



HAL
open science

Two Results on Separation Logic With Theory Reasoning

Mnacho Echenim, Nicolas Peltier

► **To cite this version:**

Mnacho Echenim, Nicolas Peltier. Two Results on Separation Logic With Theory Reasoning. ASL 2022 - Workshop on Advancing Separation Logic, Jul 2022, Haifa, Israel. hal-03701399

HAL Id: hal-03701399

<https://hal.science/hal-03701399v1>

Submitted on 23 Jun 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Two Results on Separation Logic With Theory Reasoning

Mnacho Echenim¹ and Nicolas Peltier¹

Univ. Grenoble Alpes, CNRS, LIG, F-38000 Grenoble France

1 Introduction

In Separation Logic (see, e.g., [11,18]), recursive data structures are usually specified using inductively defined predicates. The recursive rules defining the semantics of these predicates may be provided by the user. This specification mechanism is similar to the definition of a recursive data type in an imperative programming language. For instance, a nonempty list segment may be specified by using the inductive rules below, where the atom $x \mapsto (y)$ states that the memory location corresponding to x is allocated and refers to y . The symbol $*$ is a special logical connective denoting the disjoint composition of heaps.

$$\mathbf{ls}(x, y) \Leftarrow x \mapsto (y) \quad \mathbf{ls}(x, y) \Leftarrow \exists x' . (x \mapsto (x') * \mathbf{ls}(x', y))$$

Sorted lists with elements inside the interval $[u, v]$ may be specified as follows.

$$\mathbf{ils}(x, y, u, v) \Leftarrow (x \mapsto (y) \wedge x \leq v \wedge x \geq u)$$

$$\mathbf{ils}(x, y, u, v) \Leftarrow \exists x' . (x \mapsto (x') * \mathbf{ils}(x', y, x, v) \wedge x \leq v \wedge x \geq u)$$

Many verification tasks can be reduced to an entailment problem in this logic. For example, to verify that some formula ϕ is a loop invariant, we have to prove that the weakest pre-condition of ϕ w.r.t. a finite sequence of transformations is a logical consequence of ϕ . Since techniques exist for computing automatically such pre-conditions, the problem may be reduced to an entailment problem between two SL formulas. Entailment problems can also be used to express typing properties. For instance one may have to check that the entailment $\mathbf{ils}(x, y, u, v) \models \mathbf{ls}(x, y)$ holds, i.e., that a sorted list is a list, that $v \leq v' \wedge \mathbf{ils}(x, y, u, v) \models \mathbf{ils}(x, y, u, v')$ holds (if v is an upper bound then any number greater than v is also an upper bound) or that $\mathbf{ils}(x, y, u, v) * \mathbf{ils}(y, z, v, v') \models \mathbf{ils}(x, y, u, v')$ (the composition of two sorted lists is a sorted list). In general, the entailment problem is undecidable [10], and a lot of effort has been devoted to identifying decidable fragments and devising proof procedures, see e.g., [1,2,3,8,7,4]. In particular, a very general class of decidable entailment problems is described in [9]. This fragment does not allow for any theory predicate other than equality and is defined by restricting the form of the inductive rules, which must fulfill 3 conditions, formally defined below: the *progress* condition (every rule allocates a single memory location), the *connectivity* condition (the set of allocated locations has a tree-shaped structure) and the *establishment* condition (every existentially

quantified variable is eventually allocated). More recently, a 2-EXPTIME algorithm was proposed for such entailments [14], and we showed in [5] that this bound is tight. To tackle entailments such as those given above, one must be able to combine spatial reasoning with theory reasoning, and the combination of SL with data constraints has been considered by several authors (see, e.g., [16,17,15,19,12]). It is therefore natural to ask whether the above decidability result extends to the case where theory reasoning is considered. In the present paper, we show that this is not the case, even for very simple theories. More precisely, we establish two new results. First, we show that the entailment problem is undecidable for rules satisfying the above conditions if theory reasoning is allowed (Theorem 9). The result holds for a very wide class of theories, even for theories with a very low expressive power. For instance, it holds for the natural numbers with only the successor function, or with only the predicate \leq (interpreted as usual). Second, we show that every entailment can be reduced to an entailment not containing equality (Theorem 13). The intuition is that all the equality and disequality constraints can be encoded in the formulas describing the shape of the data structures. The transformation increases the number of rules exponentially but it increases the size of the rules only polynomially (hence it preserves the complexity results in [6]). This result shows that the addition of the equality predicate does not increase the expressive power. It may be useful to facilitate the definition of proof procedures for the considered fragment.

2 Preliminaries

In this section, we define the syntax and semantics of the fragment of separation logic that is considered in the paper (see for instance [13,18,9] for more details).

Syntax. Let \mathcal{V} be a countably infinite set of *variables*. We consider a set $\mathcal{P}_{\mathcal{T}}$ of \mathcal{T} -*predicates* (or *theory predicates*, denoting relations in an underlying theory of locations) and a set $\mathcal{P}_{\mathcal{S}}$ of *spatial predicates*, disjoint from $\mathcal{P}_{\mathcal{T}}$. Each symbol $p \in \mathcal{P}_{\mathcal{T}} \cup \mathcal{P}_{\mathcal{S}}$ is associated with a unique arity $\#(p)$. We assume that $\mathcal{P}_{\mathcal{T}}$ contains in particular two binary symbols \approx and $\not\approx$ and a nullary symbol **false**.

Definition 1. *Let κ be some fixed natural number. The set of SL-formulas (or simply formulas) ϕ is inductively defined as follows:*

$$\phi := \mathbf{emp} \parallel x \mapsto (y_1, \dots, y_{\kappa}) \parallel \phi_1 \vee \phi_2 \parallel \phi_1 * \phi_2 \parallel p(x_1, \dots, x_{\#(p)}) \parallel \exists x. \phi_1$$

where ϕ_1, ϕ_2 are SL-formulas, $p \in \mathcal{P}_{\mathcal{T}} \cup \mathcal{P}_{\mathcal{S}}$ and $x, x_1, \dots, x_{\#(p)}, y_1, \dots, y_{\kappa}$ are variables.

A formula of the form $x \mapsto (y_1, \dots, y_{\kappa})$ is called a *points-to atom*, and a formula $p(x_1, \dots, x_{\#(p)})$ with $p \in \mathcal{P}_{\mathcal{S}}$ is called a *predicate atom*. A *spatial atom* is either a points-to atom or a predicate atom. A \mathcal{T} -*atom* is a formula of the form $p(x_1, \dots, x_{\#(p)})$ with $p \in \mathcal{P}_{\mathcal{T}}$. An *atom* is either a spatial atom or a \mathcal{T} -atom. A \mathcal{T} -*formula* is either **emp** or a separating conjunction of \mathcal{T} -atoms. A formula of the

form $\exists x_1 \dots \exists x_n. \phi$ (with $n \geq 0$) is denoted by $\exists \mathbf{x}. \phi$, where $\mathbf{x} = (x_1, \dots, x_n)$. A formula is *predicate-free* (resp. *disjunction-free*, resp. *quantifier-free*) if it contains no predicate symbol in \mathcal{P}_S (resp. no occurrence of \vee , resp. of \exists). It is in *prenex form* if it is of the form $\exists \mathbf{x}. \phi$, where ϕ is quantifier-free and \mathbf{x} is a possibly empty vector of variables. A *symbolic heap* is a prenex disjunction-free formula, i.e., a formula of the form $\exists \mathbf{x}. \phi$, where ϕ is a separating conjunction of atoms.

Let $fv(\phi)$ be the set of variables freely occurring in ϕ . We assume (using α -renaming if needed) that all existential variables are distinct from free variables and that distinct occurrences of quantifiers bind distinct variables. A *substitution* σ is a function mapping variables to variables. The *domain* $dom(\sigma)$ of a substitution σ is the set of variables x such that $\sigma(x) \neq x$, and we let $img(\sigma) = \sigma(dom(\sigma))$. For any expression (variable, tuple of variables or formula) e , we denote by $e\sigma$ the expression obtained from e by replacing every free occurrence of a variable x by $\sigma(x)$ and by $\{x_i \leftarrow y_i \mid 1 \leq i \leq n\}$ (where the x_1, \dots, x_n are pairwise distinct) the substitution such that $\sigma(x_i) = y_i$ and $dom(\sigma) \subseteq \{x_1, \dots, x_n\}$. For all sets E , $card(E)$ is the cardinality of E . For all sequences or words w , $\|w\|$ denotes the length of w . We sometimes identify vectors with sets, if the order is unimportant, e.g., we may write $\mathbf{x} \setminus \mathbf{y}$ to denote the vector formed by the components of \mathbf{x} that do not occur in \mathbf{y} .

We assume that the symbols in $\mathcal{P}_S \cup \mathcal{P}_T \cup \mathcal{V}$ are words over a finite alphabet of some constant size, strictly greater than 1. For any expression e , we denote by $size(e)$ the size of e , i.e., the number of occurrences of symbols¹ in e . We define the *width* of a formula as follows:

$$\begin{aligned} width(\phi_1 \vee \phi_2) &= \max(width(\phi_1), width(\phi_2)) & width(\exists x. \phi) &= width(\phi) + size(\exists x) \\ width(\phi_1 * \phi_2) &= width(\phi_1) + width(\phi_2) + 1 \\ width(\phi) &= size(\phi) & \text{if } \phi \text{ is an atom} \end{aligned}$$

Note that $width(\phi)$ coincides with $size(\phi)$ if ϕ is disjunction-free.

Inductive Rules. The semantics of the predicates in \mathcal{P}_S is given by user-defined inductive rules. To ensure decidability in the case where the theory only contains the equality predicate, these rules must satisfy some additional conditions (defined in [9]):

Definition 2. A (progressing and connected) set of inductive rules (pc-SID) \mathcal{R} is a finite set of rules of the form $p(x_1, \dots, x_n) \leftarrow \exists \mathbf{u}. x_1 \mapsto (y_1, \dots, y_\kappa) * \phi$ where $fv(x_1 \mapsto (y_1, \dots, y_\kappa) * \phi) \subseteq \{x_1, \dots, x_n\} \cup \mathbf{u}$, ϕ is a possibly empty separating conjunction of predicate atoms and \mathcal{T} -formulas, and for every predicate atom $q(z_1, \dots, z_{\#(q)})$ occurring in ϕ , we have $z_1 \in \{y_1, \dots, y_\kappa\}$. We let $size(p(\mathbf{x}) \leftarrow \phi) = size(p(\mathbf{x})) + size(\phi)$, $size(\mathcal{R}) = \sum_{\rho \in \mathcal{R}} size(\rho)$ and $width(\mathcal{R}) = \max_{\rho \in \mathcal{R}} size(\rho)$.

In the following, \mathcal{R} always denotes a pc-SID. Note that the right-hand side of every inductive rule contains exactly one points-to atom, the left-hand side

¹ Each symbol s in $\mathcal{P}_S \cup \mathcal{P}_T \cup \mathcal{V}$ is counted with a weight equal to its length $\|s\|$, and all the logical symbols have weight 1.

of which is the first argument x_1 of the predicate symbol (this condition is referred to as the *progress* condition), and that this points-to atom contains the first argument of every predicate atom on the right-hand side of the rule (the *connectivity* condition).

Definition 3. We write $p(x_1, \dots, x_{\#(p)}) \Leftarrow_{\mathcal{R}} \phi$ if \mathcal{R} contains a rule (up to α -renaming) $p(y_1, \dots, y_{\#(p)}) \Leftarrow \psi$, where $x_1, \dots, x_{\#(p)}$ are not bound in ψ , and $\phi = \psi\{y_i \leftarrow x_i \mid i \in \{1, \dots, \#(p)\}\}$.

The relation $\Leftarrow_{\mathcal{R}}$ is extended to all formulas as follows: $\phi \Leftarrow_{\mathcal{R}} \phi'$ if one of the following conditions holds: (i) $\phi = \phi_1 \bullet \phi_2$ (modulo AC, with $\bullet \in \{*, \vee\}$), $\phi_1 \Leftarrow_{\mathcal{R}} \phi'_1$, no free or existential variable in ϕ_2 is bound in ϕ'_1 and $\phi' = \phi'_1 \bullet \phi_2$; or (ii) $\phi = \exists x. \psi$, $\psi \Leftarrow_{\mathcal{R}} \psi'$, x is not bound in ψ' and $\phi' = \exists x. \psi'$. We denote by $\Leftarrow_{\mathcal{R}}^+$ the transitive closure of $\Leftarrow_{\mathcal{R}}$, and by $\Leftarrow_{\mathcal{R}}^*$ its reflexive and transitive closure. A formula ψ such that $\phi \Leftarrow_{\mathcal{R}}^* \psi$ is called an \mathcal{R} -unfolding of ϕ . We denote by $\geq_{\mathcal{R}}$ the least transitive and reflexive binary relation on \mathcal{P}_S such that $p \geq_{\mathcal{R}} q$ holds if \mathcal{R} contains a rule of the form $p(y_1, \dots, y_{\#(p)}) \Leftarrow \psi$, where q occurs in ψ . If ϕ is a formula, we write $\phi \geq_{\mathcal{R}} q$ if $p \geq_{\mathcal{R}} q$ for some $p \in \mathcal{P}_S$ occurring in ϕ .

Semantics.

Definition 4. Let \mathcal{L} be a countably infinite set of locations. An SL-structure is a pair $(\mathfrak{s}, \mathfrak{h})$ where \mathfrak{s} is a store, i.e. a total function from \mathcal{V} to \mathcal{L} , and \mathfrak{h} is a heap, i.e. a partial finite function from \mathcal{L} to \mathcal{L}^κ (written as a relation: $\mathfrak{h}(\ell) = (\ell_1, \dots, \ell_\kappa)$ iff $(\ell, \ell_1, \dots, \ell_\kappa) \in \mathfrak{h}$). The size of a structure $(\mathfrak{s}, \mathfrak{h})$ is the cardinality of $\text{dom}(\mathfrak{h})$.

For every heap \mathfrak{h} , we define: $\text{loc}(\mathfrak{h}) = \{\ell_i \mid (\ell_0, \dots, \ell_\kappa) \in \mathfrak{h}, i = 0, \dots, \kappa\}$. A location ℓ (resp. a variable x) is *allocated* in a heap \mathfrak{h} (resp. in a structure $(\mathfrak{s}, \mathfrak{h})$) if $\ell \in \text{dom}(\mathfrak{h})$ (resp. $\mathfrak{s}(x) \in \text{dom}(\mathfrak{h})$). Two heaps $\mathfrak{h}_1, \mathfrak{h}_2$ are *disjoint* if $\text{dom}(\mathfrak{h}_1) \cap \text{dom}(\mathfrak{h}_2) = \emptyset$, in this case $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ denotes the union of \mathfrak{h}_1 and \mathfrak{h}_2 .

Let $\models_{\mathcal{T}}$ be a satisfiability relation between stores and \mathcal{T} -formulas, satisfying the following properties: $\mathfrak{s} \models_{\mathcal{T}} x \approx y$ (resp. $\mathfrak{s} \models_{\mathcal{T}} x \not\approx y$) iff $\mathfrak{s}(x) = \mathfrak{s}(y)$ (resp. $\mathfrak{s}(x) \neq \mathfrak{s}(y)$), $\mathfrak{s} \not\models_{\mathcal{T}} \text{false}$ and $\mathfrak{s} \models_{\mathcal{T}} \chi * \xi$ iff $\mathfrak{s} \models_{\mathcal{T}} \chi$ and $\mathfrak{s} \models_{\mathcal{T}} \xi$. For all \mathcal{T} -formulas χ, ξ , we write $\chi \models_{\mathcal{T}} \xi$ if $\mathfrak{s} \models_{\mathcal{T}} \chi \implies \mathfrak{s} \models_{\mathcal{T}} \xi$ holds for all stores \mathfrak{s} .

Definition 5. Given formula ϕ , a pc-SID \mathcal{R} and a structure $(\mathfrak{s}, \mathfrak{h})$, we write $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ and say that $(\mathfrak{s}, \mathfrak{h})$ is an \mathcal{R} -model (or simply a model if \mathcal{R} is clear from the context) of ϕ if one of the following conditions holds.

- $\phi = x \mapsto (y_1, \dots, y_\kappa)$ and $\mathfrak{h} = \{(\mathfrak{s}(x), \mathfrak{s}(y_1), \dots, \mathfrak{s}(y_\kappa))\}$.
 - ϕ is a \mathcal{T} -formula, $\mathfrak{h} = \emptyset$ and $\mathfrak{s} \models_{\mathcal{T}} \phi$.
 - $\phi = \phi_1 \vee \phi_2$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi_i$, for some $i = 1, 2$.
 - $\phi = \phi_1 * \phi_2$ and there exist disjoint heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ and $(\mathfrak{s}, \mathfrak{h}_i) \models_{\mathcal{R}} \phi_i$, for all $i = 1, 2$.
 - $\phi = \exists x. \phi$ and $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} \phi$, for some store \mathfrak{s}' coinciding with \mathfrak{s} on all variables distinct from x .
 - $\phi = p(x_1, \dots, x_{\#(p)})$, $p \in \mathcal{P}_S$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi$ for some ψ such that $\phi \Leftarrow_{\mathcal{R}} \psi$.
- If Γ is a sequence of formulas, then we write $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \Gamma$ if $(\mathfrak{s}, \mathfrak{h})$ satisfies at least one formula in Γ .

We emphasize that a \mathcal{T} -formula is satisfied only in structures with empty heaps. This convention is used to simplify notations, because it avoids having to consider both standard and separating conjunctions. Note that Definition 5 is well-founded because of the progress condition: the size of \mathfrak{h} decreases at each recursive call of a predicate atom. We write $\phi \models_{\mathcal{R}} \psi$ if every \mathcal{R} -model of ϕ is an \mathcal{R} -model of ψ and $\phi \equiv_{\mathcal{R}} \psi$ if $\phi \models_{\mathcal{R}} \psi$ and $\psi \models_{\mathcal{R}} \phi$.

Every formula can be transformed into prenex form using the well-known equivalences: $(\exists x.\phi) \bullet \psi \equiv \exists x.(\phi \bullet \psi)$, for all $\bullet \in \{\vee, *\}$, where $x \notin \text{fv}(\psi)$.

Establishment. The notion of establishment [9] is defined as follows:

Definition 6. A pc-SID is established if for every atom α , every predicate-free formula $\exists \mathbf{x}.\phi$ such that $\alpha \leftarrow_{\mathcal{R}}^* \exists \mathbf{x}.\phi$ (up to a transformation into prenex form) and ϕ is quantifier-free, and every $x \in \mathbf{x}$, ϕ is of the form $x' \mapsto (y_1, \dots, y_\kappa) * \chi * \psi$, where χ is a separating conjunction of equations (possibly **emp**) such that $\chi \models_{\mathcal{T}} x \approx x'$.

In the remainder of the paper, we assume that every considered pc-SID is established.

Sequents. We consider sequents denoting entailment problems and defined as follows:

Definition 7. A sequent is an expression of the form $\phi_0 \vdash_{\mathcal{R}} \phi_1, \dots, \phi_n$, where \mathcal{R} is a pc-SID and ϕ_0, \dots, ϕ_n are formulas. A sequent is disjunction-free if ϕ_0, \dots, ϕ_n are disjunction-free, and established if \mathcal{R} is established. We define:

$$\begin{aligned} \text{size}(\phi_0 \vdash_{\mathcal{R}} \phi_1, \dots, \phi_n) &= \sum_{i=0}^n \text{size}(\phi_i) + \text{size}(\mathcal{R}), & \text{fv}(\phi_1, \dots, \phi_n) &= \bigcup_{i=0}^n \text{fv}(\phi_i), \\ \text{width}(\phi_0 \vdash_{\mathcal{R}} \phi_1, \dots, \phi_n) &= \max\{\text{width}(\phi_i), \text{width}(\mathcal{R}), \text{card}(\bigcup_{i=0}^n \text{fv}(\phi_i)) \mid 0 \leq i \leq n\}. \end{aligned}$$

Definition 8. A structure $(\mathfrak{s}, \mathfrak{h})$ is a countermodel of a sequent $\phi \vdash_{\mathcal{R}} \Gamma$ iff \mathfrak{s} is injective, $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ and $(\mathfrak{s}, \mathfrak{h}) \not\models_{\mathcal{R}} \Gamma$. A sequent is valid if it has no countermodel. Two sequents are equivalent if they are both valid or both non-valid².

The restriction to injective countermodels is for technical convenience only and does not entail any loss of generality.

3 An Undecidability Result

This section contains the main result of the paper. It shows that no terminating procedure for checking the validity of (established) sequents possibly exists, even for theories with a very low expressive power.

² Hence two non-valid sequents with different countermodels are equivalent.

Theorem 9. *The validity problem is undecidable for established sequents $\phi \vdash_{\mathcal{R}} \psi$ if $\mathcal{P}_{\mathcal{T}}$ contains predicates S and \overline{S} , where:*

- S is interpreted as a relation \mathfrak{S} satisfying the following property: there exists a set of pairwise distinct locations $\{\alpha_i, \alpha'_i, \alpha''_i \mid i \in \mathbb{N}\}$ such that for all $i \in \mathbb{N}$, $(\alpha_i, \alpha'_i) \in \mathfrak{S}$, $(\alpha''_i, \alpha'_i) \notin \mathfrak{S}$, and for all locations $\ell \in \{\alpha_j, \alpha'_j, \alpha''_j \mid j \in \mathbb{N}\}$ if $\alpha_i \neq \ell$, $(\alpha_i, \ell) \in \mathfrak{S}$ and $(\alpha''_i, \ell) \notin \mathfrak{S}$, then $\ell = \alpha'_i$;
- $\overline{S}(x, y)$ and $\neg S(x, y)$ are interpreted equivalently when x and y refer to distinct locations.

For instance, the hypotheses of Theorem 9 are trivially satisfied on the natural numbers if \mathfrak{S} is the successor function or if \mathfrak{S} is the usual order \leq , and \overline{S} is \geq (with $\alpha_i = 3.i$, $\alpha'_i = 3.i + 1$, $\alpha''_i = 3.i + 2$ in both cases, since $\alpha_i + 1 \approx \ell \Rightarrow \alpha'_i \approx \ell$ and $\alpha_i \leq \ell \wedge \alpha''_i > \ell \wedge \ell \neq \alpha_i \Rightarrow \alpha'_i \approx \ell$). More generally, the conditions hold if the domain is infinite and \mathfrak{S} is any injective function f such that $f(x) \neq x$. In this case, the sequences $\alpha_i, \alpha'_i, \alpha''_i$ may be constructed inductively: for every $i \in \mathbb{N}$, α_i is any element e such that both e and $f(e)$ do not occur in $\{\alpha_j, \alpha'_j, \alpha''_j \mid j < i\}$, α'_i is $f(\alpha_i)$ and α''_i is any element not occurring in $\{\alpha_j, \alpha'_j, \alpha''_j \mid j < i\} \cup \{\alpha_i, \alpha'_i\}$. Note that in this case the locations α''_i are actually irrelevant, but these locations play an essential rôle in the undecidability proof when \mathfrak{S} is \leq . The remainder of the section is devoted the proof of Theorem 9.

Proof. The proof goes by a reduction from the Post Correspondence Problem (PCP). We recall that the PCP consists in determining, given two sequences of words $u = (u_1, \dots, u_n)$ and $v = (v_1, \dots, v_n)$, whether there exists a nonempty sequence $(i_1, \dots, i_k) \in \{1, \dots, n\}^k$ such that $u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$. It is well-known that this problem is undecidable. We assume, w.l.o.g., that $\|u_i\| > 1$ and $\|v_i\| > 1$ for all $i \in \{1, \dots, n\}$. A word w such that $w = u_{i_1} \dots u_{i_k} = v_{i_1} \dots v_{i_k}$ is called a *witness*. Positions inside words of the sequences (u_1, \dots, u_n) and (v_1, \dots, v_n) will be denoted by pairs (i, j) , encoding the j -nth character of the words u_i or v_i . More formally, if $p = (i, j)$, and $w \in \{u, v\}$, then we denote by $w(p)$ the j -th symbol of the word w_i , provided this symbol is defined. We write $p \triangleleft q$ if both $u(p)$ and $v(q)$ are defined and $u(p) = v(q)$.

Let $m = \max\{\|u_i\|, \|v_i\| \mid i \in \{1, \dots, n\}\}$. We denote by \mathbf{P} the set of pairs of the form (i, j) with $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$, and by \mathbf{B} the set of pairs of the form $(i, 1)$. For $w \in \{u, v\}$, we denote by \mathbf{E}_w the set of pairs of the form $(i, \|w_i\|)$, where $i \in \{1, \dots, n\}$, and we write $(i, j) \rightarrow^w (i', j')$ either $i' = i$, $j < \|w_i\|$ and $j' = j + 1$, or $j = \|w_i\|$ and $j' = 1$. Note that i' is arbitrary in the latter case (intuitively $(i, j) \rightarrow^w (i', j')$ states that the character corresponding to the position (i, j) may be followed in a witness by the character at position (i', j')). Let \mathbf{v} be a vector of variables, where all elements $p \in \mathbf{P}$ are associated with pairwise distinct variables in \mathbf{v} . To simplify notations, we will also denote by p the variable associated with p . We assume the vector \mathbf{v} also contains a special variable \perp , distinct from the variables $p \in \mathbf{P}$. We construct a representation of potential witnesses as heaps. The encoding is given for $\kappa = 6$, although in principle this encoding could be defined with $\kappa = 2$ by encoding tuples as binary trees. Witnesses are encoded by linked lists, with links on the

last argument, and starting with a dummy element (\perp, \dots, \perp) . Except for the first dummy element, each location in the list refers to two locations associated with pairs $p, q \in \mathbf{P}$ denoting positions inside the two sequences u_1, \dots, u_n and v_1, \dots, v_n respectively, and to three additional allocated locations the rôle of which will be detailed below.

$$\begin{aligned}
W(x, \mathbf{v}) &\Leftarrow \exists x'. x \mapsto (\perp, \perp, \perp, \perp, \perp, x') * W_{p,p}(x', \mathbf{v}) \\
&\quad \text{where } p \in \mathbf{B} \\
W_{p,q}(x, \mathbf{v}) &\Leftarrow \exists x', y, z, u. x \mapsto (p, q, y, z, u, x') * W_{p',q'}(x', \mathbf{v}) * P(y, \perp) * P(z, \perp) \\
&\quad * P(u, \perp) \quad \text{where } p \triangleleft q, p \xrightarrow{u} p' \text{ and } q \xrightarrow{v} q' \\
W_{p,q}(x, \mathbf{v}) &\Leftarrow \exists y, z, u. x \mapsto (p, q, y, z, u, \perp) * P(y, \perp) * P(z, \perp) \\
&\quad * P(u, \perp) \quad \text{where } p = (i, \|u_i\|), q = (i, \|v_i\|), \text{ and } p \triangleleft q \\
P(x, y) &\Leftarrow x \mapsto (y, y, y, y, y, y)
\end{aligned}$$

By construction, the structures that validate $W(x, \mathbf{v})$ are of the form $(\mathfrak{s}, \mathfrak{h})$, where the store \mathfrak{s} verifies:

- $\mathfrak{s}(x) = \ell$ and $\mathfrak{s}(\perp) = \ell'$;
- for all $i = 1, \dots, m'$, $\mathfrak{s}(p_i) = \ell_i^p$ and $\mathfrak{s}(q_i) = \ell_i^q$, where $p_i, q_i \in \mathbf{P}$ are such that $p_i \triangleleft q_i$, $p_i \xrightarrow{u} p_{i+1}$ and $q_i \xrightarrow{v} q_{i+1}$,

and the heap \mathfrak{h} is of the form (with $\ell_{m'+1} = \ell'$):

$$\begin{aligned}
\mathfrak{h} = &\{(\ell, \ell', \ell', \ell', \ell', \ell', \ell_1)\} \cup \{(\ell_i, \ell_i^p, \ell_i^q, \ell_i^y, \ell_i^z, \ell_i^u, \ell_{i+1}) \mid i = 1, \dots, m'\} \\
&\cup \{(\ell_i^y, \ell', \ell', \ell', \ell', \ell', \ell') \mid i = 1, \dots, m'\} \\
&\cup \{(\ell_i^z, \ell', \ell', \ell', \ell', \ell', \ell') \mid i = 1, \dots, m'\} \\
&\cup \{(\ell_i^u, \ell', \ell', \ell', \ell', \ell', \ell') \mid i = 1, \dots, m'\}.
\end{aligned}$$

Still by construction, we have $p_1 = q_1 \in \mathbf{B}$, $p_{m'} \in \mathbf{E}_u$ and $q_{m'} \in \mathbf{E}_v$, and there exists i such that $p_{m'}$ and $q_{m'}$ are of the form $(i, \|u_i\|)$ and $(i, \|v_i\|)$, respectively. This entails that the words $u(p_1) \dots u(p_{m'})$ and $v(p_1) \dots v(p_{m'})$ are of form $u_{i_1} \dots u_{i_k}$ and $v_{j_1} \dots v_{j_{k'}}$, for some sequences (i_1, \dots, i_k) and $(j_1, \dots, j_{k'})$ of elements in $\{1, \dots, n\}$, and both words are identical. However the sequences (i_1, \dots, i_k) and $(j_1, \dots, j_{k'})$ may be distinct (but we have $i_1 = j_1$ and $i_k = j_{k'}$). To check that the PCP has a solution, we must therefore verify that there exists a structure of the form above such that $(i_1, \dots, i_k) = (j_1, \dots, j_{k'})$. To this purpose, we introduce predicates that are satisfied when this condition does not hold, i.e., such that either $k \neq k'$ or $i_l \neq j_l$ for some $l \in \{2, \dots, k-1\}$. This is done by using the additional locations ℓ_i^y, ℓ_i^z and ℓ_i^u to relate the indices $I_l = \|u_{i_1} \dots u_{i_{l-1}}\| + 1$ and $J_l = \|v_{j_1} \dots v_{j_{l-1}}\| + 1$, corresponding to the beginning of the words u_{i_l} and v_{j_l} respectively in $u_{i_1} \dots u_{i_k}$ and $v_{j_1} \dots v_{j_{k'}}$. The predicates relate the locations of the form ℓ_i^y, ℓ_i^z and ℓ_i^u using the relation \mathfrak{S} . More precisely, they are associated with rules that guarantee that all the countermodels of the right-hand side of the sequent will satisfy the following properties:

1. $k = k'$ and for all $l \in \{1, \dots, k\}$, $(\ell_{I_l}^y, \ell_{J_l}^z) \in \mathfrak{S}$ and $(\ell_{I_l}^u, \ell_{J_l}^z) \notin \mathfrak{S}$.
2. $i_l = j_l$ for $1 \leq l \leq k$.

Note that the hypothesis of the theorem ensures that the locations ℓ_i^y, ℓ_i^z and ℓ_i^u can be chosen in such a way that there is a *unique* location ℓ_i^z satisfying

$(\ell_{I_l}^y, \ell_{J_l}^z) \in \mathfrak{S} \wedge (\ell_{J_l}^u, \ell_{I_l}^z) \notin \mathfrak{S}$, thus property 1 above can be used to relate the indices I_l and J_l , which, in turns, allows us to enforce property 2. Two predicates are used to guarantee that condition 1 holds for the countermodels. Predicate A is satisfied by those structures for which the condition is not satisfied for $l = 1$, and B is satisfied for those structures for which either $k \neq k'$ or there is an $l \in \{1, \dots, k-1\}$ such that the condition is satisfied at l , but not at $l+1$. Thus the structures that satisfy $W(x, \mathbf{v})$ and that are countermodels of the disjunction of A and B are exactly the structures for which $k = k'$ and $(\ell_{I_l}^y, \ell_{J_l}^z) \in \mathfrak{S} \wedge (\ell_{I_l}^u, \ell_{J_l}^z) \notin \mathfrak{S}$ for $l = 1, \dots, k$. Predicate C is then used to guarantee that for all $1 \leq l \leq k$, $p_{I_l} = q_{J_l}$: this predicate is satisfied by the structures for which there is an l such that $(\ell_{I_l}^y, \ell_{J_l}^z) \in \mathfrak{S} \wedge (\ell_{I_l}^u, \ell_{J_l}^z) \notin \mathfrak{S}$ holds but $p_{I_l} \neq q_{J_l}$. We first give the rules for predicate A . For conciseness, we allow for disjunctions in the right-hand side of the rules (they can be eliminated by transformation into dnf).

$$\begin{aligned} A(x, \mathbf{v}) &\Leftarrow \exists x'. x \mapsto (\perp, \perp, \perp, \perp, \perp, x') * A'(x', \mathbf{v}) \\ A'(x, \mathbf{v}) &\Leftarrow \exists x', y, z, u. x \mapsto (p, q, y, z, u, x') * W_{p', q'}(x, \mathbf{v}) * P(y, \perp) \\ &\quad * P(z, \perp) * P(u, \perp) * (\overline{S}(y, z) \vee S(u, z)), \text{ for every } p, q, p', q' \in \mathbb{P} \end{aligned}$$

Note that since y, z, u are allocated in distinct predicates, they must be distinct, hence $\overline{S}(y, z)$ is equivalent to $\neg S(y, z)$ and $S(u, z)$ is equivalent to $\neg \overline{S}(u, z)$.

We now define the rules for predicate B , which is meant to ensure that the condition “ $(\ell_{I_l}^y, \ell_{J_l}^z) \in \mathfrak{S}$ and $(\ell_{I_l}^u, \ell_{J_l}^z) \notin \mathfrak{S}$ ” (\dagger) propagates, i.e., that if it holds for some l then it also holds for $l+1$. This predicate has additional parameters y, y', z, z', u, u' corresponding to the locations $\ell_{I_l}^y, \ell_{I_{l+1}}^y, \ell_{J_l}^z, \ell_{J_{l+1}}^z, \ell_{I_l}^u, \ell_{I_{l+1}}^u$ which “break” the propagation of (\dagger). By definition y, y', z, z', u, u' must be chosen in such a way that the \mathcal{T} -formula $S(y, z) * \overline{S}(u, z) * (\overline{S}(y', z') \vee S(u', z'))$ holds. The predicates $B_{a,b}$ with $a, b \in \{0, 1, 2\}$ allocate all the locations $\ell_1, \dots, \ell_{m'}$ and in particular the “faulty” locations associated with y, y', z, z', u, u' . Intuitively, a (resp. b) denote the number of variables in $\{y, y'\}$ (resp. $\{z, z'\}$) that have been allocated. The rules for predicates $B_{a,b}$ are meant to guarantee that the following conditions are satisfied for variables y and y' (similar constraints hold for z and z'):

- y is allocated before y' ,
- y is allocated for a variable p corresponding to the beginning of a word ($p \in \mathbb{B}$),
- when y has been allocated, no variable $p \in \mathbb{B}$ can occur on the right-hand side of a points-to atom until y' is allocated.

Several cases are distinguished depending on whether the locations associated with y and z (resp. y' and z') are in the same heap image of a location or not. Note that u and u' are allocated in the same rules as y and y' respectively. The predicate B also tackles the case where $k \neq k'$. This corresponds to the case where the recursive calls end with $B_{1,2}$ or $B_{2,1}$ in the last rule below, meaning that (\dagger) holds for some i , with either $i = k$ and $i < k'$ or $i = k'$ and $i < k$. For the sake of conciseness and readability, we denote by \mathbf{w} the vector of variables

$\mathbf{v}, y, y', z, z', u, u'$ in the rules below. We also denote by $\phi'(y, z, u)$ the formula $P(y, \perp) * P(z, \perp) * P(u, \perp)$.

$$\begin{aligned}
B(x, \mathbf{w}) &\Leftarrow x \mapsto (\perp, \perp, \perp, \perp, \perp, x') * B_{0,0}(x', \mathbf{w}) \\
&\quad * S(y, z) * \overline{S}(u, z) * (\overline{S}(y', z') \vee S(u', z')) \\
B_{a,b}(x, \mathbf{w}) &\Leftarrow \exists x', y'', z'', u''. x \mapsto (p, q, y'', z'', u'', x') * B_{a,b}(x', \mathbf{w}) * \phi'(y'', z'', u'') \\
&\quad \text{if } (a \neq 1 \text{ or } p \notin \mathbb{B}) \text{ and } (b \neq 1 \text{ or } q \notin \mathbb{B}) \\
B_{0,0}(x, \mathbf{w}) &\Leftarrow \exists x'. x \mapsto (p, q, y, z, u, x') * B_{1,1}(x', \mathbf{w}) * \phi'(y, z, u) \\
&\quad \text{if } p, q \in \mathbb{B} \\
B_{0,1}(x, \mathbf{w}) &\Leftarrow \exists x'. x \mapsto (p, q, y, z', u, x') * B_{1,2}(x', \mathbf{w}) * \phi'(y, z', u) \\
&\quad \text{if } p, q \in \mathbb{B} \\
B_{1,0}(x, \mathbf{w}) &\Leftarrow \exists x'. x \mapsto (p, q, y', z, u', x') * B_{2,1}(x', \mathbf{w}) * \phi'(y', z, u') \\
&\quad \text{if } p, q \in \mathbb{B} \\
B_{1,1}(x, \mathbf{w}) &\Leftarrow \exists x'. x \mapsto (p, q, y', z', u', x') * B_{2,2}(x', \mathbf{w}) * \phi'(y', z', u') \\
&\quad \text{if } p, q \in \mathbb{B} \\
B_{0,b}(x, \mathbf{w}) &\Leftarrow \exists x', z''. x \mapsto (p, q, y, z'', u, x') * B_{1,b}(x', \mathbf{w}) * \phi'(y, z'', u) \\
&\quad \text{if } p \in \mathbb{B} \text{ and } (b \neq 1 \text{ or } q \notin \mathbb{B}) \\
B_{1,b}(x, \mathbf{w}) &\Leftarrow \exists x', z''. x \mapsto (p, q, y', z'', u', x') * B_{2,b}(x', \mathbf{w}) * \phi'(y', z'', u') \\
&\quad \text{if } p \in \mathbb{B} \text{ and } (b \neq 1 \text{ or } q \notin \mathbb{B}) \\
B_{a,0}(x, \mathbf{w}) &\Leftarrow \exists x', y'', u''. x \mapsto (p, q, y'', z, u'', x') * B_{a,1}(x', \mathbf{w}) * \phi'(y'', z, u'') \\
&\quad \text{if } q \in \mathbb{B} \text{ and } (a \neq 1 \text{ or } p \notin \mathbb{B}) \\
B_{a,1}(x, \mathbf{w}) &\Leftarrow \exists x', y'', u''. x \mapsto (p, q, y'', z', u'', x') * B_{a,2}(x', \mathbf{w}) * \phi'(y'', z', u'') \\
&\quad \text{if } q \in \mathbb{B} \text{ and } (a \neq 1 \text{ or } p \notin \mathbb{B}) \\
B_{a,b}(x, \mathbf{w}) &\Leftarrow \exists y'', z'', u''. x \mapsto (p, q, y'', z'', u'', \perp) * \phi'(y'', z'', u'') \\
&\quad \text{if } (a, b) \in \{(2, 2), (2, 1), (1, 2)\}
\end{aligned}$$

A straightforward induction permits to verify that if the considered structure does not satisfy the formula $A(x, \mathbf{v}) \vee \exists y, z, y', z', u, u'. B(x, \mathbf{w})$ then necessarily $k = k'$ and for all $l \in \{1, \dots, k\}$, we have $(\ell_{I_l}^y, \ell_{J_l}^z) \in \mathfrak{S} \wedge (\ell_{I_l}^u, \ell_{J_l}^z) \notin \mathfrak{S}$.

There remains to check that $p_{I_i} = q_{J_i}$ for all $i \in \{1, \dots, k\}$. To this aim, we design an atom $C(x, \mathbf{v})$ that will be satisfied by structures not validating this condition, assuming the condition (\dagger) above is fulfilled. This predicate allocates the location ℓ and introduces existential variables y, z, u denoting the faulty locations $\ell_{I_i}^y, \ell_{J_i}^z$ and $\ell_{I_i}^u$, i.e., the locations corresponding to the index i such that $p_{I_i} \neq q_{J_i}$. By (\dagger) , these locations must be chosen in such a way that the constraints $S(y, z)$ and $\overline{S}(y, z)$ are satisfied. The predicate $C(x, \mathbf{v})$ also guesses pairs p, q such that $p \neq q$ (denoting the distinct pairs p_{x_i} and q_{y_i}) and invokes the predicate $C_{p,q}^{(0,0)}$ to allocate all the remaining locations. As for the previous rules, the predicates $C_{p,q}^{a,b}$, for $p, q \in \mathbb{B}$ $a, b \in \{0, 1\}$ allocate $\ell_1, \dots, \ell_{m'}$, where a (resp. b) denotes the number of variables in $\{y\}$ (resp. $\{z\}$) that have already been allocated. In the rules below, we denote by \mathbf{u} the vector \mathbf{v}, y, z, u . In all the rules we have $p', q' \in \mathbb{P}$.

$$\begin{aligned}
C(x, \mathbf{v}) &\Leftarrow \exists y, z, u. x \mapsto (\perp, \perp, \perp, \perp, \perp, x') * C_{p,q}^{0,0}(x', \mathbf{u}) * S(y, z) * \overline{S}(u, z) \\
&\quad \text{if } p \neq q \text{ and } p, q \in \mathbf{B} \\
C_{p,q}^{a,b}(x, \mathbf{u}) &\Leftarrow \exists x', y'', z'', u''. x \mapsto (p', q', y'', z'', u'', x') * C_{p,q}^{a,b}(x, \mathbf{u}) \\
&\quad * \phi'(y'', z'', u'') \\
C_{p,q}^{0,0}(x, \mathbf{u}) &\Leftarrow \exists x'. x \mapsto (p, q, y, z, u, x') * C_{p,q}^{1,1}(x, \mathbf{u}) * \phi'(y, z, u) \\
C_{p,q}^{0,b}(x, \mathbf{u}) &\Leftarrow \exists x', z''. x \mapsto (p, q', y, z'', u, x') * C_{p,q}^{1,b}(x, \mathbf{u}) * \phi'(y, z'', u) \\
C_{p,q}^{a,0}(x, \mathbf{u}) &\Leftarrow \exists x', y'', u''. x \mapsto (p', q, y'', z, u'', x') * C_{p,q}^{a,1}(x, \mathbf{u}) * \phi'(y'', z, u'') \\
C_{p,q}^{1,1}(x, \mathbf{u}) &\Leftarrow \exists y'', z'', u''. x \mapsto (p', q', y'', z'', u'', \perp) * \phi'(y'', z'', u'')
\end{aligned}$$

The PCP has a solution iff the sequent

$$W(x, \mathbf{v}) \vdash_{\mathcal{R}} A(x, \mathbf{v}), \exists y, z, y', z', u, u'. B(x, \mathbf{w}), C(x, \mathbf{u})$$

has a countermodel. Indeed, if a structure satisfying the atom $W(x, \mathbf{v})$ but not the disjunction $A(x, \mathbf{v}) \vee \exists y, z, y', z', u, u'. B(x, \mathbf{w}) \vee C(x, \mathbf{u})$ exists, then as explained above, there exists a word $u_{i_1} \dots u_{i_k} = v_{j_1} \dots v_{j_{k'}}$, with $(i_1, \dots, i_k) = (j_1, \dots, j_{k'})$. Conversely, if a solution of the PCP exists, then by using the locations $\alpha_l, \alpha'_l, \alpha''_l$ in the hypothesis of the lemma as $\ell_{I_l}^y, \ell_{J_l}^z, \ell_{I_l}^u$ it is easy to construct a structure satisfying $W(x, \mathbf{v})$. Further, by hypothesis, since $(\alpha_l, \alpha'_l) \in \mathfrak{S}$ and $(\alpha''_l, \alpha'_l) \notin \mathfrak{S}$, we have $(\ell_{I_l}^y, \ell_{J_l}^z) \in \mathfrak{S}$ and $(\ell_{I_l}^y, \ell_{J_l}^z) \notin \mathfrak{S}$ for all $l = 1, \dots, k$. Thus $A(x, \mathbf{v})$ and $\exists y, z, y', z', u, u'. B(x, \mathbf{w})$ do not hold. To fulfill $\neg C(x, \mathbf{u})$ we have to ensure that, for all $i, j \in \{1, \dots, k\}$, we have $(\ell_{I_i}^y, \ell_{J_j}^z) \in \mathfrak{S} \wedge (\ell_{I_i}^u, \ell_{J_j}^z) \notin \mathfrak{S} \implies p_{I_i} = q_{J_i}$. Since the considered word is a solution of the PCP, we have $p_{I_i} = q_{J_i}$ for all $i = 1, \dots, k$, hence $\neg C(x, \mathbf{u})$ is satisfied. \square

4 Eliminating Equations and Disequations

In this section, we show that the equations and disequations can always be eliminated from established sequents (in exponential time), while preserving equivalence. The intuition is that the equations can be discarded by instantiating the inductive rules, while the disequations can be replaced by assertions that the considered variables are allocated in disjoint parts of the heap.

We first introduce an additional restriction on pc-SIDs that is meant to ensure that the set of free variables allocated by a predicate atom is the same in every unfolding. The pc-SID satisfying this condition are called *alloc-compatible*. We will show that every pc-SID can be reduced to an equivalent *alloc-compatible* set. Let *alloc* be a function mapping each predicate symbol p to a subset of $\{1, \dots, \#(p)\}$. For any disjunction-free formula ϕ , we denote by *alloc*(ϕ) the set of variables $x \in fv(\phi)$ such that ϕ contains an atom of the form $x \mapsto (y_1, \dots, y_\kappa)$ or $p(z_1, \dots, z_n)$, with $x = z_i$ for some $i \in \text{alloc}(p)$.

Definition 10. *An established pc-SID \mathcal{R} is alloc-compatible if for all rules $\alpha \Leftarrow \phi$ in \mathcal{R} , we have $\text{alloc}(\alpha) = \text{alloc}(\phi)$. A sequent $\phi \vdash_{\mathcal{R}} \Gamma$ is alloc-compatible if \mathcal{R} is alloc-compatible.*

Lemma 11. *There exists an algorithm which, for every sequent $\phi \vdash_{\mathcal{R}} \Gamma$, computes an equivalent alloc-compatible sequent $\phi' \vdash_{\mathcal{R}'} \Gamma'$. Moreover, this algorithm runs in exponential time and $\text{width}(\phi' \vdash_{\mathcal{R}'} \Gamma') = \mathcal{O}(\text{width}(\phi \vdash_{\mathcal{R}} \Gamma)^2)$.*

Proof. We associate all pairs (p, A) where $p \in \mathcal{P}_S$ and $A \subseteq \{1, \dots, \#(p)\}$ with fresh, pairwise distinct predicate symbols $p_A \in \mathcal{P}_S$, with the same arity as p , and we set $\text{alloc}(p_A) = A$. For each disjunction-free formula ϕ , we denote by ϕ^* the set of formulas obtained from ϕ by replacing every predicate atom $p(\mathbf{x})$ by an atom $p_A(\mathbf{x})$ with $A \subseteq \{1, \dots, \#(p)\}$. Let \mathcal{R}' be the set of alloc-compatible rules of the form $p_A(\mathbf{x}) \Leftarrow \psi$, where $p(\mathbf{x}) \Leftarrow \phi$ is a rule in \mathcal{R} and $\psi \in \phi^*$. Note that the symbols p_A may be encoded by words of length $\mathcal{O}(\|p\| + \#(p))$, thus for every $\psi \in \phi^*$ we have $\text{width}(\psi) = \mathcal{O}(\text{width}(\phi)^2)$, hence $\text{width}(\mathcal{R}') = \mathcal{O}(\text{width}(\mathcal{R})^2)$. We show by induction on the satisfiability relation that the following equivalence holds for every structure $(\mathfrak{s}, \mathfrak{h})$: $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ iff there exists $\psi \in \phi^*$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}'} \psi$. For the direct implication, we also prove that $\text{alloc}(\psi) = \{x \in \text{fv}(\phi) \mid \mathfrak{s}(x) \in \text{dom}(\mathfrak{h})\}$.

- The proof is immediate if ϕ is a \mathcal{T} -formula, since $\phi^* = \{\phi\}$, and the truth value of ϕ does not depend on the considered pc-SID. Also, by definition $\text{alloc}(\phi) = \emptyset$ and all the models of ϕ have empty heaps.
- If ϕ is of the form $x \mapsto (y_1, \dots, y_n)$, then $\phi^* = \{\phi\}$ and the truth value of ϕ does not depend on the considered pc-SID. Also, $\text{alloc}(\phi) = \{x\}$ and we have $\text{dom}(\mathfrak{h}) = \{\mathfrak{s}(x)\}$ for every model $(\mathfrak{s}, \mathfrak{h})$ of ϕ .
- Assume that $\phi = p(x_1, \dots, x_{\#(p)})$. If $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ then there exists a formula γ such that $\phi \Leftarrow_{\mathcal{R}} \gamma$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \gamma$. By the induction hypothesis, there exists $\psi \in \gamma^*$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}'} \psi$ and $\text{alloc}(\psi) = \{x \in \text{fv}(\gamma) \mid \mathfrak{s}(x) \in \text{dom}(\mathfrak{h})\}$. Let $A = \{i \in \{1, \dots, \#(p)\} \mid \mathfrak{s}(x_i) \in \text{dom}(\mathfrak{h})\}$, so that $\text{alloc}(\psi) = \{x_i \mid i \in A\}$. By construction $p_A(x_1, \dots, x_n) \Leftarrow \psi$ is alloc-compatible, and therefore $p_A(x_1, \dots, x_n) \Leftarrow_{\mathcal{R}'} \psi$, which entails that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}'} p_A(x_1, \dots, x_n)$. By definition of A , $\text{alloc}(p_A(x_1, \dots, x_n)) = \{x \in \text{fv}(\phi) \mid \mathfrak{s}(x) \in \text{dom}(\mathfrak{h})\}$.

Conversely, assume that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}'} \psi$ for some $\psi \in \phi^*$. Necessarily ψ is of the form $p_A(x_1, \dots, x_n)$ with $A \subseteq \{1, \dots, \#(p)\}$. We have $p_A(x_1, \dots, x_n) \Leftarrow_{\mathcal{R}'} \psi'$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}'} \psi'$ for some formula ψ' . By definition of \mathcal{R}' , we deduce that $p(x_1, \dots, x_n) \Leftarrow_{\mathcal{R}} \gamma$, for some γ such that $\psi \in \gamma^*$. By the induction hypothesis, $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \gamma$, thus $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} p(x_1, \dots, x_{\#(p)})$. Since $p(x_1, \dots, x_{\#(p)}) = \phi$, we have the result.

- Assume that $\phi = \phi_1 * \phi_2$. If $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ then there exist disjoint heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $(\mathfrak{s}, \mathfrak{h}_i) \models_{\mathcal{R}} \phi_i$, for all $i = 1, 2$ and $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$. By the induction hypothesis, this entails that there exist formulas $\psi_i \in \phi_i^*$ for $i = 1, 2$ such that $(\mathfrak{s}, \mathfrak{h}_i) \models_{\mathcal{R}'} \psi_i$ and $\text{alloc}(\psi_i) = \{x \in \text{fv}(\phi_i) \mid \mathfrak{s}(x) \in \text{dom}(\mathfrak{h}_i)\}$. Let $\psi = \psi_1 * \psi_2$. It is clear that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}'} \psi_1 * \psi_2$ and $\text{alloc}(\psi) = \text{alloc}(\psi_1 * \psi_2) = \text{alloc}(\psi_1) \cup \text{alloc}(\psi_2) = \{x \in \text{fv}(\phi_1) \cup \text{fv}(\phi_2) \mid \mathfrak{s}(x) \in \text{dom}(\mathfrak{h})\} = \{x \in \text{fv}(\phi) \mid \mathfrak{s}(x) \in \text{dom}(\mathfrak{h})\}$. Since $\psi_1 * \psi_2 \in \phi^*$, we obtain the result.

Conversely, assume that there exists $\psi \in \phi^*$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}'} \psi$. Then $\psi = \psi_1 * \psi_2$ with $\psi_i \in \phi_i^*$, and we have $(\mathfrak{s}, \mathfrak{h}_i) \models_{\mathcal{R}'} \psi_i$, for $i = 1, 2$ with

$\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$. Using the induction hypothesis, we get that $(\mathfrak{s}, \mathfrak{h}_i) \models_{\mathcal{R}} \phi_i$, hence $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$.

- Assume that $\phi = \exists y.\gamma$. If $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \phi$ then $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} \gamma$, for some store \mathfrak{s}' coinciding with \mathfrak{s} on every variable distinct from y . By the induction hypothesis, this entails that there exists $\psi \in \gamma^*$ such that $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}'} \psi$ and $\text{alloc}(\psi) = \{x \in \text{fv}(\gamma) \mid \mathfrak{s}'(x) \in \text{dom}(\mathfrak{h})\}$. Then $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}'} \exists y.\psi$, and we have $\exists y.\psi \in \phi^*$. Furthermore, $\text{alloc}(\exists y.\psi) = \text{alloc}(\psi) \setminus \{y\} = \{x \in \text{fv}(\gamma) \setminus \{y\} \mid \mathfrak{s}'(x) \in \text{dom}(\mathfrak{h})\} = \{x \in \text{fv}(\phi) \mid \mathfrak{s}(x) \in \text{dom}(\mathfrak{h})\}$.

Conversely, assume that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi$, with $\psi \in \phi^*$. Then ψ is of the form $\exists y.\psi'$, with $\psi' \in \gamma^*$, thus there exists a store \mathfrak{s}' , coinciding with \mathfrak{s} on all variables other than y such that $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} \psi'$. By the induction hypothesis, this entails that $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} \psi$, thus $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists y.\gamma$. Since $\exists y.\gamma = \phi$, we have the result.

Let ϕ', Γ' be the sequence of formulas obtained from ϕ, Γ by replacing every atom α by the disjunction of all the formulas in α^* . It is clear that $\text{width}(\phi' \vdash_{\mathcal{R}'} \Gamma') \leq \text{width}(\phi \vdash_{\mathcal{R}} \Gamma)^2$. By the previous result, $\phi' \vdash_{\mathcal{R}'} \Gamma'$ is equivalent to $\phi \vdash_{\mathcal{R}} \Gamma$, hence $\phi' \vdash_{\mathcal{R}'} \Gamma'$ fulfills all the required properties. Also, since each predicate p is associated with $2^{\#(p)}$ predicates p_A , it is clear that $\phi' \vdash_{\mathcal{R}'} \Gamma'$ can be computed in time $\mathcal{O}(2^{\text{size}(\phi \vdash_{\mathcal{R}} \Gamma)})$. \square

Definition 12. Let $P \subseteq \mathcal{P}_{\mathcal{T}}$. A formula ϕ is P -constrained if for every formula ψ such that $\phi \Leftarrow_{\mathcal{R}} \psi$, and for every symbol $p \in \mathcal{P}_{\mathcal{T}}$ occurring in ψ , we have $p \in P$. A sequent $\phi \vdash_{\mathcal{R}} \Gamma$ is P -constrained if all the formulas in ϕ, Γ are P -constrained.

Theorem 13. Let $P \subseteq \mathcal{P}_{\mathcal{T}}$. There exists an algorithm that transforms every P -constrained established sequent $\phi \vdash_{\mathcal{R}} \Gamma$ into an equivalent $(P \setminus \{\approx, \not\approx\})$ -constrained established sequent $\phi' \vdash_{\mathcal{R}'} \Gamma'$. This algorithm runs in exponential time and $\text{width}(\phi' \vdash_{\mathcal{R}'} \Gamma')$ is polynomial w.r.t. $\text{width}(\phi \vdash_{\mathcal{R}} \Gamma)$.

Proof. We consider a P -constrained established sequent $\phi \vdash_{\mathcal{R}} \Gamma$. This sequent is transformed in several steps.

Step 1. The first step consists in transforming all the formulas in ϕ, Γ into disjunctions of symbolic heaps. Then for every symbolic heap γ occurring in the obtained sequent, we add all the variables freely occurring in ϕ or Γ as parameters of every predicate symbol occurring in unfoldings of γ (their arities are updated accordingly, and these variables are passed as parameters to each recursive call of a predicate symbol). We obtain an equivalent sequent $\phi_1 \vdash_{\mathcal{R}_1} \Gamma_1$, and if $v = \text{card}(\text{fv}(\phi) \cup \text{fv}(\Gamma))$ denotes the total number of free variables occurring in ϕ, Γ , then it is easy to check (since the size of each of these variables is bounded by $\text{width}(\phi \vdash_{\mathcal{R}} \Gamma)$) that $\text{width}(\phi_1 \vdash_{\mathcal{R}_1} \Gamma_1) \leq v \cdot \text{width}(\phi \vdash_{\mathcal{R}} \Gamma)^2$. By Definition 7, we have $v \leq \text{width}(\phi \vdash_{\mathcal{R}} \Gamma)$, thus $\text{width}(\phi_1 \vdash_{\mathcal{R}_1} \Gamma_1) = \mathcal{O}(\text{width}(\phi \vdash_{\mathcal{R}} \Gamma)^3)$.

Step 2. All the equations involving an existential variable can be eliminated in a straightforward way by replacing each formula of the form $\exists x.(x \approx y * \phi)$ with $\phi\{x \leftarrow y\}$. We then replace every formula $\exists \mathbf{y}.\phi$ with free variables x_1, \dots, x_n by the disjunction of all the formulas of the form $\exists \mathbf{z}.\phi\sigma * \mathbf{z} \in \mathbf{z}, \mathbf{z}' \in \mathbf{z} \cup \{x_1, \dots, x_n\}, \mathbf{z} \neq \mathbf{z}' \not\approx \mathbf{z}'$, where σ is a substitution such that $\text{dom}(\sigma) \subseteq \mathbf{y}, \mathbf{z} = \mathbf{y} \setminus \text{dom}(\sigma)$ and $\text{img}(\sigma) \subseteq$

$\mathbf{y} \cup \{x_1, \dots, x_n\}$. Similarly we replace every rule $p(x_1, \dots, x_n) \Leftarrow \exists \mathbf{y}. \phi$ by the set of rules $p(x_1, \dots, x_n) \Leftarrow \exists \mathbf{z}. \phi \sigma * \#_{z \in \mathbf{z}, z' \in \mathbf{z} \cup \{x_1, \dots, x_n\}, z \neq z'} z \not\approx z'$, where σ is any substitution satisfying the conditions above. Intuitively, this transformation ensures that all existential variables are associated to pairwise distinct locations, also distinct from any location associated to a free variable. The application of the substitution σ captures all the rule instances for which this condition does not hold, by mapping all variables that are associated with the same location to a unique representative. We denote by $\phi_2 \vdash_{\mathcal{R}_2} \Gamma_2$ the sequent thus obtained. Let v' be the maximal number of existential variables occurring in a rule in \mathcal{R} . We have $v' \leq \text{width}(\phi \vdash_{\mathcal{R}} \Gamma)$ (since the transformation in Step 1 adds no existential variable). Since at most one disequation is added for every pair of variables, and since the size of every variable is bounded by $\text{width}(\phi \vdash_{\mathcal{R}} \Gamma)$, it is clear that $\text{width}(\phi_2 \vdash_{\mathcal{R}_2} \Gamma_2) = \text{width}(\phi_1 \vdash_{\mathcal{R}_1} \Gamma_1) + v' \cdot (v + v') \cdot (1 + 2 * \text{width}(\phi \vdash_{\mathcal{R}} \Gamma)) = \mathcal{O}(\text{width}(\phi \vdash_{\mathcal{R}} \Gamma)^3)$.

Step 3. We replace every atom $\alpha = p(x_1, \dots, x_n)$ occurring in ϕ_2, Γ_2 or \mathcal{R}_2 with pairwise distinct variables x_{i_1}, \dots, x_{i_m} (with $m \leq n$ and $i_1 = 1$), by an atom $p_\alpha(x_{i_1}, \dots, x_{i_m})$, where p_α is a fresh predicate symbol, associated with rules of the form $p_\alpha(y_{i_1}, \dots, y_{i_m}) \Leftarrow \psi\{y_i \leftarrow x_i \mid i \in \{1, \dots, n\}\} \theta$, where $p(y_1, \dots, y_n) \Leftarrow \psi$ is a rule in \mathcal{R} and θ denotes the substitution $\{x_{i_k} \leftarrow y_{i_k} \mid i \in \{1, \dots, m\}\}$. By construction, $p_\alpha(x_{i_1}, \dots, x_{i_m})$ is equivalent to α . We denote by $\phi_3 \vdash_{\mathcal{R}_3} \Gamma_3$ the resulting sequent. It is clear that $\phi_3 \vdash_{\mathcal{R}_3} \Gamma_3$ is equivalent to $\phi \vdash_{\mathcal{R}} \Gamma$.

By a straightforward induction on the derivation, we can show that all atoms occurring in an unfolding of the formulas in the sequent $\phi_3 \vdash_{\mathcal{R}_3} \Gamma_3$ are of the form $q(y_1, \dots, y_{\#(q)})$, where $y_1, \dots, y_{\#(q)}$ are pairwise distinct, and that the considered unfolding also contains the disequation $y_i \not\approx y_j$, for all $i \neq j$ such that either y_i or y_j is an existential variable (note that if y_i and y_j are both free then $y_i \not\approx y_j$ is valid, since the considered stores are injective). This entails that the rules that introduce a trivial equality $u \approx v$ with $u \neq v$ are actually redundant, since unfolding any atom $q(y_1, \dots, y_{\#(q)})$ using such a rule yields a formula that is unsatisfiable. Consequently such rules can be eliminated without affecting the status of the sequent. All the remaining equations are of form $u \approx u$ hence can be replaced by **emp**. We may thus assume that the sequent $\phi_3 \vdash_{\mathcal{R}_3} \Gamma_3$ contains no equality. Note that by the above transformation all existential variables must be interpreted as pairwise distinct locations in any interpretation, and also be distinct from all free variables. It is easy to see that the fresh predicates p_α may be encoded by words of size at most $\text{width}(\phi \vdash_{\mathcal{R}} \Gamma)$, thus $\text{width}(\phi_3 \vdash_{\mathcal{R}_3} \Gamma_3) \leq \text{width}(\phi \vdash_{\mathcal{R}} \Gamma) \cdot \text{width}(\phi_2 \vdash_{\mathcal{R}_2} \Gamma_2) = \mathcal{O}(\text{width}(\phi \vdash_{\mathcal{R}} \Gamma)^4)$. By Lemma 11, we may assume that $\phi_3 \vdash_{\mathcal{R}_3} \Gamma_3$ is *alloc-compatible* (note that the transformation given in the proof of Lemma 11 does not affect the disequations occurring in the rules).

Step 4. We now ensure that all the locations that are referred to are allocated. Consider a symbolic heap γ occurring in ϕ_3, Γ_3 and any \mathcal{R}_3 -model $(\mathfrak{s}, \mathfrak{h})$ of γ , where \mathfrak{s} is injective. For the establishment condition to hold, the only unallocated locations in \mathfrak{h} of γ must correspond to locations $\mathfrak{s}(x)$ where x is a free variable. We assume the sequent contains a free variable u such that, for ev-

ery tuple $(\ell_0, \dots, \ell_\kappa) \in \mathfrak{h}$, we have $\mathfrak{s}(u) = \ell_\kappa$. This does not entail any loss of generality, since we can always add a fresh variable u to the considered problem: after Step 1, u is passed as a parameter to all predicate symbols, and we may replace every points-to atom $z_0 \mapsto (z_1, \dots, z_\kappa)$ occurring in ϕ_3, Γ_3 or \mathcal{R}_3 , by $z_0 \mapsto (z_1, \dots, z_\kappa, u)$ (note that this increases the value of κ by 1). It is clear that this ensures that \mathfrak{h} and u satisfy the above property. We also assume, w.l.o.g., that the sequent contains at least one variable u' distinct from u . Note that, since \mathfrak{s} is injective, the tuple $(\mathfrak{s}(u'), \dots, \mathfrak{s}(u'))$ cannot occur in \mathfrak{h} , because its last component is distinct from $\mathfrak{s}(u)$. We then denote by $\phi_4 \vdash_{\mathcal{R}_4} \Gamma_4$ the sequent obtained from $\phi_3 \vdash_{\mathcal{R}_3} \Gamma_3$ by replacing every symbolic heap γ in ϕ_3, Γ_3 by $(\ast_{x \in (fv(\phi_3) \cup fv(\Gamma_3)) \setminus alloc(\gamma)} x \mapsto (u', \dots, u')} * \gamma$. It is straightforward to check that $(\mathfrak{s}, \mathfrak{h}) \models \gamma$ iff there exists an extension \mathfrak{h}' of \mathfrak{h} such that $(\mathfrak{s}, \mathfrak{h}') \models (\ast_{x \in (fv(\phi_3) \cup fv(\Gamma_3)) \setminus alloc(\gamma)} x \mapsto (u', \dots, u')} * \gamma$, with $loc(\mathfrak{h}) = loc(\mathfrak{h}') = dom(\mathfrak{h}')$ and $\mathfrak{h}'(\ell) = (\mathfrak{s}(u'), \dots, \mathfrak{s}(u'))$ for all $\ell \in dom(\mathfrak{h}') \setminus dom(\mathfrak{h})$. This entails that $\phi_4 \vdash_{\mathcal{R}_4} \Gamma_4$ is valid if and only if $\phi \vdash_{\mathcal{R}} \Gamma$ is valid.

Consider a formula γ in ϕ_4, Γ_4 and some unfolding γ' of γ . Thanks to the transformation in this step and the establishment condition, if γ' contains a (free or existential) variable x then it also contains an atom $x' \mapsto \mathbf{y}$ and a separating conjunction of equations χ such that $\chi \models_{\mathcal{T}} x \approx x'$. Since all equations have been removed, $\chi = \mathbf{emp}$, thus $x = x'$. Consequently, if γ' contains a disequation $x_1 \not\approx x_2$ with $x_1 \neq x_2$, then it also contains atoms $x_1 \mapsto \mathbf{y}_1$ and $x_2 \mapsto \mathbf{y}_2$. This entails that the disequation $x_1 \not\approx x_2$ is redundant, since it is a logical consequence of $x_1 \mapsto \mathbf{y}_1 * x_2 \mapsto \mathbf{y}_2$. We deduce that the satisfiability status of $\phi_4 \vdash_{\mathcal{R}_4} \Gamma_4$ is preserved if all disequations are replaced by \mathbf{emp} . \square

Example 14. We illustrate all of the steps in the proof above.

Step 1. Consider the sequent $p(x_1, x_2) \vdash_{\mathcal{R}} r(x_1) * r(x_2)$, where \mathcal{R} is defined as follows: $\mathcal{R} = \{r(x) \leftarrow x \mapsto (x)\}$. After Step 1 we obtain the sequent $p(x_1, x_2) \vdash_{\mathcal{R}_1} r'(x_1, x_2) * r'(x_2, x_1)$, where $\mathcal{R}_1 = \{r'(x, y) \leftarrow x \mapsto (x)\}$.

Step 2. This step transforms the formula $\exists y_1 \exists y_2. p(x, y_1) * p(x, y_2)$ into the disjunction:

$$\begin{aligned} \exists y_1, y_2. p(x, y_1) * p(x, y_2) * y_1 \not\approx y_2 * y_1 \not\approx x * y_2 \not\approx x \vee \\ \exists y_2. p(x, x) * p(x, y_2) * y_2 \not\approx x \vee \\ \exists y_1. p(x, y_1) * p(x, x) * y_1 \not\approx x \vee \\ p(x, x) * p(x, x) \end{aligned}$$

Similarly, the rule $p(x) \leftarrow \exists z \exists u. x \mapsto (z) * q(z, u)$ is transformed into the set:

$$\begin{aligned} p(x) \leftarrow x \mapsto (x) * q(x, x) \\ p(x) \leftarrow \exists z. x \mapsto (z) * q(z, x) * z \not\approx x \\ p(x) \leftarrow \exists u. x \mapsto (x) * q(x, u) * u \not\approx x \\ p(x) \leftarrow \exists z \exists u. x \mapsto (z) * q(z, u) * z \not\approx x * u \not\approx x * z \not\approx u \end{aligned}$$

Step 3. Assume that \mathcal{R} contains the rules $p(y_1, y_2, y_3) \leftarrow y_1 \mapsto (y_2) * q(y_2, y_3) * y_1 \approx y_3$ and $p(y_1, y_2, y_3) \leftarrow y_1 \mapsto (y_2) * r(y_2, y_3) * y_1 \approx y_2$ and consider the sequent $p(x, y, x) \vdash_{\mathcal{R}} \mathbf{emp}$. Step 3 generates the sequent $p_\alpha(x, y) \vdash_{\mathcal{R}'} \mathbf{emp}$

(with $\alpha = p(x, y, x)$) where \mathcal{R}' contains the rules $p_\alpha(y_1, y_2) \Leftarrow y_1 \mapsto (y_2) * q(y_2, y_1) * y_1 \approx y_1$ and $p_\alpha(y_1, y_2) \Leftarrow y_1 \mapsto (y_2) * r(y_2, y_1) * y_1 \approx y_2$. The second rule is redundant, because $p_\alpha(y_1, y_2)$ is used only in a context where $y_1 \not\approx y_2$ holds.

Step 4. Let $\gamma = p(x, y, z, z') * q(x, y, z, z') * z' \mapsto (z')$, assume $\text{alloc}(\gamma) = \{x, z\}$, and consider the sequent $\gamma \vdash_{\mathcal{R}} \text{emp}$. Then γ is replaced by $p(x, y, z, z', u) * q(x, y, z, z', u) * z' \mapsto (z', u) * u \mapsto (x, x) * y \mapsto (x, x)$ (all non-allocated variables are associated with (x, x) , where x plays the rôle of the variable u' in Step 4 above). Also, every points-to atom $z_0 \mapsto (z_1)$ in \mathcal{R} is replaced by $z_0 \mapsto (z_1, u)$.

5 Discussion

The presented undecidability result is very tight. Theorem 9 applies to most theories and the proof only uses very simple data structures (namely simply linked lists). The proof of Theorem 9 could be adapted (at the cost of cluttering the presentation) to handle quantifier-free entailments and even simpler inductive systems with at most one predicate atom on the right-hand side of each rule, in the spirit of word automata.

Our logic has only one sort of variables, denoting locations, thus one cannot directly describe structures in which the heap maps locations to tuples containing both locations and data, ranging over disjoint domains. This is actually not restrictive: indeed, data can be easily encoded in our framework by considering a non-injective function $\mathbf{d}(x)$ mapping locations to data, and adding theory predicates constructed on this function, such as $\mathbf{d}(x) \approx \mathbf{d}(y)$ to state that two (possibly distinct) locations x, y are mapped to the same element. The obtained theory falls within the scope of Theorem 9 (using $\mathbf{d}(x) \approx \mathbf{d}(y)$ as the relation $S(x, y)$), provided the domain of the data is infinite. This shows that entailments with data disjoint from locations are undecidable, even if the theory only contains equations and disequations, except when the data domain is finite.

Acknowledgments. This work has been partially funded by the the French National Research Agency (ANR-21-CE48-0011). The authors wish to thank Radu Iosif for his comments on an earlier version of the paper and for fruitful discussions.

References

1. J. Berdine, C. Calcagno, and P. W. O’Hearn. A decidable fragment of separation logic. In *Proc. of FSTTCS’04*, volume 3328 of *LNCS*. Springer, 2004.
2. Cristiano Calcagno, Hongseok Yang, and Peter W O’hearn. Computability and complexity results for a spatial assertion language for data structures. In *FST TCS 2001, Proceedings*, pages 108–119. Springer, 2001.
3. B. Cook, C. Haase, J. Ouaknine, M. J. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *Proc. of CONCUR’11*, volume 6901 of *LNCS*. Springer, 2011.

4. Stéphane Demri, Didier Galmiche, Dominique Larchey-Wendling, and Daniel Méry. Separation logic with one quantified variable. In *CSR'14*, volume 8476 of *LNCS*, pages 125–138. Springer, 2014.
5. 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.
6. 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.
7. Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. Compositional entailment checking for a fragment of separation logic. *Formal Methods Syst. Des.*, 51(3):575–607, 2017.
8. Constantin Enea, Mihaela Sighireanu, and Zhilin Wu. On automated lemma generation for separation logic with inductive definitions. In *ATVA 2015, Proceedings*, pages 80–96, 2015.
9. 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.
10. Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In Franck Cassez and Jean-François Raskin, editors, *ATVA 2014, Proceedings*, volume 8837 of *LNCS*, pages 201–218. Springer, 2014.
11. 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.
12. Quang Loc Le. Compositional satisfiability solving in separation logic. In Fritz Henglein, Sharon Shoham, and Yakir Vizel, editors, *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings*, volume 12597 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2021.
13. Peter W. O’Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In Laurent Fribourg, editor, *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings*, volume 2142 of *LNCS*, pages 1–19. Springer, 2001.
14. Jens Pagel, Christoph Matheja, and Florian Zuleger. Complete entailment checking for separation logic with inductive definitions, 2020. [arXiv:2002.01202](https://arxiv.org/abs/2002.01202).
15. Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic modulo theories. In Chung-chieh Shan, editor, *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings*, volume 8301 of *LNCS*, pages 90–106. Springer, 2013.
16. Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic using SMT. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *LNCS*, pages 773–789. Springer, 2013.
17. Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and Parthasarathy Madhusudan. Natural proofs for structure, data, and separation. In Hans-Juergen Boehm and Cormac Flanagan, editors, *ACM SIGPLAN PLDI '13*, pages 231–242. ACM, 2013.

18. J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*, 2002.
19. Zhaowei Xu, Taolue Chen, and Zhilin Wu. Satisfiability of compositional separation logic with tree predicates and data constraints. In Leonardo de Moura, editor, *CADE 26*, volume 10395 of *LNCS*, pages 509–527. Springer, 2017.