak{h}(\mathfrak{s}(t_0)) = (\mathfrak{s}(t_1), \ldots, \mathfrak{s}(t_{\mathfrak{K}})) \\
(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \wedge \phi_2 & \Leftrightarrow (\mathfrak{s}, \mathfrak{h}) \models \phi_i,\ i = 1, 2 \\
(\mathfrak{s}, \mathfrak{h}) \models \neg\phi_1 & \Leftrightarrow \mathsf{fv}(\phi_1) \subseteq \mathrm{dom}(\mathfrak{s}) \text{ and } (\mathfrak{s}, \mathfrak{h}) \not\models \phi_1 \\
(\mathfrak{s}, \mathfrak{h}) \models \phi_1 * \phi_2 & \Leftrightarrow \text{there exist heaps } \mathfrak{h}_1, \mathfrak{h}_2 \text{ such that } \mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2 \text{ and } (\mathfrak{s}, \mathfrak{h}_i) \models \phi_i,\ i = 1, 2 \\
(\mathfrak{s}, \mathfrak{h}) \models \exists x \,.\, \phi & \Leftrightarrow (\mathfrak{s}[x \leftarrow \ell], \mathfrak{h}) \models \phi, \text{ for some location } \ell \in \mathbb{L}
\end{array}$$

where $\mathfrak{s}[x \leftarrow \ell]$ is the store, with domain $\mathrm{dom}(\mathfrak{s}) \cup \{x\}$, that maps x to ℓ and behaves like $\mathfrak{s}$ over $\mathrm{dom}(\mathfrak{s}) \setminus \{x\}$. For a tuple of variables $\mathbf{x} = (x_1, \ldots, x_n)$ and locations $\mathbf{l} = (l_1, \ldots, l_n)$, we call the store $\mathfrak{s}[\mathbf{x} \leftarrow \mathbf{l}] \stackrel{\text{def}}{=} \mathfrak{s}[x_1 \leftarrow l_1] \ldots [x_n \leftarrow l_n]$ an $\mathbf{x}$-*associate* of $\mathfrak{s}$. A structure $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi$, is called a *model* of ϕ. Note that $(\mathfrak{s}, \mathfrak{h}) \models \phi$ only if $\mathsf{fv}(\phi) \subseteq \mathrm{dom}(\mathfrak{s})$.

The fragment of *symbolic heaps* is obtained by confining the negation and conjunction to the formulæ $t_1 \doteq t_2 \stackrel{\text{def}}{=} t_1 \approx t_2 \wedge \mathsf{emp}$ and $t_1 \not\doteq t_2 \stackrel{\text{def}}{=} \neg t_1 \approx t_2 \wedge \mathsf{emp}$, called *equational atoms*, by abuse of language. We denote by $\mathsf{SH}^{\mathfrak{K}}$ the set of symbolic heaps, formally defined below:

$$\phi \quad := \quad \mathsf{emp} \mid t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) \mid p(t_1, \ldots, t_{\#p}) \mid t_1 \doteq t_2 \mid t_1 \not\doteq t_2 \mid \phi_1 * \phi_2 \mid \exists x \,.\, \phi_1$$

Given quantifier-free symbolic heaps $\phi_1, \phi_2 \in \mathsf{SH}^{\mathfrak{K}}$, it is not hard to check that $\exists x \,.\, \phi_1 * \exists y \,.\, \phi_2$ and $\exists x \exists y \,.\, \phi_1 * \phi_2$ have the same models (provided $x \neq y$). Consequently, each symbolic heap can be written in prenex form, as $\phi = \exists x_1 \ldots \exists x_n \,.\, \psi$, where ψ is a quantifier-free separating conjunction of points-to atoms and (dis-)equalities. A variable $x \in \mathsf{fv}(\psi)$ is *allocated* in ϕ iff there exists a (possibly empty) sequence of equalities $x \doteq \ldots \doteq t_0$ and a points-to atom $t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})$ in ψ.

The predicates from $\mathbb{P}$ are intepreted by a given set $\mathcal{S}$ of *rules* $p(x_1, \ldots, x_{\#p}) \Leftarrow \rho$, where ρ is a symbolic heap, such that $\mathsf{fv}(\rho) \subseteq \{x_1, \ldots, x_{\#_p}\}$. We say that $p(x_1, \ldots, x_{\#p})$ is the *head* and ρ is the *body* of the rule. For conciseness, we write $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$ instead of $p(x_1, \ldots, x_{\#p}) \Leftarrow \rho \in \mathcal{S}$. In the following, we shall often refer to a given set of rules $\mathcal{S}$.

▶ **Definition 1** (Unfolding). *A formula ψ is a* step-unfolding *of a formula $\phi \in \mathsf{SL}^{\mathfrak{K}}$, written $\phi \Rightarrow_{\mathcal{S}} \psi$, if ψ is obtained by replacing an occurrence of an atom $p(t_1, \ldots, t_{\#p})$ in ϕ with $\rho[t_1/x_1, \ldots, t_{\#p}/x_{\#p}]$, for a rule $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$. An* unfolding *of ϕ is a formula ψ such that $\phi \Rightarrow^*_{\mathcal{S}} \psi$.*

It is easily seen that any unfolding of a symbolic heap is again a symbolic heap. We implicitly assume that all bound variables are α-renamed throughout an unfolding, to avoid name clashes. Unfolding extends the semantics from predicate-free to arbitrary $\mathsf{SL}^{\mathfrak{K}}$ formulæ:

▶ **Definition 2.** *Given a structure $(\mathfrak{s}, \mathfrak{h})$ and a formula $\phi \in \mathsf{SL}^{\mathfrak{K}}$, we write $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} \phi$ iff there exists a predicate-free unfolding $\phi \Rightarrow^*_{\mathcal{S}} \psi$ such that $(\mathfrak{s}, \mathfrak{h}) \models \psi$. In this case, $(\mathfrak{s}, \mathfrak{h})$ is an* $\mathcal{S}$-model *of ϕ. For two formulæ $\phi, \psi \in \mathsf{SL}^{\mathfrak{K}}$, we write $\phi \models_{\mathcal{S}} \psi$ iff every $\mathcal{S}$-model of ϕ is an $\mathcal{S}$-model of ψ.*

Note that, if $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} \phi$, then $\mathrm{dom}(\mathfrak{s})$ might have to contain constants that do not occur in ϕ. For instance if $p(x) \Leftarrow_{\mathcal{S}} x \mapsto \mathsf{a}$ is the only rule with head $p(x)$, then any $\mathcal{S}$-model $(\mathfrak{s}, \mathfrak{h})$ must map a to some location, which is taken care of by the assumption $\mathbb{C} \subseteq \mathrm{dom}(\mathfrak{s})$, that applies to any store.

▶ **Definition 3** (Entailment). *Given symbolic heaps $\phi, \psi_1, \ldots, \psi_n$, such that ϕ is quantifier-free and $\mathsf{fv}(\phi) = \mathsf{fv}(\psi_1) = \ldots = \mathsf{fv}(\psi_n) = \emptyset$, the* sequent $\phi \vdash \psi_1, \ldots, \psi_n$ *is* valid for $\mathcal{S}$ *iff $\phi \models_{\mathcal{S}} \bigvee_{i=1}^{n} \psi_i$. An* entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ *consists of a set of rules $\mathcal{S}$ and a set Σ of sequents, asking whether each sequent in Σ is valid for $\mathcal{S}$.*

Note that we consider entailments between formulæ without free variables. This is not restrictive, since any free variable can be replaced by a constant from $\mathbb{C}$, with no impact on the validity status or the computational complexity of the problem. We silently assume that $\mathbb{C}$ contains enough constants to allow this replacement. For conciseness, we write $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$ for $\phi \vdash \psi_1, \ldots, \psi_n \in \Sigma$, where Σ is the set of sequents of $\mathcal{P}$. The following example shows an entailment problem asking whether the concatenation of two acyclic lists is again an acyclic list:

▶ **Example 4.** The entailment problem below consists of four rules, defining the predicates $\mathsf{ls}(x, y)$ and $\mathsf{sls}(x, y, z)$, respectively, and two sequents:

$$\begin{array}{rcl} \mathsf{ls}(x,y) & \Leftarrow & x \mapsto y * x \neq y \mid \exists v \ . \ x \mapsto v * \mathsf{ls}(v,y) * x \neq y \\ \mathsf{sls}(x,y,z) & \Leftarrow & x \mapsto y * x \neq y * x \neq z \mid \exists v \ . \ x \mapsto v * \mathsf{sls}(v,y,z) * x \neq y * x \neq z \end{array}$$

$$\mathsf{ls}(a,b) * \mathsf{ls}(b,c) \vdash \exists x \ . \ a \mapsto x * \mathsf{ls}(x,c) * a \neq c \qquad \mathsf{sls}(a,b,c) * \mathsf{ls}(b,c) \vdash \exists x \ . \ a \mapsto x * \mathsf{ls}(x,c) * a \neq c$$

Here $\mathsf{ls}(x, y)$ describes non-empty acyclic list segments with head and tail pointed to by x and y, respectively. The first sequent is invalid, because c can be allocated within the list segment defined by $\mathsf{ls}(a, b)$, in which case the entire list has a cycle starting and ending with the location associated with c. To avoid the cycle, the left-hand side of the second sequent uses the predicate $\mathsf{sls}(x, y, z)$ describing an acyclic list segment from x to y that skips the location pointed to by z. The second sequent is valid. ⌟

The complexity analysis of the decision procedure described in this paper relies on two parameters. First, the *width* of an entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is (roughly) the maximum among the sizes of the symbolic heaps occurring in $\mathcal{P}$ and the number of constants in $\mathbb{C}$. Second, the *size* of the entailment problem is (roughly) the number of symbols needed to represent it, namely:

$$\begin{array}{rcl} \mathrm{width}(\mathcal{P}) & \stackrel{\text{def}}{=} & \max\left(\{\mathrm{size}(\rho) + \#p \mid p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho\} \cup \{\mathrm{size}(\psi_i) \mid \psi_0 \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n\} \cup \{||\mathbb{C}||\}\right) \\ \mathrm{size}(\mathcal{P}) & \stackrel{\text{def}}{=} & \sum_{p(x_1,\ldots,x_{\#p}) \Leftarrow_{\mathcal{S}} \rho} (\mathrm{size}(\rho) + \#p) + \sum_{\psi_0 \vdash_{\mathcal{P}} \psi_1,\ldots,\psi_n} \sum_{i=1}^{n} \mathrm{size}(\psi_i) \end{array}$$

In the next section we give a transformation of entailment problems with a time complexity that is bounded by the product of the size and a simple exponential of the width of the input, such that, moreover, the width of the problem increases by a polynomial factor only. The latter is instrumental in proving the final 2-EXPTIME upper bound on the complexity of the entailment problem.

To alleviate the upcoming technical details, we make the following assumption:

▶ **Assumption 1.** *Distinct constants are always associated with distinct locations: for all stores* $\mathfrak{s}$*, and for all* $c, d \in \mathbb{C}$*, we have* $c \neq d$ *only if* $\mathfrak{s}(c) \neq \mathfrak{s}(d)$*.*

This assumption loses no generality, because one can enumerate all the equivalence relations on $\mathbb{C}$ and test the entailments separately for each of these relations, by replacing all the constants in the same class by a unique representative[2], while assuming that constants in distinct classes are mapped to distinct locations. The overall complexity of the procedure is still doubly exponential, since the number of such equivalence relations is bounded by the number of partitions of $\mathbb{C}$, that is $2^{\mathcal{O}(||\mathbb{C}|| \cdot \log ||\mathbb{C}||)} = 2^{\mathcal{O}(||\mathrm{width}(\mathcal{P})|| \cdot \log ||\mathrm{width}(\mathcal{P})||)}$, for any entailment problem $\mathcal{P}$. Thanks to Assumption 1, the considered symbolic heaps can be, moreover, safely assumed not to contain atoms $c \bowtie d$, with $\bowtie \in \{\doteq, \neq\}$ and $c, d \in \mathbb{C}$, since these atoms are either unsatisfiable or equivalent to emp.

3 Decidable Classes of Entailments

In general, the entailment problem (Definition 3) is undecidable and we refer the reader to [9, 1] for two different proofs. A first attempt to define a naturally expressive class of formulæ with a decidable entailment problem was reported in [8]. The entailments considered in [8] involve sets of rules restricted by three conditions, recalled below, in a slightly generalized form.

[2] The replacement must be performed also within the inductive rules, not only in the considered formulæ.

First, the *progress* condition requires that each rule adds to the heap exactly one location, associated either to a constant or to a designated parameter. Formally, we consider a mapping $\mathsf{root} : \mathbb{P} \to \mathbb{N} \cup \mathbb{C}$, such that $\mathsf{root}(p) \in [\![1 \,..\, \#p]\!] \cup \mathbb{C}$, for each $p \in \mathbb{P}$. The term $\mathsf{root}(p(t_1, \ldots, t_{\#p}))$ denotes either t_i if $\mathsf{root}(p) = i \in [\![1 \,..\, \#p]\!]$, or the constant $\mathsf{root}(p)$ itself if $\mathsf{root}(p) \in \mathbb{C}$. The notation $\mathsf{root}(\alpha)$ is extended to points-to atoms α as $\mathsf{root}(t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})) \stackrel{\text{def}}{=} t_0$. Second, the *connectivity* condition requires that all locations added during an unfolding of a predicate atom form a set of connected trees (a forest) rooted in locations associated either with a parameter of the predicate or with a constant.

▶ **Definition 5** (Progress & Connectivity). *A set of rules $\mathcal{S}$ is* progressing *if each rule in $\mathcal{S}$ is of the form $p(x_1, \ldots, x_{\#p}) \Leftarrow \exists z_1 \ldots \exists z_m \,.\, \mathsf{root}(p(x_1, \ldots, x_{\#p})) \mapsto (t_1, \ldots, t_{\mathfrak{K}}) * \psi$ and ψ contains no occurrences of points-to atoms. Moreover, $\mathcal{S}$ is* connected *if $\mathsf{root}(q(u_1, \ldots, u_{\#q})) \in \{t_1, \ldots, t_{\mathfrak{K}}\} \cup \mathbb{C}$, for each predicate atom $q(u_1, \ldots, u_{\#q})$ occurring in ψ. An entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is progressing (connected) if $\mathcal{S}$ is progressing (connected).*

The progress and connectivity conditions can be checked in polynomial time by a syntactic inspection of the rules in $\mathcal{S}$, even if the $\mathsf{root}(.)$ function is not known a priori. Note that this definition of connectivity is less restrictive that the definition from [8], that asked for $\mathsf{root}(q(u_1, \ldots, u_{\#q})) \in \{t_1, \ldots, t_{\mathfrak{K}}\}$. For instance, the set of rules $\{\mathsf{p}(x) \Leftarrow \exists y \,.\, x \mapsto y * \mathsf{p}(y) * \mathsf{p}(\mathsf{c}), \mathsf{p}(x) \Leftarrow x \mapsto \mathsf{nil}\}$, where $\mathsf{c} \in \mathbb{C}$ is progressing and connected (with $\mathsf{root}(\mathsf{p}) = 1$) in the sense of Definition 5, but not connected in the sense of [8], because $\mathsf{c} \notin (y)$. Note also that nullary predicate symbols are allowed, for instance $\mathsf{q}() \Leftarrow \mathsf{c} \mapsto \mathsf{nil}$ is progressing and connected (with $\mathsf{root}(\mathsf{q}) = \mathsf{c}$). Further, the entailment problem from Example 4 is both progressing and connected.

Third, the *establishment* condition is defined, slightly extended from its original statement [8]:

▶ **Definition 6** (Establishment). *Given a set of rules $\mathcal{S}$, a symbolic heap $\exists x_1 \ldots \exists x_n \,.\, \phi$, where ϕ is quantifier-free, is* $\mathcal{S}$-established *iff every x_i for $i \in [\![1 \,..\, n]\!]$ is allocated in each predicate-free unfolding $\phi \Rightarrow^*_{\mathcal{S}} \varphi$. A set of rules $\mathcal{S}$ is* established *if the body ρ of each rule $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$ is $\mathcal{S}$-established. An entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is established if $\mathcal{S}$ is established, and* strongly established *if, moreover, ϕ_i is $\mathcal{S}$-established, for each sequent $\phi_0 \vdash_{\mathcal{P}} \phi_1, \ldots, \phi_n$ and each $i \in [\![0 \,..\, n]\!]$.*

For example, the entailment problem from Example 4 is strongly established.

In this paper, we replace establishment with a new condition that, as we show, preserves the decidability and computational complexity of progressing, connected and established entailment problems. The new condition can be checked in time linear in the size of the problem. This condition, called *equational restrictedness* (*e-restrictedness*, for short), requires that each equational atom occurring in a formula involves at least one constant. We will show that the e-restrictedness condition is more general than establishment, in the sense that every established problem can be reduced to an equivalent e-restricted problem (Theorem 13). Moreover, the class of structures defined using e-restricted symbolic heaps is a strict superset of the one defined by established symbolic heaps.

▶ **Definition 7** (E-restrictedness). *A symbolic heap ϕ is* e-restricted *if, for every equational atom $t \bowtie u$ from ϕ, where $\bowtie \in \{\doteq, \neq\}$, we have $\{t, u\} \cap \mathbb{C} \neq \emptyset$. A set of rules $\mathcal{S}$ is* e-restricted *if the body ρ of each rule $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$ is e-restricted. An entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is* e-restricted *if $\mathcal{S}$ is e-restricted and ϕ_i is e-restricted, for each sequent $\phi_0 \vdash_{\mathcal{P}} \phi_1, \ldots, \phi_n$ and each $i \in [\![0 \,..\, n]\!]$.*

For instance, the entailment problem from Example 4 is not e-restricted, because several rule bodies have disequalities between parameters, e.g. $\mathsf{ls}(x,y) \Leftarrow x \mapsto y * x \neq y$. However, the set of rules $\{\mathsf{ls}_{\mathsf{c}}(x) \Leftarrow x \mapsto c * x \neq \mathsf{c}, \mathsf{ls}_{\mathsf{c}}(x) \Leftarrow \exists y \,.\, x \mapsto y * \mathsf{ls}_{\mathsf{c}}(y) * x \neq \mathsf{c}\}$, where $\mathsf{c} \in \mathbb{C}$ and ls_{c} is a new predicate symbol, denoting an acyclic list ending with c, is e-restricted. Note that any atom $\mathsf{ls}(x,y)$ can be replaced by $\mathsf{ls}_{\mathsf{y}}(x)$, provided that y occurs free in a sequent and can be viewed as a constant.

We show next that every established entailment problem (Definition 6) can be reduced to an e-restricted entailment problem (Definition 7). The transformation incurs an exponential blowup, however, as we show, the blowup is exponential only in the width and polynomial in the size of the input problem. This is to be expected, because checking e-restrictedness of a problem can be done in linear time, in contrast with checking establishment, which is at least co-NP-hard [11].

We begin by showing that each problem can be translated into an equivalent *normalized* problem:

▶ **Definition 8** (Normalization)**.**

(1) *A symbolic heap $\exists \mathbf{x} \,.\, \psi \in \mathsf{SH}^{\mathfrak{K}}$, where ψ is quantifier-free, is* normalized *iff for every atom α in ψ:*

- **a.** *if α is an equational atom, then it is of the form $x \neq t$ ($t \neq x$), where $x \in \mathbf{x}$,*
- **b.** *every variable $x \in \mathsf{fv}(\psi)$ occurs in a points-to or predicate atom of ψ,*
- **c.** *if α is a predicate atom $q(t_1, \ldots, t_{\#q})$, then $\{t_1, \ldots, t_{\#q}\} \cap \mathbb{C} = \emptyset$ and $t_i \neq t_j$, for all $i \neq j \in [\![1 \,..\, \#q]\!]$.*

(2) *A set of rules $\mathcal{S}$ is* normalized *iff for each rule $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$, the symbolic heap ρ is normalized and, moreover:*

- **a.** *For every $i \in [\![1 \,..\, \#p]\!]$ and every predicate-free unfolding $p(x_1, \ldots, x_{\#p}) \Rightarrow^*_{\mathcal{S}} \varphi$, φ contains a points-to atom $t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})$, such that $x_i \in \{t_0, \ldots, t_{\mathfrak{K}}\}$.*
- **b.** *There exist sets $\mathsf{palloc}_{\mathcal{S}}(p) \subseteq [\![1 \,..\, \#p]\!]$ and $\mathsf{calloc}_{\mathcal{S}}(p) \subseteq \mathbb{C}$ such that, for each predicate-free unfolding $p(x_1, \ldots, x_{\#p}) \Rightarrow^*_{\mathcal{S}} \varphi$:*
  - *$i \in \mathsf{palloc}_{\mathcal{S}}(p)$ iff φ contains an atom $x_i \mapsto (t_1, \ldots, t_{\mathfrak{K}})$, for every $i \in [\![1 \,..\, \#p]\!]$,*
  - *$c \in \mathsf{calloc}_{\mathcal{S}}(p)$ iff φ contains an atom $c \mapsto (t_1, \ldots, t_{\mathfrak{K}})$, for every $c \in \mathbb{C}$.*
- **c.** *For every predicate-free unfolding $p(x_1, \ldots, x_{\#p}) \Rightarrow^*_{\mathcal{S}} \varphi$, if φ contains an atom $t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})$ such that $t_0 \in \mathbb{V} \setminus \{x_1, \ldots, x_{\#p}\}$, then φ also contains atoms $t_0 \neq c$, for every $c \in \mathbb{C}$.*

(3) *An entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ is* normalized *if $\mathcal{S}$ is normalized and, for each sequent $\phi_0 \vdash_{\mathcal{P}} \phi_1, \ldots, \phi_n$ the symbolic heap ϕ_i is normalized, for each $i \in [\![0 \,..\, n]\!]$.*

The intuition behind Condition (2a) is that no term can "disappear" while unfolding an inductive definition. Condition (2b) states that the set of terms eventually allocated by a predicate atom is the same in all unfoldings. This allows to define the set of symbols that occur freely in a symbolic heap ϕ and are necessarily allocated in every unfolding of ϕ, provided that the set of rules is normalized:

▶ **Definition 9.** *Given a normalized set of rules $\mathcal{S}$ and a symbolic heap $\phi \in \mathsf{SH}^{\mathfrak{K}}$, the set $\mathsf{alloc}_{\mathcal{S}}(\phi)$ is defined recursively on the structure of ϕ:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{alloc}_{\mathcal{S}}(t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})) & \stackrel{\text{def}}{=} & \{t_0\} & \mathsf{alloc}_{\mathcal{S}}(p(t_1, \ldots, t_{\#p})) & \stackrel{\text{def}}{=} & \{t_i \mid i \in \mathsf{palloc}_{\mathcal{S}}(p)\} \\
\mathsf{alloc}_{\mathcal{S}}(t_1 \bowtie t_2) & \stackrel{\text{def}}{=} & \emptyset,\ \bowtie \in \{\doteq, \neq\} & & & \cup\ \mathsf{calloc}_{\mathcal{S}}(p) \\
\mathsf{alloc}_{\mathcal{S}}(\phi_1 * \phi_2) & \stackrel{\text{def}}{=} & \mathsf{alloc}_{\mathcal{S}}(\phi_1) \cup \mathsf{alloc}_{\mathcal{S}}(\phi_2) & \mathsf{alloc}_{\mathcal{S}}(\exists x \,.\, \phi_1) & \stackrel{\text{def}}{=} & \mathsf{alloc}_{\mathcal{S}}(\phi_1) \setminus \{x\}
\end{array}$$

▶ **Example 10.** The rules $p(x,y) \Leftarrow \exists z \,.\, x \mapsto z * p(z,y) * x \neq y$ and $p(x,y) \Leftarrow \exists z \,.\, x \mapsto z$ are not normalized, because they contradict Conditions (1a) and (2a) of Definition 8, respectively. A set $\mathcal{S}$ containing the rules $q(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q(y,z)$ and $q(x,y) \Leftarrow x \mapsto y$ is not normalized, because it is not possible to find a set $\mathsf{palloc}_{\mathcal{S}}(q)$ satisfying Condition (2b). Indeed, if $2 \in \mathsf{palloc}_{\mathcal{S}}(q)$ then the required equivalence does not hold for the second rule (because it does not allocate y), and if $2 \notin \mathsf{palloc}_{\mathcal{S}}(q)$ then it fails for the first one (since the predicate $q(y,z)$ allocates y). On the other hand, $\mathcal{S}' = \{p(x,y) \Leftarrow \exists z \,.\, x \mapsto z * p(z,y) * z \neq x * z \neq \mathsf{nil}, p(x,y) \Leftarrow x \mapsto y, q(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q(y,z) * z \neq \mathsf{nil}\ ,\ q(x,y) \Leftarrow x \mapsto y * r(y), r(x) \Leftarrow x \mapsto \mathsf{nil}\}$ is normalized (assuming $\mathbb{C} = \{\mathsf{nil}\}$), with $\mathsf{palloc}_{\mathcal{S}'}(p) = \mathsf{palloc}_{\mathcal{S}'}(r) = \{1\}$, $\mathsf{palloc}_{\mathcal{S}'}(q) = \{1,2\}$ and $\mathsf{calloc}_{\mathcal{S}'}(\pi) = \emptyset$, for all $\pi \in \{p,q,r\}$. Then $\mathsf{alloc}_{\mathcal{S}'}(p(x_1,x_2) * q(x_3,x_4) * r(x_5)) = \{x_1,x_3,x_4,x_5\}$. ⌟

The following lemma states that every entailment problem can be transformed into a normalized entailment problem, by a transformation that preserves progress, connectivity, e-restricted-ness and (strong) establishment.

▶ **Lemma 11.** *A progressing and connected entailment problem $\mathcal{P}$ can be translated to an equivalent progressing, connected and normalized problem $\mathcal{P}_n$, such that* $\mathrm{width}(\mathcal{P}_n) = \mathcal{O}(\mathrm{width}(\mathcal{P})^2)$ *in time* $\mathrm{size}(\mathcal{P}) \cdot 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^2)}$*. Further, $\mathcal{P}_n$ is e-restricted if $\mathcal{P}$ is e-restricted and (strongly) established if $\mathcal{P}$ is (strongly) established.*

▶ **Example 12.** The entailment problem $\mathcal{P} = (\mathcal{S},\ \{p(\mathsf{a},\mathsf{b}) \vdash \exists x,y \,.\, q(x,y)\})$ with:

$$\mathcal{S} \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} p(x,y) \Leftarrow \exists z \,.\, x \mapsto z * p(z,y) * x \neq y & q(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q(y,z) * z \neq \mathsf{a} * z \neq \mathsf{b} \\ p(x,y) \Leftarrow \exists z \,.\, x \mapsto z & q(x,y) \Leftarrow x \mapsto y \end{array} \right\}$$

may be transformed into $(\mathcal{S}',\ \{p_1() \vdash \exists x,y \,.\, q_1(x,y), \exists x,y \,.\, q_2(x,y)\})$, where $\mathcal{S}'$ is the set:

$$\begin{array}{ll} p_1() \Leftarrow \exists z \,.\, \mathsf{a} \mapsto z * p_2(z) * z \neq \mathsf{a} * z \neq \mathsf{b} & p_2(x) \Leftarrow x \mapsto \mathsf{b} * p_3() \\ p_1() \Leftarrow \mathsf{a} \mapsto \mathsf{b} * p_3() & p_2(x) \Leftarrow \exists z \,.\, x \mapsto z * p_2(z) * z \neq \mathsf{a} * z \neq \mathsf{b} \\ p_1() \Leftarrow \exists z \,.\, \mathsf{a} \mapsto z & p_2(x) \Leftarrow \exists z \,.\, x \mapsto z \\ p_3() \Leftarrow \exists z \,.\, \mathsf{b} \mapsto z & q_1(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q_1(y,z) * z \neq \mathsf{a} * z \neq \mathsf{b} \\ q_2(x,y) \Leftarrow x \mapsto y & q_1(x,y) \Leftarrow \exists z \,.\, x \mapsto y * q_2(y,z) * z \neq \mathsf{a} * z \neq \mathsf{b} \end{array}$$

The predicate atoms $p_1(), p_2(x)$ and $p_3()$ are equivalent to $p(\mathsf{a},\mathsf{b})$, $p(x,\mathsf{b})$ and $p(\mathsf{b},\mathsf{b})$, respectively. $q(x,y)$ is equivalent to $q_1(x,y) \vee q_2(x,y)$. Note that $p_2(x)$ is only used in a context where $x \neq b$ holds, thus the atom $x \neq b$ may be omitted from the rules of $p_2()$. Recall that a and b are mapped to distinct locations, by Assumption 1. ⌟

We show that every established problem $\mathcal{P}$ can be reduced to an e-restricted problem in time linear in the size and exponential in the width of the input, at the cost of a polynomial increase of its width:

▶ **Theorem 13.** *Every progressing, connected and established entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$ can be reduced in time* $\mathrm{size}(\mathcal{P}) \cdot 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^2)}$ *to a normalized, progressing, connected and e-restricted problem $\mathcal{P}_r$, such that* $\mathrm{width}(\mathcal{P}_r) = \mathcal{O}(\mathrm{width}(\mathcal{P}))$.

The class of e-restricted problems is more general than the class of established problems, in the following sense: for each established problem $\mathcal{P} = (\mathcal{S}, \Sigma)$, the treewidth of each $\mathcal{S}$-model of a $\mathcal{S}$-established symbolic heap ϕ is bounded by $\mathrm{width}(\mathcal{P})$ [8], while e-restricted symbolic heaps may have infinite sequences of models with strictly increasing treewidth:

▶ **Example 14.** Consider the set of rules $\{\mathsf{lls}(x,y) \Leftarrow x \mapsto (y,\mathsf{nil}),\ \mathsf{lls}(x,y) \Leftarrow \exists z \exists v \,.\, x \mapsto (z,v) * \mathsf{lls}(z,y)\}$. The existentially quantified variable v in the second rule in never allocated in any predicate-free unfolding of $\mathsf{lls}(\mathsf{a},\mathsf{b})$, thus the set of rules is not established. However,

it is trivially e-restricted, because no equational atoms occur within the rules. Among the models of $\mathsf{lls}(\mathsf{a},\mathsf{b})$, there are all $n \times n$-square grid structures, known to have treewidth n, for $n > 1$ [17] (such a grid can be represented as a list of length n^2, with additional links between the elements at positions i and $i+n$). ⌟

4 Normal Structures

The decidability of e-restricted entailment problems relies on the fact that, to prove the validity of a sequent, it is sufficient to consider only a certain class of structures, called *normal*, that require the variables not mapped to the same location as a constant to be mapped to pairwise distinct locations:

▶ **Definition 15.** *A structure* $(\mathfrak{s},\mathfrak{h})$ *is a* normal $\mathcal{S}$-model *of a symbolic heap* ϕ *iff there exists:*

1. *a predicate-free unfolding* $\phi \Rightarrow_{\mathcal{S}} \exists \mathbf{x} \,.\, \psi$*, where* ψ *is quantifier-free, and*
2. *an* $\mathbf{x}$*-associate* $\bar{\mathfrak{s}}$ *of* $\mathfrak{s}$*, such that* $(\bar{\mathfrak{s}},\mathfrak{h}) \models_{\mathcal{S}} \psi$ *and* $\bar{\mathfrak{s}}(x) = \bar{\mathfrak{s}}(y) \wedge x \neq y \Rightarrow \bar{\mathfrak{s}}(x) \in \mathfrak{s}(\mathbb{C})$*, for all* $x, y \in \mathsf{fv}(\psi)$*.*

▶ **Example 16.** Consider the formula $\varphi = p(x_1) * p(x_2)$, with $p(x) \Leftarrow_{\mathcal{S}} \exists z \,.\, x \mapsto z$ and $\mathbb{C} = \{\mathsf{a}\}$. Then the structures: $(\mathfrak{s},\mathfrak{h})$ and $(\mathfrak{s},\mathfrak{h}')$ with $\mathfrak{s} = \{(x_1,\ell_1),(x_2,\ell_2),(\mathsf{a},\ell_3)\}$, $\mathfrak{h} = \{(\ell_1,\ell_3),(\ell_2,\ell_3)\}$ and $\mathfrak{h}' = \{(\ell_1,\ell_4),(\ell_2,\ell_5)\}$ are normal models of φ. On the other hand, if $\mathfrak{h}'' = \{(\ell_1,\ell_4),(\ell_2,\ell_4)\}$ (with $\ell_4 \neq \ell_3$) then $(\mathfrak{s},\mathfrak{h}'')$ is a model of φ but it is not normal, because any associate of $\mathfrak{s}$ will map the existentials from the predicate-free unfolding of $p(x_1) * p(x_2)$ into the same location, different from $\mathfrak{s}(\mathsf{a})$. ⌟

Since the left-hand side symbolic heap ϕ of each sequent $\phi \vdash \psi_1, \ldots, \psi_n$ is quantifier-free and has no free variables (Definition 3) and moreover, by Assumption 1, every constant is associated a distinct location, to check the validity of a sequent it is enough to consider only structures with injective stores. We say that a structure $(\dot{\mathfrak{s}},\mathfrak{h})$ is *injective* if the store $\dot{\mathfrak{s}}$ is injective. As a syntactic convention, by stacking a dot on the symbol denoting the store, we mean that the store is injective.

The key property of normal structures is that validity of e-restricted entailment problems can be checked considering only (injective) normal structures. The intuition is that, since the (dis-)equalities occurring in the considered formula involve a constant, it is sufficient to assume that all the existential variables not equal to a constant are mapped to pairwise distinct locations, as all other structures can be obtained from such structures by applying a morphism that preserves the truth value of the considered formulæ.

▶ **Lemma 17.** *Let* $\mathcal{P} = (\mathcal{S},\Sigma)$ *be a normalized and e-restricted entailment problem and let* $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$ *be a sequent. Then* $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$ *is valid for* $\mathcal{S}$ *iff* $(\dot{\mathfrak{s}},\mathfrak{h}) \models_{\mathcal{S}} \bigvee_{i=1}^{n} \psi_i$*, for each normal injective* $\mathcal{S}$*-model* $(\dot{\mathfrak{s}},\mathfrak{h})$ *of* ϕ*.*

5 Core Formulæ

Given an e-restricted entailment problem $\mathcal{P} = (\mathcal{S},\Sigma)$, the idea of the entailment checking algorithm is to compute, for each symbolic heap ϕ that occurs as the left-hand side of a sequent $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$, a finite set of sets of formulæ $\mathcal{F}(\phi) = \{F_1, \ldots, F_m\}$, of some specific pattern, called *core formulæ*. The set $\mathcal{F}(\phi)$ defines an equivalence relation, of finite index, on the set of injective normal $\mathcal{S}$-models of ϕ, such that each set $F \in \mathcal{F}(\phi)$ encodes an equivalence class. Because the validity of each sequent can be checked by testing whether every (injective) normal model of its left-hand side is a model of some symbolic heap on

the right-hand side (Lemma 17), an equivalent check is that each set $F \in \mathcal{F}(\phi)$ contains a core formula entailing some formula ψ_i, for $i = 1, \ldots, n$. To improve the presentation, we first formalize the notions of core formulæ and abstractions by sets of core formulæ, while deferring the effective construction of $\mathcal{F}(\phi)$, for a symbolic heap ϕ, to the next section (§6). In the following, we refer to a given entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$.

First, we define core formulæ as a fragment of $\mathsf{SL}^{\mathfrak{K}}$. Consider a formula $\mathsf{loc}(x) \stackrel{\text{def}}{=} \exists y_0 \ldots \exists y_{\mathfrak{K}} \ . \ y_0 \mapsto (y_1, \ldots, y_{\mathfrak{K}}) * \bigvee_{i=0}^{\mathfrak{K}} x \approx y_i$. Note that a structure is a model of $\mathsf{loc}(x)$ iff the variable x is mapped to a location from the domain or the range of the heap. We define also the following bounded quantifiers:

$$\dot{\exists} x \ . \ \phi \stackrel{\text{def}}{=} \exists x \ . \ \bigwedge_{t \in (\mathsf{fv}(\phi) \setminus \{x\}) \cup \mathbb{C}} \neg x \approx t \wedge \phi \qquad \exists_{\mathsf{h}} x \ . \ \phi \stackrel{\text{def}}{=} \dot{\exists} x \ . \ \mathsf{loc}(x) \wedge \phi$$
$$\exists_{\neg \mathsf{h}} x \ . \ \phi \stackrel{\text{def}}{=} \dot{\exists} x \ . \ \neg \mathsf{loc}(x) \wedge \phi \qquad \forall_{\neg \mathsf{h}} x \ . \ \phi \stackrel{\text{def}}{=} \neg \exists_{\neg \mathsf{h}} x \ . \ \neg \phi$$

In the following, we shall be extensively using the $\exists_{\mathsf{h}} x \ . \ \phi$ and $\forall_{\neg \mathsf{h}} x \ . \ \phi$ quantifiers. The formula $\exists_{\mathsf{h}} x \ . \ \phi$ states that there exists a location ℓ which occurs in the domain or range of the heap and is distinct from the locations associated with the constants and free variables, such that ϕ holds when x is associated with ℓ. Similarly, $\forall_{\neg \mathsf{h}} x \ . \ \phi$ states that ϕ holds if x is associated with any location ℓ that is outside of the heap and distinct from all the constants and free variables. The use of these special quantifiers will allow us to restrict ourselves to injective stores (since all variables and constants are mapped to distinct locations), which greatly simplifies the handling of equalities.

The main ingredient used to define core formulæ are *context predicates*. Given a tuple of predicate symbols $(p, q_1, \ldots, q_n) \in \mathbb{P}^{n+1}$, where $n \geq 0$, we consider a context predicate symbol $\Gamma_{p,q_1,\ldots,q_n}$ of arity $\#p + \sum_{i=1}^{n} \#q_i$. The informal intuition of a context predicate atom $\Gamma_{p,q_1,\ldots,q_n}(\mathbf{t}, \mathbf{u}_1, \ldots, \mathbf{u}_n)$ is the following: a structure $(\mathfrak{s}, \mathfrak{h})$ is a model of this atom if there exist models $(\mathfrak{s}, \mathfrak{h}_i)$ of $q_i(\mathbf{u}_i)$, $i \in [\![1 \ .. \ n]\!]$ respectively, with mutually disjoint heaps, an unfolding ψ of $p(\mathbf{t})$ in which the atoms $q_i(\mathbf{u}_i)$ occur, and an associate $\mathfrak{s}'$ of $\mathfrak{s}$ such that $(\mathfrak{s}', \mathfrak{h} \uplus \biguplus_{i=1}^{n} \mathfrak{h}_i)$ is a model of ψ.

For readability's sake, we adopt a notation close in spirit to SL's separating implication (known as the magic wand), and we write $\circledast_{i=1}^{n} q_i(\mathbf{y}_i) \mathrel{-\!\!\bullet} p(\mathbf{x})$ for $\Gamma_{p,q_1,\ldots,q_n}(\mathbf{x}, \mathbf{y}_1, \ldots, \mathbf{y}_n)$ and $\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{x})$, when $n = 0$[3]. The set of rules defining the interpretation of context predicates is the least set defined by the inference rules below, denoted $\mathfrak{C}_{\mathcal{S}}$:

$$\frac{}{p(\mathbf{x}) \mathrel{-\!\!\bullet} p(\mathbf{y}) \Leftarrow_{\mathfrak{C}_{\mathcal{S}}} \mathbf{x} \doteq \mathbf{y}} \ \mathbf{x} \cap \mathbf{y} = \emptyset \qquad \text{(I)}$$

$$\frac{p(\mathbf{x}) \Leftarrow_{\mathcal{S}} \exists \mathbf{z} \ . \ \psi * \circledast_{j=1}^{m} p_j(\mathbf{w}_j) \quad \circledast_{i=1}^{n} q_i(\mathbf{y}_i) = \circledast_{j=1}^{m} \gamma_j}{\circledast_{i=1}^{n} q_i(\mathbf{y}_i) \mathrel{-\!\!\bullet} p(\mathbf{x}) \Leftarrow_{\mathfrak{C}_{\mathcal{S}}} \exists \mathbf{v} \ . \ \psi\sigma * \circledast_{j=1}^{m} (\gamma_j \mathrel{-\!\!\bullet} p_j(\sigma(\mathbf{w}_j)))} \ \begin{array}{l} \mathbf{x}, \mathbf{z}, \mathbf{y}_1, \ldots, \mathbf{y}_n \text{ pairwise disjoint} \\ \sigma : \mathbf{z} \rightharpoonup \mathbf{x} \cup \bigcup_{i=1}^{n} \mathbf{y}_i \\ \mathbf{v} = \mathbf{z} \setminus \mathrm{dom}(\sigma) \end{array} \qquad \text{(II)}$$

Note that $\mathfrak{C}_{\mathcal{S}}$ is not progressing, since the rule for $p(\mathbf{x}) \mathrel{-\!\!\bullet} p(\mathbf{y})$ does not allocate any location. However, if $\mathcal{S}$ is progressing, then the set of rules obtained by applying (II) only is

[3] Context predicates are similar to the *strong magic wand* introduced in [13]. A context predicate $\alpha \mathrel{-\!\!\bullet} \beta$ is also related to the usual separating implication $\alpha \mathrel{-\!\!*} \beta$ of separation logic, but it is not equivalent. Intuitively, $\mathrel{-\!\!*}$ represents a difference between two heaps, whereas $\mathrel{-\!\!\bullet}$ removes some atoms in an unfolding. For instance, if p and q are defined by the same inductive rules, up to a renaming of predicates, then $p(x) \mathrel{-\!\!*} q(x)$ always holds in a structure with an empty heap, whereas $p(x) \mathrel{-\!\!\bullet} q(x)$ holds if, moreover, $p(x)$ and $q(x)$ are the same atom.

also progressing. Rule (I) says that each predicate atom $p(\mathbf{t}) \multimap p(\mathbf{u})$, such that $\mathbf{t}$ and $\mathbf{u}$ are mapped to the same tuple of locations, is satisfied by the empty heap. To understand rule (II), let $(\mathfrak{s}, \mathfrak{h})$ be an $\mathcal{S}$-model of $p(\mathbf{t})$ and assume there are a predicate-free unfolding ψ of $p(\mathbf{t})$ and an associate $\mathfrak{s}'$ of $\mathfrak{s}$, such that $q_1(\mathbf{u}_1), \ldots, q_n(\mathbf{u}_n)$ occur in ψ and $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{S}} \psi$. If the first unfolding step is an instance of a rule $p(\mathbf{x}) \Leftarrow_{\mathcal{S}} \exists \mathbf{z} \,.\, \psi * \mathop{\ast}_{j=1}^{m} p_j(\mathbf{w}_j)$ then there exist a $\mathbf{z}$-associate $\overline{\mathfrak{s}}$ of $\mathfrak{s}$ and a split of $\mathfrak{h}$ into disjoint heaps $\mathfrak{h}_0, \ldots, \mathfrak{h}_m$ such that $(\overline{\mathfrak{s}}, \mathfrak{h}_0) \models \psi[\mathbf{t}/\mathbf{x}]$ and $(\overline{\mathfrak{s}}, \mathfrak{h}_j) \models_{\mathcal{S}} p_j(\mathbf{w}_j)[\mathbf{t}/\mathbf{x}]$, for all $j \in [\![1 \,..\, m]\!]$. Assume, for simplicity, that $\mathbf{u}_1 \cup \ldots \cup \mathbf{u}_n \subseteq \mathrm{dom}(\overline{\mathfrak{s}})$ and let $\overline{\mathfrak{h}}_1, \ldots, \overline{\mathfrak{h}}_n$ be disjoint heaps such that $(\overline{\mathfrak{s}}, \overline{\mathfrak{h}}_i) \models_{\mathcal{S}} q_i(\mathbf{u}_i)$. Then there exists a partition $\big\{\{i_{j,1}, \ldots, i_{j,k_j}\} \mid j \in [\![1 \,..\, m]\!]\big\}$ of $[\![1 \,..\, n]\!]$, such that $\overline{\mathfrak{h}}_{i_{j,1}}, \ldots, \overline{\mathfrak{h}}_{i_{j,k_j}} \subseteq \mathfrak{h}_j$, for all $j \in [\![1 \,..\, m]\!]$. Let $\gamma_j \stackrel{\text{def}}{=} \mathop{\ast}_{\ell=1}^{k_j} q_\ell(\mathbf{u}_\ell)$, then $(\overline{\mathfrak{s}}, \mathfrak{h}_j \setminus (\overline{\mathfrak{h}}_{i_{j,1}} \cup \ldots \cup \overline{\mathfrak{h}}_{i_{j,k_j}})) \models_{\mathfrak{C}_{\mathcal{S}}} \gamma_j \multimap p_j(\mathbf{w}_j)[\mathbf{t}/\mathbf{x}]$, for each $j \in [\![1 \,..\, m]\!]$. This observation leads to the inductive definition of the semantics for $\mathop{\ast}_{i=1}^{n} q_i(\mathbf{u}_i) \multimap p(\mathbf{t})$, by the rule that occurs in the conclusion of (II), where the substitution $\sigma : \mathbf{z} \rightharpoonup \mathbf{x} \cup \bigcup_{i=1}^{n} \mathbf{y}_i$ is used to instantiate[4] some of the existentially quantified variables from the original rule $p(\mathbf{x}) \Leftarrow_{\mathcal{S}} \exists \mathbf{z} \,.\, \psi * \mathop{\ast}_{j=1}^{m} p_j(\mathbf{w}_j)$.

▶ **Example 18.** Consider the set $\mathcal{S} = \{p(x) \Leftarrow \exists z_1, z_2 \,.\, x \mapsto (z_1, z_2) * q(z_1) * q(z_2), q(x) \Leftarrow x \mapsto (x, x)\}$. We have $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} p(x)$ with $\mathfrak{s} = \{(x, \ell_1)\}$ and $\mathfrak{h} = \{(\ell_1, \ell_2, \ell_3), (\ell_2, \ell_2, \ell_2), (\ell_3, \ell_3, \ell_3)\}$. The atom $q(y) \multimap p(x)$ is defined by the following non-progressing rules (we only consider the rules corresponding to the case where σ is the identity, since the other rules are redundant):

$$q(y) \multimap p(x) \Leftarrow \exists z_1, z_2 \,.\, x \mapsto (z_1, z_2) * q(y) \multimap q(z_1) * \mathsf{emp} \multimap q(z_2)\ q(y) \multimap q(x) \Leftarrow x \doteq y$$
$$q(y) \multimap p(x) \Leftarrow \exists z_1, z_2 \,.\, x \mapsto (z_1, z_2) * \mathsf{emp} \multimap q(z_1) * q(y) \multimap q(z_2)\ \mathsf{emp} \multimap q(x) \Leftarrow x \mapsto (x, x)$$

The two rules for $q(y) \multimap p(x)$ correspond to the two ways of distributing $q(y)$ over $q(z_1)$, $q(z_2)$. We have $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, with $\mathfrak{h}_1 = \{(\ell_1, \ell_2, \ell_3), (\ell_2, \ell_2, \ell_2)\}$ and $\mathfrak{h}_2 = \{(\ell_3, \ell_3, \ell_3)\}$. It is easy to check that $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_1) \models_{\mathfrak{C}_{\mathcal{S}}} q(y) \multimap p(x)$, and $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_2) \models_{\mathfrak{C}_{\mathcal{S}}} q(y)$. Note that we also have $(\mathfrak{s}[y \leftarrow \ell_2], \mathfrak{h}'_1) \models_{\mathfrak{C}_{\mathcal{S}}} q(y) \multimap p(x)$, with $\mathfrak{h}'_1 = \{(\ell_1, \ell_2, \ell_3), (\ell_3, \ell_3, \ell_3)\}$. ⌟

Having introduced context predicates, the pattern of core formulæ is defined below:

▶ **Definition 19.** *A* core formula φ *is an instance of the pattern:*

$$\exists_{\mathsf{h}} \mathbf{x} \forall_{\neg \mathsf{h}} \mathbf{y} \,.\, \mathop{\ast}_{i=1}^{n} \left(\mathop{\ast}_{j=1}^{k_i} q_j^i(\mathbf{u}_j^i) \multimap p_i(\mathbf{t}_i) \right) * \mathop{\ast}_{i=n+1}^{m} t_0^i \mapsto (t_1^i, \ldots, t_{\mathfrak{K}}^i) \quad \textit{such that:}$$

- **(i)** *each variable occurring in* $\mathbf{y}$ *also occurs in an atom in* φ*;*
- **(ii)** *for every variable* $x \in \mathbf{x}$*, either* $x \in \mathbf{t}_i \setminus \bigcup_{i=1}^{k_i} \mathbf{u}_j^i$ *for some* $i \in [\![1 \,..\, n]\!]$*, or* $x = t_j^i$*, for some* $i \in [\![n+1 \,..\, m]\!]$ *and some* $j \in [\![0 \,..\, \mathfrak{K}]\!]$*;*
- **(iii)** *each term* t *occurs at most once as* $t = \mathsf{root}(\alpha)$*, where* α *is an atom of* φ*.*

We also define the set of terms $\mathrm{roots}(\varphi) \stackrel{\text{def}}{=} \mathrm{roots}_{\mathsf{lhs}}(\varphi) \cup \mathrm{roots}_{\mathsf{rhs}}(\varphi)$*, with* $\mathrm{roots}_{\mathsf{lhs}}(\varphi) \stackrel{\text{def}}{=} \{\mathsf{root}(q_j^i(\mathbf{u}_j^i)) \mid i \in [\![1 \,..\, n]\!], j \in [\![1 \,..\, k_i]\!]\}$ *and* $\mathrm{roots}_{\mathsf{rhs}}(\varphi) \stackrel{\text{def}}{=} \{\mathsf{root}(p_i(\mathbf{t}_i)) \mid i \in [\![1 \,..\, n]\!]\} \cup \{t_0^i \mid i \in [\![n+1 \,..\, m]\!]\}$.

Note that an unfolding of a core formula using the rules in $\mathfrak{C}_{\mathcal{S}}$ is not necessarily a core formula, because of the unbounded existential quantifiers and equational atoms that occur in the rules from $\mathfrak{C}_{\mathcal{S}}$. Note also that a core formula cannot contain an occurrence of a predicate of the form $p(\mathbf{t}) \multimap p(\mathbf{t})$ because otherwise, Condition (iii) of Definition 19 would be violated.

Lemma 20 shows that any symbolic heap is equivalent to an effectively computable finite disjunction of core formulæ, when the interpretation of formulæ is restricted to injective

[4] Note that this instantiation is, in principle, redundant (i.e. the same rules are obtained if $\mathrm{dom}(\sigma) = \emptyset$ by chosing appropriate $\mathbf{z}$-associates) but we keep it to simplify the related proofs.

structures. For a symbolic heap $\phi \in \mathsf{SH}^{\mathfrak{K}}$, we define the set $\mathcal{T}(\phi)$, recursively on the structure of ϕ, implicitly assuming w.l.o.g. that $\mathsf{emp} * \phi = \phi * \mathsf{emp} = \phi$:

$$\mathcal{T}(\mathsf{emp}) \stackrel{\text{def}}{=} \{\mathsf{emp}\} \qquad \mathcal{T}(t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})) \stackrel{\text{def}}{=} \{t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})\}$$

$$\mathcal{T}(p(\mathbf{t})) \stackrel{\text{def}}{=} \{\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t})\} \qquad \mathcal{T}(\phi_1 * \phi_2) \stackrel{\text{def}}{=} \{\psi_1 * \psi_2 \mid \psi_i \in \mathcal{T}(\phi_i),\ i = 1, 2\}$$

$$\mathcal{T}(t_1 \doteq t_2) \stackrel{\text{def}}{=} \begin{cases} \{\mathsf{emp}\} & \text{if } t_1 = t_2 \\ \emptyset & \text{if } t_1 \neq t_2 \end{cases} \qquad \mathcal{T}(t_1 \neq t_2) \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } t_1 = t_2 \\ \{\mathsf{emp}\} & \text{if } t_1 \neq t_2 \end{cases}$$

$$\mathcal{T}(\exists x \,.\, \phi_1) \stackrel{\text{def}}{=} \{\exists_{\mathsf{h}} x \,.\, \psi \mid \psi \in \mathcal{T}(\phi_1)\} \cup \{\psi \mid \psi \in \mathcal{T}(\phi_1[t/x]),\quad t \in (\mathsf{fv}(\phi_1) \setminus \{x\}) \cup \mathbb{C}\}$$

For instance, if $\phi = \exists x \,.\, p(x, y) * x \neq y$ and $\mathbb{C} = \{\mathsf{c}\}$, then $\mathcal{T}(\phi) = \{\exists_{\mathsf{h}} x \,.\, \mathsf{emp} \mathrel{-\!\!\bullet} p(x, y),\ \mathsf{emp} \mathrel{-\!\!\bullet} p(\mathsf{c}, y)\}$. Note that $\mathcal{T}(y \neq y) = \emptyset$, thus $\mathsf{emp} \mathrel{-\!\!\bullet} p(y, y) \notin \mathcal{T}(\phi)$.

▶ **Lemma 20.** *Assume $\mathcal{S}$ is normalized. Consider an e-restricted normalized symbolic heap $\phi \in \mathsf{SH}^{\mathfrak{K}}$ with no occurrences of context predicate symbols, and an injective structure $(\dot{\mathfrak{s}}, \mathfrak{h})$, such that $\mathrm{dom}(\dot{\mathfrak{s}}) = \mathsf{fv}(\phi) \cup \mathbb{C}$. We have $(\dot{\mathfrak{s}}, \mathfrak{h}) \models_{\mathcal{S}} \phi$ iff $(\dot{\mathfrak{s}}, \mathfrak{h}) \models_{\mathfrak{C}_{\mathcal{S}}} \psi$, for some $\psi \in \mathcal{T}(\phi)$.*

Next, we give an equivalent condition for the satisfaction of a context predicate atom, that relies on an unfolding of a symbolic heap into a core formula:

▶ **Definition 21.** *A formula φ is a* core unfolding *of a predicate atom $\circledast_{i=1}^{n} q_i(\mathbf{u}_i) \mathrel{-\!\!\bullet} p(\mathbf{t})$, written $\circledast_{i=1}^{n} q_i(\mathbf{u}_i) \mathrel{-\!\!\bullet} p(\mathbf{t}) \rightsquigarrow_{\mathfrak{C}_{\mathcal{S}}} \varphi$, iff there exists:*

1. *a rule $\circledast_{i=1}^{n} q_i(\mathbf{y}_i) \mathrel{-\!\!\bullet} p(\mathbf{x}) \Leftarrow_{\mathfrak{C}_{\mathcal{S}}} \exists \mathbf{z} \,.\, \phi$, where ϕ is quantifier free, and*
2. *a substitution $\sigma = [\mathbf{t}/\mathbf{x}, \mathbf{u}_1/\mathbf{y}_1, \ldots, \mathbf{u}_n/\mathbf{y}_n] \cup \zeta$, $\zeta \subseteq \{(z, t) \mid z \in \mathbf{z},\ t \in \mathbf{t} \cup \bigcup_{i=1}^{n} \mathbf{u}_i\}$, such that $\varphi \in \mathcal{T}(\phi\sigma)$.*

A core unfolding of a predicate atom is always a quantifier-free formula, obtained from the translation (into a disjunctive set of core formulæ) of the quantifier-free matrix of the body of a rule, in which some of the existentially quantified variables in the rule occur instantiated by the substitution σ. For instance, the rule $\mathsf{emp} \mathrel{-\!\!\bullet} p(x) \Leftarrow_{\mathfrak{C}_{\mathcal{S}}} \exists y \,.\, x \mapsto y$ induces the core unfoldings $\mathsf{emp} \mathrel{-\!\!\bullet} p(a) \rightsquigarrow_{\mathcal{S}} a \mapsto a$ and $\mathsf{emp} \mathrel{-\!\!\bullet} p(a) \rightsquigarrow_{\mathcal{S}} a \mapsto u$, via the substitutions $[a/x, a/y]$ and $[a/x, u/y]$, respectively. Note that a core unfolding of an atom ϕ may contain variables not occurring in ϕ, corresponding to the existential variables occurring in the rules, such as the variable u in the previous example.

We now define an equivalence relation, of finite index, on the set of injective structures. Intuitively, an equivalence class is defined by the set of core formulæ that are satisfied by all structures in the class (with some additional conditions). First, we introduce the overall set of core formulæ, over which these equivalence classes are defined:

▶ **Definition 22.** *Let $\mathcal{V}_{\mathcal{P}} \stackrel{\text{def}}{=} \mathcal{V}^1_{\mathcal{P}} \cup \mathcal{V}^2_{\mathcal{P}}$, such that $\mathcal{V}^1_{\mathcal{P}} \cap \mathcal{V}^2_{\mathcal{P}} = \emptyset$ and $||\mathcal{V}^i_{\mathcal{P}}|| = \mathrm{width}(\mathcal{P})$, for $i = 1, 2$ and denote by $\mathsf{Core}(\mathcal{P})$ the set of core formulæ φ such that $\mathrm{roots}(\varphi) \cap \mathsf{fv}(\varphi) \subseteq \mathcal{V}^1_{\mathcal{P}}$, $\mathrm{roots}(\varphi) \setminus \mathsf{fv}(\varphi) \subseteq \mathcal{V}^2_{\mathcal{P}} \cup \mathbb{C}$ and no variable in $\mathcal{V}^1_{\mathcal{P}}$ is bound in φ.*

Note that $\mathsf{Core}(\mathcal{P})$ is a finite set, because both $\mathcal{V}_{\mathcal{P}}$ and $\mathbb{C}$ are finite. Intuitively, $\mathcal{V}^1_{\mathcal{P}}$ will denote "local" variables introduced by unfolding the definitions on the left-hand sides of the entailments, whereas $\mathcal{V}^2_{\mathcal{P}}$ will denote existential variables occurring on the right-hand sides. The sets $\mathcal{V}^1_{\mathcal{P}}$ and $\mathcal{V}^2_{\mathcal{P}}$ can be chosen arbitrarily, provided the conditions of Definition 22 are satisfied. Second, we characterize an injective structure by the set of core formulæ it satisfies:

▶ **Definition 23.** *For a core formula $\varphi = \exists_{\mathsf{h}} \mathbf{x} \forall_{\neg \mathsf{h}} \mathbf{y} \,.\, \psi$, we denote by $\mathcal{W}_{\mathcal{S}}(\dot{\mathfrak{s}}, \mathfrak{h}, \varphi)$ the set of stores $\dot{\bar{\mathfrak{s}}}$ that are injective $(\mathbf{x} \cup \mathbf{y})$-associates of $\dot{\mathfrak{s}}$, and such that:*

(1) $(\dot{\bar{\mathfrak{s}}}, \mathfrak{h}) \models_{\mathfrak{C}_{\mathcal{S}}} \psi$,
(2) $\dot{\bar{\mathfrak{s}}}(\mathbf{x}) \subseteq \mathrm{loc}(\mathfrak{h})$, *and*
(3) $\dot{\bar{\mathfrak{s}}}(\mathbf{y}) \cap \mathrm{loc}(\mathfrak{h}) = \emptyset$.

The elements of this set are called witnesses for $(\dot{\mathfrak{s}}, \mathfrak{h})$ and φ.

The core abstraction *of an injective structure* $(\dot{\mathfrak{s}}, \mathfrak{h})$ *is the set* $\mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$ *of core formulæ* $\varphi \in \mathsf{Core}(\mathcal{P})$ *for which there exists a witness* $\dot{\bar{\mathfrak{s}}} \in \mathcal{W}_{\mathcal{S}}(\dot{\mathfrak{s}}, \mathfrak{h}, \varphi)$ *such that* $\dot{\bar{\mathfrak{s}}}(\mathrm{roots}_{\mathsf{lhs}}(\varphi)) \cap \mathrm{dom}(\mathfrak{h}) = \emptyset$.

An injective structure $(\dot{\mathfrak{s}}, \mathfrak{h})$ satisfies each core formula $\varphi \in \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$, a fact that is witnessed by an extension of the store assigning the universally quantified variables random locations outside of the heap. Further, any core formula φ such that $(\dot{\mathfrak{s}}, \mathfrak{h}) \models \varphi$ and $\mathrm{roots}_{\mathsf{lhs}}(\varphi) = \emptyset$ occurs in $\mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$.

Our entailment checking algorithm relies on the definition of the *profile* of a symbolic heap. Since each symbolic heap is equivalent to a finite disjunction of existential core formulæ, when interpreted over injective normal structures, it is sufficient to consider only profiles of core formulæ:

▶ **Definition 24.** *The* profile *of an entailment problem* $\mathcal{P} = (\mathcal{S}, \Sigma)$ *is the relation* $\mathcal{F} \subseteq \mathsf{Core}(\mathcal{P}) \times 2^{\mathsf{Core}(\mathcal{P})}$ *such that, for any core formula* $\phi \in \mathsf{Core}(\mathcal{P})$ *and any set of core formulæ* $F \in 2^{\mathsf{Core}(\mathcal{P})}$*, we have* $(\phi, F) \in \mathcal{F}$ *iff* $F = \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$*, for some injective normal* $\mathfrak{C}_{\mathcal{S}}$*-model* $(\dot{\mathfrak{s}}, \mathfrak{h})$ *of* ϕ*, with* $\mathrm{dom}(\dot{\mathfrak{s}}) = \mathsf{fv}(\phi) \cup \mathbb{C}$.

Assuming the existence of a profile, the effective construction of which will be given in Section 6, the following lemma provides an algorithm that decides the validity of $\mathcal{P}$:

▶ **Lemma 25.** *Let* $\mathcal{P} = (\mathcal{S}, \Sigma)$ *be a normalized e-restricted entailment problem and* $\mathcal{F} \subseteq \mathsf{Core}(\mathcal{P}) \times 2^{\mathsf{Core}(\mathcal{P})}$ *be a profile for* $\mathcal{P}$*. Then* $\mathcal{P}$ *is valid iff, for each sequent* $\phi \vdash_{\mathcal{P}} \psi_1, \ldots, \psi_n$*, each core formula* $\varphi \in \mathcal{T}(\phi)$ *and each pair* $(\varphi, F) \in \mathcal{F}$*, we have* $F \cap \mathcal{T}(\psi_i) \neq \emptyset$*, for some* $i \in [\![1 \,..\, n]\!]$.

The proof relies on Lemma 17, according to which entailments can be tested by considering only normal models. As one expects, Lemma 20 is used in this proof to ensure that the translation $\mathcal{T}(.)$ of symbolic heaps into core formulæ preserves the injective models.

6 Construction of the Profile Relation

For a given normalized entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$, we describe the construction of a profile $\mathcal{F}_{\mathcal{P}} \subseteq \mathsf{Core}(\mathcal{P}) \times 2^{\mathsf{Core}(\mathcal{P})}$, recursively on the structure of core formulæ. We assume that the set of rules $\mathcal{S}$ is progressing, connected and e-restricted. The relation $\mathcal{F}_{\mathcal{P}}$ is the least set satisfying the recursive constraints (1), (2), (3) and (4), given in this section. Since these recursive definitions are monotonic, the least fixed point exists and is unique.

Points-to Atoms. For a points-to atom $t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}})$, with $t_0, \ldots, t_{\mathfrak{K}} \in \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$, we have:

$$\begin{array}{c}(t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}),\ F) \in \mathcal{F}_{\mathcal{P}}, \text{ iff } F \text{ is the set containing } t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) \text{ and all core formulæ} \\ \text{of the form } \forall_{\neg \mathfrak{h}} \mathbf{z} \,.\, \circledast_{i=1}^{n} q_i(\mathbf{u}_i) \mathrel{-\!\!\bullet} p(\mathbf{t}) \in \mathsf{Core}(\mathcal{P}),\ \text{where } \mathbf{z} = (\mathbf{t} \cup \mathbf{u}_1 \cup \ldots \cup \mathbf{u}_n) \setminus (\{t_0, \ldots, t_{\mathfrak{K}}\} \cup \mathbb{C}) \\ \text{such that } \mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t}) \rightsquigarrow_{\mathfrak{C}_{\mathcal{S}}} t_0 \mapsto (t_1, \ldots, t_{\mathfrak{K}}) * \circledast_{i=1}^{n} \mathsf{emp} \mathrel{-\!\!\bullet} q_i(\mathbf{u}_i)\end{array} \tag{1}$$

For instance, if $\mathcal{S} = \{p(x) \Leftarrow \exists y, z \,.\, x \mapsto y * q(y, z),\ q(x, y) \Leftarrow x \mapsto y\}$, with $\mathcal{V}^1_{\mathcal{P}} = \{u, v\}$ and $\mathcal{V}^2_{\mathcal{P}} = \{z\}$, then $\mathcal{F}_{\mathcal{P}}$ contains the pair $(u \mapsto v, F)$ with $F = \{u \mapsto v, \mathsf{emp} \mathrel{-\!\!\bullet} q(u, v), \forall_{\neg \mathfrak{h}} z \,.\, q(v, z) \mathrel{-\!\!\bullet} p(u)\}$.

Predicate Atoms. Since profiles involve only the core formulæ obtained by the syntactic translation of a symbolic heap, the only predicate atoms that occur in the argument of a profile are of the form $\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t})$. We consider the constraint:

$$(\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t}),\ F) \in \mathcal{F}_{\mathcal{P}} \text{ if } (\exists_{\mathsf{h}}\mathbf{y}\ .\ \psi,\ F) \in \mathcal{F}_{\mathcal{P}}, \mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t}) \rightsquigarrow_{\mathfrak{C}_{\mathcal{S}}} \psi \in \mathsf{Core}(\mathcal{P}) \text{ and } \mathbf{y} = \mathsf{fv}(\psi) \setminus \mathbf{t} \quad (2)$$

Separating Conjunctions. Computing the profile of a separating conjunction is the most technical point of the construction. To ease the presentation, we assume the existence of a binary operation called *composition*:

▶ **Definition 26.** *Given a set $D \subseteq \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$, a binary operator $\circledast_D : 2^{\mathsf{Core}(\mathcal{P})} \times 2^{\mathsf{Core}(\mathcal{P})} \to 2^{\mathsf{Core}(\mathcal{P})}$ is a* composition *if $\mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}_1) \circledast_D \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}_2) = \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$, for any injective structure $(\dot{\mathfrak{s}}, \mathfrak{h})$, such that*

- **(i)** $\mathrm{dom}(\dot{\mathfrak{s}}) \subseteq \mathcal{V}^1_{\mathcal{P}}$,
- **(ii)** $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$,
- **(iii)** $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \subseteq \dot{\mathfrak{s}}(\mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C})$,
- **(iv)** $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \cap \mathrm{dom}(\mathfrak{h}) \subseteq \dot{\mathfrak{s}}(D) \subseteq \mathrm{dom}(\mathfrak{h})$.

We recall that $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) = \mathrm{loc}(\mathfrak{h}_1) \cap \mathrm{loc}(\mathfrak{h}_2)$. If $\mathcal{S}$ is a normalized set of rules, then for any core formula ϕ whose only occurrences of predicate atoms are of the form $\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t})$, we define $\mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi)$ as the homomorphic extension of $\mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\mathsf{emp} \mathrel{-\!\!\bullet} p(\mathbf{t})) \stackrel{\text{def}}{=} \mathsf{alloc}_{\mathcal{S}}(p(\mathbf{t}))$ to ϕ (see Definition 9). Assuming that $\mathcal{S}$ is a normalized set of rules and that a composition operation $\circledast_D$ (the construction of which will be described below, see Lemma 30) exists, we define the profile of a separating conjunction:

$$\begin{gathered}
(\phi_1 * \phi_2, \mathsf{add}(X_1, F_1) \circledast_D \mathsf{add}(X_2, F_2)) \in \mathcal{F}_{\mathcal{P}}, \text{ if } (\phi_i, F_i) \in \mathcal{F}_{\mathcal{P}}\ \ X_i \stackrel{\text{def}}{=} \mathsf{fv}(\phi_{3-i}) \setminus \mathsf{fv}(\phi_i),\ i = 1, 2 \\
\mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi_1) \cap \mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi_2) = \emptyset, \quad D \stackrel{\text{def}}{=} \mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi_1 * \phi_2) \cap (\mathsf{fv}(\phi_1) \cap \mathsf{fv}(\phi_2) \cup \mathbb{C}) \\
\mathsf{add}(x, F) \stackrel{\text{def}}{=} \{\exists_{\mathsf{h}}\mathbf{y} \forall_{\neg \mathsf{h}}\mathbf{z}\ .\ \psi \mid \exists_{\mathsf{h}}\mathbf{y} \forall_{\neg \mathsf{h}}\mathbf{z} \forall_{\neg \mathsf{h}}\hat{x}\ .\ \psi[\hat{x}/x] \in F\},\ \mathsf{add}(\{x_1, \ldots, x_n\}, F) \stackrel{\text{def}}{=} \mathsf{add}(x_1, \ldots \mathsf{add}(x_n, F))
\end{gathered} \quad (3)$$

The choice of the set D above ensures (together with the restriction to normal models) that $\circledast_D$ is indeed a composition operator. Intuitively, since the considered models are normal, every location in the frontier between the heaps corresponding to ϕ_1 and ϕ_2 will be associated with a variable, thus D denotes the set of allocated locations on the frontier. Note that, because $\mathcal{P}$ is normalized, $\mathsf{alloc}_{\mathfrak{C}_{\mathcal{S}}}(\phi_1 * \phi_2)$ is well-defined. Because the properties of the composition operation hold when the models of its operands share the same store (Definition 26), we use the $\mathsf{add}(x, F)$ function that adds free variables (mapped to locations outside of the heap) to each core formula in F.

Existential Quantifiers. Since profiles involve only core formulæ obtained by the syntactic translation of a symbolic heap (Lemma 25), it is sufficient to consider only existentially quantified core formulæ, because the syntactic translation $\mathcal{T}(.)$ does not produce universal quantifiers. The profile of an existentially quantified core formula is given by the constraint:

$$\begin{gathered}
(\exists_{\mathsf{h}}x'\ .\ \phi[x'/x],\ \mathsf{rem}(x, F)) \in \mathcal{F}_{\mathcal{P}}, \text{ if } x \in \mathsf{fv}(\phi),\ x' \in \mathcal{V}^2_{\mathcal{P}},\ x' \text{ not bound in } \phi,\ (\phi, F) \in \mathcal{F}_{\mathcal{P}}, \\
\mathsf{rem}(x, F) \stackrel{\text{def}}{=} \{\exists_{\mathsf{h}}\hat{x}\ .\ \psi[\hat{x}/x] \mid \psi \in F,\ x \in \mathsf{fv}(\psi),\ \hat{x} \text{ not in } \psi\} \cap \mathsf{Core}(\mathcal{P}) \cup \{\psi \mid \psi \in F,\ x \notin \mathsf{fv}(\psi)\} \\
\mathsf{rem}(\{x_1, \ldots, x_n\}, F) \stackrel{\text{def}}{=} \mathsf{rem}(x_1, \ldots \mathsf{rem}(x_n, F) \ldots)
\end{gathered} \quad (4)$$

Note that $\hat{x}$ is a fresh variable, which is not bound or free in ψ. In particular, if $x \in \mathrm{roots}(\psi)$, then we must have $\hat{x} \in \mathcal{V}^2_{\mathcal{P}}$, so that $\exists_{\mathsf{h}}\hat{x}\ .\ \psi[\hat{x}/x] \in \mathsf{Core}(\mathcal{P})$. Similarly the variable x is replaced by a fresh variable $x' \in \mathcal{V}^2_{\mathcal{P}}$ in $\exists_{\mathsf{h}}x'\ .\ \phi[x'/x]$ to ensure that $\exists_{\mathsf{h}}x'\ .\ \phi[x'/x]$ is a core formula.

The Profile Function. Let $\mathcal{F}_{\mathcal{P}}$ be the least relation that satisfies the constraints (1), (2), (3) and (4). We prove that $\mathcal{F}_{\mathcal{P}}$ is a valid profile for $\mathcal{P}$, in the sense of Definition 24:

▶ **Lemma 27.** *Given a progressing and normalized entailment problem $\mathcal{P} = (\mathcal{S}, \Sigma)$, a symbolic heap $\varphi \in \mathsf{SH}^{\mathfrak{R}}$ with $\mathsf{fv}(\varphi) \subseteq \mathcal{V}^1_{\mathcal{P}}$, a core formula $\phi \in \mathcal{T}(\varphi)$ and a set of core formulæ $F \subseteq \mathsf{Core}(\mathcal{P})$, we have $(\phi, F) \in \mathcal{F}_{\mathcal{P}}$ iff $F = \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h})$, for some injective normal $\mathfrak{C}_{\mathcal{S}}$-model $(\dot{\mathfrak{s}}, \mathfrak{h})$ of ϕ, with $\mathrm{dom}(\dot{\mathfrak{s}}) = \mathsf{fv}(\varphi) \cup \mathbb{C}$.*

The composition operation $\circledast_D$ works symbolically on core formulæ, by saturating the separating conjunction of two core formulæ via a *modus ponens*-style consequence operator.

▶ **Definition 28.** *Given formulæ ϕ, ψ, we write $\phi \Vdash \psi$ if $\phi = \varphi * [\alpha \mathrel{-\!\!\bullet} p(\mathbf{t})] * [(\beta * p(\mathbf{t})) \mathrel{-\!\!\bullet} q(\mathbf{u})]$ and $\psi = \varphi * [(\alpha * \beta) \mathrel{-\!\!\bullet} q(\mathbf{u})]$ (up to the commutativity of $*$ and the neutrality of emp) for some formula φ, predicate atoms $p(\mathbf{t})$ and $q(\mathbf{u})$ and conjunctions of predicate atoms α and β.*

▶ **Example 29.** Consider the structure $(\mathfrak{s}, \mathfrak{h})$ and the rules of Example 18. We have $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, with $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_1) \models_{\mathfrak{C}_{\mathcal{S}}} q(y) \mathrel{-\!\!\bullet} p(x)$ and $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_2) \models_{\mathcal{S}} q(y)$, i.e., $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}_2) \models_{\mathfrak{C}_{\mathcal{S}}} \mathsf{emp} \mathrel{-\!\!\bullet} q(y)$, thus $(\mathfrak{s}[y \leftarrow \ell_3], \mathfrak{h}) \models_{\mathfrak{C}_{\mathcal{S}}} q(y) \mathrel{-\!\!\bullet} p(x) * \mathsf{emp} \mathrel{-\!\!\bullet} q(y) \Vdash \mathsf{emp} \mathrel{-\!\!\bullet} p(x)$. ⌟

We define a relation on the set of core formulæ $\mathsf{Core}(\mathcal{P})$, parameterized by a set $D \subseteq \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$:

$$\begin{array}{c} \exists_{\mathfrak{h}} \mathbf{x}_1 \forall_{\neg \mathfrak{h}} \mathbf{y}_1 \,.\, \psi_1, \exists_{\mathfrak{h}} \mathbf{x}_2 \forall_{\neg \mathfrak{h}} \mathbf{y}_2 \,.\, \psi_2 \Vdash\!\!\!\vdash_D \exists_{\mathfrak{h}} \mathbf{x} \forall_{\neg \mathfrak{h}} \mathbf{y} \,.\, \psi \\ \text{if } \psi_1 * \psi_2 \Vdash^* \psi, \mathbf{x}_1 \cap \mathbf{x}_2 = \emptyset, \mathbf{x} = (\mathbf{x}_1 \cup \mathbf{x}_2) \cap \mathsf{fv}(\psi), \mathbf{y} = ((\mathbf{y}_1 \cup \mathbf{y}_2) \cap \mathsf{fv}(\psi)) \setminus \mathbf{x}, \mathrm{roots}_{\mathsf{lhs}}(\psi) \cap D = \emptyset. \end{array} \tag{5}$$

The composition operator is defined by lifting the $\Vdash\!\!\!\vdash$ relation to sets of core formulæ:

$$F_1 \circledast_D F_2 \stackrel{\text{def}}{=} \{\psi \mid \phi_1 \in F_1, \phi_2 \in F_2, \phi_1, \phi_2 \Vdash\!\!\!\vdash_D \psi\} \tag{6}$$

We show that $\circledast_D$ is indeed a composition, in the sense of Definition 26:

▶ **Lemma 30.** *Let $\mathcal{S}$ be a normalized, progressing, connected and e-restricted set of rules, $D \subseteq \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$ be a set of terms and $(\dot{\mathfrak{s}}, \mathfrak{h})$ be an injective structure, with $\mathrm{dom}(\dot{\mathfrak{s}}) \subseteq \mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C}$. Let $\mathfrak{h}_1$ and $\mathfrak{h}_2$ be two disjoint heaps, such that:*
(1) $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$,
(2) $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \subseteq \dot{\mathfrak{s}}(\mathcal{V}^1_{\mathcal{P}} \cup \mathbb{C})$ *and*
(3) $\mathrm{Fr}(\mathfrak{h}_1, \mathfrak{h}_2) \cap \mathrm{dom}(\mathfrak{h}) \subseteq \dot{\mathfrak{s}}(D) \subseteq \mathrm{dom}(\mathfrak{h})$.
Then, we have $\mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}) = \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}_1) \circledast_D \mathcal{C}_{\mathcal{P}}(\dot{\mathfrak{s}}, \mathfrak{h}_2)$.

7 Main Result

In this section, we state the main complexity result of the paper. As a prerequisite, we prove that the size of the core formulæ needed to solve an entailment problem $\mathcal{P}$ is polynomial in $\mathrm{width}(\mathcal{P})$ and the number of such formulæ is simply exponential in $\mathrm{width}(\mathcal{P}) + \log(\mathrm{size}(\mathcal{P}))$.

▶ **Lemma 31.** *Given an entailment problem $\mathcal{P}$, for every formula $\phi \in \mathsf{Core}(\mathcal{P})$, we have $\mathrm{size}(\phi) = \mathcal{O}(\mathrm{width}(\mathcal{P})^2)$ and $||\mathsf{Core}(\mathcal{P})|| = 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^3 \times \log(\mathrm{size}(\mathcal{P})))}$.*

▶ **Theorem 32.** *Checking the validity of progressing, connected and e-restricted entailment problems is* 2-EXPTIME*-complete.*

Proof. 2-EXPTIME-hardness follows from [6]; since the reduction in [6] involves no (dis-)equality, the considered systems are trivially e-restricted. We now prove 2-EXPTIME-membership. Let $\mathcal{P}$ be an e-restricted problem. By Lemma 11, we compute, in time $\mathrm{size}(\mathcal{P}) \cdot 2^{\mathcal{O}(\mathrm{width}(\mathcal{P})^2)}$, an equivalent normalized e-restricted problem $\mathcal{P}_n$ of $\mathrm{size}(\mathcal{P}_n) =$

$\text{size}(\mathcal{P}) \times 2^{\mathcal{O}(\text{width}(\mathcal{P})^2)}$ and $\text{width}(\mathcal{P}_n) = \mathcal{O}(\text{width}(\mathcal{P})^2)$. We fix an arbitrary set of variables $\mathcal{V}_{\mathcal{P}_n} = \mathcal{V}^1_{\mathcal{P}_n} \uplus \mathcal{V}^2_{\mathcal{P}_n}$ with $||\mathcal{V}^i_{\mathcal{P}_n}|| = \text{width}(\mathcal{P}_n)$, for $i = 1, 2$ and we compute the relation $\mathcal{F}_{\mathcal{P}_n}$, using a Kleene iteration, as explained in Section 6 (Lemma 27). By Lemma 31, if $\psi \in \mathsf{Core}(\mathcal{P}_n)$ then $\text{size}(\psi) = \mathcal{O}(\text{width}(\mathcal{P})^2)$ and if $(\psi, F) \in \mathcal{F}_{\mathcal{P}_n}$ then $||F|| = 2^{\mathcal{O}(\text{width}(\mathcal{P}_n)^3 \times \log(\text{size}(\mathcal{P}_n)))} = 2^{\mathcal{O}(\text{width}(\mathcal{P})^8 \times \log(\text{size}(\mathcal{P})))}$, hence $\mathcal{F}_{\mathcal{P}}$ can be computed in $2^{2^{\mathcal{O}(\text{width}(\mathcal{P})^8 \times \log(\text{size}(\mathcal{P})))}}$ steps. It thus suffices to check that each of these steps can be performed in polynomial time w.r.t. $\mathsf{Core}(\mathcal{P}_n)$ and $\text{size}(\mathcal{P}_n)$. This is straightforward for points-to atoms, predicate atoms and existential formulæ, by iterating on the rules in $\mathcal{P}_n$ and applying the construction rules (1), (2) and (4) respectively. For the disjoint composition, one has to compute the relation $\Vdash^*$, needed to build the operator $\circledast_D$, according to (5) and (6). We use again a Kleene iteration. It is easy to check that $\phi \Vdash \psi \Rightarrow \text{size}(\psi) \leq \text{size}(\phi)$, furthermore, one only needs to check relations of the form $\phi_1 * \phi_2 \Vdash \psi$ with $\phi_1, \phi_2, \psi \in \mathsf{Core}(\mathcal{P}_n)$. This entails that the number of iteration steps is $2^{\mathcal{O}(\text{width}(\mathcal{P})^8 \times \log(\text{size}(\mathcal{P})))}$ and, moreover, each step can be performed in time polynomial w.r.t. $\mathsf{Core}(\mathcal{P}_n)$. Finally, we apply Lemma 25 to check that all the entailments in $\mathcal{P}_n$ are valid. This test can be performed in time polynomial w.r.t. $||\mathcal{F}_{\mathcal{P}_n}||$ and $\text{size}(\mathcal{P}_n)$. ◀

8 Conclusion and Future Work

We presented a class of SL formulæ built from a set of inductively defined predicates, used to describe pointer-linked recursive data structures, whose entailment problem is 2-EXPTIME-complete. This fragment, consisting of so-called e-restricted formulæ, is a strict generalization of previous work defining three sufficient conditions for the decidability of entailments between SL formulæ, namely progress, connectivity and establishment [8, 12, 14]. On one hand, every progressing, connected and established entailment problem can be translated into an e-restricted problem. On the other hand, the models of e-restricted formulæ form a strict superset of the models of established formulæ. The proof for the 2-EXPTIME upper bound for e-restricted entailments leverages a novel technique used to prove the upper bound of established entailments [12, 14]. A natural question is whether the e-restrictedness condition can be dropped. We conjecture that this is not the case, and that entailment is undecidable for progressing, connected and non-e-restricted sets. Another issue is whether the generalization of symbolic heaps to use guarded negation, magic wand and septraction from [15] is possible for e-restricted entailment problems. The proof of these conjectures is on-going work.

References

1 Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *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 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

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 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. URL: `https://easychair.org/publications/paper/DdNg`.

7 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag New York, Inc., 2006.

8 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.

9 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.

10 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.

11 Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. Unified reasoning about robustness properties of symbolic-heap separation logic. In Hongseok Yang, editor, *Programming Languages and Systems (ESOP'17)*, pages 611–638. Springer Berlin Heidelberg, 2017.

12 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.

13 Koji Nakazawa, Makoto Tatsuta, Daisuke Kimura, and Mitsuru Yamamura. Cyclic Theorem Prover for Separation Logic by Magic Wand. In *ADSL 18 (First Workshop on Automated Deduction for Separation Logics)*, July 2018. Oxford, United Kingdom.

14 Jens Pagel, Christoph Matheja, and Florian Zuleger. Complete entailment checking for separation logic with inductive definitions, 2020. `arXiv:2002.01202`.

15 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. URL: `https://easychair.org/publications/paper/VTGk`.

16 J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*, 2002.

17 Neil Robertson and P.D Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.