What Is Decidable in Separation Logic Beyond Progress, Connectivity and Establishment?

Tanguy Bozec, Nicolas Peltier, Quentin Petitjean, Mihaela Sighireanu

What is Decidable in Separation Logic Beyond Progress, Connectivity and Establishment? ⋆

Tanguy Bozec[1], Nicolas Peltier[1][0000−0002−8943−7000], Quentin Petitjean[2]✉[0009−0004−6504−8336], and Mihaela Sighireanu[2][0000−0002−1925−089X]

[1] Univ. Grenoble Alpes, CNRS, LIG, 38000 Grenoble France
[2] Univ. Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190 Gif-sur-Yvette, France

Abstract. The predicate definitions in Separation Logic (SL) play an important role: they capture a large spectrum of unbounded heap shapes due to their inductiveness. This expressiveness power comes with a limitation: the entailment problem is undecidable if predicates have general inductive definitions (ID). Iosif et al. [10] proposed syntactic and semantic conditions, called PCE, on the ID of predicates to ensure the decidability of the entailment problem. We provide a (possibly nonterminating) algorithm to transform arbitrary ID into equivalent PCE definitions when possible. We show that the existence of an equivalent PCE definition for a given ID is undecidable, but we identify necessary conditions that are decidable. The algorithm has been implemented, and experimental results are reported on a benchmark, including significant examples from SL-COMP.

Keywords: Separation logic · Inductive definitions · Bounded treewidth fragment · PCE fragment · Symbolic heaps · Decision procedures

1 Introduction

Separation logic (SL) [11,13] is widely used in verification to reason about programs manipulating dynamically allocated memory. Formulas in SL are defined from atoms of the form $x \to (y_1, \ldots, y_k)$, stating that at location (i.e., a memory address), x is allocated a memory block containing the tuple built from values of $y_1, \ldots, y_k$, and $\mathtt{emp}$, stating that the heap is empty, i.e., that there are no allocated locations. SL includes the standard logical connectives and quantifiers, together with a special connective $\varphi_1 \star \varphi_2$, called separating conjunction, asserting that formulas φ_1 and φ_2 are satisfied on disjoint parts of the heap. This particular feature of the logic ensures the scalability of program analyses by enabling *local reasoning*: the properties of a program may be asserted and established by referring only to the part of the heap that is affected by the program. To specify recursive data structures, the SL formulas include predicate atoms defined by inductive rules with a fixpoint semantics. For instance, list segments from x to y may be defined by the following rules:

$$\mathtt{ls}(x, y) \Leftarrow \mathtt{emp} \star x \approx y\,, \qquad \mathtt{ls}(x, y) \Leftarrow \exists z.\,(x \to (z) \star \mathtt{ls}(z, y))\,. \tag{1}$$

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

Many problems in verification boil down to checking the validity of entailments between formulas in SL. In general, unsurprisingly, entailment is undecidable. However, several fragments have been identified for which the entailment problem is decidable. Among these fragments, the so-called *PCE fragment* is one of the most expressive ones [10]. Decidability was initially established by reduction to monadic second-order logic on graphs with bounded treewidth. Later, more efficient algorithms were proposed [12,4], and the problem turned out to be 2-ExpTime-complete [3]. The PCE fragment is defined by restricting the syntax and the semantics of the inductive rules defining the predicates. Each rule is required to satisfy three properties (formally defined later): (P)rogress, (C)onnectivity and (E)stablishment. Informally, the conditions respectively assert that: (P) every rule allocates *exactly* one location; (C) the allocated locations have a tree-shaped structure which mimics the call tree of the predicates, and (E) every location not associated with a free variable is (eventually) allocated. A PCE formula is a formula in which all predicates are defined by PCE rules. Most usual data structures in programming can be defined using PCE rules. However, the PCE conditions impose rigid constraints on the rules' syntax, which are not necessarily satisfied in practice by user-provided rules. For instance, the above rules of $\mathtt{ls}$ (Eq. (1)) are *not PCE* (because the first rule of $\mathtt{ls}$ allocates no location), while the following ones, although specifying non-empty list segments, are PCE:

$$\mathtt{ls}^+(x,y) \Leftarrow x \to (y)\,, \qquad \mathtt{ls}^+(x,y) \Leftarrow \exists z.\,(x \to (z) \star \mathtt{ls}^+(z,y))\,. \tag{2}$$

The non-PCE formula $\mathtt{ls}(x,y)$ can then be written as a PCE formula $(\mathtt{emp} \star x \approx y) \vee \mathtt{ls}^+(x,y)$. Other, rather natural, definitions of $\mathtt{ls}^+$ can be given, which are not PCE (the second rule of $\mathtt{ls}^m$ allocates no location, and the second rule of $\mathtt{ls}^e$ is not connected):

$$\mathtt{ls}^m(x,y) \Leftarrow x \to (y)\,, \qquad \mathtt{ls}^m(x,y) \Leftarrow \exists z.\,(\mathtt{ls}^m(x,z) \star \mathtt{ls}^m(z,y))\,, \tag{3}$$

$$\mathtt{ls}^e(x,y) \Leftarrow x \to (y)\,, \qquad \mathtt{ls}^e(x,y) \Leftarrow \exists z.\,(\mathtt{ls}^e(x,z) \star z \to (y))\,. \tag{4}$$

Similarly, the following definition of lists of odd length is not PCE:

$$\mathtt{ls}^1(x,y) \Leftarrow x \to (y)\,, \qquad \mathtt{ls}^1(x,y) \Leftarrow \exists z_1,z_2.\,(x \to (z_1) \star z_1 \to (z_2) \star \mathtt{ls}^1(z_2,y))\,, \tag{5}$$

but it is clear that it can be transformed into a PCE definition by replacing the inductive rule (at right) with the following ones:

$$\mathtt{ls}^1(x,y) \Leftarrow \exists z_1.\,(x \to (z_1) \star \mathtt{ls}^2(z_1,y))\,, \qquad \mathtt{ls}^2(z_1,y) \Leftarrow \exists z_2.\,(z_1 \to (z_2) \star \mathtt{ls}^1(z_2,y))\,. \tag{6}$$

A natural question thus arises, which has not been investigated so far: can algorithms be provided to identify whether a formula can be rewritten into an equivalent PCE formula and to effectively compute such a formula (and the associated inductive rules) if possible? The present paper aims to address these issues.

Contributions. We first observe that the PCE problem — i.e., the problem of testing whether a given formula admits an equivalent PCE formula — is undecidable. The result follows from the undecidability of testing whether context-free grammar is regular. Then, we provide a procedure for transforming some formulas that do not satisfy the

PCE conditions into equivalent PCE formulas. Equivalence is guaranteed in all cases, but the procedure does not always terminate. We also identify cases for which the formulas cannot possibly admit any equivalent PCE formula. More precisely, we identify a property called *PCE-compatibility*, which is strictly weaker than PCE, in the sense that any formula that is equivalent to a PCE formula is PCE-compatible, but the converse does not hold, and we prove that this property is decidable. To sum up, given a formula φ, the procedure may either terminate with a negative answer (if φ is not PCE-compatible) or may terminate with a positive answer and output a PCE formula equivalent to φ or may diverge (if φ is PCE-compatible, but no equivalent PCE formula can be obtained).

To our knowledge, there is no published work on this topic. In [9], the authors proposed inductive definitions (ID, termed "recursive definitions" in [10]) with syntactic restrictions incomparable to PCE since they require linearity and compositionality of the ID to obtain decidability of the entailment problem. This class of ID (disregarding data constraints) may be translated by our procedure into PCE form, i.e., they are PCE-compatible. In [5], other decidable fragments of entailment problems are considered, which do not fulfil the PCE conditions but can be reduced to PCE entailment. Unlike the present approach, the reduction proposed in [5] does not preserve the equivalence of formulas. In [4], the establishment condition is replaced by a condition on the equalities occurring in the problem.

Due to the lack of space and readability, some basic notations, detailed proofs of our results, and additional information about the implementation and experimental results are included in the appendix.

2 Separation Logic with Inductive Definitions

We recall the definition of the syntax and semantics of SL with inductive definitions. Missing definitions, further explanations and examples can be found in [10]. We briefly review standard notations: $\mathrm{card}(A)$ denotes the cardinality of set A, and $A \uplus B$ denotes the disjoint union of sets A and B. The set $\{x \in \mathbb{Z} \mid i \leq x \leq j\}$ is denoted by $[\![i, j]\!]$. The domain of a function f is written $\mathrm{dom}(f)$. The equivalence class of an element x w.r.t. some equivalence relation $\bowtie$ is written $[x]_\bowtie$, and the set $\{[x]_\bowtie \mid x \in S\}$ is written $\overline{S}_\bowtie$. The relation $\bowtie$ will sometimes be omitted if it is clear from the context. We often identify an equivalence relation $\bowtie$ with the set of its equivalence classes. For any binary relation $\rightarrow$, we denote by $\rightarrow^*$ its reflexive and transitive closure. A set R is a set of *roots* for $\rightarrow$ if for all elements x, y such that $x \rightarrow y$, there exists $r \in R$ such that $r \rightarrow^* x$. It is *minimal* if, moreover, there is no set of roots R' such that $R' \subset R$ (where $\subset$ denotes strict inclusion).

Definition 1 (SL formulas). *Let $\mathcal{V}$ be a countably infinite set of* variables*, and let $\mathcal{P}$ be a set of* spatial predicate symbols*, where each symbol $p \in \mathcal{P}$ is associated with a unique arity $\#(p)$ (with countably infinite sets of predicate symbols of each arity). The set of* SL-formulas *(or simply formulas) φ is inductively defined as follows:*

$$\varphi := \mathtt{emp} \mid x \rightarrow (y_1, \ldots, y_k) \mid x \approx y \mid x \not\approx y \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \star \varphi_2 \mid p(x_1, \ldots, x_{\#(p)}) \mid \exists x.\, \varphi_1$$

where φ_1, φ_2 are formulas, $p \in \mathcal{P}$, $k \in \mathbb{N}$ and $x, y, x_1, \ldots, x_{\#(p)}, y_1, \ldots, y_k \in \mathcal{V}$.

Note that negations are not supported. The considered fragment is similar to that of [4] (with disjunctions added), with the slight difference that points-to atoms $x \to (y_1, \ldots, y_k)$ contain tuples of arbitrary length $k \geq 0$. Let $fv(\varphi)$ be the set of free variables in φ. A *substitution* σ is a function from variables to variables; its domain $\mathrm{dom}(\sigma)$ is the set of variables x such that $\sigma(x) \neq x$, and its image $\mathrm{img}(\sigma) = \sigma(\mathrm{dom}(\sigma))$. For any expression (variable, tuple or set 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 $\sigma(x)$. A *symbolic heap* is a formula containing no occurrence of $\vee$. By distributivity of $\star$ and $\exists$ over $\vee$, any formula φ can be reduced to an equivalent disjunction of symbolic heaps, denoted by $dnf(\varphi)$. An *inductive rule associated with the predicate* p has the form $p(x_1, \ldots, x_n) \Leftarrow \varphi$, where $x_1, \ldots, x_n$ are pairwise distinct variables, $n = \#(p)$, and φ is a formula with $fv(\varphi) \subseteq \{x_1, \ldots, x_n\}$. If φ is not a symbolic heap, then $p(x_1, \ldots, x_n) \Leftarrow \varphi$ may be replaced by the rules $\{p(x_1, \ldots, x_n) \Leftarrow \varphi_i \mid i \in [\![1, m]\!]\}$, where $\varphi_1, \ldots, \varphi_m$ are symbolic heaps such that $\bigvee_{i=1}^{m} \varphi_i$ is $dnf(\varphi)$. We assume in the following that this transformation is applied eagerly to every rule. A *set of inductive definitions* (SID) $\mathcal{R}$ is a set of inductive rules such that, for all predicates p, $\mathcal{R}$ contains finitely many rules associated with p. We write $p(y_1, \ldots, y_n) \Leftarrow_{\mathcal{R}} \psi$ if $\mathcal{R}$ contains a rule $p(x_1, \ldots, x_n) \Leftarrow \varphi$, with $\psi = \varphi\{x_i \mapsto y_i \mid i \in [\![1, n]\!]\}$.

Definition 2 (SL structure). *Let $\mathcal{L}$ be a countably infinite set of so-called* locations*. An SL-structure is a pair (s, h) where s is a* store*, i.e., a partial function from $\mathcal{V}$ to $\mathcal{L}$, and h is a* heap*, i.e., a partial finite function from $\mathcal{L}$ to $\mathcal{L}^*$, which can be written as a relation: $\mathsf{h}(\ell) = (\ell_1, \ldots, \ell_k)$ iff $(\ell, \ell_1, \ldots, \ell_k) \in \mathsf{h}, k \in \mathbb{N}$.*

For any heap h, we let $ref(\mathsf{h}) = \{\ell \mid \ell_0 \in \mathrm{dom}(\mathsf{h}), \ell \text{ occurs in } \mathsf{h}(\ell_0)\}$, $loc(\mathsf{h}) = ref(\mathsf{h}) \cup \mathrm{dom}(\mathsf{h})$ and $dgl(\mathsf{h}) = loc(\mathsf{h}) \setminus \mathrm{dom}(\mathsf{h})$ (for "dangling pointers"). Locations in $\mathrm{dom}(\mathsf{h})$ and variables x such that $\mathsf{s}(x) \in \mathrm{dom}(\mathsf{h})$ are *allocated*. We write $\ell \to_{\mathsf{h}} \ell'$ iff $\ell \in \mathrm{dom}(\mathsf{h})$, and ℓ' occurs in $\mathsf{h}(\ell)$.

Definition 3 (SL semantics). *Given a formula φ, a SID $\mathcal{R}$ and a structure (s, h) with $fv(\varphi) \subseteq \mathrm{dom}(\mathsf{s})$, the satisfaction relation $\models_{\mathcal{R}}$ is inductively defined as the least relation such that $(\mathsf{s}, \mathsf{h}) \models_{\mathcal{R}} \varphi$ iff one of the following conditions holds:*

- *$\varphi = \mathtt{emp}$ and $\mathsf{h} = \emptyset$; or $\varphi = (x \to (y_1, \ldots, y_k))$ and $\mathsf{h} = \{(\mathsf{s}(x), \mathsf{s}(y_1), \ldots, \mathsf{s}(y_k))\}$;*
- *$\varphi = (x \approx y)$, $\mathsf{s}(x) = \mathsf{s}(y)$ and $\mathsf{h} = \emptyset$; or $\varphi = (x \not\approx y)$, $\mathsf{s}(x) \neq \mathsf{s}(y)$ and $\mathsf{h} = \emptyset$;*
- *$\varphi = \varphi_1 \vee \varphi_2$ and $(\mathsf{s}, \mathsf{h}) \models_{\mathcal{R}} \varphi_i$, for some $i \in \{1, 2\}$; or $\varphi = \varphi_1 \star \varphi_2$ and there exist disjoint domain heaps $\mathsf{h}_1, \mathsf{h}_2$ such that $\mathsf{h} = \mathsf{h}_1 \uplus \mathsf{h}_2$ and $(\mathsf{s}, \mathsf{h}_i) \models_{\mathcal{R}} \varphi_i$, for all $i \in \{1, 2\}$;*
- *$\varphi = \exists x. \psi$ and $(\mathsf{s}', \mathsf{h}) \models_{\mathcal{R}} \psi$, for some s' matching s on all variables distinct from x;*
- *$\varphi = p(x_1, \ldots, x_{\#(p)})$, $p \in \mathcal{P}$ and $(\mathsf{s}, \mathsf{h}) \models_{\mathcal{R}} \psi$ for some ψ such that $\varphi \Leftarrow_{\mathcal{R}} \psi$.*

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

Definition 4 (SL model). *An $\mathcal{R}$-*model *of φ is a structure (s, h) such that $(\mathsf{s}, \mathsf{h}) \models_{\mathcal{R}} \varphi$. Given two pairs $(\varphi, \mathcal{R})$ and $(\varphi', \mathcal{R}')$, where φ, φ' are formulas and $\mathcal{R}, \mathcal{R}'$ are SID, we write $(\varphi, \mathcal{R}) \equiv (\varphi', \mathcal{R}')$ iff $(\mathsf{s}, \mathsf{h}) \models_{\mathcal{R}} \varphi \iff (\mathsf{s}, \mathsf{h}) \models_{\mathcal{R}'} \varphi'$ holds for all structures (s, h).*

We emphasize that the atoms $x \approx y$ or $x \not\approx y$ only hold for empty heaps (this convention simplifies notations as it avoids the use of standard conjunction). Formulas are taken modulo the usual properties of SL connectives: associativity and commutativity of $\star$ and $\vee$, neutrality of $\mathtt{emp}$ for $\star$, commutativity of $\approx, \not\approx$, and also modulo prenex form and α-renaming. We also assume that bound variables are renamed to avoid any name collision. Rules are defined up to a renaming of free variables.

3 The PCE Problem

We now recall the conditions from [10], ensuring the decidability of the entailment problem.

Definition 5 (PCE rule and SID). *Let r be a function mapping every spatial predicate $p \in \mathcal{P}$ to an element of $[\![1, \#(p)]\!]$. For any atom $p(x_1, \ldots, x_n)$, the variable $x_{r(p)}$ is the* root *of $p(x_1, \ldots, x_n)$, and the root of an atom $x \to (y_1, \ldots, y_k)$ is x. A rule $p(x_1, \ldots, x_n) \Leftarrow \varphi$ is PCE w.r.t. some SID $\mathcal{R}$ if it is:*

- progressing, *i.e., φ is of the form $\exists u_1, \ldots, u_m. (x_i \to (y_1, \ldots, y_k) \star \psi)$, where $m \geq 0$, ψ is a formula with no occurrence of $\to, \exists, \vee$, and $i = r(p)$;*
- connected, *i.e., moreover, all spatial predicate atoms occurring in ψ are of the form $q(z_1, \ldots, z_{\#(q)})$ with $z_{r(q)} \in \{y_1, \ldots, y_k\}$;*
- established, *i.e., moreover, for all $i \in [\![1, m]\!]$, and for all structures $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \psi$, either $\mathfrak{s}(u_i) \in \mathrm{dom}(\mathfrak{h})$ or $\mathfrak{s}(u_i) \in \{\mathfrak{s}(x_j) \mid j \in [\![1, n]\!]\}$.*

A SID $\mathcal{R}$ is PCE if every rule is PCE w.r.t. $\mathcal{R}$. A formula φ is PCE if every predicate used in φ is defined by PCE rules.

The problem we are investigating in the present paper is the following:

Definition 6 (PCE problem). *Given a pair $(\varphi, \mathcal{R})$, the* PCE problem *lies in deciding whether there exists a formula φ' and a PCE SID $\mathcal{R}'$ such that $(\varphi, \mathcal{R}) \equiv (\varphi', \mathcal{R}')$.*

Assuming that φ is atomic is sufficient (complex formulas may be introduced by inductive rules), but the possibility that φ' is non-atomic allows for greater expressiveness. If one restricts oneself to list-shaped structures denoting words, then the PCE conditions essentially state that the set of denoted words is regular. This entails the following result, obtained by reduction from the regularity of context-free languages (see App. A):

Theorem 1. *The PCE problem is undecidable.*

It may be observed that the structures $(\mathfrak{s}, \mathfrak{h})$ satisfying PCE pairs $(\varphi, \mathcal{R})$ necessarily satisfy two essential properties. First, due to the connectivity condition, these structures necessarily admit a bounded number of roots, which correspond to locations assigned by $\mathfrak{s}$ to (possibly quantified) variables occurring inside φ (at some root position in a predicate or points-to atom, as defined in Def. 5).

Structures with multiple roots are permitted (e.g., doubly linked lists), but due to the connectivity condition, if x is the root of an atom φ, then, for every model $(\mathfrak{s}, \mathfrak{h})$ of φ, the

singleton $\{\mathfrak{s}(x)\}$ is a set of roots for $\rightarrow_{\mathfrak{h}}$ (i.e., all locations in $loc(\mathfrak{h})$ must be accessible from $\mathfrak{s}(x)$). Disjoint structures built in parallel (such as two lists with the same length) are not allowed[3]. Second, these structures also admit a bounded number of "dangling pointers" (i.e., elements of $dgl(\mathfrak{h})$), which again correspond (by $\mathfrak{s}$) to variables occurring in φ, since all the variables introduced by unfolding rules must be allocated due to the establishment property. The latter property turned out to be essential for decidability [6]. This yields the definition of a property called *PCE-compatibility*:

Definition 7 (PCE-compatibility). *Let $k \in \mathbb{N}$. A structure $(\mathfrak{s}, \mathfrak{h})$ is* k-PCE-compatible *if (i) $\mathrm{card}(dgl(\mathfrak{h})) \leq k$ and (ii) there exists a set of roots R for $\rightarrow_{\mathfrak{h}}$ with $\mathrm{card}(R) \leq k$. A pair $(\varphi, \mathcal{R})$ is* k-PCE-compatible *if every $\mathcal{R}$-model of φ is k-PCE-compatible.*

Proposition 1. *Let φ be a formula, and $\mathcal{R}$ be a PCE SID. Every $\mathcal{R}$-model $(\mathfrak{s}, \mathfrak{h})$ of φ is k-PCE-compatible, where k is the number of (free or bound) variables in φ.*

Example 1. Let us consider the formula $\varphi = p(x, y)$ and the SID $\mathcal{R}_1$ below. For readability, we employ the same variable names in predicate definitions and predicate calls to avoid introducing the renaming of variables:

$$\begin{array}{ll} p(x,y) \Leftarrow \exists z.\, z \rightarrow (x,y)\,, & q(y) \Leftarrow \exists z,u,t.\, (y \rightarrow (z,t) \star r(z,u,t))\,, \\ p(x,y) \Leftarrow x \rightarrow (y) \star q(y)\,, & r(z,u,t) \Leftarrow u \not\approx t \star z \rightarrow (u) \star t \rightarrow (t)\,. \end{array} \tag{7}$$

The SID $\mathcal{R}_1$, and thus $(\varphi, \mathcal{R}_1)$, are not PCE. In the first rule for p, z is root but not a free variable, the rule defining q is not established for the existential variable u and the rule defining r does not respect the progress condition as it has two points-to atoms.

4 Overview of Our Procedure

The (nonterminating) algorithm for transforming a pair $(\Phi, \mathcal{R})$ into an equivalent PCE pair is divided into four main steps (from now on, we denote the target formula by Φ, whereas the meta-variable φ is reserved for formulas occurring in inductive rules).

Step 1: We compute abstractions of the models of Φ (and of all relevant predicate atoms). The aim is to extract relevant information about the constraints satisfied by these models concerning (dis)equalities, heap reachability and allocated locations. The abstractions are constructed over a set of variables that includes the variables freely occurring in the formulas, together with some additional variables — the so-called *invisible variables* — that correspond to existential variables that either occur in Φ or are introduced by unfolding inductive rules. The usefulness of invisible variables will be demonstrated later. The computation does not terminate in general, as the set of abstractions is infinite (due to the presence of invisible variables). However, we prove that the computation terminates exactly when the considered formula is k-PCE-compatible (for some $k \in \mathbb{N}$). Furthermore, we introduce a technique — the so-called *ISIV* condition — to detect when the formula is not k-PCE-compatible during the computation of

[3] Indeed, to satisfy the connectivity condition the two lists must be defined in distinct atoms (as they are not connected). But then it is impossible to ensure that they have the same number of elements.

the abstractions. This ensures termination in all cases and also proves that the problem of deciding whether a given pair is k-PCE-compatible, for some k, is decidable. This step is detailed in Sect. 5.

Step 2: We transform the set of rules in order to ensure that every predicate is associated with a unique abstraction, in which all invisible variables are replaced by visible ones. This step always terminates. It adds some combinatorial explosion that could be reduced by a smart transformation, but it greatly simplifies the technical developments. This step is detailed in Sect. 6.

Step 3: We apply some transformations on the SID to ensure that every abstraction admits exactly one root. This step may fail in the case where the structures described by the rules do not have this property. See Sect. 7.

Step 4: We recursively transform any rule $p(\vec{x}) \Leftarrow \varphi$ into a PCE rule by decomposing φ into a separating conjunction $y \to (z_1, \ldots, z_k) \star \varphi_1 \star \cdots \star \varphi_k$ where y is the root of the structure and every φ_i encodes a structure of root z_i. Each of these formulas φ_i may then be associated with fresh predicate atoms if needed. The process is repeated until one gets a fixpoint. Equivalence is always preserved, but termination is not guaranteed. This step is detailed in Sect. 8.

Before describing all these steps, we wish to convey some general explanations about the difficulties that arise when one tries to enforce each condition in Def. 5.

The progress condition can often be enforced by introducing additional predicates to ensure that each rule allocates exactly one location. For instance, the definition of lists of odd length in Eq. (5) is not PCE, but it can be transformed into a PCE definition by replacing the inductive rule (at right) with the two inductive rules given in Eq. (6) (introducing a new predicate $\mathtt{ls}^2(x, y)$). The key point is that the root of the structure must be associated with a parameter of the predicate, which sometimes requires the addition of new existential variables in the formula. For instance, the formula $p(x)$ with $p(x) \Leftarrow \exists y.\, y \to (x)$ will be written: $\exists y.\, p'(x, y)$ with $p'(x, y) \Leftarrow y \to (x)$. The set of roots is computed in Step 1 above, and invisible roots (like y in the above example) are made visible during Step 2. Note that this technique is applicable only if the number of such roots is bounded; the ISIV condition will ensure that this constraint is satisfied.

The connectivity condition is enforced by using the abstract reachability relation computed during Step 1 to identify the predicate atoms that do not satisfy this condition and by modifying the rules to delay the call to these predicates until the connectivity condition is satisfied. For instance, the first rule below is modified into the second one:

$$q(x) \Leftarrow \exists y_1, y_2, y_3.\, (x \to (y_1, y_2) \star \mathtt{ls}^+(y_1, y_3) \star \mathtt{ls}^+(y_3, y_3) \star \mathtt{ls}^+(y_2, y_2)), \quad (8)$$

$$q(x) \Leftarrow \exists y_1, y_2, y_3.\, (x \to (y_1, y_2) \star q'(y_1, y_3) \star \mathtt{ls}^+(y_2, y_2)), \quad (9)$$

where $q'(y_1, y_3)$ is defined similarly to $\mathtt{ls}^+(y_1, y_3)$ in Eq. (2) except the first rule:

$$q'(y_1, y_3) \Leftarrow y_1 \to (y_3) \star \mathtt{ls}^+(y_3, y_3), \qquad q'(y_1, y_3) \Leftarrow \exists z.\, (y_1 \to (z) \star q'(z, y_3)). \quad (10)$$

The establishment condition may be enforced in two ways. If the considered existential variable only occurs in pure atoms (disequalities or equalities), then it can be eliminated using usual quantifier elimination techniques. For instance, the predicate $r(x) \Leftarrow \exists y.\, x \to () \star x \not\approx y$ can be reduced into $r(x) \Leftarrow x \to ()$ since a location y distinct

from x always exists (recall that the equational atom $x \not\approx y$ only holds for empty heaps). Otherwise, one must collect the set of all variables that are reachable but not allocated and associate them with new existential variables in φ (and parameters of predicates). For instance, the formula $r'(x)$ with $r'(x) \Leftarrow \exists y.\, x \to (y)$ is transformed into $\exists y.\, r''(x, y)$ with $r''(x, y) \Leftarrow x \to (y)$. These variables correspond to invisible variables computed during Step 1 and transformed into visible variables in Step 2. Again, the ISIV condition ensures that the number of such variables is bounded.

5 Abstracting Models and Formulas

We formalize the notion of abstraction that summarizes the main features (locations defined and allocated, reachability, etc.) of models and SL-formulas. Then, we define two relations between abstractions and SL-structures. Finally, we define the abstraction process for a formula, i.e., how we attach a set of abstractions to an SL-formula.

Definition 8 (Abstraction). *An* abstraction *is a tuple* $A = \langle V, \backsim, \neq, V_v, \overline{V}_a, h, \rightsquigarrow \rangle$ *where: (i)* V *is a set of variables and* $\backsim$ *is an equivalence relation on* V*; (ii)* $\neq$ *(disequality relation) is a symmetric and irreflexive binary relation on* $\overline{V}$*; (iii)* $V_v \subseteq V$ *is a finite set of variables called* visible variables*; (iv)* $\overline{V}_a \subseteq \overline{V}$ *is a subset of classes of variables called* allocated variables*; (v)* $h : \overline{V}_a \longrightarrow \overline{V}^*$ *is a partial* heap mapping *which associates a tuple of classes of variables of arbitrary size to some class of allocated variables; (vi)* $\rightsquigarrow \subseteq \overline{V} \times \overline{V}$ *is a* reachability relation *which is a relation such that* $\forall [x] \in \overline{V}_a$ *and* $\forall [y] \in h([x])$, $([x], [y]) \in \rightsquigarrow$*. The set of all abstractions is denoted by* $\mathcal{A}$*. We designate the components of an abstraction* A *using the dotted notation by* $A\!\bullet\!V$*,* $A\!\bullet\!V_v$*, etc. The set of* invisible variables *of* A *is* $A\!\bullet\!V_{inv} \triangleq A\!\bullet\!V \setminus A\!\bullet\!V_v$.

Abstractions are taken modulo renaming of invisible variables: two abstractions, A_1 and A_2, are considered equal, denoted $A_1 = A_2$, if there exists a renaming σ of invisible variables such that $A_1 = A_2\sigma$.

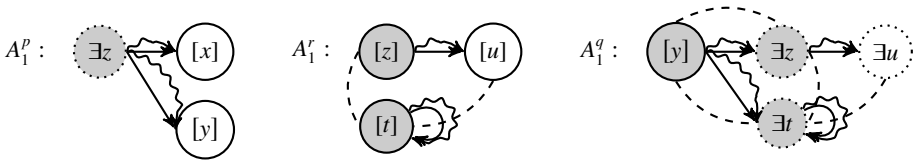

Fig. 1. Examples of abstractions.

Example 2. Fig. 1 graphically represents three abstractions denoted A_1^p, A_1^r and A_1^q. Equivalence classes are represented by circles and are labelled by variable names. Allocated classes are filled grey; invisible variables are prefixed with $\exists$, and [ ] are omitted. Disequalities are represented with dashed lines, while heap and reachability relations are represented with tick resp. snaked arrows.

An SL-structure is a model of an abstraction if its store is *coherent* with the abstraction (i.e., it maps equal variables to the same location and disequal variables to different locations) and its heap contains at least all the reachability relations of the abstraction. However, the model may contain more allocated locations and paths between locations. On the other hand, an abstraction of an SL-structure captures *exactly* the visibility of variables in the store, the equivalence between variables and the reachability of locations in the heap; it abstracts the paths between locations labelled by (visible or invisible) variables and going through locations not labelled by some variable.

Definition 9 (Model and Abstraction). *A structure* $(\mathfrak{s}, \mathfrak{h})$ *is a* model *of an abstraction* A*, denoted by* $(\mathfrak{s}, \mathfrak{h}) \models A$*, if there exists a functional extension* $\dot{\mathfrak{s}}$ *of* $\mathfrak{s}$ *satisfying the following conditions: (i)* $\mathrm{dom}(\dot{\mathfrak{s}}) = A{\bullet}V$ *and* $\mathrm{dom}(\mathfrak{s}) = A{\bullet}V_v$*; (ii) If* $(x, y) \in A{\bullet}\backsim$ *then* $\dot{\mathfrak{s}}(x) = \dot{\mathfrak{s}}(y)$*; (iii) If* $([x], [y]) \in A{\bullet}\not\approx$ *then* $\dot{\mathfrak{s}}(x) \neq \dot{\mathfrak{s}}(y)$*; (iv) For all* $x \in A{\bullet}V$*, if* $[x] \in A{\bullet}\overline{V}_a$ *then* $\dot{\mathfrak{s}}(x) \in \mathrm{dom}(\mathfrak{h})$*; (v) For all* $[x] \in A{\bullet}\overline{V}_a$ *if* $A{\bullet}h([x]) = ([y_1], \ldots, [y_k])$ *then* $\mathfrak{h}(\dot{\mathfrak{s}}(x)) = (\dot{\mathfrak{s}}(y_1), \ldots, \dot{\mathfrak{s}}(y_k))$*; (vi) For all* $x, y \in V$*, if* $([x], [y]) \in A{\bullet}\rightsquigarrow$ *then there exists a path* $\ell_0 \to_{\mathfrak{h}} \cdots \to_{\mathfrak{h}} \ell_n$ *in* $\mathfrak{h}$ *such that* $\ell_0 = \dot{\mathfrak{s}}(x)$*,* $\ell_n = \dot{\mathfrak{s}}(y)$ *and* $\{\ell_1, \ldots, \ell_{n-1}\} \cap \mathrm{img}(\dot{\mathfrak{s}}) = \emptyset$*. If* $(\mathfrak{s}, \mathfrak{h}) \models A$ *and the converses of Items (ii), (iii) and (vi) hold, then* A is an abstraction of $(\mathfrak{s}, \mathfrak{h})$*. The set of all abstractions of* $(\mathfrak{s}, \mathfrak{h})$ *is denoted by* $\mathfrak{abs}(\mathfrak{s}, \mathfrak{h})$.

Example 3. Consider the structure $(\mathfrak{s}_1, \mathfrak{h}_1)$ defined over the set of variables $\{x, y\}$ with $\mathfrak{s}_1(x) = \ell_1$, $\mathfrak{s}_1(y) = \ell_2 \neq \ell_1$, $\mathfrak{h}_1(\ell_0) = (\ell_1, \ell_2)$. A_1^p from Fig. 1 is an abstraction of $(\mathfrak{s}_1, \mathfrak{h}_1)$ for $\dot{\mathfrak{s}}_1(z) = \ell_0$. Moreover, A_1^p has as model $(\mathfrak{s}_2, \mathfrak{h}_2)$ with $\mathfrak{s}_1(x) = \mathfrak{s}_1(y) = \ell_1$, $\mathfrak{h}_1(\ell_0) = (\ell_1, \ell_1)$.

The following operations on abstractions are used in our abstraction process.

Definition 10 (Pure abstractions). *The* empty abstraction*, denoted* $A_{\mathtt{emp}}$*, has all its components empty sets. Let* V_0 *be a set of variables. The* abstraction of equalities *over* V_0*, denoted* $A_{\approx}(V_0)$*, is* $\langle V_0, \{V_0\}, \emptyset, V_0, \emptyset, \emptyset, \emptyset\rangle$*, i.e., all variables are visible and in the same equivalence class. The* abstraction of disequalities *over* V_0 *is* $A_{\not\approx}(V_0) = \langle V_0, \mathtt{Id}_{V_0}, V_0^2 \setminus \mathtt{Id}_{V_0}, V_0, \emptyset, \emptyset, \emptyset\rangle$*, i.e., all variables are visible and pairwise distinct, and none is allocated.*

Note that we identify equivalence relations with the set of their equivalence classes so that $\{V_0\}$ denotes the relation $\{(x, y) \mid x, y \in V_0\}$.

Definition 11 (Quantified abstractions). *Let* $V_0 \subseteq A{\bullet}V$ *be a set of variables. The* hiding of V_0 in A*, denoted by* $A_{\exists(V_0)}$*, is the abstraction having the same components as* A *except the set of visible variables, i.e.,* $A_{\exists(V_0)}{\bullet}V_v = A{\bullet}V_v \setminus V_0$.

Definition 12 (Separated abstractions). *Let* A_1 *and* A_2 *be two abstractions; w.l.o.g., we consider that* $A_1{\bullet}V_{inv} \cap A_2{\bullet}V_{inv} = \emptyset$*, i.e., the sets of invisible variables are disjoint (modulo renaming). Let* $V^\star = A_1{\bullet}V \cup A_2{\bullet}V$ *and the equivalence relation* $\backsim_\star$ *over* $V^\star$ *defined by the transitive closure of* $A_1{\bullet}\backsim \cup A_2{\bullet}\backsim$*. Consider now the relation* $\not\approx_\star$ *over* $\overline{V^\star}_{\backsim_\star}$ *(the set of equivalence classes of* $\backsim_\star$*) defined by the symmetric closure of the relation:* $\{([x]_{\backsim_\star}, [y]_{\backsim_\star}) \mid x, y \in V^\star, ([x]_{A_i{\bullet}\backsim}, [y]_{A_i{\bullet}\backsim}) \in A_i{\bullet}\not\approx, i \in \{1, 2\}\} \cup \{([x_1]_{\backsim_\star}, [x_2]_{\backsim_\star}) \mid x_i \in V^\star, [x_i]_{A_i{\bullet}\backsim} \in A_i{\bullet}\overline{V}_a, i \in \{1, 2\}\}$*. If* $\not\approx_\star$ *is irreflexive, then* A_1 *and* A_2 *are* separated.

Definition 13 (Separating abstractions). *The* separating composition $A_1 \star A_2$ *of two separated abstractions* A_1 *and* A_2 *is the abstraction* $A_\star$ *such that:*

- $A_\star{\bullet}V = V^\star$; $A_\star{\bullet}{\backsim} = {\backsim_\star}$; $A_\star{\bullet}{\not\approx} = {\not\approx_\star}$;
- $A_\star{\bullet}V_v = A_1{\bullet}V_v \cup A_2{\bullet}V_v$;
- $A_\star{\bullet}\overline{V}_a = \{[x]_{A_\star{\bullet}\backsim} \mid [x]_{A_i{\bullet}\backsim} \in A_i{\bullet}\overline{V}_a, i \in \{1,2\}\}$;
- $A_\star{\bullet}h = [[x]_{A_\star{\bullet}\backsim} \mapsto ([y_1]_{A_\star{\bullet}\backsim}, \ldots, [y_n]_{A_\star{\bullet}\backsim}) \mid A_i{\bullet}h([x]_{A_i{\bullet}\backsim}) = ([y_1]_{A_i{\bullet}\backsim}, \ldots, [y_n]_{A_i{\bullet}\backsim}),$ $i \in \{1,2\}]$;
- $A_\star{\bullet}{\rightsquigarrow} = \{([x]_{A_\star{\bullet}\backsim}, [y]_{A_\star{\bullet}\backsim}) \mid ([x]_{A_i{\bullet}\backsim}, [y]_{A_i{\bullet}\backsim}) \in A_i{\bullet}{\rightsquigarrow}, i \in \{1,2\}\}$.

The following definitions are used to build the reachability relation in abstractions by replacing chains $[x_0] \mapsto [x_1] \mapsto \ldots \mapsto [x_{n-1}] \mapsto [x_n]$ related by $A{\bullet}h$ with the tuple $([x_0], [x_n])$ in $A{\bullet}{\rightsquigarrow}$ if the variables x_i with $i \in [1, n-1]$ are not "special" for A.

Definition 14 (Roots). *The* roots of an abstraction A, $\mathrm{root}(A)$*, is the set of minimal sets of roots of* $A{\bullet}{\rightsquigarrow}$*. We denote by* $x \in_\forall \mathrm{root}(A)$ *or* $[x] \in_\forall \mathrm{root}(A)$ *that* $[x]$ *belongs to all sets in* $\mathrm{root}(A)$ *and by* $x \in_\exists \mathrm{root}(A)$ *or* $[x] \in_\exists \mathrm{root}(A)$ *that* $[x]$ *belongs to at least one set in* $\mathrm{root}(A)$.

As $A{\bullet}{\rightsquigarrow}$ may contain cycles, roots are not uniquely defined. However, the algorithm for computing abstractions will ensure that $\mathrm{root}(A)$ is always non-empty.

Definition 15 (Special and persistent variables). *A variable* $x \in A{\bullet}V_{inv}$ *is* special *if its equivalence class is a singleton and it satisfies one of the following conditions: (i)* $x \in_\forall \mathrm{root}(A)$*, i.e., x occurs in all sets of roots of A; (ii)* $[x] \notin A{\bullet}\overline{V}_a$*, i.e., x is not allocated, and there exists* $[y] \in A{\bullet}\overline{V}_a$ *such that* $([y], [x]) \in A{\bullet}{\rightsquigarrow}$*, i.e., x is reachable from an allocated variable; (iii) there exists* $y \in A{\bullet}V_v$ *such that* $y \in_\exists \mathrm{root}(A)$ *and* $[x] \in A{\bullet}h([y])$*, i.e., x is pointed to by a possible root that is visible; (iv) there exists* $[y] \in A{\bullet}\overline{V}_a$ *such that* $[y] \in_\forall \mathrm{root}(A)$ *and* $[x] \in A{\bullet}h([y])$*, i.e., x is pointed to by a necessary root that is visible or invisible. An invisible variable is* persistent *if it satisfies one of the items (i) or (ii) above. The set of persistent variables is denoted by* $A{\bullet}V_{per}$.

Example 4. Abstractions A_1^p and A_1^q in Fig. 1 have a singleton set of roots built from one class: $\mathrm{root}(A_1^p) = \{\{[z]\}\}$ and $\mathrm{root}(A_1^q) = \{\{[y]\}\}$, while A_1^r has a unique set of roots but containing two classes $\mathrm{root}(A_1^r) = \{\{[z], [t]\}\}$. The variable z is not visible in A_1^p, but it is special and persistent since it fulfils the condition (i) of Def. 15. All the variables in A_1^q are special, but only y and u are persistent.

Definition 16 (Disconnected variable). *A variable* $x \in A{\bullet}V_v$ *is disconnected if it satisfies the following two conditions: (1)* $[x] \notin A{\bullet}\overline{V}_a$*, i.e., x is not allocated; and (2) for all* $[y] \in A{\bullet}\overline{V}_a$, $([y], [x]) \notin A{\bullet}{\rightsquigarrow}$*, i.e., x is not pointed by an allocated variable.*

If a variable is disconnected, any variable in its equivalence class is also disconnected. Moreover, a disconnected variable cannot be special. For any equivalence relation $\bowtie$, we denote by $\bowtie \setminus x$ the restriction of $\bowtie$ to the elements distinct from x. Similarly, $\overline{S} \setminus x$ denotes the set $\{[y] \mid y \in S, y \neq x\}$, and for any relation $\rightarrow$ on equivalence classes of $\bowtie$, $\rightarrow \setminus x$ is the corresponding relation on equivalence classes of $\bowtie \setminus x$.

Definition 17 (Deletion of variables not special). *Let A be an abstraction and $x \in A.V_{inv}$ a variable that is not special. We define* $\mathrm{rem}(A, x)$*, the abstraction obtained by deleting x from A as follows:* $A_{\mathrm{rem}} = \langle A.V \setminus \{x\}, A.\smile \setminus x, A.\neq \setminus x, A.V_v, A.\overline{V}_a \setminus x, A.h \setminus x, \rightsquigarrow' \setminus x\rangle$ *with* $\rightsquigarrow' = \{([y], [z]) \mid [y], [z] \in A.\overline{V} \wedge ([y], [x]) \in A.\rightsquigarrow \wedge ([x], [z]) \in A.\rightsquigarrow\} \cup A.\rightsquigarrow$*. We denote by* $\mathrm{rem}(A)$ *the abstraction obtained by removing all variables not special in A.*

Definition 18 (Set of abstractions of a symbolic heap). *Let φ be a symbolic heap formula of SL. The set of abstractions of a formula φ, denoted* $\mathrm{abs}(\varphi)$*, is inductively constructed using the rules in Tab. 1.*

Example 5. Consider the pair $(\varphi, \mathcal{R})$ introduced by Ex. 1. The abstractions of φ are built by firstly building the abstractions of the predicates $r(z, u, t)$ and then $q(y)$ — that calls r — defined by the rules in Eq. (7). Then $\varphi = p(x, y)$ has two abstractions. The first is A_1^p from Fig. 1, obtained from the non-recursive rule of p. The second is A_2^p in Fig. 2, obtained from A_2 by removing variables z and t using the procedure in Def. 17 because they are not special. The abstraction A_2 is obtained by applying the rule [Sep] on A_1^q in Fig. 1, which is an abstraction of $q(y)$, and the abstraction obtained by the rule [Pto] for $x \to (y)$.

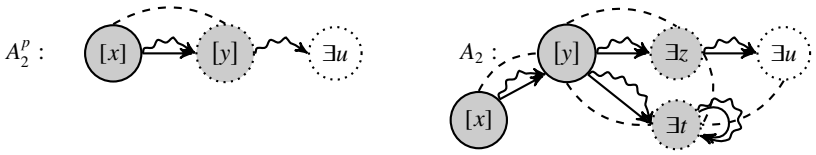

Fig. 2. Abstraction A_2

Given $A \in \mathrm{abs}(\varphi)$, we consider the implicit tree of construction of A using rules in Def. 18: every node of this tree is an abstraction created by one of the rules [Ex], [Pred] and [Sep], and every leaf is an abstraction of an atomic formula. Therefore, every node of this tree is associated with a formula, which is a sub-formula of an unfolding of φ.

Definition 19 (Condition "Infinite Set of Invisible Variables" (ISIV)). *The abstraction $A \in \mathrm{abs}(p(x_1, \ldots, x_n))$ satisfies the condition ISIV if there exists an abstraction A' in the construction tree of A such that:*

1. *A' is associated with a renaming $p(y_1, \ldots, y_n)$ of $p(x_1, \ldots, x_n)$;*
2. *A has strictly more persistent variables than A':* $\mathrm{card}(A'.V_{per}) < \mathrm{card}(A.V_{per})$*;*
3. *the projections of abstractions A and A' on their visible variables are equal (modulo a renaming of the arguments $x_i \leftarrow y_i$).*

Intuitively, the condition asserts that a "loop" exists in the unfolding tree of p, where persistent variables are introduced inside the loop. As one can go through the loop an arbitrary number of times, this entails that some branch exists with an unbounded

Table 1. Computing Abstractions of a Symbolic Heap Formula

$$\textsc{Emp}\ \frac{}{\mathrm{abs}(\mathtt{emp}) \ni A_{\mathtt{emp}}} \qquad \textsc{Eq}\ \frac{}{\mathrm{abs}(x \approx y) \ni A_{\approx}(\{x, y\})} \qquad \textsc{NEq}\ \frac{}{\mathrm{abs}(x \not\approx y) \ni A_{\not\approx}(\{x, y\})}$$

$$\textsc{Pto}\ \frac{\begin{array}{c} A.V = A.V_v = \{x, y_1, \ldots, y_n\} \qquad A.{\backsim} = \mathtt{Id} \qquad A.{\not\approx} = \emptyset \\ A.\overline{V}_a = \{[x]\} \qquad A.h = [[x] \mapsto ([y_1], \ldots, [y_n])] \qquad A.{\rightsquigarrow} = \{([x], [y_i]) \mid i \in [\![1, n]\!]\} \end{array}}{\mathrm{abs}(x \to (y_1, \ldots, y_n)) \ni A}$$

$$\textsc{Sep}\ \frac{\mathrm{abs}(\psi_1) \ni A_1 \qquad \mathrm{abs}(\psi_2) \ni A_2 \qquad A_1, A_2 \text{ are separated} \qquad A = \mathrm{rem}(A_1 \star A_2)}{\mathrm{abs}(\psi_1 \star \psi_2) \ni A}$$

$$\textsc{Ex}\ \frac{\mathrm{abs}(\psi) \ni A' \qquad A = \mathrm{rem}(A'_{\exists([x])})}{\mathrm{abs}(\exists x.\, \psi) \ni A}$$

$$\textsc{Pred}\ \frac{\mathrm{abs}(\exists y_1, \ldots, y_n.\, (y_1 \approx x_1 \star \cdots \star y_n \approx x_n \star \psi)) \ni A \qquad p(y_1, \ldots, y_n) \Leftarrow \psi \in \mathcal{R}}{\mathrm{abs}(p(x_1, \ldots, x_n)) \ni A}$$

number of persistent variables, which in turn entails that non-k-PCE-compatible models exist. If this condition is satisfied by one abstraction built during this step, the algorithm fails. The following theorem states that the algorithm is correct and complete (proof in App. B):

Theorem 2. *Let φ be a formula and let $\mathcal{R}$ be an SID. We suppose that the construction of abstractions terminates without failing. If $A \in \mathrm{abs}(\varphi)$, then there exists a model $(\mathfrak{s}, \mathfrak{h})$ of φ such that A is an abstraction of $(\mathfrak{s}, \mathfrak{h})$. Moreover, if φ admits a model $(\mathfrak{s}, \mathfrak{h})$, then there exists an abstraction A of φ such that $(\mathfrak{s}, \mathfrak{h}) \models A$.*

We also show (see App. C) that the algorithm terminates, provided the ISIV condition is used to dismiss pairs $(\varphi, \mathcal{R})$ that are not k-PCE-compatible (thus that cannot admit any equivalent PCE pair, by Prop. 1):

Theorem 3. *Let φ be a formula and let $\mathcal{R}$ be an SID. If there exists $k \in \mathbb{N}$ such that $(\varphi, \mathcal{R})$ is k-PCE-compatible, then the computation of* $\mathrm{abs}(\varphi)$ *terminates without failure (hence the ISIV condition is never fulfilled). Otherwise, the ISIV condition eventually applies during the computation of* $\mathrm{abs}(\varphi)$*. Consequently, the problem of testing whether $(\varphi, \mathcal{R})$ is k-*PCE-compatible *for some $k \in \mathbb{N}$ is decidable.*

6 Predicates with Exactly One Abstraction

We describe an algorithm reducing any pair $(\Phi, \mathcal{R})$ into an equivalent pair $(\Phi^\dagger, \mathcal{R}^\dagger)$ such that every predicate atom admits exactly one abstraction with no invisible variables. We also get rid of some existential variables when possible. The eventual goal is to ensure that the rules that were obtained are established (in the sense of Def. 5). We need

to introduce some definitions and notations. A *disconnected set* for an n-ary predicate p and an abstraction $A \in \mathrm{abs}(p(x_1, \ldots, x_n))$ is any subset I of $\{1, \ldots, n\}$ such that all variables x_i for $i \in I$ are disconnected in A. Let $\mathcal{R}$ be an SID. Let $x_1, \ldots, x_n, \ldots$ be an infinite sequence of pairwise distinct variables, which will be used to denote the formal parameters of the predicates. For each n-ary predicate p occurring in $\mathcal{R}$, for each abstraction $A \in \mathrm{abs}(p(x_1, \ldots, x_n))$ and for all disconnected sets I for p, A, we introduce a fresh predicate p^A_I, of arity $n + m - \mathrm{card}(I)$, where $m = \mathrm{card}(A{\bullet}V_{inv})$. Intuitively, p^A_I will denote some "projection" of the structures corresponding to the abstraction A. The additional arguments will denote the invisible variables. The removed arguments correspond to disconnected variables.

Example 6. The predicate p, defined by rules on the left in Ex. 1, has two abstractions (one by rule), A^p_1 and A^p_2, where all roots are connected. In the same example, predicates q and r also have only one abstraction. For all these predicates, the sets I are always $\emptyset$.

The rules associated with p^A_I are obtained from those associated with p as follows. For every formula φ such that $p(x_1, \ldots, x_n) \Leftarrow_{\mathcal{R}} \varphi$, where φ is of the form $\exists \vec{y}.\, (q_1(\vec{u}_1) \star \cdots \star q_k(\vec{u}_k) \star \varphi')$ and φ' contains no predicate symbol, and for all abstractions $A_i \in \mathrm{abs}(q_i(x_1, \ldots, x_{\#(q_i)}))$ (for $i \in [\![1, k]\!]$), we add the rule:

$$p^A_I(\vec{s}, x'_1, \ldots, x'_m) \Leftarrow \exists \vec{z}.\, (q_1{}^{A_1}_{J_1}(\vec{t}_1, \vec{v}_1) \star \cdots \star q_k{}^{A_k}_{J_k}(\vec{t}_k, \vec{v}_k) \star \varphi'')\sigma \qquad (11)$$

if all the following conditions hold:

- A is the abstraction computed from φ as explained in Def. 18, selecting A_i for the abstraction of $q_i(x_1, \ldots, x_{\#(q_i)})$, i.e., $A = \mathrm{rem}(A'_{\exists(\vec{z})})$, where $\{A''\} = \mathrm{abs}(\varphi')$ (since φ' contains no predicate) and $A' = A_1 \star \cdots \star A_k \star A''$ is the abstraction computed from the matrix $(q_1(\vec{u}_1) \star \cdots \star q_k(\vec{u}_k) \star \varphi')$ of φ.
- $\vec{s}$ (resp. $\vec{t}_i$) is the subsequence of $x_1, \ldots, x_n$ (resp. of $\vec{u}_i$) obtained by removing all components of rank $j \in I$ (resp. $j \in J_i$). Intuitively, I and J_i denote the parameters that are removed from the arguments of p and q_i, respectively.
- J_i is a subset of $\{1, \ldots, \#(q_i)\}$, and for all variables z occurring as the j-th component of $\vec{u}_i$, the following equivalence holds: $j \in J_i$ iff $z \in \vec{z} \cup \{x_i \mid i \in I\}$ and z is disconnected in A'. Note that the last condition entails that the j-th component of $\vec{u}_i$ is also disconnected in A_i; hence the predicate $q_i{}^{A_i}_{J_i}$ exists. Intuitively, a variable is removed if it is disconnected, and either it is existentially quantified in the rule, or it is a free variable that was removed from the argument of p.
- $(x'_1, \ldots, x'_m)$ and $\vec{v}_i$ are the sequences of invisible variables in A and A_i, respectively (the order is irrelevant and can be chosen arbitrarily). We assume by renaming that the $A_i{\bullet}V_{inv}$ are pairwise disjoint.
- σ is any substitution with $\mathrm{dom}(\sigma) \subseteq \vec{y}$ and $\mathrm{img}(\sigma) \subseteq \vec{y} \cup \vec{s}$ such that for all $y \in \vec{y}$ and for all $y' \in \vec{y} \cup \vec{s}$: $\sigma(y) = \sigma(y') \iff (y, y') \in A'{\bullet}{\backsim}$. Intuitively, σ is applied to get rid of superfluous existential variables by instantiating them when it is possible, i.e., when the variable is known to be equal to a free variable or another existential variable[4].

[4] In the latter case several substitutions exist, one of them can be chosen arbitrarily (the resulting rules are identical up to α-renaming, e.g., $\exists x \exists y(x \approx y \star q(x, y))$ can be written $\exists x(x \approx y \star q(x, y))\{y \leftarrow x\}$ or $\exists y(x \approx y \star q(x, y))\{x \leftarrow y\}$).

– φ'' is obtained from φ' by removing all pure atoms containing a variable that is disconnected in A' and does not occur in $\vec{s}$.
– $\vec{z}$ is the sequence of variables occurring either in the formula φ'' or in the sequences $\vec{t}_i$ or $\vec{v}_i$ (for some $i \in [\![1, k]\!]$) but not in $\{x_1, \ldots, x_n, x'_1, \ldots, x'_m\} \cup \mathrm{dom}(\sigma)$ (again, the order is irrelevant). These variables correspond to variables from $\vec{y}$ or $\vec{v}_i$ that can be eliminated during the computation of A using the rule introduced in Def. 17.

The obtained set of rules is denoted by $\mathcal{R}^\dagger$. It is clear that $\mathcal{R}^\dagger$ is finite (up to α-renaming) if $\mathcal{R}$ is finite and $\mathrm{abs}(p(x_1, \ldots, x_n))$ is finite for all n-ary predicates p in $\mathcal{R}$.

Example 7. The new rules for p, q, and r defined in the SID $\mathcal{R}_1$ in Ex. 1 are given below:

$$\begin{aligned} &p_\emptyset^{A_1^p}(x, y, z) \Leftarrow z \to (x, y)\,, && q_\emptyset^{A_1^q}(y, z, t, u) \Leftarrow y \to (z, t) \star r(z, t, u)\,, \\ &p_\emptyset^{A_2^p}(x, y, u) \Leftarrow \exists z, t.\, (x \to (y) \star && r_\emptyset^{A_1^r}(z, t, u) \Leftarrow u \not\approx t \star z \to (u) \star t \to (t)\,. \\ &\qquad\qquad q_\emptyset^{A_1^q}(y, z, t, u))\,, \end{aligned} \tag{12}$$

The arity of predicates $p_\emptyset^{A_2^p}$ and $q_\emptyset^{A_1^q}$ has been changed to include the invisible but special variable u, and the predicate $p_\emptyset^{A_1^p}$ now does not have an invisible root any more.

Example 8. In this example, we show how disconnected variables may be eliminated. Let p, q be predicates defined by the rules: $p(x, y) \Leftarrow \exists z.\, (x \to (y) \star q(x, z))$, $q(x, y) \Leftarrow x \not\approx y$. $p(x_1, x_2)$ and $q(x_1, x_2)$ both admit one abstraction, A_p and A_q, respectively, defined by:

$$A_p = (\{x_1, x_2\}, \{\{x_1\}, \{x_2\}\}, \emptyset, \{x_1, x_2\}, \{[x_1]\}\{[x_1] \mapsto [x_2]\}, \emptyset)\,, \tag{13}$$

$$A_q = (\{x_1, x_2\}, \{\{x_1\}, \{x_2\}\}, \{([x_1]\,, [x_2])\}, \{x_1, x_2\}, \emptyset, \emptyset, \emptyset)\,. \tag{14}$$

The above transformation produces the rules: $p_\emptyset^{A_p}(x, y) \Leftarrow (x \to (y) \star q_{\{2\}}^{A_q}(x))$ and $q_{\{2\}}^{A_r}(x) \Leftarrow \mathtt{emp}$. The variable z is eliminated, as it is disconnected in the abstraction corresponding to $x \to (y) \star q(x, z)$. This yields the introduction of a predicate $q_{\{2\}}^{A_r}$ in which the second argument of q is dismissed.

The above transformation may be applied to the formulas Φ occurring in pairs $(\Phi, \mathcal{R})$. Since the establishment condition applies only to the variables occurring in the rule and not to the existential variables of Φ, there is no need to eliminate any predicate argument in this case; thus, we may simply take $I = \emptyset$ for the predicates p_I^A such that p appears in Φ. Predicates of the form q_I^B with $I \neq \emptyset$ will never appear at the root level in Φ, but they may appear in the rules of the predicates $p_\emptyset^A$ (in practice, such rules will be computed on demand). More precisely, we denote by $\Phi^\dagger$ the formula obtained from Φ by replacing every atom $p(y_1, \ldots, y_n)$ in Φ by the formula $\bigvee_{A \in \mathrm{abs}(p(x_1, \ldots, x_n))} \exists \vec{y}_A.\, p_\emptyset^A(y_1, \ldots, y_n, \vec{y}_A)$, where $\vec{y}_A$ is the sequence of variables in $A{\bullet}V_{inv}$ (with arbitrary order). Note that in the case where $\mathrm{abs}(p(x_1, \ldots, x_n)) = \emptyset$, $p(y_1, \ldots, y_n)$ is replaced by an empty disjunction, i.e., by false. The properties of this transformation are stated by the following result (proof in App. D):

Theorem 4. *$(\Phi, \mathcal{R}) \equiv (\Phi^\dagger, \mathcal{R}^\dagger)$. Moreover, for all predicates p_I^A defined in $\mathcal{R}^\dagger$, the set $\mathrm{abs}(p_I^A(\vec{y}, x'_1, \ldots, x'_m))$ contains exactly one abstraction.*

7 Abstractions with Exactly One Root

We introduce an algorithm that transforms the considered SID by introducing and removing predicates such that the abstraction of each predicate p defined by the new $\mathcal{R}$ has only one root. This transformation is done in two steps: first, change predicates with an abstraction without roots, and then change predicates with an abstraction with more than one root. The transformation may fail if the structures corresponding to a given recursive predicate have multiple roots, as such structures cannot be defined by PCE rules (e.g., two parallel lists of the same length).

Removal of Abstractions Without Root: Let us consider every predicate p such that its abstraction $A_p \in \mathrm{abs}(p(\vec{x}))$ satisfies $\mathrm{root}(A_p) = \emptyset$. Because the abstraction of p has no root, the associated structure has no allocated locations, and the predicate can only be unfolded into formulas that do not contain points-to. Thus, for each unfolding of p of abstraction A, which cannot be unfolded any more, it only contains equalities and disequalities that are abstracted in A by $A_{\bullet\sim}$ and $A_{\bullet\not\approx}$. As a consequence, we can create a formula $\varphi_A = (\star_{i,j\in I_\approx} a_i \approx a_j) \star (\star_{i,j\in I_\not\approx} b_i \not\approx b_j)$ with $\{a_i \approx a_j \mid i, j \in I_\approx\} = A_{\bullet\sim}$ and $\{b_i \not\approx b_j \mid i, j \in I_{\not\approx}\} = A_{\bullet\not\approx}$. We can then replace every occurrence of p with φ_A.

Removal of Abstractions With Several Roots: We suppose now that for all predicates p, the abstraction $A_p \in \mathrm{abs}(p(\vec{x}))$ verifies $\mathrm{root}(A_p) \neq \emptyset$. Now let us consider every predicate p such that its abstraction $A_p \in \mathrm{abs}(p(\vec{x}))$ has at least two roots, i.e., for all $R \in \mathrm{root}(A_p)$, $\mathrm{card}(R) \geq 2$. If p does not call itself, we unfold p by replacing each occurrence of p with its definition using the rules in SID. Otherwise, the transformation is considered impossible, and it fails.

At this point, if the transformation does not fail, we obtain:

Proposition 2 (Every abstraction has a single root). *After applying the transformation in this section, for all predicates p, for all abstractions $A \in \mathrm{abs}(p(\vec{x}))$, there exists a set $R \in \mathrm{root}(A)$ such that* $\mathrm{card}(R) = 1$.

Remark 1. We wish to emphasize that the failure of the above operation does not imply that the transformation is unfeasible. For instance, one could, in principle, define two lists of arbitrary (possibly distinct) lengths using one single inductive predicate, adding elements in one of the lists in a non-deterministic way, although such a definition is very unlikely to occur in practice. Then, our algorithm would fail (as it will detect that the structure has two roots), although a PCE presentation exists. Extending the algorithm to cover such cases is part of future work.

8 Transformation into PCE Rules

The last step of the transformation is a procedure reducing any pair $(\Phi^\dagger, \mathcal{R}^\dagger)$ into an equivalent pair $(\Phi^\ddagger, \mathcal{R}^\ddagger)$ such that $\Phi^\ddagger$ and $\mathcal{R}^\ddagger$ are PCE formula resp. SID.

To this aim, we first introduce so-called *derived predicates* (adapted and extended from [4]), the rules of which can be computed from the rules defining predicate symbols. The aim is to extract from the call tree of a spatial atom the part that corresponds to another atom. Given a SID $\mathcal{R}$ and two spatial atoms γ and λ, we denote by $\gamma \multimap \lambda$ the

atom defined by the following rules:

$$\begin{aligned}
&\gamma \mathrel{-\!\!\bullet} \lambda \Leftarrow \exists \vec{x}.\, (\varphi \star (\gamma \mathrel{-\!\!\bullet} \lambda'))\,, && \text{for all } \varphi, \lambda' \text{ with } \lambda \Leftarrow_{\mathcal{R}} \exists \vec{x}.\, (\varphi \star \lambda') \quad \text{(up to AC of } \star\text{)},\\
&\gamma \mathrel{-\!\!\bullet} \lambda \Leftarrow x_1 \approx y_1 \star \cdots \star x_n \approx y_n\,, && \text{if } \gamma = p(x_1, \ldots, x_n) \text{ and } \lambda = p(y_1, \ldots, y_n), \text{ or}\\
& && \gamma = x_1 \to (x_2, \ldots, x_n) \text{ and } \lambda = y_1 \to (y_2, \ldots, y_n).
\end{aligned} \tag{15}$$

We assume that all such rules occur in $\mathcal{R}$. Intuitively, $\gamma \mathrel{-\!\!\bullet} \lambda$ encodes a structure defined as the atom λ but in which a call to γ is removed. It is easy to see that $\gamma \mathrel{-\!\!\bullet} \lambda$ is unsatisfiable if λ is a points-to atom and γ is a predicate atom. By definition, $(x_1 \to (x_2, \ldots, x_n)) \mathrel{-\!\!\bullet} (y_1 \to (y_2, \ldots, y_m))$ is equivalent to $x_1 \approx y_1 \star \cdots \star x_n \approx y_n$ if $m = n$ and unsatisfiable otherwise. These remarks can be used to simplify the rules above (e.g., by removing rules with unsatisfiable bodies).

For instance, given the rules $p(x) \Leftarrow \exists y.\, (x \to (y) \star p(y))$ and $p(x) \Leftarrow x \to ()$, the derived atoms $p(x') \mathrel{-\!\!\bullet} p(x)$ and $(x' \to ()) \mathrel{-\!\!\bullet} p(x)$ both denote a list segment from x to x', whereas $(x' \to (x'')) \mathrel{-\!\!\bullet} p(x)$ denotes a list with a "hole" at x'. The corresponding rules are, after simplification:

$$p(x') \mathrel{-\!\!\bullet} p(x) \Leftarrow \exists y.\, (x \to (y) \star (p(x') \mathrel{-\!\!\bullet} p(y)))\,, \qquad p(x') \mathrel{-\!\!\bullet} p(x) \Leftarrow x \approx x'\,, \tag{16}$$

$$x' \to () \mathrel{-\!\!\bullet} p(x) \Leftarrow \exists y.\, (x \to (y) \star (x' \to () \mathrel{-\!\!\bullet} p(y)))\,, \qquad x' \to () \mathrel{-\!\!\bullet} p(x) \Leftarrow x \approx x'\,, \tag{17}$$

$$(x' \to (x'')) \mathrel{-\!\!\bullet} p(x) \Leftarrow \exists y.\, (x \to (y) \star (x' \to (x'') \mathrel{-\!\!\bullet} p(y)))\,, \tag{18}$$

$$(x' \to (x'')) \mathrel{-\!\!\bullet} p(x) \Leftarrow x \approx x' \star p(x'')\,. \tag{19}$$

The operator $\mathrel{-\!\!\bullet}$ can be nested, for instance $(x_1 \to (x'_1)) \mathrel{-\!\!\bullet} (p(x_2) \mathrel{-\!\!\bullet} p(x))$ denotes a list segment from x to x_2 with a hole at x_1.

Consider a rule $\rho = p(x_1, \ldots, x_n) \Leftarrow \varphi$, where φ' denotes the quantifier-free formula such that $\varphi = \exists \vec{z}.\, \varphi'$. By Thm. 4, the formulas φ and φ' have unique abstractions A_φ and $A_{\varphi'}$, respectively (in what follows the notations $[x]$ and $\rightsquigarrow$ always refer to abstraction $A_{\varphi'}$). Recall that, at this point, establishment is ensured, and all roots are visible. As φ only has a unique abstraction, there is a unique $k \in [\![1, n]\!]$ such that $[x_k]$ is the root of A_φ and the tuple pointed to by the location associated with x_k contains only locations associated with variables $y_1, \ldots, y_m$ that are visible or special in A_φ, with $A_\varphi.h([x_k]) = ([y_1], \ldots, [y_m])$. To make the rule ρ PCE, it must be rewritten to have the form $p(x_1, \ldots, x_n) \Leftarrow \exists \vec{z}.\, x_k \to (y_1, \ldots, y_m) \star q_1(\vec{w_1}) \star \cdots \star q_l(\vec{w_l}) \star \psi$, where ψ is a pure formula, and the root of each atom $q_i(\vec{w_i})$ is in $\{y_1, \ldots, y_m\}$. There are two cases:

Case 1: Assume that φ contains a points-to atom $x'_k \to (y'_1, \ldots, y'_l)$, with $\left[x'_k\right] = [x_k]$ and $\left[y'_i\right] = [y_i]$ for all $i \in [\![1, l]\!]$. The formula φ' is of the form $x'_k \to (y'_1, \ldots, y'_m) \star \psi \star \psi'$, where ψ contains only points-to and predicate atoms and ψ' is a pure formula. The formula ψ may be decomposed into $\varphi_1 \star \cdots \star \varphi_{l'}$, where each formula φ_i allocates only variables z such that $\left[y_{j_i}\right] \rightsquigarrow^* [z]$, where $y_{j_1}, \ldots, y_{j_{l'}}$ are variables in $\{y_1, \ldots, y_l\}$ such that the $\left[y_{j_i}\right]$ are pairwise distinct. Such a decomposition necessarily exists[5] since $[x_k]$ is the root of $\rightsquigarrow$, and every class reachable from $[x_k]$ must be reachable from one of the $[y_i]$. For $i \in [\![1, l']\!]$, if φ_i is not a predicate atom, then we create a fresh predicate q_i whose arguments are all the variables $\vec{w_i}$ that appear in φ_i, we create the rule $q_i(\vec{w_i}) \Leftarrow \varphi_i$, and we replace in φ the formula φ_i by $q_i(\vec{w_i})$. We get a rule ρ' that is now PCE.

[5] If several decompositions exist, then one of them is chosen arbitrarily.

Case 2: Now assume that φ contains no such points-to atom $x'_k \to (y'_1, \ldots, y'_l)$. We have to extract this points-to from some rule that, when unfolded, creates it and add it to a new rule equivalent to ρ. Because A_φ is unique and because every predicate also has a unique abstraction, only one atom can allocate x_k, and this atom must be a predicate atom (because of case 1). Thus φ' is of the form $q(\vec{w}) \star \varphi''$, where x_k is allocated in every model of $q(\vec{w})$. By the previous construction, the atom $q(\vec{w})$ may be replaced by $x_k \to (y_1, \ldots, y_l) \star (x_k \to (y_1, \ldots, y_l) \mathrel{-\!\!\bullet} q(\vec{w}))$. We get a new rule $\rho' = p(x_1, \ldots, x_n) \Leftarrow \exists \vec{z}.\, x_k \to (y_1, \ldots, y_l) \star (x_k \to (y_1, \ldots, y_l) \mathrel{-\!\!\bullet} q(\vec{w})) \star \varphi''$ which fulfils the previous condition, and we may apply the transformation described in the previous item to ρ'. The new rules associated with $x_k \to (y_1, \ldots, y_l) \mathrel{-\!\!\bullet} p'_1(\vec{x'_1})$ are added to the set of rules.

The above transformations are applied until all rules are PCE. Note that termination is not guaranteed (indeed, not all k-PCE-compatible pairs $(\Phi, \mathcal{R})$ admit an equivalent PCE pair, and the existence of such a pair is undecidable by Thm. 1). To enforce termination in some cases, a form of memoization may be used: the predicates introduced above may be reused if the corresponding formulas are equivalent. As logical equivalence is hard to test (undecidable in general), we only check that the rules associated with both predicates are identical up to a renaming of existential variables and spatial predicates. In practice, termination may be ensured by imposing limitations on the number of rules or predicates. We show (see App. E) that if the transformation terminates, we obtain the desired result.

Theorem 5. *Let $(\Phi^\dagger, \mathcal{R}^\dagger)$ be any pair obtained by applying the transformations in Sects. 6 and 7. If the computation of $(\Phi^\ddagger, \mathcal{R}^\ddagger)$ terminates, then $(\Phi^\dagger, \mathcal{R}^\dagger) \equiv (\Phi^\ddagger, \mathcal{R}^\ddagger)$. Also, the SID $\mathcal{R}^\ddagger$, and thus $\Phi^\ddagger$, are PCE.*

9 Experimental Evaluation and Conclusion

We devised an algorithm to construct PCE rules for a given formula (if possible). The existence of such a presentation is undecidable, but we identify a property called PCE-compatibility, which is decidable and weaker. Our algorithm helps to relax the rigid conditions on the PCE presentations. It is also able to construct PCE rules in some more complex cases by performing deep, global transformations on the rules. We have implemented an initial version of the algorithm in OCaml using the Cyclist [2] framework and applied it to benchmarks taken from this framework and SL-COMP [1]. The program comprises approximately 3000 lines of code. To ensure efficiency, the implemented procedure is somewhat simplified compared to the algorithm described in this paper: in Step 8, we avoid the use of derived predicates and instead employ a fixed-depth unfolding of predicate atoms (the other sections strictly adhere to the theoretical definitions). All tests are performed with a timeout of 30 seconds. The running time is low in most examples. In the 145 tested examples, 105 are successfully transformed into equivalent PCE-formulas, 20 trigger the ISIV condition (the structures are not k-PCE-compatible), 3 examples fail at Step 7 (recursive structures with multiple roots) and 17 other timeout. The program and input data are available at https://hal.science/hal-04549937. App. G provides more details about the implementation and experimental results. We find the results highly encouraging, as about

86% of the tested examples are successfully managed. Therefore, this tool may be used to provide a measure of the difficulty of the examples in the SL-COMP benchmark.

We end the paper by identifying some lines of future work. For efficiency, we first plan to refine the transformation by avoiding the systematic reduction to one-abstraction predicates given in Sect. 6. Indeed, this transformation is very convenient from a theoretical point of view but introduces some additional computational blow-up, which could be avoided in some cases. We wish to strengthen the definition of k-PCE-compatible ID in order to capture additional properties of PCE definitions. Notice that the semi-decidability of the PCE problem is an open question. Finally, it could also be interesting to extend the transformation to E-restricted IDs, a fragment of non-established IDs introduced in [4], for which the entailment is decidable.

References

1. SL-COMP website. URL: https://sl-comp.github.io/.
2. James Brotherston, Nikos Gorogiannis, and Rasmus L. Petersen. A Generic Cyclic Theorem Prover. In Ranjit Jhala and Atsushi Igarashi, editors, *Programming Languages and Systems*, volume 7705 of *Lecture Notes in Computer Science*, pages 350–367, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-35182-2_25.
3. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME hard. In Elvira Albert and Laura Kovacs, editors, *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 73 of *EPiC Series in Computing*, pages 191–211. EasyChair, May 2020. ISSN: 2398-7340. URL: https://easychair.org/publications/paper/DdNg, doi:10.29007/f5wh.
4. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Decidable Entailments in Separation Logic with Inductive Definitions: Beyond Establishment. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2021.20, doi:10.4230/LIPIcs.CSL.2021.20.
5. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Unifying Decidable Entailments in Separation Logic with Inductive Definitions. In André Platzer and Geoff Sutcliffe, editors, *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pages 183–199, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-79876-5_11.
6. Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Entailment is Undecidable for Symbolic Heap Separation Logic Formulæ with Non-Established Inductive Rules. *Information Processing Letters*, 173:106169, January 2022. URL: https://www.sciencedirect.com/science/article/pii/S0020019021000843, doi:10.1016/j.ipl.2021.106169.
7. Sheila Greibach. A note on undecidable properties of formal languages. *Math. Systems Theory*, 2(1):1–6, March 1968. doi:10.1007/BF01691341.
8. Sheila A. Greibach. A New Normal-Form Theorem for Context-Free Phrase Structure Grammars. *J. ACM*, 12(1):42–52, January 1965. URL: https://dl.acm.org/doi/10.1145/321250.321254, doi:10.1145/321250.321254.
9. Xincai Gu, Taolue Chen, and Zhilin Wu. A Complete Decision Procedure for Linearly Compositional Separation Logic with Data Constraints. In Nicola Olivetti and Ashish

Tiwari, editors, *Automated Reasoning*, volume 9706 of *Lecture Notes in Computer Science*, pages 532–549, Cham, 2016. Springer International Publishing. doi:10.1007/978-3-319-40229-1_36.

10. Radu Iosif, Adam Rogalewicz, and Jiri Simacek. The Tree Width of Separation Logic with Recursive Definitions. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38. Springer, Berlin, Heidelberg, 2013. ISSN: 1611-3349. URL: https://link.springer.com/chapter/10.1007/978-3-642-38574-2_2, doi:10.1007/978-3-642-38574-2_2.
11. Samin S. Ishtiaq and Peter W. O'Hearn. BI as an assertion language for mutable data structures. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '01, pages 14–26, New York, NY, USA, January 2001. Association for Computing Machinery. doi:10.1145/360204.375719.
12. Christoph Matheja, Jens Pagel, and Florian Zuleger. A Decision Procedure for Guarded Separation Logic Complete Entailment Checking for Separation Logic with Inductive Definitions. *ACM Trans. Comput. Logic*, 24(1):1:1–1:76, January 2023. doi:10.1145/3534927.
13. J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, pages 55–74, Copenhagen, Denmark, July 2002. ISSN: 1043-6871. URL: https://ieeexplore.ieee.org/document/1029817, doi:10.1109/LICS.2002.1029817.

A Proof of Theorem 1 (Undecidability of the PCE Problem)

We recall Thm. 1:

Theorem 1. *The PCE problem is undecidable.*

The proof is by reduction from the problem of testing whether a contex-free language is regular, which is well-known to be undecidable by Greibach's theorem [7].

The proof relies on an encoding of words as list-shaped SL structures. We consider words defined on an alphabet Σ. All symbols $a \in \Sigma$ are associated with pairwise distinct variables, which, to simplify notations, will also be denoted by a. Let $\vec{z}$ be the sequence of variables $a \in \Sigma$ (the order is unimportant).

Definition 20. *Let x, y be two variables not occurring in $\vec{z}$. A structure $(\mathfrak{s}, \mathfrak{h})$* encodes a word $a_1, \ldots, a_n$ *if $\mathfrak{h}$ is of the form $\{(\ell_i, \ell_{i+1}, \mathfrak{s}(a_i)) \mid i \in [\![1, n]\!]\}$ with $\mathfrak{s}(x) = \ell_1$ and $\mathfrak{s}(y) = \ell_{n+1}$. A pair $(\varphi, \mathcal{R})$* encodes a language $L \subseteq \Sigma^*$ *(w.r.t. x and y) if the following equivalence statement holds: $w \in L$ iff there exists a structure m such that $m \models_{\mathcal{R}} \varphi$ and m encodes w.*

Lemma 1. *Let $L \subseteq \Sigma^*$ be a context-free language not containing the empty word ε. There is an algorithm taking as an input a grammar G generating L and returning a pair $(\varphi, \mathcal{R})$ that encodes L. Moreover, if L is regular then $\mathcal{R}$ is PCE.*

Proof. We assume, w.l.o.g., that $G = (\Sigma, N, P, S)$ is in Greibach's normal form [8]. Thus all rules in P are of the form $A \to aB_1 \ldots B_n$ with possibly $n = 0$ (since $\varepsilon \notin w$, no ϵ-rule is needed). We associate all nonterminal symbols $A \in N$ with pairwise distinct predicate of arity $2 + \mathrm{card}(\Sigma)$, also denoted by A. We define a set of rules $\mathcal{R}$ as follows:

1. For each rule $A \to a$ we introduce a rule $A(x, y, \vec{z}) \Leftarrow x \to (y, a)$.
2. For each rule $A \to aB$ we introduce a rule $A(x, y, \vec{z}) \Leftarrow \exists y_1 . (x \to (y_1, a) \star B(y_1, y, \vec{z}))$.
3. For each rule $A \to aB_1 \ldots B_n$ with $n > 1$ we introduce a rule $A(x, y, \vec{z}) \Leftarrow \exists y_1, \ldots, y_n . (x \to (y_1, a) \star B(y_1, y_2, \vec{z}) \star \cdots \star B(y_n, y, \vec{z}))$.

By definition of the rules, it is easy to verify that a structure $(\mathfrak{s}, \mathfrak{h})$ validates $S(x, y, \vec{z})$ if $\mathfrak{h} = \{(\ell_i, \ell'_i, \ell_{i+1}) \mid i \in [\![1, n]\!]\}$, where, for all $i \in [\![1, n]\!]$, there exists a_i such that $\ell'_i = \mathfrak{s}(a_i)$, with $a_1 \ldots a_n \in L$. Consequently, a structure $(\mathfrak{s}, \mathfrak{h})$ validates $S(x, y, \vec{z})$ iff it encodes a word in L, and therefore $(S(x, y, \vec{z}), \mathcal{R})$ encodes L.

If G is regular, then P contains only rules of type 1 or 2 above, and it is easy to check that $\mathcal{R}$ is PCE.

Lemma 2. *Let $L \subseteq \Sigma^*$. If there exist a PC SID $\mathcal{R}$ and a formula φ such that $(\varphi, \mathcal{R})$ encodes L w.r.t. x, y, then L is regular.*

Proof. The proof is by induction on φ. We assume, w.l.o.g., that $\star$ is pushed innermost into the formula (by distributivity of $\star$ over $\vee$).

– We first consider the case where φ is a predicate symbol. We assume, w.l.o.g., that all the variables in $\vec{z}$ are passed as parameters to every predicate symbol, so that all atoms in $\varphi, \mathcal{R}$ are of the form $p(\vec{u}, \vec{z})$. As $\mathcal{R}$ is progressing, every rule in $\mathcal{R}$ must be of the form (assuming w.l.o.g., that the root of the predicate is the first parameter): $A(u, \vec{v}, \vec{z}) \Leftarrow u \rightarrow (\vec{t}) \star \psi$. Moreover, as $(\varphi, \mathcal{R})$ encodes some language L, we may assume that $\vec{t}$ is necessarily of length 2 (as, by Def. 20, all tuples occurring in encodings are of length 2). Moreover (again by definition of the encodings), the second component of every referred tuple must be equal to some location $\mathfrak{s}(a)$ with $a \in \Sigma$, thus we may assume, w.l.o.g., that $\vec{t}$ is of the form (u', a), with $a \in \Sigma$. Still by definition of the encodings, a cannot be allocated, thus ψ contains at most one predicate atom, of root u' (by connectivity).
  We construct a grammar $G = (\Sigma, N, P, S)$ as follows. S is the predicate of φ. All predicates A are associated with pairwise distinct nonterminal symbols also denoted by A, and P contains a rule $A \rightarrow a.B$ for each rule of the above form in $\mathcal{R}$, where B denotes the unique predicate in ψ if such a predicate exists, or ε otherwise. It is easy to check that G is regular and that $L(G) = L$.
– The proof for $\varphi = \texttt{emp}$ is trivial as $(\varphi, \mathcal{R})$ cannot encode L in this case (as $L \neq \{\varepsilon\}$ by hypothesis).
– Assume that $\varphi = \varphi_1 \vee \varphi_2$. Then φ_1 and φ_2 necessarily encode languages L_1, L_2 with $L = L_1 \cup L_2$. By the induction hypothesis L_1 and L_2 are regular, thus L is also regular.
– Assume that $\varphi = \exists x. \psi$. Then it is clear that $(\psi, \mathcal{R})$ also encodes L, thus L is regular by the induction hypothesis.
– Now assume that φ is a separated conjunction of atoms φ_i. Then the φ_i's correspond to parts the structures encoding L. As $\mathcal{R}$ is connected, necessarily these parts must themselves encode words, and φ_i must encode some language L_i, w.r.t. the root x_i of φ_i and some variable y_i that must be either the root of another atom φ_j $(j \neq i)$ or the variable y. We may assume, by reordering the atoms if needed, that $y_i = x_{i+1}$ (for all $i \in [\![1, n-1]\!]$) and $y_n = y$. Then $L = L_1 \ldots . L_n$. By the induction hypothesis, every language L_i is regular, thus L is also regular.

Putting things together we get the result:

Proof. (Of Thm. 1) Assume that the PCE problem is decidable. We construct an algorithm to test whether a context-free language is regular, thus yielding a contradiction. Let G be a context-free variable (we assume that $\varepsilon \notin L(G)$, as it is clear that the problem of testing whether $L(G)$ is regular is still undecidable in this case). By Lem. 1, we construct a pair $(\varphi, \mathcal{R})$ encoding $L(G)$. Then we check whether there exists a PCE pair $(\varphi', \mathcal{R}')$ such that $(\varphi', \mathcal{R}') \equiv (\varphi, \mathcal{R})$. If such a pair exists then by Lem. 2, L is necessarily regular. Otherwise, L cannot be regular, as otherwise a regular grammar generating L would exist, and we would get a PCE pair encoding L by applying again Lem. 1.

B Proofs of Theorem 2 (Correction and Completeness of the Computation of Abstractions)

We recall Thm. 2:

Theorem 2. *Let φ be a formula and let $\mathcal{R}$ be an SID. We suppose that the construction of abstractions terminates without failing. If $A \in \mathrm{abs}(\varphi)$, then there exists a model $(\mathfrak{s}, \mathfrak{h})$ of φ such that A is an abstraction of $(\mathfrak{s}, \mathfrak{h})$. Moreover, if φ admits a model $(\mathfrak{s}, \mathfrak{h})$, then there exists an abstraction A of φ such that $(\mathfrak{s}, \mathfrak{h}) \models A$.*

We first establish the correctness of the algorithm, by proving that for every abstraction of a formula φ, there exists a model of φ that is also a model of A. To prove the result by induction, we actually need to establish a stronger property, stated below (Lem. 3). For readability, we include some definitions. Let $m = (\mathfrak{s}, \mathfrak{h})$. We denote by $m_{|V}$ the model $(\mathfrak{s}_{|V}, \mathfrak{h})$. Let A be an abstraction. A store $\mathfrak{s}$ is an *A-store* iff $\mathrm{dom}(\mathfrak{s}) = A.V_v$ and for all $x, y \in A.V_v$, $\mathfrak{s}(x) = \mathfrak{s}(y) \iff (x, y) \in A.\backsim$. Let $m = (\mathfrak{s}, \mathfrak{h})$. For every total mapping $\lambda : \mathcal{L} \to \mathcal{L}$ such that λ is injective on $\mathrm{dom}(\mathfrak{h})$, we define the structure $\lambda(m) = (\lambda \circ \mathfrak{s}, \lambda(\mathfrak{h}))$, with $\lambda(\mathfrak{h}) = \{(\lambda(\ell_0), \ldots, \lambda(\ell_n)) \mid (\ell_0, \ldots, \ell_n) \in \mathfrak{h}\}$. A total mapping $\lambda : \mathcal{L} \to \mathcal{L}$ is *compatible* with $(A, \mathfrak{s})$ (with A an abstraction and $\mathfrak{s}$ a store) if $\lambda(\ell_1) = \lambda(\ell_2) \wedge \ell_1 \neq \ell_2 \Rightarrow \ell_1, \ell_2 \in \mathrm{img}(\mathfrak{s})$ and $([x], [y]) \in A.\neq \implies \lambda(\mathfrak{s}(x)) \neq \lambda(\mathfrak{s}(y))$.

Lemma 3 (Correctness). *For every formula φ and for each A in $\mathrm{abs}(\varphi)$, (I) there exists an SL-structure $m = (\mathfrak{s}, \mathfrak{h})$ such that $\mathfrak{s}$ is an A-store, $A \in \mathfrak{abs}(m)$ and $\lambda(m) \models_{\mathcal{R}} \varphi$, for every mapping λ compatible with A and $\mathfrak{s}$. Furthermore, (II) there exists an extension $\dot{\mathfrak{s}}$ of $\mathfrak{s}$ satisfying the conditions of Def. 9 such that for all $\ell \in loc(\mathfrak{h})$ there exists a variable $x \in A.V$ with $\dot{\mathfrak{s}}(x) \to^*_{\mathfrak{h}} \ell$.*

Proof. The proof proceeds by structural induction on $\mathrm{abs}(\varphi)$; the result is established for all abstractions constructed during the computation of $\mathrm{abs}(\varphi)$, even those that are not in normal form w.r.t. the simplification rule in Def. 17. See Def. 18 for notations.

- $\varphi = \mathtt{emp}$**:** The only abstraction $A_{\mathtt{emp}}$ in $\mathrm{abs}(\varphi)$ has all components equal to the empty set. Let $m = (\mathfrak{s}, \mathfrak{h})$ with $\mathfrak{s} = \emptyset$ and $\mathfrak{h} = \emptyset$. For any λ, the structure $\lambda(m) = m$ and so $\lambda(m)$ is an SL-model of φ. Moreover, $\mathfrak{s}$ is an $A_{\mathtt{emp}}$-store and $A_{\mathtt{emp}} \in \mathfrak{abs}(m)$, w.r.t. $\dot{\mathfrak{s}} = \mathfrak{s} = \emptyset$. Property (II) holds trivially for empty $\mathfrak{s}$ and $\mathfrak{h}$.
- $\varphi = x \approx y$**:** The set $\mathrm{abs}(\varphi)$ is a singleton $A = A_{\approx}(\{x, y\})$ as defined in Def. 10. Let $m = (\mathfrak{s}, \mathfrak{h})$ with $\mathfrak{s} = [x \mapsto \ell, y \mapsto \ell]$ for some $\ell \in \mathcal{L}$ and $\mathfrak{h} = \emptyset$. Since $\mathrm{dom}(\mathfrak{s})$ is a singleton, the structure $\lambda(m)$ is an SL-model of φ for every mapping λ and $\mathfrak{s}$ is an A-store. Moreover, $A \in \mathfrak{abs}(m)$ by taking $\dot{\mathfrak{s}} = \mathfrak{s}$, since $V = V_v = \{x, y\} = \mathrm{dom}(\dot{\mathfrak{s}})$, $\dot{\mathfrak{s}}(x) = \dot{\mathfrak{s}}(y)$ and the other components are empty. For (II), note that $\dot{\mathfrak{s}}(y) = \ell$.
- $\varphi = x \not\approx y$**:** The set $\mathrm{abs}(\varphi)$ is a singleton $A = A_{\not\approx}(\{x, y\})$ as defined in Def. 10. Let $m = (\mathfrak{s}, \mathfrak{h})$ such that $\mathfrak{h} = \emptyset$ and $\mathfrak{s} = [x \mapsto \ell_1, y \mapsto \ell_2]$ with $\ell_1, \ell_2 \in \mathcal{L}$ and $\ell_1 \neq \ell_2$. It is easy to check that $\lambda(m)$ is an SL-model of φ for any λ compatible with $(A, \mathfrak{s})$ because $\lambda(\mathfrak{s}(x)) \neq \lambda(\mathfrak{s}(y))$. Moreover, $\mathfrak{s}$ is an A-store and $A \in \mathfrak{abs}(m)$ for $\dot{\mathfrak{s}} = \mathfrak{s}$.
  For (II), note that $\dot{\mathfrak{s}}(y) = \ell$.
- $\varphi = y_0 \to (y_1, \ldots, y_n)$**:** The set $\mathrm{abs}(\varphi)$ is a singleton A, as defined in the rule [Pto] in Tab. 1. Let $m = (\mathfrak{s}, \mathfrak{h})$ such that $\mathfrak{s} = [y_0 \mapsto \ell_0, y_1 \mapsto \ell_1, \ldots, y_n \mapsto \ell_n]$ and $\mathfrak{h} = [\ell_0 \mapsto (\ell_1, \ldots, \ell_n)]$ with $\{\ell_0, \ell_1, \ldots, \ell_n\} \subseteq \mathcal{L}$ such that $\mathfrak{s}$ is a bijective function (i.e., it associates each variable to only one ℓ_i) and $\mathrm{img}(\mathfrak{s})$ contains pairwise distinct elements. Because $\mathrm{dom}(\mathfrak{s}) = A.V_v$ and $\mathfrak{s}$ is a bijection, then $\mathfrak{s}$ is an A-store.
  We show that $A \in \mathfrak{abs}(m)$ for $\dot{\mathfrak{s}} = \mathfrak{s}$. Notice that each $\ell_i \in \mathrm{img}(\mathfrak{s})$ identifies an equivalence class of $A.\backsim = \mathtt{Id}$. We have $A.V = A.V_v = \{y_0, \ldots, y_n\} = \mathrm{dom}(\dot{\mathfrak{s}})$,

$\mathrm{dom}(\mathfrak{h}) = \{\ell_0\} = \{\dot{\mathfrak{s}}(y_0)\}$ and $A\text{-}\overline{V}_a = \{[y_0]\}$; $A\text{-}h = \{[y_0] \mapsto ([y_1], \ldots, [y_n])\}$ and $\mathfrak{h}(\mathfrak{s}(y_0)) = (\dot{\mathfrak{s}}(y_1), \ldots, \dot{\mathfrak{s}}(y_n))$, and $A\text{-}\rightsquigarrow = \{([y_0], [y_i]) \mid i \in [\![1, n]\!]\}$.

Since $\mathrm{img}(\mathfrak{s})$ includes all ℓ_i, $\mathfrak{h}$ has one element, and $A\text{-}\neq$ is empty, then any λ compatible with $\mathfrak{s}$ and A gives that $\lambda(m)$ is an SL-model of φ and $\mathfrak{s}$ is an A-store.

For (II), with the above $\dot{\mathfrak{s}}$, since $\rightarrow_{\mathfrak{h}} = \{(\dot{\mathfrak{s}}(y_0), \dot{\mathfrak{s}}(y_i)) \mid i \in [\![1, n]\!]\}$, for any $\ell_i \in \mathrm{img}(\mathfrak{s})$ we have that $\dot{\mathfrak{s}}(y_0) \rightarrow_{\mathfrak{h}} \ell_i$ for all $i \in [\![0, n]\!]$.

$\varphi = \exists x.\, \psi$**:** Remember that we ignore for the moment the *rem* operation in rule [Ex] (Tab. 1) Therefore, $\mathrm{abs}(\varphi)$ contains abstractions $A = A'_{\exists(x)}$ with $A' \in \mathrm{abs}(\psi)$. By induction hypothesis, there is an SL-structure $m = (\mathfrak{s}', \mathfrak{h})$ such that $A' \in \mathfrak{abs}(m)$, w.r.t. some extension $\dot{\mathfrak{s}}$ of $\mathfrak{s}'$, $\mathfrak{s}'$ is an A'-store and $\lambda(m) \models_{\mathcal{R}} \varphi$ for all mappings λ compatible with $(A', \mathfrak{s}')$. Let $\mathfrak{s}$ be the restriction of $\mathfrak{s}'$ to the variables distinct from x and let $m_\exists = (\mathfrak{s}, \mathfrak{h})$. Since $A\text{-}V_v = A'\text{-}V_v \setminus \{x\} = \mathrm{dom}(\mathfrak{s})$, $A\text{-}\backsim$ is the restriction of $A'\text{-}\backsim$ on the variable distinct from x and $\mathfrak{s}'$ is an A'-store, then $\mathfrak{s}$ is an A-store. Since $\mathrm{img}(\mathfrak{s}) \subseteq \mathrm{img}(\mathfrak{s}')$ and A inherits from A' all the disequality constraints between the variables in $A\text{-}V$, any mapping λ compatible with $(A', \mathfrak{s}')$ is also compatible with $(A, \mathfrak{s})$. Thus, $\lambda(m_\exists)$ is an SL-model of φ, for all mappings λ compatible with $(A, \mathfrak{s})$. We show that $A \in \mathfrak{abs}(m_\exists)$, w.r.t. the store $\dot{\mathfrak{s}}$ (which is also an extension of $\mathfrak{s}$). The only difference between A and A' is that $A\text{-}V_v = A'\text{-}V_v \setminus x$. Thus, since we use the same extension $\dot{\mathfrak{s}}$ for A' and A, we only need to consider the condition (1) in Def. 9 that depend on $A\text{-}V_v$ and $\mathfrak{s}$. We have $A\text{-}V_v = A'\text{-}V_v \setminus \{x\} = \mathrm{dom}(\mathfrak{s}') \setminus \{x\} = \mathrm{dom}(\mathfrak{s})$, thus the condition holds.

For (II), notice that $A'_{\exists(x)}$ does not change the reachability relation of A'. In absence of *rem* application, by the induction hypothesis, the property is satisfied for A.

$\varphi = \psi_1 \star \psi_2$**:** From rule [Sep] from Tab. 1, $\mathrm{abs}(\varphi)$ contains abstractions $A = A_1 \star A_2$, where A_1 and A_2 are separated abstractions with $A_i \in \mathrm{abs}(\psi_i)$ $(i \in \{1, 2\})$. By applying the induction hypothesis for each ψ_i, there is an SL-structure $m_i = (\mathfrak{s}_i, \mathfrak{h}_i)$ such that $A_i \in \mathfrak{abs}(m_i)$ w.r.t. some extension $\dot{\mathfrak{s}}_i$ of $\mathfrak{s}_i$, $\mathfrak{s}_i$ is an A_i-store and $\lambda(m_i) \models_{\mathcal{R}} \psi_i$, for all mappings λ compatible with $(A_i, \mathfrak{s}_i)$. We may assume, by renaming locations if needed (using λ), that $\mathrm{img}(\mathfrak{s}_1) \cap \mathrm{img}(\mathfrak{s}_2) = \mathit{ref}(\mathfrak{h}_1) \cap \mathit{ref}(\mathfrak{h}_2) = \emptyset$.

Let $e \mapsto \ell_e$ be any injective mapping from equivalence classes of $A\text{-}\backsim$ to elements of $\mathcal{L}$ not occurring in $\mathrm{img}(\mathfrak{s}_i)$ or $\mathfrak{h}_i$. Let γ_i be the function mapping every location $\mathfrak{s}_i(x)$ to $\ell_{[x]}$ (where $[x]$ denotes the equivalence class of x w.r.t. $A\text{-}\backsim$). Note that γ_i is well-defined, since $A_i\text{-}\backsim \subseteq A\text{-}\backsim$. By definition $\gamma_1(\mathfrak{s}_1)$ and $\gamma_2(\mathfrak{s}_2)$ coincide on all variables in $\mathrm{dom}(\mathfrak{s}_1) \cap \mathrm{dom}(\mathfrak{s}_2)$. We denote by $\mathfrak{s}$ the union of $\gamma_1(\mathfrak{s}_1)$ and $\gamma_2(\mathfrak{s}_2)$; from the properties of $\mathfrak{s}_i$, $\mathfrak{s}$ is an A-store.

From Def. 13, $A = A_1 \star A_2$ inherits all disequality constraints in each A_i. Therefore, if a mapping λ is compatible with $(A, \mathfrak{s})$ then $\lambda \circ \gamma_i$ is compatible with $(A_i, \mathfrak{s}_i)$. Thus the structure $\lambda(\gamma_i(m_i))$ is a model of ψ_i, for all λ compatible with $(A, \mathfrak{s})$.

Let $m'_i = (\mathfrak{s}'_i, \mathfrak{h}'_i)$, the image of m_i by γ_i. Observe that $\mathfrak{h}'_1$ and $\mathfrak{h}'_2$ have disjoint domains. Indeed, if there exists $l \in \mathrm{dom}(\mathfrak{h}'_1) \cap \mathrm{dom}(\mathfrak{h}'_2)$, then we necessarily have $l = \lambda(\gamma_i(\mathfrak{s}_i(x_i))) = \mathfrak{s}(x_i)$ for some $x_i \in \mathrm{dom}(\mathfrak{s}_i) = A_i\text{-}V_v$ for $i \in \{1, 2\}$ (A_1 and A_2 do not share invisible variables). Since $A_i \in \mathfrak{abs}(m_i)$ by induction hypothesis, we deduce that the class of x_i occur in $A_i\text{-}\overline{V}_a$, for $i \in \{1, 2\}$. By Def. 12 of $A_1 \star A_2$, we have $([x_1], [x_2]) \in A\text{-}\neq$, which contradicts the fact that λ is compatible with $(A, \mathfrak{s})$. Thus $\mathfrak{h} = \mathfrak{h}'_1 \cup \mathfrak{h}'_2$ is defined and $(\lambda(\mathfrak{s}), \lambda(\mathfrak{h})) \models_{\mathcal{R}} \varphi$.

We check that $A \in \mathfrak{abs}(\mathfrak{s}, \mathfrak{h})$ w.r.t. an extension $\dot{\mathfrak{s}}$ of $\mathfrak{s}$ that is also an extension of $\gamma(\dot{\mathfrak{s}}_i)$ for $i = 1, 2$ since A_1 and A_2 do not share invisible variables $\mathrm{dom}(\dot{\mathfrak{s}}) = \mathrm{dom}(\dot{\mathfrak{s}}_1) \cup \mathrm{dom}(\dot{\mathfrak{s}}_2) = A{\bullet}V$ and $\mathrm{dom}(\mathfrak{s}) = \mathrm{dom}(\mathfrak{s}_1) \cup \mathrm{dom}(\mathfrak{s}_2) = A{\bullet}V_v$ (Condition 1, Def. 9).
Let $\backsim_i = A_i{\bullet}\backsim$ in the following. The Condition (2) is satisfied if both variables belong only to some A_i, or both variables are shared and visible, since $A_i \in \mathfrak{abs}(m_i)$. If one variable or both are not shared but they are in the same equivalence class of A, then by Def. 13 of $\backsim_\star$, there exist two shared equivalent two these ones in the same equivalence class $\backsim_i$; by the A_i-store property of $\mathfrak{s}_i$ and the definition of $\dot{\mathfrak{s}}$, then both variables are mapped to the same location. Conversely, if $\dot{\mathfrak{s}}$ maps two variables to the same location, then they are in the same equivalence class of $A{\bullet}\backsim$ by definition.
The Condition (3) holds because a location $\dot{\mathfrak{s}}(x)$ is allocated in $\mathfrak{h}$ iff it is allocated in some $\mathfrak{h}'_i$ for $i \in \{1, 2\}$ and thus $\dot{\mathfrak{s}}(x) = \gamma_i(\dot{\mathfrak{s}}_i(x)))$. Since $A_i \in \mathfrak{abs}(m_i)$ then $[x]_{\backsim_i} \in A_i{\bullet}\overline{V}_a$. Moreover, by Def. 13 of $A_1 \star A_2$, then $[x] \in A{\bullet}\overline{V}_a$ iff there exists x' and $i \in \{1, 2\}$ such that $(x, x') \in A{\bullet}\backsim$ and $[x']_{\backsim_i} \in A_i{\bullet}\overline{V}_a$. Thus $\dot{\mathfrak{s}}(x) \in \mathrm{dom}(\mathfrak{h})$ iff $[x] \in A{\bullet}\overline{V}_a$.
The Condition (4): Assume that $A{\bullet}h([y_0]) = ([y_1], \ldots, [y_n])$. By Def. 13, this entails that, for $i \in \{1, 2\}$, $A_i{\bullet}h([y'_0]_{\backsim_i}) = ([y'_1]_{\backsim_i}, \ldots, [y'_n]_{\backsim_i})$, with $(y_j, y'_j) \in A{\bullet}\backsim$. Since $A_i \in \mathfrak{abs}(m_i)$, we get $\mathfrak{h}_i(\dot{\mathfrak{s}}_i(y'_0)) = (\dot{\mathfrak{s}}_i(y'_1), \ldots, \dot{\mathfrak{s}}_i(y'_n))$, so that $\mathfrak{h}(\dot{\mathfrak{s}}(y_0)) = (\dot{\mathfrak{s}}(y_1), \ldots, \dot{\mathfrak{s}}(y_n))$.
The Condition (5): By the above assumption on $\mathfrak{h}_1$ and $\mathfrak{h}_2$, any path $\ell_0 \rightarrow_{\mathfrak{h}} \ldots \rightarrow_{\mathfrak{h}} \ell_n$ in $\mathfrak{h}$ such that $\ell_0 = \dot{\mathfrak{s}}(x)$, $\ell_n = \dot{\mathfrak{s}}(y)$ and $\{\ell_1, \ldots, \ell_{n-1}\} \cap \mathrm{img}(\dot{\mathfrak{s}}) = \emptyset$ must be a path in some $\gamma_i(\mathfrak{h}_i)$ for $i \in \{1, 2\}$. Thus, such a path exists iff $([x']_{\backsim_i}, [y']_{\backsim_i}) \in A_i{\bullet}\rightsquigarrow$, for $i \in \{1, 2\}$, with $(x, x'), (y, y') \in A{\bullet}\backsim$. By construction of $A{\bullet}\rightsquigarrow$ (Def. 13), this is equivalent to state that $([x], [y]) \in A{\bullet}\rightsquigarrow$.
Property (II) follows immediately from the fact that $loc(\mathfrak{h}) = loc(\mathfrak{h}_1) \cup loc(\mathfrak{h}_2)$ and from the induction hypothesis.

$\varphi = p(x_1, \ldots, x_n)$**:** By rule [Pred] from Tab. 1, $A \in \mathrm{abs}(\exists y_1, \ldots, y_n.\, (y_1 \approx x_1 \star \cdots \star y_n \approx x_n \star \psi))$, for some rule $p(y_1, \ldots, y_n) \Leftarrow \psi$. By the induction hypothesis there exists an SL-model $(\mathfrak{s}, \mathfrak{h})$ of $\exists y_1, \ldots, y_n.\, (y_1 \approx x_1 \star \cdots \star y_n \approx x_n \star \psi)$ such that A is an abstraction of $(\mathfrak{s}, \mathfrak{h})$. As φ is a logical consequence of $\exists y_1, \ldots, y_n.\, (y_1 \approx x_1 \star \cdots \star y_n \approx x_n \star \psi)$, $(\mathfrak{s}, \mathfrak{h})$ is also an SL-model of φ and the proof is completed.

It only remains to show that the simplification of abstractions by *rem* (removing invisible variables that are not needed — see Def. 15) preserves the properties (I) and (II). Let A be an abstraction obtained from some abstraction A' by removing some variable u, i.e., $A = rem(A', u)$ (see Def. 17). Let φ be a formula and assume that an SL-structure $m = (\mathfrak{s}, \mathfrak{h})$ exists such that $\mathfrak{s}$ is an A'-store, $\lambda(m) \models_{\mathcal{R}} \varphi$ for all mappings λ compatible with $(A', \mathfrak{s})$ and A' is an abstraction of m, w.r.t. some store $\dot{\mathfrak{s}}$. Since $A{\bullet}V_v = A'{\bullet}V_v$, and $A{\bullet}\backsim$ and $A'{\bullet}\backsim$ coincide on $A{\bullet}V_v$, $\mathfrak{s}$ is an A-store. As A inherits all the disequality constraints between variables in $A{\bullet}V_v$, every mapping λ compatible with $A, \mathfrak{s}$ is also compatible with $(A', \mathfrak{s})$, thus we only have to show that A is an abstraction of m. Let $\dot{\mathfrak{s}}'$ be the restriction of $\dot{\mathfrak{s}}$ to the variables distinct from u. As $\dot{\mathfrak{s}}$ and $\dot{\mathfrak{s}}'$ coincide on any variable other than u, and since the variable u does not occur in $A{\bullet}V_v$, it is straightforward to check that Conditions 1-4 in Def. 9 hold. We only have to consider the last condition 5 since it depends negatively on $\mathrm{img}(\dot{\mathfrak{s}}')$.

- Assume that a path $\ell_0 \rightarrow_{\mathfrak{h}} \ldots \rightarrow_{\mathfrak{h}} \ell_n$ in $\mathfrak{h}$ exists such that $\ell_0 = \dot{\mathfrak{s}}(x)$, $\ell_n = \dot{\mathfrak{s}}(y)$ and $\{\ell_1, \ldots, \ell_{n-1}\} \cap \mathrm{img}(\dot{\mathfrak{s}}') = \emptyset$. If $\{\ell_1, \ldots, \ell_{n-1}\}$ does not contain $\dot{\mathfrak{s}}(u)$, then

$\{\ell_1, \ldots, \ell_{n-1}\} \cap \mathrm{img}(\dot{\mathfrak{s}}) = \emptyset$, thus $([x], [y]) \in A'.\rightsquigarrow$. By definition of A, we deduce that $([x], [y]) \in A.\rightsquigarrow$. If $\ell_i = \dot{\mathfrak{s}}(u)$ for some $i \in [\![1, n]\!]$ then, assuming that the path is minimal, we get $\{\ell_1, \ldots, \ell_{i-1}, \ell_i, \ell_{n-1}\} \cap \mathrm{img}(\dot{\mathfrak{s}}) = \emptyset$, and therefore $([x], [u]) \in A'.\rightsquigarrow$ and $([u], [y]) \in A'.\rightsquigarrow$. By definition of A, we deduce that $([x], [y]) \in A.\rightsquigarrow$.

– Conversely, if $([x], [y]) \in A.\rightsquigarrow$, then we have either $([x], [y]) \in A'.\rightsquigarrow$ and there exists a path from $\dot{\mathfrak{s}}'(x)$ to $\dot{\mathfrak{s}}'(y)$ not crossing $\mathrm{img}(\dot{\mathfrak{s}})$, or $([x], [u]), ([u], [y]) \in A'.\rightsquigarrow$ and there exists paths from $\dot{\mathfrak{s}}(x)$ to $\dot{\mathfrak{s}}(u)$ and from $\dot{\mathfrak{s}}(u)$ to $\dot{\mathfrak{s}}(y)$ (respectively). In both cases we deduce that a path exists from $\dot{\mathfrak{s}}(x)$ to $\dot{\mathfrak{s}}(y)$.

Finally, let $\ell \in loc(\mathfrak{h})$. By the induction hypothesis there exists a variable $u' \in A'.V$ such that $\dot{\mathfrak{s}}(u') \rightarrow_{\mathfrak{h}} \ell$. If $u \neq u'$ then $u' \in A.V$ and the proof is completed. Now assume that $u = u'$. By Def. 17, u cannot be special. By Def. 15, this entails that there exists a set of roots R for A' such that $u \notin R$. Then $(r, u) \in A.\rightsquigarrow^*$ for some $r \in R$. This entails that $\dot{\mathfrak{s}}(r) \rightarrow_{\mathfrak{h}} \dot{\mathfrak{s}}(u)$, thus $\dot{\mathfrak{s}}(r) \rightarrow_{\mathfrak{h}} \ell$.

For all SL structures $m = (\mathfrak{s}, \mathfrak{h})$ and for all subsets E of $\mathcal{V}$, we denote by $m_{|E}$ the structure $(\mathfrak{s}_{|E}, \mathfrak{h})$.

We then show that the algorithm is complete, in the sense that every model of a formula φ can be associated with an abstraction of φ. Again, a stronger property must be proven (Lem. 4). We need the following:

Definition 21 (Coherence). *A store $\mathfrak{s}$ is* coherent with an abstraction A *if it satisfies the following conditions:*

1. $\mathrm{dom}(\mathfrak{s}) = A.V_v$;
2. *for all $x, y \in A.V_v$ if $(x, y) \in A.\backsim$ then $\mathfrak{s}(x) = \mathfrak{s}(y)$ and*
3. *for all $x, y \in A.V_v$ if $([x], [y]) \in A.\neq$ then $\mathfrak{s}(x) \neq \mathfrak{s}(y)$.*

Lemma 4 (Completeness). *For every formula φ and for each SL-structure $m = (\mathfrak{s}, \mathfrak{h})$ such that $m \models_{\mathcal{R}} \varphi$, there exists an abstraction A of φ and an SL-structure m' such that A is an abstraction of $m'_{|fv(\varphi)}$, $m' \models_{\mathcal{R}} \varphi$, and $m = \lambda(m')$ for some total mapping λ.*

Proof. The proof proceeds by structural induction on the construction of $\models_{\mathcal{R}}$. See Def. 3 for notations. Let $m = (\mathfrak{s}, \mathfrak{h})$ be an SL-model of φ.

$\varphi =$ `emp`**:** By definition, we must have $\mathfrak{h} = \emptyset$. By taking $m' = m$ and thus the identity for λ, we get the result. Indeed, the abstraction with all components equal to the empty set, which is in $\mathrm{abs}(\varphi)$ is an abstraction of $m'_{|fv(\varphi)}$ by taking $\dot{\mathfrak{s}} = \emptyset$.

$\varphi = x \approx y$**:** We must have $\mathfrak{h} = \emptyset$ and $\mathfrak{s}(x) = \mathfrak{s}(y)$. Thus, $\mathfrak{s}_{|fv(\varphi)} = [x \mapsto l, y \mapsto l]$, with $l \in \mathcal{L}$. Taking $A = A_{\approx}(\{x, y\})$, $m' = m$ and $\lambda = \mathtt{Id}$, we get the result, since by definition $A \in \mathrm{abs}(\varphi)$ and A is an abstraction of $m'_{|fv(\varphi)}$.

$\varphi = x \not\approx y$**:** We have $\mathfrak{h} = \emptyset$ and $\mathfrak{s}(x) \neq \mathfrak{s}(y)$. Thus, $\mathfrak{s}_{|fv(\varphi)} = [x \mapsto \ell_1, y \mapsto \ell_2]$, with $\ell_1, \ell_2 \in \mathcal{L}$ and $\ell_1 \neq \ell_2$. Taking $A = A_{\not\approx}(\{x, y\})$, $m' = m$ and $\lambda = \mathtt{Id}$, we get the result.

$\varphi = y_0 \rightarrow (y_1, \ldots, y_n)$**:** We have $fv(\varphi) \subseteq \mathrm{dom}(\mathfrak{s})$ and $\mathfrak{h} = \{(\mathfrak{s}(y_0), \ldots, \mathfrak{s}(y_n))\}$ according to Def. 3. Let $\mathfrak{s}'(x) = \mathfrak{s}(x)$ for all $x \notin \{y_0, \ldots, y_n\}$ and $\mathfrak{s}'(y_i) = \ell_i$ with $\ell_0, \ldots, \ell_n$ locations in $\mathcal{L}$ such that $\mathfrak{s}'$ is injective on $\{y_0, \ldots, y_n\}$. Let λ be the identity on $\mathcal{L} \setminus \{\ell_0, \ldots, \ell_n\}$ and defined by $\lambda(\ell_i) = \mathfrak{s}(y_i)$ on $\{\ell_0, \ldots, \ell_n\}$. We have $\mathfrak{s} = \lambda \circ \mathfrak{s}'$. Let $\mathfrak{h}' = \{(\mathfrak{s}(y_0), \ell_1, \ldots, \ell_n)\}$, it verifies $\mathfrak{h} = \lambda(\mathfrak{h}')$. $m' = (\mathfrak{s}', \mathfrak{h}')$ verifies $m = \lambda(m')$ and

$m' \models_{\mathcal{R}} \varphi$. The set $\mathrm{abs}(\varphi)$ is a singleton A, as defined by the Pto rule in Def. 18. It is easy to check that A is an abstraction of $m'|_{fv(\varphi)}$ as it is done in the fourth item in the proof of Lem. 3.

$\varphi = \exists x.\, \psi$**:** By Def. 3, $fv(\varphi) \subseteq \mathrm{dom}(\mathfrak{s})$ and there exists a store $\mathfrak{s}_\exists$, coinciding with $\mathfrak{s}$ on all variables distinct from x, such that $m_\exists = (\mathfrak{s}_\exists, \mathfrak{h}) \models_{\mathcal{R}} \psi$. By the induction hypothesis, there exists an abstraction $A_\exists$ of ψ, an SL-structure $m'_\exists = (s'_\exists, h'_\exists)$ and a mapping $\lambda_\exists$ such that $A_\exists$ is an abstraction of $m'_\exists|_{fv(\psi)}$, $m'_\exists \models_{\mathcal{R}} \psi$ and $m_\exists = \lambda_\exists(m'_\exists)$. Let A be the abstraction of φ obtained from $A_\exists$ as defined by the rule Ex in Def. 18. Let ℓ be any location not occurring in $\mathrm{img}(\mathfrak{s}'_\exists) \cup loc(\mathfrak{h}'_\exists)$ and λ defined such that $\lambda(\ell) = \mathfrak{s}(x)$ and λ coincides with $\lambda_\exists$ on all locations other than ℓ. Let $\mathfrak{s}'$ such that $\mathrm{dom}(\mathfrak{s}') = \mathrm{dom}(\mathfrak{s})$, $\mathfrak{s}'(y) = \mathfrak{s}'_\exists(y)$ for $y \in \mathrm{dom}(\mathfrak{s}) \setminus \{x\}$, $\mathfrak{s}'(x) = \ell$ if $x \in \mathrm{dom}(\mathfrak{s})$ and let $m' = (\mathfrak{s}', \mathfrak{h}')$ with $\mathfrak{h}' = \mathfrak{h}'_\exists$. By construction, $m = \lambda(m')$ because $\mathfrak{h} = \lambda_\exists(\mathfrak{h}'_\exists) = \lambda(\mathfrak{h}')$, $\mathfrak{s}(y) = \mathfrak{s}_\exists(y) = \lambda_\exists(\mathfrak{s}'_\exists(y)) = \lambda(\mathfrak{s}'(y))$ for $y \in \mathrm{dom}(\mathfrak{s}) \setminus \{x\}$ and, if $x \in \mathrm{dom}(\mathfrak{s})$, then $\mathfrak{s}(x) = \lambda(\ell) = \lambda(\mathfrak{s}'(x))$. By definition, $\mathfrak{s}'_\exists$ coincides with $\mathfrak{s}'$ on all variables distinct from x and $m'_\exists \models_{\mathcal{R}} \psi$ so $m' \models_{\mathcal{R}} \varphi$. $A_\exists$ is an abstraction of $m'_\exists|_{fv(\psi)}$ w.r.t. $\dot{\mathfrak{s}}_\exists$. By taking $\dot{\mathfrak{s}} = \dot{\mathfrak{s}}_\exists$ we show as the fifth item of the proof of Lem. 3 that A is an abstraction of $m'|_{fv(\varphi)}$.

$\varphi = \psi_1 \star \psi_2$**:** According to Def. 3, $fv(\varphi) \subseteq \mathrm{dom}(\mathfrak{s})$ and there exist disjoints domain heap $\mathfrak{h}_1$, $\mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ and $m_i = (\mathfrak{s}, \mathfrak{h}_i) \models_{\mathcal{R}} \psi_i$, for all $i \in \{1, 2\}$. By the induction hypothesis, there exist abstractions A_i of ψ_i, SL-structures $m'_i = (\mathfrak{s}'_i, \mathfrak{h}'_i)$ and mappings λ_i such that A_i is an abstraction of $m'_i|_{fv(\psi_i)}$, $m'_i \models_{\mathcal{R}} \psi_i$ and $m_i = \lambda_i(m'_i)$ for each $i \in \{1, 2\}$. We can ensure that $A_1{\bullet}V_{inv} \cap A_2{\bullet}V_{inv} = \emptyset$, by renaming invisible variables in both abstractions as well as $\mathrm{img}(\mathfrak{s}'_1) \cap \mathrm{img}(\mathfrak{s}'_2) = \emptyset$ and $ref(\mathfrak{h}'_1) \cap ref(\mathfrak{h}'_2) = \emptyset$ by renaming locations.

To begin we check that abstractions A_1 and A_2 are separated, i.e., that the disequality relation $\{([x_1]_{A_\star \bullet \backsim}, [x_2]_{A_\star \bullet \backsim}) \mid x_i \in A_\star{\bullet}V, [x_i]_{A_i \bullet \backsim} \in A_i{\bullet}\overline{V}_a, i \in \{1,2\}\} \cup \{([x]_{A_\star \bullet \backsim}, [y]_{A_\star \bullet \backsim}) \mid x, y \in A_\star{\bullet}V, ([x]_{A_i \bullet \backsim}, [y]_{A_i \bullet \backsim}) \in A_i{\bullet}\neq, i \in \{1,2\}\}$ is irreflexive. If there exists $([x], [x])$ in the first set, it implies that there exists x_i such that $[x_i]_{A \bullet \backsim} = [x]_{A \bullet \backsim}$ and $[x_i]_{A_i \bullet \backsim} \in A_i{\bullet}\overline{V}_a$ for $i \in \{1,2\}$ and such that $(x_1, x_2) \in A_j{\bullet}\backsim$, for a $j \in \{1,2\}$. Note that x, x_1 and x_2 must be visible because $A_1{\bullet}V_{inv} \cap A_2{\bullet}V_{inv} = \emptyset$. As A_i is an abstraction of $m'_i|_{fv(\psi_i)}$, $\mathfrak{s}'_i(x_i) \in \mathrm{dom}(\mathfrak{h}'_i)$, i.e., $\mathfrak{s}(x_i) = \mathfrak{s}_i(x_i) = \lambda_i(\mathfrak{s}'_i(x_i)) \in \lambda_i(\mathrm{dom}(\mathfrak{h}'_i)) = \mathrm{dom}(\mathfrak{h}_i)$. Similarly, $\mathfrak{s}(x_1) = \lambda_j(\mathfrak{s}'_j(x_1)) = \lambda_j(\mathfrak{s}'_j(x_2)) = \mathfrak{s}(x_2)$ which contradicts $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$. By construction of A_1 and A_2 $([x], [x])$ cannot exist in the second set.

Let A be the abstraction of φ obtained from A_1 and A_2 as defined in the rule Sep of Def. 18.

Let $e \mapsto \ell_e$ be any injective mapping from equivalence classes of $A{\bullet}\backsim$ to elements of $\mathcal{L}$ not occurring in $\mathrm{img}(\mathfrak{s})$, $loc(\mathfrak{h})$, $\mathrm{img}(\mathfrak{s}'_i)$ or $loc(\mathfrak{h}'_i)$. Let γ_i be the function mapping every location $\mathfrak{s}'_i(x)$ to $\ell_{[x]}$ (where $[x]$ denotes the equivalence class of x w.r.t. $A{\bullet}\backsim$) if $[x]$ is defined, and to pairwise distinct fresh locations otherwise. γ_i is defined as the identity on all other locations. Note that γ_i is well-defined, as $\mathfrak{s}_i$ is necessarily coherent with A (since it is coherent with A_i).

Let $\mathfrak{s}'$ be defined by $\mathfrak{s}'(x) = \ell_{[x]}$ if $\ell_{[x]}$ is defined and $\mathfrak{s}'(x) = \mathfrak{s}(x)$ if not. Consider the mapping $\lambda : \mathfrak{s}'(x) \mapsto \mathfrak{s}(x)$. Observe that λ is well-defined. Indeed, if $\mathfrak{s}'(x) = \mathfrak{s}'(y)$ for $x \neq y$, then by construction of $e \mapsto \ell_e$, If both $\ell_{[x]}$ and $\ell_{[y]}$ are defined, thus

$[x] = [y]$ and $\mathfrak{s}(x) = \mathfrak{s}(y)$ (because $A_{\bullet\backsim} = A_{1\bullet\backsim} \cup A_{2\bullet\backsim}$ and $\mathfrak{s}$ is coherent with A_i). As a result, λ is well defined and $\mathfrak{s} = \lambda \circ \mathfrak{s}'$.
Let $\mathfrak{h}' = \gamma_1(\mathfrak{h}'_1) \cup \gamma_2(\mathfrak{h}'_2)$. We prove that $\gamma_1(\mathfrak{h}'_1)$ and $\gamma_2(\mathfrak{h}'_2)$ are disjoint. Assume, for the sake of contradiction, that there exists $\ell \in \mathrm{dom}(\gamma_1(\mathfrak{h}'_1)) \cap \mathrm{dom}(\gamma_2(\mathfrak{h}'_2))$, then for all $i \in \{1,2\}$ there exists ℓ_i such that $\ell_i \in \mathrm{dom}(\mathfrak{h}'_i)$ and $\ell = \gamma_i(\ell_i)$. Because γ_i is the identity on locations not in $\mathrm{img}(\mathfrak{s}'_i)$ and because $\mathfrak{h}'_1$ and $\mathfrak{h}'_2$ are distinct, for all $i \in \{1,2\}$, there exists x_i such that $\ell_i = \mathfrak{s}'_i(x_i)$. That means $\gamma_1(\mathfrak{s}'_1(x_1)) = \ell = \gamma_2(\mathfrak{s}'_2(x_2))$ i.e., $[x_1] = [x_2]$ thus $\mathfrak{s}(x_1) = \mathfrak{s}(x_2)$. However, $\mathfrak{s}(x_i) = \lambda_i(\mathfrak{s}'_i(x_i)) = \lambda_i(\ell_i) \in \mathrm{dom}(\mathfrak{h}_i)$. Finally, $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) \neq \emptyset$ which is impossible. As a result, $\mathfrak{h}' = \gamma_1(\mathfrak{h}'_1) \uplus \gamma_2(\mathfrak{h}'_2)$. We note that for all $\ell \in \mathrm{img}(\mathfrak{s}'_i)$, $\lambda_i(\ell) = \lambda_i(\mathfrak{s}'_i(x)) = \mathfrak{s}(x) = \lambda(\mathfrak{s}'(x)) = \lambda(\ell_{[x]}) = \lambda(\gamma_i(\mathfrak{s}'_i(x))) = \lambda(\gamma_i(\ell))$. Thus $\lambda_i = \lambda \circ \gamma_i$. As a result, $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2 = \lambda_1(\mathfrak{h}'_1) \uplus \lambda_2(\mathfrak{h}'_2) = \lambda(\gamma_1(\mathfrak{h}'_1)) \uplus \lambda(\gamma_2(\mathfrak{h}'_2)) = \lambda(\mathfrak{h}')$.
We prove that $(\mathfrak{s}', \mathfrak{h}') \models_{\mathcal{R}} \varphi$. We need to show that $(\mathfrak{s}', \gamma_i(\mathfrak{h}'_i)) \models_{\mathcal{R}} \psi_i$. We know that $\mathfrak{s}' = \gamma_i \circ \mathfrak{s}'_i$, and $(\mathfrak{s}'_i, \mathfrak{h}'_i) \models_{\mathcal{R}} \psi_i$. Thus $\psi_i \Leftarrow^*_{\mathcal{R}} \exists \vec{u} \psi'_i$, with $(\dot{\mathfrak{s}}'_i, \mathfrak{h}'_i) \models_{\mathcal{R}} \psi'_i$ for some store $\dot{\mathfrak{s}}'_i$ coinciding with $\mathfrak{s}'_i$ on all variables not occurring in $\vec{u}$ and ψ'_i contains no predicate symbol and no quantifier. Let $\dot{\mathfrak{s}}'$ be a store coinciding with $\mathfrak{s}'$ on all variables not occurring in $\vec{u}$ and with $\dot{\mathfrak{s}}'_i$ otherwise. Consider any atom a occurring in ψ'_i. By definition, there exists a subheap $\mathfrak{h}^a_i$ of $\mathfrak{h}'_i$ such that $(\dot{\mathfrak{s}}'_i, \mathfrak{h}^a_i) \models_{\mathcal{R}} a$. If a is a points-to atom, then this entails that $(\dot{\mathfrak{s}}', \gamma_i(\mathfrak{h}^a_i)) \models_{\mathcal{R}} a$. If a is an equation $x \approx y$, then $\dot{\mathfrak{s}}'_i(x) = \dot{\mathfrak{s}}'_i(y)$ and $\mathfrak{h}^a_i = \emptyset$, so that $\gamma_i(\dot{\mathfrak{s}}'_i(x)) = \gamma_i(\dot{\mathfrak{s}}_i(y))$, and $(\dot{\mathfrak{s}}', \gamma_i(\mathfrak{h}^a_i)) \models_{\mathcal{R}} a$. If a is a disequation $x \not\approx y$, then $\dot{\mathfrak{s}}'_i(x) \neq \dot{\mathfrak{s}}'_i(y)$ and $\mathfrak{h}^a_i = \emptyset$. If $\dot{\mathfrak{s}}'_i(x)$ or $\dot{\mathfrak{s}}'_i(y)$ does not occur in $\mathfrak{s}'_i(fv(\varphi))$ then this entails, by definition of γ_i, that $\gamma_i(\dot{\mathfrak{s}}'_i(x)) \neq \gamma_i(\dot{\mathfrak{s}}'_i(y))$. Otherwise, there exist $x', y' \in fv(\varphi)$ such that $\dot{\mathfrak{s}}'_i(x) = \mathfrak{s}'_i(x')$ and $\dot{\mathfrak{s}}'_i(y) = \mathfrak{s}'_i(y')$. As A is an abstraction of m' we get $(x', y') \in A_{\bullet\backsim}$, so that $\gamma_i(\mathfrak{s}'_i(x')) \neq \gamma_i(\mathfrak{s}'_i(y'))$, by definition of γ_i. Thus $(\dot{\mathfrak{s}}', \gamma_i(\mathfrak{h}^a_i)) \models_{\mathcal{R}} a$. Consequently, $(\dot{\mathfrak{s}}', \gamma_i(\mathfrak{h}'_i)) \models_{\mathcal{R}} \psi'_i$ and therefore $(\mathfrak{s}', \gamma_i(\mathfrak{h}'_i)) \models_{\mathcal{R}} \psi_i$.
To show that A is an abstraction of $(\mathfrak{s}'_{|fv(\psi)}, \mathfrak{h}')$ we use the same proof as in the sixth item of the proof of Lem. 3.

$\varphi = p(x_1, \ldots, x_n)$**:** According to Def. 3, $fv(\varphi) \subseteq \mathrm{dom}(\mathfrak{s})$ and $m \models_{\mathcal{R}} \psi$ for some ψ such that $\varphi \Leftarrow_{\mathcal{R}} \psi$. The formula ψ is equivalent to a formula $\psi' = \exists y_1, \ldots, y_n. (y_1 \approx x_1 \star \cdots \star y_n \approx x_n \star \psi'')$, where $\mathcal{R}$ contains a rule of the form $p(y_1, \ldots, y_n) \Leftarrow \psi''$, so $m \models_{\mathcal{R}} \psi'$. By the induction hypothesis, there exist an abstraction $A_{\psi'}$ of ψ', an SL-structure m' and a mapping λ such that $A_{\psi'}$ is an abstraction of $m'_{|fv(\psi')}$, $m' \models_{\mathcal{R}} \psi'$ and $m = \lambda(m')$. By the rule [Pred] in Def. 18, $A_{\psi'}$ is an abstraction of φ. It is clear that $m' \models_{\mathcal{R}} \varphi$ because ψ' and ψ are equivalent and it is easy to verify that $A_{\psi'}$ is also an abstraction of $m'_{|fv(\varphi)}$.

We also must show that if A is an abstraction of $(\mathfrak{s}'_{|fv(\psi)}, \mathfrak{h}')$ for a given φ, then the simplification according to Def. 17 does not have any impact, i.e., $rem(A)$ is also an abstraction of $(\mathfrak{s}'_{|fv(\psi)}, \mathfrak{h}')$. This follows immediately from the last part of the proof of Lem. 3. We only need to take $\lambda = \mathtt{Id}$ in the proof to get the result.

C Proof of Theorem 3 (Termination of the Computation of Abstractions)

We recall Thm. 3:

Theorem 3. *Let φ be a formula and let $\mathcal{R}$ be an SID. If there exists $k \in \mathbb{N}$ such that $(\varphi, \mathcal{R})$ is k-PCE-compatible, then the computation of* abs(φ) *terminates without failure (hence the ISIV condition is never fulfilled). Otherwise, the ISIV condition eventually applies during the computation of* abs(φ). *Consequently, the problem of testing whether $(\varphi, \mathcal{R})$ is* k-PCE-compatible *for some $k \in \mathbb{N}$ is decidable.*

We first assume that $(\varphi, \mathcal{R})$ is k-PCE-compatible and show that the computation of abs(φ) is finite. To this aim, we need to establish several interesting results. The first lemma states that all minimal sets of roots have the same cardinality:

Proposition 3. *Let $\rightarrow$ be a binary relation. If R_1 and R_2 are two minimal sets of roots for $\rightarrow$, then* $\mathrm{card}(R_1) = \mathrm{card}(R_2)$.

Proof. Let $R_1 = \{x_1, \ldots, x_n\}$ and $R_2 = \{y_1, \ldots, y_m\}$. As R_2 is a set of roots, necessarily for all $i \in \{1, \ldots, n\}$ there exists $j_i \in \{1, \ldots, m\}$ such that $y_{j_i} \rightarrow^* x_i$. As R_1 is a set of roots, this entails that the set $\{y_{j_1}, \ldots, y_{j_n}\}$ is also a set of roots. By minimality of R_2, necessarily this set is of cardinality at least m (as $\{y_{j_1}, \ldots, y_{j_n}\} \subseteq R_2$), thus $n \geq m$. By symmetry, we also have $m \geq n$ thus $m = n$.

We then observe that a variable that is recognized as persistent at some point during the computation will always remain persistent, hence cannot be eliminated:

Proposition 4. *Let φ be a formula and let A be an abstraction in* abs(φ). *For every abstraction A' occurring in the construction tree of A, $A'.V_{per} \subseteq A.V_{per}$.*

Proof. Let $x \in A'.V_{per}$. By hypothesis, x is persistent in A', hence fulfils Item 1 or 2 of Def. 15. Observe that none of the construction rules may possible introduce new equations involving x, and cannot add new elements in $\overline{V}_a$, h or $\rightsquigarrow$ involving the class of x. Indeed, by definition of the construction, such an addition would be possible only at a time where x is visible, and (as invisible variables are renamed to avoid variable collision) x cannot be visible outside of A'. Consequently, x must be persistent also in A, hence $x \in A.V_{per}$.

The following lemma shows that the number of variables occurring in the abstractions of φ is bounded:

Lemma 5. *Let $(\varphi, \mathcal{R})$ be a k-PCE-compatible pair. Let A be an abstraction in* abs(φ). *The number of variables in A is bounded by a number depending only on k, φ and $\mathcal{R}$.*

Proof. By Lem. 3, there exist a model $(\mathfrak{s}, \mathfrak{h})$ of φ and an extension $\dot{\mathfrak{s}}$ of $\mathfrak{s}$ such that A is an abstraction of $(\mathfrak{s}, \mathfrak{h})$ w.r.t. $\dot{\mathfrak{s}}$ and for all $\ell \in loc(\mathfrak{h})$, there exists $x \in A.V$ with $\dot{\mathfrak{s}}(x) \rightarrow^*_{\mathfrak{h}} \ell$. This entails that for all sets of roots $R = \{[x_i] \mid i \in [\![1, n]\!]\}$ for A, the set $\{\dot{\mathfrak{s}}(x_i) \mid i \in [\![1, n]\!]\}$ is a set of roots for $\rightarrow_{\mathfrak{h}}$. Furthermore, this set is necessarily minimal, as R is minimal for $A.\rightsquigarrow$. Since φ is k-PCE-compatible and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \varphi$, necessarily $n \leq k$ (using Prop. 3). Consequently, the set S of invisible variables occurring in all sets of roots of A is also of cardinality at most k (as invisible variables are alone in their classes). Moreover, for every variable $x \in A.V$ such that $[x] \notin A.\overline{V}_a$, we have $\dot{\mathfrak{s}}(x) \notin \mathrm{dom}(\mathfrak{h})$. Using again the fact that φ is k-PCE-compatible and that A is an abstraction of $(\mathfrak{s}, \mathfrak{h})$, we deduce that the set $\{[x] \mid [x] \notin A.\overline{V}_a, x \in A.V\}$ is of cardinality at most k. Since invisible variables

are alone in their classes, this entails that the set $\{x \mid [x] \notin A.\overline{V}_a, x \in A.V_{inv}\}$ is also of cardinality at most k. By Def. 15, all special variables either occur in $\{x \mid [x] \notin A.\overline{V}_a, x \in A.V_{inv}\} \cup S$ or are referred to by some visible variables or some variable in S. This entails there exist at most $(n+k) \times m + 2 \times k$ special variables, where n denotes the number of free variables in φ, and m denotes the maximal natural number such that $\mathcal{R}$ or φ contains a points-to atom of the form $u_0 \to (u_1, \ldots, u_m)$. As non special variables are removed, the maximal number of variables is $(n+k) \times m + 2 \times k + n$.

The following result is immediate to prove and entails that the number of abstractions of φ is also bounded:

Proposition 5. *For any $k \in \mathbb{N}$, the number of abstractions containing at most k variables is bounded.*

Conversely, we need to show that the ISIV condition eventually applies if φ is not k-PCE-compatible.

Lemma 6. *If the computation of* $\mathrm{abs}(\varphi)$ *does not terminate, then the ISIV condition applies at some point during the computation.*

Proof. By hypothesis, $\mathrm{abs}(\varphi)$ has construction trees of unbounded sizes. We may assume, w.l.o.g., that all these construction trees are of minimal size. Consider a branch b in a construction tree, and let $(s_i)_{i \in [\![1,n]\!]}$ be the sequence of atoms occurring along the branch (starting from the root), and $(A_i)_{i \in [\![1,n]\!]}$ be the corresponding sequence of abstractions. If the length of b is unbounded, then necessarily n is also unbounded (since $\mathcal{R}$ is finite, hence the depth of the formulas in $\mathcal{R}$ is bounded). Note that by Prop. 4, $i \leq j \implies A_j.V_{per} \subseteq A_i.V_{per}$. Consider the subsequence i_j $(1 \leq j \leq m)$ of $1, \ldots, n$ such that $A_{i_j}.V_{per} \neq A_{i_{j+1}}.V_{per}$, for all $j \in [\![1, m-1]\!]$. As the construction tree is minimal, A_i cannot be a renaming of A_j if $i \neq j$, thus by Prop. 5, if n is unbounded, then necessarily m is also unbounded. By the pigeonghole argument, if m is large enough, there exist j, j' with $j < j'$ and a renaming η such that $s_{i_j} = \eta(s_{i_{j'}})$ and A_{i_j} coincides with $\eta(A_{i_{j'}})$ on all the variables in s_{i_j}. Indeed, by Prop. 5, the number of restrictions of A_i to the variables in s_i is finite. By definition the ISIV condition is fulfilled, with $A = A_{i_j}$ and $A' = A_{i_{j'}}$.

The last lemma shows that the ISIV condition is correct, in the sense that it applies only when the consider formula is not k-PCE-compatible:

Lemma 7. *Given a predicate $p(x_1, \ldots, x_m)$, if an abstraction $A \in \mathrm{abs}(p(x_1, \ldots, x_m))$ fulfils the ISIV condition (see Def. 19), then, for all $n \in \mathbb{N}$, $\mathrm{abs}(p(x_1, \ldots, x_m))$ contains an abstraction A_n with at least n persistent variables, where $A.V_v = A_n.V_v$ and A_n and A agree on all variables in $A.V_v$.*

Proof. We establish the result by induction on n. The proof for $n = 0$ is immediate (taking $A_0 = A$). Now assume that the property holds for n. By definition, the construction tree of A contains an occurrence of an abstraction $A' \in \mathrm{abs}(p(y_1, \ldots, x_m))$, where $p(y_1, \ldots, y_m)$ is a renaming of $p(x_1, \ldots, x_m)$. Since $A_n \in \mathrm{abs}(p(x_1, \ldots, x_m))$, we may replace A' by the abstraction A'_n obtained from A_n by applying the renaming $\{x_i \mapsto y_i \mid i \in \mathbb{N}\}$ (and also renaming invisible variables to avoid any collision).

Since A' and A'_n coincide on all variables $y_1, \ldots, y_m$, we get an abstraction $A_{n+1} \in \mathrm{abs}(p(x_1, \ldots, x_m))$ which coincides with A on every variable $x_1, \ldots, x_m$. By Prop. 4, all persistent variables in A'_n are persistent in A_{n+1}, as well as all the persistent variables in A not occurring in A'. By the ISIV condition, $\mathrm{card}(A{\bullet}V_{per} \setminus A'{\bullet}V_{per}) > 0$, hence $\mathrm{card}(A_{n+1}{\bullet}V_{per}) \geq \mathrm{card}(A_n{\bullet}V_{per}) + 1 \geq n + 1$.

Putting things together we get the result:

Proof. (Of Thm. 3) If φ is k-PCE-compatible then by Lem. 5 the number of variables occurring in abstractions generated during the computation of $\mathrm{abs}(\varphi)$ is bounded, which entails by Prop. 5 that the number of such abstractions is also bounded. Consequently, the computation of $\mathrm{abs}(\varphi)$ necessarily terminates, and (by Lem. 7), the ISIV condition cannot apply. Otherwise, the ISIV condition applies by Lem. 6.

D Proof of Theorem 4 (Unicity of Predicate Abstraction)

We recall Thm. 4

Theorem 4. *$(\Phi, \mathcal{R}) \equiv (\Phi^\dagger, \mathcal{R}^\dagger)$. Moreover, for all predicates p^A_I defined in $\mathcal{R}^\dagger$, the set $\mathrm{abs}(p^A_I(\vec{y}, x'_1, \ldots, x'_m))$ contains exactly one abstraction.*

The first lemma states that every structure denoted by p is also denoted by some predicate p^A_I:

Lemma 8. *Let $p(x_1, \ldots, x_n)$ be an atom and let $(\mathfrak{s}, \mathfrak{h})$ be a structure verifying $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} p(x_1, \ldots, x_n)$. There exists an abstraction $A \in \mathrm{abs}(p(x_1, \ldots, x_n))$ such that $(\mathfrak{s}, \mathfrak{h}) \models A$, and for all disconnected sets I for p, A, $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists x'_1, \ldots, x'_m.\, p^A_I(\vec{y}, x'_1, \ldots, x'_m)$, where $\{x'_1, \ldots, x'_m\} = A{\bullet}V_{inv}$ and $\vec{y}$ is the sequence of variables x_i with $i \notin I$.*

Proof. The proof is by induction on the satisfiability relation. By definition, there exists a formula φ such that $p(x_1, \ldots, x_n) \Leftarrow_{\mathcal{R}} \varphi$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \varphi$. The formula φ is necessarily of the form (up to AC) $\exists \vec{z}.\, (q_1(\vec{u}_1) \star \cdots \star q_k(\vec{u}_k) \star \varphi')$, where φ' contains no predicate atom. Since $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \varphi$, there exist an extension $\dot{\mathfrak{s}}$ of $\mathfrak{s}$ to $\vec{z}$ and disjoint heaps $\mathfrak{h}_0, \ldots, \mathfrak{h}_k$ such that $\mathfrak{h} = \mathfrak{h}_0 \cup \ldots \mathfrak{h}_k$, $(\dot{\mathfrak{s}}, \mathfrak{h}_0) \models_{\mathcal{R}} \varphi'$ and $(\dot{\mathfrak{s}}, \mathfrak{h}_i) \models_{\mathcal{R}} q_i(\vec{u}_i)$, for all $i \in [\![1, k]\!]$. Let $\dot{\mathfrak{s}}_i$ be the store mapping every variable x_j (for $j \in [\![1, \#(q_i)]\!]$) to the image by $\dot{\mathfrak{s}}$ of the j-nth component of $\vec{u}_i$. By definition of $\dot{\mathfrak{s}}_i$, we have $(\dot{\mathfrak{s}}_i, \mathfrak{h}_i) \models_{\mathcal{R}} q_i(x_1, \ldots, x_{\#(q_i)})$, hence by the induction hypothesis, there exist abstractions $A_i \in \mathrm{abs}(q_i(x_1, \ldots, x_{\#(q_i)})$ (for $i \in [\![1, k]\!]$) such that $(\dot{\mathfrak{s}}_i, \mathfrak{h}_i) \models A_i$, and for every disconnected set I_i for q_i, A_i, $(\dot{\mathfrak{s}}_i, \mathfrak{h}_i) \models_{\mathcal{R}} \exists \vec{x}'_i.\, q_{i\,I_i}^{A_i}(\vec{y}_i, \vec{x}'_i)$, where $\vec{x}'_i$ is the sequence of invisible variables in A_i and $\vec{y}_i$ is the sequence of variables x_i with $i \notin I_i$. By definition of $\dot{\mathfrak{s}}_i$, this entails that $(\dot{\mathfrak{s}}, \mathfrak{h}_i) \models_{\mathcal{R}} \exists \vec{x}'_i.\, q_{i\,I_i}^{A_i}(\vec{u}'_i, \vec{x}'_i)$, where $\vec{u}'_i$ denotes the subsequence of $\vec{u}_i$ obtained by keeping only the components $i \notin I_i$. Let A be the abstraction obtained from the rule $p(x_1, \ldots, x_n) \Leftarrow \varphi$ and abstractions $A_1, \ldots, A_k$ as explained in Def. 18. Note that A necessarily exists, as we have $(\dot{\mathfrak{s}}, \mathfrak{h}_0) \models_{\mathcal{R}} \varphi'$ and $(\dot{\mathfrak{s}}', \mathfrak{h}_i) \models A_i$, for all $i \in [\![1, n]\!]$, hence the construction of the abstraction cannot fail, moreover we have $(\mathfrak{s}, \mathfrak{h}) \models A$. By definition $A \in \mathrm{abs}(p(x_1, \ldots, x_n))$. We denote by A' the corresponding abstraction of $q_1(\vec{u}_1) \star \cdots \star q_k(\vec{u}_k) \star \varphi'$. By definition, $\mathcal{R}^\dagger$ contains a rule

$$p^A_I(\vec{s}, x'_1, \ldots, x'_m) \Leftarrow \exists \vec{z}.\, (q_1{}_{J_1}^{A_1}(\vec{t}_1, \vec{v}_1) \star \cdots \star q_k{}_{J_k}^{A_k}(\vec{t}_k, \vec{v}_k) \star \varphi'')\sigma$$

satisfying the conditions on rule Eq. (11) above. By letting $I_i = J_i$, in the above assertion, we get $(\dot{\mathfrak{s}}, \dot{\mathfrak{h}}_i) \models_{\mathcal{R}} \exists \vec{x}'_i.\, q_{i I_i}^{A_i}(\vec{t}_i, \vec{x}'_i)$, so that $(\dot{\mathfrak{s}}, \dot{\mathfrak{h}}_i) \models_{\mathcal{R}} \exists \vec{x}'_i.\, q_{i I_i}^{A_i}(\vec{t}_i, \vec{x}'_i)\sigma$, by definition of σ (since $(\dot{\mathfrak{s}}, \dot{\mathfrak{h}}) \models A'$ and $(x, \sigma(x)) \in A'\!\bullet\!\backsim$). Now, observe that all atoms occurring in φ' but not in φ'' are equational (indeed, if φ' contains an atom $v_0 \to (v_1, \ldots, v_l)$ then necessarily $v_0, \ldots, v_l$ cannot be disconnected in A'). As $(\dot{\mathfrak{s}}, \dot{\mathfrak{h}}_0) \models_{\mathcal{R}} \varphi'$, this entails that we also have $(\dot{\mathfrak{s}}, \dot{\mathfrak{h}}_0) \models_{\mathcal{R}} \varphi''$, and as φ'' contains no predicate atom, $(\dot{\mathfrak{s}}, \dot{\mathfrak{h}}_0) \models_{\mathcal{R}} \varphi''$, so that $(\dot{\mathfrak{s}}, \dot{\mathfrak{h}}_0) \models_{\mathcal{R}} \varphi''\sigma$. Thus $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists \vec{y}. \exists \vec{x}'_1. \ldots \exists \vec{x}'_k.\, (q_1{}_{J_1}^{A_1}(\vec{t}_1, \vec{v}_1) \star \cdots \star q_k{}_{J_k}^{A_k}(\vec{t}_k, \vec{v}_k) \star \varphi'')\sigma$, and therefore $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \exists x'_1, \ldots, x'_m.\, p_I^A(\vec{y}, x'_1, \ldots, x'_m)$ (as, by definition, the invisible variables $x'_1, \ldots, x'_m$ of A, are either invisible in $A'_1, \ldots, A'_k$ or occur in $\vec{y} \setminus \vec{z}$).

Conversely, all structures denoted by p_I^A are also denoted by p. To prove this assertion, we first need to relate the abstractions associated with p_I^A to A. Let p be an n-ary predicate and let A be an abstraction in $\mathrm{abs}(p(x_1, \ldots, x_n))$. For any disconnected set I for p and A, we denote by A^I the abstraction B identical to A, except that B has no invisible variables and that every variable x_i with $i \in I$ is removed. Formally: $B\!\bullet\!V = A\!\bullet\!V \setminus \{x_i \mid i \in I\}$, $B\!\bullet\!V_v = B\!\bullet\!V$, $(x, y) \in B\!\bullet\!\backsim \iff ((x, y) \in A\!\bullet\!\backsim \wedge\, x, y \in B\!\bullet\!V)$ and $([x], [y]) \in B\!\bullet\!\backsim \iff (([x], [y]) \in A\!\bullet\!\backsim \wedge\, x, y \in B\!\bullet\!V)$. Note that the removal of the variables x_i with $i \in I$ affects neither $A\!\bullet\!\overline{V}_a$, $A\!\bullet\!h$ nor $A\!\bullet\!\rightsquigarrow$, nor the equivalence class of the other variables, as by hypothesis all the variables x_i with $i \in I$ are disconnected.

The next lemma shows the second part of Thm. 4.

Lemma 9 (Unicity of the abstraction). *Let p be an n-ary predicate symbol. For all predicates p_I^A, $\mathrm{abs}(p_I^A(\vec{y}, x'_1, \ldots, x'_m)) = \{A^I\}$, where $\vec{y}$ is the sequence of variables $x_1, \ldots, x_n$ with $i \notin I$, and $x'_1, \ldots, x'_m$ are the invisible variables in A.*

Proof. The proof is by induction on $\mathrm{abs}(\varphi)$. By definition (since $A \in \mathrm{abs}(p(x_1, \ldots, x_n))$, the SID $\mathcal{R}^\dagger$ contains (at least) one rule $p_I^A(\vec{s}, x'_1, \ldots, x'_m) \Leftarrow \exists \vec{z}.\, (q_1{}_{J_1}^{A_1}(\vec{t}_1, \vec{v}_1) \star \cdots \star q_k{}_{J_k}^{A_k}(\vec{t}_k, \vec{v}_k) \star \varphi'')\sigma$ satisfying all the conditions on the rule Eq. (11) above. Consider any rule of this form. By the induction hypothesis, for every $i \in [\![1, k]\!]$, $\mathrm{abs}(q_{i J_i}^{A_i}(\vec{y}_i, \vec{v}_i)) = \{B_i\}$, where $B_i = {A_i}^{J_i}$ and $\vec{y}_i$ is the sequence of variables x_j with $j \in [\![1, \#(q_i)]\!] \setminus J_i$. Let B and B' be the abstractions corresponding to the formulas $(q_1{}_{J_1}^{A_1}(\vec{t}_1, \vec{v}_1) \star \cdots \star q_k{}_{J_k}^{A_k}(\vec{t}_k, \vec{v}_k) \star \varphi'')\sigma$ and $\exists \vec{z}.\, (q_1{}_{J_1}^{A_1}(\vec{t}_1, \vec{v}_1) \star \cdots \star q_k{}_{J_k}^{A_k}(\vec{t}_k, \vec{v}_k) \star \varphi'')\sigma$, respectively, constructed from $B_1, \ldots, B_k$ as explained in Def. 18. As $(x, \sigma(x)) \in A'\!\bullet\!\backsim$ holds for all $x \in \mathrm{dom}(\sigma)$, A is also the abstraction of the formula $\exists \vec{y}.\, (q_1(\vec{u}_1) \star \cdots \star q_k(\vec{u}_k) \star \varphi')\sigma$ (all variables y in $\mathrm{dom}(\sigma)$ are removed by the simplification rule, as they occur in the same equivalence class as $\sigma(y)$). We denote by A'' the corresponding abstraction of $(q_1(\vec{u}_1) \star \cdots \star q_k(\vec{u}_k) \star \varphi')\sigma$ (by definition, A'' is obtained from A' by removing all variables in $\mathrm{dom}(\sigma)$). By the conditions above, the variables occurring in $A''\!\bullet\!V$ but not in $B'\!\bullet\!V$ must be disconnected in A' (hence also in A''). As disconnected variables are not allocated and are alone in their classes, the removal of such variables does not affect the properties of the other variables, thus A'' and B' agree on all variables in $B'\!\bullet\!V$ (except that all the variables in B' are visible). If a variable is invisible in B, then it must be some special variable in $\vec{y}$, thus it must be also invisible in A, which is impossible since by hypothesis all such variables occur in $x'_1, \ldots, x'_m$ (hence are necessarily visible in B). Thus B has no invisible variable, and A and B agree on all variables in $B\!\bullet\!V$, except that all the variables in B are visible. By the conditions on the rules in $\mathcal{R}^\dagger$, all

the disconnected variables in $\vec{y}$ must be removed, as well as all variables x_i with $i \in I$. Moreover, all the variables x_i with $i \in I$ occur in $B{\bullet}V$. Thus $B = A^I$.

Lemma 10. *Let $p(x_1, \ldots, x_n)$ be an atom, let $A \in \mathrm{abs}(p(x_1, \ldots, x_n))$ and let I be a disconnected set for p and A. We denote by $\{x'_1, \ldots, x'_m\}$ the set of invisible variables in A and by $\vec{y}$ the sequence of variables x_i with $i \notin I$. Let $(\mathfrak{s}, \mathfrak{h})$ be a structure such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} p^A_I(\vec{y}, x'_1, \ldots, x'_m)$, and let $\dot{\mathfrak{s}}$ be any extension of $\mathfrak{s}$ to $\{x_i \mid i \in I\}$ that is coherent with A. Then $(\dot{\mathfrak{s}}, \mathfrak{h}) \models_{\mathcal{R}} p(x_1, \ldots, x_n)$.*

Proof. The proof is again done by induction on $\models_{\mathcal{R}}$. By definition, since $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} p^A_I(\vec{y}, x'_1, \ldots, x'_m)$, there exist a rule $p^A_I(\vec{s}, x'_1, \ldots, x'_m) \Leftarrow \exists \vec{z}. (q_1{}^{A_1}_{J_1}(\vec{t}_1, \vec{v}_1) \star \cdots \star q_k{}^{A_k}_{J_k}(\vec{t}_k, \vec{v}_k) \star \varphi'')\sigma$ of the form of Eq. (11) above, an extension $\mathfrak{s}'$ of $\mathfrak{s}$ and disjoint heaps $\mathfrak{h}_0, \ldots, \mathfrak{h}_k$ such that $(\mathfrak{s}', \mathfrak{h}_0) \models_{\mathcal{R}} \varphi''\sigma$ and $(\mathfrak{s}', \mathfrak{h}_i) \models_{\mathcal{R}} q_i{}^{A_i}_{J_i}(\vec{t}_i, \vec{v}_i)\sigma$ (for all $i \in [\![1, k]\!]$). Let $\dot{\mathfrak{s}}'$ be any extension of $\mathfrak{s}'$ to the variables occurring in $\vec{u}_1, \ldots, \vec{u}_k$ (as defined in the conditions on rule Eq. (11) above) but not in $\mathrm{dom}(\mathfrak{s}')$ such that $\dot{\mathfrak{s}}'$ coincide with $\dot{\mathfrak{s}}$ on all variables in $\mathrm{dom}(\dot{\mathfrak{s}})$ and maps all other variables to pairwise distinct locations not occurring in $\mathrm{img}(\dot{\mathfrak{s}}) \cup \mathrm{img}(\mathfrak{s}')$. Consider the store $\dot{\mathfrak{s}}'_i$ mapping every variable x_j (for $j \in [\![1, \#(q_j)]\!]$) to the j-nth component of $\vec{u}_i\sigma$ and coinciding with $\dot{\mathfrak{s}}'$ on $\vec{v}_i$. By definition, $(\dot{\mathfrak{s}}'_i, \mathfrak{h}_i) \models_{\mathcal{R}} q_i{}^{A_i}_{J_i}(\vec{y}_i, \vec{v}_i)$, where $\vec{y}_i$ denotes the subsequence of $x_1, \ldots, x_{\#(q_i)}$ obtained by removing the components in J_i. Observe that $\dot{\mathfrak{s}}'_i$ is coherent with A_i. Indeed, consider any equality or disequality constraint $x \bowtie y$ induced by A_i. If $x \bowtie y$ occur in ${A_i}^{J_i}$ then it is necessarily fulfilled since by Lem. 9, $\dot{\mathfrak{s}}'_i$ is coherent with ${A_i}^{J_i}$ (as $(\dot{\mathfrak{s}}'_i, \mathfrak{h}_i) \models_{\mathcal{R}} q_i{}^{A_i}_{J_i}(\vec{y}_i, \vec{v}_i)$). Otherwise, one of the variables, say x, must occur in $A_i{\bullet}V$ but not in ${A_i}^{J_i}{\bullet}V$, hence must be disconnected in A_i, which entails that the constraint is negative (as a variable that is not alone in its class cannot be disconnected). If (at some point during the construction of A) both variables x, y occur in the same classes as variables from A, then the constraint must be satisfied, as $\dot{\mathfrak{s}}$ is coherent with A and A inherits all the constraints from A_i. Otherwise $\dot{\mathfrak{s}}'(x)$ is necessarily distinct from $\dot{\mathfrak{s}}'(y)$, by definition of $\dot{\mathfrak{s}}'$, hence the constraint $x \neq y$ is necessarily satisfied. Consequently, by the induction hypothesis, we get $(\dot{\mathfrak{s}}'_i, \mathfrak{h}_i) \models_{\mathcal{R}} q_i(x_1, \ldots, x_{\#(q_i)})$, so that $(\dot{\mathfrak{s}}', \mathfrak{h}_i) \models_{\mathcal{R}} q_i(\vec{u}_i)\sigma$, i.e., $(\dot{\mathfrak{s}}', \mathfrak{h}_i) \models_{\mathcal{R}} q_i(\vec{u}_i)$ (since, by the conditions on the rule Eq. (11), $(x, \sigma(x)) \in A'{\bullet}\backsim$). All atoms occurring in φ' but not in φ'' are either equations $w \approx w$ or disequations $w \not\approx w'$ with $(w, w') \in A'{\bullet}\neq$ (indeed, if φ' contains a points-to atom or a non-trivial equation, then the variable in it cannot be disconnected in A'). This entails that $(\dot{\mathfrak{s}}', \mathfrak{h}_0) \models_{\mathcal{R}} \varphi'$. Consequently, $(\dot{\mathfrak{s}}', \mathfrak{h}) \models_{\mathcal{R}} q_1(\vec{u}_1) \star \cdots \star q_k(\vec{u}_k) \star \varphi'$, thus $(\dot{\mathfrak{s}}, \mathfrak{h}) \models_{\mathcal{R}} p(x_1, \ldots, x_n)$.

The first part of Thm. 4 follows immediately from Lems. 8 and 10:

Corollary 1 (Soundness of the transformation). $(\varphi, \mathcal{R}) \equiv (\varphi^\dagger, \mathcal{R}^\dagger)$.

E Proof of Theorem 5

We recall Thm. 5:

Theorem 5. *Let $(\Phi^\dagger, \mathcal{R}^\dagger)$ be any pair obtained by applying the transformations in Sects. 6 and 7. If the computation of $(\Phi^\ddagger, \mathcal{R}^\ddagger)$ terminates, then $(\Phi^\dagger, \mathcal{R}^\dagger) \equiv (\Phi^\ddagger, \mathcal{R}^\ddagger)$. Also, the SID $\mathcal{R}^\ddagger$, and thus $\Phi^\ddagger$, are PCE.*

We state a property of derived predicates which will be sufficient for our purpose:

Lemma 11. *Let $\mathcal{R}$ be an SID. Let γ be an atom and let $(\mathfrak{s}, \mathfrak{h})$ be a structure. Let $x, y_1, \ldots, y_k$ be variables. If $\mathfrak{h}(\mathfrak{s}(x)) = (\mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k))$ then $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \gamma \iff (\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} x \to (y_1, \ldots, y_k) \star (x \to (y_1, \ldots, y_k) \mathrel{-\!\!\bullet} \gamma)$.*

Proof. – Assume that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \gamma$. The proof is by induction on the satisfiability relation. We distinguish two cases.

- If γ is of the form $u \to (v_1, \ldots, v_k)$, then $\mathfrak{h} = \{(\mathfrak{s}(u), \mathfrak{s}(v_1), \ldots, \mathfrak{s}(v_k))\}$. By the hypothesis of the lemma, we deduce that $\mathfrak{s}(u) = \mathfrak{s}(x)$ and $\mathfrak{s}(v_i) = \mathfrak{s}(y_i)$ for all $i \in [\![1, k]\!]$, so that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} x \to (y_1, \ldots, y_k) \star x \approx u \star y_1 \approx v_1 \star \cdots \star y_k \approx v_k$. By definition, $x \to (y_1, \ldots, y_k) \mathrel{-\!\!\bullet} \gamma$ is $x \approx u \star y_1 \approx v_1 \star \cdots \star y_k \approx v_k$ hence the proof is completed.
- Otherwise, γ must be a predicate atom $p(u_1, \ldots, u_n)$, and by definition, there exists a rule $p(v_1, \ldots, v_n) \Leftarrow \exists \vec{z}. \psi$ and an extension $\mathfrak{s}'$ of $\mathfrak{s}$ with $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} \psi\{v_i \Leftarrow u_i \mid i \in [\![1, n]\!]\}$. Since $\mathfrak{h}(\mathfrak{s}(x)) = (\mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k))$, ψ necessarily contains an atom allocating x, i.e., ψ is of the form $\gamma_1 \star \gamma_2$ where γ_1 is an atom and there exist disjoint heaps $\mathfrak{h}_i$ (for $i \in \{1, 2\}$) with $(\mathfrak{s}', \mathfrak{h}_i) \models_{\mathcal{R}} \gamma_i$ and $x \in \mathrm{dom}(\mathfrak{h}_1)$. By the induction hypothesis, we get $(\mathfrak{s}', \mathfrak{h}_1) \models_{\mathcal{R}} x \to (y_1, \ldots, y_k) \star (x \to (y_1, \ldots, y_k) \mathrel{-\!\!\bullet} \gamma_1)$, so that $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} x \to (y_1, \ldots, y_k) \star (x \to (y_1, \ldots, y_k) \mathrel{-\!\!\bullet} \gamma_1) \star \gamma_2$. By definition of the rule defining $x \to (y_1, \ldots, y_k) \mathrel{-\!\!\bullet} \gamma$ this entails that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} x \to (y_1, \ldots, y_k) \star (x \to (y_1, \ldots, y_k) \mathrel{-\!\!\bullet} \gamma)$.

– Conversely, assume that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} x \to (y_1, \ldots, y_k) \star (x \to ((y_1, \ldots, y_k) \mathrel{-\!\!\bullet} \gamma)$. Again, we distinguish two cases.

- Assume that γ is of the form $u \to (v_1, \ldots, v_k)$. Then $(x \to (y_1, \ldots, y_k)) \mathrel{-\!\!\bullet} \gamma$ is $x \approx u \star y_1 \approx v_1 \star \cdots \star y_k \approx v_k$ hence $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} u \to (v_1, \ldots, v_k)$, i.e., $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \gamma$.
- Otherwise γ must be a predicate atom $p(u_1, \ldots, u_n)$. Because $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} x \to (y_1, \ldots, y_k) \star (x \to (y_1, \ldots, y_k) \mathrel{-\!\!\bullet} \gamma)$, by definition of the rules defining derived predicates, necessarily $\gamma \Leftarrow_{\mathcal{R}} \exists \vec{x}. (\lambda \star \varphi)$, with $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} x \to (y_1, \ldots, y_k) \star ((x \to (y_1, \ldots, y_k)) \mathrel{-\!\!\bullet} \lambda) \star \varphi$, for some extension $\mathfrak{s}'$ of $\mathfrak{s}$. By the induction hypothesis, we get $(\mathfrak{s}', \mathfrak{h}) \models_{\mathcal{R}} \lambda \star \varphi$, so that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{R}} \gamma$.

Now for the proof of Thm. 5, it is clear that the first case preserve equivalence as it only replaces some subformulas by predicates that can only be unfolded into these subformulas. The second case preserves equivalence by Lem. 11.

Now we prove that the obtained rules must be PCE. Establishment is ensured by Sect. 6. Indeed, since the obtained abstractions contain no invisible variable, all non allocated variables originally introduced by unfolding are replaced by visible variables, which are quantified existentially in $\Phi^\dagger$. The fact that the described structures admit exactly one root is ensured by not failing in Sect. 7. This allows the transformation in Sect. 8 to ensure connectivity and progress. Indeed, if progress is not satisfied then Case 2 in Sect. 8 applies, and otherwise, if connectivity is not satisfied, then Case 1 applies. As a result, every rule in $\mathcal{R}^\ddagger$ is PCE.

F Examples

F.1 Example 1

We resume and complete here the running example.

The considered pair $(\varphi, \mathcal{R})$ is composed of the formula $\varphi = p(x, y)$ and the SID $\mathcal{R}$ defined by the following rules:

$$\begin{aligned} &p(x,y) \Leftarrow \exists z.\, z \to (x,y)\,, && q(y) \Leftarrow \exists z,u,t.\, (y \to (z,t) \star r(z,u,t))\,, \\ &p(x,y) \Leftarrow x \to (y) \star q(y)\,, && r(z,u,t) \Leftarrow u \not\approx t \star z \to (u) \star t \to (t)\,. \end{aligned} \tag{20}$$

As said in Ex. 1, this SID is not PCE. Indeed, the first rule for p has a root z that is not a free variable, the rule defining q is not established for the existential variable u and the rule defining r does not respect the progress condition as it has two points-to atoms.

Now let us compute the abstractions of φ thanks to the rules introduced in Sect. 5. They are represented in Fig. 3. Equivalence classes are represented by circles and are labelled by variable names. Allocated classes are filled grey; invisible variables are prefixed with $\exists$ and [ ] are omitted. Disequalities are represented with dashed lines, while heap and reachability relations are represented with tick resp. snaked arrows. The set abs(φ) contains two abstractions, A_1^p and A_2^p. The former is built directly from the non recursive rule of p. The latter is obtained by firstly building the abstractions A_1^r for the atom $r(z, u, t)$ and then A_1^q for the atom $q(y)$ — that calls $r(z, u, t)$ — using the rules in Eq. (20). The abstraction A_2 is obtained by applying the rule [Sep] on A_1^q and on the abstraction obtained by the rule [Pto] for $x \to (y)$. Finally A_2^p is obtained from A_2 by removing variables z and t using the procedure in Def. 17 because they are not special.

In Fig. 3 are represented the abstractions used to created the abstraction of φ.

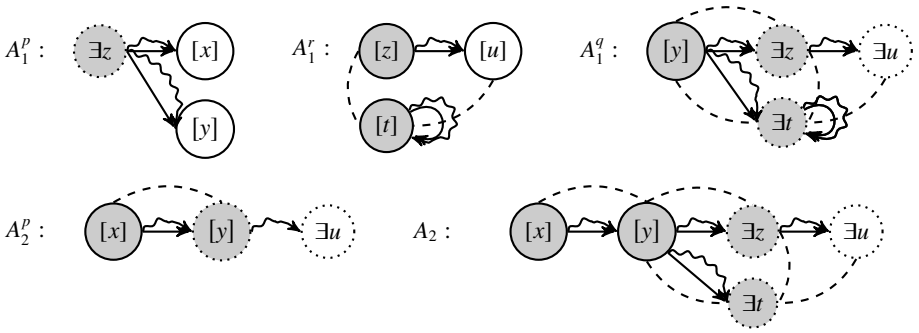

Fig. 3. Abstractions used to construct abs(φ).

We note that abstractions A_1^p and A_1^q in Fig. 3 have a singleton set of roots built from one class: root(A_1^p) = {{[z]}} and root(A_1^q) = {{[y]}}; both A_2 and thus A_2^p have the same singleton set of roots: root(A_2) = root(A_2^p) = {{[x]}}; while A_1^r has a unique set of roots but containing two classes: root(A_1^r) = {{[z], [t]}}.

The variable z is not visible in A_1^p but it is special and persistent since it fulfils the condition 1 of Def. 15. All the invisible variables in A_1^q are special but only y and u are persistent. In A_2, variables z, t, and u are still invisible but z and t are not longer special and thus are removed to form abstraction A_2^p. In the latter, y become invisible and special.

All the abstraction are now constructed. The transformation described in Sect. 6 defines new predicates and new rules to impose the constraint that each predicate to have a unique abstraction.

The predicate p defined by the rules on the left in Eq. (20) has two abstractions (one by rule), A^p_1 and A^p_2, where all roots are connected. In the same example, predicates q and r have also only one abstraction. For all these predicates, the sets I, defined in Sect. 6 are always $\emptyset$.

The new rules for the predicates p, q, r defined in the SID $\mathcal{R}$ in Eq. (20) are given below in a new SID $\mathcal{R}^\dagger$:

$$\begin{aligned} p_\emptyset^{A^p_1}(x,y,z) &\Leftarrow z \to (x,y)\,, \\ p_\emptyset^{A^p_2}(x,y,u) &\Leftarrow \exists z,t.\,(x \to (y) \star q_\emptyset^{A^q_1}(y,z,t,u))\,, \\ q_\emptyset^{A^q_1}(y,z,u,t) &\Leftarrow y \to (z,t) \star r(z,u,t)\,, \\ r_\emptyset^{A^r_1}(z,t,u) &\Leftarrow u \not\approx t \star z \to (u) \star t \to (t)\,. \end{aligned} \tag{21}$$

The arity of predicates $p_\emptyset^{A^p_2}$ and $q_\emptyset^{A^q_1}$ has been changed to include the invisible variables u, z and t and the predicate $p_\emptyset^{A^p_1}$ now does not have an invisible root any more.

The unique abstraction of each predicate defined by $\mathcal{R}^\dagger$'s rules are represented in Fig. 4.

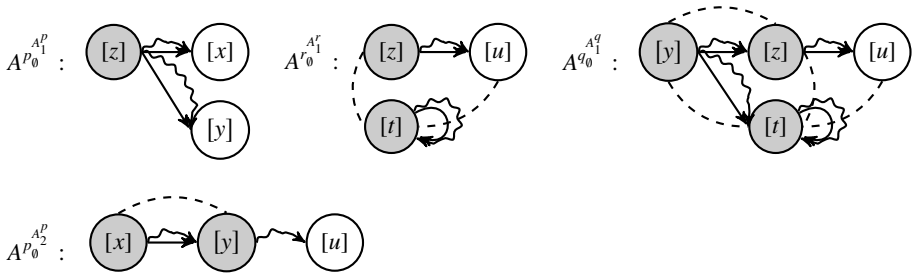

Fig. 4. Abstractions of new predicates $p_\emptyset^{A^p_1}$, $p_\emptyset^{A^p_2}$, $q_\emptyset^{A^q_1}$, and $r_\emptyset^{A^r_1}$.

This transformation also modify φ to be: $\varphi^\dagger = \exists z.\, p_\emptyset^{A^p_1}(x,y,z) \vee \exists u.\, p_\emptyset^{A^p_2}(x,y,u)$.

We remark that all the abstractions have at least one root, thus the first step of Sect. 7 does not apply. However, the predicate $r_\emptyset^{A^r_1}$ still has two roots; since it is not recursive, it can be unfolded.

The unfolding gives us a new SID $\mathcal{R}^\dagger_2$:

$$\begin{aligned} p_\emptyset^{A^p_1}(x,y,z) &\Leftarrow z \to (x,y)\,, \\ p_\emptyset^{A^p_2}(x,y,u) &\Leftarrow \exists z,t.\,(x \to (y) \star q_\emptyset^{A^q_1}(y,z,t,u))\,, \\ q_\emptyset^{A^q_1}(y,z,t,u) &\Leftarrow y \to (z,t) \star z \to (u) \star t \to (t) \star u \not\approx t\,. \end{aligned} \tag{22}$$

The abstractions of the remaining predicates do not change.

Finally, the transformation in Sect. 8 applies only on the rule of $q_{\emptyset}^{A_1^q}$ as $p_{\emptyset}^{A_1^p}$ and $p_{\emptyset}^{A_2^p}$ are already PCE. The Case 1 applies: only two new predicates q_z and q_u for each points-to atom reachable from the root have to be created to obtain PCE rules.

Therefore, the final pair is $(\varphi^\dagger, \mathcal{R}^\ddagger)$ with the SID $\mathcal{R}^\ddagger$ defined as:

$$\begin{aligned}
&p_{\emptyset}^{A_1^p}(x,y,z) \Leftarrow z \to (x,y)\,, && q_z(z,u) \Leftarrow z \to (u)\,,\\
&p_{\emptyset}^{A_2^p}(x,y,u) \Leftarrow \exists z,t.\,(x \to (y) \star q_{\emptyset}^{A_1^q}(y,z,t,u))\,, && q_t(t) \Leftarrow t \to (t)\,.\\
&q_{\emptyset}^{A_1^q}(y,z,t,u) \Leftarrow y \to (z,t) \star q_z(z,u) \star q_t(t) \star u \not\approx t\,,
\end{aligned} \tag{23}$$

F.2 Example 2

This example contains a very simple recursive predicate. It illustrates how the abstraction is built for recursive predicates. In this case, the abstraction process terminates since the SID is 1-PCE-compatible; in fact this set of rules is PCE. However, our transformation change the set of rules.

$$\begin{aligned}
&\mathtt{ls}^+(x,y) \Leftarrow x \to (y)\,,\\
&\mathtt{ls}^+(x,y) \Leftarrow \exists u.\,(x \to (u) \star \mathtt{ls}^+(u,y))\,.
\end{aligned} \tag{24}$$

The abstractions are indexed by the word of SID rules applied (from most recent to the oldest), e.g., $r221$ means applied removing after 2 times call of second rule for ls and first rule for ls.

Here, $\mathrm{abs}(\mathtt{ls}^+(x,y)) = \{A_1, A_{21}, A_{r221}\}$ with:

- $A_1{\bullet}V = \{x,y\}$; $A_1{\bullet}{\backsim} = \mathrm{Id}$; $A_1{\bullet}{\not\approx} = \emptyset$; $A_1{\bullet}V_v = \{x,y\}$; $A_1{\bullet}\overline{V}_a = \{[x]\}$; $A_1{\bullet}h = [[x] \mapsto ([y])]$; $A_1{\bullet}{\rightsquigarrow} = \{([x],[y])\}$. A_1 abstract the formula of the first rule defining ls.
- $A_{21}{\bullet}V = \{x,y,u_1\}$; $A_{21}{\bullet}{\backsim} = \mathrm{Id}$; $A_{21}{\bullet}{\not\approx} = \{([x],[u_1])\}$; $A_{21}{\bullet}V_v = \{x,y\}$; $A_{21}{\bullet}\overline{V}_a = \{[x],[u_1]\}$; $A_{21}{\bullet}h = [[x] \mapsto ([u_1]),[u_1] \mapsto ([y])]$; $A_{21}{\bullet}{\rightsquigarrow} = \{([x],[u_1]),([u_1],[y])\}$. A_{21} is the second abstraction computed by unfolding using the second rule of ls where $\mathtt{ls}^+(u,y)$ is replaced with A_1.
- $A_{r221}{\bullet}V = \{x,y,u_2\}$; $A_{r221}{\bullet}{\backsim} = \mathrm{Id}$; $A_{r221}{\bullet}{\not\approx} = \{([x],[u_2])\}$; $A_{r221}{\bullet}V_v = \{x,y\}$; $A_{r221}{\bullet}\overline{V}_a = \{[x],[u_2]\}$; $A_{r221}{\bullet}h = [[x] \mapsto ([u_2])]$; $A_{r221}{\bullet}{\rightsquigarrow} = \{([x],[u_2]),([u_2],[y])\}$. A_{r221} is the third abstraction computed by the instantiation of $\mathtt{ls}^+(u,y)$ in the second rule second with A_{21} and the removing of variable disconnected (u_1).

Notice that there are only three abstractions for $\mathtt{ls}^+(x,y)$ even if the number of unfoldings is infinite. Indeed, we can see that A_1 abstracts the unfolding $x \to (y)$ and A_{21} the unfolding $x \to (u) \star u \to (y)$, i.e., a list of one element resp. a list of two elements.

All the lists with three or more elements are abstracted by A_{r211} thanks to the fact that not special variables can be removed. Indeed, when A_{r211} is computed using the abstraction A_{21} for $\mathtt{ls}^+(z,y)$, the variable u_1 may be removed. Indeed, u_1 is not a root, it is not pointed by a root and it is not a visible variable of $\mathtt{ls}^+(x,y)$. Thus the abstraction

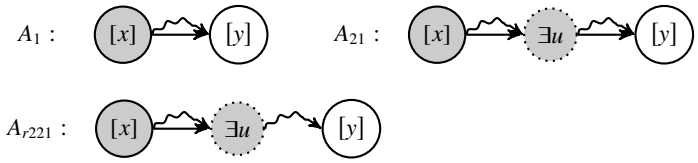

Fig. 5. Abstractions of $\mathtt{ls}^+(x,y)$.

only remembers that from u_1 we can access to y. But it does not remember the number of points-to that are used to do so.

Those abstractions are represented in Fig. 5.

To obtain the transformed form by our procedure, we notice that all abstractions have one universal root and all parameters are used. Therefore, we obtain the following PCE SID:

$$\begin{aligned}
\mathtt{ls}^+_{A_1}(x,y) &\Leftarrow x \to (y)\,,\\
\mathtt{ls}^+_{A_{21}}(x,y,u) &\Leftarrow x \to (u) \star \mathtt{ls}^+_{A_1}(u,y)\,,\\
\mathtt{ls}^+_{A_{r221}}(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star \mathtt{ls}^+_{A_{21}}(u,y,u'))\,,\\
\mathtt{ls}^+_{A_{r221}}(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star \mathtt{ls}^+_{A_{r221}}(u,y,u'))\,,
\end{aligned} \tag{25}$$

and the following formula: $\mathtt{ls}^+_{A_1}(x,y) \vee \exists u.\,\mathtt{ls}^+_{A_{21}}(x,y,u) \vee \exists u.\,\mathtt{ls}^+_{A_{r221}}(x,y,u)$.

As the SID is already PCE, the next transformations will not have any effect.

F.3 Example 3

Let us consider the pair $(\mathtt{ls}^e(x,y), \mathcal{R}_{\mathtt{ls}^e})$ with $\mathcal{R}_{\mathtt{ls}^e}$ defined as following:

$$\begin{aligned}
\mathtt{ls}^e(x,y) &\Leftarrow x \to (y)\,,\\
\mathtt{ls}^e(x,y) &\Leftarrow \exists u.\,(\mathtt{ls}^e(x,u) \star u \to (y))\,.
\end{aligned} \tag{26}$$

The SID $\mathcal{R}_{\mathtt{ls}^e}$ is not PCE. It is exactly the one defined in Eq. (5) that defines a non-empty list constructed from the end. Similarly to the previous example, $\mathtt{ls}^e(x,y)$ has three abstractions, the same as previously. They are represented again in Fig. 6.

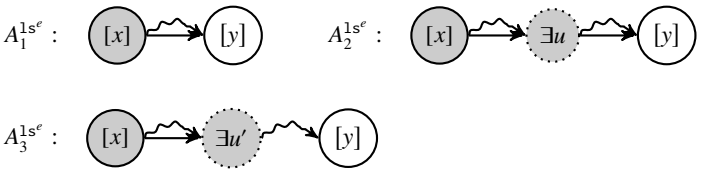

Fig. 6. Abstractions of $\mathtt{ls}^e(x,y)$.

Once again the first transformation is applied to obtain predicates with only one abstraction. We obtain the following PCE SID $\mathcal{R}^{\dagger}_{\mathtt{ls}^e}$:

$$\begin{aligned}
\mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(x,y) &\Leftarrow x \to (y)\,,\\
\mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,y,u) &\Leftarrow \mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(x,u) \star u \to (y)\,,\\
\mathtt{ls}^e_{A^{\mathtt{ls}^e}_3}(x,y,u) &\Leftarrow \exists u'.(\mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,u',u) \star u' \to (y))\,,\\
\mathtt{ls}^e_{A^{\mathtt{ls}^e}_3}(x,y,u) &\Leftarrow \exists u'.(\mathtt{ls}^e_{A^{\mathtt{ls}^e}_3}(x,u',u) \star u' \to (y))\,,
\end{aligned} \tag{27}$$

and the following formula: $\mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(x,y) \vee \exists u.\, \mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,y,u) \vee \exists u.\, \mathtt{ls}^e_{A^{\mathtt{ls}^e}_3}(x,y,u)$.

Each new predicate has a unique root, therefore the transformations of Sect. 7 do not apply.

However, the rules are not all PCE yet. The first rule is PCE and thus not changed. The three last rules fall in the Case 2 of Sect. 8.

Let us consider the second rule. The transformation creates the following rule:

$$\mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,y,u) \Leftarrow x \to (u) \star (x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(x,u)) \star u \to (y)\,,$$

with a new predicate $x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(x,u)$ defined by the rule:

$$x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(x,u) \Leftarrow x \to (u) \mathrel{-\!\!\bullet} x \to (u)\,,$$

witch simplifies in $x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(x,u) \Leftarrow \mathtt{emp}$. The second rule thus is simplified to: $\mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,y,u) \Leftarrow x \to (u) \star u \to (y)$, and then is transformed by the Case 1 into: $\mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,y,u) \Leftarrow x \to (u) \star q_u(u,y)$, with $q_u(u,y) \Leftarrow u \to (y)$ that is recognized to be equivalent to $\mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(u,y)$. Finally the second rule is transformed into:

$$\mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,y,u) \Leftarrow x \to (u) \star \mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(u,y)\,. \tag{28}$$

Now for the third rule. It is transformed into:

$$\mathtt{ls}^e_{A^{\mathtt{ls}^e}_3}(x,y,u) \Leftarrow \exists u'.(x \to (u) \star (x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,u',u)) \star u' \to (y))\,,$$

with a new predicate $x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,u',u)$ defined, after simplification, by:

$$x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(x,u',u) \Leftarrow \mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(u',u)\,.$$

As before we group the two last atoms of the third rule to obtain:

$$\begin{aligned}
\mathtt{ls}^e_{A^{\mathtt{ls}^e}_3}(x,y,u) &\Leftarrow \exists u'.(x \to (u) \star q_u(u,y,u'))\,,\\
q_u(u,y,u') &\Leftarrow \mathtt{ls}^e_{A^{\mathtt{ls}^e}_1}(u,u') \star u' \to (y))\,.
\end{aligned}$$

An equivalence between $q_u(u,y,u')$ and $\mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(u,y,u')$ can be observe and thus the third rule is transformed into:

$$\mathtt{ls}^e_{A^{\mathtt{ls}^e}_3}(x,y,u) \Leftarrow \exists u'.(x \to (u) \star \mathtt{ls}^e_{A^{\mathtt{ls}^e}_2}(u,y,u'))\,. \tag{29}$$

For the last rule, we obtain:

$$\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,y,u) \Leftarrow \exists u'.\,(x \to (u) \star (x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,u',u)) \star u' \to (y))\,,$$

with a new predicate $x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,u',u)$ defined, after simplification, by the two rules:

$$x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,u',u) \Leftarrow \exists u''.\,\mathtt{ls}^{e}_{A_2^{\mathtt{ls}^e}}(u,u',u'')\,,$$
$$x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,u',u) \Leftarrow \exists u''.\,((x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,u'',u)) \star u'' \to (u'))\,.$$

If we rename $(x \to (u) \mathrel{-\!\!\bullet} \mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,u',u)$ into $DA(x,u',u)$, observe that x is not used in the rule, and apply a substitution $[u \leftarrow x, u' \leftarrow y, u'' \leftarrow u]$, we obtain the following rules:

$$DA(x,y) \Leftarrow \exists u.\,\mathtt{ls}^{e}_{A_2^{\mathtt{ls}^e}}(x,y,u)\,,$$
$$DA(x,y) \Leftarrow \exists u.\,(DA(x,u) \star u \to (y))\,.$$

However, this predicate has more than one abstraction. The abstractions of $DA(x,y)$ are exactly $A_2^{\mathtt{ls}^e}$ and $A_3^{\mathtt{ls}^e}$, and if we apply Sects. 6 and 7 to DA we will obtain rules similar to $\mathtt{ls}^{e}_{A_2^{\mathtt{ls}^e}}$ and $\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}$. Therefore, $DA(x,y)$ is equivalent to $\exists u.\,\mathtt{ls}^{e}_{A_2^{\mathtt{ls}^e}}(x,y,u) \vee \exists u.\,\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,y,u)$. We can thus replace $DA(u',u)$, in the last rule of predicate $\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}$, by $\exists u''.\,\mathtt{ls}^{e}_{A_2^{\mathtt{ls}^e}}(u,u',u'') \vee \exists u''.\,\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(u,u',u'')$ to obtain the two following rules:

$$\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,y,u) \Leftarrow \exists u',u''.\,(x \to (u) \star \mathtt{ls}^{e}_{A_2^{\mathtt{ls}^e}}(u,u',u'') \star u' \to (y))\,,$$
$$\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,y,u) \Leftarrow \exists u',u''.\,(x \to (u) \star \mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(u,u',u'') \star u' \to (y))\,.$$

Then, the Case 1 applies and, as before, we obtain:

$$\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,y,u) \Leftarrow \exists u''.\,(x \to (u) \star q_u(u,y,u''))\,,$$
$$\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(x,y,u) \Leftarrow \exists u''.\,(x \to (u) \star q'_u(u,y,u''))\,,$$

with

$$q_u(u,y,u'') \Leftarrow \exists u'.\,(\mathtt{ls}^{e}_{A_2^{\mathtt{ls}^e}}(u,u',u'') \star u' \to (y))\,,$$
$$q'_u(u,y,u'') \Leftarrow \exists u'.\,(\mathtt{ls}^{e}_{A_3^{\mathtt{ls}^e}}(u,u',u'') \star u' \to (y)\,.$$

The rule of the predicate $q_u(u,y,u'')$ is the same as the third rule of Eq. (27), and thus it is transformed into:

$$q_u(x,y,u) \Leftarrow \exists u'.\,(x \to (u) \star \mathtt{ls}^{e}_{A_2^{\mathtt{ls}^e}}(u,y,u'))\,.$$

For the rule of the predicate $q'_u(u,y,u'')$, we have the same rule as the last rule of of Eq. (27), and thus a similar transformation is applied to it, and we obtain:

$$q'_u(x,y,u) \Leftarrow \exists u''.\,(x \to (u) \star p_u(u,y,u''))\,,$$
$$q'_u(x,y,u) \Leftarrow \exists u''.\,(x \to (u) \star p'_u(u,y,u''))\,,$$

with

$$p_u(u,y,u'') \Leftarrow \exists u'.\,(\mathtt{ls}^e_{A_2^{\mathtt{ls}^e}}(u,u',u'') \star u' \to (y))\,,$$
$$p'_u(u,y,u'') \Leftarrow \exists u'.\,(\mathtt{ls}^e_{A_3^{\mathtt{ls}^e}}(u,u',u'') \star u' \to (y))\,.$$

And because we just applied the same transformation as before on the same rules, then p_u is equivalent to q_u and p'_u is equivalent to q'_u and we can replace them:

$$q'_u(x,y,u) \Leftarrow \exists u''.\,(x \to (u) \star q_u(u,y,u''))\,,$$
$$q'_u(x,y,u) \Leftarrow \exists u''.\,(x \to (u) \star q'_u(u,y,u''))\,.$$

The last rule is thus transformed into these rules:

$$\begin{aligned}
\mathtt{ls}^e_{A_3^{\mathtt{ls}^e}}(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star q'_u(u,y,u'))\,,\\
q_u(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star \mathtt{ls}^e_{A_2^{\mathtt{ls}^e}}(u,y,u'))\,,\\
q'_u(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star q_u(u,y,u'))\,,\\
q'_u(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star q'_u(u,y,u'))\,.
\end{aligned} \tag{30}$$

Now to summarise:
If we regroup the first rule of Eq. (27) and the rules we get in Eqs. (28) to (30) we have the PCE form of $(\mathtt{ls}^e(x,y), \mathcal{R}_{\mathtt{ls}^e})$.

The final PCE SID is the following:

$$\begin{aligned}
\mathtt{ls}^e_{A_1^{\mathtt{ls}^e}}(x,y) &\Leftarrow x \to (y)\,,\\
\mathtt{ls}^e_{A_2^{\mathtt{ls}^e}}(x,y,u) &\Leftarrow x \to (u) \star \mathtt{ls}^e_{A_1^{\mathtt{ls}^e}}(u,y)\,,\\
\mathtt{ls}^e_{A_3^{\mathtt{ls}^e}}(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star \mathtt{ls}^e_{A_2^{\mathtt{ls}^e}}(u,y,u'))\,,\\
\mathtt{ls}^e_{A_3^{\mathtt{ls}^e}}(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star q_u(u,y,u'))\,,\\
\mathtt{ls}^e_{A_3^{\mathtt{ls}^e}}(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star q'_u(u,y,u'))\,,\\
q_u(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star \mathtt{ls}^e_{A_2^{\mathtt{ls}^e}}(u,y,u'))\,,\\
q'_u(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star q_u(u,y,u'))\,,\\
q'_u(x,y,u) &\Leftarrow \exists u'.\,(x \to (u) \star q'_u(u,y,u'))\,.
\end{aligned} \tag{31}$$

and the final formula is $\mathtt{ls}^e_{A_1^{\mathtt{ls}^e}}(x,y) \vee \exists u.\,\mathtt{ls}^e_{A_2^{\mathtt{ls}^e}}(x,y,u) \vee \exists u.\,\mathtt{ls}^e_{A_3^{\mathtt{ls}^e}}(x,y,u)$.

G Implementation and Experiments

We have implemented an initial version of the algorithm in OCaml using the Cyclist [2] framework. The prototype uses the front-end of Cyclist including the parsing of input files (in local format or in SL-COMP [1] format) and the abstract syntax tree of formulas and predicate definitions as well as the semantic analysis (mainly typing of predicates definitions). The module implementing our algorithm comprises

approximately 3000 lines of code. The sources are available in the artefact archive at https://hal.science/hal-04549937.

To ensure termination and efficiency, the implemented procedure simplifies the Step 8 to avoid the use of derived predicates. Instead, we employ a fixed-depth unfolding of predicate atoms. The other sections strictly adhere to the algorithm implemented.

Our benchmark includes 145 examples including the ones presented in this paper (main part or Appendix F), some taken in the benchmark of Cyclist (see https://www.cyclist-prover.org/) and some in the SL-COMP benchmark (mainly divisions `qf_shls_entl`, `qf_shlid_entl`, `qf_shid_entl` and `shid_entl`). These examples contain the definition of various data structures, including lists (singly linked, doubly linked) built from start or from end, skip-lists up to three levels, nested lists, binary trees, union-find, etc. The number of predicate definitions per file varies from 1 to 7, and the number of rules from 1 to 8. We include all the predicate definitions from SL-COMP benchmark that may be expressed in our fragment.

We split the 145 tested examples on the following four categories:

- `OK-CPCE`: 105 examples are successfully transformed into equivalent new PCE-formulas;
- `NOK-ISIV`: 20 examples trigger the ISIV condition (which proves that the structures are not k-PCE-compatible);
- `NOK-CPCE-rec`: 3 examples fail at Step 7 (recursive structures with multiple roots);
- `NOK-CPCE-maxit`: 17 other exhaust the allocated execution time or the number of iterations at Step 8; the maximal number of iterations is set to 10 and a timeout on computation of the transformation is set to 30 seconds.

Each category is stored in a separate directory in the artefact archive.

We ran our experiments on a Apple M2 with 8 cores and 24 GB RAM. We measured the global running time (denoted by T_G) and the percent of this time taken by the first step in charge of building the abstractions ($\%T_A$) and testing ISIV condition; the remaining time is taken by the transformation to PCE. We also measured the number of rules generated by the transformation (Rout) and the number of rules in the input (Rin).

Table 2 provide the experimental results for the categories where the transformation is not done. Most examples in the `NOK-ISIV` section correspond to structures with "dangling edges", which model arbitrary data stored inside the structure. Such structures do not fulfil the establishment property (see Def. 5) hence cannot be defined with PCE rules. In contrast, the failure at Step 7 for examples in the `NOK-CPCE-rec` section does *not* entail that no equivalent PCE definition exists. We notice here that only 3 examples from SL-COMP benchmark trigger the ISIV condition, which is an interesting insight for the organizers of this competition.

Table 3 includes a representative selection of the different results for the `OK-CPCE` category (we include in this table all examples of predicate definitions from the SL-COMP benchmark – in files with extension `.smt2`). In most cases, the computation of abstraction dominates the global execution time. However, when it is not the case, e.g., `sll-vc01.smt2`, the formula is very big, so its transformation dominates the execution time.

The transformation sometimes greatly increases the number of rules and predicates, which can be attributed to two factors. Firstly, at Step 6, one predicate is introduced for

Table 2. Experimental results for categories `NOK-ISIV`, `NOK-CPCE-maxit` and `NOK-CPCE-rec`

Category	File	#Pin	#Rin/Rout	T_G (ms)	$\%T_A$
NOK-ISIV	binlistfirst	1	2 / –	< 1	98%
	binlistsecond	1	2 / –	< 1	98%
	binpath	1	3 / –	1	99%
	bintree	1	2 / –	1	99%
	bintreeseg	2	5 / –	11	100%
	bsll	1	2 / –	< 1	97%
	`dll_entails_ls.sb.smt2`	2	4 / –	6	100%
	`dll_entails_lspre.sb.smt2`	2	4 / –	6	100%
	`dll_entails_lsrev.sb.smt2`	2	4 / –	6	100%
	dls	1	2 / –	< 1	98%
	funqueue	3	4 / –	2	100%
	funqueuelists	2	4 / –	3	100%
	isiv	1	2 / –	< 1	99%
	lsls	2	4 / –	10	100%
	ncls2	1	2 / –	< 1	98%
	sll	3	6 / –	7	100%
	sptrue	1	2 / –	< 1	95%
	sptrue2	1	2 / –	< 1	96%
	tree	2	4 / –	261	100%
NOK-CPCE-maxit	`bsll_nil`	1	2 / –	$> 3 \times 10^6$	
	dll2-entails-dll2-rev.smt2	3	6 / –	$> 3 \times 10^6$	
	dll2-rev-entails-dll2.smt2	3	6 / –	$> 3 \times 10^6$	
	dll2-spaghetti.smt2	3	6 / –	$> 3 \times 10^6$	
	ncrls	1	2 / –	1052	100%
	node-tll-tll-entails-tll.smt2	1	2 / –	$> 3 \times 10^6$	
	nonpce1	2	2 / –	3	89%
	nonpce10	2	3 / –	1	85%
	nonpce11	2	4 / –	1	86%
	nonpce5	2	2 / –	6	89%
	rlist	1	2 / –	1077	100%
	tll-pp-entails-tll-pp-rev.smt2	3	6 / –	$> 3 \times 10^6$	
	tll-pp-rev-entails-tll-pp.smt2	3	6 / –	$> 3 \times 10^6$	
	tree-pp-entails-tree-pp-rev.smt2	3	8 / –	$> 3 \times 10^6$	
	tree-pp-rev-entails-tree-pp.smt2	3	8 / –	$> 3 \times 10^6$	
	uf	2	3 / –	8	85%
NOK-CPCE-rec	`funqueuelists_nil`	2	4 / –	46	83%
	test	1	2 / –	2	84%

each abstraction. While this convention greatly simplifies theoretical developments, it negatively impacts the size of the input, particularly for tree-shaped structures where rules involve multiple predicate calls. Secondly, the system does not attempt to reuse predicates when possible, due to the additional bookkeeping involved. Improving both aspects is left to future work.

Overall, we find the results highly encouraging, as approximately 86% of the examples are successfully managed. This success is either achieved by transforming them into equivalent PCE definitions or by demonstrating the infeasibility of such a transformation.

These results may be used by the organisers of SL-COMP to identify the "difficult" problems in the current benchmark and to propose a mesure of this difficulty based on, e.g., the number of rules generated by the translation.

Table 3. Experimental results for OK-CPCE

Category	File	#Pin	#Rin/Rout	T_G (ms)	$\%T_A$
OK-CPCE	append_sll_cll_slk-1.smt2	7	9 / 25	6	88%
	append_sll_ls_slk-1.smt2	6	8 / 21	5	89%
	append_sll_slk-1.smt2	4	5 / 13	3	83%
	bolognesa-10-e01.tptp.smt2	1	2 / 4	96	1%
	clones-01-e01.tptp.smt2	1	2 / 4	1	84%
	dll-entails-dll-rev.smt2	2	4 / 12	22	68%
	dll-entails-dll0+.smt2	2	4 / 12	24	70%
	dll-rev-entails-dll.smt2	2	4 / 12	22	68%
	dll-vc01.smt2	1	2 / 6	9	75%
	append_dllnull_entails_dllnull.sb.smt2	2	4 / 10	11	75%
	append_tail_entails_dll.sb.smt2	1	2 / 6	10	74%
	append_tail_entails_dllnull.sb.smt2	2	4 / 10	11	75%
	append_tail_entails_dllrev.sb.smt2	1	2 / 6	9	73%
	concat.sb.smt2	1	2 / 6	10	74%
	concat_dllrev.sb.smt2	2	4 / 12	20	70%
	entails_dllrev.sb.smt2	2	4 / 12	20	70%
	nil_tl_entails_dllnull.sb.smt2	2	4 / 10	11	75%
	elist	1	2 / 6	2	87%
	elseg4_slk-1.smt2	2	3 / 8	4	86%
	eolseg_01.sb.smt2	4	8 / 24	6	87%
	exF-1	3	4 / 5	2	86%
	funqueue_nil	3	4 / 59	69	89%
	listoe	2	3 / 5	1	85%
	ls-vc01.smt2	1	2 / 4	1	84%
	exF-2	1	2 / 4	1	83%
	ls.smt2	2	5 / 22	6	88%
	ls2_entail_ls_01.sb.smt2	2	5 / 22	6	87%
	ls_entail_ls_nonrec_01.sb.smt2	2	5 / 16	4	87%
	lsbt	2	4 / 200	125	82%
	lsls_nil	2	4 / 36	23	86%
	lsnil	1	2 / 4	1	84%
	lss-vc01.smt2	1	2 / 4	1	84%
	ncls	1	2 / 4	1	84%
	ncls2_nil	1	2 / 8	5	85%
	nll-vc01.smt2	2	4 / 56	45	83%
	node-dll-rev-dll-entails-dll.smt2	2	4 / 12	23	68%
	node-node-dll-entails-dll.smt2	2	3 / 7	12	69%
	odd-lseg3_slk-1.smt2	1	2 / 5	1	84%
	skl2-vc01.smt2	2	4 / 56	51	85%
	skl3-vc01.smt2	3	6 / 1524	20117	83%
	sll-vc01.smt2	1	2 / 4	2603	0%
	smallfoot-vc01.tptp.smt2	1	2 / 4	1	84%
	tseg_join_2.sb.smt2	2	5 / 840	1397	85%
	tseg_join_2_entail_tree.sb.smt2	2	5 / 840	1387	85%
	tseg_join_tree.sb.smt2	2	5 / 840	1388	85%
	tseg_join_tree_entail_tseg.sb.smt2	2	5 / 840	1384	85%